aws

ユーザーガイド

AWS Glue



Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

AWS Glue: ユーザーガイド

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon の商標およびトレードドレスはAmazon 以外の製品およびサービスに使用することはできません。また、お客様に誤解を与える可能性がある形式で、または Amazon の信用を損なう形式で使用することもできません。Amazon が所有していないその他のすべての商標は Amazon との提携、関連、支援関係の有無にかかわらず、それら該当する所有者の資産です。

Table of Contents

AWS Glue とは 1
AWS Glue の機能 2
AWS Glue のイノベーションについて学ぶ 3
AWS Glue の開始方法 3
AWS Glue へのアクセス 4
関連サービス 4
使用方法 5
独立で実行されるサーバーレス ETL ジョブ 6
概念 7
AWS Glue の用語 9
コンポーネント 12
AWS Glue コンソール 12
AWS Glue Data Catalog 13
AWS Glue クローラおよび分類子 14
AWS Glue ETL オペレーション 14
AWS Glue でのストリーミング ETL 14
AWS Glue ジョブシステム 15
ビジュアル ETL コンポーネント 15
AWS Glue for Spark と AWS Glue for Ray 21
AWS Glue for Ray とは 22
半構造化されたスキーマをリレーショナルスキーマに変換する 23
AWS Glue 型 25
AWS Glue データカタログ型 25
Spark スクリプトを使用した AWS Glue の型 25
AWS Glue クローラーの型 26
使用開始方法 27
AWS Glue クローン作成の概要 27
IAM アクセス許可のセットアップ 29
次のステップ 34
AWS Glue Studio の設定 34
AWS Glue Studio 中でのノートブックの使用開始 47
使用状況プロファイルの設定 50
使用プロファイルの管理 51
使用プロファイルとジョブ 64

AWS Glue Data Catalog の開始方法 65
概要 65
ステップ 1: データベースを作成する 65
ステップ 2。テーブルを作成する 67
次のステップ 68
データストアへのネットワークアクセスを設定する 72
AWS Glue の PyPI に接続する VPC のセットアップ 73
VPC での DNS のセットアップ 75
暗号化のセットアップ 75
開発のためのネットワーク設定 80
開発エンドポイント用にネットワークを設定する 80
ノートブックサーバー用の Amazon EC2 のセットアップ 82
データ検出とカタログ化 84
データカタログの入力 86
AWS Glue クローラー の使用 87
メタデータの手動定義 176
他の AWS のサービスとの統合 196
データカタログの設定 197
トランザクションテーブルへのデータ入力と管理 200
Iceberg テーブルの作成 200
Iceberg テーブルの最適化 204
Iceberg テーブルのクエリパフォーマンスの最適化 234
データカタログの管理 236
スキーマを更新し、新規パーティションを追加する 237
列統計を使用したクエリのパフォーマンスの最適化 245
Data Catalog の暗号化 269
Lake Formation を使用したデータカタログの保護 269
AWS Glue Data Catalog ビューの使用 269
データカタログにアクセスする 274
AWS Glue Iceberg REST エンドポイントを使用したデータカタログへの接続方法 275
AWS Glue Iceberg REST 拡張エンドポイントを使用したデータカタログへの接続 278
AWS Glue Apache Iceberg 用の REST API 278
スタンドアロンの Spark アプリケーションからデータカタログへの接続 298
Amazon Redshift と Apache Iceberg 間のデータマッピング 299
AWS Glue Iceberg REST Catalog API を使用する際の考慮事項と制限事項 301
データカタログのベストプラクティス 301

Amazon CloudWatch での Data Catalog 使用状況メトリクスのモニタリング 302
Data Catalog メトリクスの概要 302
CloudWatch ダッシュボードにメトリクスを追加する 303
..... 304
AWS Glue スキーマレジストリ 305
スキーマ 306
レジストリ 308
スキーマのバージョニングと互換性 309
オープンソース Serde ライブラリ 314
Schema Registry のクォータ 314
仕組み 315
使用開始 317
AWS Glue Schema Registry との統合 340
AWS Glue スキーマレジストリへの移行 367
データへの接続 370
コネクタと接続の使用に関する概要 370
統合接続 371
サポートされている認可タイプ 375
考慮事項 375
使用可能な接続 376
制限 379
AWS Glue 接続プロパティ 379
必要な接続プロパティ 380
JDBC 接続プロパティ 381
MongoDB と MongoDB Atlas の接続プロパティ 386
Salesforce 接続プロパティ 387
Snowflake 接続 388
Vertica 接続 389
SAP HANA 接続 390
Azure SQL 接続 391
Teradata Vantage 接続 391
OpenSearch Service 接続 392
Azure Cosmos 接続 393
SSL 接続プロパティ 393
認証用の Kafka 接続プロパティ 396
Google BigQuery 接続 397

Vertica 接続 389
での接続認証情報の保存 AWS Secrets Manager 398
AWS Glue 接続の追加 399
Adobe Analytics に対する接続 401
Adobe Marketo Engage への接続 409
Amazon Redshift に接続する 419
Asana に対する接続 424
Azure Cosmos DB に対する接続 433
Azure SQL に対する接続 436
Blackbaud Raiser's Edge NXT への接続 440
CircleCI への接続 449
Datadog に対する接続 461
Docusign Monitor への接続 468
Domo への接続 476
Dynatrace への接続 484
Facebook 広告への接続 490
Facebook Page Insights に対する接続 498
Freshdesk に対する接続 520
Freshsales に対する接続 530
Google 広告に対する接続 537
Google Analytics 4 への接続 564
Google BigQuery に対する接続 576
Google 検索コンソールに対する接続 581
Google Sheets への接続 591
への接続 HubSpot 598
Instagram 広告への接続 624
Intercom への接続 631
Jira Cloud への接続 654
Kustomer への接続 679
LinkedIn に対する接続 707
Mailchimp への接続 715
Microsoft Teams への接続 725
Mixpanel に対する接続 736
Monday への接続 751
MongoDB に対する接続 759
Oracle への接続 NetSuite 763

OpenSearch Service に対する接続 774
Okta への接続 777
PayPal に対する接続 786
Pendo への接続 793
Pipedrive への接続 802
Productboard への接続 810
QuickBooks に対する接続 819
Salesforce への接続 827
Salesforce Marketing Cloud への接続 839
Salesforce Commerce Cloud への接続 864
Salesforce Marketing Cloud Account Engagement への接続 874
SAP HANA に対する接続 887
SAP OData への接続 891
SendGrid に対する接続 913
への接続 ServiceNow 922
Slack への接続 933
Smartsheet に対する接続 945
Snapchat 広告への接続 957
Snowflake に接続する 965
Stripe への接続 970
Teradata に対する接続 1018
Twilio に対する接続 1022
Vertica に対する接続 1041
WooCommerce に対する接続 1045
Zendesk への接続 1072
Zoho CRM への接続 1100
Zoom Meetings に対する接続 1111
独自の JDBC ドライバーを使用した JDBC 接続の追加 1121
カスタムコネクタと接続の使用 1126
カスタムコネクタを作成する 1128
コネクタ用の接続を作成する 1132
カスタムコネクタを使用したジョブのオーサリング 1140
コネクタと接続を管理する 1148
カスタムコネクタの開発 1151
AWS Glue Studio でのコネクタおよび接続の使用に関する制約事項 1153
AWS Glue 接続のテスト 1154

を経由するように AWS 通話を設定する VPC 1154
のデータJDBCストアへの接続 VPC 1155
Elastic Network Interface を使用したVPCデータへのアクセス 1156
Elastic Network Interface プロパティ 1157
MongoDB または MongoDB Atlas 接続を使用する 1157
VPC エンドポイントを使用した Amazon S3 データストアのクロール 1158
前提条件 1158
Amazon S3 への接続を作成する 1160
Amazon S3 への接続をテストする 1162
Amazon S3 データストアのクローラーを作成する 1164
Amazon S3 backed データカタログテーブルのクローラーを作成する 1166
クローラーの実行 1167
トラブルシューティング 1167
AWS Lake Formation で保護されたデータの操作 1167
フルテーブルアクセス 1167
接続の問題のトラブルシューティング 1169
チュートリアル: Elasticsearch 向けの AWS Glue コネクタを使用する 1170
前提条件 1170
ステップ 1: (オプション) OpenSearch クラスター情報の AWS シークレットを作成する .. 1171
ステップ 2: コネクタをサブスクライブする 1171
ステップ 3: AWS Glue Studio でコネクタをアクティブにして、接続を作成します。 1173
ステップ 4: ETL ジョブの IAM ロールを設定する 1173
ステップ 5: OpenSearch 接続を使用するジョブを作成する 1174
ステップ 6: ジョブを実行する 1175
インタラクティブセッションで AWS Glue ジョブを構築する 1177
AWS Glue インタラクティブセッションの概要 1177
制限 1178
AWS Glue のインタラクティブセッションの開始方法 1178
ローカルでインタラクティブセッションを設定するための前提条件 1178
Jupyter と AWS Glue インタラクティブセッション Jupyter カーネルのインストール 1178
Jupyter の実行 1178
セッション認証情報とリージョンの設定 1179
インタラクティブセッションプレビューからのアップグレード 1181
SageMaker AI Studio でのインタラクティブセッションの使用 1181
Microsoft Visual Studio Code によるインタラクティブセッションの使用 1182
IAM を使用したインタラクティブセッション 1185

AWS Glue Jupyter および AWS Glue Studio ノートブックのインタラクティブセッションの設定 1193
Jupyter Magics の概要 1193
Jupyter 向け AWS Glue インタラクティブセッションでサポートされているマジック 1194
ネーミングセッション 1212
インタラクティブセッションの IAM ロールの指定 1213
名前付きプロファイルを使用したセッションの設定 1213
スクリプトまたはノートブックの AWS Glue ジョブジョブへの変換 1215
AWS Glue インタラクティブセッションでのストリーミングオペレーションの使用 1215
ストリーミングセッションタイプの切り替え 1215
インタラクティブな開発のためのサンプリング入力ストリーム 1215
インタラクティブセッションでのストリーミングアプリケーションの実行 1217
ローカルでの開発とテスト 1218
AWS Glue Studio を使用した開発 1219
インタラクティブセッションによる開発 1219
Docker を使用したローカルでの AWS Glue ジョブの開発 1219
開発者エンドポイント 1229
開発エンドポイントからインタラクティブセッションへ移行する 1231
開発エンドポイントを使用してスクリプトを開発する 1232
ノートブックの管理 1261
AWS Glue Studio でビジュアル ETL ジョブを作成する 1263
コンソールにサインインする 1263
AWS Glue Studio でジョブを作成するための次のステップ 1264
AWS Glue Studio でビジュアル ETL ジョブを開始する 1264
AWS Glue Studio でのジョブの開始 1264
ジョブエディタの機能 1266
AWS Glue マネージド変換によるデータの変換 1274
カスタムビジュアル変換によるデータの変換 1348
データレイクフレームワークを AWS Glue Studio で使用する 1365
データソースへの接続 1377
データターゲットノードの設定 1386
ジョブスクリプトの編集またはアップロード 1391
ジョブ図でノードの親ノードを変更する 1397
ジョブ図からノードを削除する 1397
AWS Glue データカタログノードにソースパラメータとターゲットパラメータを追加する 1402

AWS Glue で Git のバージョン管理システムを使用する 1404
AWS Glue Studio ノートブックによるコードの作成 1413
制約事項 1413
ノートブックの使用の概要 1413
AWS Glue Studio の中のノートブックを使用した ETL ジョブの作成 1414
ノートブックエディタコンポーネント 1415
ノートブックとジョブスクリプトの保存 1417
ノートブックセッションの管理 1417
AWS Glue Studio ノートブックでの Amazon Q Developer の使用 1419
ジョブ実行のモニタリング 1420
ジョブモニタリングダッシュボードにアクセスする 1420
ジョブモニタリングダッシュボードの概要 1420
ジョブの実行ビュー 1420
ジョブの実行ログの表示 1425
ジョブの実行の詳細を表示する 1425
Spark ジョブ実行の Amazon CloudWatch メトリクスを表示 1428
Ray ジョブ実行の Amazon CloudWatch メトリクスを表示 1429
機密データを検出して処理する 1431
データのスキャン方法の選択 1431
検出する PII エンティティの選択 1433
検出感度のレベルの指定 1437
特定された PII データによる対処方法の選択 1438
詳細なアクションオーバーライドの追加 1439
ジョブの管理 1440
ジョブの実行の開始 1440
ジョブの実行のスケジュール 1441
ジョブスケジュールの管理 1442
ジョブの実行の停止 1443
ジョブの表示 1443
最近のジョブの実行の情報を表示する 1444
ジョブスクリプトの表示 1445
ジョブのプロパティを変更する 1446
ジョブの保存 1449
ジョブのクローンを作成する 1451
ジョブの削除 1451
ジョブの使用 1453

AWS Glue バージョン 1454
AWS Glue バージョン 1454
AWS Glue バージョンサポートポリシー 1473
Spark ジョブの AWS Glue の AWS Glue バージョン 5.0 への移行 1475
Spark ジョブの AWS Glue の AWS Glue バージョン 4.0 への移行 1490
AI によるアップグレード分析 1505
Spark ジョブの使用 1522
ジョブのパラメータ 1522
Spark と PySpark のジョブ 1532
ストリーミング ETLジョブ 1676
FindMatches によるレコードのマッチング 1691
Spark プログラムの移行 1725
Ray ジョブの使用 1732
AWS Glue for Ray の使用を開始する 1733
サポートされている Ray のランタイム環境 1734
Ray ジョブのワーカーの考慮 1735
Ray ジョブのパラメータ 1735
Ray ジョブメトリクス 1738
Python シェルジョブのプロパティの設定 1740
制限 1740
Python シェルジョブのジョブプロパティの定義 1741
サポートされている Python シェルジョブのライブラリ 1743
独自の Python ライブラリの提供 1745
AWS Glue の Python シェルジョブで AWS CloudFormation を使用する 1748
Python シェル 3.6 から Python シェル 3.9 への移行 1749
AWS Glue Python シェルジョブからの移行 1749
モニタリング 1751
AWSタグ 1752
EventBridge による自動化 1757
リソースのモニタリング 1760
CloudTrail による口グ記録 1763
ジョブ実行ステータス 1766
AWS Glue Streaming 1770
ストリーミングのユースケース 1770
AWS Glue Streaming を使用することの利点は何ですか。 1771
AWS Glue Streaming はどのようなときに使うべきですか。 1772

サポートされているデータソース 1773
サポートされるデータターゲット 1773
チュートリアル: AWS Glue Studio を使用して最初のストリーミングワークロードを構築する 1773
前提条件 1774
Amazon Kinesis からストリーミングデータを消費する 1774
チュートリアル: AWS Glue Studio ノートブックを使用して最初のストリーミングワークロードを構築する 1785
前提条件 1786
Amazon Kinesis からストリーミングデータを消費する 1786
Streaming の概念 1793
AWS Glue Streaming ジョブの構造 1793
ストリーミング接続 1797
Kafka 接続 1797
Kinesis 接続 1804
AWS Glue Streaming 自動スケーリング 1811
AWS Glue Studio の Auto-Scaling を有効にする 1811
AWS CLI または SDK を使用した Auto Scaling の有効化 1812
仕組み 1813
メンテナンスウィンドウ 1815
メンテナンスウィンドウの設定 1815
メンテナンスウィンドウの動作 1816
ジョブのモニタリング 1817
データ損失処理 1819
高度な AWS Glue のストリーミングの概念 1820
ストリームを処理する際の時間に関する考慮事項 1820
ウィンドウ処理 1821
遅延データとウォーターマークの処理 1826
AWS Glue Streaming ジョブのモニタリング 1828
AWS Glue ストリーミングメトリクスの視覚化 1829
AWS Glue ストリーミングメトリックの使用 1830
最高のパフォーマンスを得る方法 1835
ゼロ ETL 統合 1837
AWS Glue のゼロ ETL 機能 1837
前提条件 1838
ソースリソースの設定 1839

ターゲットリソースの設定 1840
Amazon Redshift データウェアハウスの作成 1845
VPC の設定 1845
クロスアカウント統合の設定 1849
ソースの設定 1850
特別な SAP エンティティのサポート 1850
Amazon DynamoDB ソース 1850
Salesforce ソース 1851
Salesforce MCAE ソース 1852
追加の Salesforce 設定 1853
SAP OData データソース 1853
ServiceNow ソース 1853
Zendesk ソース 1853
Zoho CRM ソース 1854
Facebook 広告ソース 1855
Instagram 広告ソース 1856
サポートされていない Salesforce フィールド 1856
サポートされていない ServiceNow フィールド 1859
ターゲットの設定 1864
S3 ストレージを使用した SageMaker Lakehouse 1865
S3 Tables ターゲット 1866
Amazon Redshift ストレージを使用した SageMaker Lakehouse 1870
Amazon Redshift データウェアハウスターゲット 1871
統合ターゲットの設定 1872
パーティションとスキーマのネスト解除 1873
デフォルトのネスト解除とパーティショニングの動作 1874
スキーマのネスト解除 1874
ゼロ ETL 統合のデータパーティショニング 1878
制限 1888
一般的なタスク 1889
統合の作成 1889
統合の変更 1891
統合の削除 1891
API の使用 1891
統合のモニタリング 1892
統合状態 1892

ログの表示 1892
メトリクスの表示 1895
イベント通知 1895
制限 1896
AWS Glue Data Quality 1907
利点と主な特徴 1907
仕組み 1908
AWS Glue Data Catalog のデータ品質 1908
AWS Glue ETL ジョブのデータ品質 1909
AWS Glue Data Quality エントリポイントの比較 1909
考慮事項 1911
用語 1911
制限 1912
AWS Glue Data Quality のリリースノート 1912
一般提供を開始: 新機能 1913
2023 年 11 月 27 日 (プレビュー) 1913
2024 年 3 月 12 日 1913
2024 年 6 月 26 日 1913
2024 年 8 月 7 日 1914
2024 年 11 月 22 日 1914
2024 年 12 月 6 日 1914
2025 年 6 月 30 日 1914
AWS Glue Data Quality での異常検出 1914
仕組み 1915
異常検出アルゴリズムの詳細 1920
AWS Glue Data Quality の IAM アクセス許可 1921
IAM アクセス許可 1921
評価実行のスケジューリングに必要な IAM の設定 1924
IAM ポリシーの例 1925
Data Catalog で AWS Glue Data Quality を使用する 1930
前提条件 1930
手順の例 1931
ルールの推奨事項の生成 1931
ルールの推奨のモニタリング 1933
推奨ルールセットの編集 1933
新しいルールセットの作成 1934

ルールセットを実行してデータ品質を評価する 1936
データ品質スコアと結果を表示する 1937
関連トピック 1938
AWS Glue Studio を使用したデータ品質の評価 1938
利点 1939
AWS Glue Studio で ETL ジョブのデータ品質を評価する 1939
Data Quality ルールビルダー 1950
AWS Glue ETL ジョブでの異常検出の設定 1955
データ品質のスコアと異常を表示 1958
AWS Glue Studio ノートブックにおける ETL ジョブ向けの Data Quality 1961
前提条件 1962
AWS Glue Studio での ETL ジョブの作成 1962
データ品質定義言語 (DQDL) リファレンス 1967
構文 1969
ルールタイプリファレンス 1985
API を使用したデータ品質の測定および管理 2047
前提条件 2047
AWS Glue Data Quality 推奨事項の操作 2047
AWS Glue Data Quality ルールセットの操作 2050
AWS Glue Data Quality 実行の操作 2053
AWS Glue Data Quality 評価結果の操作 2058
アラート、デプロイ、スケジュールの設定 2064
Amazon EventBridge 統合でのアラートと通知の設定 2064
CloudWatch 統合でアラートと通知をセットアップする 2072
データ品質評価の結果をクエリする 2073
データ品質ルールをデプロイする 2077
データ品質ルールのスケジューリング 2077
AWS Glue Data Quality の保管中のデータ暗号化 2078
AWS 所有キー 2078
カスタマーマネージドキー 2078
カスタマーマネージドキーを作成 2081
セキュリティ設定を作成する 2083
AWS Glue Data Quality 暗号化コンテキスト 2084
AWS Glue Data Quality の暗号化キーのモニタリング 2085
詳細はこちら 2089
AWS Glue Data Quality のエラーのトラブルシューティング 2090

エラー: モジュールが見つからない 2091
エラー: アクセス許可が不十分である 2091
エラー: ルールセットが一意ではない 2091
エラー: テーブルに特殊文字が含まれている 2091
エラー: 大規模なルールセットによるオーバーフロー 2091
エラー: ルールのステータスが失敗である 2092
AnalysisException: Unable to verify existence of default database 2092
指定されたキーマップは所定のデータフレームに適していません 2092
java.lang.RuntimeException : Failed to fetch data. 2093
LAUNCH ERROR: Error downloading from S3 for bucket 2093
InvalidInputException (status: 400): DataQuality rules cannot be parsed 2094
エラー: EventBridge が、設定したスケジュールに基づいて Glue DQ ジョブをトリガーしない 2094
CustomSQL エラー 2094
動的ルール 2095
ユーザークラスの例外: org.apache.spark.sql.AnalysisException: org.apache.hadoop.hive.ql.metadata.HiveException 2097
UNCLASSIFIED_ERROR; IllegalArgumentException : 解析エラー: ルールまたはアナライザーが指定されていません。、入力時に実行可能な代替手段がありません。 2098
AWS Glue の Amazon Q データ統合 2099
Amazon Q とは 2099
AWS Glue の Amazon Q データ統合 2099
Amazon Q データ統合の操作 2100
ベストプラクティス 2102
サービスの改善 2103
考慮事項 2103
Amazon Q データ統合の設定 2103
IAM 許可の設定 2103
サポートされているコード生成 2106
インタラクションの例 2107
Amazon Q チャットインタラクション 2107
AWS Glue Studio ノートブックのインタラクション 2108
コンテキスト認識の使用 2112
インタラクションの例 2112
制限 2116
オーケストレーション 2117

トリガーを使用したジョブとクローラの開始 2117
AWS Glue トリガー 2117
トリガーの追加 2120
トリガーの有効化と無効化 2124
設計図とワークフローを使用した複雑な ETL アクティビティの実行 2125
ワークフローの概要 2125
ワークフローの手動による作成と構築 2129
EventBridge イベントを使用してのワークフローの開始 2134
ワークフローを開始した EventBridge イベントの表示 2141
ワークフローの実行およびモニタリング 2142
ワークフロー実行の停止 2145
ワークフロー実行の修復と再開 2146
ワークフローの実行プロパティの取得と設定 2152
AWS Glue APIを使用したワークフローのクエリ 2153
設計図とワークフローの制限 2157
設計図エラーのトラブルシューティング 2158
設計図のペルソナとロールのアクセス許可 2163
設計図の開発 2168
設計図の概要 2168
設計図の開発 2172
設計図の登録 2197
設計図の表示 2200
設計図の更新 2201
設計図からのワークフローの作成 2204
設計図の実行の表示 2206
AWS Glue の場合は AWS CloudFormation 2208
サンプルデータベース 2210
サンプルのデータベース、テーブル、パーティション 2211
サンプルの Grok 分類子 2215
サンプルの JSON 分類子 2216
サンプルの XML 分類子 2217
Amazon S3 クローラーのサンプル 2218
サンプルの接続 2221
サンプルの JDBC クローラー 2222
Amazon S3 から Amazon S3 へのサンプルジョブ 2225
Amazon S3 に書き込む JDBC のサンプルジョブ 2226

サンプルのオンデマンドトリガー ... 2228
サンプルのスケジュールされたトリガー ... 2229
サンプルの条件付きトリガー ... 2230
機械学習変換のサンプル ... 2232
サンプルのデータ品質ルールセット ... 2233
EventBridge スケジューラを使用するサンプルのデータ品質ルールセット ... 2234
サンプルの開発エンドポイント ... 2236
AWS Glue プログラミングガイド ... 2239
独自のカスタムスクリプトの提供 ... 2239
AWS Glue for Spark ... 2240
チュートリアル: Spark スクリプトの記述 ... 2241
PySpark での ETL ... 2254
Scala での ETL ... 2487
機能と最適化 ... 2572
AWS Glue for Ray ... 2835
チュートリアル: Ray スクリプトを記述する ... 2836
AWS Glue for Ray での Ray Core と Ray Data の使用 ... 2842
ファイルと Python ライブラリ ... 2844
データへの接続 ... 2849
AWS SDK の操作 ... 2851
AWS Glue API ... 2853
セキュリティ ... 2880
– データ型 – ... 2880
DataCatalogEncryptionSettings ... 2881
EncryptionAtRest ... 2881
ConnectionPasswordEncryption ... 2882
EncryptionConfiguration ... 2883
S3Encryption ... 2883
CloudWatchEncryption ... 2883
JobBookmarksEncryption ... 2884
SecurityConfiguration ... 2884
GluePolicy ... 2884
DataQualityEncryption ... 2885
— 操作 — ... 2885
GetDataCatalogEncryptionSettings (get_data_catalog_encryption_settings) ... 2886
PutDataCatalogEncryptionSettings (put_data_catalog_encryption_settings) ... 2887

PutResourcePolicy (put_resource_policy) 2887
GetResourcePolicy (get_resource_policy) 2889
DeleteResourcePolicy (delete_resource_policy) 2889
CreateSecurityConfiguration (create_security_configuration) 2890
DeleteSecurityConfiguration (delete_security_configuration) 2891
GetSecurityConfiguration (get_security_configuration) 2892
GetSecurityConfigurations (get_security_configurations) 2892
GetResourcePolicies (get_resource_policies) 2893
カタログオブジェクト 2894
カタログ 2894
データベース 2906
テーブル 2916
パーティション 2960
接続 2984
ユーザー定義関数 3024
Athena カタログをインポートする 3031
テーブルオプティマイザー 3033
— データ型 — 3033
TableOptimizer 3033
TableOptimizerConfiguration 3034
TableOptimizerVpcConfiguration 3035
CompactionConfiguration 3035
IcebergCompactionConfiguration 3035
TableOptimizerRun 3036
BatchGetTableOptimizerEntry 3038
BatchTableOptimizer 3038
BatchGetTableOptimizerError 3039
RetentionConfiguration 3039
IcebergRetentionConfiguration 3040
OrphanFileDeletionConfiguration 3040
IcebergOrphanFileDeletionConfiguration 3040
CompactionMetrics 3041
RetentionMetrics 3041
OrphanFileDeletionMetrics 3041
IcebergCompactionMetrics 3042
IcebergRetentionMetrics 3042

IcebergOrphanFileDeletionMetrics 3042
RunMetrics 3043
— 操作 — 3043
GetTableOptimizer (get_table_optimizer) 3044
BatchGetTableOptimizer (batch_get_table_optimizer) 3045
ListTableOptimizerRuns (list_table_optimizer_runs) 3046
CreateTableOptimizer (create_table_optimizer) 3047
DeleteTableOptimizer (delete_table_optimizer) 3048
UpdateTableOptimizer (update_table_optimizer) 3049
クローラーおよび分類子 3050
分類子 3050
クローラー 3064
列統計 3091
スケジューラー 3106
ETL スクリプトの自動生成 3109
– データ型 – 3109
CodeGenNode 3109
CodeGenNodeArg 3109
CodeGenEdge 3110
ロケーション 3110
CatalogEntry 3111
MappingEntry 3111
— 操作 — 3112
CreateScript (create_script) 3112
GetDataflowGraph (get_dataflow_graph) 3113
GetMapping (get_mapping) 3113
GetPlan (get_plan) 3114
ビジュアルジョブ API 3116
– データタイプ – 3116
CodeGenConfigurationNode 3120
JDBCConnectorOptions 3126
StreamingDataPreviewOptions 3128
AthenaConnectorSource 3128
jdbcConnectorSource 3129
SparkConnectorSource 3130
CatalogSource 3130

MySQLCatalogSource 3131
PostgreSQLCatalogSource 3131
OracleSQLCatalogSource 3132
MicrosoftSQLServerCatalogSource 3132
CatalogKinesisSource 3132
DirectKinesisSource 3133
KinesisStreamingSourceOptions 3134
CatalogKafkaSource 3137
DirectKafkaSource 3137
KafkaStreamingSourceOptions 3138
RedShiftSource 3140
AmazonRedshiftSource 3141
AmazonRedshiftNodeData 3141
AmazonRedshiftAdvancedOption 3144
オプション 3144
S3CatalogSource 3144
S3SourceAdditionalOptions 3145
S3CsvSource 3145
DirectJDBCSource 3148
S3DirectSourceAdditionalOptions 3148
S3jsonSource 3149
S3parquetSource 3150
S3DeltaSource 3152
S3CatalogDeltaSource 3152
CatalogDeltaSource 3153
S3HudiSource 3154
S3CatalogHudiSource 3154
S3ExcelSource 3155
CatalogHudiSource 3156
DynamoDBCatalogSource 3157
RelationalCatalogSource 3157
jdbcConnectorTarget 3158
SparkConnectorTarget 3159
BasicCatalogTarget 3160
MySQLCatalogTarget 3160
PostgreSQLCatalogTarget 3161

OracleSQLCatalogTarget 3161
MicrosoftSQLServerCatalogTarget 3162
RedshiftTarget 3162
AmazonRedshiftTarget 3163
UpsertRedshiftTargetOptions 3163
S3CatalogTarget 3164
S3GlueParquetTarget 3164
CatalogSchemaChangePolicy 3165
S3directTarget 3166
S3HudiCatalogTarget 3166
S3HudiDirectTarget 3167
S3DeltaCatalogTarget 3168
S3DeltaDirectTarget 3169
S3HyperDirectTarget 3170
S3IcebergDirectTarget 3171
DirectSchemaChangePolicy 3172
ApplyMapping 3173
Mapping 3173
SelectFields 3174
DropFields 3174
RenameField 3175
スピゴット 3175
Join 3176
JoinColumn 3177
SplitFields 3177
SelectFromCollection 3177
FillMissingValues 3178
フィルター 3178
FilterExpression 3179
FilterValue 3179
CustomCode 3180
SparkSQL 3180
SqlAlias 3181
DropNullFields 3181
NullCheckboxList 3182
NullValueField 3182

Datatype 3183
結合 3183
Union 3184
PIIDetection 3184
集計 3185
DropDuplicates 3186
GovernedCatalogTarget 3186
GovernedCatalogSource 3187
AggregateOperation 3187
GlueSchema 3188
GlueStudioSchemaColumn 3188
GlueStudioColumn 3188
DynamicTransform 3189
TransformConfigParameter 3190
EvaluateDataQuality 3191
DQResultsPublishingOptions 3191
DQStopJobOnFailureOptions 3192
EvaluateDataQualityMultiFrame 3192
レシピ 3193
RecipeReference 3194
SnowflakeNodeData 3194
SnowflakeSource 3196
SnowflakeTarget 3197
ConnectorDataSource 3197
ConnectorDataTarget 3198
RecipeStep 3199
RecipeAction 3199
ConditionExpression 3200
ジョブ 3200
ジョブ 3200
ジョブ実行 3225
トリガー 3244
統合 API 3258
— データ型 — 3258
Integration 3258
IntegrationConfig 3260

IntegrationPartition 3260
IntegrationError 3261
IntegrationFilter 3262
InboundIntegration 3262
SourceProcessingProperties 3263
TargetProcessingProperties 3263
SourceTableConfig 3264
TargetTableConfig 3264
— 操作 — 3265
CreateIntegration (create_integration) 3265
ModifyIntegration (modify_integration) 3268
DescribeIntegrations (describe_integrations) 3271
DeleteIntegration (delete_integration) 3272
DescribeInboundIntegrations (describe_inbound_integrations) 3274
CreateIntegrationTableProperties (create_integration_table_properties) 3275
UpdateIntegrationTableProperties (update_integration_table_properties) 3277
GetIntegrationTableProperties (get_integration_table_properties) 3278
DeleteIntegrationTableProperties (delete_integration_table_properties) 3279
CreateIntegrationResourceProperty (create_integration_resource_property) 3280
UpdateIntegrationResourceProperty (update_integration_resource_property) 3281
GetIntegrationResourceProperty (get_integration_resource_property) 3282
UntagResource (untag_resource) 3283
ListTagsForResource (list_tags_for_resource) 3284
— 例外 — 3284
ResourceNotFoundException 3285
InternalServerException 3285
IntegrationAlreadyExistsFault 3285
IntegrationConflictOperationFault 3285
IntegrationQuotaExceededFault 3286
KMSKeyNotAccessibleFault 3286
IntegrationNotFoundFault 3286
TargetResourceNotFound 3286
InvalidIntegrationStateFault 3287
インタラクティブセッション 3287
— データ型 — 3287
セッション 3287

SessionCommand 3290
Statement 3290
StatementOutput 3291
StatementOutputData 3291
ConnectionsList 3292
— 操作 — 3292
CreateSession (create_session) 3292
StopSession (stop_session) 3295
DeleteSession (delete_session) 3296
GetSession (get_session) 3297
ListSessions (list_sessions) 3298
RunStatement (run_statement) 3299
CancelStatement (cancel_statement) 3300
GetStatement (get_statement) 3301
ListStatements (list_statements) 3301
DevEndpoints 3302
– データ型 – 3302
DevEndpoint 3303
DevEndpointCustomLibraries 3306
— 操作 — 3307
CreateDevEndpoint (create_dev_endpoint) 3307
UpdateDevEndpoint (update_dev_endpoint) 3313
DeleteDevEndpoint (delete_dev_endpoint) 3314
GetDevEndpoint (get_dev_endpoint) 3315
GetDevEndpoints (get_dev_endpoints) 3316
BatchGetDevEndpoints (batch_get_dev_endpoints) 3317
ListDevEndpoints (list_dev_endpoints) 3317
スキーマレジストリ 3319
– データ型 – 3319
RegistryId 3319
RegistryListItem 3320
MetadataInfo 3320
OtherMetadataValueListItem 3321
SchemaListItem 3321
SchemaVersionListItem 3322
MetadataKeyValuePair 3322

SchemaVersionErrorItem 3322
ErrorDetails 3323
SchemaVersionNumber 3323
SchemaId 3323
— 操作 — 3324
CreateRegistry (create_registry) 3325
CreateSchema (create_schema) 3326
GetSchema (get_schema) 3330
ListSchemaVersions (list_schema_versions) 3332
GetSchemaVersion (get_schema_version) 3333
GetSchemaVersionsDiff (get_schema_versions_diff) 3334
ListRegistries (list_registries) 3335
ListSchemas (list_schemas) 3336
RegisterSchemaVersion (register_schema_version) 3337
UpdateSchema (update_schema) 3338
CheckSchemaVersionValidity (check_schema_version_validity) 3340
UpdateRegistry (update_registry) 3341
GetSchemaByDefinition (get_schema_by_definition) 3342
GetRegistry (get_registry) 3343
PutSchemaVersionMetadata (put_schema_version_metadata) 3344
QuerySchemaVersionMetadata (query_schema_version_metadata) 3345
RemoveSchemaVersionMetadata (remove_schema_version_metadata) 3347
DeleteRegistry (delete_registry) 3348
DeleteSchema (delete_schema) 3349
DeleteSchemaVersions (delete_schema_versions) 3350
ワークフロー 3351
– データ型 – 3351
JobNodeDetails 3352
CrawlerNodeDetails 3352
TriggerNodeDetails 3352
Crawl 3353
ノード 3353
Edge 3354
ワークフロー 3354
WorkflowGraph 3356
WorkflowRun 3356

WorkflowRunStatistics 3357
StartingEventBatchCondition 3358
Blueprint 3358
BlueprintDetails 3359
LastActiveDefinition 3360
BlueprintRun 3360
— 操作 — 3362
CreateWorkflow (create_workflow) 3362
UpdateWorkflow (update_workflow) 3364
DeleteWorkflow (delete_workflow) 3365
GetWorkflow (get_workflow) 3366
ListWorkflows (list_workflows) 3366
BatchGetWorkflows (batch_get_workflows) 3367
GetWorkflowRun (get_workflow_run) 3368
GetWorkflowRuns (get_workflow_runs) 3369
GetWorkflowRunProperties (get_workflow_run_properties) 3370
PutWorkflowRunProperties (put_workflow_run_properties) 3370
CreateBlueprint (create_blueprint) 3372
UpdateBlueprint (update_blueprint) 3373
DeleteBlueprint (delete_blueprint) 3373
ListBlueprints (list_blueprints) 3374
BatchGetBlueprints (batch_get_blueprints) 3375
StartBlueprintRun (start_blueprint_run) 3376
GetBlueprintRun (get_blueprint_run) 3377
GetBlueprintRuns (get_blueprint_runs) 3377
StartWorkflowRun (start_workflow_run) 3378
StopWorkflowRun (stop_workflow_run) 3379
ResumeWorkflowRun (resume_workflow_run) 3380
使用状況プロファイル 3381
— データ型 — 3381
ProfileConfiguration 3381
ConfigurationObject 3382
UsageProfileDefinition 3382
— 操作 — 3383
CreateUsageProfile (create_usage_profile) 3383
GetUsageProfile (get_usage_profile) 3384

UpdateUsageProfile (update_usage_profile) 3385
DeleteUsageProfile (delete_usage_profile) 3386
ListUsageProfiles (list_usage_profiles) 3386
機械学習 3387
– データ型 – 3387
TransformParameters 3388
EvaluationMetrics 3388
MLTransform 3389
FindMatchesParameters 3392
FindMatchesMetrics 3393
ConfusionMatrix 3394
GlueTable 3395
TaskRun 3395
TransformFilterCriteria 3396
TransformSortCriteria 3397
TaskRunFilterCriteria 3398
TaskRunSortCriteria 3398
TaskRunProperties 3399
FindMatchesTaskRunProperties 3399
ImportLabelsTaskRunProperties 3400
ExportLabelsTaskRunProperties 3400
LabelingSetGenerationTaskRunProperties 3400
SchemaColumn 3401
TransformEncryption 3401
MLUserDataEncryption 3401
ColumnImportance 3402
— 操作 — 3402
CreateMLTransform (create_ml_transform) 3403
UpdateMLTransform (update_ml_transform) 3406
DeleteMLTransform (delete_ml_transform) 3408
GetMLTransform (get_ml_transform) 3409
GetMLTransforms (get_ml_transforms) 3412
ListMLTransforms (list_ml_transforms) 3413
StartMLEvaluationTaskRun (start_ml_evaluation_task_run) 3414
StartMLLabelingSetGenerationTaskRun (start_ml_labeling_set_generation_task_run) 3415
GetMLTaskRun (get_ml_task_run) 3416

GetMLTaskRuns (get_ml_task_runs) 3417
CancelMLTaskRun (cancel_ml_task_run) 3419
StartExportLabelsTaskRun (start_export_labels_task_run) 3420
StartImportLabelsTaskRun (start_import_labels_task_run) 3421
Data Quality 3422
— データ型 — 3422
DataSource 3423
DataQualityRulesetListDetails 3423
DataQualityTargetTable 3424
DataQualityRulesetEvaluationRunDescription 3424
DataQualityRulesetEvaluationRunFilter 3425
DataQualityEvaluationRunAdditionalRunOptions 3425
DataQualityRuleRecommendationRunDescription 3426
DataQualityRuleRecommendationRunFilter 3426
DataQualityResult 3427
DataQualityAnalyzerResult 3428
DataQualityObservation 3429
MetricBasedObservation 3429
DataQualityMetricValues 3430
DataQualityRuleResult 3430
DataQualityResultDescription 3431
DataQualityResultFilterCriteria 3432
DataQualityRulesetFilterCriteria 3432
DataQualityAggregatedMetrics 3433
StatisticAnnotation 3434
TimestampedInclusionAnnotation 3434
AnnotationError 3435
DatapointInclusionAnnotation 3435
StatisticSummaryList 3436
StatisticSummary 3436
RunIdentifier 3437
StatisticModelResult 3437
— 操作 — 3438
StartDataQualityRulesetEvaluationRun (start_data_quality_ruleset_evaluation_run) 3439
CancelDataQualityRulesetEvaluationRun (cancel_data_quality_ruleset_evaluation_run) 3441
GetDataQualityRulesetEvaluationRun (get_data_quality_ruleset_evaluation_run) 3441

ListDataQualityRulesetEvaluationRuns (list_data_quality_ruleset_evaluation_runs) 3443
StartDataQualityRuleRecommendationRun (start_data_quality_rule_recommendation_run) 3444
CancelDataQualityRuleRecommendationRun
(cancel_data_quality_rule_recommendation_run) 3445
GetDataQualityRuleRecommendationRun (get_data_quality_rule_recommendation_run) .. 3446
ListDataQualityRuleRecommendationRuns (list_data_quality_rule_recommendation_runs) 3448
GetDataQualityResult (get_data_quality_result) 3449
BatchGetDataQualityResult (batch_get_data_quality_result) 3451
ListDataQualityResults (list_data_quality_results) 3451
CreateDataQualityRuleset (create_data_quality_ruleset) 3452
DeleteDataQualityRuleset (delete_data_quality_ruleset) 3454
GetDataQualityRuleset (get_data_quality_ruleset) 3454
ListDataQualityRulesets (list_data_quality_rulesets) 3455
UpdateDataQualityRuleset (update_data_quality_ruleset) 3456
ListDataQualityStatistics (list_data_quality_statistics) 3458
TimestampFilter 3459
CreateDataQualityRulesetRequest 3459
GetDataQualityRulesetResponse 3460
GetDataQualityResultResponse 3461
StartDataQualityRuleRecommendationRunRequest 3462
GetDataQualityRuleRecommendationRunResponse 3463
BatchPutDataQualityStatisticAnnotation (batch_put_data_quality_statistic_annotation) 3465
GetDataQualityModel (get_data_quality_model) 3465
GetDataQualityModelResult (get_data_quality_model_result) 3466
ListDataQualityStatisticAnnotations (list_data_quality_statistic_annotations) 3467
PutDataQualityProfileAnnotation (put_data_quality_profile_annotation) 3468
機密データ 3469
— データ型 — 3469
CustomEntityType 3469
— 操作 — 3469
CreateCustomEntityType (create_custom_entity_type) 3470
DeleteCustomEntityType (delete_custom_entity_type) 3471
GetCustomEntityType (get_custom_entity_type) 3471
BatchGetCustomEntityTypes (batch_get_custom_entity_types) 3472
ListCustomEntityTypes (list_custom_entity_types) 3473
API のタグ付け 3474

— データ型 — ... 3474
Tag ... 3474
— 操作 — ... 3475
TagResource (tag_resource) ... 3475
UntagResource (untag_resource) ... 3476
GetTags (get_tags) ... 3476
一般的なデータ型 ... 3477
Tag ... 3477
DecimalNumber ... 3478
ErrorDetail ... 3478
PropertyPredicate ... 3478
ResourceUri ... 3479
ColumnStatistics ... 3479
ColumnStatisticsError ... 3480
ColumnError ... 3480
ColumnStatisticsData ... 3480
BooleanColumnStatisticsData ... 3481
DateColumnStatisticsData ... 3482
DecimalColumnStatisticsData ... 3482
DoubleColumnStatisticsData ... 3483
LongColumnStatisticsData ... 3483
StringColumnStatisticsData ... 3484
BinaryColumnStatisticsData ... 3484
文字列パターン ... 3484
例外 ... 3487
AccessDeniedException ... 3487
AlreadyExistsException ... 3487
ConcurrentModificationException ... 3488
ConcurrentRunsExceededException ... 3488
CrawlerNotRunningException ... 3488
CrawlerRunningException ... 3488
CrawlerStoppingException ... 3489
EntityNotFoundException ... 3489
FederationSourceException ... 3489
FederationSourceRetryableException ... 3490
GlueEncryptionException ... 3490

IdempotentParameterMismatchException 3490
IllegalWorkflowStateException 3490
InternalServiceException 3491
InvalidExecutionEngineException 3491
InvalidInputException 3491
InvalidStateException 3492
InvalidTaskStatusTransitionException 3492
JobDefinitionErrorException 3492
JobRunInTerminalStateException 3492
JobRunInvalidStateTransitionException 3493
JobRunNotInTerminalStateException 3493
LateRunnerException 3493
NoScheduleException 3494
OperationTimeoutException 3494
ResourceNotReadyException 3494
ResourceNumberLimitExceededException 3494
SchedulerNotRunningException 3495
SchedulerRunningException 3495
SchedulerTransitioningException 3495
UnrecognizedRunnerException 3495
ValidationException 3496
VersionMismatchException 3496
AWS Glue API コード例 3497
基本 3507
Hello AWS Glue 3508
基本を学ぶ 3518
アクション 3654
セキュリティ 3799
データ保護 3800
保管中のデータの暗号化 3800
Encrypting data in transit 3818
FIPS 準拠 3819
キーの管理 3819
AWS の他のサービスへの AWS Glue の依存関係 3819
開発エンドポイント 3820
Identity and Access Management 3821

対象者 3822
アイデンティティによる認証 3822
ポリシーを使用したアクセス権の管理 3826
AWS Glue の仕組み IAM 3829
AWS Glue の IAM アクセス許可の設定 3836
AWS Glue のアクセスコントロールポリシーの例 3871
AWS マネージドポリシーの付与 3897
AWS Glue リソース ARN の指定 3906
クロスアカウントアクセス許可の付与 3915
トラブルシューティング 3923
AWS Lake Formation アクセスコントロールモデル 3925
AWS Lake Formation で AWS Glue を使用したフルテーブルアクセス 3925
FGAC のための Lake Formation 3937
AWS Glue での Amazon S3 Access Grants の使用 3950
AWS Glue が S3 Access Grants と連携する仕組み 3950
AWS Glue での S3 Access Grants の使用に関する考慮事項 3951
AWS Glue での S3 Access Grants の設定 3951
ログ記録とモニタリング 3954
コンプライアンス検証 3955
耐障害性 3956
インフラストラクチャセキュリティ 3956
AWS Glue のインターフェイス VPC エンドポイント (AWS PrivateLink) の設定 3957
共有 Amazon VPCs の設定 3959
AWS Glue のトラブルシューティング 3960
AWS Glue トラブルシューティング情報の収集 3960
Spark に関連するエラーのトラブルシューティング 3961
エラー: リソースを利用できません。 3962
エラー: VPC の subnetId に S3 エンドポイントまたは NAT ゲートウェイが見つかりませんでした 3962
エラー: 必要なセキュリティグループのインバウンドルール 3963
エラー: 必要なセキュリティグループのアウトバウンドルール 3963
エラー: ジョブの実行に失敗しました、渡されたロールには AWS Glue サービスでロールを引き受けるためのアクセス許可が必要です 3963
エラー:DescribeVpcEndpoints アクションは許可されていません。VPC ID を検証できません vpc-id 3964

エラー:DescribeRouteTables アクションは許可されていません。VPC id (vpc-id) の Subnet Id (Subnet id) を検証できません。 3964
エラー: ec2:DescribeSubnets の呼び出しに失敗しました 3964
エラー: ec2:DescribeSecurityGroups の呼び出しに失敗しました 3964
エラー: AZ のサブネットが見つかりませんでした 3964
エラー: JDBC ターゲットへの書き込み時のジョブ実行の例外 3964
Error: Amazon S3: The operation is not valid for the object's storage class 3965
エラー:Amazon S3 タイムアウト 3965
エラー: Amazon S3 へのアクセスが拒否されました 3966
エラー: Amazon S3 のアクセスキー ID が存在しません 3966
エラー: URI に s3a:// を使用しながら Amazon S3 にアクセスするとジョブ実行が失敗します 3966
エラー: Amazon S3 のサービストークンが有効期限切れです 3968
エラー: ネットワークインターフェイスのプライベート DNS が見つかりません 3968
エラー: 開発エンドポイントのプロビジョニングに失敗しました 3969
エラー: ノートブックサーバー CREATE_FAILED 3969
エラー: ローカルノートブックの起動に失敗する 3969
エラー: クローラの実行に失敗しました 3970
エラー: パーティションが更新されませんでした 3970
Error: Job bookmark update failed due to version mismatch (エラー: バージョンの不一致のため、ジョブのブックマークの更新に失敗しました) 3971
エラー : ジョブのブックマークが有効なときにジョブがデータを再処理しています 3971
エラー: AWS Glue 内の VPC 間のフェイルオーバー動作 3972
クローラーが Lake Formation アクセス許可を使用している場合のクローラーエラー 3973
エラー:「The S3 location: s3://examplepath is not registered」(S3 ロケーション: s3://examplepath は登録されていません) 3973
エラー:「User/Role is not authorized to perform: lakeformation:GetDataAccess on resource」(ユーザー/ロールは、リソースに対する lakeformation:GetDataAccess の実行が許可されていません) 3973
エラー:「Insufficient Lake Formation permission(s) on (Database name: exampleDatabase, Table Name: exampleTable)」(データベース名: exampleDatabase、テーブル名: exampleTable に対する Lake Formation へのアクセス許可が不十分です) 3974
エラー:「Insufficient Lake Formation permission(s) on s3://examplepaths」(s3://examplepath に対する Lake Formation へのアクセス許可が不十分です) 3974
Lake Formation の認証情報を使用したクローラーの設定に関するよくある質問 3975
Ray エラーをトラブルシューティングする 3976

Ray ジョブのログを検査する 3976
Ray ジョブのエラーをトラブルシューティングする 3977
AWS Glue 機械学習の例外 3979
CancelMLTaskRunActivity 3979
CreateMLTaskRunActivity 3979
DeleteMLTransformActivity 3980
GetMLTaskRunActivity 3980
GetMLTaskRunsActivity 3981
GetMLTransformActivity 3981
GetMLTransformsActivity 3981
GetSaveLocationForTransformArtifactActivity 3982
GetTaskRunArtifactActivity 3982
PublishMLTransformModelActivity 3983
PullLatestMLTransformModelActivity 3983
PutJobMetadataForMLTransformActivity 3984
StartExportLabelsTaskRunActivity 3984
StartImportLabelsTaskRunActivity 3984
StartMLEvaluationTaskRunActivity 3985
StartMLLabelingSetGenerationTaskRunActivity 3986
UpdateMLTransformActivity 3987
AWS Glue のクォータ 3988
AWS Glue パフォーマンスの向上 3989
ジョブタイプのチューニング戦略 3989
Spark パフォーマンスの向上 3989
プッシュダウンによる読み取りの最適化 3990
Amazon S3 に保存されているファイルの述語プッシュダウン 3990
JDBC ソースを操作するときのプッシュダウン 3991
AWS Glue のプッシュダウンに関する注意事項と制限事項 3994
での自動スケーリングの使用 AWS Glue 3994
要件 3995
で Auto Scaling を有効にする AWS Glue Studio 1811
または AWS CLIで Auto Scaling を有効にする SDK 1812
インタラクティブセッションによる Auto Scaling の有効化 3998
ヒントと注意点 3998
Amazon CloudWatch メトリクスによる Auto Scaling のモニタリング 3999
Amazon CloudWatch Logs による Auto Scaling のモニタリング 4000

Spark UI による Auto Scaling のモニタリング 4001
Auto Scaling ジョブの実行DPU使用状況のモニタリング 4001
制限 4002
実行に上限を設定してワークロードをパーティション化する 4002
ワークロードのパーティション化の有効化 4002
ジョブを自動的に実行する AWS Glue トリガーの設定 4004
既知の問題 4005
クロスジョブデータアクセスの防止 4005
ドキュメント履歴 4008
以前の更新 4069
AWS 用語集 4071

AWS Glue とは

AWS Glue は、分析を行うユーザーが複数のソースからのデータを簡単に検出、準備、移動、統合できるようにするサーバーレスのデータ統合サービスです。分析、機械学習、アプリケーション開発に使用できます。また、ジョブの作成、実行、ビジネスワークフローの実装のための生産性向上に役立つツールやデータ運用ツールも追加されています。

AWS Glue を使用すれば、70 を超える多様なデータソースを検出して接続し、一元化されたデータカタログでデータを管理できます。抽出、変換、ロード (ETL) パイプラインを視覚的に作成、実行、モニタリングして、データをデータレイクにロードできます。また、Amazon Athena、Amazon EMR、Amazon Redshift Spectrum を使用して、カタログ化されたデータをすぐに検索し、クエリできます。

AWS Glue は、主要なデータ統合機能を単一のサービスに統合します。これは、データ検出、最新の ETL、クリーニング、変換、一元化されたカタログ作成が含まれます。また、サーバーレスなので、管理するインフラストラクチャがありません。AWS Glue は、ETL、ELT、ストリーミングなどのすべてのワークロードを 1 つのサービスで柔軟にサポートすることで、さまざまなワークロードやユーザータイプのユーザーをサポートします。

また、AWS Glue アーキテクチャ全体でデータを簡単に統合できます。AWS 分析サービスと Amazon S3 データレイクを統合します。AWS Glue では、さまざまな技術的スキルセットに合わせてカスタマイズされたソリューションを備えており、デベロッパーからビジネスユーザーまで、すべてのユーザーが使いやすい統合インターフェイスとジョブ作成ツールを用意しています。

オンデマンドで拡張できる AWS Glue を使用すれば、データの価値を最大化する価値の高いアクティビティに集中できます。あらゆるデータサイズに合わせてスケーリングでき、すべてのデータ型とスキーマの差異をサポートします。俊敏性を高め、コストを最適化するため、AWS Glue は、組み込みの高可用性と従量制料金を提供します。

料金については、AWS Glue の料金を参照してください。

AWS Glue Studio

AWS Glue Studio は、AWS Glueでのデータ統合ジョブの作成、実行、モニタリングを容易にするグラフィカルインターフェイスです。データ変換ワークフローを視覚的に作成し、AWS Glue の Apache Spark ベースのサーバーレス ETL エンジンでシームレスに実行することができます。

AWS Glue Studio を使用すると、データを収集、変換、クリーニングするジョブを作成および管理できます。AWS Glue Studio を使用して、ジョブスクリプトのトラブルシューティングや編集ができます。

トピック

- AWS Glue の機能
- AWS Glue のイノベーションについて学ぶ
- AWS Glue の開始方法
- AWS Glue へのアクセス
- 関連サービス

AWS Glue の機能

AWS Glue の機能は、次の 3 つの主要なカテゴリに分類されます。

- データの検出と整理
- 分析用データの変換、準備、クリーニング
- データパイプラインの構築とモニタリング

データの検出と整理

- 複数のデータストアを統合して検索 – AWS ですべてのデータをカタログ化することで、複数のデータソースやシンクで保存、インデックス作成、検索を行うことができます。
- データを自動的に検出 – AWS Glue クローラーを使用して自動的にスキーマ情報を推測し、AWS Glue Data Catalog のスキーマ情報に統合します。
- スキーマとアクセス許可を管理 – データベースとテーブルへのアクセスを検証し、制御します。
- さまざまなデータソースに接続 – オンプレミスと AWS の両方で複数のデータソースを活用し、AWS Glue 接続を使用してデータレイクを構築します。

分析用データの変換、準備、クリーニング

- ジョブキャンバスインターフェイスでデータを視覚的に変換 – ビジュアルジョブエディターで ETL プロセスを定義し、データを抽出、変換、ロードするコードを自動的に生成します。
- シンプルなジョブスケジューリングで複雑な ETL パイプラインを構築 – スケジュール、オンデマンド、またはイベントに基づいて AWS Glue ジョブを呼び出します。
- 転送中のストリーミングデータのクリーニングと変換 – 継続的なデータ消費が可能になり、転送中のデータをクリーニングして変換します。これにより、ターゲットデータストアでの分析が数秒でできるようになります。

- 組み込みの機械学習によるデータの重複排除とクリーニング – `FindMatches` 機能を使用することで、機械学習の専門知識がなくても、分析用のデータをクリーニングして準備できます。この機能は、相互に不完全な一致であるレコードを重複排除して検索します。
- 組み込みのジョブノートブック – AWS Glue ジョブノートブックは、AWS Glue での最小限のセットアップでサーバーレスノートブックを提供するため、すぐに使用を開始できます。
- ETL コードの編集、デバッグ、テスト – AWS Glue インタラクティブセッションを使用することで、データをインタラクティブに探索して準備できます。任意の IDE またはノートブックを使用して、データをインタラクティブに探索、実験、処理できます。
- 機密データの定義、検出、修正 – AWS Glue の機密データ検出により、データパイプラインとデータレイク内の機密データを定義、識別、処理できます。

データパイプラインの構築とモニタリング

- ワークロードに基づいて自動的にスケーリング – ワークロードに基づいて、リソースを動的にスケールアップまたはスケールダウンできます。これにより、ワーカーは必要な場合にのみジョブに割り当てられます。
- イベントベースのトリガーでジョブを自動化 – イベントベースのトリガーでクローラーまたは AWS Glue ジョブを開始し、依存するジョブとクローラーのチェーンを設計します。
- ジョブの実行とモニタリング – 選択したエンジン (Spark または Ray) を使用して AWS Glue ジョブを実行します。また、自動モニタリングツール、AWS Glue ジョブ実行のインサイト、AWS CloudTrail を使用してモニタリングします。Apache Spark UI を使用して、Spark を利用したジョブのモニタリングを改善します。
- ETL と統合アクティビティのワークフローを定義 – ETLのワークフローと、複数のクローラー、ジョブ、トリガーの統合アクティビティを定義します。

AWS Glue のイノベーションについて学ぶ

AWS Glue の最新イノベーションについて学び、AWS Glue を使用して、セルフサービスによるデータの準備を組織全体で可能にする方法を確認できます。

AWS Glue を従来の設定にとらわれずにスケールする方法と、ジョブのモニタリングとパフォーマンスのために AWS Glue を設定する方法を確認できます。

AWS Glue の開始方法

以下のセクションから開始することが推奨されます。

- AWS Glue クローン作成の概要
- AWS Glue の概念
- AWS Glue 用の IAM アクセス許可のセットアップ
- AWS Glue Data Catalog の開始方法
- AWS Glue でジョブを作成する
- AWS Glue のインタラクティブセッションの開始方法
- AWS Glue でのオーケストレーション

AWS Glue へのアクセス

次のインターフェイスを使用して、AWS Glue ジョブの作成、表示、管理ができます。

- AWS Glue コンソール – AWS Glue ジョブを作成、表示、管理するためのウェブインターフェイスを提供します。コンソールにアクセスするには、AWS Glue を参照してください。
- AWS Glue Studio – AWS Glue ジョブを視覚的に作成および編集するためのグラフィカルインターフェイスを提供します。詳細については、「AWS Glue Studio でビジュアル ETL ジョブを作成する」を参照してください。
- AWS CLI リファレンスの AWS Glue セクション – AWS Glue で使用できる AWS CLI コマンドを提供します。詳細については、「AWS Glue 向けの AWS CLI リファレンス」を参照してください。
- AWS Glue API – デベロッパー向けの完全な API リファレンスを提供します。詳細については、「AWS Glue API」を参照してください。

関連サービス

AWS Glue のユーザーは、以下も使用します。

- AWS Lake Formation–AWS Glue Data Catalog内のリソースへの詳細に設定されたアクセスコントロールを行う認可レイヤーであるサービス。
- AWS Glue DataBrew – コードを記述せずに、データのクリーニングおよび正規化に使用できるビジュアルデータ準備ツール。

AWS Glue: 仕組み

AWS Glue は他の AWS のサービスを使用して ETL (抽出、変換、ロード) ジョブを調整し、データウェアハウスとデータレイクを構築して、出力ストリームを生成します。AWS Glue は API オペレーションを呼び出して、データの変換、ランタイムログの作成、ジョブロジックの保存を行い、ジョブ実行のモニタリングに役立つ通知を作成します。AWS Glue コンソールはこれらのサービスを管理アプリケーションに接続して、お客様が ETL ワークの作成とモニタリングに集中できるようにします。管理およびジョブ開発のオペレーションは、コンソールがお客様に代わって実行します。データソースへのアクセスとデータターゲットへの書き込みを行うために必要な、認証情報と他のプロパティは、お客様が AWS Glue に提供する必要があります。

AWS Glue は、ワークロードを実行するために必要なリソースのプロビジョニングおよび管理を行います。AWS Glue が代わって行うため、ETL ツールのインフラストラクチャを作成する必要はありません。リソースが必要な場合、起動時間を削減するために、AWS Glue はインスタンスのウォームプールからインスタンスを使用してワークロードを実行します。

AWS Glue では、データカタログにあるテーブル定義を使用してジョブを作成します。ジョブは、目的のデータ変換タスクを実行する手順を含むスクリプトで構成されます。トリガーを使用し、スケジュールに基づいて、または指定されたイベントの結果としてジョブを開始します。ターゲットデータが存在する場所、およびターゲットに入力するソースデータを指定します。入力に基づいて、AWS Glue はデータをソースからターゲット形式に変換します。または、AWS Glue コンソールまたは API でカスタムスクリプトを指定して、特定の要件に従ってデータを処理することもできます。

データソースと送信先

AWS Glue for Spark は、次のような複数のシステムやデータベースとの間でデータを読み書きできます。

- Amazon S3
- Amazon DynamoDB
- Amazon Redshift
- Amazon Relational Database Service (Amazon RDS)
- JDBC アクセスが可能なサードパーティのデータベース
- MongoDB と Amazon DocumentDB (MongoDB 互換)
- その他のマーケットプレイスコネクタと Apache Spark プラグイン

データストリーム

AWS Glue for Spark は、次のシステムからデータをストリーミングできます。

- Amazon Kinesis Data Streams
- Apache Kafka

AWS Glue は複数の AWS リージョンで利用可能です。詳細については、AWS の「Amazon Web Services 全般のリファレンス のリージョンとエンドポイント」を参照してください。

トピック

- 独立で実行されるサーバーレス ETL ジョブ
- AWS Glue の概念
- AWS Glue コンポーネント
- AWS Glue for Spark と AWS Glue for Ray
- AWS Glue を使用して半構造化されたスキーマをリレーショナルスキーマに変換する
- AWS Glue 型システム

独立で実行されるサーバーレス ETL ジョブ

AWS Glue は、Spark または Ray から選択したエンジンを使用して、サーバーレス環境で ETL ジョブを実行します。AWS Glue は、独自のサービスアカウントでプロビジョニングして管理する仮想リソースでこれらのジョブを実行します。

AWS Glue は、以下を実行するよう設計されています。

- お客様のデータを分離します。
- 伝送中と保管時のお客様のデータを保護します。
- 一時的な制限された認証情報を使用して、またはアカウント内の IAM ロールに対するお客様の同意を得て、お客様のリクエストに応え必要な時だけお客様のデータにアクセスします。

ETL ジョブのプロビジョニング時に、Virtual Private Cloud (VPC) にある入力データソースおよび出力データターゲットを提供します。また、データソースおよびターゲットにアクセスするために必要な、IAM ロール、VPC ID、サブネット ID、およびセキュリティグループを提供します。各タプル (顧客アカウント ID、IAM ロール、サブネット ID、およびセキュリティグループ) に、AWS Glue

は、AWS Glue サービスアカウント内に存在する他のすべての環境からネットワークおよび管理レベルで分離された新しい環境を作成します。

AWS アカウント内で、データカタログ、ジョブ、クローラーなどの AWS Glue リソースを作成して設定します。これらのリソースは、作成プロセス中に指定した IAM ロールとネットワーク設定 (サブネットとセキュリティグループ) に関連付けられます。

AWS Glue はプライベート IP アドレスを使用して、サブネットで Elastic Network Interface を作成します。ジョブはこれらの Elastic Network Interface を使用して、データソースおよびデータターゲットにアクセスします。ジョブ実行環境内外へのトラフィックと、ジョブ実行環境内でのトラフィックは、VPC およびネットワークポリシーにより管理されます。ただし、1 つ例外があり、AWS Glue ライブラリに対する呼び出しは、AWS Glue VPC 経由で AWS Glue API オペレーションにトラフィックをプロキシできます。すべての AWS Glue API 呼び出しはログに記録されます。そのため、データの所有者は、監査ログをアカウントに配信する AWS CloudTrail を有効にすることで API アクセスを監査できます。

ETL ジョブを実行する AWS Glue 管理の環境は、他の AWS のサービスと同じセキュリティ実施方法で保護されています。プラクティスと共有されたセキュリティ責任の概要については、「Introduction to AWS Security Processes」のホワイトペーパーを参照してください。

AWS Glue の概念

AWS Glue はフルマネージド型の ETL (抽出、変換、ロード) サービスであり、さまざまなデータソースとターゲット間でデータを簡単に移動できます。主なコンポーネントは次のとおりです。

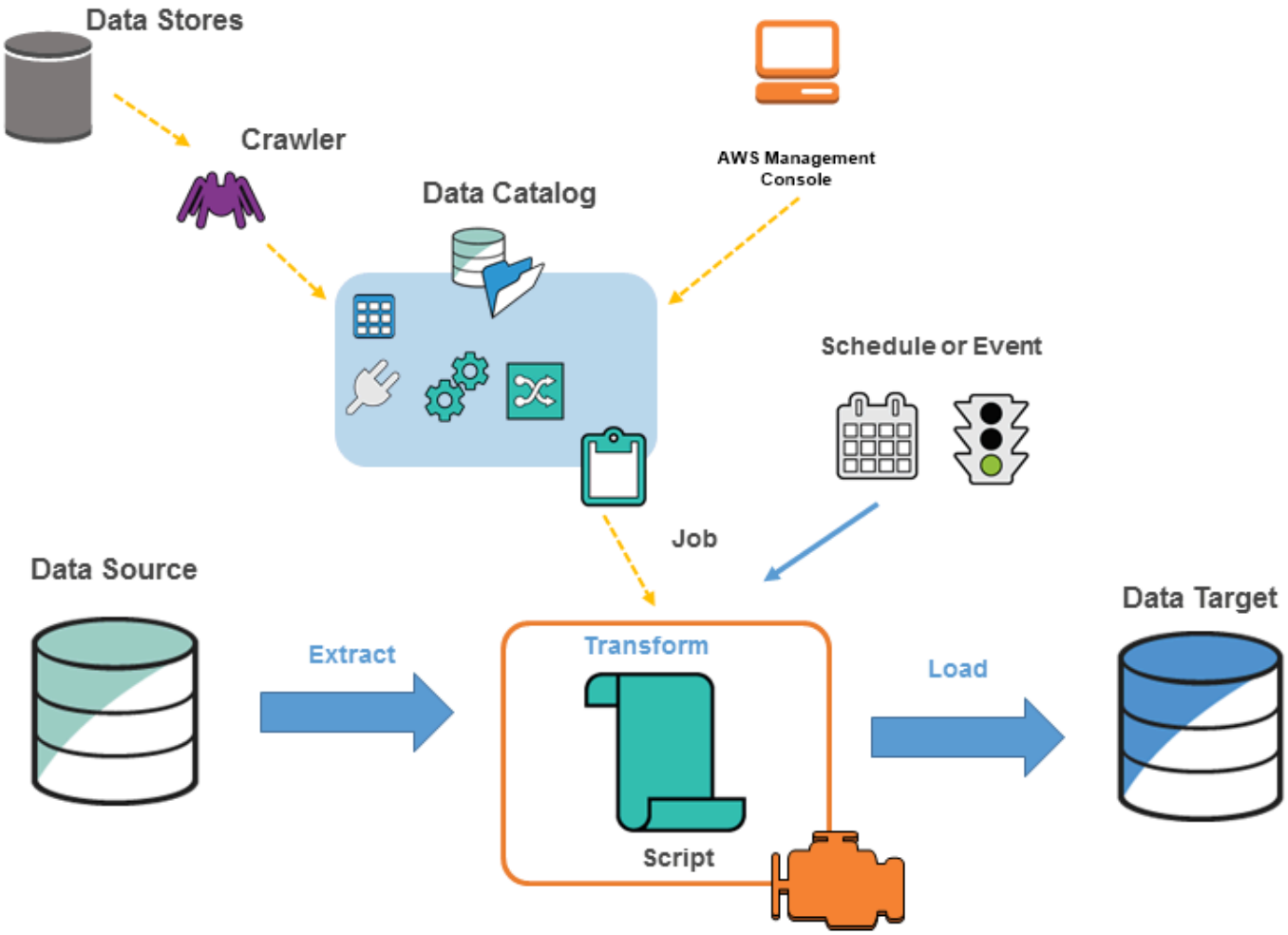
- データカタログ: ETL ワークフローのテーブル定義、ジョブ定義、その他の制御情報を含むメタデータストア。
- クローラー: データソースへの接続、データスキーマの推測、データカタログでメタデータテーブル定義の作成を行うプログラム。
- ETL ジョブ: ソースからデータの抽出、Apache Spark スクリプトを使用して変換、ターゲットにロードするビジネスロジック。
- トリガー: スケジュールまたはイベントに基づいてジョブ実行を開始するメカニズム。

一般的なワークフローに含まれる内容は次のとおりです。

1. データカタログでデータソースおよびターゲットを定義します。
2. クローラーを使用して、データソースのテーブルメタデータでデータカタログに入力します。

3. 変換スクリプトを使用して ETL ジョブを定義し、データを移動および処理します。
4. ジョブをオンデマンドまたはトリガーに基づいて実行します。
5. ダッシュボードを使用してジョブのパフォーマンスをモニタリングします。

次の図は、AWS Glue 環境のアーキテクチャを示します。



でジョブAWS Glueを指定し、データソースからデータターゲットへのデータの抽出、変換、ロード (ETL) に必要な作業を完了します。通常は、以下のアクションを実行します。

- データストアソースの場合は、クローラを指定し、AWS Glue Data Catalog にメタデータテーブルの定義を入力します。データストアでクローラを指定し、クローラは Data Catalog のテーブル定義を作成します。ストリーミングソースの場合は、Data Catalog テーブルを手動で定義し、データストリームのプロパティを指定します。

テーブル定義に加えて、AWS Glue Data Catalog には ETL ジョブを定義するために必要な他のメタデータが含まれています。このメタデータを使用して、データを変換するジョブを定義できます。

- AWS Glue はデータを変換するスクリプトを生成できます。または、AWS Glue コンソールまたは API でスクリプトを提供できます。
- ジョブをオンデマンドで実行する、または、指定したトリガーが発生すると開始するようにセットアップできます。トリガーは、時間ベースのスケジュールまたはイベントです。

ジョブが実行されると、スクリプトはデータソースからデータを抽出し、データを変換してデータターゲットにロードします。スクリプトは AWS Glue の Apache Spark 環境で実行されます。

> **Important**
>
> AWS Glue のテーブルとデータベースは、AWS Glue Data Catalog のオブジェクトです。それらにはメタデータが含まれ、データストアからのデータは含まれません。

AWS Glue で正常に処理するために、CSV などのテキストベースのデータは **UTF-8** でエンコードする必要があります。詳細については、Wikipedia の「UTF-8」を参照してください。

AWS Glue の用語

AWS Glue は、複数のコンポーネントの相互作用に依存して、抽出、変換、ロード (ETL) ワークフローを作成および管理します。

AWS Glue Data Catalog

AWS Glue の持続的なメタデータストア。これには、AWS Glue 環境を管理するためのテーブル定義、ジョブ定義、およびその他のコントロール情報が含まれています。各 AWS アカウントには、リージョンごとに 1 つの AWS Glue Data Catalog があります。

分類子

データのスキーマを決定します。AWS Glue は、一般的なファイルタイプの分類子を提供します (CSV、JSON、AVRO、XML など)。また、JDBC 接続を使用する一般的なリレーショナルデータ

ベース管理システムの分類子を提供します。独自の分類子を記述するには、 grok パターンを使用する、または、XML ドキュメント内の行タグを指定します。

Connection

特定のデータストアに接続するために必要なプロパティを含む Data Catalog オブジェクトです。

Crawler

データストア (ソースまたはターゲット) に接続し、分類子の優先順位リストを進行してデータのスキーマを判断し、AWS Glue Data Catalog にメタデータテーブルを作成するプログラムです。

データベース

論理グループに分類される、一連の関連付けられた Data Catalog テーブル定義です。

データストア、データソース、データターゲット

データストア は、データを永続的に保存するリポジトリです。例として、Amazon S3 バケット、リレーショナルデータベースなどがあります。データソースは、プロセスまたは変換への入力として使用されるデータストアです。データターゲットはプロセスまたは変換の書込み先であるデータストアです。

開発エンドポイント

エンドポイントは、AWS Glue ETL スクリプトの開発およびテストに使用できる環境です。

動的フレーム

構造や配列などのネストされたデータをサポートする分散テーブルです。各レコードは自己記述型であり、半構造化データのスキーマの柔軟性を持つよう設計されています。各レコードには、データとそのデータを記述するスキーマの両方が含まれます。動的フレームと Apache Spark DataFrames の両方を ETL スクリプトで使用し、それらの間で変換できます。動的フレームは、データクリーニングと ETL 用の一連の高度な変換を提供します。

ジョブ

ETL 作業を実行するために必要なビジネスロジックです。変換スクリプト、データソース、およびデーターゲットで構成されます。ジョブ実行は、スケジュールされたトリガーにより、または、イベントにトリガーされることで開始されます。

ジョブパフォーマンスダッシュボード

AWS Glue は、ETL ジョブ用の包括的な実行ダッシュボードを提供します。ダッシュボードには、特定の時間枠からのジョブ実行に関する情報が表示されます。

ノートブックインターフェイス。

ジョブ作成とデータ探索を容易にするワンクリック設定により、ノートブック体験を強化します。ノートブックと接続は自動的に構成されます。Jupyter Notebook をベースとしたノートブックインターフェースにより、AWS Glue のサーバーレス Apache Spark ETL インフラストラクチャを使用するスクリプトやワークフローを、対話的に開発、デバッグし、デプロイすることができます。また、ノートブック環境では、アドホッククエリ、データ分析、ビジュアライゼーション (表やグラフなど) を実行できます。

スクリプト

ソースからデータを抽出し、変換し、ターゲットにロードするコード。AWS Glue は PySpark または Scala スクリプトを生成します。

テーブル

データを表すメタデータ定義。データが、Amazon Simple Storage Service (Amazon S3) ファイル、Amazon Relational Database Service (Amazon RDS) テーブル、または別の一連のデータのどれにある場合でも、テーブルはデータのスキーマを定義します。AWS Glue Data Catalog のテーブルは、列名、データ型の定義、パーティション情報、および基本データセットに関するその他のメタデータで構成されています。データのスキーマは AWS Glue のテーブル定義で表されます。実際のデータは、ファイルまたはリレーショナルデータベーステーブルにあっても、元のデータストアに残ります。AWS Glue はファイルとリレーショナルデータベースのテーブルを AWS Glue Data Catalog に格納します。それらは、ETL ジョブを作成する際にソースおよびターゲットとして使用されます。

変換

データを操作して別の形式にするために使用するコードのロジック。

Trigger トリガー)

ETL ジョブを開始します。トリガーはスケジュールされた時間またはイベントに基いて定義できます。

ビジュアルジョブエディタ

ビジュアルジョブエディターは、AWS Glue での抽出、変換、ロード (ETL) ジョブの作成、実行、およびモニタリングが簡単に行えるグラフィカルなインターフェイスです。データ変換ワークフローを視覚的に構成し、AWS Glue の Apache Spark ベースのサーバーレス ETL エンジンでそれらをシームレスに実行して、ジョブの各ステップでスキーマとデータの結果を検査できます。

ワーカー

AWS Glue では、ETL ジョブの実行にかかる時間に対してのみお支払いが発生します。管理するリソースはなく、前払い料金もありません。また、起動時間やシャットダウン時間に対しての課金もありません。ETL ジョブの実行に使用されたデータ処理ユニット (DPUs) の数に基づいて時間あたりの料金が請求されます。単一のデータ処理ユニット (DPU) は、ワーカーとも呼びます。AWS Glue には、ジョブのレイテンシーとコスト要件を満たす構成を選択するのに役立つような 3 つのワーカータイプが用意されています。ワーカーには、Standard、G.1X、G.2X、および G.025X 構成があります。

AWS Glue コンポーネント

AWS Glue は、抽出、変換、ロード (ETL) ワークロードを設定し管理するためのコンソールと API オペレーションを備えています。いくつかの言語に固有な SDK と AWS Command Line Interface (AWS CLI) を介して API オペレーションを使用できます。AWS CLI の使用については、「AWS CLI コマンドリファレンス」を参照してください。

AWS Glue は AWS Glue Data Catalog を使用して、データソース、変換、およびターゲットについてのメタデータを保存します。Data Catalog は Apache Hive メタストアのドロップインリプレースメントです。AWS Glue Jobs system は、データの ETL オペレーションの定義、スケジューリング、および実行のためのマネージド型インフラストラクチャを備えています。AWS Glue API の詳細については、「AWS Glue API」を参照してください。

AWS Glue コンソール

AWS Glue コンソールを使用して、ETL ワークフローを定義しオーケストレーションします。コンソールは AWS Glue Data Catalog および AWS Glue Jobs system のいくつかの API オペレーションを呼び出して、次のタスクを実行します。

- ジョブ、テーブル、クローラ、接続などの AWS Glue オブジェクトを定義します。
- いつクローラが実行するかをスケジュールします。

- ジョブトリガーのイベントやスケジュールを定義します。
- AWS Glue オブジェクトのリストを検索しフィルタリングします。
- 変換スクリプトを編集します。

AWS Glue Data Catalog

AWS Glue Data Catalog は AWS クラウド内にある永続的な技術メタデータストアです。

各 AWS アカウントには、AWS リージョンごとに 1 つの AWS Glue Data Catalog があります。各データカタログは、データベースに構成された高度にスケーラブルなテーブルのコレクションです。テーブルは、Amazon RDS、Apache Hadoop 分散ファイルシステム、Amazon OpenSearch Service などのソースに保存されている構造化データまたは半構造化データの集合をメタデータで表現したものです。AWS Glue Data Catalog は統一されたリポジトリを提供するため、異種システムはデータサイロのデータを追跡するためにメタデータを保存し検索することができます。その後は、メタデータを使用して、多種多様なアプリケーション全体で一貫したやり方でそのデータをクエリしたり変換したりすることができます。

データカタログを AWS Identity and Access Management ポリシーと Lake Formation と併用することで、テーブルやデータベースへのアクセスを制御します。この方法により、企業内の様々なグループが、機密情報を高度かつ詳細に保護しながら、より広範な組織に対してデータを安全に公開できるようになります。

また、データカタログを CloudTrail や Lake Formation を併用することで、スキーマ変更の追跡やデータアクセス制御を備えた包括的な監査およびガバナンス機能も提供します。これにより、データの不適切な変更や誤った共有を防止できます。

AWS Glue Data Catalog のセキュリティ保護と監査の詳細については、以下を参照してください。

- AWS Lake Formation – 詳細については、AWS Lake Formation デベロッパーガイドの「AWS Lake Formation とは？」を参照してください。
- CloudTrail – 詳細については、「AWS CloudTrail ユーザーガイド」の「CloudTrail とは」を参照してください。

次に示すのは、AWS Glue Data Catalog を使用する AWS の他のサービスとオープンソースプロジェクトです。

- Amazon Athena – 詳細については、Amazon Athena ユーザーガイドの「テーブル、データベース、およびデータカタログの理解」を参照してください。

- Amazon Redshift Spectrum – 詳細については、Amazon Redshift データベースデベロッパーガイドの「Amazon Redshift Spectrum を使用した外部データのクエリ」を参照してください。
- Amazon EMR – 詳細については、Amazon EMR マネジメントガイドの「AWS Glue Data Catalog への Amazon EMR アクセスにリソースベースのポリシーを使用する」を参照してください。
- Apache Hive メタストア用 AWS Glue Data Catalog クライアント – この GitHub プロジェクトの詳細については、「AWS Glue Data Catalog Client for Apache Hive Metastore」を参照してください。

AWS Glue クローラおよび分類子

AWS Glue では、あらゆる種類のリポジトリにあるデータのスキャン、分類、スキーマ情報の抽出、そのメタデータの AWS Glue Data Catalog への自動保存ができるクローラを設定することもできます。そこから AWS Glue Data Catalog は ETL オペレーションをガイドするのに使用できます。

クローラおよび分類子の設定方法については、「クローラーを使用したデータカタログへの入力 」を参照してください。AWS Glue API を使用してクローラおよび分類子をプログラムする方法については、「クローラーおよび分類子 API」を参照してください。

AWS Glue ETL オペレーション

AWS Glue は、Data Catalog のメタデータを使用して、さまざまな ETL オペレーションを実行するために使用や変更ができる AWS Glue 拡張機能を備えた、Scala または PySpark (Apache Spark 用の Python API) スクリプトを自動生成できます。たとえば、未加工データを抽出、クリーンアップ、および変換してからその結果を別のリポジトリに保存して、クエリと分析を行うことができます。このようなスクリプトは、CSV ファイルをリレーショナル形式に変換し、Amazon Redshift に保存する場合があります。

AWS Glue ETL 機能の使用方法の詳細については、「Spark スクリプトのプログラミング」を参照してください。

AWS Glue でのストリーミング ETL

AWS Glue を使用すると、連続実行のジョブを使用して、ストリーミングデータに対して ETL 操作を実行できます。AWS Glue のストリーミング ETL は、Apache Spark 構造化ストリーミングエンジン上に構築され、Amazon Kinesis Data Streams、Apache Kafka、および Amazon Managed Streaming for Apache Kafka (Amazon MSK) からストリームを取り込むことができます。ストリー

ミング ETL では、ストリーミングデータのクリーニングと変換を行い、Simple Storage Service (Amazon S3) または JDBC データストアにロードできます。AWS Glue のストリーミング ETL を使用すると、IoT ストリーム、クリックストリーム、ネットワークログなどのイベントデータを処理できます。

ストリーミングデータソースのスキーマがわかっている場合は、Data Catalog テーブルで指定できます。そうでない場合は、ストリーミング ETL ジョブでスキーマ検出を有効にできます。ジョブは、受信データからスキーマを自動的に決定します。

ストリーミング ETL ジョブは、AWS Glue 組み込みの変換および Apache Spark 構造化ストリーミングネイティブの変換の両方を使用できます。詳細については、「Operations on streaming DataFrames/Datasets」 を参照してください。

詳しくは、「the section called “ストリーミング ETLジョブ”」を参照してください。

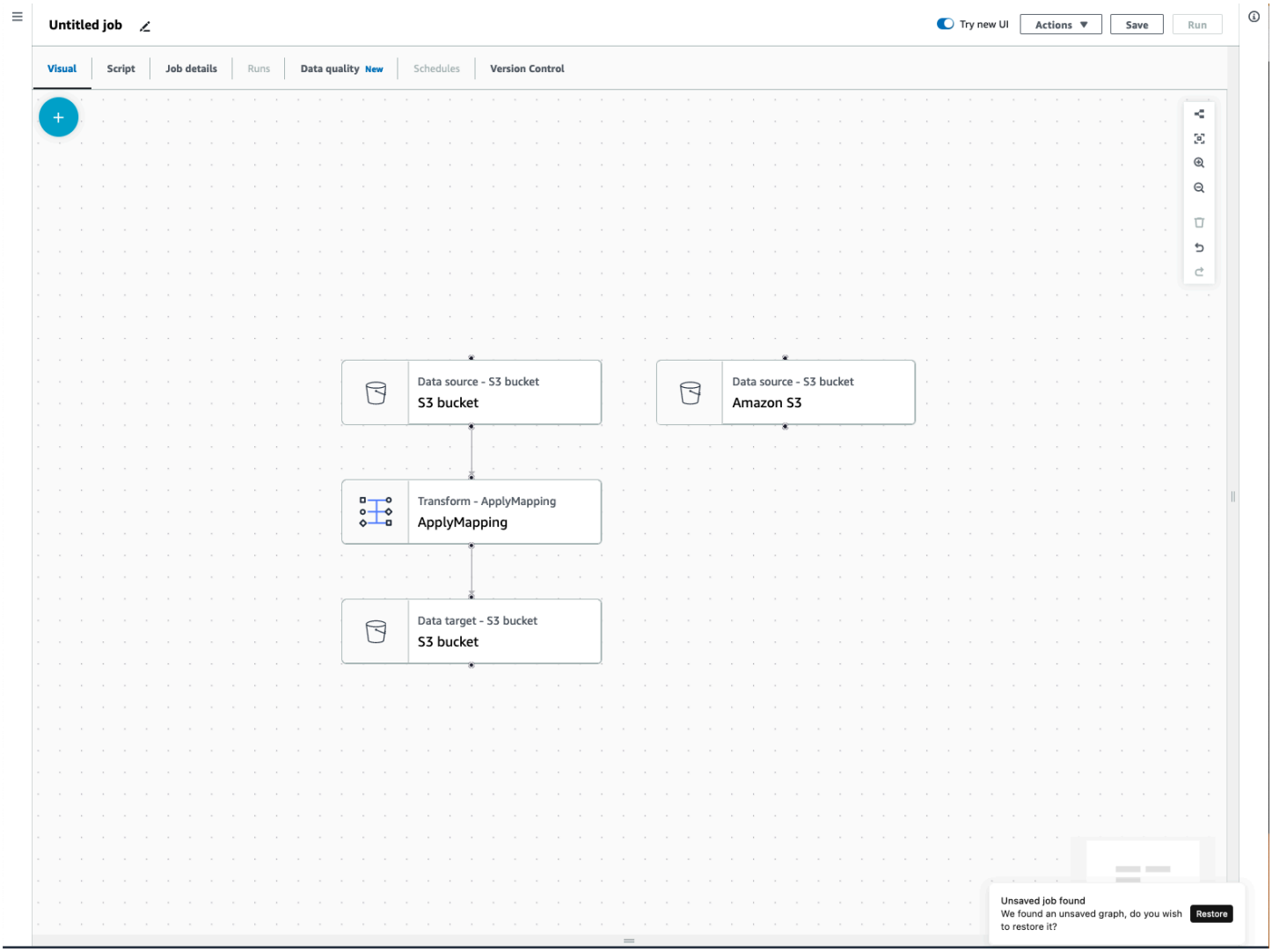
AWS Glue ジョブシステム

AWS Glue Jobs system は、ETL ワークフローをオーケストレーションするためのマネージド型インフラストラクチャを提供します。データを抽出したり変換したり異なる場所へ転送したりするのに使用するスクリプトを自動化するジョブを AWS Glue で作成できます。ジョブはスケジュールしたり連鎖させることができます。または新しいデータの到着などのイベントによってトリガーすることができます。

AWS Glue Jobs system の使用方法の詳細については、「AWS Glue のモニタリング」を参照してください。AWS Glue Jobs system API を使用したプログラミングについては、「ジョブ API」を参照してください。

ビジュアル ETL コンポーネント

AWS Glue では、操作可能なビジュアルキャンバスから ETL ジョブを作成できます。



ETL ジョブメニュー

キャンバスの上部にあるメニューオプションから、ジョブに関するさまざまなビューや設定の詳細にアクセスできます。

- [Visual] - ビジュアルジョブエディタのキャンバスです。ここでは、ノードを追加してジョブを作成できます。
- スクリプト - ETL ジョブのスクリプト表現。AWSGlue は、ジョブの視覚的表現に基づいてスクリプトを生成します。スクリプトを編集したり、ダウンロードしたりもできます。

> **Note**
>
> スクリプトを編集することを選択した場合、ジョブ作成体験は完全にスクリプト専用モードに変更されます。それ以後、ビジュアルエディタを使用してジョブを編集することは

できません。そのため、スクリプトを編集することを選択する前に、すべてのジョブソース、変換、ターゲットを追加し、ビジュアルエディタで必要な変更をすべて行ってください。

- [ジョブの詳細] - [ジョブの詳細] タブでは、ジョブのプロパティを設定することでジョブを設定できます。基本的なプロパティとして、ジョブの名前と説明、IAM ロール、ジョブタイプ、AWS Glue バージョン、言語、ワーカータイプ、ワーカー数、ジョブブックマーク、Flex 実行、廃止数、ジョブタイムアウトなどがあります。高度なプロパティとして、接続、ライブラリ、ジョブパラメータ、タグなどがあります。
- [Runs] - ジョブの実行後に、このタブにアクセスして過去のジョブを表示できます。
- [Data quality] - [Data quality] は、データアセットの品質を評価およびモニタリングします。このタブでは、データ品質の活用方法を詳しく確認したり、データ品質変換をジョブに追加したりできます。
- [Schedules] - このタブには、スケジュールされたジョブが表示されます。このジョブにスケジュールがアタッチされていない場合、このタブにはアクセスできません。
- [Version control] - ジョブを Git リポジトリに設定することで、ジョブで Git を使用できます。

ビジュアル ETL パネル

キャンバスで作業するときに、ノードの設定、データのプレビュー、出力スキーマの表示などに役立つパネルがいくつか用意されています。

- [プロパティ] - キャンバス上のノードを選択すると、[プロパティ] パネルが表示されます。
- [Data preview] - [Data preview] パネルには、データ出力のプレビューが表示されるため、ジョブを実行して出力を確認する前に判断を行えます。
- [Output schema] - [Output schema] タブでは、変換ノードのスキーマを表示および編集できます。

パネルサイズを変更する

画面の右側にある [プロパティ] パネルと、[Data preview] タブや [Output schema] タブを含む下部パネルのサイズを変更するには、パネルの端をクリックして左右または上下にドラッグします。

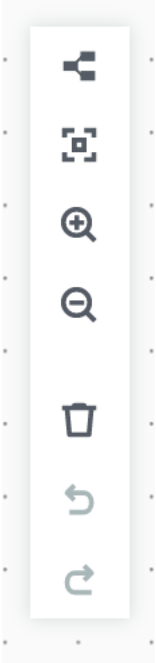
- [プロパティ] パネル - [プロパティ] パネルのサイズを変更するには、画面の右側にあるキャンバスの端をクリックしてドラッグし、左にドラッグして幅を拡大します。デフォルトでは、パネルは折りたたまれており、ノードを選択すると、[プロパティ] パネルがデフォルトサイズで表示されます。

- [Data preview] と [Output schema] パネル - 下部パネルのサイズを変更するには、画面の下部にあるキャンバスの下端をクリックしてドラッグし、上にドラッグして高さを拡大します。デフォルトでは、パネルは折りたたまれており、ノードを選択すると、下部パネルがデフォルトサイズで表示されます。

ジョブキャンバス

ビジュアル ETL キャンバスでは、ノードの追加、削除、移動/順序変更を直接行えます。これは、データソースで始まりデーターゲットで終わる、完全に機能する ETL ジョブを作成するためのワークスペースと考えてください。

キャンバス上のノードを操作するときに、ツールバーを使用できます。ツールバーでは、ズームインとズームアウト、ノードの削除、ノード間の接続の作成または編集、ジョブフローの向きの変更、アクションを元に戻す/やり直すなどが行えます。

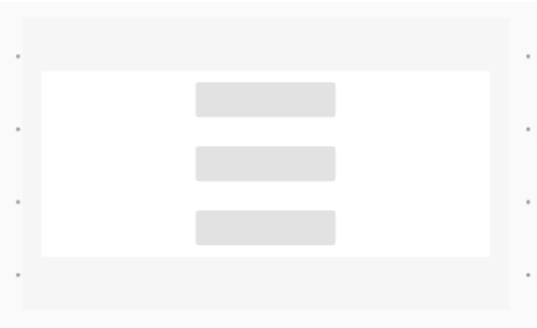


フローティングツールバーはキャンバス画面の右上に固定されており、アクションを実行する複数のアイコン画像が表示されています。

- レイアウトアイコン - レイアウトアイコンは、ツールバーの一番上のアイコンです。デフォルトでは、ビジュアルジョブの方向は上から下です。ノードを左から右に水平に配置することで、ビジュアルジョブの配置方向を変更します。レイアウトアイコンを再度クリックすると、方向は上から下に戻ります。

- リセンタリングアイコン - リセンタリングアイコンは、キャンバスの表示位置を中央に変更します。大規模なジョブで、中央位置に戻るときに活用できます。
- ズームインアイコン - ズームインアイコンは、キャンバス上のノードのサイズを拡大します。
- ズームアウトアイコン - ズームアウトアイコンは、キャンバス上のノードのサイズを縮小します。
- ゴミ箱アイコン - ゴミ箱アイコンは、ビジュアルジョブからノードを削除します。先にノードを選択しておく必要があります。
- 元に戻すアイコン - 元に戻すアイコンは、ビジュアルジョブで最後に実行したアクションを取り消します。
- やり直すアイコン - やり直すアイコンは、ビジュアルジョブで最後に実行されたアクションをやり直します。

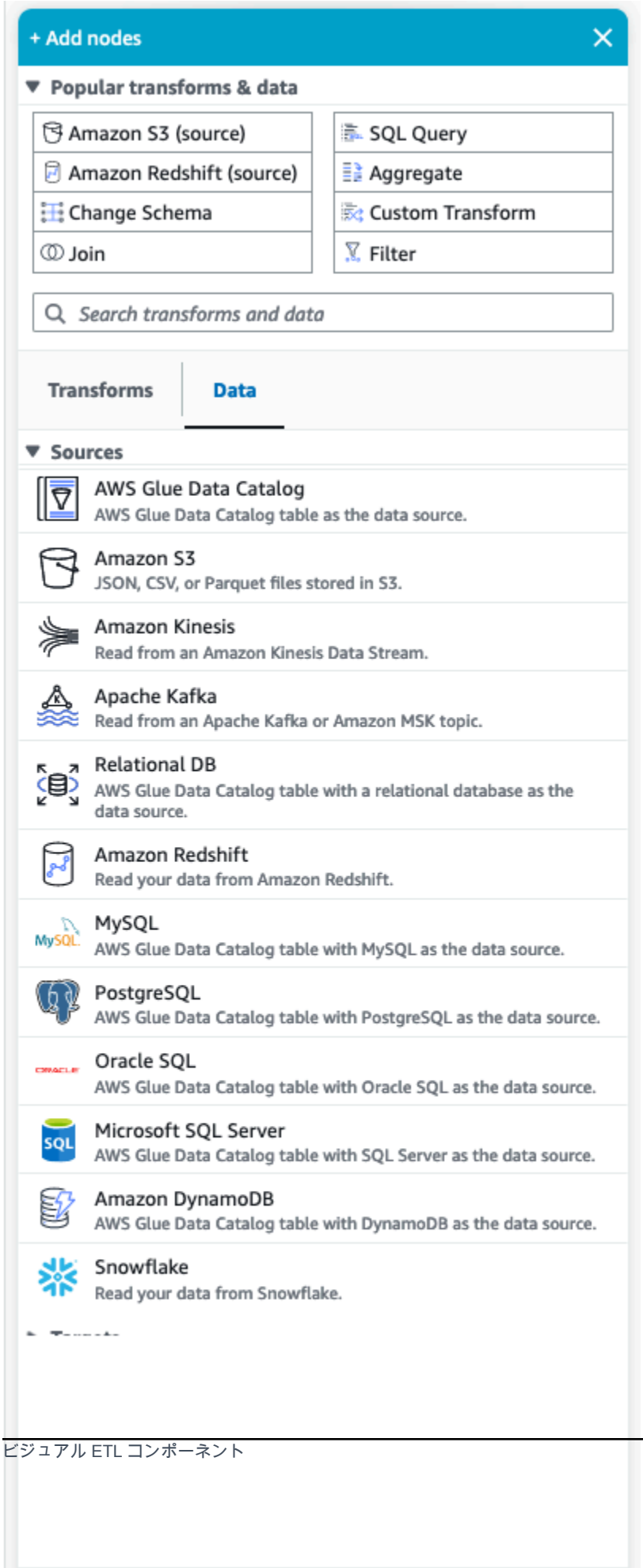
ミニマップを使用する



リソースパネル

リソースパネルには、使用可能なデータソース、変換アクション、接続がすべて含まれています。キャンバスでリソースパネルを表示するには、[+] アイコンをクリックします。これで、リソースパネルが開きます。

リソースパネルを閉じるには、リソースパネルの右上隅にある [X] をクリックします。これにより、再び開く準備ができるまでパネルが非表示になります。


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

[Popular transforms & data]

パネルの上部に、[Popular transforms & data] というコレクションがあります。これらのノードは、AWS Glue で一般的に使用されるものです。いずれかを選択してキャンバスに追加します。また、[Popular transforms & data] の見出しの横にある三角アイコンをクリックすると、[Popular transforms & data] を非表示にできます。

[Popular transforms & data] セクションの下で、変換およびデータソースノードを検索できます。入力すると結果が表示されます。検索クエリに追加する文字が多いほど、表示される結果は少なくなります。検索結果には、ノードの名前または説明が表示されます。結果に表示されたノードを選択してキャンバスに追加します。

[Transforms] と [Data]

ノードを [Transforms] と [Data] に整理する 2 つのタブがあります。

[Transforms] – [Transforms] タブを選択すると、使用可能なすべての変換を選択できます。変換を選択してキャンバスに追加します。また、[Transforms] リストの下部にある [変換の追加] をクリックすると、カスタムビジュアル変換を作成するためのドキュメントへの新しいページが開きます。手順に従うことで、独自の変換を作成できます。この変換はその後、使用可能な変換のリストに表示されます。

[Data] – [Data] タブには、[ソース] と [ターゲット] のノードがすべて含まれています。[ソース] と [ターゲット] を非表示にするには、[ソース] または [ターゲット] の見出しの横にある三角アイコンをクリックします。三角アイコンを再度クリックすると、[ソース] と [ターゲット] を再表示できます。ソースノードまたはターゲットノードを選択し、キャンバスに追加します。また、[Manage Connections] をクリックすると、新しい接続を追加できます。このとき、コンソールのコネクタページが開きます。

AWS Glue for Spark と AWS Glue for Ray

AWS Glue on Apache Spark (AWS Glue ETL) では、PySpark を使用して、大規模なデータを処理する Python コードを書くことができます。Spark はこの問題に対する一般的な解決策ですが、Python 専門の経歴を持つデータエンジニアは、直感的な移行ではないと感じるかもしれません。Spark DataFrame モデルはシームレスに「Python らしい」というわけではなく、Scala 言語と、それが構築されている Java ランタイムを反映しています。

AWS Glue では、Python シェルジョブを使用してネイティブな Python データ統合を実行できます。これらのジョブは、単一の Amazon EC2 インスタンス上で実行され、そのインスタンスの容量に

よって制限されます。これにより、処理できるデータのスループットが制限され、ビッグデータを処理する場合は維持費が高くなります。

AWS Glue for Ray では、Spark の学習に多額の投資をすることなく Python ワークロードをスケールアップできます。Ray のパフォーマンスが向上する特定のシナリオを使用できます。選択肢が提供されることで、Spark と Ray の両方の強みを活用できます。

AWS Glue ETL と AWS Glue for Ray は根本的に異なるため、サポートしている機能も異なります。サポートされている機能については、ドキュメントを確認してください。

AWS Glue for Ray とは

Ray は Python を中心に、ワークロードのスケールアップに使用できるオープンソースの分散計算フレームワークです。Ray の詳細については、Ray のウェブサイトを参照してください。 AWS GlueRay ジョブおよびインタラクティブセッションにより、AWS Glue 内で Ray を使用できます。

AWS Glue for Ray を使用して、複数のマシンで並列に実行される計算用の Python スクリプトを作成できます。Ray ジョブおよびインタラクティブセッションでは、pandas などの使い慣れた Python ライブラリを使用して、ワークフローを簡単に記述して実行できます。Ray データセットの詳細については、Ray ドキュメントの「Ray データセット」を参照してください。pandas の詳細については、Pandas のウェブサイトを参照してください。

AWS Glue for Ray を使用すると、わずか数行のコードだけで、企業規模のビッグデータに対する pandas のワークフローを実行できます。Ray ジョブは、AWS Glue コンソールまたは AWS SDK から作成できます。AWS Glue インタラクティブセッションを開いて、サーバーレスの Ray 環境でコードを実行することもできます。AWS Glue Studio のビジュアルジョブはまだサポートされていません。

AWS Glue for Ray ジョブを使用すると、スケジュールに従って、または Amazon EventBridge からのイベントに応じてスクリプトを実行できます。ジョブはログ情報とモニタリング統計を CloudWatch に保存するため、スクリプトの正常性と信頼性を確認できます。AWS Glue ジョブのシステムについての詳細は、「the section called “Ray ジョブの使用”」を参照してください。

Ray は、負荷に基づいてリアルタイムで再設定を行うマシンのクラスターに処理を分散することで、Python コードのスケーリング作業を自動化します。これにより、特定のワークロードで 1 ドルあたりのパフォーマンスが向上します。Ray ジョブでは、AWS Glue のジョブモデルに自動スケーリングがネイティブに組み込まれているため、この機能を最大限に活用できます。Ray ジョブは AWS Graviton 上で実行されるため、全体的な価格パフォーマンスが向上します。

コストの削減に加えて、ネイティブの自動スケーリングを使用すると、クラスターのメンテナンス、調整、管理に時間を費やすことなく Ray のワークロードを実行できます。pandas や AWS SDK for Pandas など、使い慣れたオープンソースライブラリをそのまま使用できます。これにより、AWS Glue for Ray で開発する際のイテレーション速度が向上します。AWS Glue for Ray を使用すると、費用対効果の高いデータ統合ワークロードをすばやく開発して実行できるようになります。

AWS Glue を使用して半構造化されたスキーマをリレーショナルスキーマに変換する

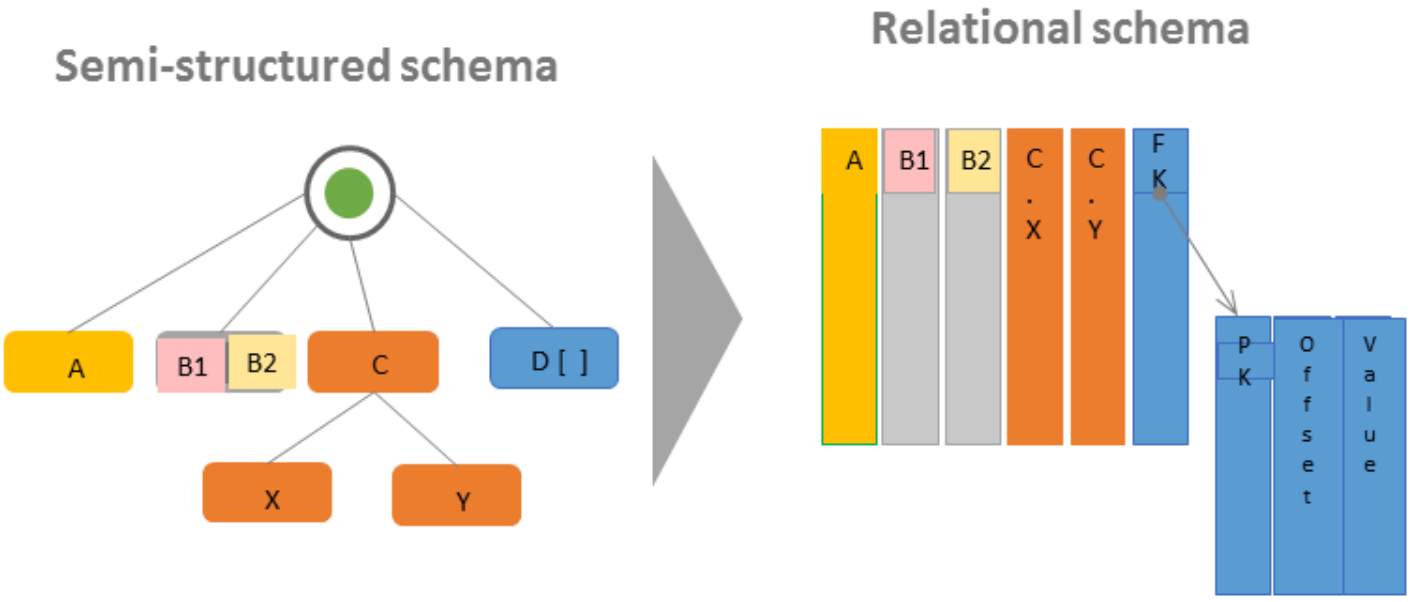
半構造化データをリレーショナルテーブルに変換することが一般的です。概念的には、階層的なスキーマをリレーショナルスキーマに平坦化します。AWS Glue は、この変換を臨機応変に実行できます。

通常、半構造化されたデータには、データ内のエンティティを識別するためのマークアップが含まれています。固定されたスキーマのない、ネスト化されたデータ構造を持つことができます。半構造化データの詳細については、Wikipedia の 半構造化データ を参照してください。

リレーショナルデータは、行と列で構成されるテーブルで表されます。テーブル間の関係は、プライマリキー (PK) と外部キー (FK) の関係によって表すことができます。詳細については、Wikipedia の リレーショナルデータベース を参照してください。

AWS Glue は、クローラを使用して半構造化データのスキーマを推測します。次に、ETL (抽出、変換、ロード) ジョブを使用してデータをリレーショナルスキーマに変換します。例えば、JSON データを解析して、Amazon Simple Storage Service (Amazon S3) ソースファイルから Amazon Relational Database Service (Amazon RDS) テーブルに変換したい場合があります。AWS Glue がスキーマの違いを処理する方法を理解すると、変換プロセスを理解するうえで役立ちます。

この図は、AWS Glue が半構造化スキーマをリレーショナルスキーマに変換する方法を示しています。



この図表は、以下を示すものです：

- 単一の値 A は、直接リレーショナル列に変換されます。
- 値のペアである B1 および B2 は、2 つのリレーショナル列に変換されます。
- 子の X と Y を持つ C 構造は、2 つのリレーショナル列に変換されます。
- 配列 D[] は、別のリレーショナルテーブルを指す外部キー (FK) 列のリレーショナル列に変換されます。2 番目のリレーショナルテーブルには、プライマリキー (PK) に加えて、オフセットと配列の項目の値を含む列があります。

AWS Glue 型システム

AWS Glue は、複数の型のシステムを使用し、さまざまな方法でデータを格納するデータシステムに対して汎用性の高いインターフェイスを提供します。このドキュメントでは、AWS Glue 型システムとデータ標準を明確にします。

AWS Glue データカタログ型

データカタログは、さまざまなデータシステムに格納されているテーブルとフィールドのレジストリであり、メタストアです。AWS Glue クローラーや AWS Glue with Spark ジョブなどの AWS Glue コンポーネントをデータカタログに書き込むときは、フィールドの型を追跡するための内部型システムを使用して書き込みを行います。これらの値は、AWS Glue コンソールのテーブルスキーマのデータ型列に表示されます。このタイプのシステムは、Apache Hive の型のシステムに基づいています。Apache Hive の型のシステムの詳細については、Apache Hive wiki の「型」を参照してください。特定の型とサポートの詳細については、スキーマビルダーの一部として AWS Glue コンソールに例が用意されています。

検証、互換性、その他の用途

データカタログは、型フィールドに書き込まれた型を検証しません。AWS Glue コンポーネントがデータカタログを読み書きする場合、相互に互換性を保つことができます。AWSGlue コンポーネントは、Hive 型との高い互換性を保つことも目的としています。ただし、AWS Glue コンポーネントはすべての Hive 型との互換性を保証するものではありません。これにより、データカタログ内のテーブルを操作するときに Athena DDL などのツールとの相互運用が可能になります。

データカタログは型を検証しないため、他のサービスは、Hive の型のシステムに厳密に準拠するシステムやその他のシステムを使用することで、型を追跡するためにデータカタログを利用できます。

Spark スクリプトを使用した AWS Glue の型

Spark スクリプトを使用した AWS Glue がデータセットを解釈または変換する場合、スクリプトで使用されているデータセットのインメモリ表現である `DynamicFrame` が提供されます。`DynamicFrame` の目的は、Spark `DataFrame` と似ています。つまり、Spark がデータに対して変換をスケジュールして実行できるようにデータセットをモデル化します。`toDF` および `fromDF` メソッドを提供することにより、`DynamicFrame` の型表現が `DataFrame` と相互互換であることを保証します。

型情報を `DataFrame` に推測または提供できる場合は、特に文書化されていない限り、`DynamicFrame` に推測または提供できます。特定のデータ形式向けに最適化されたリーダーま

たはライターを提供するとき、Spark がお客様のデータを読み書きできる場合は、文書に記載されている制限に従って、提供されているリーダーおよびライターも読み取りまたは書き込みを行うことができます。リーダーおよびライターの詳細については、「the section called “データ形式に関するオプション”」を参照してください。

Choice 型

`DynamicFrames` は、ディスク上の行ごとの値が一貫性のない型を持つ可能性があるデータセットのフィールドをモデル化するメカニズムを提供します。例えば、フィールドで特定の行に文字列として格納された数値が、他の行では整数として格納されている場合があります。このメカニズムは `Choice` と呼ばれるインメモリ型です。`ResolveChoice` メソッドのような、Choice の列を具体的な型に解決するための変換を行うことができます。AWSGlue ETL では、通常の操作では Choice 型をデータカタログに書き込むことはありません。Choice 型は、データセットの DynamicFrame メモリモデルのコンテキストでのみ存在します。Choice 型の使用例については、「the section called “データ準備サンプル”」を参照してください。

AWS Glue クローラーの型

クローラーは、データセット用に一貫性のある使いやすいスキーマを作成し、それをデータカタログに格納して他の AWS Glue コンポーネントや Athena で使用できるようにすることを目的としています。クローラーは、データカタログの前のセクション「the section called “AWS Glue データカタログ型”」で説明したように、型を扱います。列に 2 つ以上の型の値が含まれる「Choice」型のシナリオで使用可能な型を作成するために、クローラーは潜在的な型をモデル化した `struct` 型を作成します。

AWS Glue の開始方法

以下のセクションでは、AWS Glue の設定について説明します。AWS Glue の使用を開始するにあたり、すべての設定事項が必要になるわけではありません。IAM アクセス許可、暗号化、VPC を使用する場合の DNS、データストアにアクセスするための環境、インタラクティブセッションを使用する場合など、必要に応じて指示に従って設定することができます。

トピック

- AWS Glue クローン作成の概要
- AWS Glue 用の IAM アクセス許可のセットアップ
- AWS Glue 使用状況プロファイルの設定
- AWS Glue Data Catalog の開始方法
- データストアへのネットワークアクセスを設定する
- AWS Glue での暗号化のセットアップ
- AWS Glue のための開発用ネットワークの設定

AWS Glue クローン作成の概要

AWS Glue では、AWS Glue Data Catalog にメタデータを格納します。このメタデータを使用して、データソースを変換してデータウェアハウスやデータレイクをロードする ETL ジョブを調整します。以下の手順では、一般的なワークフローと、AWS Glue を使用して作業する際に行う選択肢のいくつかについて説明します。

> **Note**
>
> 以下の手順に従うことも、手順 1 ~ 3 を自動的に実行するワークフローを作成することもできます。詳しくは、「the section called “設計図とワークフローを使用した複雑な ETL アクティビティの実行”」を参照してください。

1. AWS Glue Data Catalog にテーブル定義を入力します。

   永続データストアの場合は、コンソールでクローラを追加して AWS Glue Data Catalog にデータを追加できます。[Add crawler (クローラを追加)] ウィザードは、テーブルのリストまたはクローラのリストから開始できます。クローラがアクセスするための 1 つ以上のデータストアを選択

します。スケジュールを作成して、クローラの実行頻度を決定することもできます。データストリームの場合は、テーブル定義を手動で作成して、ストリームプロパティを定義できます。

必要に応じて、データのスキーマを推測するカスタム分類子を提供できます。 grok パターンを使用してカスタム分類子を作成できます。ただし、AWS Glue には、カスタム分類子がデータを認識しない場合にクローラによって自動的に使用される組み込み分類子が用意されています。クローラを定義する時に、分類子を選択する必要はありません。AWS Glue の分類子の詳細については、「分類子の定義と管理」を参照してください。

一部のタイプのデータストアをクロールするには、認証とロケーション情報を提供する接続が必要です。必要に応じて、AWS Glue コンソールでこの必要な情報を提供する接続を作成できます。

クローラはデータストアを読み取り、データ定義と名前付きテーブルを AWS Glue Data Catalog に作成します。これらのテーブルは、選択したデータベースに整理されます。また、手動で作成したテーブルを Data Catalog に入力することもできます。この方法では、スキーマおよびその他のメタデータを提供して、Data Catalog にテーブル定義を作成します。この方法は少し面倒で エラーが発生しやすいため、より良い方法として、クローラにテーブル定義を作成させることができます。

AWS Glue Data Catalog にテーブル定義を入力する方法の詳細については、「テーブルの作成」を参照してください。

2. ソースからターゲットへのデータの変換を記述するジョブを定義します。

一般に、ジョブを作成するには、次の選択をする必要があります。

- ジョブのソースとなるテーブルを AWS Glue Data Catalog から選択します。ジョブでは、このテーブル定義を使ってデータソースにアクセスし、データの型式を解釈します。
- ジョブのターゲットとなるテーブルまたは場所を AWS Glue Data Catalog から選択します。ジョブはこの情報を使用して、データストアにアクセスします。
- ソースをターゲットに変換するスクリプトを生成するように AWS Glue に指示します。AWS Glue は、ソーススキーマからターゲットスキーマ形式にデータを変換する組み込み変換を呼び出すコードを生成します。これらの変換は、データのコピー、列の名前の変更、データのフィルタリングなどの操作を実行し、必要に応じてデータを変換します。このスクリプトは、AWS Glue コンソールで変更できます。

AWS Glue でジョブを定義する方法の詳細については、「AWS Glue Studio でビジュアル ETL ジョブを作成する」を参照してください。

3. ジョブを実行してデータを変換します。

   オンデマンドでジョブを実行するか、次のいずれかのトリガータイプに基づいてジョブを開始することができます。

   - cron スケジュールに基づいたトリガー。
   - イベントベースのトリガー。たとえば、別のジョブが正常に完了すると、AWS Glue ジョブを開始できます。
   - オンデマンドでジョブを開始するトリガー。

   AWS Glue のトリガーの詳細については、「トリガーを使用したジョブとクローラの開始」を参照してください。
4. スケジュールされたクローラとトリガーされたジョブをモニタリングします。

   AWS Glue コンソールを使用して以下を表示します。

   - ジョブの実行の詳細とエラー。
   - クローラは詳細とエラーを実行します。
   - AWS Glue アクティビティに関する通知

   AWS Glue でクローラとジョブをモニタリングする方法の詳細については、「AWS Glue のモニタリング」を参照してください。

AWS Glue 用の IAM アクセス許可のセットアップ

このトピックの手順は、AWS Glue の AWS Identity and Access Management (IAM) アクセス許可をすばやくセットアップするのに役立ちます。以下のタスクを実行します。

- IAM ID に AWS Glue リソースへのアクセスを許可します。
- ジョブの実行、データへのアクセス、AWS Glue Data Quality タスクの実行のためのサービスロールを作成します。

AWS Glue の IAM アクセス許可をカスタマイズするために使用できる詳細な手順については、「AWS Glue の IAM アクセス許可の設定」を参照してください。

AWS Management Console で AWS Glue の IAM アクセス許可をセットアップするには

1. AWS Management Consoleにサインインし、AWS Glue コンソール (https://console.aws.amazon.com/glue/) を開きます。

2. [開始方法] を選択します。
3. [AWS Glue のアカウントの準備] で、[IAM アクセス許可のセットアップ] を選択します。
4. AWS Glue アクセス許可を付与する IAM ID (ロールまたはユーザー) を選択します。AWS Glue は `AWSGlueConsoleFullAccess` マネージドポリシーをこれらの ID にアタッチします。これらのアクセス許可を手動で設定する場合、またはデフォルトのサービスロールを設定するだけでよい場合は、このステップをスキップできます。
5. [次へ] を選択します。
6. ロールとユーザーが必要とする Amazon S3 アクセス権のレベルを選択します。このステップで選択したオプションは、選択したすべての ID に適用されます。

   a. [S3 ロケーションの選択] で、アクセスを許可する Amazon S3 ロケーションを選択します。

   b. 次に、上記で選択したロケーションに対して各 ID に付与するアクセス権を [読み取り専用] (推奨) にするか [読み取りおよび書き込み] にするかを選択します。AWS Glue は、選択した読み取りまたは書き込みアクセス許可と ロケーションとの組み合わせに基づいて、アクセス許可ポリシーを ID に追加します。

   次の表は、AWS Glue が Amazon S3 アクセスに付与するアクセス許可を示しています。

選択内容	AWS Glue がアタッチする内容
変更なし	アクセス許可は付与されません。AWS Glue は ID のアクセス許可を何も変更しません。
特定の Amazon S3 ロケーションへのアクセスを許可する (読み取り専用)	カスタマー管理ポリシー (`AWSGlueConsole-S3-read-only-policy`) は、読み取り専用アクセス許可を持つ特定の Amazon S3 ロケーションへのアクセスを許可します。 `{ "Version": "2012-10-17", "Statement": [ { "Effect": "Allow", "Action": [`

選択内容	AWS Glue がアタッチする内容

	"s3:GetObject"
	],
	"Resource": [
	"arn:aws:
	s3:::amzn-s3-demo-bucket1/*",
	"arn:aws:
	s3:::amzn-s3-demo-bucket2/*",
	"arn:aws:
	s3:::amzn-s3-demo-bucket3",
	"arn:aws:
	s3:::amzn-s3-demo-bucket"
	],
	"Condition": {
	"StringEquals": {
	"aws:Reso
	urceAccount":"0000000000"
	}
	}
	}
	]
	}

選択内容	AWS Glue がアタッチする内容

特定の Amazon S3 ロケーションへのアクセスを許可する (読み取りと書き込み)	AWSGlueConsole-S3-read-and-write-policy は、読み取りと書き込みのアクセス許可を持つ特定の Amazon S3 ロケーションへのアクセスを許可します。
	
	{
	"Version": "2012-10-17",
	"Statement": [
	{
	"Effect": "Allow",
	"Action": [
	"s3:GetObject",
	"s3:PutObject"
	],
	"Resource": [
	"arn:aws:
	s3:::aes-siem-00000000000-log/
	*",
	"arn:aws:
	s3:::aes-siem-00000000000-s
	napshot/*",
	"arn:aws:
	s3:::aes-siem-00000000000-l
	og",
	"arn:aws:
	s3:::aes-siem-00000000000-s
	napshot"
	],
	"Condition": {
	"StringEquals": {
	"aws:Reso
	urceAccount":"00000000000"
	}
	}
	}
	]
	}

7. [次へ] を選択します。

8. アカウントのデフォルトの AWS Glue サービスロールを選択します。サービスロールは、AWS Glue がユーザーに代わって他の AWS のサービスのリソースにアクセスするのに使用する IAM ロールです。詳細については、「AWS Glue のサービスロール」を参照してください。

   - 標準の AWS Glue サービスロールを選択すると、AWS Glue は AWS アカウント に新しい IAM ロールを `AWSGlueServiceRole` という名前で作成して、次のマネージドポリシーをアタッチします。アカウントに既に `AWSGlueServiceRole` という名前の IAM ロールがある場合は、AWS Glue はこれらのポリシーをその既存のロールにアタッチします。

     - AWSGlueServiceRole – この管理ポリシーは、AWS Glue がユーザーに代わってリソースにアクセスして管理するために必要です。これにより、AWS Glue は AWS Glue ジョブ、クローラー、接続などのさまざまなリソースを作成、更新、削除できます。このポリシーは、ログ記録の目的で Amazon CloudWatch ログにアクセスするための AWS Glue のアクセス許可も付与します。開始するには、このポリシーを使用して AWS Glue の使用方法を学ぶことをお勧めします。AWS Glue に慣れるにつれて、必要に応じてリソースへのアクセスを微調整できるポリシーを作成できます。

     - AWSGlueConsoleFullAccess – この管理ポリシーは、AWS Management Console を介した AWS Glue サービスへのフルアクセスを付与します。このポリシーは、AWS Glue 内で任意のオペレーションを実行するアクセス許可を付与し、必要に応じて任意の AWS Glue リソースを作成、変更、削除することを可能にします。ただし、このポリシーでは、基盤となるデータストアや、ETL プロセスに関係する可能性のあるその他の AWS のサービスにアクセスするためのアクセス許可は付与されないことに注意してください。`AWSGlueConsoleFullAccess` ポリシーによって付与されるアクセス許可の範囲が広いため、最小特権の原則に従って慎重に割り当てる必要があります。通常、可能な場合は、特定のユースケースと要件に合わせたより詳細なポリシーを作成して使用することをお勧めします。

     - AWSGlueConsole-S3-read-only-policy – このポリシーは、指定された Amazon S3 バケットから AWS Glue がデータを読み取ることを許可しますが、Amazon S3 のデータに対する書き込みと変更を行うアクセス許可は付与しません。

       AWSGlueConsole-S3-read-and-write – このポリシーは、ETL プロセスの一環として、指定された Amazon S3 バケットに対して AWS Glue がデータを読み書きすることを許可します。

   - 既存の IAM ロールを選択すると、AWS Glue は、そのロールをデフォルトとして設定しますが、ロールに `AWSGlueServiceRole` アクセス許可は追加されません。ロールが AWS Glue のサービスロールとして使用されるように設定したことを確認してください。詳細につい

ては、「ステップ 1: AWS Glue サービスの IAM ポリシーを作成する」および「ステップ 2: AWS Glue 用の IAM ロールを作成する」を参照してください。

9. [次へ] を選択します。
10. 最後に、選択したアクセス許可を確認し、[変更を適用] を選択します。変更を適用すると、AWS Glue は選択した ID に IAM アクセス許可を追加します。IAM コンソール (https://console.aws.amazon.com/iam/) で新しいアクセス許可の表示や変更ができます。

これで、AWS Glue の最小限の IAM アクセス許可のセットアップが完了しました。実稼働環境では、「「AWS Glue」 のセキュリティ」と「AWS Glue のアイデンティティとアクセスの管理」をよく理解し、ユースケースに対して AWS リソースを確保できるようにすることをお勧めします。

次のステップ

IAM アクセス許可の設定が完了したので、次のトピックを確認して AWS Glue の使用を開始できます。

- 「AWS スキルビルダー」の「AWS Glue の開始方法」
- AWS Glue Data Catalog の開始方法

AWS Glue Studio の設定

ビジュアル ETL 用に AWS Glue を初めて使用する場合は、このセクションのタスクを完了してください。

トピック

- AWS Glue Studio ユーザーに必要な IAM アクセスアクセス許可を確認します
- ETL ジョブに必要なIAMアクセス許可を確認する
- AWS Glue Studio に対する IAM のアクセス許可の設定
- ETL ジョブの VPC を設定します

AWS Glue Studio ユーザーに必要な IAM アクセスアクセス許可を確認します

AWS Glue Studio を使用するには、ユーザーはさまざまな AWS リソースにアクセスできる必要があります。ユーザーは、Amazon S3 バケット、IAM ポリシーとロール、および AWS Glue Data Catalog オブジェクト。

AWS Glue サービスのアクセス許可

AWS Glue Studio は、AWS Glue サービスのアクションとリソースを使用します。AWS Glue Studio を効果的に使用するには、ユーザーはこれらのアクションやリソースに対するアクセス許可が必要です。AWS Glue Studio ユーザーに `AWSGlueConsoleFullAccess` マネージドポリシーを付与するか、より少ないアクセス許可の組み合わせでカスタムポリシーを作成できます。

> **Important**
>
> セキュリティのベストプラクティスに従って、ポリシーを強化して、Amazon S3 バケットおよび Amazon CloudWatch ロググループへのアクセスをさらに制限することが推奨されます。Amazon S3 ポリシーの例については、「Writing IAM Policies: How to Grant Access to an Amazon S3 Bucket」を参照してください。

AWS Glue Studio のためのカスタム IAM ポリシーの作成

AWS Glue Studio では、カスタムポリシーを作成するためのアクセス許可セットを、より小規模にできます。カスタムポリシーは、オブジェクトまたはアクションのサブセットに対しアクセス許可を付与できます。カスタムポリシーを作成する際には、以下の情報を参考にしてください。

AWS Glue Studio API を使用するには、IAM アクセス許可のアクションポリシーに `glue:UseGlueStudio` を含めます。`glue:UseGlueStudio` を使用することで、時間の経過とともに API にさらにアクションが追加されても、すべての AWS Glue Studio アクションにアクセスできます。

AWS Glue で定義されるアクションの詳細については、「AWS Glue で定義されるアクション」を参照してください。

データ準備の作成アクション

- SendRecipeAction
- GetRecipeAction

有向非巡回 (DAG) に関するアクション

- CreateDag
- UpdateDag

- GetDag
- DeleteDag

ジョブに関するアクション

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

ジョブ実行に関するアクション

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

スキーマに関するアクション

- GetSchema
- GetInferredSchema

データベースに関するアクション

- GetDatabases

プランに関するアクション

- GetPlan

テーブルに関するアクション

- SearchTables
- GetTables
- GetTable

接続に関するアクション

- CreateConnection
- DeleteConnection
- UpdateConnection
- GetConnections
- GetConnection

マッピングに関するアクション

- GetMapping

S3 プロキシに関するアクション

- ListBuckets
- ListObjectsV2
- GetBucketLocation

セキュリティ設定に関するアクション

- GetSecurityConfigurations

スクリプトに関するアクション

- CreateScript (AWS Glue での同じ名前の API とは異なります)

AWS Glue Studio API へのアクセス

AWS Glue Studio にアクセスするには、IAM アクセス許可のアクションポリシーリストに `glue:UseGlueStudio` を追加します。

以下の例では、`glue:UseGlueStudio` はアクションポリシーに含まれますが、AWS Glue Studio API は個別に識別されません。これは、`glue:UseGlueStudio` を含めることで、IAMアクセス許可で個別に AWS Glue Studio APIを指定する必要なしに、内部 API へのアクセス許可が自動的に付与されるためです。

この例では、リストされた追加のアクションポリシー (`glue:SearchTables` など) が AWS Glue Studio API ではないため、場合によっては IAM アクセス許可に含める必要があります。Amazon S3 プロキシアクションを含めて、付与する Amazon S3 アクセス許可のレベルを指定することもできます。以下のポリシーの例では、選択した IAM ロールに十分なアクセス許可がある場合に、AWS Glue Studio を開く、ビジュアルジョブを作成する、ビジュアルジョブを保存/実行するためのアクセス許可が付与されます。

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

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

ノートブックとデータのプレビューアクセス許可

データプレビューとノートブックを使用すると、ジョブを実行することなく、ジョブの任意の段階（読み取り、変換、書き込み）でデータのサンプルを表示できます。データにアクセスするときに使用する AWS Identity and Access Management の AWS Glue Studio (IAM)ロールを指定します。IAM ロールは想定可能であることを意図しており、標準の長期認証情報 (パスワードやアクセスキーなど) を関連付けることはありません。その代わり、AWS Glue Studio がそのロールを引き受けて、IAM は一時的なセキュリティ認証情報を提供します。

データプレビューとノートブックコマンドが正しく動作するようにするには、文字列 `AWSGlueServiceRole` で始まる名前のロールを使用します。ロールに別の名前を使用する場合は、`iam:passrole` アクセス許可を追加し、IAM のロールに対するポリシーを設定する必要があります。詳しくは、「「AWSGlueServiceRole*」 という名前ではないロールの IAM ポリシーを作成します。 」を参照してください。

> **Warning**
>
> ロールがノートブックに対する `iam:passrole` アクセス許可を与えた場合、ロールチェーンを実装すると、あるユーザーが意図せずにノートブックにアクセスする可能性があります。現在、ノートブックへアクセス許可されているユーザーをモニタリングできる監査は実装されていません。

IAM ID によるデータプレビューセッションの作成を認めない場合は、「the section called “ID によるデータプレビューセッションの作成を拒否する”」の例を参照してください。

Amazon CloudWatch のアクセス許可

AWS Glue Studio を使用して Amazon CloudWatch ジョブをモニタリングできます。これは、AWS Glue から raw データを収集して読み取り可能なほぼリアルタイムのメトリクスに処理します。デフォルトでは、AWS Glue メトリクスデータは CloudWatch に自動的に送信されます。詳細については、 Amazon CloudWatch ユーザーガイドの「Amazon CloudWatch とは何ですか?」およびAWS Glue デベロッパーガイドの「AWS Glue メトリクス」を参照してください。

CloudWatch ダッシュボードにアクセスするには、AWS Glue Studio では以下のいずれかが必要です。

- `AdministratorAccess` ポリシー
- `CloudWatchFullAccess` ポリシー

- これらの特定のアクセス許可の 1 つ以上を含むカスタムポリシー。
  - `cloudwatch:GetDashboard` および `cloudwatch:ListDashboards` でダッシュボードを表示する
  - `cloudwatch:PutDashboard` でダッシュボードを作成または変更する
  - `cloudwatch:DeleteDashboards` でダッシュボードを削除する

ポリシーを使用してIAMユーザーのアクセス許可を変更する方法の詳細については、IAM ユーザーガイドのIAM ユーザーのアクセス許可の変更を参照してください。

ETL ジョブに必要なIAMアクセス許可を確認する

を使用してジョブを作成する場合 AWS Glue Studio、ジョブは、作成時に指定したIAMロールのアクセス許可を引き受けます。このIAMロールには、データソースからデータを抽出し、ターゲットにデータを書き込み、 AWS Glue リソースにアクセスするアクセス許可が必要です。

ジョブ用に作成するロールの名前は、 によって正しく使用される文字列で始`AWSGlueServiceRole`まる必要があります。AWS Glue Studio。 例えば、ロール に名前を付けることができます`AWSGlueServiceRole-FlightDataJob`。

データソースとデータターゲットのアクセス許可

An AWS Glue Studio ジョブには、ジョブで使用するソース、ターゲット、スクリプト、一時ディレクトリの Amazon S3 へのアクセス権が必要です。特定の Amazon S3 リソースにきめ細かいアクセス許可を付与するポリシーを作成できます。

- データソースには、`s3:ListBucket` および `s3:GetObject` アクセス許可が必要です。
- データターゲットには、`s3:ListBucket`、`s3:PutObject`、および `s3:DeleteObject` アクセス許可が必要です。

> **Note**
>
> IAM ポリシーでは、 AWS Glue 変換のホストに使用される特定のバケット`s3:GetObject`を許可する必要があります。
> 次のバケットは AWS サービスアカウントによって所有されており、世界中で読み取り可能です。これらのバケットは、 AWS Glue Studio ビジュアルエディタからアクセスできる変換のサブセットに関連するソースコードのリポジトリとして機能します。バケットのアクセス許可は、バケットの他のAPIアクションを拒否するように設定されます。変換用に提供して

いるこれらのスクリプトは誰でも読むことができますが、サービスチーム外のユーザーはその中に何も「入力」することができません。 AWS Glue ジョブが実行されると、そのファイルはローカルインポートとしてプルインされるため、ファイルはローカルコンテナにダウンロードされます。その後、そのアカウントとの通信はそれ以上行われません。

リージョン: バケット名

- af-south-1: aws-glue-studio-transforms-762339736633prod-af-south--1
- ap-east-1: aws-glue-studio-transforms-125979764932-prod-ap-east-1
- ap-northeast-2: aws-glue-studio-transforms-673535381443prod-ap-northeast--2
- ap-northeast-3: aws-glue-studio-transforms-149976050262-prod-ap-northeast-3
- ap-south-1: aws-glue-studio-transforms-584702181950-prod-ap-south-1
- ap-south-2: aws-glue-studio-transforms-380279651983prod-ap-south-2
- ap-southeast-1: aws-glue-studio-transforms-737106620487-prod-ap-southeast-1
- ap-southeast-2: aws-glue-studio-transforms-234881715811-prod-ap-southeast-2
- ap-southeast-3: aws-glue-studio-transforms-151265630221-prod-ap-southeast-3
- ap-southeast-4: aws-glue-studio-transforms-052235663858-prod-ap-southeast-4
- ca-central-1: aws-glue-studio-transforms-622716468547-prod-ca-central-1
- ca-west-1: aws-glue-studio-transforms-915795495192-prod-ca-west-1
- eu-central-1: aws-glue-studio-transforms-560373232017-prod-eu-central-1
- eu-central-2: aws-glue-studio-transforms-907358657121prod-eu-central-2
- eu-north-1: aws-glue-studio-transforms-312557305497prod-eu-north-1
- eu-south-1: aws-glue-studio-transforms-939684186351-prod-eu-south-1
- eu-south-2: aws-glue-studio-transforms-239737454084prod-eu-south-2
- eu-west-1: aws-glue-studio-transforms-244479516193prod-eu-west--1
- eu-west-2: aws-glue-studio-transforms-804222392271prod-eu-west-2
- eu-west-3: aws-glue-studio-transforms-371299348807-prod-eu-west-3
- il-central-1: aws-glue-studio-transforms-806964611811-prod-il-central-1
- me-central-1: aws-glue-studio-transforms-733304270342prod-me-central--1
- me-south-1: aws-glue-studio-transforms-112120182341-prod-me-south-1

- sa-east-1: aws-glue-studio-transforms-881619130292-prod-sa-east-1
- us-east-1: aws-glue-studio-transforms-510798373988prod-us-east-1
- us-east-2: aws-glue-studio-transforms-251189692203prod-us-east-2
- us-west-1: aws-glue-studio-transforms-593230150239prod-us-west-1
- us-west-2: aws-glue-studio-transforms-818035625594prod-us-west-2
- ap-northeast-1: aws-glue-studio-transforms-200493242866-prod-ap-northeast-1
- cn-north-1: aws-glue-studio-transforms-071033555442-prod-cn-north-1
- cn-northwest-1: aws-glue-studio-transforms-070947029561-prod-cn-northwest-1
- us-gov-west-1: aws-glue-studio-transforms-227493901923-prod-us-gov-west-1-2604

データソース Amazon Redshift として を選択した場合は、クラスターアクセス許可のロールを指定できます。 Amazon Redshift クラスターに対して実行されるジョブは、一時的な認証情報を使用して Amazon S3 for temporary storage にアクセスするコマンドを発行します。ジョブが 1 時間を超えて実行されると、これらの認証情報が期限切れになり、ジョブが失敗します。この問題を回避するために、一時的な認証情報を使用してジョブに必要なアクセス許可を付与するロールを Amazon Redshift クラスター自体に割り当てることができます。詳細については、AWS Glue データベースデベロッパーガイドから「Moving Data to and from Amazon Redshift」を参照してください。

ジョブが Amazon S3 以外のデータソースまたはターゲットを使用している場合は、ジョブが使用するIAMロールに必要なアクセス許可をアタッチして、これらのデータソースとターゲットにアクセスする必要があります。詳細については、AWS Glue デベロッパーガイドの「Setting Up Your Environment to Access Data Stores」を参照してください。

データストアにコネクタと接続を使用している場合は、追加のアクセス許可が必要です。詳細については、」the section called “コネクタの使用に必要なアクセス許可”。

ジョブの削除に必要なアクセス許可

In AWS Glue Studio コンソールで複数のジョブを選択して削除できます。このアクションを実行するには、`glue:BatchDeleteJob` アクセス許可が必要です。これは、AWS Glue コンソール。これには、ジョブを削除するための`glue:DeleteJob`アクセス許可が必要です。

AWS Key Management Service アクセス許可

AWS Key Management Service （ AWS KMS) でサーバー側の暗号化を使用する Amazon S3 ソースとターゲットにアクセスする場合は、 にポリシーをアタッチします。AWS Glue

Studio ジョブがデータを復号化できるようにするジョブで使用されるロール。ジョブロールには、kms:ReEncrypt、kms:GenerateDataKey、および kms:DescribeKey のアクセス許可が必要です。さらに、ジョブロールには、 AWS KMS カスタマーマスターキー () で暗号化された Amazon S3 オブジェクトをアップロードまたはダウンロードするためのkms:Decryptアクセス許可が必要ですCMK。

の使用には追加料金がかかります AWS KMS CMKs。詳細については、AWS Key Management Service 「 デベロッパーガイド」の「コンセプト - カスタマーマスターキー (CMKs) 」とAWS Key Management Service 「料金」を参照してください。 AWS Key Management Service

コネクタの使用に必要なアクセス許可

を使用している場合 AWS Glue データストアにアクセスするためのカスタムコネクタと接続、 の実行に使用されるロール AWS Glue ETL ジョブには、追加のアクセス許可がアタッチされている必要があります。

- から購入したコネクタにアクセスAmazonEC2ContainerRegistryReadOnlyするためのAWS管理ポリシー AWS Marketplace。
- glue:GetJob および glue:GetJobs アクセス許可。
- AWS Secrets Manager 接続で使用されるシークレットにアクセスするためのアクセス許可。ポリシー例のシークレット値を取得するアクセス許可の例を参照してくださいIAM。

が AWS Glue ETL ジョブはVPC実行中の Amazon 内で実行されVPC、 の説明に従って を設定VPC する必要がありますthe section called “ETL ジョブの VPC を設定します”。

AWS Glue Studio に対する IAM のアクセス許可の設定

AWS 管理者ユーザーを使用して、ロールを作成し、ユーザーとジョブロールにポリシーを割り当てることができます。

AWSGlueConsoleFullAccess AWS マネージドポリシーを使用すると、AWS Glue Studio コンソールを使用するために必要なパーミッションを提供することができます。

独自のポリシーを作成するには、AWS Glueデベロッパーガイド の「Create an IAM Policy for the AWS Glue Service」に従ってください。AWS Glue Studio ユーザーに必要な IAM アクセスアクセス許可を確認します で前述の IAM アクセス許可を含めます。

トピック

- AWS Glue Studio ユーザーにポリシーを添付する。
- 「AWSGlueServiceRole*」 という名前ではないロールの IAM ポリシーを作成します。

AWS Glue Studio ユーザーにポリシーを添付する。

AWS コンソールにサインインする AWS Glue Studio ユーザーは、特定のリソースへのアクセス権限が必要です。IAM ポリシーをユーザーに割り当てることで、これらの権限を提供します。

ユーザーに AWSGlueConsoleFullAccess マネージドポリシーをアタッチするには

1. AWS Management Console にサインインして、IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. ナビゲーションペインで、[ポリシー] を選択します。
3. ポリシーのリストで、[AWSGlueConsoleFullAccess] ポリシーの横にあるチェックボックスを選択します。[Filter (フィルター)] メニューと検索ボックスを使用して、ポリシーのリストをフィルタリングできます。
4. [Policy actions] を選択して、[Attach] を選択します。
5. ポリシーをアタッチするユーザーを選択します。[Filter] メニューと検索ボックスを使用して、プリンシパルエンティティのリストをフィルタリングできます。ポリシーをアタッチするユーザーを選択し、[ポリシーのアタッチ] を選択します。
6. 必要に応じて前の手順を繰り返し、ユーザーに追加のポリシーをアタッチします。

「AWSGlueServiceRole*」 という名前ではないロールの IAM ポリシーを作成します。

AWS Glue Studio で使用されるロールの IAM ポリシーを設定する方法

1. AWS Management Console にサインインして、IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. 新しい IAM ポリシーを追加します。既存のポリシーに追加するか、新しい IAM インラインポリシーを作成できます。IAM ポリシーを作成するには

    1. [Policies] を選択してから、[Create Policy] を選択します [ご利用開始にあたって] ボタンが表示されたら、そのボタンを選択して、[ポリシーの作成] を選択します。
    2. [独自のポリシーを作成] の横で、[選択] を選択します。
    3. [ポリシー名] に、後で参照しやすい任意の値を入力します。オプションとして、[説明] に説明のテキストを入力します。

4. [ポリシードキュメント] に、以下の形式のポリシーステートメントを入力してから、[ポリシーの作成] を選択します。

3. 次のブロックをコピーして、[構文] 配列の下のポリシーに貼り付け、*my-interactive-session-role-prefix* を、AWS Glue のアクセス許可に関連付けるすべての一般的なロールのプレフィックスに置き換えます。

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

ポリシーに含まれる、[Version] (バージョン) 配列と [Statement] (ステートメント) 配列の完全な例を以下に示します。

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

```
    ]
}
```

4. ユーザーのポリシーを有効にするには、[ユーザー] を選択します。
5. ポリシーをアタッチする ユーザーを選択します。

ETL ジョブの VPC を設定します

Amazon Virtual Private Cloud (Amazon VPC) を使用すると、AWS クラウド 内の論理的に分離された独自の領域に仮想プライベートクラウド (VPC) と呼ばれる仮想ネットワークを定義できます。AWS のリソース (インスタンスなど) を VPC 内部で起動できます。VPC は、お客様自身のデータセンターで運用されている従来のネットワークによく似ていますが、 のスケーラブルなインフラストラクチャを使用できるというメリットがありますAWS お客様の VPC はお客様が設定できます。IP アドレスレンジの選択、サブネットの作成、ルートテーブル、ネットワークゲートウェイ、セキュリティの設定ができます。VPC のインスタンスをインターネットに接続できます。VPC を自社のデータセンターに接続し、AWS クラウド をデータセンターの拡張部分として使用できます。各サブネットでのリソースの保護には、セキュリティグループ、ネットワークアクセスコントロールリストなど、複数のセキュリティレイヤーを使用できます。詳細については、Amazon VPC ユーザーガイドを参照してください。

コネクタを使用するときに、AWS Glue ETL ジョブをVPC 内で実行するように設定できます。必要に応じて、次のように VPC を設定する必要があります。

- AWS にないデータストア用のパブリックネットワークアクセス。ジョブによってアクセスされるすべてのデータストアは、VPC サブネットから使用できる必要があります。
- ジョブが VPC リソースとパブリックインターネットの両方にアクセスする必要がある場合は、VPC 内にネットワークアドレス変換 (NAT) ゲートウェイが必要になります。

  詳細については、AWS Glue デベロッパーガイドの「Setting Up Your Environment to Access Data Stores」を参照してください。

AWS Glue Studio 中でのノートブックの使用開始

AWS Glue Studio を通してノートブックをスタートすると、ほんの数秒後にデータを調査し、ジョブスクリプトの開発を開始できるように、すべての設定ステップが実行されます。

以下のセクションでは、AWS Glue Studio で ETL ジョブのノートブックを使用するために、ロールを作成し、適切なアクセス許可を付与する方法について説明します。

AWS Glue で定義されたアクションの詳細については、「AWS Glue で定義されたアクション」を参照してください。

トピック

- IAM ロールに対するアクセス許可の付与

IAM ロールに対するアクセス許可の付与

ノートブックを使用するには、AWS Glue Studio のセットアップが事前に必要です。

AWS Glue でノートブックを使用するには、ロールに次のものが必要です。

- `sts:AssumeRole` アクションに対する AWS Glue との信頼関係。タグ付けが必要な場合は `sts:TagSession`。
- ノートブック、AWS Glue、インタラクティブセッションのすべての許可を含む IAM ポリシー。
- PassRole の IAM ポリシー (ロールがノートブックからインタラクティブセッションに渡されるようにする必要があるため)。

例えば、新しいロールを作成するときに、`AWSGlueConsoleFullAccessRole` などの標準の AWS 管理ポリシーをロールに追加して、ノートブックのオペレーション用と IAM PassRole ポリシー用にそれぞれ新しいポリシーを追加できます。

AWS Glue との信頼関係に必要なアクション

ノートブックセッションを開始する際に、ノートブックに渡されるロールの信頼関係に `sts:AssumeRole` を追加する必要があります。セッションにタグが含まれている場合は、`sts:TagSession` アクションも渡す必要があります。これらのアクションがないと、ノートブックセッションを開始できません。

例:

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

```
            "Action": "sts:AssumeRole"
        }
    ]
}
```

ノートブックの IAM 許可を含むポリシー

次のサンプルポリシーで、ノートブックに必要な AWS IAM 権限について説明します。新しいロールを作成する場合は、以下を含むポリシーを作成します。

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

次の IAM ポリシーを使用して、特定のリソースへのアクセスを許可できます。

- AwsGlueSessionUserRestrictedNotebookServiceRole: セッションを除くすべての AWS Glue リソースへのフルアクセス権限を提供します。ユーザーが、ユーザーに関連付けられているノートブックセッションのみを作成して使用できるようにします。このポリシーには、AWS Glue が他の AWS サービスで AWS Glue リソースを管理するために必要な他のアクセス許可も含まれています。
- AwsGlueSessionUserRestrictedNotebookPolicy: ユーザー自身に関連付けられているノートブックセッションのみを作成および使用できるようにするアクセス許可を提供します。このポリシーには、ユーザーが制限付き AWS Glue セッションロールを渡すことを明示的に許可するアクセス許可も含まれます。

ロールを渡すための IAM ポリシー

ロールを使用してノートブックを作成すると、そのロールがインタラクティブセッションに渡され、その両方で同じロールを使用できるようになります。このように、iam:PassRole のアクセス許可がロールのポリシーの一部として必要です。

次の例を使用して、ロール用の新しいポリシーを作成します。アカウント番号とロール名を自分のものに置き換えます。

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

AWS Glue 使用状況プロファイルの設定

クラウドプラットフォームを使用する主な利点の 1 つは、その柔軟性です。ただし、コンピューティングリソースの作成が容易であるため、管理対象外のままにしてガードレールなしで放置すると、クラウドコストが螺旋するリスクがあります。その結果、管理者は、インフラストラクチャコストの増加を回避しながら、ユーザーが不要なわずらわしさなく作業できるようにバランスを取る必要があります。

AWS Glue 使用ファイルを用いて、管理者は、デベロッパー、テスター、製品チームなど、アカウント内のさまざまなクラスのユーザーの異なったプロファイルを作成できます。各プロファイルは、さまざまなタイプのユーザーに割り当てることができるパラメータの一意のセットです。例えば、デベロッパーはより多くのワーカーを必要とし、最大ワーカー数を増やすことができますが、製品チームに必要なワーカーは少なく、タイムアウトやアイドルタイムアウトの値は低くなります。

ジョブとジョブ実行の動作の例

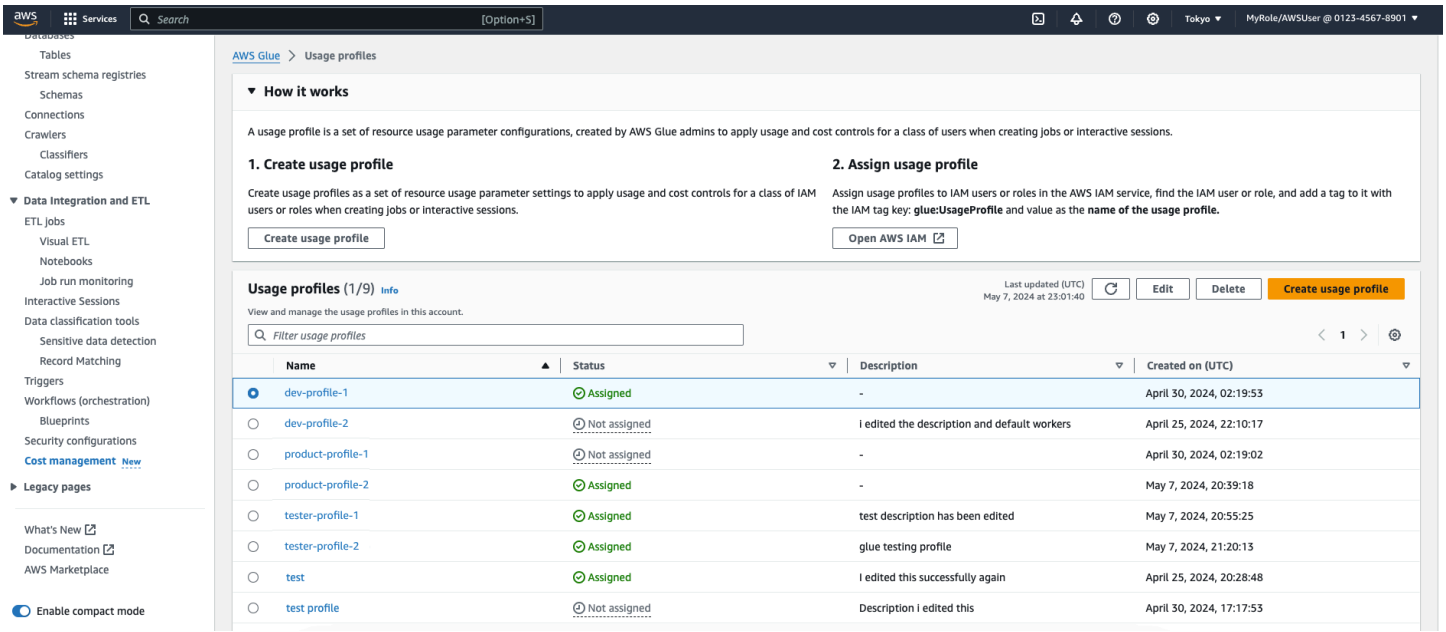
ジョブがプロファイル A を持つユーザー A によって作成されたとします。ジョブは特定のパラメータ値で保存されます。プロファイル B のユーザー B はジョブの実行を試みます。

ユーザー A がジョブを作成したときに、特定の数のワーカーを設定しなかった場合、ユーザー A のプロファイルのデフォルトセットが適用され、ジョブの定義とともに保存されました。

ユーザー B がジョブを実行すると、そのジョブに保存された値とともに実行されます。ユーザー B の独自のプロファイルがより制限されており、その数のワーカーで実行することが許可されていない場合、ジョブの実行は失敗します。

リソースとしての使用状況プロファイル

AWS Glue 使用状況プロファイルは、Amazon リソースネーム (ARN) によって識別されるリソースです。アクションベースおよびリソースベースの承認を含む、すべてのデフォルトの IAM (アイデンティティとアクセス管理) コントロールが適用されます。管理者は、AWS Glue リソースを作成するユーザーの IAM ポリシーを更新し、プロファイルを使用するアクセスを許可する必要があります。



トピック

- 使用プロファイルの作成と管理
- 使用プロファイルとジョブ

使用プロファイルの作成と管理

AWS Glue 使用プロファイルの作成

管理者は使用プロファイルを作成し、さまざまなユーザーに割り当てる必要があります。使用プロファイルを作成するときは、デフォルト値と、さまざまなジョブおよびセッションパラメータに許可

される範囲の値を指定してください。ジョブまたはインタラクティブセッションには、少なくとも 1 つのパラメータを設定する必要があります。ジョブにパラメータ値が指定されていない場合、使用するデフォルト値をカスタマイズすることができます。また、このプロファイルを使用するときにユーザーがパラメータ値を指定した場合、範囲の制限または検証に許可される値のセットを設定することができます。

デフォルトは、管理者がジョブの作成者を支援するために設定したベストプラクティスです。ユーザーが新しいジョブを作成し、タイムアウト値を設定しない場合は、使用プロファイルのデフォルトのタイムアウトが適用されます。作成者にプロファイルがない場合、AWS Glue サービスのデフォルトが適用され、ジョブの定義に保存されます。ランタイム時には、AWS Glue はプロファイルに設定された制限 (最小、最大、許可されたワーカー) を適用します。

パラメータは一度設定すると、他のすべてのパラメータは任意となります。ジョブまたはインタラクティブセッション用にカスタマイズできるパラメータは次のとおりです。

- ワーカーの数 - コンピューティングリソースの過剰な使用を避けるため、ワーカー数を制限します。デフォルト値、最小値、最大値を設定できます。最小値は 1 です。
- ワーカータイプ - ワークロードに関連するワーカータイプを制限します。デフォルトのタイプを設定し、ユーザープロファイルのワーカータイプを許可できます。
- タイムアウト - ジョブまたはインタラクティブセッションが終了する前にリソースを実行および消費できる最大時間を定義します。ジョブの長時間実行を避けるためにタイムアウト値をセットアップします。

  デフォルト値、最小値、最大値を分単位で設定できます。最小値は1 (分) です。AWS Glue デフォルトのタイムアウトは 2,880 分ですが、使用プロファイルで任意のデフォルト値を設定できます。

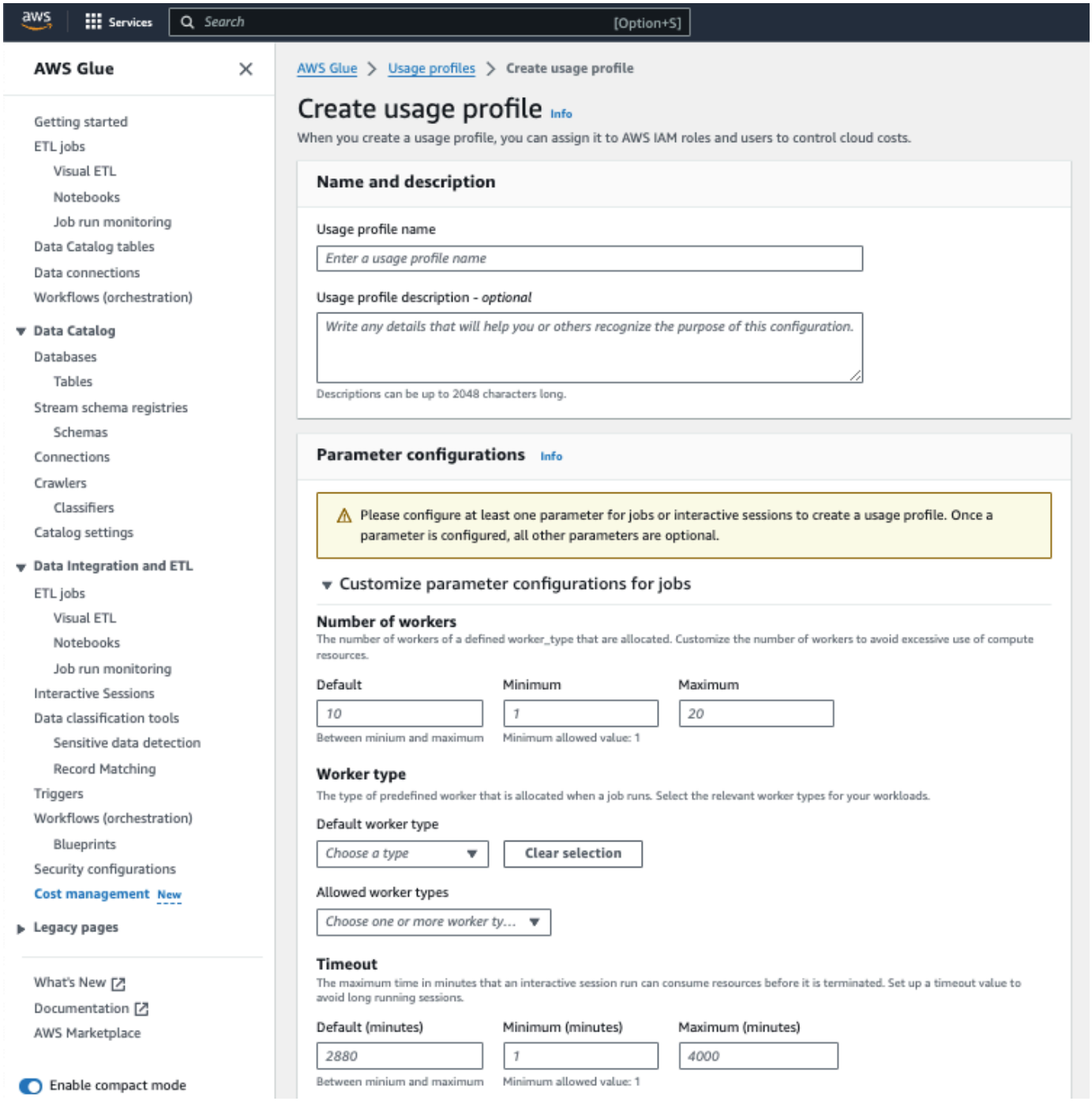
  「デフォルト」の値を設定するのがベストプラクティスです。ユーザーが値を設定しなかった場合、この値がジョブまたはセッションの作成に使用されます。
- アイドルタイムアウト - セルの実行後からタイムアウトするまでにインタラクティブセッションが非アクティブ状態である分数を定義します。作業の完了後、インタラクティブセッション終了のアイドルタイムアウトを定義します。アイドルタイムアウトの範囲は、タイムアウトの制限内である必要があります。

  デフォルト値、最小値、最大値を分単位で設定できます。最小値は1 (分) です。AWS Glue デフォルトのタイムアウトは 2,880 分ですが、使用プロファイルで任意のデフォルト値を設定できます。

「デフォルト」の値を設定するのがベストプラクティスです。ユーザーが値を設定しなかった場合に、この値がセッションの作成に使用されます。

管理者として AWS Glue 使用プロファイルを作成するには (コンソール)

1. 左側のナビゲーションメニューで [コスト管理] を選択します。
2. [使用プロファイルの作成] を選択します。
3. 使用プロファイルの [使用プロファイル名] を入力します。
4. 他のユーザーが使用プロファイルの目的を認識できるようなオプションの説明を入力します。
5. プロファイルのパラメータのうち少なくとも 1 つを定義します。フォーム内のどのフィールドもパラメータフィールドです。例えば、最小セッションアイドルタイムアウト値です。
6. 使用プロファイルに適用される任意のタグを定義します。
7. [Save] を選択します。



使用プロファイルを作成するには (AWS CLI)

1. 次のコマンドを入力します。

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

ここでは、config.json はインタラクティブセッション (SessionConfiguration) とジョブ (JobConfiguration) のパラメータ値を定義できます。

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

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

2. 次のコマンドを入力すると、作成された使用プロファイルを確認できます。

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

レスポンスは以下のようになります。

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

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

使用プロファイルの管理に使用される追加 CLI コマンド:

- aws glue list-usage-profiles
- aws glue update-usage-profile --name *profile-name* --configuration *file://config.json*
- aws glue delete-usage-profile --name *profile-name*

使用プロファイルの編集

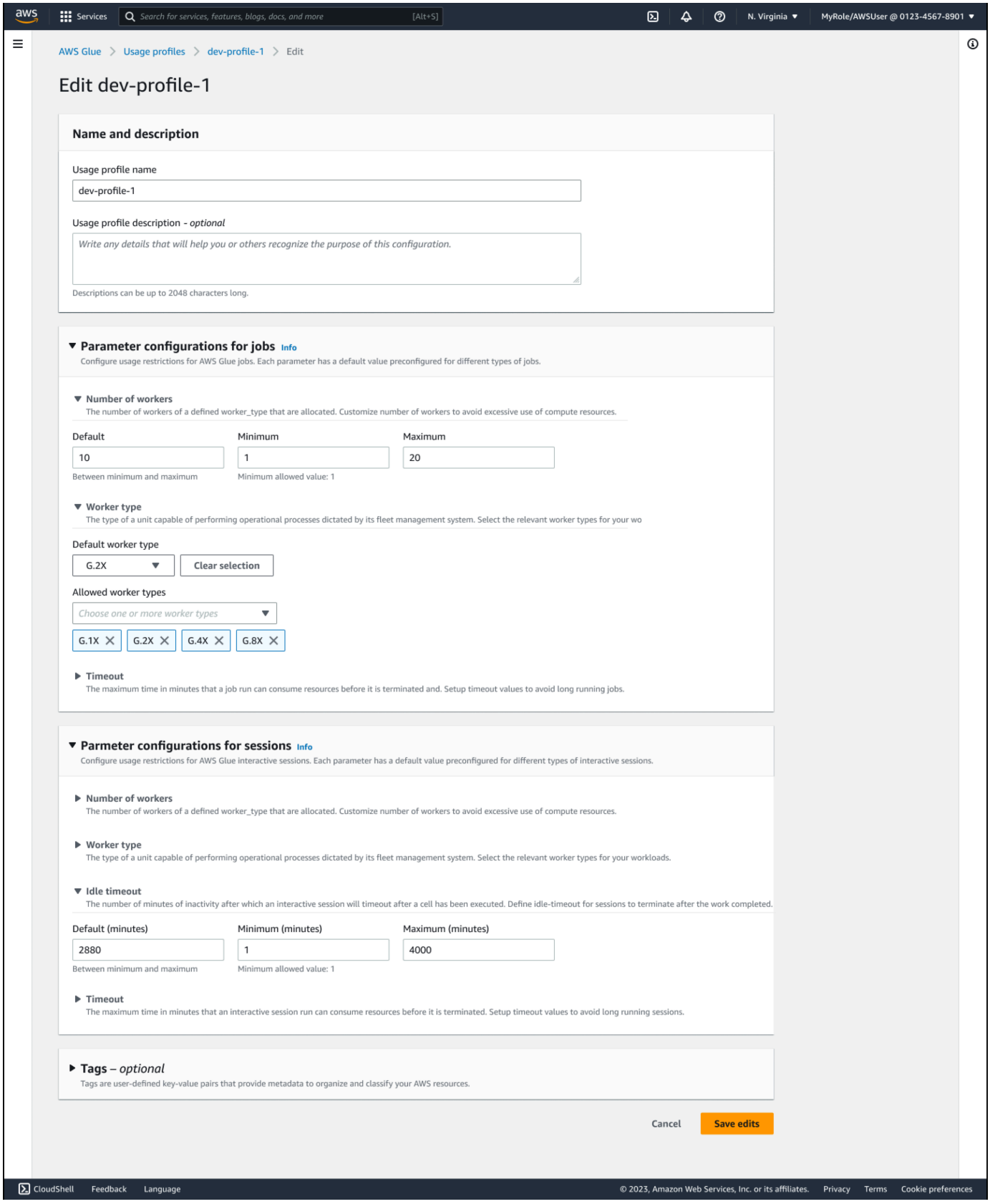
管理者は、作成した使用プロファイルを編集して、ジョブとインタラクティブセッションのプロファイルパラメータ値を変更できます。

使用プロファイルを編集するには:

管理者として AWS Glue 使用プロファイルを編集するには (コンソール)

1. 左側のナビゲーションメニューで [コスト管理] を選択します。

2. 編集する権限を持つ使用プロファイルを選択し、[編集] を選択します。
3. 必要に応じてプロファイルを変更します。デフォルトでは、すでに値を持つパラメータは展開されます。
4. [編集を保存] を選択します。


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

使用プロファイルを編集するには (AWS CLI)

- 次のコマンドを入力します。上記の create コマンドと同じ `--configuration` ファイル構文が使用されます。

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

ここで、config.json はインタラクティブセッション (`SessionConfiguration`) とジョブ (`JobConfiguration`) のパラメータ値を定義します。

使用プロファイルの割り当て

[使用プロファイル] ページの [使用ステータス] 列には、使用プロファイルがユーザーに割り当てられているかどうかが表示されます。ステータスにカーソルを合わせると、割り当てられた IAM エンティティが表示されます。

管理者は、AWS Glue リソースを作成するユーザー/ロールに AWS Glue 使用プロファイルを割り当てることができます。プロファイルの割り当てとは、2 つのアクションの組み合わせです:

- IAM ユーザー/ロールタグを `glue:UsageProfile` キーで更新し、その後に
- ユーザー/ロールの IAM ポリシーを更新。

AWS Glue Studio を使用してジョブ/インタラクティブセッションを作成するユーザーの場合、管理者は次のロールにタグを付けます:

- ジョブの制限については、管理者はログインしたコンソールロールにタグを付けます
- インタラクティブセッションの制限については、管理者はノートブックの作成時にユーザーが提供するロールにタグを付けます

AWS Glue リソースを作成する IAM ユーザー/ロールで管理者が更新する必要があるポリシーの例を次に示します:

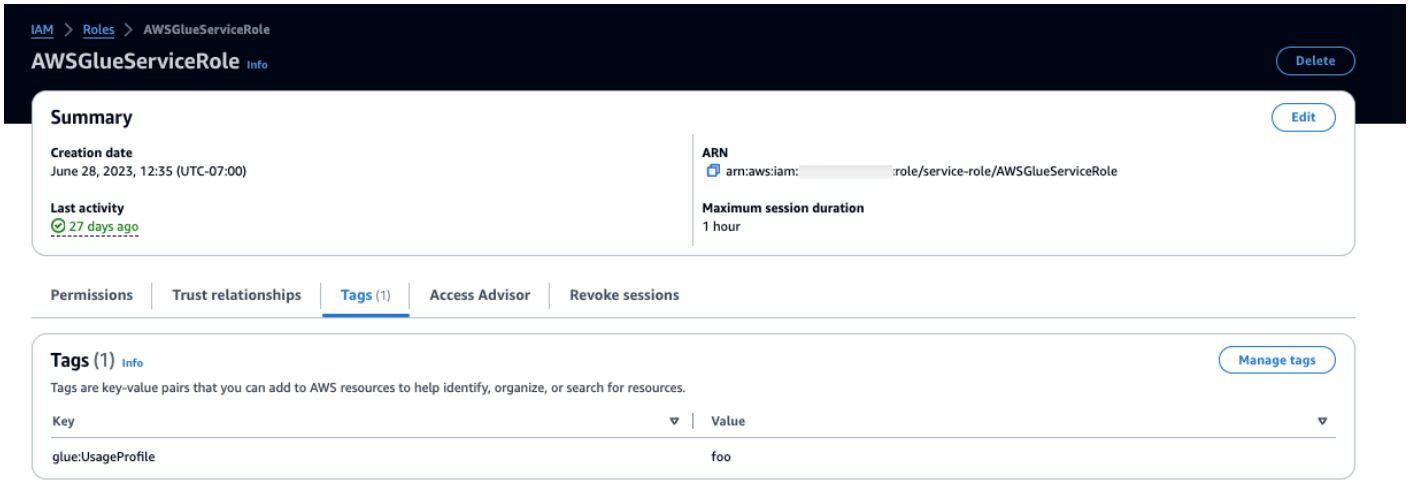
```
{
    "Effect": "Allow",
    "Action": [
        "glue:GetUsageProfile"
    ],
```

```
    "Resource": [
        "arn:aws:glue:us-east-1:123456789012:usageProfile/foo"
    ]
}
```

AWS Glue は、AWS Glue 使用プロファイルで指定された値に基づいてジョブ、ジョブ実行、およびセッションリクエストを検証し、リクエストが許可されていない場合は例外を発生させます。同期 API の場合、エラーがユーザーにスローされます。非同期パスの場合、失敗したジョブ実行は、入力パラメータがユーザー/ロールに割り当てられたプロファイルの許容範囲外であるというエラーメッセージで作成されます。

使用プロファイルをユーザー/ロールに割り当てるには:

1. IAM (Identity and Access Management) コンソールを開きます。
2. ナビゲーションペインで、[ユーザー] または [ロール] を選択します。
3. ユーザーまたはロールを選択します。
4. [タグ] タブを選択します。
5. [新しいタグを追加] をクリックします。
6. [キー] が `glue:UsageProfile`、[値] が使用プロファイルの名前になっているタグを追加します。
7. [Save changes] (変更の保存) を選択します。



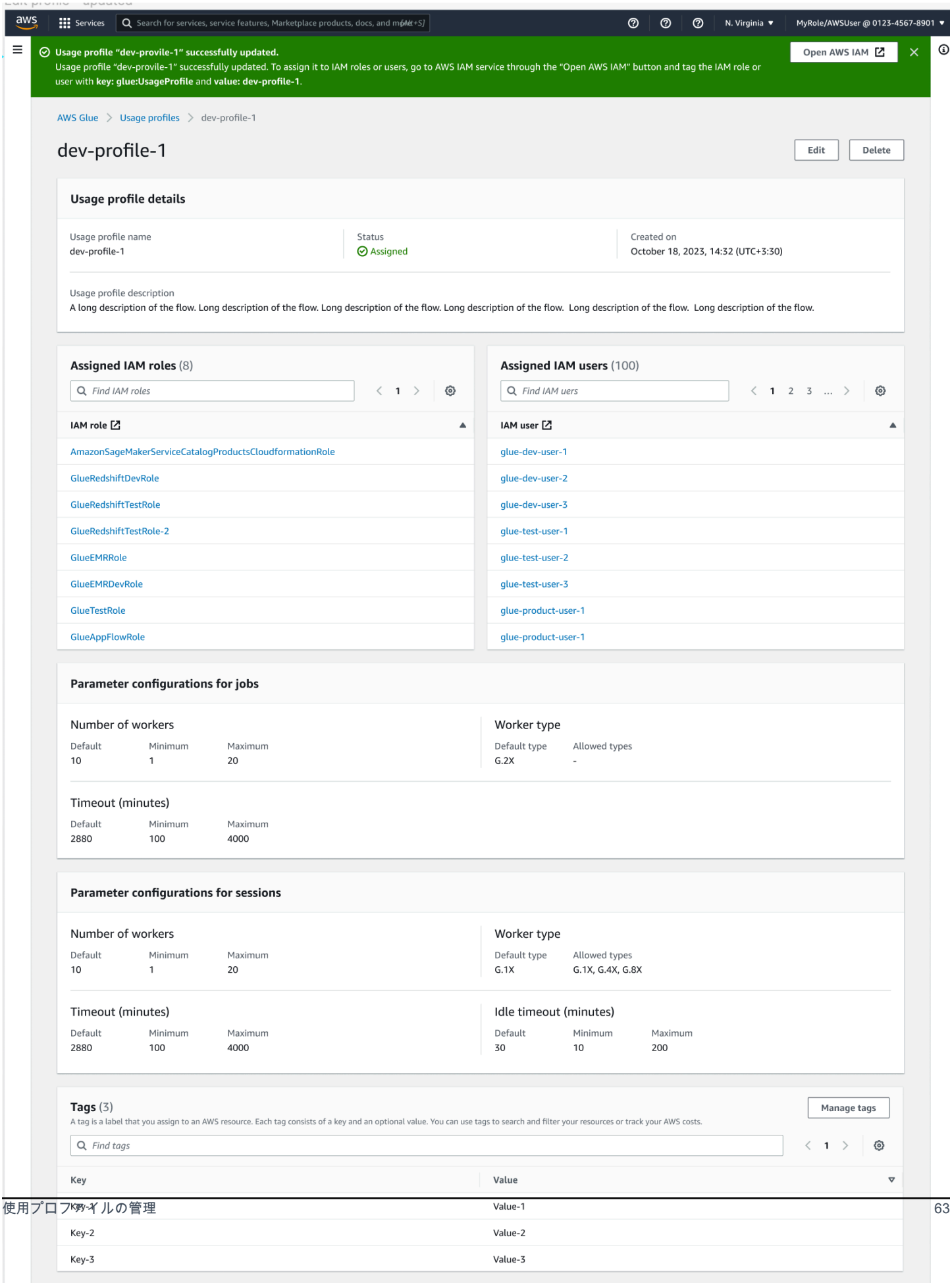
割り当てられた使用プロファイルを表示

ユーザーは割り当てられた使用プロファイルを表示できるほか、使用プロファイルを使って、API コールを行って AWS Glue ジョブやセッションリソースを作成したり、ジョブを開始したりできます。

プロファイルのアクセス許可は IAM ポリシーで提供されます。発信者ポリシーに `glue:UsageProfile` のアクセス許可がある限り、ユーザーはプロファイルを表示できます。それ以外の場合は「アクセス拒否」というエラーが表示されます。

割り当てられた使用プロファイルを表示するには:

1. 左側のナビゲーションメニューで [コスト管理] を選択します。
2. 表示するアクセス許可がある使用プロファイルを選択します。


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

使用プロファイルとジョブ

使用プロファイルでジョブの作成

ジョブの作成中に、使用プロファイルで設定された制限とデフォルトが適用されます。プロファイルは保存時にジョブに割り当てられます。

使用プロファイルでジョブの実行

ジョブの実行を開始すると、AWS Glue は発信者のプロファイルに設定された制限を適用します。直接発信者がいない場合、Glue は作成者によってジョブに割り当てられたプロファイルの制限を適用します。

> **Note**
>
> ジョブがスケジュールに従って実行されると (AWS Glue ワークフローまたは AWS Glue トリガーによる)、作成者がジョブに割り当てたプロファイルが適用されます。
> 外部サービス (Step Functions、MWAA) または `StartJobRun` API によってジョブが実行されると、発信者のプロファイル制限が適用されます。

AWS Glue ワークフローまたは AWS Glue トリガーの場合: プロファイルの制限 (最小、最大、および許可されたワーカー) がランタイムでスケジュールされた実行が適用されるために、既存のジョブを更新して新しいプロファイル名を保存する必要があります。

ジョブに割り当てられた使用プロファイルを表示

ジョブに割り当てられたプロファイル (スケジュールされた AWS Glue ワークフローまたは AWS Glue トリガーでのランタイムに使用される) を表示するには、ジョブの [詳細] タブを確認してください。ジョブ実行の詳細タブで、過去の実行で使用されたプロファイルを確認することもできます。

ジョブにアタッチされた使用プロファイルの更新または削除

ジョブに割り当てられたプロファイルは、更新時に変更されます。作成者に使用プロファイルが割り当てられていない場合、ジョブにアタッチされたプロファイルはすべてそこから削除されます。

AWS Glue Data Catalog の開始方法

AWS Glue Data Catalog は永続的な技術メタデータストアです。AWS クラウドでメタデータを保存、注釈付け、および共有するために使用することができるマネージド型サービスです。詳細については、「AWS Glue Data Catalog」を参照してください。

AWS Glue コンソールおよび一部のユーザーインターフェイスが最近更新されました。

概要

このチュートリアルを使用して、最初の AWS Glue データカタログを作成できます。このデータカタログは、データソースとして Simple Storage Service (Amazon S3) バケットを使用します。

このチュートリアルでは、AWS Glue コンソールを使用して以下の作業を行います。

1. データベースの作成
2. テーブルを作成する
3. データソースとして Amazon S3 バケットを使用します。

これらのステップを完了すると、データソースとして Amazon S3 バケットを使用して、AWS Glue データカタログにデータを入力できるようになります。

ステップ 1: データベースを作成する

開始するには、AWS Management Console にサインインして AWS Glue コンソール を開きます。

AWS Glue コンソールを使用してデータベースを作成する:

1. AWS Glue コンソールで、左側のメニューにある [Data catalog] (データカタログ) から、[Databases] (データベース) を選択します。
2. [Add database] (データベースの追加) を選択します。
3. [データベースの作成] ページで、データベースの名前を入力します。[Location - optional] セクションで、データカタログのクライアントが使用する URI の場所を設定します。これがわからない場合は、データベースの作成を続行してください。
4. (オプション)。データベースの説明を入力します。

5. [データベースの作成] を選択します。

これで、AWS Glue コンソールで最初のデータベースの作成が終わりました。新しいデータベースが、使用可能なデータベースのリストに表示されます。データベースは、[Databases] (データベース) ダッシュボードからデータベースの名前を選択することで編集できます。

次のステップ

[Other ways to create a database:] (データベースを作成するその他の方法)

AWS Glue コンソールでデータベースを作成しましたが、データベースを作成する方法は他にもあります。

- クローラーを使用して、データベースとテーブルを自動的に作成することができます。クローラーを使用してデータベースを設定するには、「AWS Glue コンソールでクローラーを使用する」を参照してください。
- AWS CloudFormation テンプレートを使用できます。「AWS Glue リソースをAWS Glue Data Catalog テンプレートで作成する」を参照してください。
- AWS Glue データベース API オペレーションを使用してデータベースを作成することもできます。

  `create` オペレーションを使用してデータベースを作成するには、`DatabaseInput` (必須) パラメータを含めてリクエストを構成します。

  例:

  CLI、Boto3、または DDL を使用して、チュートリアルで使用した S3 バケットからの同じ flights_data.csv ファイルに基づいてテーブルを定義する方法の一例を以下に示します。

  CLI

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

  Boto3

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

response = glueClient.create_database(
```

```
    DatabaseInput={
        'Name': 'boto3db'
    }
)
```

Database API のデータタイプ、構造、およびオペレーションの詳細については、「データベース API」を参照してください。

次のステップ

次のセクションでは、テーブルを作成し、そのテーブルをデータベースに追加します。

データカタログの設定とアクセス許可を確認することもできます。「AWS Glue コンソールでデータカタログ設定を使用する」を参照してください。

ステップ 2。テーブルを作成する

このステップでは、AWS Glue コンソールを使用してテーブルを作成します。

1. AWS Glue コンソールで、左側のメニューにある[Tables] (テーブル) を選択します。
2. [Add table (テーブルの追加)] を選択します。
3. [Table details] (テーブルの詳細) でテーブルの名前を入力して、テーブルのプロパティを設定します。
4. [Databases] (データベース) セクションのドロップダウンメニューから、ステップ 1 で作成したデータベースを選択します。
5. [Add a data store] (データストアの追加) セクションのデフォルト状態では、ソースのタイプとして [S3] が選択されています。
6. [Data located in] (データがある場所) で、[Specified path in another account] (他のアカウントの指定したパス) を選択します。
7. パスをコピーして、[Include path] (パスを含める) の入力フィールドに貼り付けます。

   ```
   s3://crawler-public-us-west-2/flight/2016/csv/
   ```
8. [Data format] (データ形式) セクションにある [Classification] (分類) で [CSV] を選択し、[Delimiter] (区切り記号) では [comma (,)] (カンマ (,)) を選択します。[Next] を選択します。
9. スキーマを定義するように求められます。スキーマは、データレコードの構造と形式を定義します。[Add column] (列の追加) を選択します。(詳細については、「スキーマレジストリ」を参照してください)。

10. 列のプロパティを指定します。

    a. 列名を入力します。

    b. [Column type] (列タイプ) の場合、デフォルトで 'string' が既に選択されています。

    c. [Column number] (列番号) の場合、デフォルトで '1' が既に選択されています。

    d. [追加] を選択します。

11. パーティションインデックスを追加するように求められます。これはオプションです。このステップをスキップするには、[Next] (次へ) を選択します。

12. テーブルプロパティのサマリーが表示されます。すべてが期待とおりに動作している場合、[作成] を選択します。それ以外の場合は、[Back] (戻る) を選択して、必要に応じて編集します。

これで、テーブルを手動で作成し、データベースに関連付ける作業が完了しました。新しく作成したテーブルは [Tables] (テーブル) ダッシュボードに表示されます。ダッシュボードからは、すべてのテーブルの編集と管理を行うことができます。

詳細については、「AWS Glue コンソールでテーブルを操作する」を参照してください。

次のステップ

次のステップ

データカタログを入力した後は、AWS Glue でジョブの作成を開始できます。「AWS Glue Studio でビジュアル ETL ジョブの作成」を参照してください。

コンソールを使用する以外にも、データカタログでテーブルを定義する方法として、次のような方法があります。

- クローラーを作成して実行する
- AWS Glue のクローラーに分類子の追加
- AWS Glue テーブル API を使用する
- AWS Glue Data Catalog テンプレートを使用する
- Apache Hive メタストアを移行する
- AWS CLI、Boto3、またはデータ定義言語 (DDL) を使用する

  CLI、Boto3、または DDL を使用して、チュートリアルで使用した S3 バケットからの同じ flights_data.csv ファイルに基づいてテーブルを定義する方法の一例を以下に示します。

AWS CLI コマンドを構造化する方法については、ドキュメントを参照してください。CLI の例には、「aws glue create-table --table-input」値の JSON 構文が含まれています。

CLI

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

```
            "columnsOrdered": "true",
            "compressionType": "none",
            "delimiter": ",",
            "skip.header.line.count": "1",
            "typeOfData": "file"
        }
    }
```

Boto3

```
import boto3

glue_client = boto3.client("glue")

response = glue_client.create_table(
    DatabaseName='sampledb',
    TableInput={
        'Name': 'flights_data_manual',
    'StorageDescriptor': {
      'Columns': [{
        'Name': 'year',
        'Type': 'bigint'
      }, {
        'Name': 'quarter',
        'Type': 'bigint'
      }],
      'Location': 's3://crawler-public-us-west-2/flight/2016/csv',
      'InputFormat': 'org.apache.hadoop.mapred.TextInputFormat',
      'OutputFormat':
 'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat',
      'Compressed': False,
      'NumberOfBuckets': -1,
      'SerdeInfo': {
        'SerializationLibrary':
 'org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe',
        'Parameters': {
          'field.delim': ',',
          'serialization.format': ','
        }
      },
    },
    'PartitionKeys': [{
```

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

DDL

```
CREATE EXTERNAL TABLE `sampledb`.`flights_data` (
  `year` bigint,
  `quarter` bigint)
PARTITIONED BY (
  `mon` string)
ROW FORMAT DELIMITED
  FIELDS TERMINATED BY ','
STORED AS INPUTFORMAT
  'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION
  's3://crawler-public-us-west-2/flight/2016/csv/'
TBLPROPERTIES (
  'classification'='csv',
  'columnsOrdered'='true',
  'compressionType'='none',
  'delimiter'=',',
  'skip.header.line.count'='1',
  'typeOfData'='file')
```

データストアへのネットワークアクセスを設定する

抽出、変換、ロード (ETL) ジョブを実行するには，AWS Glue がデータストアにアクセスできる必要があります。ジョブをVirtual Private Cloud (VPC) サブネットで実行する必要がない場合 (例えば、Amazon S3 から Amazon S3 へのデータ変換など)、追加の設定は必要ありません。

ジョブを VPC サブネットで実行する必要がある場合 (プライベートサブネット内の JDBC データストアからのデータの変換など)、AWS Glue は、ジョブを VPC 内の他のリソースに安全に接続できるようにする Elastic Network Interfaces をセットアップします。それぞれの Elastic Network Interface には、指定したサブネット内の IP アドレス範囲からプライベート IP アドレスが割り当てられます。パブリック IP アドレスは割り当てられません。AWS Glue 接続で指定されたセキュリティグループは、各 Elastic Network Interface に適用されます。詳しくは、「AWS Glue から Amazon RDS データストアに JDBC 接続するための Amazon VPC の設定」を参照してください。

ジョブによってアクセスされるすべての JDBC データストアは、VPC サブネットから使用できる必要があります。VPC 内から Amazon S3 にアクセスするには VPC エンドポイント が必須です。ジョブが VPC リソースとパブリックインターネットの両方にアクセスする必要がある場合は、 VPC 内にネットワークアドレス変換 (NAT) ゲートウェイが必要になります。

ジョブまたは開発エンドポイントは、一度に 1 つの VPC (およびサブネット) にのみアクセスできます。異なる VPC のデータストアにアクセスする必要がある場合は、次のオプションがあります。

- VPC ピア接続を使用してデータストアにアクセスします。VPC ピア接続の詳細については、「VPC ピア接続の基本」を参照してください。
- 中間的なストレージとして Amazon S3 バケットを使用します。ジョブ 1 での Amazon S3 出力をジョブ 2 への入力としながら、作業を 2 つのジョブに分割します。

Amazon VPC を使用して Amazon Redshift データストアに接続する方法の詳細については、「the section called “Redshift の設定”」を参照してください。

Amazon VPC を使用して Amazon RDS データストアに接続する方法の詳細については、「the section called “Amazon RDS データストアに接続するための VPC の 設定”」を参照してください。

Amazon VPC で必要なルールを設定したら、データストアに接続するために必要なプロパティを使用して、AWS Glue で接続を作成します。接続の詳細については、「データへの接続」を参照してください

> **Note**
>
> AWS Glue の DNS 環境を必ず設定してください。詳細については、「VPC での DNS のセットアップ」を参照してください。

トピック

- AWS Glue の PyPI に接続する VPC のセットアップ
- VPC での DNS のセットアップ

AWS Glue の PyPI に接続する VPC のセットアップ

Python Package Index (PyPI) は、Python プログラミング言語用のソフトウェアのリポジトリです。このトピックでは、(セッションの作成者が `--additional-python-modules` フラグを使用して指定した) pip インストールパッケージの使用をサポートするために必要な詳細について説明します。

コネクタで AWS Glue インタラクティブセッションを使用すると、コネクタに指定されたサブネット経由で VPC ネットワークが使用されます。そのため、特別な設定をセットアップしない限り、AWS サービスやその他のネットワーク送信先は使用できません。

この問題には、次のような解決策があります。

- セッションからアクセス可能なインターネットゲートウェイの使用。
- パッケージセットの依存関係の推移閉包を含む PyPI/simple リポジトリを使用した S3 バケットのセットアップと使用。
- PyPI をミラーリングし、VPC にアタッチされている CodeArtifact リポジトリの使用。

インターネットゲートウェイをセットアップする

技術面については、「NAT ゲートウェイのユースケース」で詳しく説明していますが、`--additional-python-modules` を使用するには以下の要件に注意してください。具体的には、`--additional-python-modules` には、VPC の設定によって決まる pypi.org へのアクセス権が必要です。次の要件に注意してください。

1. ユーザーのセッションに追加の Python モジュールを pip インストール経由でインストールする要件。セッションがコネクタを使用する場合、設定が影響を受ける可能性があります。

2. --additional-python-modules でコネクタを使用している場合、セッションが開始されると、pypi.org に到達するためのネットワークパスがコネクタの PhysicalConnectionRequirements に関連付けられたサブネットで指定される必要があります。
3. 設定が正しいかどうかを判断する必要があります。

ターゲット PyPI/simple リポジトリをホストする Amazon S3 バケットのセットアップ

この例では、Amazon S3 に一連のパッケージとその依存関係について PyPI のミラーリングをセットアップします。

一連のパッケージに PyPI のミラーリングを設定するには:

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

既存のアーティファクトリポジトリがある場合は、pip 用のインデックス URL があります。これを、上記の Amazon S3 バケットのサンプル URL の代わりに指定できます。

カスタム index-url をいくつかのサンプルパッケージで使用するには:

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

VPC にアタッチされた PyPI の CodeArtifact ミラーリングのセットアップ

ミラーリングをセットアップするには:

1. コネクタが使用するサブネットと同じリージョンにリポジトリを作成します。

   [Public upstream repositories]、[pypi-store] の順に選択します。

2. サブネットの VPC からリポジトリにアクセスできるようにします。
3. `python-modules-installer-option` を使用して正しい `--index-url` を指定します。

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

詳細については、「VPC から CodeArtifact を使用する」を参照してください。

VPC での DNS のセットアップ

ドメインネームシステム (DNS) は、インターネットで使用する名前を対応する IP アドレスに解決するための標準です。DNS ホスト名は、ホスト名とドメイン名で構成され、コンピュータに一意の名前を付けます。DNS サーバーは DNS ホスト名を対応する IP アドレスに解決します。

VPC で DNS をセットアップするには、DNS ホスト名と DNS 解決の両方が VPC で有効になっていることを確認します。VPC ネットワーク属性の `enableDnsHostnames` と `enableDnsSupport` を `true` に設定する必要があります。これらの属性を表示および変更するには、VPC コンソール (https://console.aws.amazon.com/vpc/) を開きます。

詳細については、「Using DNS with Your VPC」を参照してください。また、AWS CLI を使用して modify-vpc-attribute コマンドを実行し、VPC ネットワーク属性を設定することもできます。

> **Note**
>
> Route 53 を使用している場合は、設定によって DNS ネットワーク属性が上書きされないようにする必要があります。

AWS Glue での暗号化のセットアップ

次のワークフローの例では、AWS Glue で暗号化を使用するときに設定するオプションに注目します。この例では、特定の AWS Key Management Service (AWS KMS) キーの使用方法を示していますが、特定のニーズに基づいて他の設定を選択することもできます。このワークフローでは、AWS Glue の設定時に暗号化に関連するオプションのみをハイライト表示しています。

1. AWS Glue コンソールのユーザーがすべての AWS Glue API オペレーションを許可するアクセス許可ポリシー (例: "glue:*") を使用しない場合は、次のアクションが許可されていることを確認してください。
   - "glue:GetDataCatalogEncryptionSettings"
   - "glue:PutDataCatalogEncryptionSettings"
   - "glue:CreateSecurityConfiguration"
   - "glue:GetSecurityConfiguration"
   - "glue:GetSecurityConfigurations"
   - "glue:DeleteSecurityConfiguration"
2. 暗号化されたカタログにアクセスまたは書き込みするクライアント (つまりコンソールユーザー、クローラー、ジョブ、開発エンドポイントなど) には、以下のアクセス許可が必要です。

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

3. 暗号化された接続パスワードにアクセスするユーザーまたはロールには、次のアクセス許可が必要です。

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

4. 暗号化されたデータを Amazon S3 に書き込む抽出、変換、ロード (ETL) ジョブのロールには、以下のアクセス許可が必要です。

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

5. 暗号化された Amazon CloudWatch Logs を書き込む ETL ジョブまたはクローラーでは、キーポリシーおよび IAM ポリシーに以下のアクセス許可が必要です。

   キーポリシー (IAM ポリシーではない) の場合:

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

   キーポリシーの詳細については、AWS Key Management Service デベロッパーガイドの「Using Key Policies in AWS KMS」を参照してください。

   `logs:AssociateKmsKey` 許可にアタッチする IAM ポリシーの場合:

```
{
```

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

6. 暗号化されたジョブのブックマークを使用する ETL ジョブには、次のアクセス許可が必要です。

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

7. AWS Glue コンソールで、ナビゲーションペインの [設定] を選択します。

   a. [Data catalog settings] (データカタログ設定) ページで [Metadata encryption] (メタデータの暗号化) を選択し、Data Catalog を暗号化します。このオプションは、選択した AWS KMS キーを使用して、Data Catalog のすべてのオブジェクトを暗号化します。

   b. [AWS KMS key] で、[aws/glue] を選択します。自身で作成した AWS KMS キーを選択することもできます。

> **Important**
>
> AWS Glue は、対称カスタマーマスターキー (CMK) のみをサポートしています。[AWS KMS key] (KMS キー) リストには、対称キーのみが表示されます。ただし、[Choose a AWS KMS key ARN] (KMS キー ARN を選択します) を選択した場合、任意のキータイプの ARN をコンソールで入力します。対称キーには ARN だけを入力するようにしてください。

暗号化が有効になっている場合、Data Catalog にアクセスするクライアントには、AWS KMS のアクセス許可が付与されている必要があります。

8. ナビゲーションペインで、[セキュリティ設定] を選択します。セキュリティ設定は、AWS Glue プロセスを設定するために使用できるセキュリティプロパティのセットです。次に [セキュリティ設定の追加] を選択します。設定で、次のオプションのいずれかを選択します。

   a. [S3 暗号化] を選択します。[暗号化モード] では、SSE-KMSを選択します。[AWS KMS キー] については、[aws/s3] を選択します (ユーザーがこのキーを使用するアクセス権限を持っていることを確認してください)。これにより、ジョブによって Amazon S3 に書き込まれたデータは、AWS 管理の AWS Glue AWS KMS キーを使用できるようになります。

   b. [CloudWatch Logs の暗号化] を選択し、CMK を選択します (ユーザーにこのキーを使用するアクセス許可があることを確認してください)。詳しくは、AWS Key Management Service デベロッパーガイドの「Encrypt Log Data in CloudWatch Logs Using AWS KMS」を参照してください。

   > **Important**
   >
   > AWS Glue は、対称カスタマーマスターキー (CMK) のみをサポートしています。[AWS KMS key] (KMS キー) リストには、対称キーのみが表示されます。ただし、[Choose a AWS KMS key ARN] (KMS キー ARN を選択します) を選択した場合、任意のキータイプの ARN をコンソールで入力します。対称キーには ARN だけを入力するようにしてください。

   c. [詳細プロパティ] を選択し、[ジョブのブックマーク暗号化モード] を選択します。[AWS KMS キー] には、[aws/glue] を選択します (ユーザーがこのキーを使用するアクセス権限を持っていることを確認してください)。これにより、AWS Glue AWS KMS キーを使用して Amazon S3 に書き込まれたジョブブックマークの暗号化が可能になります。

9. ナビゲーションペインで [Connections (接続)] を選択します。

   a. [接続の追加] を選択して、ETL ジョブのターゲットとなる Java Database Connectivity (JDBC) データ・ストアへの接続を作成します。

   b. Secure Sockets Layer (SSL) 暗号化が使用されるようにするには、[Require SSL connection (SSL接続を要求)] を選択し、接続をテストします。

10. ナビゲーションペインで ジョブを選択します。

   a. [ジョブを追加] を選択して、データを変換するジョブを作成します。

   b. ジョブ定義で、作成したセキュリティ構成を選択します。

11 AWS Glue コンソールで、オンデマンドでジョブを実行します。ジョブによって書き込まれた Amazon S3 データ、ジョブによって書き込まれた CloudWatch Logs、およびジョブブックマークがすべて暗号化されていることを確認します。

AWS Glue のための開発用ネットワークの設定

AWS Glue で抽出、変換、ロード (ETL) スクリプトを実行するには、開発エンドポイントを使用してスクリプトを開発し、テストすることができます。開発エンドポイントは、AWS Glue バージョン 2.0 ジョブでの使用がサポートされていません。バージョン 2.0 以降では、Jupyter Notebook といずれかの AWS Glue カーネルを使用する開発方法が推奨されています。詳しくは、「the section called "AWS Glue のインタラクティブセッションの開始方法"」を参照してください。

開発エンドポイント用にネットワークを設定する

開発エンドポイントを設定するときは、Virtual Private Cloud (VPC)、サブネット、およびセキュリティグループを指定します。

> **Note**
>
> AWS Glue の DNS 環境を必ず設定してください。詳しくは、「VPC での DNS のセットアップ」を参照してください。

AWS Glue による必要なリソースに対するアクセスを許可するには、サブネットのルートテーブルに行を追加して、Amazon S3 のプレフィックスリストを VPC エンドポイントに関連付けます。プレフィックスリスト ID は、VPC からのトラフィックが VPC エンドポイント経由で AWS のサービスにアクセスできるようにする、アウトバウンドセキュリティグループのルールを作成するために必要です。この開発エンドポイントに関連付けられているノートブックサーバーへの接続をローカルマシンから簡単に行うには、ルートテーブルに行を追加してインターネットゲートウェイ ID を追加します。詳細については、「VPC エンドポイント」を参照してください。サブネットのルートテーブルを更新すると次の表のようになります。

デスティネーション	ターゲット		
10.0.0.0/16	local		
Amazon S3 の pl-id	vpce-id		

デスティネーション	ターゲット		
0.0.0.0/0	igw-xxxx		

AWS Glue がコンポーネント間で通信できるようにするには、すべての TCP ポートに対して自己参照のインバウンドルールを持つセキュリティグループを指定します。自己参照ルールを作成することで、ソースを VPC 内の同じセキュリティグループに制限することができ、ネットワーク全体には公開されません。VPC のデフォルトのセキュリティグループには、すでに ALL Traffic (すべてのトラフィック) の自己参照インバウンドルールがある場合があります。

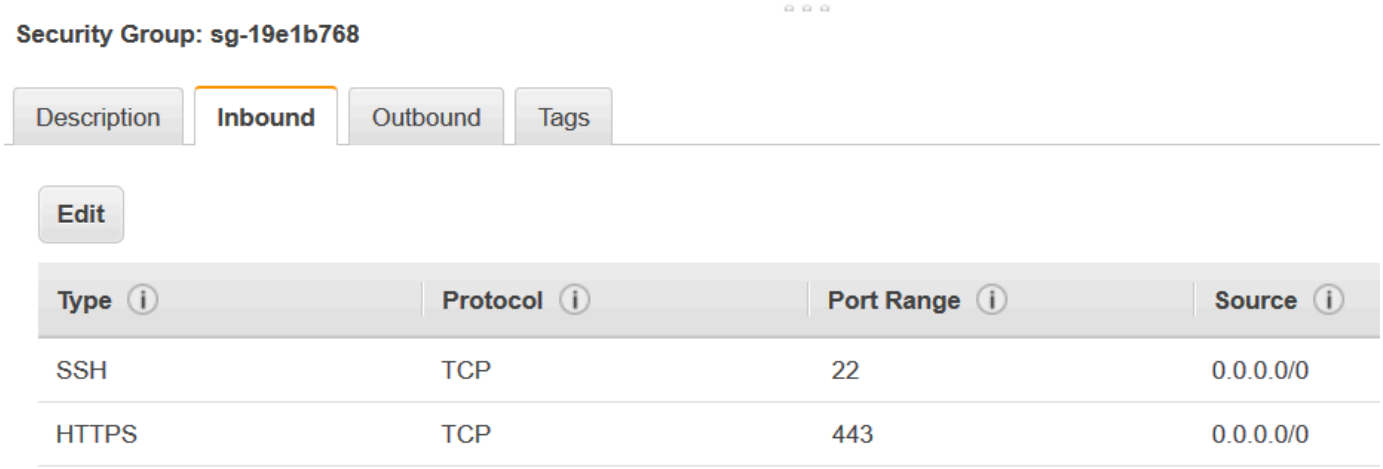
セキュリティグループを設定するには

1. AWS Management Console にサインインし、Amazon EC2 コンソール (https://console.aws.amazon.com/ec2/) を開きます。
2. 左のナビゲーションペインで セキュリティグループを選択します。
3. リストから既存のセキュリティグループを選択するか、[Create Security Group] (セキュリティグループの作成) を選択して、開発エンドポイントで使用します。
4. セキュリティグループペインで、[Inbound] (インバウンド) タブに移動します。
5. 自己参照ルールを追加して、AWS Glue コンポーネントが通信できるようにします。具体的には、[Type (タイプ)] `All TCP`、[Protocol (プロトコル)] は `TCP`、[Port Range (ポート範囲)] にはすべてのポートが含まれ、[Source (ソース)] は [Group ID (グループ ID)] と同じセキュリティグループ名であるというルールを追加または確認します。

   インバウンドルールは次のようになります。

タイプ	プロトコル	ポート範囲	ソース
すべての TCP	TCP	0 ~ 65535	*security-group*

   次に、自己参照インバウンドルールの例を示します。



6. アウトバウンドトラフィックのルールも追加します。すべてのポートへのアウトバウンドトラフィックを開くか、または [Type (タイプ)] `All TCP`、[Protocol (プロトコル)] は `TCP`、[Port Range (ポート範囲)] にすべてのポートが含まれ、[Source (ソース)] は [Group ID (グループ ID)] と同じセキュリティグループ名の自己参照ルールを作成します。

   アウトバウンドルールは、次のいずれかのルールのようになります。

タイプ	プロトコル	ポート範囲	デスティネーション
すべての TCP	TCP	0 ~ 65535	*security-group*
すべてのトラフィック	すべて	すべて	0.0.0.0/0

ノートブックサーバー用の Amazon EC2 のセットアップ

開発エンドポイントでは、Jupyter Notebook で ETL スクリプトをテストするためのノートブックサーバーを作成できます。ノートブックとの通信を可能にするには、HTTPS (ポート 443) と SSH (ポート 22) の両方のインバウンドルールを持つセキュリティグループを指定します。ルールのソースが 0.0.0.0/0 か、ノートブックに接続しているマシンの IP アドレスであることを確認してください。

セキュリティグループを設定するには

1. AWS Management Console にサインインし、Amazon EC2 コンソール (https://console.aws.amazon.com/ec2/) を開きます。

2. 左のナビゲーションペインで セキュリティグループを選択します。
3. リストから既存のセキュリティグループを選択するか、[Create Security Group] (セキュリティグループの作成) を選択して、ノートブックサーバーで使用します。開発エンドポイントに関連付けられているセキュリティグループは、ノートブックサーバーの作成にも使用されます。
4. セキュリティグループペインで、[Inbound] (インバウンド) タブに移動します。
5. 次のようなインバウンドルールを追加します。

タイプ	プロトコル	ポート範囲	ソース
SSH	TCP	22	0.0.0.0/0
HTTPS	TCP	443	0.0.0.0/0

次に、セキュリティグループのインバウンドルールの例を示します。

Security Group: sg-19e1b768

Description | **Inbound** | Outbound | Tags

Edit

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

AWS Glue でのデータ検出とカタログ化

AWS Glue Data Catalog は、組織のデータセットに関するメタデータを保存する一元化されたリポジトリです。データソースの場所、スキーマ、およびランタイムメトリクスへのインデックスとして機能します。メタデータはメタデータテーブルに保存され、そこでは各テーブルが 1 つのデータストアを表します。

データソースを自動的にスキャンしてメタデータを抽出するクローラーを使用してデータカタログに入力できます。クローラーは、内部 (AWS ベース) および AWS 外部のデータソースに接続できます。

サポートされるデータソースについては、「クロールでサポートされているデータソース」を参照してください。

特定の要件に応じてテーブル構造、スキーマ、パーティション構造を定義することで、データカタログにテーブルを手動で作成することもできます。

メタデータテーブルの手動作成について詳しくは、「メタデータの手動定義」を参照してください。

データカタログ内の情報を使用して、ETL ジョブを作成し、モニタリングできます。データカタログは他の AWS 分析サービスと統合され、データソースの統合ビューを提供するため、データの管理と分析が容易になります。

- Amazon Athena – SQL を使用して、Amazon S3 データのデータカタログにテーブルメタデータを保存し、クエリします。
- AWS Lake Formation – きめ細かなデータアクセスポリシーを一元的に定義および管理し、データアクセスを監査します。
- Amazon EMR – ビッグデータ処理のためにデータカタログで定義されたデータソースにアクセスします。
- Amazon SageMaker AI – 機械学習モデルを迅速かつ確実に構築、トレーニング、デプロイします。

データカタログの主な機能

データカタログの主な側面を次に示します。

メタデータリポジトリ

データカタログは中央メタデータリポジトリとして機能し、データソースの場所、スキーマ、プロパティに関する情報を保存します。このメタデータは、従来のリレーショナルデータベースカタログと同様に、データベースとテーブルにまとめられます。

自動データ検出可能性

AWS Glue クローラー では、新規または更新されたデータソースを自動的に検出してカタログ化できるため、手動メタデータ管理のオーバーヘッドが軽減され、データカタログが最新の状態を維持できます。データソースをカタログ化することで、データカタログでは、ユーザーやアプリケーションが組織内で利用可能なデータアセットを簡単に検出して理解できるようになり、データの再利用とコラボレーションが促進されます。

データカタログは、Amazon S3、Amazon RDS、Amazon Redshift、Apache Hive など、さまざまなデータソースをサポートしています。AWS Glue クローラー を使用して、これらのソースからメタデータを自動的に推測して保存できます。

詳細については、「クローラーを使用したデータカタログへの入力 」を参照してください。

スキーマ管理

データカタログは、スキーマの推論、進化、バージョニングなど、データソースのスキーマを自動的にキャプチャして管理します。AWS Glue ETL ジョブを使用して、データカタログ内のスキーマとパーティションを更新できます。

テーブル最適化

Amazon Athena、Amazon EMR、AWS Glue ETL ジョブなどの AWS 分析サービスによる読み取りパフォーマンスを向上させるために、データカタログは、データカタログ内の Iceberg テーブル用にマネージド圧縮 (小さな Amazon S3 オブジェクトを圧縮してより大きなオブジェクトにコンパクト化するプロセス) を提供しています。AWS Glue コンソール、AWS Lake Formation コンソール、AWS CLI API、または AWS API を使用して、データカタログ内の個々の Iceberg テーブルの圧縮を有効または無効にすることができます。

詳細については、「Iceberg テーブルの最適化」を参照してください。

列統計

追加のデータパイプラインを設定することなく、Parquet、ORC、JSON、ION、CSV、XML などのデータ形式でデータカタログテーブルの列レベルの統計を計算できます。列統計は、列内の値に関するインサイトを得ることで、データプロファイルを理解するのに役立ちます。データカ

タログは、最小値、最大値、null 値の合計、個別の値の合計、値の平均長、true 値の合計出現数などの列の値の統計の生成をサポートしています。

詳細については、「列統計を使用したクエリのパフォーマンスの最適化」を参照してください。

データリネージュ

データカタログは、データに対して実行された変換とオペレーションの記録を維持し、データ系統情報を提供します。この系統情報は、データ出所の監査、コンプライアンス、理解に役立ちます。

AWS の他のサービスとの統合

データカタログは、AWS Lake Formation、Amazon Athena、Amazon Redshift Spectrum、Amazon EMR などの他の AWS サービスとシームレスに統合されます。この統合により、1 つの一貫性のあるメタデータレイヤーを使用して、さまざまなデータストアのデータをクエリおよび分析できます。

セキュリティとアクセスコントロール

AWS Glue は AWS Lake Formation と統合して、データカタログリソースのきめ細かなアクセスコントロールをサポートすることで、組織のポリシーと要件に基づいてアクセス許可を管理し、データアセットに安全にアクセスできます。AWS Glue は AWS Key Management Service (AWS KMS) と統合して、データカタログに保存されているメタデータを暗号化します。

トピック

- AWS Glue データカタログの入力
- トランザクションテーブルへのデータ入力と管理
- データカタログの管理
- データカタログにアクセスする
- AWS Glue データカタログのベストプラクティス
- Amazon CloudWatch での Data Catalog 使用状況メトリクスのモニタリング
- AWS Glue スキーマレジストリ

AWS Glue データカタログの入力

次の方法を使用して AWS Glue Data Catalog にデータを入力できます。

- AWS Glue クローラー – AWS Glue クローラー は、データベース、データレイク、ストリーミングデータなどのデータソースを自動的に検出してカタログ化できます。クローラーは、さまざまなデータソースのメタデータを自動的に検出して推測できるため、データカタログにデータを入力する最も一般的な推奨方法です。
- メタデータの手動追加 – AWS Glue コンソール、Lake Formation コンソール、AWS CLI API、または AWS Glue API を使用して、データベース、テーブル、および接続の詳細を手動で定義し、これらをデータカタログに追加できます。手動入力は、クロールできないデータソースをカタログ化する場合に便利です。
- 他の AWS サービスとの統合 — AWS Lake Formation や Amazon Athena などのサービスからのメタデータをデータカタログに入力できます。これらのサービスは、データカタログでデータソースを検出して登録できます。
- 既存のメタデータリポジトリからの入力 – Apache Hive Metastore などの既存のメタデータストアがある場合は、AWS Glue を使用してそのメタデータをデータカタログにインポートできます。詳細については、GitHub で「Migration between the Hive Metastore and the AWS Glue Data Catalog」を参照してください。

トピック

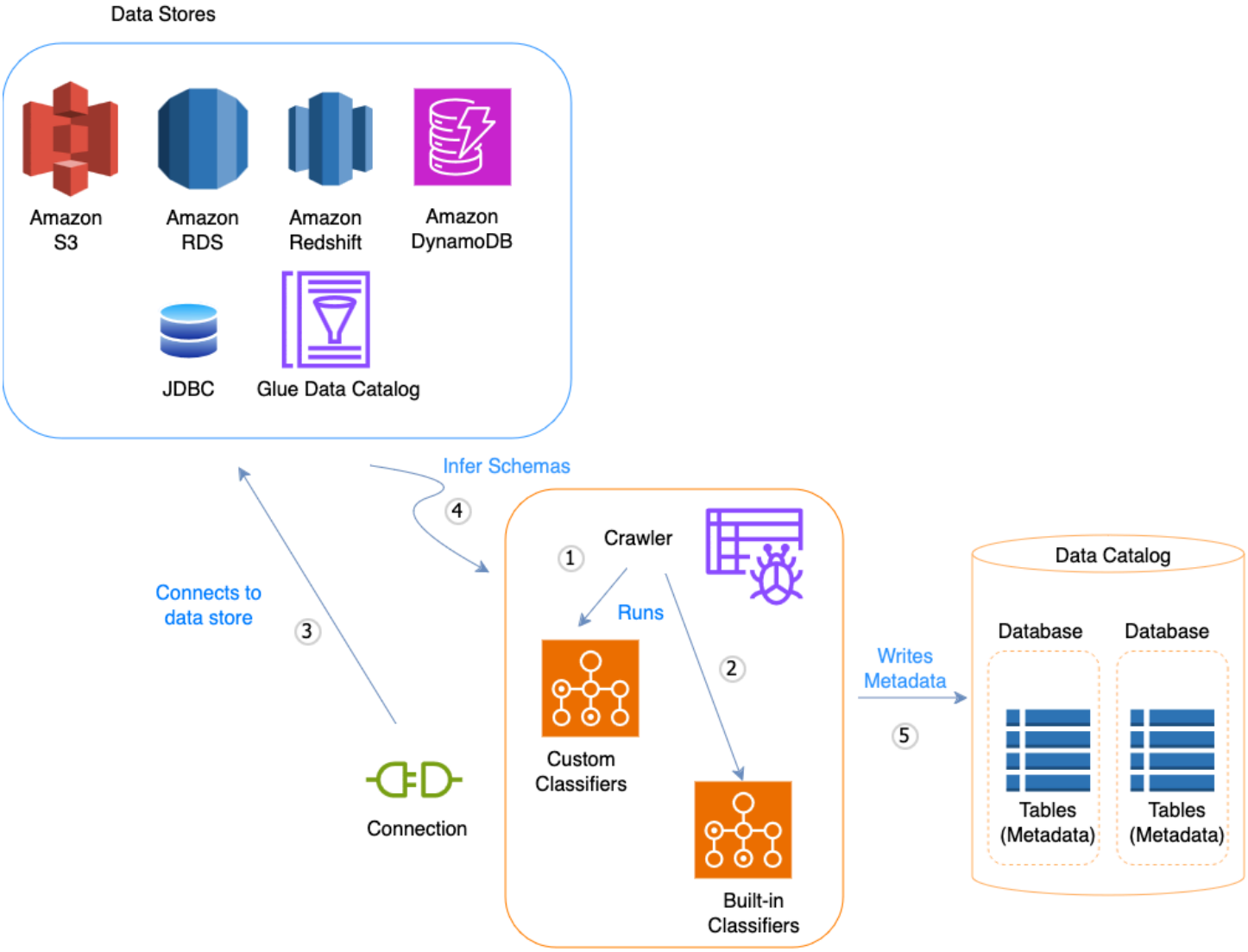
- クローラーを使用したデータカタログへの入力
- メタデータの手動定義
- 他の AWS のサービスとの統合
- データカタログの設定

クローラーを使用したデータカタログへの入力

AWS Glue クローラー を使用して、AWS Glue Data Catalog にデータベースとテーブルを入力できます。これは、AWS Glue ユーザーが最もよく使用する基本的な方法です。クローラーは 1 回の実行で複数のデータストアをクロールできます。完了すると、クローラーはデータカタログで 1 つ以上のテーブルを作成または更新します。AWS Glue で定義した抽出、変換、ロード (ETL) ジョブは、これらのデータカタログテーブルをソースおよびターゲットとして使用します。ETL ジョブは、ソースおよびターゲットのデータカタログテーブルで指定されているデータストアに対して読み取りと書き込みを行います。

ワークフロー

次のワークフロー図は、AWS Glue クローラーがデータストアや他の要素とやり取りしてデータカタログに入力する方法を示しています。



クローラーが AWS Glue Data Catalog に入力する一般的なワークフローを以下に示します。

1. クローラーが選択した任意のカスタム分類子を実行し、データの形式とスキーマを推論します。カスタム分類子のコードを提供すると、指定した順序で実行されます。

   データの構造を正常に認識した最初のカスタム分類子がスキーマを作成するために使用されます。リスト内の下位のカスタム分類子はスキップされます。

2. カスタム分類子と一致するデータのスキーマがない場合は、組み込み分類子がデータのスキーマを認識します。組み込み分類子の例に、JSON を認識する分類子があります。

3. クローラーがデータストアに接続します。一部のデータストアでは、クローラーがアクセスするために接続プロパティを必要とします。
4. データの推測されたスキーマが作成されます。
5. クローラーはデータカタログにメタデータを書き込みます。テーブル定義にはデータストア内のデータに関するメタデータが含まれています。テーブルは、Data Catalog でテーブルのコンテナとなるデータベースに書き込まれます。テーブルの属性には分類が含まれます。これは、テーブルのスキーマを推測した分類子により作成されるラベルです。

トピック

- クローラーの仕組み
- クローラーは、どのようにパーティションを作成するタイミングを判断していますか?
- クロールでサポートされているデータソース
- クローラーの前提条件
- 分類子の定義と管理
- クローラーの設定
- クローラのスケジュール
- クローラーの結果と詳細の表示
- クローラーの動作のカスタマイズ
- チュートリアル: AWS Glue クローラの追加

クローラーの仕組み

クローラーを実行すると、クローラーは以下のアクションを使用してデータストアを調査します。

- ローデータの形式、スキーマ、および関連プロパティを確認するためにデータを分類する – カスタム分類子を作成して分類の結果を設定できます。
- データをテーブルまたはパーティションにグループ化する – データはクローラーのヒューリスティックに基づいてグループ化されます。
- メタデータをデータカタログに書き込む – クローラーでテーブルやパーティションを追加、更新、削除する方法を設定できます。

クローラーを定義する場合、データの形式を評価してスキーマを推測する分類子を 1 つ以上選択します。クローラーを実行すると、リストで最初にデータストアの認識に成功した分類子を使用して

テーブルのスキーマが作成されます。組み込み分類子を使用するか、独自に定義することができます。カスタム分類子は、クローラーを定義する前に別のオペレーションで定義します。AWS Glue には、JSON、CSV、Apache Avro などの形式の共通ファイルからスキーマを推論するための組み込み分類子が用意されています。AWS Glue の組み込み分類子の最新のリストについては、「組み込み分類子」を参照してください。

クローラーで作成するメタデータテーブルは、クローラーの定義時にデータベースに含まれます。クローラーがデータベースを指定しない場合、テーブルはデフォルトのデータベースに配置されます。さらに、各テーブルには、最初にデータストアの認識に成功した分類子により入力された分類子の列があります。

クロールするファイルが圧縮されている場合、クローラーはダウンロードして処理する必要があります。クローラーを実行すると、ファイルを調査して形式と圧縮タイプを判定し、これらのプロパティをデータカタログに書き込みます。一部のファイル形式 (Apache Parquet など) では、ファイルの書き込み時にファイルの一部を圧縮できます。これらのファイルでは、圧縮されたデータはファイルの内部コンポーネントであり、AWS Glue はテーブルをデータカタログ内に書き込むときに `compressionType` プロパティに値を入力しません。一方、ファイル全体を圧縮アルゴリズム (gzip など) で圧縮する場合は、テーブルをデータカタログ内に書き込むときに `compressionType` プロパティが事前設定されます。

クローラーは、作成するテーブルの名前を生成します。AWS Glue Data Catalogに保存されるテーブルの名前は、以下のルールに従います。

- 英数字とアンダースコア (_) のみを使用できます。
- カスタムプレフィックスは 64 文字より長くすることはできません。
- 名前の長さは最大 128 文字までです。クローラーは、生成した名前が制限内に収まるように切り捨てます。
- 重複するテーブル名が発生した場合、クローラーは名前にハッシュ文字列のサフィックスを追加します。

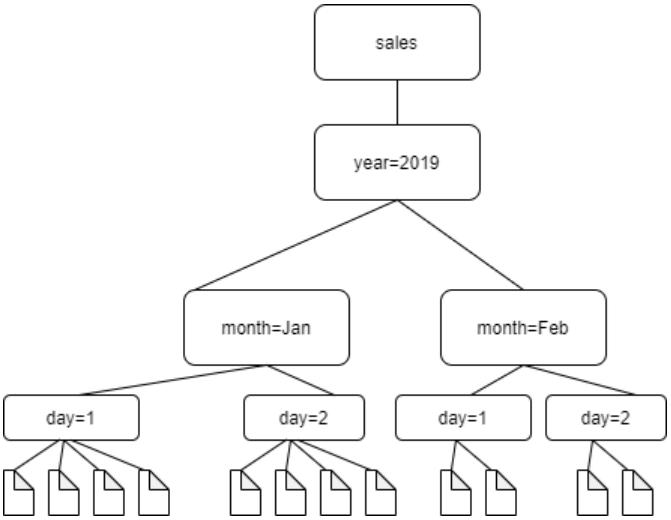
クローラーが複数回実行される場合 (おそらくスケジュールに基づいて)、データストア内の新規または変更されたファイルやテーブルが検索されます。クローラーの出力には、前回の実行以降に検索された、新しいテーブルとパーティションが含まれています。

クローラーは、どのようにパーティションを作成するタイミングを判断していますか?

AWS Glue クローラーは、Amazon S3 データをスキャンしてバケット内に複数のフォルダを検出すると、フォルダ構造のテーブルのルート、およびどのフォルダがテーブルのパーティションであるか

を確認します。テーブルの名前は Amazon S3 プレフィックスまたはフォルダ名に基づいています。クロールするフォルダレベルを指すインクルードパスを指定します。フォルダレベルの大半のスキーマが類似している場合、クローラーはテーブルを別個に作成せずに、テーブルのパーティションを作成します。クローラーで別個のテーブルを作成するには、クローラーを定義するときに各テーブルのルートフォルダを別個のデータストアとして追加します。

例えば、次の Amazon S3 フォルダ構造を想定します。



4 つの最下位レベルのフォルダへのパスは次のとおりです。

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

クローラーターゲットが Sales に設定されており、day=*n* フォルダ内のすべてのファイルが同じ形式 (暗号化されていない JSON など) で、同じか非常によく似たスキーマを持っているとします。クローラーは、パーティションキー year、month、および day で、4 つのパーティションを持つ単一のテーブルを作成します。

次の例では、次の Amazon S3 構造を想定します。

```
s3://bucket01/folder1/table1/partition1/file.txt
s3://bucket01/folder1/table1/partition2/file.txt
s3://bucket01/folder1/table1/partition3/file.txt
s3://bucket01/folder1/table2/partition4/file.txt
s3://bucket01/folder1/table2/partition5/file.txt
```

table1 と table2 のファイルのスキーマが類似しており、クローラーにインクルードパス s3://bucket01/folder1/ で定義されているデータストアが 1 つの場合、クローラーは 2 つのパーティションキー列を持つ 1 つのテーブルを作成します。最初のパーティションキー列には table1 および table2 が含まれ、2 番目のパーティションキー列には、table1 パーティションに対して partition1 から partition3 まで、および table2 パーティションに対して partition4 と partition5 が含まれます。2 つの個別のテーブルを作成するには、2 つのデータストアを持つクローラーを定義します。この例では、最初のインクルードパスを s3://bucket01/folder1/table1/ として定義し、2 番目を s3://bucket01/folder1/table2 として定義します。

> **Note**
>
> Amazon Athenaの場合、各テーブルは Amazon S3 プレフィックス (すべてのオブジェクトを含む) に対応します。オブジェクト別にスキーマが異なる場合、Athena では同じプレフィックス内の異なるオブジェクトを別個のテーブルとして認識しません。これは、クローラーで同じ Amazon S3 プレフィックスから複数のテーブルを作成する場合に発生することがあります。そのため、Athena のクエリで結果が何も返されない場合があります。Athena でテーブルを適切に認識してクエリを実行するには、Amazon S3 フォルダ構造内の異なるテーブルスキーマごとに別個のインクルードパスを持つクローラーを作成します。詳細については、「Athena で AWS Glue を使用するときのベストプラクティス」およびこちらの「AWS ナレッジセンターの記事」を参照してください。

クロールでサポートされているデータソース

クローラーは、次に示すファイルベースおよびテーブルベースのデータストアをクロールできます。

クローラーが使用するアクセスタイプ	データストア
ネイティブクライアント	• Amazon Simple Storage Service (Amazon S3) • Amazon DynamoDB • Delta Lake 2.0.x • Apache Iceberg 1.5 • Apache Hudi 0.14
JDBC	Amazon Redshift

クローラーが使用するアクセスタイプ	データストア
	Snowflake Amazon Relational Database Service (Amazon RDS) 内、または Amazon RDS の外部: • Amazon Aurora • MariaDB • Microsoft SQL Server • MySQL • Oracle • PostgreSQL
MongoDB クライアント	• MongoDB • MongoDB Atlas • Amazon DocumentDB (MongoDB 互換性)

> **Note**
>
> 現在、AWS Glue ではデータストリームのクローラーがサポートされていません。

JDBC、MongoDB、MongoDB Atlas、Amazon DocumentDB (MongoDB 互換) データストアでは、クローラーがデータストアに接続するときに使用可能な AWS Glue 接続 を指定する必要があります。Amazon S3 の場合、オプションで Network タイプの接続を指定できます。接続とは、認証情報、URL、Amazon Virtual Private Cloud 情報などの接続情報を保存するデータカタログオブジェクトです。詳細については、「データへの接続」を参照してください。

クローラーがサポートするドライバーのバージョンは次のとおりです。

製品	クローラーがサポートするドライバー
PostgreSQL	42.2.1

製品	クローラーがサポートするドライバー
Amazon Aurora	ネイティブクローラードライバーと同じ
MariaDB	8.0.13
Microsoft SQL Server	6.1.0
MySQL	8.0.13
Oracle	11.2.2
Amazon Redshift	4.1
Snowflake	3.13.20
MongoDB	4.7.2
MongoDB Atlas	4.7.2

次に、さまざまなデータストアに関する注意事項を示します。

Amazon S3

自分のアカウントのパスをクロールするか、または別のアカウントのパスをクロールするかを選択できます。フォルダにあるすべての Amazon S3 ファイルが同じスキーマを持つ場合、クローラーはテーブルを 1 つ作成します。また、Amazon S3 オブジェクトが分割されている場合、メタデータテーブルは 1 つしか作成されず、そのテーブルのパーティション情報がデータカタログに追加されます。

Amazon S3 と Amazon DynamoDB

クローラーは、AWS Identity and Access Management (IAM) ロールをアクセス許可のために使用して、データストアにアクセスします。クローラーに渡すロールは、クロールされる Amazon S3 パスと Amazon DynamoDB テーブルにアクセスするためのアクセス許可を持っている必要があります。

Amazon DynamoDB

AWS Glue コンソールを使用してクローラーを定義するとき、1 つのDynamoDB テーブルを指定します。AWS Glue API を使用している場合、テーブルのリストを指定できます。クローラーの実行時間を短縮するために、データの小さなサンプルのみをクロールするように選択できます。

Delta Lake

各 Delta Lake データストアで、Delta テーブルの作成方法を指定します。

- [Create Native tables] (ネイティブテーブルの作成): Delta トランザクションログの直接クエリをサポートするクエリエンジンとの統合を可能にします。詳細については、「Delta Lake テーブルのクエリを実行する」を参照してください。
- [Create Symlink tables] (Symlink テーブルの作成): 指定された設定パラメータに基づいて、マニフェストファイルを分割キーで分割して `_symlink_manifest` フォルダを作成します。

Iceberg

Iceberg データストアごとに、Iceberg テーブルのメタデータを含む Amazon S3 パスを指定します。クローラーは Iceberg テーブルのメタデータを検出すると、それをデータカタログに登録します。クローラーがテーブルを更新し続けるように、スケジュールを設定できます。

データストアに次のパラメータを定義できます。

- Exclusions: 特定のフォルダをスキップできます。
- Maximum Traversal Depth: Amazon S3 バケットでクローラーがクロールできる深度の制限を設定します。デフォルトの最大トラバーサル深度は 10 で、設定できる最大深度は 20 です。

Hudi

Hudi データストアごとに、Hudi テーブルのメタデータを含む Amazon S3 パスを指定します。クローラーは Hudi テーブルのメタデータを検出すると、それをデータカタログに登録します。クローラーがテーブルを更新し続けるように、スケジュールを設定できます。

データストアに次のパラメータを定義できます。

- Exclusions: 特定のフォルダをスキップできます。
- Maximum Traversal Depth: Amazon S3 バケットでクローラーがクロールできる深度の制限を設定します。デフォルトの最大トラバーサル深度は 10 で、設定できる最大深度は 20 です。

> **Note**
>
> Hudi 0.13.1 およびタイムスタンプ型と互換性がないため、論理型として `millis` を含むタイムスタンプ列は、`bigint` として解釈されます。解決策は、今後の Hudi リリースで提供される可能性があります。

Hudi テーブルは次のように分類され、それぞれが固有の意味を持っています。

- 書き込み時コピー (CoW): データは列形式 (Parquet) で保存され、更新ごとに書き込み中にファイルの新しいバージョンが作成されます。
- 読み取り時マージ (MoR): データは、列形式 (Parquet) と行形式 (Avro) の組み合わせを使用して保存されます。更新は、行形式の差分ファイルに記録され、必要に応じて圧縮されて、新しいバージョンの列形式のファイルが作成されます。

CoW データセットでは、レコードが更新されるたびに、そのレコードを含むファイルが更新された値で書き換えられます。MoR データセットでは、更新があるたびに、Hudi によって変更されたレコードの行のみが書き込まれます。MoR は、読み取りが少なく書き込みまたは変更が多いワークロードに適しています。CoW は、頻繁に変更されないデータの読み取りが多いワークロードに適しています。

Hudi にはデータにアクセスするためのクエリタイプが 3 種類用意されています。

- スナップショットクエリ: 指定されたコミットまたは圧縮アクションの時点で最新のテーブルスナップショットを参照するクエリです。MoR テーブルの場合、スナップショットクエリは、クエリ時に最新のファイルスライスのベースファイルとデルタファイルをマージして、テーブルの最新の状態を提示します。
- 増分クエリ: クエリは、指定されたコミット/圧縮以降にテーブルに書き込まれた新しいデータのみを参照します。これにより、増分データパイプラインを有効にするための変更ストリームが効果的に提供されます。
- 読み取り最適化クエリ: MoR テーブルの場合、クエリは圧縮された最新のデータを参照します。CoW テーブルの場合、クエリはコミットされた最新のデータを参照します。

書き込み時コピー (CoW) テーブルの場合、クローラーは ReadOptimized serde `org.apache.hudi.hadoop.HoodieParquetInputFormat` を使用して、データカタログに 1 つのテーブルを作成します。

読み取り時マージ (MoR) テーブルの場合、クローラーはデータカタログ内の同じテーブルの場所に次の 2 つのテーブルを作成します。

- サフィックス `_ro` を持ち、ReadOptimized serde `org.apache.hudi.hadoop.HoodieParquetInputFormat` を使用するテーブル。
- サフィックス `_rt` を持ち、スナップショットクエリを可能にする RealTime Serde `org.apache.hudi.hadoop.realtime.HoodieParquetRealtimeInputFormat` を使用するテーブル。

MongoDB と Amazon DocumentDB (MongoDB 互換)

MongoDB バージョン 3.2 以降がサポートされています。クローラーの実行時間を短縮するために、データの小さなサンプルのみをクロールするように選択できます。

リレーショナルデータベース

データベースのユーザー名とパスワードを使用して認証します。データベースエンジンのタイプに応じて、どのオブジェクト (データベース、スキーマ、テーブルなど) をクロールするかを選択できます。

Snowflake

Snowflake JDBC クローラーは、テーブル、外部テーブル、ビュー、マテリアライズドビューのクローリングをサポートしています。マテリアライズドビュー定義は入力されません。

Snowflake 外部テーブルについては、Amazon S3 の場所を指している場合のみクローラーによってクローリングされます。クローラーは、テーブルスキーマの他に、Amazon S3 の場所、ファイル形式、および出力をデータカタログテーブルのテーブルパラメータとしてクローリングします。分割された外部テーブルのパーティション情報は入力されないことに注意してください。

Snowflake クローラーを使用して作成されたデータカタログテーブルでは、ETL は現在サポートされていません。

クローラーの前提条件

クローラーは、定義する時に指定する AWS Identity and Access Management (IAM) ロール のアクセス許可を取得します。この IAM ロールには、データストアからデータを抽出してデータカタログに書き込むためのアクセス許可が必要です。AWS Glue コンソールには、AWS Glue プリンシパルサービスの信頼ポリシーがアタッチされた IAM ロールだけがリスト表示されています。コンソールから、クローラーがアクセスするAmazon S3 データストアにアクセスするための IAM ポリシーを持つ IAM ロールを作成できます。AWS Glue のロールの指定の詳細については、「AWS Glue のアイデンティティベースのポリシー」を参照してください。

> **Note**
>
> Delta Lake データストアをクロールする場合、Amazon S3 の場所に対する読み取り/書き込み権限が必要です。

クローラーには、ロールを作成して次のポリシーをアタッチできるようになりました。

- AWSGlueServiceRole AWS 管理ポリシー。データカタログに必要なアクセス許可を付与します。
- データソースに対するアクセス許可を付与するインラインポリシー。
- ロールに対する iam:PassRole アクセス許可を付与するインラインポリシー。

AWS Glue コンソールクローラーウィザードにロールを作成させるのが、より迅速なアプローチです。これによって作成されるロールは、クローラー専用であり、AWSGlueServiceRole AWS 管理ポリシーと、指定したデータソースに必要なインラインポリシーを含んでいます。

クローラーに既存のロールを指定する場合は、そのロールに AWSGlueServiceRole ポリシーまたは同等のポリシー (またはこのポリシーのスコープダウンバージョン) と、必要なインラインポリシーが含まれてることを確認します。例えば、Amazon S3 データストアの場合、インラインポリシーは少なくとも次のようになります。

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

Amazon DynamoDB データストアの場合、ポリシーは、少なくとも次のようになります。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:DescribeTable",
        "dynamodb:Scan"
      ],
```

```
      "Resource": [
        "arn:aws:dynamodb:region:account-id:table/table-name*"
      ]
    }
  ]
}
```

また、クローラーが AWS Key Management Service (AWS KMS) 暗号化 Amazon S3 データを読み取る場合、IAM ロールには AWS KMS キーの復号化のアクセス許可が必要です。詳細については、「ステップ 2: AWS Glue 用の IAM ロールを作成する」を参照してください。

分類子の定義と管理

分類子はデータをデータストアに読み取ります。データの形式を認識すると、スキーマが生成されます。分類子も、形式の認識がどれほど確実かを示す確信度数を返します。

AWS Glue では一連の組み込み分類子が用意されていますが、カスタム分類子を作成することもできます。AWS Glue は、クローラ定義で指定した順序で、カスタム分類子を最初に呼び出します。カスタム分類子から返された結果に応じて、AWS Glue が組み込みの分類子を呼び出す場合もあります。処理中に分類子が `certainty=1.0` を返した場合、正しいスキーマを 100% 確実に作成できることを示しています。次に、AWS Glue はその分類子の出力を使用します。

分類子が `certainty=1.0` を返さない場合、AWS Glue は最も高い確実性を持つ分類子の出力を使用します。どの分類子からも `0.0` 以上の確実性が返されない場合、AWS Glue は `UNKNOWN` のデフォルト分類文字列を返します。

分類子を使用するタイミング

データストアをクロールして AWS Glue Data Catalog でメタデータテーブルを定義する際に分類子を使用します。順序付けられた一連の分類子を使用してクローラーをセットアップできます。クローラーが分類子を呼び出す際、分類子はデータが認識されるかどうかを判断します。分類子がデータを認識できない、または 100% 確実ではない場合、クローラーはリストにある次の分類子を呼び出して、データを認識できるかどうか判断します。

AWS Glue コンソールを使用して分類子を作成する方法の詳細については、「AWS Glue コンソールを使用した分類子の作成」を参照してください。

カスタム分類子

分類子の出力には、ファイルの分類や形式 (例えば、`json`)、およびファイルのスキーマを示す文字列が含まれます。カスタム分類子の場合は、分類子のタイプに基づいてスキーマを作成するための口

ジックを定義します。分類子のタイプには、grok パターン、XML タグ、および JSON パスに基づくスキーマの定義が含まれています。

分類子の定義を変更すると、変更前の分類子を使用してクロールしたデータは再分類されません。クローラーは、以前にクロールしたデータを追跡します。新しいデータは、更新された分類子で分類されるため、スキーマが更新される場合があります。データのスキーマが更新された場合は、クローラーの実行時に分類子を更新してスキーマの変更を反映してください。データを再分類して不正な分類子を修正するには、更新された分類子を使用して新しいクローラーを作成します。

AWS Glue でカスタム分類子を作成する方法については、「さまざまなデータ形式のカスタム分類子の記述」を参照してください。

> **Note**
>
> 組み込み分類子のいずれかでデータ形式が認識される場合、カスタム分類子を作成する必要はありません。

組み込み分類子

AWS Glue は、JSON、CSV、ウェブログ、および多くのデータベースシステムを含む、さまざまな形式の組み込み分類子を提供します。

AWS Glue が入力データ形式に 100% 確実に適合するカスタム分類子を検出できない場合、次の表に示すような順番で組み込み分類子を呼び出します。組み込み分類子は、形式が一致するか (`certainty=1.0`)、または一致しないか (`certainty=0.0`) どうかを示す結果を返します。`certainty=1.0` を持つ最初の分類子は、Data Catalog での分類文字列とメタデータテーブルのスキーマを提供します。

分類子タイプ	分類文字列	メモ
Apache Avro	`avro`	ファイルの先頭にあるスキーマを読み取って形式を判断します。
Apache ORC	`orc`	ファイルのメタデータを読み取って形式を判断します。
Apache Parquet	`parquet`	ファイルの末尾にあるスキーマを読み取って形式を判断します。

分類子タイプ	分類文字列	メモ
JSON	`json`	ファイルの先頭を読み取って形式を判断します。
バイナリ型 JSON	`bson`	ファイルの先頭を読み取って形式を判断します。
XML	`xml`	ファイルの先頭から読み取って形式を判断します。AWS Glue は、ドキュメントの XML タグに基づいてテーブルスキーマを判定します。 カスタム XML を作成してドキュメントの行を指定するには、「XML カスタム分類子の書き込み」を参照してください。
Amazon Ion	`ion`	ファイルの先頭を読み取って形式を判断します。
Combined Apache ログ	`combined_apache`	grok パターンを通じてログ形式を判断します。
Apache ログ	`apache`	grok パターンを通じてログ形式を判断します。
Linux カーネルログ	`linux_kernel`	grok パターンを通じてログ形式を判断します。
Microsoft ログ	`microsoft_log`	grok パターンを通じてログ形式を判断します。
Ruby ログ	`ruby_logger`	ファイルの先頭を読み取って形式を判断します。
Squid 3.x ログ	`squid`	ファイルの先頭を読み取って形式を判断します。
Redis 監視ログ	`redismonlog`	ファイルの先頭を読み取って形式を判断します。
Redis ログ	`redislog`	ファイルの先頭を読み取って形式を判断します。
CSV	`csv`	カンマ (,)、パイプ (|)、タブ (\t)、セミコロン (;)、および Ctrl-A (\u0001)の区切り記号をチェックします。Ctrl-A は `Start Of Heading` の Unicode 制御文字です。
Amazon Redshift	`redshift`	JDBC 接続を使用してメタデータをインポートします。

分類子タイプ	分類文字列	メモ
MySQL	`mysql`	JDBC 接続を使用してメタデータをインポートします。
PostgreSQL	`postgresql`	JDBC 接続を使用してメタデータをインポートします。
Oracle データベース	`oracle`	JDBC 接続を使用してメタデータをインポートします。
Microsoft SQL Server	`sqlserver`	JDBC 接続を使用してメタデータをインポートします。
Amazon DynamoDB	`dynamodb`	DynamoDB テーブルからデータを読み取ります。

以下の圧縮形式のファイルが分類できます。

- ZIP (単一ファイルのみを含むアーカイブに対応)。Zip は他のサービスでは十分にサポートされていないため注意してください (アーカイブのため)。
- BZIP
- GZIP
- LZ4
- Snappy (標準および Hadoop ネイティブの Snappy 形式の両方をサポート)

組み込み CSV 分類子

組み込み CSV 分類子では、CSV ファイルの内容を解析して AWS Glue テーブルのスキーマを判断します。この分類子は以下の区切り記号を確認します。

- カンマ (,)
- パイプ (|)
- タブ (\t)
- セミコロン (;)
- Ctrl-A (\u0001)

Ctrl-A は `Start Of Heading` の Unicode 制御文字です。

CSV として分類されるためには、テーブルのスキーマに少なくとも 2 つのデータ列と 2 つのデータ行が必要です。CSV 分類子では、いくつかのヒューリスティックを使用して特定のファイルにヘッダーがあるかどうかを判断します。分類子で最初のデータ行にヘッダーを確認できない場合は、列のヘッダーが `col1`、`col2`、`col3` のように表示されます。組み込み CSV 分類子では、以下のファイルの特性を評価することで、ヘッダーを推測するかどうかを決めます。

- ヘッダー候補の各列が STRING データ型として解析されます。
- 最後の列を除き、ヘッダー候補の各列に 150 文字未満のコンテンツがあります。末尾の区切り記号を許可するには、ファイル全体で最後の列を空欄にすることができます。
- ヘッダー候補の各列が、列名に関する AWS Glue `regex` 要件を満たす必要があります。
- ヘッダー行は、データ行と十分に異なっている必要があります。これを判断するには、1 つ以上の行が STRING 型以外として解析されることを確認します。すべての列が STRING 型である場合、最初のデータ行は以降の行と十分に異なっていないため、ヘッダーとして使用できません。

> **Note**
>
> 組み込み CSV 分類子で必要な AWS Glue テーブルが作成されない場合は、以下のいずれかの代替方法を使用できます。
>
> - Data Catalog で列名を変更し、`SchemaChangePolicy` を LOG に設定して、今後のクローラー実行に備えてパーティションの出力設定を `InheritFromTable` に設定します。
> - データを分類するためのカスタム grok 分類子を作成し、必要な列を割り当てます。
> - 組み込み CSV 分類子では、`LazySimpleSerDe` をシリアル化ライブラリとして参照するテーブルを作成します。これは、型の推定に適しています。ただし、CSV データ内に引用符で囲まれた文字列がある場合は、テーブル定義を編集して SerDe ライブラリを `OpenCSVSerDe` に変更します。推定した型を STRING に調整し、`SchemaChangePolicy` を LOG に設定して、将来のクローラ実行に関してパーティションの出力設定を `InheritFromTable` に設定します。SerDe ライブラリの詳細については、Amazon Athena ユーザーガイドの「SerDe リファレンス」を参照してください。

さまざまなデータ形式のカスタム分類子の記述

AWS Glue でデータを分類するためのカスタム分類子を提供できます。grok パターン、XML タグ、JavaScript Object Notation (JSON)、またはカンマ区切り値 (CSV) を使用してカスタム分類子を作成できます。AWS Glue クローラがカスタム分類子を呼び出します。分類子がデータを認識すると、データの分類とスキーマがクローラに返されます。組み込みの分類子にデータが一致しない場合、または、クローラにより作成されたテーブルをカスタマイズする場合は、カスタム分類子を定義する必要があるかもしれません。

AWS Glue コンソールを使用して分類子を作成する方法の詳細については、「AWS Glue コンソールを使用した分類子の作成」を参照してください。

AWS Glue は、組み込みの分類子の前に、指定した順序でカスタム分類子を実行します。クローラがデータに一致する分類子を検出すると、分類の文字列とスキーマが AWS Glue Data Catalog に書き込まれるテーブルの定義で使用されます。

トピック

- Grok カスタム分類子の書き込み
- XML カスタム分類子の書き込み
- JSON カスタム分類子の書き込み
- CSV カスタム分類子の書き込み

Grok カスタム分類子の書き込み

Grok は、一致するパターンによりテキストデータを解析するために使用するツールです。grok パターンは名前のついた一連の正規表現 (regex) で、一度に 1 行のデータごとに一致させるために使用されます。AWS Glue は grok パターンを使用してデータのスキーマを推測します。grok パターンがデータと一致すると、AWS Glue はそのパターンを使用してデータの構造を判断し、フィールドにマッピングします。

AWS Glue は数多くの組み込みパターンを提供します。または、独自のパターンを定義することもできます。組み込みパターンとカスタム分類子の定義にあるカスタムパターンを使用して grok パターンを作成できます。grok パターンをカスタマイズしてカスタムテキストファイル形式を分類できます。

> **Note**
>
> AWS Glue grok カスタム分類子は、AWS Glue Data Catalog で作成されたテーブルの `GrokSerDe` シリアル化ライブラリを使用します。AWS Glue Data Catalog で Amazon

Athena、Amazon EMR、または Redshift Spectrum を使用している場合、GrokSerDe のサポートに関する情報については、これらのサービスに関するドキュメントを確認してください。現在、Amazon EMR と Redshift Spectrum の GrokSerDe を使用して作成されたテーブルのクエリを実行する際に問題が発生することがあります。

grok パターンのコンポーネントの基本的な構文を以下に示します。

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

名付けられた PATTERN に一致するデータは、スキーマの field-name 列にデフォルトのデータ型 string でマッピングされます。必要に応じて、フィールドのデータ型は結果のスキーマの byte、boolean、double、short、int、long、または float にキャストできます。

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

たとえば、num フィールドを int データ型にキャストするには、以下のパターンを使用することができます。

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

パターンは他のパターンで構成できます。たとえば、SYSLOG タイムスタンプのパターンを月、日、時間のパターン (Feb 1 06:25:43 など) で定義できます。このデータの場合、次のパターンを定義できます。

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

Note

grok パターンは、一度に 1 行ずつしか処理できません。複数行のパターンはサポートされていません。また、パターン内の改行はサポートされていません。

grok 分類子のカスタム値

grok 分類子を定義する場合、以下の値を指定しカスタム分類子を作成します。

名前

分類子の名前。

分類

分類されたデータの形式を説明するために記述されたテキスト文字列 (例: `special-logs`)。

Grok パターン

データストアに適用される一連のパターンで、一致があるかどうかを決定します。これらのパターンは AWS Glue の組み込みパターンと定義されたカスタムパターンによるものです。

grok パターンのシンプルな例を次に示します。

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

データが `TIMESTAMP_ISO8601` と一致すると、スキーマの列 `timestamp` が作成されます。動作は、例にある他の名前付きパターンに似ています。

カスタムパターン

独自に定義するオプションのカスタムパターン。これらのパターンは、データを分類する grok パターンにより参照されます。データに適用される grok パターンでこれらのカスタムパターンを参照できます。各カスタムコンポーネントパターンは別々の行にある必要があります。正規表現 (regex) 構文は、パターンを定義するために使用されます。

以下は、カスタムパターンを使用する例です。

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

最初の名前付きカスタムパターンである `CRAWLERLOGLEVEL` は、列挙された文字列の 1 つとデータが一致するときに一致となります。2 番目のカスタムパターン `MESSAGEPREFIX` は、メッセージのプレフィックス文字列との一致を試みます。

AWS Glue は、作成日時、最終更新時間、分類子のバージョンを追跡します。

組み込みパターン

AWS Glue は、カスタム分類子を構築するために使用できる多くの一般的なパターンを提供します。分類子の定義の `grok pattern` に名前付きパターンを追加します。

次のリストは、各パターンの行です。各行で、パターン名の後に定義があります。正規表現 (regex) 構文は、パターンを定義するために使用されます。

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

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

 # Networking
 MAC (?:%{CISCOMAC:UNWANTED}|%{WINDOWSMAC:UNWANTED}|%{COMMONMAC:UNWANTED})
 CISCOMAC (?:(?:[A-Fa-f0-9]{4}\.){2}[A-Fa-f0-9]{4})
 WINDOWSMAC (?:(?:[A-Fa-f0-9]{2}-){5}[A-Fa-f0-9]{2})
 COMMONMAC (?:(?:[A-Fa-f0-9]{2}:){5}[A-Fa-f0-9]{2})
 IPV6 ((([0-9A-Fa-f]{1,4}:){7}([0-9A-Fa-f]{1,4}|:))|(([0-9A-Fa-f]{1,4}:){6}(:[0-9A-
Fa-f]{1,4}|((25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)(\.(25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d))
{3})|:))|(([0-9A-Fa-f]{1,4}:){5}(((:[0-9A-Fa-f]{1,4}){1,2})|:((25[0-5]|2[0-4]\d|1\d
\d|[1-9]?\d)(\.(25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)){3})|:))|(([0-9A-Fa-f]{1,4}:){4}(((:
[0-9A-Fa-f]{1,4}){1,3})|((:[0-9A-Fa-f]{1,4})?:((25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)(\.
(25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)){3}))|:))|(([0-9A-Fa-f]{1,4}:){3}(((:[0-9A-Fa-f]
```

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

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

 # Months: January, Feb, 3, 03, 12, December
 MONTH \b(?:Jan(?:uary)?|Feb(?:ruary)?|Mar(?:ch)?|Apr(?:il)?|May|Jun(?:e)?|Jul(?:y)?|
Aug(?:ust)?|Sep(?:tember)?|Oct(?:ober)?|Nov(?:ember)?|Dec(?:ember)?)\b
 MONTHNUM (?:0?[1-9]|1[0-2])
 MONTHNUM2 (?:0[1-9]|1[0-2])
 MONTHDAY (?:(?:0[1-9])|(?:[12][0-9])|(?:3[01])|[1-9])

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

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

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

 # Shortcuts
 QS %{QUOTEDSTRING:UNWANTED}

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

 MESSAGESLOG %{SYSLOGBASE} %{DATA}
```

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


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

XML カスタム分類子の書き込み

XML は、ファイル内のタグを使用してドキュメントの構造を定義します。XML カスタム分類子で、行の定義に使用されるタグ名を指定できます。

XML 分類子のカスタム分類子値

XML 分類子を定義する場合、AWS Glue に以下の値を指定し分類子を作成します。この分類子の分類フィールドは `xml` に設定してあります。

名前

分類子の名前。

行タグ

XML ドキュメントでテーブル行を定義する XML タグ名、山括弧 < > なし。名前は XML タグ規則に沿って命名する必要があります。

> **Note**
>
> 行データを含む要素は、自動で閉じる空の要素にすることはできません。たとえば、次の空の要素は によって解析されませんAWS Glue。
>
> ```
> <row att1=”xx” att2=”yy” />
> ```

空の要素は次のように記述できます。

```
<row att1=”xx” att2=”yy”> </row>
```

AWS Glue は、作成日時、最終更新時間、分類子のバージョンを追跡します。

たとえば、次の XML ファイルがあるとします。筆者と役職の列のみを含む AWS Glue テーブルを作成するには、AnyCompany として [行タグ] を使用し、AWS Glue コンソールで分類子を作成します。次に、このカスタム分類子を使用するクローラを追加して実行します。

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

JSON カスタム分類子の書き込み

JSON は、データ交換形式です。名前と値のペア、または順序付きの値のリストでデータ構造を定義します。JSON カスタム分類子では、データ構造への JSON パスを指定し、それを使用してテーブルのスキーマを定義できます。

AWS Glue のカスタム分類子の値

JSON 分類子を定義する場合、AWS Glue に以下の値を指定し分類子を作成します。この分類子の分類フィールドは `json` に設定してあります。

名前

分類子の名前。

JSON パス

テーブルスキーマを定義するために使用されるオブジェクトを指す JSON パス。JSON パスは、ドット表記またはブラケット表記で記述できます。以下の演算子がサポートされています。

説明
JSON オブジェクトのルート要素。すべてのパス式はこれで始まります
ワイルドカード文字。JSON パスで名前または数値が必要な箇所でいつでも使用可能。
ドット表記の子。JSON オブジェクトの子フィールドを指定します。
ブラケット表記の子。JSON オブジェクトの子フィールドを指定します。1 つの子フィールドのみを指定できます。
配列インデックス。インデックスにより配列の値を指定します。

AWS Glue は、作成日時、最終更新時間、分類子のバージョンを追跡します。

Example JSON 分類子を使用して配列からレコードをプルする

JSON データがレコードの配列だとします。たとえば、ファイルの最初の数行は次のようになります。

```
[
  {
    "type": "constituency",
    "id": "ocd-division\/country:us\/state:ak",
    "name": "Alaska"
  },
  {
    "type": "constituency",
    "id": "ocd-division\/country:us\/state:al\/cd:1",
```

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

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

組み込み JSON 分類子を使用してクローラを実行する場合、ファイル全体がスキーマを定義するために使用されます。JSON パスを指定しないので、クローラはデータを 1 つのオブジェクト、つまり、ただの配列として処理します。たとえば、スキーマは次のようになります。

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

ただし、JSON 配列の各レコードに基づいたスキーマを作成するには、カスタム JSON 分類子を作成し、JSON パスを `$[*]` として指定します。この JSON パスを指定すると、分類子は配列内の 12 レコードすべてに問合せてスキーマを決定します。結果のスキーマには、各オブジェクトに次の例と同様の個別のフィールドが含まれています。

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

Example JSON 分類子を使用してファイルの一部のみを確認する

JSON データが、http://everypolitician.org/ から取られた JSON ファイルの例 `s3://awsglue-datasets/examples/us-legislators/all/areas.json` のパターンと同様だとします。JSON ファイルのサンプルオブジェクトは、次のようになります。

```
{
  "type": "constituency",
  "id": "ocd-division\/country:us\/state:ak",
  "name": "Alaska"
}
```

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

```
  ],
  "id": "ocd-division\/country:us\/state:ak",
  "name": "Alaska"
}
```

組み込み JSON 分類子を使用してクローラを実行する場合、ファイル全体がスキーマを作成するために使用されます。最終的に次のようなスキーマになります。

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

ただし、「id」オブジェクトだけを使用してスキーマを作成するには、カスタム JSON 分類子を作成し、JSON パスを $.id と指定します。その後、スキーマは「id」フィールドのみに基づくものとなります。

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

このスキーマで抽出されたデータの最初の数行は次のようになります。

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

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

JSON ファイルの「identifier」のように、深くネストされたオブジェクトに基づいてスキーマを作成するには、カスタム JSON 分類子を作成して JSON パスを $.identifiers[*].identifier と指定します。スキーマは前の例と似ていますが、JSON ファイル内の別のオブジェクトに基づいています。

スキーマは次のようになります。

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

テーブルからのデータの最初の数行のリストには、スキーマが「identifier」オブジェクトのデータに基づくものであることが示されます。

```
{"record": "Regional/North_America/United_States/Alaska/"}
{"record": "/m/0hjy"}
{"record": "US02"}
{"record": "5879092"}
{"record": "4001016-8"}
{"record": "destination/alaska"}
{"record": "1116270"}
{"record": "139487266"}
{"record": "n79018447"}
{"record": "01490999-8dec-4129-8254-eef6e80fadc3"}
{"record": "Alaska-state"}
{"record": "place/Alaska"}
{"record": "Q797"}
```

```
{"record": "Regional/North_America/United_States/Alabama/"}
{"record": "/m/0gyh"}
{"record": "US01"}
{"record": "4829764"}
{"record": "4084839-5"}
{"record": "161950"}
{"record": "131885589"}
```

JSON ファイルの「name」配列の「other_names」フィールドのように、別の深くネストされたオブジェクトに基づいてテーブルを作成するには、カスタム JSON 分類子を作成して JSON パスを $.other_names[*].name と指定します。スキーマは前の例と似ていますが、JSON ファイル内の別のオブジェクトに基づいています。スキーマは次のようになります。

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

テーブルからのデータの最初の数行のリストには、「name」配列内の「other_names」オブジェクトのデータに基づくものであることが示されます。

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

CSV カスタム分類子の書き込み

カスタム CSV 分類子を使用すると、カスタム csv 分類子フィールドの各列のデータ型を指定できます。各列のデータ型をコンマで区切って指定できます。データ型を指定することで、クローラーが推測したデータ型を上書きし、データが適切に分類されるようにすることができます。

分類子で CSV を処理するための SerDe を設定でき、データカタログに適用されます。

カスタム分類子を作成すると、その分類子を別のクローラーで再利用することもできます。

- ヘッダーのみ (データなし) の csv ファイルの場合、十分な情報が提供されないため、これらのファイルは UNKNOWN に分類されます。[Column headings] (列見出し) オプションで CSV の [Has headings](見出しあり) を指定し、データ型を指定すると、これらのファイルを正しく分類できます。

カスタム CSV 分類子を使用して、さまざまな種類の CSV データのスキーマを推測できます。分類子に指定できるカスタム属性には、区切り記号、CSV SerDe オプション、ヘッダーに関するオプション、およびデータに対して特定の検証を実行するかどうかなどがあります。

AWS Glue のカスタム分類子の値

CSV 分類子を定義する場合、AWS Glue に以下の値を入力して分類子を作成します。この分類子の分類フィールドは `csv` に設定してあります。

分類子名

分類子の名前。

CSV Serde

分類子で CSV を処理するため、データカタログに適用される SerDe を設定します。オプションは、「Open CSV SerDe」、「Lazy Simple SerDe」、および「None」です。クローラーが検出を行う場合は、None 値を指定できます。

列の区切り文字

行内の各列エントリを区切るものを示すカスタム記号。ユニコード文字を入力します。区切り記号を入力できない場合は、コピーと貼り付けをすることができます。これは、システムでサポートされていない文字 (通常は □ と表示) も含め、印刷可能な文字にも有効です。

引用記号

コンテンツを単一の列の値に結合するものを示すカスタム記号。列の区切り文字とは異なる必要があります。ユニコード文字を入力します。区切り記号を入力できない場合は、コピーと貼り付けをすることができます。これは、システムでサポートされていない文字 (通常は □ と表示) も含め、印刷可能な文字にも有効です。

列見出し

CSV ファイルで列見出しを検出する方法の動作を示します。カスタム CSV ファイルに列見出しがある場合は、列見出しのカンマ区切りリストを入力します。

処理オプション: 単一列のファイルを許可します

1 つの列のみを含むファイルの処理を有効にします。

処理オプション: 列値を識別する前に空白を削除します

列の値の型を識別する前に値をトリミングするかどうかを指定します。

カスタムデータ型 – オプション

カスタムデータ型をコンマで区切って入力します。CSV ファイル内のカスタムデータ型を指定します。カスタムデータ型は、サポートされているデータ型である必要があります。サポートされているデータ型は、「BINARY」、「BOOLEAN」、「DATE」、「DECIMAL」、「DOUBLE」、「FLOAT」、「INT」、「LONG」、「SHORT」、「STRING」、「TIMESTAMP」です。サポートされていないデータ型の場合、エラーが表示されます。

AWS Glue コンソールを使用した分類子の作成

分類子は、データのスキーマを決定します。カスタムの分類子を記述し、AWS Glue から指定します。

分類子の作成

AWS Glue コンソールに分類子を追加するには、[Add classifier (分類子を追加)] を選択します。分類子を定義する場合、以下の値を指定します。

- 分類子名 – 分類子の一意の名前を指定します。
- 分類子タイプ – この分類子によって推測されたテーブルの分類タイプ。
- 最終更新 – 分類子が最後に更新された時刻。

分類子名

分類子の一意の名前を指定します。

分類子タイプ

作成する分類子のタイプを選択します。

選択した分類子のタイプに応じて、分類子の次のプロパティを設定します。

Grok

- 分類

  分類されたデータの形式またはタイプを記述、またはカスタムラベルを指定します。

- Grok パターン

  これを使用してデータを解析して構造化スキーマにします。grok パターンは、データストアの形式を記述する名前付きパターンで構成されています。この grok パターンは、AWS Glue によって提供された名前付きの組み込みパターンと、[Custom patterns (カスタムパターン)] フィールドに書き込み、含まれるカスタムパターンを使用して書き込みます。grok デバッガーの結果は AWS Glue の結果と正確には一致しませんが、grok デバッガーでサンプルデータを使用してパターンを試すことをお勧めします。ウェブ上で grok デバッガーを見つけることができます。AWS Glue によって提供される名前付き組み込みパターンは、一般にウェブ上で利用可能な grok パターンと互換性があります。

  名前付きパターンを反復的に追加して grok パターンを作成し、デバッガーで結果を確認します。このアクティビティを使用すると、AWS Glue クローラが grok パターンを実行するときにデータを解析できるという確信が得られます。

- カスタムパターン

  grok 分類子の場合、これらは、記述した [Grok pattern] (Grok パターン) のオプションの構成要素です。組み込みのパターンでデータを解析できない場合は、カスタムパターンを記述する必要があります。これらのカスタムパターンはこのフィールドで定義され、[Grok pattern] (Grok パターン) フィールドで参照されます。各カスタムパターンは個別の行に定義されています。組込みパターンと同様に、[regular expression (regex)] (正規表現) 構文を使用する名前付きパターン定義で構成されています。

  たとえば、次の `MESSAGEPREFIX` という名前は、その後に正規表現の定義が続いてデータに適用され、パターンに従っているかどうかが判断されます。

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

XML

- 行タグ

  XML 分類子では、これは XML 文書のテーブル行を定義する XML タグの名前です。山括弧 < > を付けずに名前を入力します。名前は XML タグ規則に沿って命名する必要があります。

  詳しくは、「XML カスタム分類子の書き込み 」を参照してください。

JSON

- JSON パス

  JSON 分類子の場合、これは、作成するテーブルの行を定義するオブジェクト、配列、または値への JSON パスです。名前をドットで入力するか、AWS Glue でサポートされる演算子を使用して JSON 構文を括弧で囲んでください。

  詳細については、「JSON カスタム分類子の書き込み」の演算子のリストを参照してください。

CSV

- 列の区切り文字

  行内の各列エントリの区切りを示す単一の文字または記号。リストから区切り文字または記号を選択するか、`Other` を選択して、カスタム区切り文字または記号を入力します。

- 引用記号

  コンテンツを結合して単一の列の値にすることを示す単一の文字または記号。列の区切り文字とは異なる必要があります。リストから引用記号を選択するか、`Other` を選択して、カスタム引用文字を入力します。

- 列見出し

CSV ファイルで列見出しを検出する方法の動作を示します。Has headings、No headings または Detect headings を選択できます。カスタム CSV ファイルに列見出しがある場合は、列見出しのカンマ区切りリストを入力します。

- 単一列のファイルを許可

  CSV とみなされるためには、データが 2 列以上かつ 2 行以上必要です。このオプションを使用すると、1 つの列のみを含むファイルが処理できます。

- 列の値を識別する前に空白を削除

  このオプションによって、列の値のタイプを識別する前に値の空白を削除するかどうかを指定します。

- カスタムデータ型

  (オプション) – カスタムデータ型をカンマ区切りリストに入力します。サポートされているデータ型は、「BINARY」、「BOOLEAN」、「DATE」、「DECIMAL」、「DOUBLE」、「FLOAT」、「INT」、「LONG」、「SHORT」、「STRING」、「TIMESTAMP」です。

- CSV Serde

  (オプション) - 分類子で CSV を処理するための SerDe を設定でき、データカタログに適用されます。Open CSV SerDe、Lazy Simple SerDe、または None から選択します。クローラーが検出を行う場合は、None 値を指定できます。

詳細については、「さまざまなデータ形式のカスタム分類子の記述」を参照してください。

分類子の表示

作成したすべての分類子のリストを表示するには、https://console.aws.amazon.com/glue/ で AWS Glue コンソールを開き、[Classifiers] (分類子) タブを選択します。

リストには、各分類子に関する次のプロパティが表示されます。

- 分類子 – 分類子名。分類子を作成するときは、その名前を指定する必要があります。
- 分類 – この分類子によって推測されたテーブルの分類タイプ。
- 最終更新 – 分類子が最後に更新された時刻。

分類子の管理

コンソールの [ClassifiersAWS Glue] (分類子) リストから、分類子の追加、編集、削除ができます。分類子の詳細を表示するには、リスト内の分類子名を選択します。詳細には、分類子を作成したときに定義した情報が含まれます。

クローラーの設定

クローラーはデータストアにアクセスし、メタデータを識別して、テーブル定義を AWS Glue Data Catalog に作成します。 コンソールの [Crawlers (クローラー)AWS Glue] ペインには、作成したクローラーがすべて一覧表示されます。リストには、クローラーの最後の実行のステータスとメトリクスが表示されます。

このトピックでは、クローラーのパラメータの設定、クロールするデータソースの定義、セキュリティの設定、クロールされたデータの管理など、重要な側面について触れながら、クローラーを設定する工程を段階的に説明します。

トピック

- ステップ 1: クローラープロパティを設定する
- ステップ 2: データソースと分類子を選択する
- ステップ 3: セキュリティ設定を構成する
- ステップ 4: 出力とスケジュールを設定する
- ステップ 5: 確認して作成する

ステップ 1: クローラープロパティを設定する

クローラーを設定するには

1. AWS Management Consoleにサインインし、AWS Glue コンソール (https://console.aws.amazon.com/glue/) を開きます。ナビゲーションペインで [Crawlers (クローラー)] を選択します。
2. [クローラーの作成] を選択して、[クローラーの追加] ウィザードの手順に従います。ウィザードは、クローラーの作成に必要なステップをガイドします。スキーマを定義するためにカスタム分類子を追加する場合は、「分類子の定義と管理」を参照してください。
3. クローラーの名前と説明 (オプション) を入力します。必要に応じて、クローラーに [タグキー] とオプションの [タグ値] でタグを付加できます。作成後、タグキーは読み取り専用になりま

す。リソースを整理、識別しやすいように、いくつかのリソースでタグを使用します。詳細については、「AWS Glue の AWS タグ」を参照してください。

名前

名前に使用できるのは、文字 (A ~ Z)、数字 (0 ~ 9)、ハイフン (-)、またはアンダースコア (_) で、長さは 255 文字までです。

説明

説明の長さは、最大 2048 文字です。

タグ

タグを使用して、リソースの整理と識別を行います。詳細については、次を参照してください:

- AWSAWS Glue のタグ

ステップ 2: データソースと分類子を選択する

次に、クローラーのデータソースと分類子を設定します。

サポートされるデータソースについては、「クロールでサポートされているデータソース」を参照してください。

データソースの設定

に適切なオプションを選択します。データは既にマッピングされていますか。AWS Glue テーブル？「まだ」または「はい」を選択します。デフォルトでは、「未実施」が選択されています。

クローラーは、クロールのソースとして直接データストアにアクセスでき、また既存のデータカタログのテーブルをソースとして使用することもできます。クローラーが既存のカタログテーブルを使用する場合、それらのカタログのテーブルで指定されたデータストアをクロールします。

- [Not yet] (未実施): クローリングする 1 つまたは複数のデータソースを選択します。クローラーは、さまざまなタイプの複数のデータストア (Amazon S3、 JDBCなど) をクロールできます。

  一度に設定できるデータストアは 1 つだけです。接続情報とインクルードパスと除外パターンを指定すると、別のデータストアを追加することもできます。
- はい: から既存のテーブルを選択します。AWS Glue データカタログ。カタログテーブルは、クロールするデータストアを指定します。クローラーは 1 回の実行でのカタログテーブルのみをクロールできます。他のソースタイプを混在させることはできません。

ソースとしてカタログテーブルを指定する一般的な理由は、(データストアの構造をすでに知っていたため) テーブルを手動で作成した場合に、新しいパーティションの追加を含め、クローラーにテーブルを更新された状態に保持させるためです。他の理由の説明については、「クローラーを使用して手動で作成されたデータカタログテーブルを更新する」を参照してください。

既存のテーブルをクローラーソースのタイプとして指定する場合は、次の条件が適用されます。

- データベース名はオプションです。
- Amazon S3、Amazon DynamoDB、または Delta Lake データストアを指定するカタログテーブルのみが使用できます。
- クローラーが実行されるとき、新しいカタログテーブルは作成されません。既存のテーブルは、新しいパーティションの追加を含め、必要に応じて更新されます。
- データストアで見つかる削除されたオブジェクトは無視されます。カタログテーブルが削除されません。代わりに、クローラーによりログメッセージが書き込まれます (`SchemaChangePolicy.DeleteBehavior=LOG`)
- Amazon S3 パスごとに単一のスキーマを作成するクローラー設定オプションはデフォルトで有効になっており、無効にすることはできません。(`TableGroupingPolicy=CombineCompatibleSchemas`) 詳しくは、「各 Amazon S3 インクルードパスの単一のスキーマを作成する」を参照してください。
- カタログテーブルをソースとして他のソースタイプ (例: Amazon S3 または Amazon DynamoDB) と混在させることはできません。

Delta テーブルを使用するには、まず Athena DDLまたは を使用して Delta テーブルを作成します AWS Glue API。

Athena を使用して、場所を Amazon S3 フォルダに設定し、テーブルタイプを 'DELTA' に設定します。

```
CREATE EXTERNAL TABLE database_name.table_name
LOCATION 's3://bucket/folder/'
TBLPROPERTIES ('table_type' = 'DELTA')
```

を使用して AWS Glue API、テーブルパラメータマップ内でテーブルタイプを指定します。テーブルパラメータには、次のキーおよび値のペアを含める必要があります。テーブルを作成する方法の詳細については、「create_table の Boto3 ドキュメント」を参照してください。

```
{
    "table_type":"delta"
}
```

データソース

クローラーでスキャンするデータソースのリストを選択または追加します。

(オプション) データソースJDBCとして を選択した場合は、JDBCドライバー情報が保存されている接続アクセスを指定するときに、独自のドライバーを使用できます。

インクルードパス

クロールで何を含め、何を除外するかを評価する際、クローラーは必要なインクルードパスを評価することから始めます。Amazon S3、MongoDB、MongoDB Atlas、Amazon DocumentDB (MongoDB 互換)、リレーショナルデータストアの場合は、インクルードパスを指定する必要があります。

Amazon S3 データストアの場合

このアカウントのパスを指定するか、別のアカウントのパスを指定するかを選択し、Amazon S3 パスを参照して選択します。

Amazon S3 データストアの場合、インクルードパスの構文は `bucket-name/folder-name/file-name.ext` です。バケット内のすべてのオブジェクトをクロールするには、インクルードパスにバケット名のみ指定します。除外パターンは、インクルードパスを基準とする相対パスです。

Delta Lake データストアの場合

Delta テーブルへの 1 つ以上の Amazon S3 パスを s3://*bucket*/*prefix*/*object*.

Iceberg または Hudi データストアの場合

Iceberg または Hudi テーブルメタデータが s3:// のフォルダを含む 1 つ以上の Amazon S3 パスを指定します。*bucket*/*prefix*.

Iceberg および Hudi データストアの場合、Iceberg/Hudi フォルダはルートフォルダの子フォルダにある場合があります。クローラーは、Hudi フォルダのパス以下にあるすべてのフォルダをスキャンします。

JDBC データストアの場合

Enter *<database>*/*<schema>*/*<table>* または *<database>*/*<table>*データベース製品に応じて、。Oracle Database と MySQL は、パスでスキーマをサポートしていません。パーセ

ント (%) 文字を に置き換えることができます。*<schema>* または *<table>*。 例えば、システム識別子 (SID) が の Oracle データベースの場合orcl、 orcl/%を入力して、接続で という名前のユーザーがアクセスできるすべてのテーブルをインポートします。

> **Important**
>
> このフィールドでは、大文字と小文字が区別されます。

> **Note**
>
> 独自のJDBCドライバーバージョンを導入する場合は、AWS Glue クローラーは でリソースを消費します AWS Glue と Amazon S3 バケットを に割り当てて、提供されたドライバーが環境で実行されていることを確認します。リソースの追加使用量はアカウントに反映されます。ドライバーは、 AWS Glue 接続。

MongoDB、MongoDB Atlas、Amazon DocumentDB データストアの場合

MongoDB、MongoDB Atlas、Amazon DocumentDB (MongoDB 互換) の場合、構文は database/collection です。

JDBC データストアの場合、構文は database-name/schema-name/table-name または ですdatabase-name/table-name。構文は、データベースエンジンでデータベース内のスキーマがサポートされているかどうかに依存します。例えば、MySQL や Oracle などのデータベースエンジンの場合、インクルードパスschema-nameに を指定しないでください。インクルードパスでスキーマやテーブルの代わりにパーセント記号 (%) を使用することで、データベース内のすべてのスキーマやテーブルを表すことができます。インクルードパスでデータベースの代わりにパーセント記号 (%) を使用することはできません。

最大トラバーサル深度 (Iceberg または Hudi データストアのみ)

クローラーが Amazon S3 パス内の Iceberg または Hudi メタデータフォルダを検出するために通過できる Amazon S3 パスの最大深度を定義します。このパラメータの目的は、クローラーの実行時間を制限することです。デフォルト値は 10 で、最大値は 20 です。

除外パターン

この機能を利用すると、特定のファイルまたはテーブルをクロールから除外できます。エクスクルードパスは、インクルードパスを基準とする相対パスです。たとえば、JDBCデータストア内のテーブルを除外するには、除外パスにテーブル名を入力します。

クローラーは、 を使用してJDBCデータストアに接続します。AWS Glue 接続文字列を含むJDBCURI接続。クローラーは、 のJDBCユーザー名とパスワードを使用してデータベースエンジン内のオブジェクトにのみアクセスできます。AWS Glue 接続。クローラーは、JDBC接続を介してアクセスできるテーブルのみを作成できます。クローラーが を使用してデータベースエンジンにアクセスするとJDBCURI、インクルードパスを使用して、データベースエンジン内のどのテーブルがデータカタログに作成されるかを決定します。例えば、My ではSQL、 の包含パスを指定するとMyDatabase/%、 内のすべてのテーブルMyDatabaseがデータカタログに作成されます。Amazon Redshift にアクセスする場合、MyDatabase/% というインクルードパスを指定すると、データベース MyDatabase の各スキーマ内のすべてのテーブルがデータカタログで作成されます。MyDatabase/MySchema/% のインクルードパスを指定すると、データベース MyDatabase のすべてのテーブルとスキーマ MySchema が作成されます。

インクルードパスの指定後に、1 つ以上の Unix 形式の glob 除外パターンを指定することで、インクルードパスに含まれる予定であったオブジェクトをクロールから除外できます。これらのパターンはインクルードパスに適用されて、どのオブジェクトを除外するか決定します。また、これらのパターンはクローラによって作成されるテーブルのプロパティとして保存されます。AWS Glue PySpark などの拡張機能はcreate_dynamic_frame.from_catalog、テーブルプロパティを読み取って、除外パターンで定義されたオブジェクトを除外します。

AWS Glue は、除外globパターンで次のパターンをサポートします。

除外パターン	説明
*.csv	.csv で終わる現在のフォルダ内のオブジェクト名を表す Amazon S3 パスと一致する
.	ドットを含むオブジェクト名すべてと一致する
*.{csv,avro}	.csv か .avro で終わるオブジェクト名と一致する
foo.?	foo. で始まり、その後に 1 文字の拡張子が続くオブジェクト名と一致する
myfolder/*	/myfolder/mysource など、myfolder のサブフォルダの 1 つのレベルにあるオブジェクトと一致する

除外パターン	説明
myfolder/*/*	/myfolder/mysource/data　など、myfolder のサブフォルダの 2 つのレベルにあるオブジェクトと一致する
myfolder/**	myfolder のすべてのサブフォルダにあるオブジェクト (/myfolder/mysource/mydata や /myfolder/mysource/data など) と一致する
myfolder**	/myfolder　や /myfolder/mydata.txt　などの myfolder 以下のファイルと同様に、サブフォルダ myfolder と一致する
Market*	JDBC データベース内のテーブルをMarket、 Market_us　や などの で始まる名前と照合します。 Market_fr

AWS Glue は次のようにglob除外パターンを解釈します。

- スラッシュ (/) 文字は、Amazon S3 キーをフォルダ階層に区切る区切り記号です。
- アスタリスク (*) 記号は、フォルダの境界を超えない、0 文字以上の名前の要素と一致します。
- 二重アスタリスク (**) は、フォルダやスキーマの境界を越える 0 個以上の文字に相当します。
- 疑問符 (?) 記号は、名前の要素のちょうど 1 文字に相当します。
- バックスラッシュ (\) 文字は、本来ならば特殊文字として解釈される文字を工スケープ処理するために使用されます。\\ 式はバックスラッシュ 1 つに相当し、\{ は左括弧に相当します。
- 角括弧 ([ ]) は、一連の文字の中から、名前の要素の 1 文字に相当する角括弧式を作成します。たとえば、[abc] は a、b、または c に一致します。ハイフン (-) は、範囲を指定するために使用されます。つまり、[a-z] は a から z (この値を含みます) までに相当する範囲を指定します。これらのフォームは組み合わせることができます。そのため、[abce-g] は a、b、c、e、f、または g に一致します。角括弧 ([) の後の文字が感嘆符 (!) の場合、角括弧式は否定の意味になります。たとえば、[!a-c] は a、b、または c 以外のすべての文字に一致します。

角括弧式内では、*、?、および \ 文字は、文字通りの意味です。ハイフン (-) 文字は、角括弧内で最初の文字だった場合、または式を否定する ！ の次の文字だった場合は、文字通りの意味です。

- 中括弧 ({ }) は、グループ内のサブパターンが一致する場合にグループが一致するサブパターンのグループを囲みます。カンマ (,) 文字は、サブパターンを分割するために使用されます。グループはネストできません。
- ファイル名の先頭のピリオドまたはドット文字は、マッチ操作では通常の文字として扱われます。たとえば、* 除外パターンは、ファイル名 .hidden に一致します。

Example Amazon S3 除外パターンの例

各除外パターンは、インクルードパスに対して評価されます。例えば、次の Amazon S3 ディレクトリ構造があるとします。

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

インクルードパスが s3://mybucket/myfolder/ の場合、以下は除外パターンのサンプル結果の一部です。

除外パターン	結果
departments/**	departments フォルダ内のすべてのファイルとフォルダを除外し、employees フォルダとそのファイルを含めます。
departments/market*	market-us.json 、market-emea.json 、および market-ap.json を除外します

除外パターン	結果
`**.csv`	名前が .csv で終わる myfolder 以下のすべてのオブジェクトを除外します
`employees/*.csv`	employees フォルダ内のすべての .csv ファイルを除外します。

Example Amazon S3 パーティションのサブセットの除外の例

データが日別に分割されていて、1 年の日別に異なる Amazon S3 パーティションに入っているとします。2015 年 1 月には、31 のパーティションがあります。ここで、1 月の第 1 週のみのデータをクロールするには、1 日〜7 日を除くすべてのパーティションを除外する必要があります。

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

この glob パターンの各パートを見てみます。最初のパートは 2015/01/{[!0],0[8-9]}** で、"0" で始まらないすべての日付、および 2015 年 01 月の 08 日目および 09 日目を除外しています。"**" を日数パターンのサフィックスとして使用すると、下位レベルフォルダへのフォルダ境界を越えることに注意してください。"*" を使用すると、下位レベルフォルダは除外されます。

2 番目のパートは 2015/0[2-9]/** で、2015 年 02 から 09 月までの日を除外します。

3 番目のパートは 2015/1[0-2]/** で、2015 年 10、11、12 月の日を除外します。

Example JDBC パターンを除外する

次のスキーマ構造でJDBCデータベースをクロールするとします。

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

インクルードパスが MyDatabase/MySchema/% の場合、以下は除外パターンのサンプル結果の一部です。

除外パターン	結果
HR*	HR で始まる名前を持つテーブルを除外
Market_*	Market_ で始まる名前を持つテーブルを除外
**_Table	_Table で終わる名前を持つテーブルをすべて除外

追加のクローラーソースパラメータ

各ソースのタイプには、異なる追加パラメータのセットが必要です。

接続

を選択または追加する AWS Glue 接続。接続の詳細については、「データへの接続」を参照してください

追加メタデータ - オプション (JDBCデータストアの場合)

クローラーがクローリングするための追加のメタデータプロパティを選択します。

- [Comments] (コメント): 関連するテーブルレベルと列レベルのコメントをクローリングします。
- [Raw types] (未加工型): 表の列の未加工のデータ型を追加のメタデータに保持します。デフォルトの動作として、クローラーは未処理のデータ型を Hive 互換の型に変換します。

JDBC ドライバークラス名 - オプション (JDBCデータストアの場合)

データソースに接続するクローラーのカスタムJDBCドライバークラス名を入力します。

- Postgres: org.postgresql.Driver
- マイ SQL: com.mysql.jdbc.Driver、com.mysql.cj.jdbc.Driver
- Redshift: com.amazon.redshift.jdbc.Driver、com.amazon.redshift.jdbc42.Driver
- Oracle: oracle.jdbc.driver。OracleDriver
- SQL サーバー: com.microsoft.sqlserver.jdbc。SQLServerDriver

JDBC ドライバー S3 パス - オプション (JDBCデータストア用)

`.jar` ファイルへの既存の Amazon S3 パスを選択します。これは、クローラーがデータソースに接続するためにカスタムJDBCドライバーを使用する場合に`.jar`ファイルを保存する場所です。

データサンプリングを有効にする (Amazon DynamoDB、MongoDB、MongoDB Atlas、Amazon DocumentDB データストアのみ)

データサンプルのみをクロールするかどうかを選択します。選択しない場合は、テーブル全体がクロールされます。テーブルが高スループットテーブルではない場合、すべてのレコードのスキャンには時間がかかることがあります。

クエリ用のテーブルを作成する (Delta Lake データストア用のみ)

Delta Lake テーブルの作成方法を次の中から選択します。

- [Create Native tables] (ネイティブテーブルの作成): Delta トランザクションログの直接クエリをサポートするクエリエンジンとの統合を可能にします。
- [Create Symlink tables] (Symlink テーブルの作成): 指定された設定パラメータに基づいて、マニフェストファイルを分割キーで分割して Symlink マニフェストを作成します。

スキャンレート - オプション (DynamoDB データストア用のみ)

クローラーで使用する DynamoDB テーブル読み取りキャパシティーユニットの割合を指定します。読み取りキャパシティーユニットは、DynamoDB で定義されている用語で、テーブルに対して実行できる読み取り回数/秒のレート制限として機能する数値です。0.1 ~ 1.5 の値を入力します。指定しない場合、プロビジョニングされたテーブルの場合は 0.5、オンデマンドテーブルの場合は最大の設定済みキャパシティーの 1/4 にデフォルト設定されます。 AWS Glue クローラーでは、プロビジョニングされたキャパシティモードのみを使用する必要があることに注意してください。

> **Note**
>
> DynamoDB データストアの場合、テーブルの読み取りと書き込みを処理するプロビジョニング容量モードを設定します。 AWS Glue クローラーは、オンデマンドキャパシティモードでは使用しないでください。

ネットワーク接続 - オプション (Amazon S3、Delta、Iceberg、Hudi、および Catalog ターゲットデータストア用)

必要に応じて、この Amazon S3 ターゲットで使用するネットワーク接続を含めます。各クローラーは 1 つのネットワーク接続に制限されているため、他の Amazon S3 ターゲットも同じ接続を使用します (空欄の場合は接続なし)。

接続の詳細については、「データへの接続」を参照してください

ファイルのサブセットのみのサンプリングとサンプルサイズ (Amazon S3 データストア用のみ)

データセット内のサンプルファイルをクロールするときにクロールされる各リーフフォルダ内のファイル数を指定します。この機能をオンにすると、このデータセット内のすべてのファイルをクロールする代わりに、クローラーはクロールする各リーフフォルダ内の一部のファイルをランダムに選択します。

サンプリングするクローラーは、データ形式について事前に知識があり、フォルダ内のスキーマが変更されないことを知っているお客様に最適です。この機能をオンにすると、クローラーの実行時間が大幅に短縮されます。

有効な値は、1 から 249 までの整数です。指定しない場合、すべてのファイルがクロールされます。

以降のクローラー実行

このフィールドは、すべての Amazon S3 データソースに影響するグローバルフィールドです。

- [Crawl all sub-folders] (すべてのサブフォルダをクローリング): 以降クローリングするたびに、すべてのフォルダを再度クローリングします。
- [Crawl new sub-folders only] (新しいサブフォルダのみをクローリング): 最近のクローリング以降に追加された Amazon S3 フォルダのみをクローリングします スキーマに互換性がある場合、新しいパーティションが既存のテーブルに追加されます。詳細については、「the section called “増分クロールのスケジューリング”」を参照してください。
- [Crawl based on events] (イベントに基づいてクローリング): Amazon S3 イベントを利用して、クローリングするフォルダを制御します。詳細については、「Amazon S3 イベント通知を使用した加速クロール」を参照してください。

カスタム分類子 - オプション

クローラーを定義する前にカスタム分類子を定義します。分類子は、指定されたファイルがクローラーで処理可能な形式であるかどうかをチェックします。処理可能な場合、分類子は、そのデータ形式と一致する `StructType` オブジェクトの形式でスキーマを作成します。

詳細については、「分類子の定義と管理」を参照してください。

ステップ 3: セキュリティ設定を構成する

IAM ロール

クローラーはこのロールを取得します。これは、AWS 管理ポリシー `AWSGlueServiceRole` と同様のアクセス許可を持っている必要があります。Amazon S3 および DynamoDB ソースの場合、データストアへのアクセス許可も必要です。クローラーが AWS Key Management Service (AWS KMS) で暗号化された Amazon S3 データを読み取る場合は、ロールには AWS KMS キーでの復号化のアクセス許可が必要です。

Amazon S3 データストアの場合、ロールにアタッチされた追加のアクセス許可は次のようになります。

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

Amazon DynamoDB データストアの場合、ロールにアタッチされた追加のアクセス許可は次のようになります。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:DescribeTable",
```

```
        "dynamodb:Scan"
      ],
      "Resource": [
        "arn:aws:dynamodb:region:account-id:table/table-name*"
      ]
    }
  ]
}
```

自身の JDBC ドライバーを追加するには、アクセス許可を新たに追加する必要があります。

- ジョブアクション CreateJob、DeleteJob、GetJob、GetJobRun、StartJobRun にアクセス権限を付与します。
- Amazon S3 アクション s3:DeleteObjects、s3:GetObject、s3:ListBucket、s3:PutObject にアクセス権限を付与します。

> **Note**
>
> Amazon S3 バケットポリシーが無効になっている場合、s3:ListBucket は必要ありません。

- サービスプリンシパルに Amazon S3 ポリシーのバケット/フォルダへのアクセス権限を付与します。

Amazon S3 ポリシーの例:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject",
                "s3:ListBucket",
                "s3:DeleteObject"
            ],
            "Resource": [
                "arn:aws:s3:::bucket-name/driver-parent-folder/driver.jar",
                "arn:aws:s3:::bucket-name"
```

```
            ]
        }
    ]
}
```

AWS Glue は以下のフォルダ (_crawler と _glue_job_crawler) を Amazon S3 バケットに JDBC ドライバーと同じレベルで作成します。例えば、ドライバーパスが <s3-path/driver_folder/driver.jar> の場合、以下のフォルダがまだ存在しない場合は作成されます。

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

必要に応じて、クローラーにセキュリティ設定を追加して、保管時の暗号化オプションを指定することができます。

詳細については、ステップ 2: AWS Glue 用の IAM ロールを作成するおよびAWS Glue のアイデンティティとアクセスの管理を参照してください。

Lake Formation 設定 - オプション

クローラーが Lake Formation の認証情報をデータソースのクローリングに使用できるようにします。

[Use Lake Formation credentials for crawling S3 data source] (S3 データソースのクローリングに Lake Formation 認証情報を使用する) をオンにすると、クローラーは Lake Formation 認証情報を使用してデータソースをクローリングできるようになります。データソースが別のアカウントに属する場合は、登録されているアカウント ID を指定する必要があります。そうしないと、クローラーはアカウントに関連付けられているデータソースのみをクローリングします。Amazon S3 とデータカタログのデータソースにのみ適用されます。

セキュリティ設定 - オプション

設定にはセキュリティ設定が含まれます。詳細については、次を参照してください:

- AWS Glue によって書き込まれたデータの暗号化

> **Note**
>
> クローラーにセキュリティ設定を設定すると、変更できますが、削除することはできません。クローラーのセキュリティレベルを下げるには、設定内でセキュリティ機能を明示的に `DISABLED` に設定するか、新しいクローラーを作成します。

ステップ 4: 出力とスケジュールを設定する

出力設定

オプションには、検出されたスキーマの変更、データストア内の削除されたオブジェクトをクローラーが処理する方法があります。詳細については、「クローラーの動作のカスタマイズ」を参照してください。

クローラースケジュール

AWS Glue では、オンデマンドでクローラーを実行することも、クローラーとジョブの時間ベースのスケジュールを定義することもできます。これらのスケジュールの定義は、Unix と同様の cron 構文を使用します。詳細については、「クローラのスケジュール」を参照してください。

ステップ 5: 確認して作成する

設定したクローラー設定を確認して、クローラーを作成します。

クローラのスケジュール

AWS Glue クローラは、オンデマンドで、または定期的なスケジュールで実行できます。スケジュールに基づいてクローラーを設定する場合は、クローラーの実行頻度、実行する曜日、実行時間などの特定の制約を指定できます。これらのカスタムスケジュールは cron 形式で作成できます。詳細については、Wikipedia の「cron」を参照してください。

クローラスケジュールを設定するときは、cron の機能と制限を考慮する必要があります。たとえば、毎月 31 日にクローラを実行することを選択した場合、いくつかの月には 31 日間はないことに注意してください。

トピック

- クローラースケジュールを作成する
- 既存のクローラーのスケジュールを作成する

クローラースケジュールを作成する

AWS Glue コンソールまたは AWS CLI を使ってクローラーのスケジュールを作成できます。

AWS Management Console

1. AWS Management Console にサインインし、AWS Glue コンソール (https://console.aws.amazon.com/glue/) を開きます。
2. ナビゲーションペインで [Crawlers (クローラー)] を選択します。
3. セクション「クローラーの設定」のステップ 1 ~ 3 に従います。
4. ステップ 4: 出力とスケジュールを設定する で、クローラースケジュールを選択し、実行頻度を設定します。クローラーの実行は毎時、毎日、毎週、毎月から選ぶことができますが、cron 式を使用してカスタムスケジュールを定義することもできます。

   cron 式はスケジュールパターンを表す文字列で、スペースで区切られた 6 つのフィールド (* * * * * <minute> <hour> <day of month> <month> <day of week> <year>) で構成されます。

   例えば、毎日午前 0 時にタスクを実行する場合、cron 式は 0 0 * * ? * になります。

   詳細については、「Cron 式」を参照してください。
5. 設定したクローラー設定を確認し、スケジュールに従って実行されるクローラーを作成します。

AWS CLI

```
aws glue create-crawler
 --name myCrawler \
 --role AWSGlueServiceRole-myCrawler  \
 --targets '{"S3Targets":[{Path="s3://amzn-s3-demo-bucket/"}]}' \
 --schedule cron(15 12 * * ? *)
```

cron を使用してジョブおよびクローラをスケジュールする方法の詳細については、「ジョブとクローラの時間ベースのスケジュール」を参照してください。

既存のクローラーのスケジュールを作成する

既存のクローラーの定期的なスケジュールを設定するには、次の手順に従います。

AWS Management Console

1. AWS Management Consoleにサインインし、AWS Glue コンソール (https://console.aws.amazon.com/glue/) を開きます。
2. ナビゲーションペインで [Crawlers (クローラー)] を選択します。
3. 使用可能なリストから、スケジュールするクローラーを選択します。
4. [アクション] メニューで、[編集] を選択します。
5. [ステップ 4: 出力とスケジュールを設定する] まで下にスクロールし、[編集] を選択します。
6. [クローラースケジュール] でクローラースケジュールを更新します。
7. [Update] (更新) を選択します。

AWS CLI

既存のクローラー設定を更新するには、次の CLI コマンドを使用します。

```
aws glue update-crawler-schedule
   --crawler-name myCrawler
   --schedule cron(15 12 * * ? *)
```

クローラーの結果と詳細の表示

クローラーが正常に実行されると、データカタログにテーブル定義が作成されます。クローラーの実行が完了したら、ナビゲーションペインで [Tables] (テーブル) をクリックして、指定したデータベースにクローラーにより作成されたテーブルを表示します。

クローラー自体に関連する情報は、次のように表示されます。

- AWS Glue コンソールの [Crawlers] (クローラー) ページに、クローラーの次のプロパティが表示されます。

プロパティ	説明
名前	クローラーを作成する場合、一意の名前を付ける必要があります。

プロパティ	説明
ステータス	クローラーには、準備完了、開始中、停止中、スケジュールあり、スケジュール停止などの状態があります。実行中のクローラーは、開始中から停止中まで処理していきます。クローラーにアタッチされたスケジュールを再開または一時停止できます。
スケジュール	クローラーをオンデマンドで実行するか、または、スケジュールで頻度を選択できます。クローラーのスケジュールの詳細については、「クローラのスケジュール」を参照してください。
最後の実行	クローラーを最後に実行した日時。
ログ	クローラーの最後の実行からの使用可能なログにリンクします。
最後の実行以後のテーブルの変更	クローラーの最後の実行により更新された AWS Glue Data Catalog のテーブルの数。

- クローラーの履歴を表示するには、ナビゲーションペインで [Crawlers] (クローラー) を選択して、作成したクローラーを表示します。使用可能なクローラーのリストからクローラーを選択します。クローラーのプロパティとクローラーの履歴は [Crawler runs] (クローラーの実行) タブに表示されます。

  [Crawler runs] (クローラーの実行) タブには、クローラーの各実行履歴に対して、[Start time (UTC)] (開始時刻 (UTC))、[End time (UTC)] (終了時刻 (UTC))、[Duration] (所要時間)、[Status] (ステータス)、[DPU hours] (DPU 時間)、[Table changes] (表の変更) などが表示されます。

  [クローラーの実行] タブは、クローラー履歴機能の起動日以降に発生したクロールのみを表示し、最大 12 か月分のクロールのみを保持します。古いクロールは返されません。

- 追加情報を表示するには、クローラー詳細ページにあるタブを選択します。各タブには、クローラーに関連する情報が表示されます。
  - [Schedule] (スケジュール): クローラー用に作成されたすべてのスケジュールがここに表示されます。

- [Data sources] (データソース): クローラーによってスキャンされたすべてのデータソースがここに表示されます。
- [Classifiers] (分類子): クローラーに割り当てられたすべての分類子がここに表示されます。
- [Tags] (タグ): タグが作成され、AWS リソースに割り当てられていれば、ここに表示されます。

クローラーによって設定されたデータカタログテーブルのパラメータ

これらのテーブルプロパティは、AWS Glue クローラーによって設定されます。ユーザーが `classification` および `compressionType` プロパティを使用することを想定しています。テーブルサイズの推定値など、その他のプロパティは内部計算に使用されるものであり、その精度やお客様のユースケースへの適用性は保証されません。これらのパラメータを変更すると、クローラーの動作が変わる可能性があるため、このワークフローはサポートされません。

プロパティキー	プロパティ値
`UPDATED_BY_CRAWLER`	更新を実行するクローラーの名前。
`connectionName`	データストアに接続するために使用されるクローラーのデータカタログ内にある接続名。
`recordCount`	ファイルサイズとヘッダーに基づいた、テーブル内の推定レコード数。
`skip.header.line.count`	スキップヘッダーにスキップされた行数。CSV に分類されたテーブルに設定します。
`CrawlerSchemaSerializerVersion`	内部使用目的
`classification`	クローラーによって推測されたデータの形式。AWS Glue クローラーでサポートされるデータ形式の詳細については、「the section called “組み込み分類子”」を参照してください。
`CrawlerSchemaDeserializerVersion`	内部使用目的

プロパティキー	プロパティ値
sizeKey	クロールされたテーブル内のファイルの合計サイズ。
averageRecordSize	テーブル内の行の平均サイズ (バイト単位)。
compressionType	テーブル内のデータで使用される圧縮タイプ。AWS Glue クローラーでサポートされる圧縮タイプの詳細については、「the section called “組み込み分類子”」を参照してください。
typeOfData	file、table、または view。
objectCount	テーブルの Amazon S3 パスの下にあるオブジェクトの数。

これらの追加のテーブルプロパティは、Snowflake データストアの AWS Glue クローラーによって設定されます。

プロパティキー	プロパティ値
aws:RawTableLastAltered	Snowflake テーブルの最後に変更されたタイムスタンプを記録します。
ViewOriginalText	SQL ステートメントを表示します。
ViewExpandedText	Base64 形式でエンコードされた SQL ステートメントを表示します。
ExternalTable:S3Location	Snowflake 外部テーブルの Amazon S3 の場所。
ExternalTable:FileFormat	Snowflake 外部テーブルの Amazon S3 ファイル形式。

これらの追加のテーブルプロパティは、Amazon Redshift、Microsoft SQL Server、MySQL、PostgreSQL、Oracle などの JDBC タイプのデータストアの AWS Glue クローラーによって設定されます。

プロパティキー	プロパティ値
`aws:RawType`	クローラーがデータカタログにデータを保存すると、データ型が Hive 互換型に変換されるため、ネイティブデータ型の情報が失われることがよくあります。クローラーは、ネイティブレベルのデータ型を提供する `aws:RawType` パラメータを出力します。
`aws:RawColumnComment`	コメントがデータベース内の列に関連付けられている場合、クローラーはカタログテーブル内の対応するコメントを出力します。コメント文字列は 255 バイトに切り捨てられます。 Microsoft SQL Server ではコメントはサポートされていません。
`aws:RawTableComment`	コメントがデータベース内のテーブルに関連付けられている場合、クローラーはカタログテーブル内の対応するコメントを出力します。コメント文字列は 255 バイトに切り捨てられます。 Microsoft SQL Server ではコメントはサポートされていません。

クローラーの動作のカスタマイズ

AWS Glue クローラー を設定する場合、クローラーの動作を定義するためのオプションをいくつか使用できます。

- 増分クロール – テーブルスキーマに新しいパーティションのみを追加する増分クロールを実行するようにクローラーを設定できます。
- パーティションインデックス – クローラーは、特定のパーティションを効率的に検索するために、デフォルトで Amazon S3 および Delta Lake ターゲットのパーティションインデックスを作成します。
- Amazon S3 イベントを使用してクロール時間を短縮する – Amazon S3 または Data Catalog ターゲット全体を一覧表示するのではなく、Amazon S3 イベントを使用してイベントをトリガーしたサブフォルダからのすべてのファイルを一覧表示して 2 つのクロール間の変更を識別するようにクローラーを設定できます。

- スキーマの変更の処理 – クローラーが既存のスキーマに対してスキーマの変更を行わないようにすることができます。AWS Management Console または AWS Glue API を使用して、特定のタイプの変更をクローラーで処理する方法を設定できます。
- 複数の Amazon S3 パスの単一のスキーマ – データに互換性がある場合は、各 S3 パスの単一のスキーマを作成するようにクローラーを設定できます。
- テーブルの場所とパーティションレベル – テーブルレベルのクローラーオプションを使用すると、クローラーにテーブルの配置場所やパーティションの作成方法を柔軟に指定できます。
- テーブルしきい値 – テーブルしきい値を指定することで、クローラーが作成できるテーブルの最大数を指定できます。
- AWS Lake Formation 認証情報 – Lake Formation 認証情報を使用するようにクローラーを設定すると、同じ AWS アカウント または別の AWS アカウント 内の Amazon S3 データストアまたは Data Catalog テーブルにアクセスできます。これらのテーブルは、Amazon S3 の場所を基盤としています。

AWS Glue コンソールを使用してクローラーを追加する方法の詳細については、「クローラーの設定」を参照してください。

トピック

- 新しいパーティションを追加するための増分クロールのスケジューリング
- パーティションインデックスの生成
- クローラーによる既存のスキーマの変更を防止する
- 各 Amazon S3 インクルードパスの単一のスキーマを作成する
- テーブルの場所とパーティションレベルの指定
- クローラーが作成できるテーブルの最大数を指定する
- Lake Formation 認証情報を使用するようにクローラーを設定する
- Amazon S3 イベント通知を使用した加速クロール

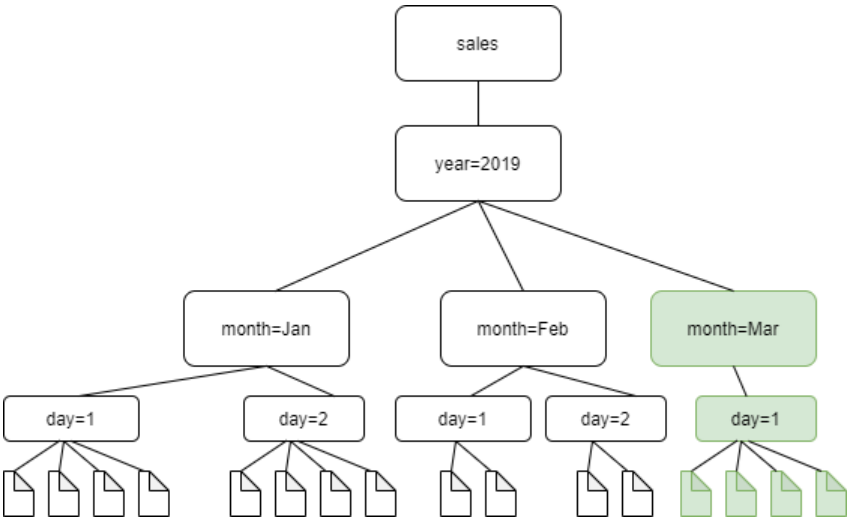
新しいパーティションを追加するための増分クロールのスケジューリング

テーブルスキーマに新しいパーティションのみを追加する増分クロールを実行するように AWS Glue クローラー を設定できます。クローラーは、初回実行時にデータソース全体を処理するフルクロールを実行し、完全なスキーマと既存のすべてのパーティションを AWS Glue Data Catalog に記録します。

最初のフルクロールの後のクロールは増分となり、クローラーは前回のクロール以降に導入された新しいパーティションのみを識別して追加します。このアプローチにより、クローラーは実行ごとにデータソース全体を処理する必要がなくなり、代わりに新しいパーティションにだけ焦点を当てるため、クロール時間が短縮されます。

> **Note**
>
> 増分クロールは、既存のパーティションの変更や削除を検出しません。この設定は、安定したスキーマを持つデータソースに最適です。大規模なスキーマ変更が発生した場合は、新しいスキーマを正確に取得するために、一時的にクローラーがフルクロールを実行するように設定し、その後増分クローリングモードに戻すことをお勧めします。

次の図は、増分クロール設定を有効にすると、クローラーが新しく追加されたフォルダ、month=March のみを検出してカタログに追加する状況を示しています。



クローラーを更新して増分クロールを実行するには、次の手順に従います。

AWS Management Console

1. AWS Management Consoleにサインインし、AWS Glue コンソール (https://console.aws.amazon.com/glue/) を開きます。
2. [Data Catalog] で [クローラー] を選択します。
3. 増分的にクロールするように設定するクローラーを選択します。
4. [編集] を選択します。
5. [ステップ 2] を選択します。データソースと分類子を選択します。

6. 増分的にクロールするデータソースを選択します。
7. [編集] を選択します。
8. [それ以降のクローラー実行] で、[新しいサブフォルダのみをクローリング] を選択します。
9. [Update] (更新) を選択します。

クローラーのスケジュールを作成するには、「the section called “クローラのスケジュール”」を参照してください。

AWS CLI

```
aws glue update-crawler \
 --name myCrawler \
 --recrawl-policy RecrawlBehavior=CRAWL_NEW_FOLDERS_ONLY \
 --schema-change-policy UpdateBehavior=LOG,DeleteBehavior=LOG
```

注意と制限

このオプションをオンにすると、クローラーの編集時に Amazon S3 ターゲットデータストアを変更できなくなります。このオプションは、ある特定のクローラー設定に影響します。オンにすると、クローラーの更新動作と削除動作が LOG になります。これにより、以下のように処理されます。

- スキーマに互換性がないオブジェクトを検出した場合、クローラーはデータカタログにオブジェクトを追加せず、この詳細を CloudWatch Logs のログとして追加します。
- データカタログで削除されたオブジェクトは更新されません。

パーティションインデックスの生成

Data Catalog は、特定のパーティションを効率的に検索するためのパーティションインデックスの作成をサポートしています。詳細については、「パーティションインデックスの作成」を参照してください。AWS Glue クローラーは、デフォルトで Amazon S3 および Delta Lake ターゲットのパーティションインデックスを作成します。

AWS Management Console

1. AWS Management Consoleにサインインし、AWS Glue コンソール (https://console.aws.amazon.com/glue/) を開きます。
2. [Data Catalog] で [クローラー] を選択します。

3. クローラーを定義すると、[出力とスケジュールを設定する] ページの [詳細オプション] で、[パーティションインデックスを自動的に作成] オプションがデフォルトで有効になります。

   このオプションを無効にするには、コンソールの [パーティションインデックスを自動的に作成] チェックボックスの選択を解除できます。

4. クローラー設定が完了したら、[クローラーの作成] を選択します。

AWS CLI

AWS CLI を使用し、`configuration` パラメータで `CreatePartitionIndex` を設定してこのオプションを無効にすることもできます。デフォルト値は True です。

```
aws glue update-crawler \
    --name myCrawler \
    --configuration '{"Version": 1.0, "CreatePartitionIndex": false }'
```

パーティションインデックスの使用に関する注意事項

- クローラーによって作成されたテーブルには、デフォルトでは `partition_filtering.enabled` 変数がありません。詳細については、「AWS Glue パーティションインデックスとフィルタリング」を参照してください。
- 暗号化されたパーティションのパーティションインデックスの作成はサポートされていません。

クローラーによる既存のスキーマの変更を防止する

実行時に、AWS Glue クローラー が Data Catalog に対してスキーマの変更を行わないようにすることができます。デフォルトでは、クローラーは Data Catalog 内のスキーマを更新して、クロールするデータソースと一致させます。ただし、特にデータを変換またはクリーンアップし、元のスキーマに変更を上書きしない場合は、クローラーが既存のスキーマを変更できないようにする必要があります。

テーブル定義の既存のスキーマを上書きしないようにクローラーを設定するには、次の手順に従います。

AWS Management Console

1. AWS Management Consoleにサインインし、AWS Glue コンソール (https://console.aws.amazon.com/glue/) を開きます。
2. [Data Catalog] で [クローラー] を選択します。
3. リストからクローラーを選択したら、[編集] を選択します。
4. [ステップ 4: 出力とスケジュールを設定する] を選択します。
5. [詳細オプション] で、[新しい列のみを追加] または [変更を無視し、Data Catalog のテーブルを更新しない] を選択します。
6. すべての新規パーティションと既存パーティションをテーブルのメタデータで更新するように設定オプションを設定できます。この設定により、パーティションスキーマがテーブルから継承されます。
7. [Update] (更新) を選択します。

AWS CLI

次の例は、既存のスキーマを変更せず、新しい列のみを追加するようにクローラーを設定する方法を示しています。

```
aws glue update-crawler \
  --name myCrawler \
  --configuration '{"Version": 1.0, "CrawlerOutput": {"Tables":
 {"AddOrUpdateBehavior": "MergeNewColumns"}}}'
```

次の例は、既存のスキーマを変更せず、新しい列を追加しないようにクローラーを設定する方法を示しています。

```
aws glue update-crawler \
  --name myCrawler \
  --schema-change-policy UpdateBehavior=LOG \
  --configuration '{"Version": 1.0, "CrawlerOutput": {"Partitions":
 { "AddOrUpdateBehavior": "InheritFromTable" }}}'
```

API

クローラーの実行時にテーブルスキーマを一切変更しない場合は、スキーマ変更ポリシーを LOG に設定します。

API を使用してクローラーを設定する場合は、以下のパラメータを設定します。

- SchemaChangePolicy 構造の UpdateBehavior フィールドを LOG に設定します。
- クローラー API で次の JSON オブジェクトの文字列表現を使用して Configuration フィールドを設定します。

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

各 Amazon S3 インクルードパスの単一のスキーマを作成する

デフォルトでは、Amazon S3 に保存されたデータのテーブルをクローラーが定義するときに、データの互換性とスキーマの類似性の両方が考慮されます。考慮されるデータ互換性要因には、データが同じ形式 (JSON など) かどうか、同じ圧縮タイプ (GZIP など) かどうか、Amazon S3 パスの構造、他のデータ属性などです。スキーマの類似性は、別個の Amazon S3 オブジェクトのスキーマがどれくらい似ているかの尺度です。

このオプションを説明するため、インクルードパス s3://amzn-s3-demo-bucket/table1/ を使用してクローラーを定義するとします。クローラーが実行されると、次の特性を持つ 2 つの JSON ファイルが検索されます。

- ファイル 1 – S3://amzn-s3-demo-bucket/table1/year=2017/data1.json
- ファイルのコンテンツ – {“A”: 1, “B”: 2}
- スキーマ – A:int, B:int

- ファイル 2 – S3://amzn-s3-demo-bucket/table1/year=2018/data2.json
- ファイルのコンテンツ – {“C”: 3, “D”: 4}
- スキーマ – C: int, D: int

デフォルトでは、スキーマの類似性が十分ではないため、クローラーは `year_2017` および `year_2018` という 2 つのテーブルを作成します。ただし、オプション [Create a single schema for each S3 path (各 S3 パスの単一のスキーマを作成する)] を選択し、データに互換性がある場合、クローラーは 1 つのテーブルを作成します。テーブルにはスキーマ `A:int,B:int,C:int,D:int` と `partitionKey year:string` があります。

AWS Management Console

1. AWS Management Consoleにサインインし、AWS Glue コンソール (https://console.aws.amazon.com/glue/) を開きます。
2. [Data Catalog] で [クローラー] を選択します。
3. 新しいクローラーを設定するときは、[出力およびスケジューリング] で、[詳細オプション] の [各 S3 パスの単一のスキーマを作成する] オプションを選択します。

AWS CLI

可能な場合は、`CombineCompatibleSchemas` へのクローラーを共通テーブル定義に設定できます。このオプションを使用しても、クローラーはデータ互換性を考慮に入れますが、指定されたインクルードパスで Amazon S3 オブジェクトを評価するときに特定のスキーマの類似性を無視します。

AWS CLI を使用してクローラーを設定する場合は、以下の設定オプションを設定します。

```
aws glue update-crawler \
   --name myCrawler \
   --configuration '{"Version": 1.0, "Grouping": {"TableGroupingPolicy":
 "CombineCompatibleSchemas" }}'
```

API

API を使用してクローラーを設定する場合は、以下の設定オプションを設定します。

クローラー API で次の JSON オブジェクトの文字列表現を使用して `Configuration` フィールドを設定します。

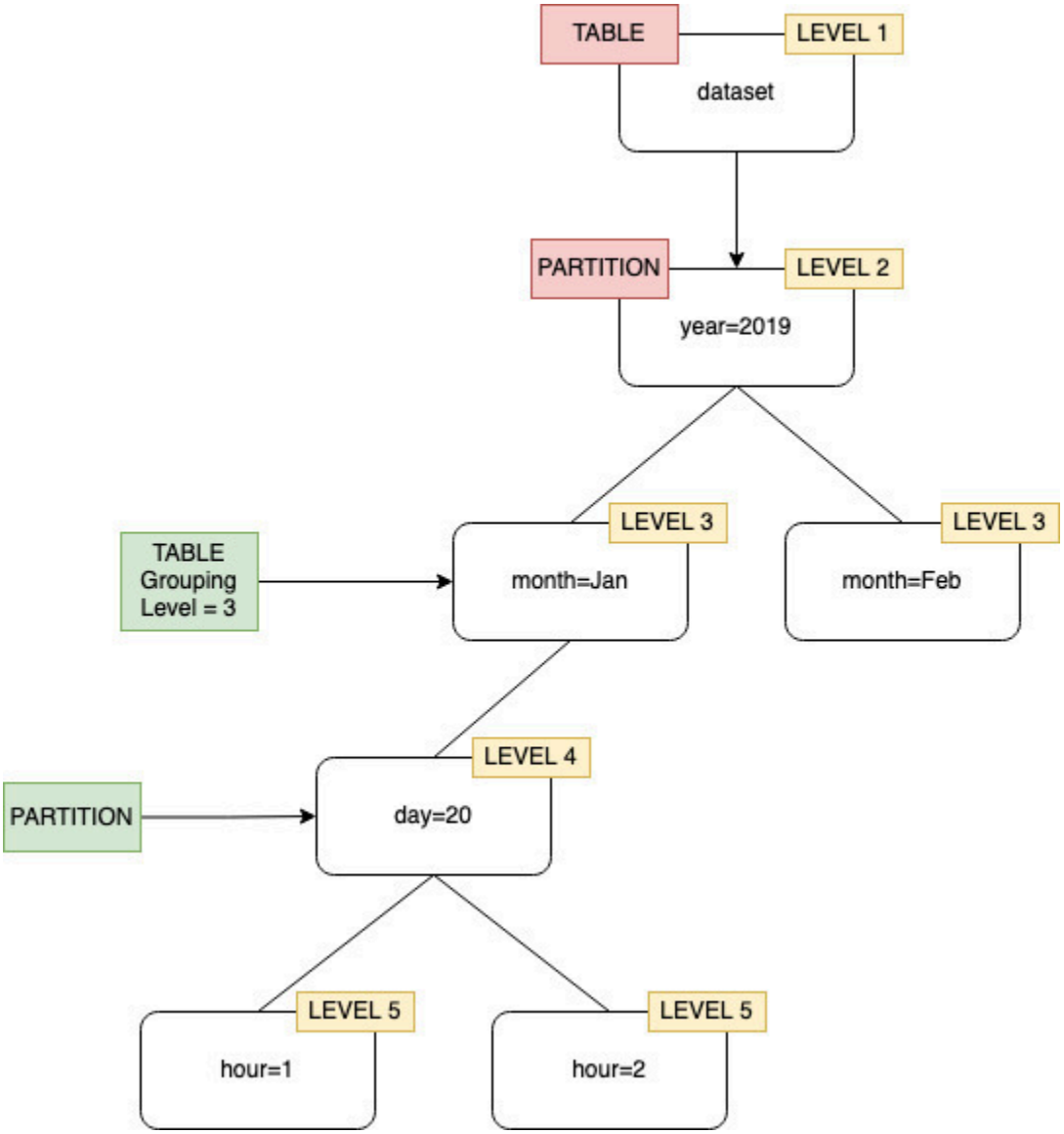
```
{
   "Version": 1.0,
   "Grouping": {
```

```
      "TableGroupingPolicy": "CombineCompatibleSchemas" }
}
```

テーブルの場所とパーティションレベルの指定

デフォルトでは、Amazon S3 に保存されたデータのテーブルをクローラーが定義するときに、クローラーはスキーマを結合して最上位テーブル (`year=2019`) を作成しようとします。場合によっては、フォルダ `month=Jan` のテーブルをクローラーが作成することを期待することがありますが、兄弟フォルダ (`month=Mar`) が同じテーブルにマージされているので、代わりにクローラーはパーティションを作成します。

テーブルレベルのクローラーオプションを使用すると、クローラーにテーブルの配置場所やパーティションの作成方法を柔軟に指定できます。テーブルレベル を指定すると、その絶対レベルにAmazon S3 バケットからテーブルが作成されます。

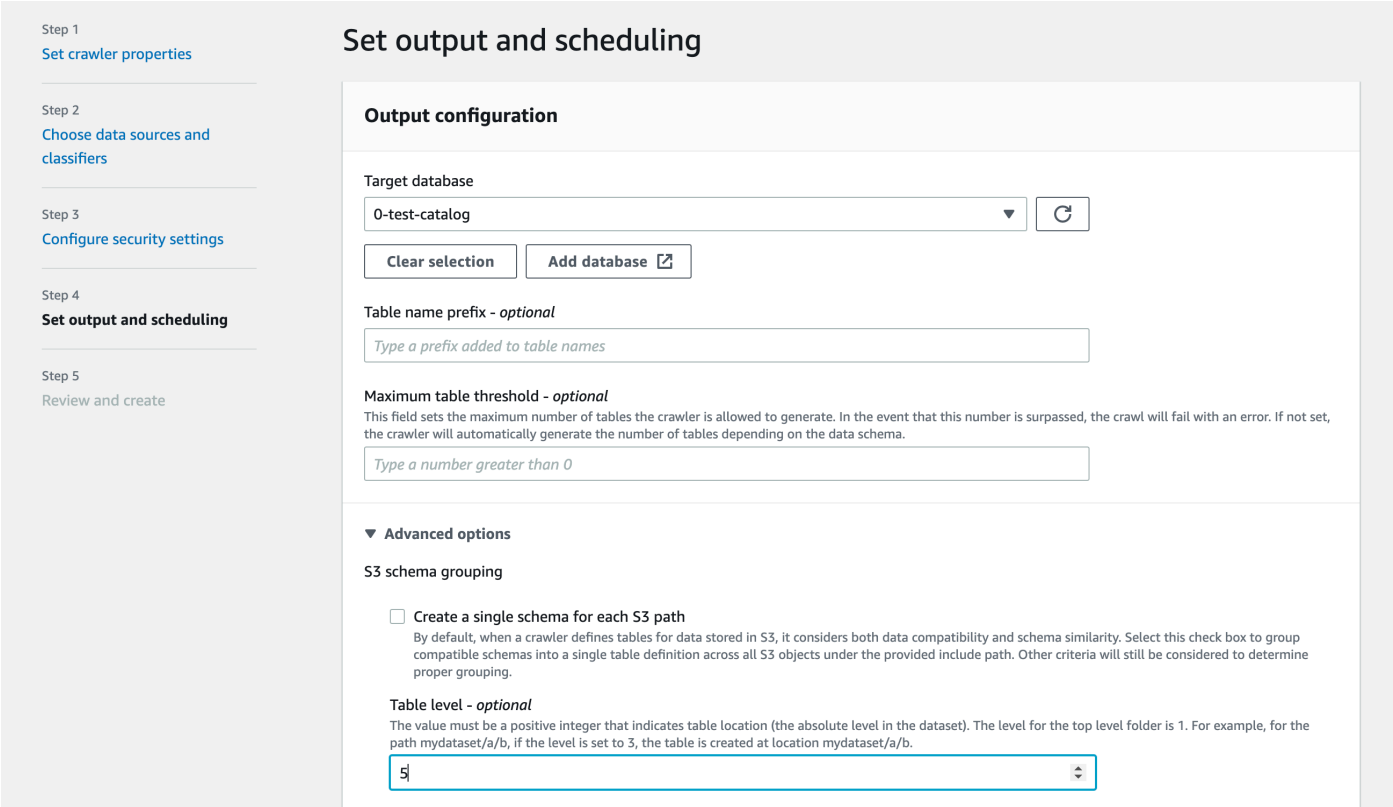


コンソールでクローラーを設定するとき、テーブルレベル クローラーオプションの値を指定できます。値は、テーブルの場所 (データセット内の絶対レベル) を示す正の整数である必要があります。最上位レベルのフォルダのレベルは 1 です。例えば、`mydataset/year/month/day/hour` というパスで、レベルが 3 に設定されている場合、テーブルは `mydataset/year/month` という場所に作成されます。

AWS Management Console

1. AWS Management Consoleにサインインし、AWS Glue コンソール (https://console.aws.amazon.com/glue/) を開きます。
2. [Data Catalog] で [クローラー] を選択します。

3. クローラーを設定するときは、[出力とスケジューリング] で、[詳細オプション] の [テーブルレベル] を選択します。



AWS CLI

AWS CLI を使用してクローラーを設定するときは、次のサンプルコードに示すように `configuration` パラメータを設定します。

```
aws glue update-crawler \
  --name myCrawler \
  --configuration '{"Version": 1.0, "Grouping": { "TableLevelConfiguration": 2 }}'
```

API

API を使用してクローラー設定をすると、次の JSON オブジェクトの文字列表現をする `Configuration` フィールドを設定します。例:

```
configuration = jsonencode(
{
   "Version": 1.0,
   "Grouping": {
```

```
            TableLevelConfiguration = 2
        }
})
```

CloudFormation

この例では、CloudFormation テンプレート内のコンソールで使用できる [Table Level] (テーブルレベル) オプションを設定します。

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

クローラーが作成できるテーブルの最大数を指定する

(オプション) AWS Glue コンソールまたは AWS CLI を使用して `TableThreshold` を指定することで、クローラーが作成できるテーブルの最大数を指定できます。クローリング中にクローラーによって検出されたテーブルがこの入力値より多い場合、クローリングが失敗し、データカタログにデータが書き込まれません。

このパラメータは、クローラーによって検出および作成されるテーブルが予想よりもはるかに多い場合に役立ちます。これには、以下のような複数の理由が考えられます。

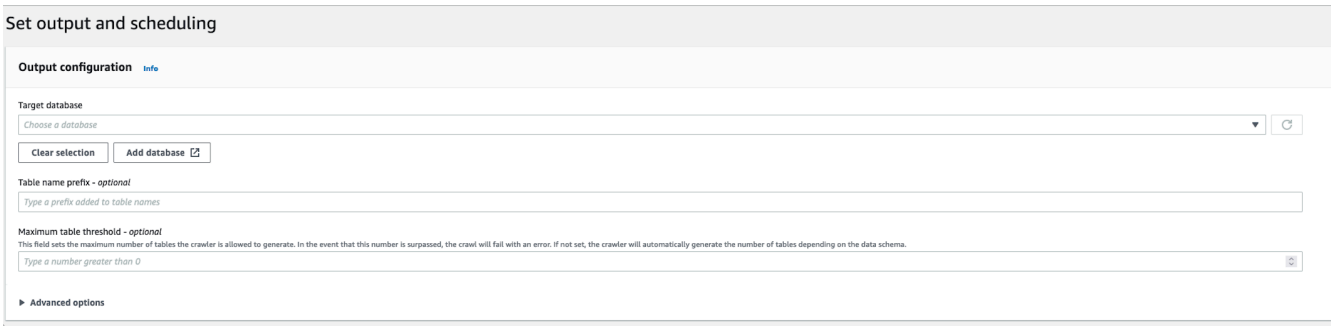
- AWS Glue ジョブを使用して Amazon S3 ロケーションに入力する場合、フォルダと同じレベルに空のファイルができる可能性があります。このような場合、この Amazon S3 ロケーションでクローラーを実行すると、ファイルとフォルダが同じレベルに存在するため、クローラーは複数のテーブルを作成します。
- `"TableGroupingPolicy": "CombineCompatibleSchemas"` を設定しない場合、予想よりも多くのテーブルが作成される可能性があります。

`TableThreshold` を 1 以上の整数値として指定します。この値はクローラーごとに設定されます。つまり、クローリングごとに、この値が考慮されます。例えば、あるクローラーでは `TableThreshold` 値を 5 に設定したとします。各クローリングで、AWS Glue は検出されたテーブルの数をこのテーブルのしきい値 (5) と比較します。検出されたテーブルの数が 5 未満の場合、AWS Glue はデータカタログにテーブルを書き込み、そうでない場合、データカタログに書き込むことなくクローリングが失敗します。

AWS Management Console

AWS Management Console を使用して **TableThreshold** を設定する場合は、次のようになります。

1. AWS Management Consoleにサインインし、AWS Glue コンソール (https://console.aws.amazon.com/glue/) を開きます。
2. クローラーを設定するときに、[出力とスケジューリング] で、クローラーが生成できるテーブルの数に最大テーブルしきい値を設定します。



AWS CLI

AWS CLI を使用して TableThreshold を設定する場合は、次のようになります。

```
aws glue update-crawler \
    --name myCrawler \
    --configuration '{"Version": 1.0, "CrawlerOutput": {"Tables":
 { "TableThreshold": 5 }}}'
```

API

API を使用して TableThreshold を設定する場合は、次のようになります。

```
"{"Version":1.0,
"CrawlerOutput":
{"Tables":{"AddOrUpdateBehavior":"MergeNewColumns",
"TableThreshold":5}}}";
```

エラーメッセージはログに記録され、テーブルパスを特定してデータをクリーンアップするのに役立ちます。テーブル数が指定したテーブルしきい値を超えているためにクローラーが失敗した場合は、次のようなログの例がアカウントで確認できます。

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

CloudWatch では、検出されたすべてのテーブルの場所を INFO メッセージとしてログに記録します。失敗の理由としてエラーがログに記録されます。

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

Lake Formation 認証情報を使用するようにクローラーを設定する

AWS Lake Formation 認証情報を使用するようにクローラーを設定すると、同じ AWS アカウント または別の AWS アカウント 内の基盤となる Amazon S3 ロケーションで Amazon S3 データストアまたはデータカタログテーブルにアクセスできます。クローラーとデータカタログテーブルが同じアカウントに存在する場合は、既存のデータカタログテーブルをクローラーのターゲットとして設定できます。現在、Data Catalog テーブルをクローラーのターゲットとして使用する場合、単一のカタログテーブルを含む単一のカタログターゲットのみが許可されています。

> **Note**
>
> データカタログテーブルをクローラーターゲットとして定義する場合、データカタログテーブルの基盤となるロケーションが Amazon S3 ロケーションであることを確認してください。Lake Formation の認証情報を使用するクローラーは、基盤となる Amazon S3 ロケーションを備えたデータカタログターゲットのみをサポートしています。

クローラーと登録された Amazon S3 ロケーションまたはデータカタログテーブルが同じアカウントに存在する場合は、セットアップが必要になります (アカウント内クローリング)

クローラーが Lake Formation の認証情報を使用してデータストアまたはデータカタログテーブルにアクセスできるようにするには、データロケーションを Lake Formation で登録する必要があります。また、クローラーの IAM ロールには、Amazon S3 バケットが登録されている送信先からデータを読み込む許可が付与されている必要があります。

AWS Management Console または AWS Command Line Interface (AWS CLI) を使用して、次の設定手順を完了できます。

AWS Management Console

1. クローラーソースにアクセスするようにクローラーを設定する前に、データストアまたはデータカタログのデータロケーションを Lake Formation で登録します。Lake Formation コンソール (https://console.aws.amazon.com/lakeformation/) で、クローラーが定義されている AWS アカウント で Amazon S3 ロケーションをデータレイクのルートロケーションとして登録します。詳細については、「Amazon S3 ロケーションの登録」を参照してください。
2. クローラーの実行に使用される IAM ロールにデータロケーション許可を付与し、クローラーが Lake Formation の送信先からデータを読み込めるようにします。詳細については、「データロケーション許可の付与 (同じアカウント)」を参照してください。
3. クローラーロールにデータベースへのアクセス権限 (`Create`) を付与します。 このデータベースは出力データベースとして指定されています。詳細については、「Lake Formation コンソールと名前付きリソース方式を使用したデータベース許可の付与」を参照してください。
4. IAM コンソール (https://console.aws.amazon.com/iam/) で、クローラー用の IAM ロールを作成します。ロールに `lakeformation:GetDataAccess` ポリシーを追加します。
5. AWS Glue コンソール (https://console.aws.amazon.com/glue/) で、クローラーを設定する際に、オプション [Use Lake Formation credentials for crawling Amazon S3 data source] (Amazon S3 データソースのクローリングに Lake Formation の認証情報を使用する) を選択してください。

> **Note**
>
> accountId フィールドは、アカウント内クローリングのためのオプションです。

AWS CLI

```
aws glue --profile demo create-crawler --debug --cli-input-json '{
    "Name": "prod-test-crawler",
    "Role": "arn:aws:iam::111122223333:role/service-role/AWSGlueServiceRole-prod-
test-run-role",
    "DatabaseName": "prod-run-db",
    "Description": "",
    "Targets": {
    "S3Targets":[
```

```
                {
                 "Path": "s3://amzn-s3-demo-bucket"
                }
                ]
                },
   "SchemaChangePolicy": {
      "UpdateBehavior": "LOG",
      "DeleteBehavior": "LOG"
  },
  "RecrawlPolicy": {
    "RecrawlBehavior": "CRAWL_EVERYTHING"
  },
  "LineageConfiguration": {
    "CrawlerLineageSettings": "DISABLE"
  },
  "LakeFormationConfiguration": {
    "UseLakeFormationCredentials": true,
    "AccountId": "111122223333"
  },
  "Configuration": {
           "Version": 1.0,
           "CrawlerOutput": {
             "Partitions": { "AddOrUpdateBehavior": "InheritFromTable" },
             "Tables": {"AddOrUpdateBehavior": "MergeNewColumns" }
           },
           "Grouping": { "TableGroupingPolicy": "CombineCompatibleSchemas" }
         },
  "CrawlerSecurityConfiguration": "",
  "Tags": {
    "KeyName": ""
  }
}'
```

クローラーと登録された Amazon S3 ロケーションが異なるアカウントに存在する場合は、セットアップが必要になります (クロスアカウントクローリング)

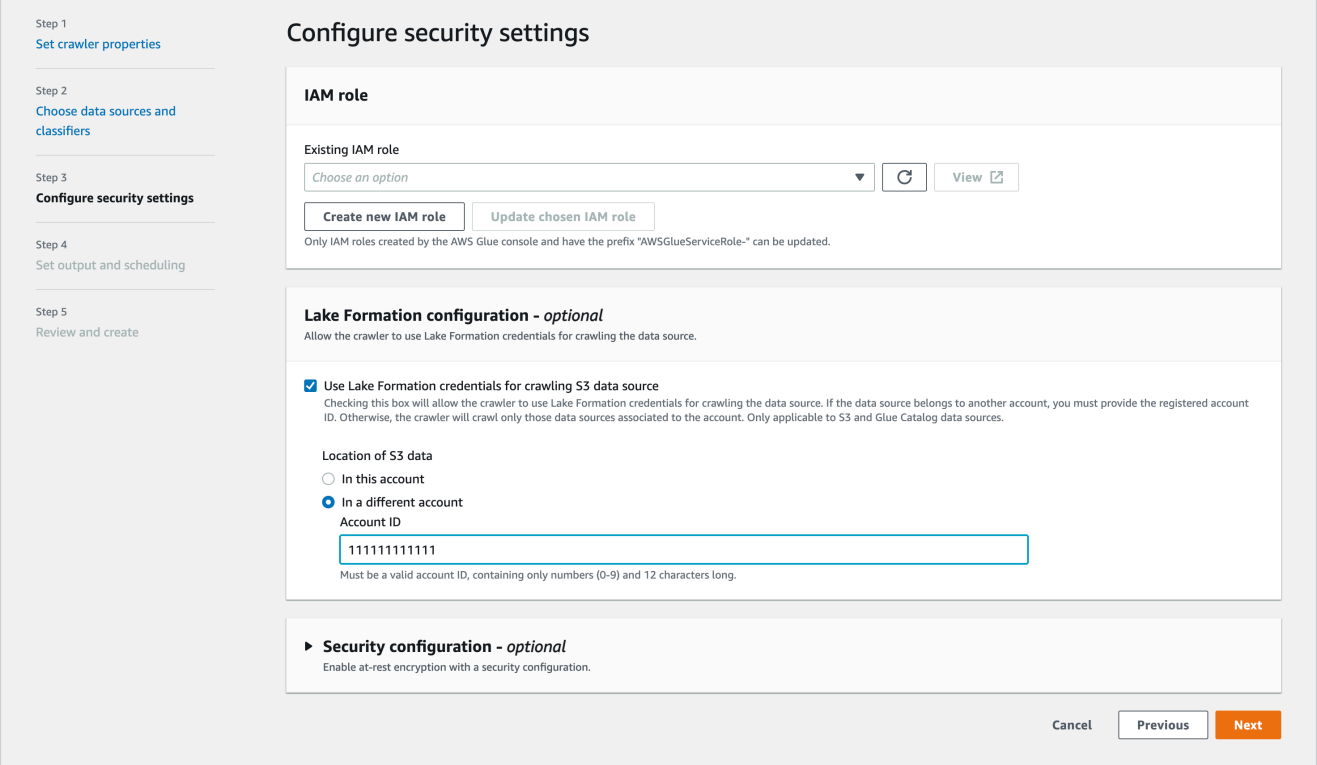
クローラーが Lake Formation の認証情報を使用して別のアカウントのデータストアにアクセスできるようにするには、まず Amazon S3 のデータロケーションを Lake Formation で登録する必要があります。次に、以下の手順を実行して、クローラーのアカウントにデータロケーション許可を付与します。

AWS Management Console または AWS CLI を使用して、次のステップを実行できます。

AWS Management Console

1. Amazon S3 ロケーションが登録されているアカウント (アカウント B) では、次の操作を行います。
   a. Lake Formation で Amazon S3 パスを登録します。詳細については、「Amazon S3 ロケーションの登録」を参照してください。
   b. クローラーを実行するアカウント (アカウント A) にデータロケーション許可を付与します。詳細については、「Grant data location permissions」(データロケーション許可を付与する) を参照してください。
   c. 基盤となるロケーションをターゲットの Amazon S3 ロケーションとして、Lake Formation に空のデータベースを作成します。詳細については、「データベースの作成」を参照してください。
   d. アカウント A (クローラーを実行するアカウント) に、前のステップで作成したデータベースへのアクセス権を付与します。詳細については、「Granting database permissions」(データベース許可の付与) を参照してください。
2. クローラーが作成され実行されるアカウント (アカウント A) では、次の操作を行います。
   a. AWS RAM コンソールを使用して、外部アカウント (アカウント B) から共有されたデータベースを受け入れます。詳細については、「AWS Resource Access Manager からのリソース共有招待の承諾」を参照してください。
   b. クローラー用の IAM ロールを作成します。ロールに `lakeformation:GetDataAccess` ポリシーを追加します。
   c. Lake Formation コンソール (https://console.aws.amazon.com/lakeformation/) で、ターゲットの Amazon S3 ロケーションに対するデータロケーション許可をクローラーの実行に使用される IAM ロールに付与して、クローラーが Lake Formation の送信先からデータを読み込めるようにします。詳細については、「Granting data location permissions」(データロケーション許可の付与) を参照してください。
   d. 共有データベースでリソースリンクを作成します。詳細については、「Create a resource link」(リソースリンクを作成する) を参照してください。
   e. クローラーロールに共有データベースと (`Describe`) リソースリンクに対するアクセス許可 (`Create`) を付与します。リソースリンクはクローラーの出力で指定されます。
   f. AWS Glue コンソール (https://console.aws.amazon.com/glue/) で、クローラーを設定する際に、オプション [Use Lake Formation credentials for crawling Amazon S3 data source] (Amazon S3 データソースのクローリングに Lake Formation の認証情報を使用する) を選択してください。

クロスアカウントクローリングの場合は、ターゲットの Amazon S3 ロケーションが Lake Formation で登録されている AWS アカウント ID を指定します。アカウント内クローリングの場合、accountId フィールドはオプションです。



AWS CLI

```
aws glue --profile demo create-crawler --debug --cli-input-json '{
    "Name": "prod-test-crawler",
    "Role": "arn:aws:iam::111122223333:role/service-role/AWSGlueServiceRole-prod-
test-run-role",
    "DatabaseName": "prod-run-db",
    "Description": "",
    "Targets": {
    "S3Targets":[
                {
                 "Path": "s3://amzn-s3-demo-bucket"
                }
                ]
                },
   "SchemaChangePolicy": {
      "UpdateBehavior": "LOG",
      "DeleteBehavior": "LOG"
```

```
  },
  "RecrawlPolicy": {
    "RecrawlBehavior": "CRAWL_EVERYTHING"
  },
  "LineageConfiguration": {
    "CrawlerLineageSettings": "DISABLE"
  },
  "LakeFormationConfiguration": {
    "UseLakeFormationCredentials": true,
    "AccountId": "111111111111"
  },
  "Configuration": {
           "Version": 1.0,
           "CrawlerOutput": {
             "Partitions": { "AddOrUpdateBehavior": "InheritFromTable" },
             "Tables": {"AddOrUpdateBehavior": "MergeNewColumns" }
           },
           "Grouping": { "TableGroupingPolicy": "CombineCompatibleSchemas" }
         },
  "CrawlerSecurityConfiguration": "",
  "Tags": {
    "KeyName": ""
  }
}'
```

Note

- Lake Formation 認証情報を使用するクローラーは、Amazon S3 およびデータカタログターゲットでのみサポートされます。
- Lake Formation 認証情報供給を使用するターゲットの場合、基盤となる Amazon S3 ロケーションは同じバケットに属している必要があります。例えば、すべてのターゲットロケーションが同じバケット (amzn-s3-demo-bucket1) にある限り、ユーザーは複数のターゲット (s3://amzn-s3-demo-bucket1/folder1、s3://amzn-s3-demo-bucket1/folder2) を使用できます。別のバケット (s3://amzn-s3-demo-bucket1/folder1、s3://amzn-s3-demo-bucket2/folder2) を指定することはできません。
- 現在、データカタログのターゲットクローラーでは、単一のカタログテーブルを含む単一のカタログターゲットのみが許可されています。

Amazon S3 イベント通知を使用した加速クロール

Amazon S3 または Data Catalog ターゲットからオブジェクトを一覧表示する代わりに、Amazon S3 イベントを使用して変更を検索するようにクローラーを設定できます。この機能は、Amazon S3 または Data Catalog ターゲット全体を一覧表示するのではなく、Amazon S3 イベントを使用してイベントをトリガーしたサブフォルダからのすべてのファイルを一覧表示して 2 つのクロール間の変更を識別することによって、再クロール時間を短縮します。

最初のクロールでは、ターゲットからのすべての Amazon S3 オブジェクトを一覧表示します。最初のクロールの成功後、手動または設定されたスケジュールでリクロールを選択できます。クローラーは、すべてのオブジェクトをリストするのではなく、それらのイベントのオブジェクトのみをリストします。

ターゲットが Data Catalog テーブルの場合、クローラーは変更内容 (テーブル内の追加パーティションなど) で Data Catalog 内の既存のテーブルを更新します。

Amazon S3 イベントベースのクローラーに移行する利点は以下のとおりです。

- ターゲットからのすべてのオブジェクトの一覧表示を要しない場合は、より速く再クロールできます。その代わりに、オブジェクトが追加または削除される特定のフォルダが一覧表示されます。
- オブジェクトが追加または削除される特定のフォルダの一覧表示を行うと、全体的なクロールコストが削減されます。

Amazon S3 イベントクロールは、クローラーのスケジュールに基づいて SQS キューから Amazon S3 イベントを使うことで実行します。キューにイベントがない場合、費用はかかりません。Amazon S3 イベントは、SQS キューに直接送信するように設定できます。また、複数のコンシューマーが同じイベント、SNS と SQS の組み合わせを必要とする場合にも設定できます。詳細については、「the section called “Amazon S3 イベント通知のアカウントを設定します。”」を参照してください。

イベントモードでクローラーを作成して設定した後の最初のクロールは、Amazon S3 または Data Catalog ターゲットの完全な一覧表示を行う一覧表示モードで実行されます。次のログは、最初に成功したクロール、「クロールは Amazon S3 イベントを使用して実行します」の後に、Amazon S3 イベントを使用してクロールのオペレーションを確認します。

Amazon S3 イベントクロールを作成し、クロールに影響を与える可能性のあるクローラーのプロパティを更新すると、クロールがリストモードで動作し、「クロールは S3 イベントモードで実行されていません」というログが追加されます。

> **Note**
>
> クロールごとに消費するメッセージの最大数は 100,000 メッセージです。

考慮事項と制限事項

Amazon S3 イベント通知を使用して変更を検索するようにクローラーを設定する場合、以下の考慮事項と制限が適用されます。

- 削除されたパーティションの重要な動作

  データカタログテーブルで Amazon S3 イベントクローラーを使用する場合:

  - `DeletePartition` API コールを使用してパーティションを削除する場合、そのパーティション内のすべての S3 オブジェクトも削除し、さらに S3 イベント通知を設定するときに [すべてのオブジェクト削除イベント] を選択する必要があります。削除イベントが設定されていない場合、クローラは次の実行時に削除されたパーティションを再作成します。
- ターゲットが Amazon S3 または Data Catalog であるかにかかわらず、クローラーがサポートするのは単一のターゲットのみです。
- プライベート VPC の SQS はサポートされていません。
- Amazon S3 サンプリングはサポートされていません。
- クローラーターゲットは、Amazon S3 ターゲットの場合はフォルダ、Data Catalog ターゲットの場合は 1 つ、または複数の AWS Glue Data Catalog テーブルにする必要があります。
- 「すべての」パスのワイルドカードをサポートしていません: s3: //%
- Data Catalog ターゲットの場合、Amazon S3 イベントモードでは、すべてのカタログテーブルが同じ Amazon S3 バケットをポイントする必要があります。
- Data Catalog ターゲットの場合、カタログテーブルは Delta Lake 形式の Amazon S3 ロケーションをポイントしない必要があります (_symlink フォルダが含まれる、またはカタログテーブルの `InputFormat` を確認)。

トピック

- Amazon S3 イベント通知のアカウントを設定します。
- Amazon S3 ターゲットの Amazon S3 イベント通知用のクローラーを設定する
- Data Catalog テーブルの Amazon S3 イベント通知用のクローラーを設定する

Amazon S3 イベント通知のアカウントを設定します。

以下の設定タスクを実行します。括弧内の値は、スクリプトの構成可能な設定を参照している点に注意してください。

1. Amazon S3 バケットのイベント通知を設定する必要があります。

   詳細については、「Amazon S3 イベント通知」を参照してください。

2. Amazon S3 イベントベースのクローラーを使用するには、S3 ターゲットと同じプレフィックスからフィルタリングされたイベントが含まれた Amazon S3 バケットでイベント通知を有効にし、SQS に保存する必要があります。SQS とイベント通知は、「チュートリアル: 通知のバケットを設定する」の手順に従って、コンソールから設定できます。

3. クローラーが使用するロールに次の SQS ポリシーを追加します。

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

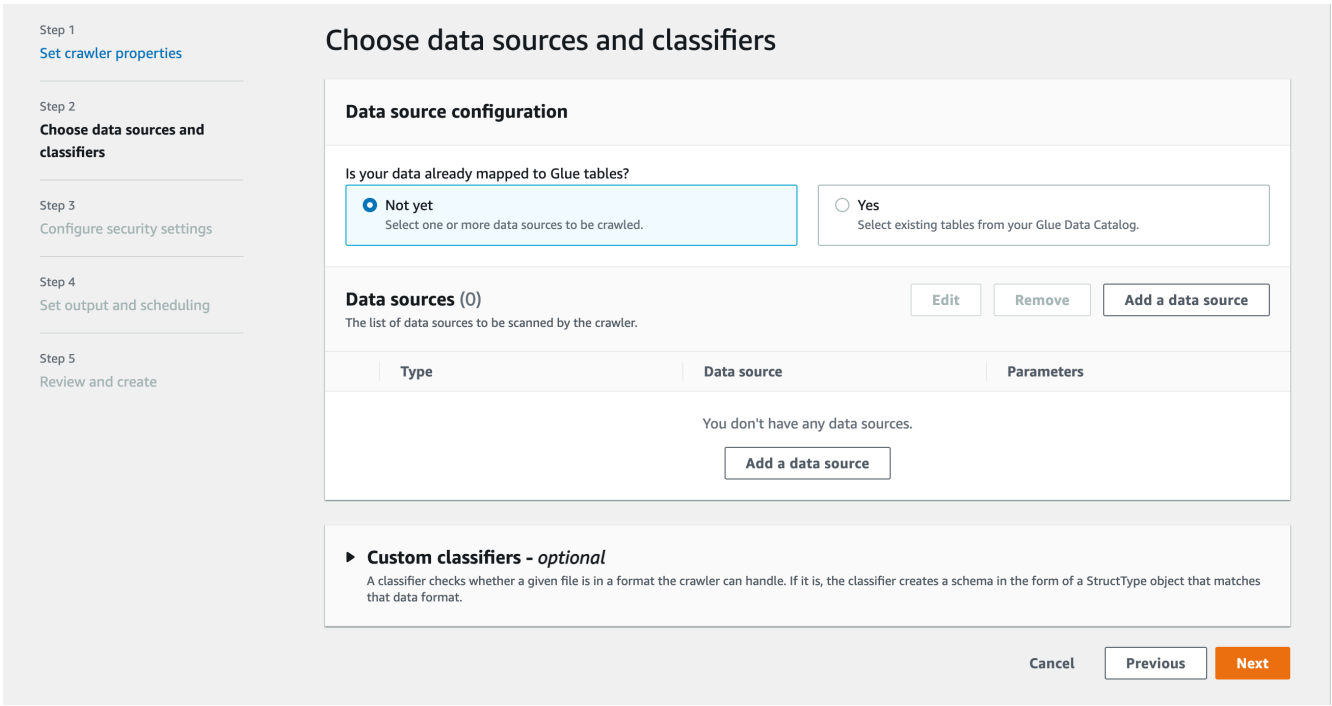
Amazon S3 ターゲットの Amazon S3 イベント通知用のクローラーを設定する

AWS Management Console または AWS CLI を使用して Amazon S3 ターゲットの Amazon S3 イベント通知用のクローラーを設定するには、以下の手順を実行します。

AWS Management Console

1. AWS Management Console にサインインし、https://console.aws.amazon.com/guardduty/ で GuardDuty コンソールを開きます。
2. クローラーのプロパティを設定します。詳細については、「AWS Glue コンソールでのクローラー設定オプションの設定」を参照してください。
3. [データソースの設定] セクションに、[データは AWS Glue テーブルにマッピング済みですか?] という質問が表示されています。

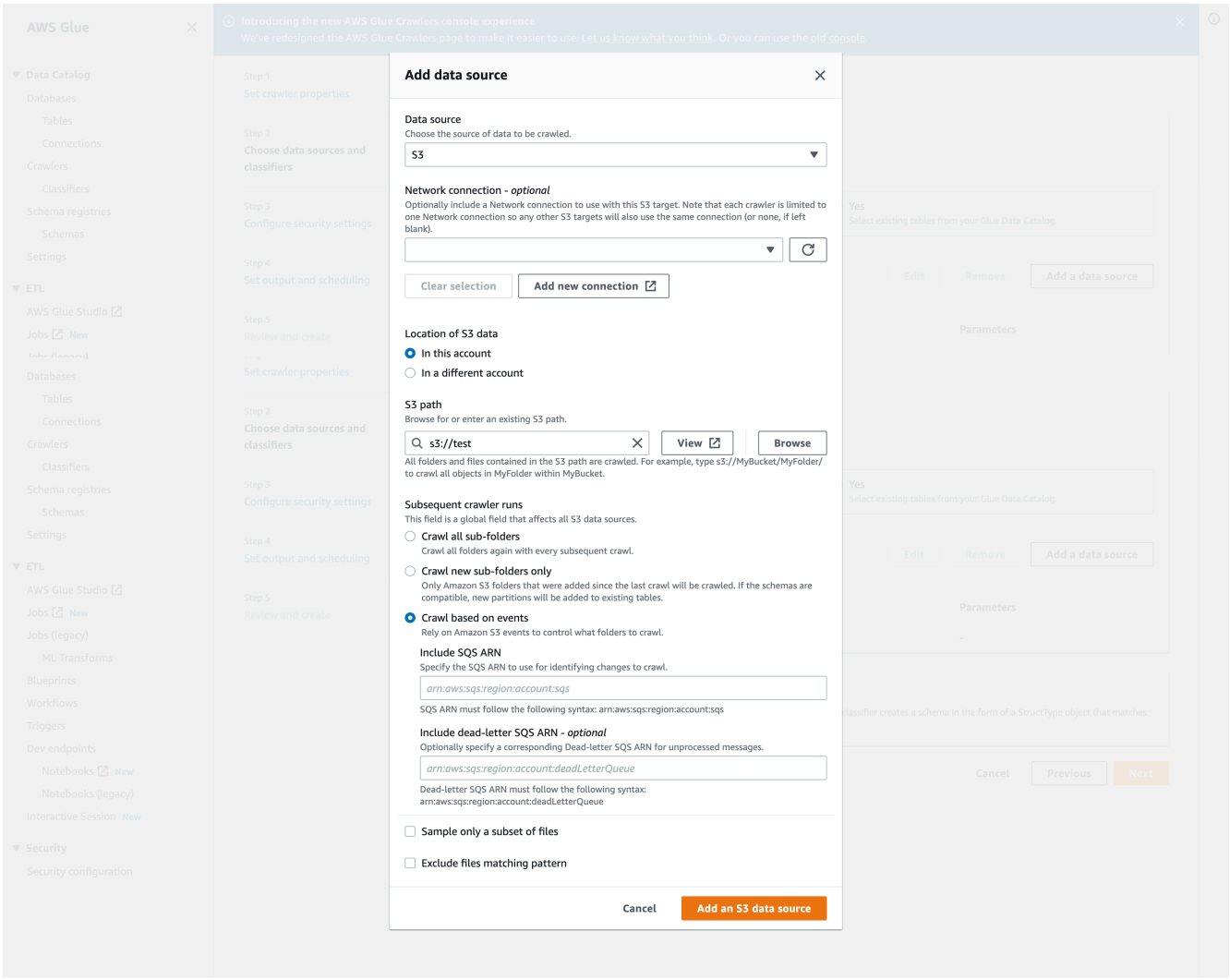
   デフォルトでは、[Not yet] (まだです) が選択されています。Amazon S3 のデータソースを使用しており、データがまだ AWS Glue テーブルにマップされていないため、これはデフォルトままにしておきます。
4. [Data sources] (データソース) セクションで、[Add a data source] (データソースを追加) を選択します。



5. [Add data source] (データソースの追加) ダイアログで、Amazon S3 データソースを以下のように設定します。

   - [Data source] (データソース): デフォルトで、Amazon S3 が選択されています。
   - [Network connection] (ネットワーク接続) (オプション): [Add new connection] (新しい接続を追加) を選択します。

- [Location of Amazon S3 data] (Amazon S3 データの場所): デフォルトで、[In this account] (このアカウント内) が選択されています。
- [Amazon S3 path] (Amazon S3 パス): フォルダとファイルがクロールされる Amazon S3 パスを指定します。
- [Subsequent crawler runs] (それ以降のクローラー実行): クローラーに関する Amazon S3 イベント通知を使用するには、[Crawl based on events] (イベントに基づくクロール) を選択します。
- [Include SQS ARN] (SQS ARN を含める): 有効な SQS ARN を含むデータストアパラメータを指定します。(例えば、`arn:aws:sqs:region:account:sqs`)
- [Include dead-letter SQS ARN] (配信不能 SQS ARN を含める) (オプション): 有効な Amazon 配信不能 SQS ARN を指定します。(例えば、`arn:aws:sqs:region:account:deadLetterQueue`)
- [Add an Amazon S3 data source] (Amazon S3 データソースを追加) を選択します。



AWS CLI

次に示すのは、イベント通知を使用して Amazon S3 ターゲットバケットをクロールするようにクローラーを設定するための Amazon S3 AWS CLI コールの例です。

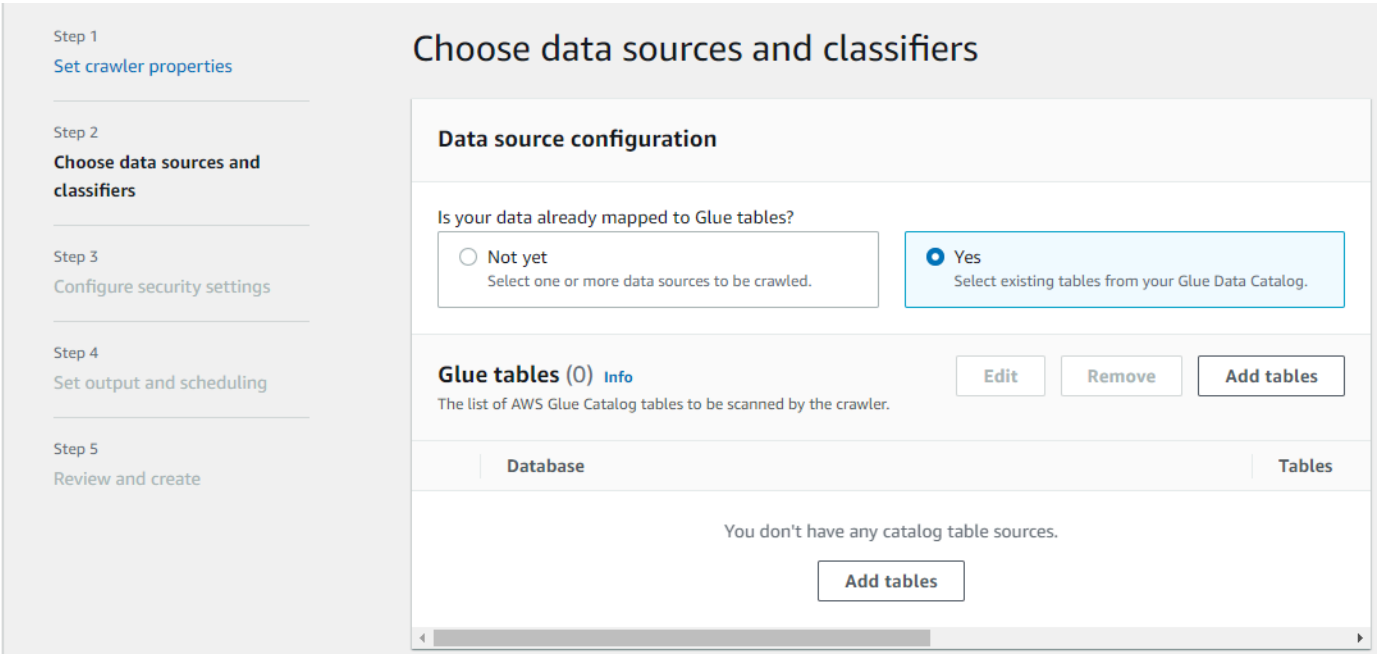
```
Create Crawler:
aws glue update-crawler \
    --name myCrawler \
    --recrawl-policy RecrawlBehavior=CRAWL_EVENT_MODE \
    --schema-change-policy UpdateBehavior=UPDATE_IN_DATABASE,DeleteBehavior=LOG
    --targets '{"S3Targets":[{"Path":"s3://amzn-s3-demo-bucket/", "EventQueueArn":
 "arn:aws:sqs:us-east-1:012345678910:MyQueue"}]}'
```

Data Catalog テーブルの Amazon S3 イベント通知用のクローラーを設定する

Data Catalog テーブルがある場合は、AWS Glue コンソールを使用して Amazon S3 イベント通知用のクローラーを設定します。

1. クローラーのプロパティを設定します。詳細については、「AWS Glue コンソールでのクローラー設定オプションの設定」を参照してください。
2. [データソースの設定] セクションに、[データは AWS Glue テーブルにマッピング済みですか?] という質問が表示されています。

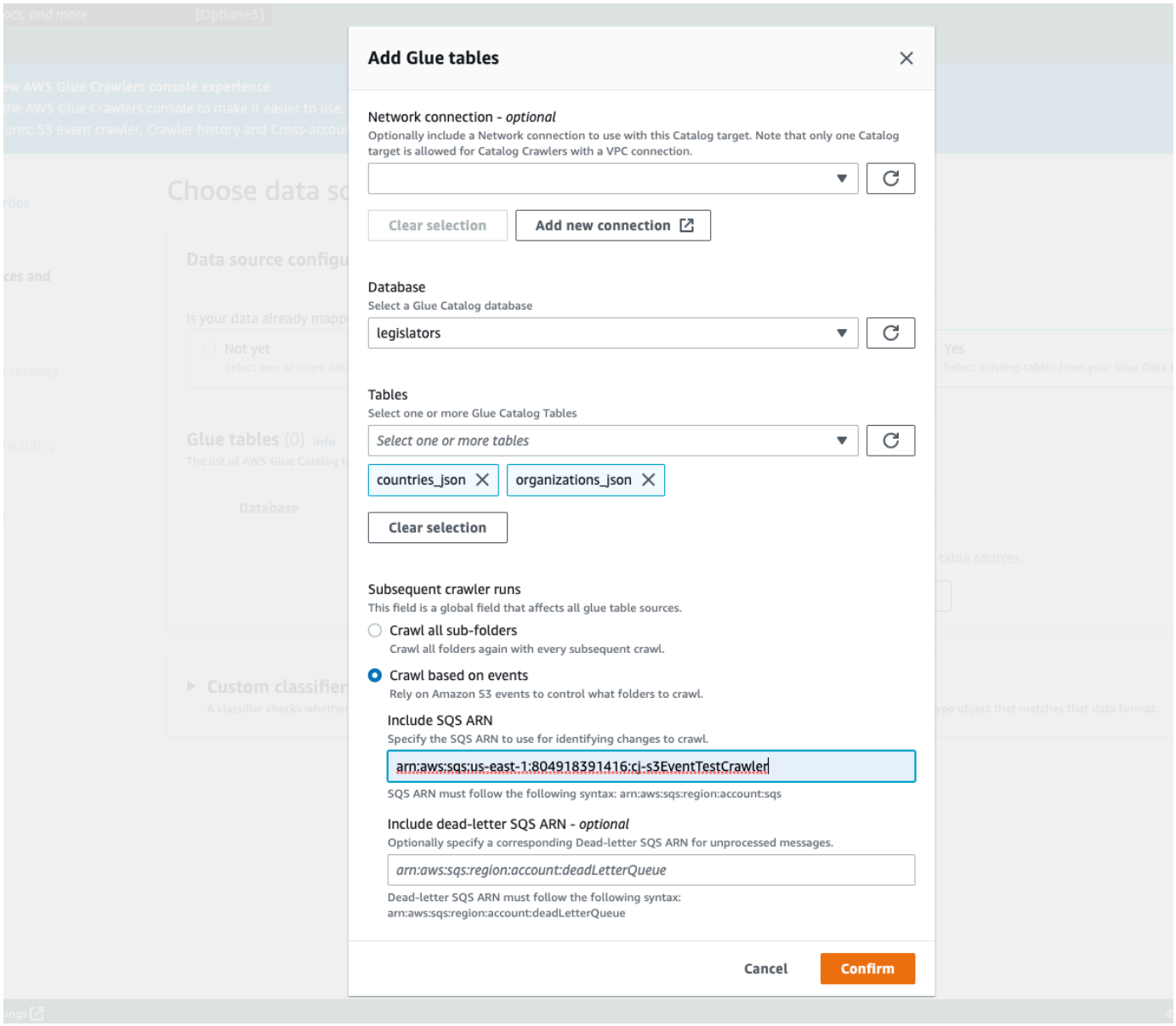
   [Yes] (はい) を選択して、Data Catalog からの既存のテーブルをデータソースとして選択します。
3. [Glue tables] (Glue テーブル) セクションで、[Add tables] (テーブルを追加する) を選択します。



4. [Add table] (テーブルを追加する) モーダルで、データベースとテーブルを設定します。

   - [Network connection] (ネットワーク接続) (オプション): [Add new connection] (新しい接続を追加) を選択します。
   - [Database] (データベース): Data Catalog 内のデータベースを選択します。

- [Tables] (テーブル): Data Catalog 内のデータベースから 1 つ、または複数のテーブルを選択します。
- [Subsequent crawler runs] (それ以降のクローラー実行): クローラーに関する Amazon S3 イベント通知を使用するには、[Crawl based on events] (イベントに基づくクロール) を選択します。
- [Include SQS ARN] (SQS ARN を含める): 有効な SQS ARN を含むデータストアパラメータを指定します。(例えば、`arn:aws:sqs:region:account:sqs`)
- [Include dead-letter SQS ARN] (配信不能 SQS ARN を含める) (オプション): 有効な Amazon 配信不能 SQS ARN を指定します。(例えば、`arn:aws:sqs:region:account:deadLetterQueue`)
- [確認] を選択します。



チュートリアル: AWS Glue クローラの追加

この AWS Glue シナリオでは、主要な航空会社の到着データを分析して、各月の出発空港の人気度を計算するよう求められています。2016 年のフライトデータが CSV 形式で Amazon S3 に保存されています。データの変換と分析を行う前に、AWS Glue Data Catalog 内のそのメタデータをカタログ化します。

このチュートリアルでは、これらのフライトログからメタデータを推測し、データ Amazon S3 にテーブルを作成するクローラを追加します。

トピック

- 前提条件
- ステップ 1: クローラの追加
- ステップ 2: クローラを実行する
- ステップ 3: AWS Glue Data Catalog オブジェクトを表示する

前提条件

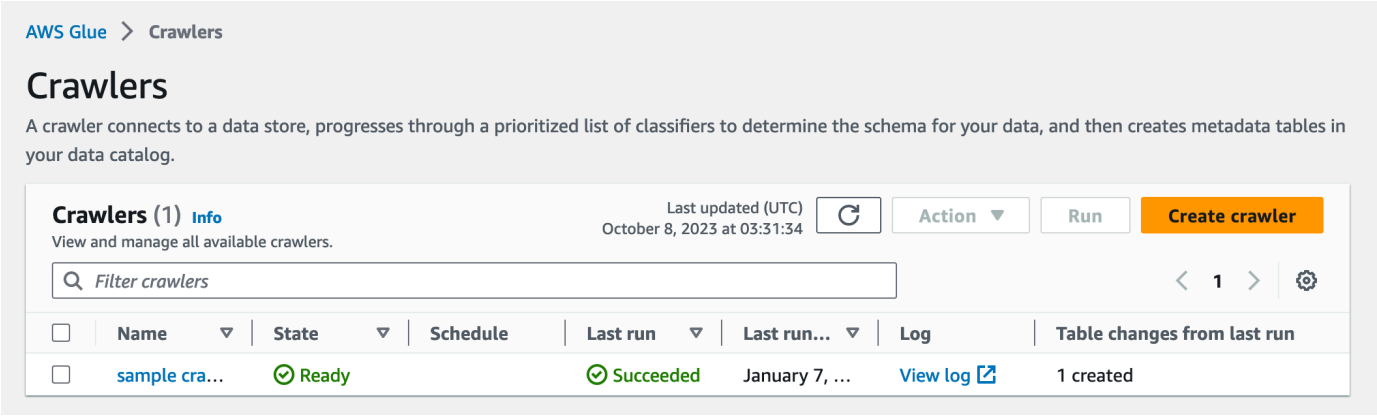
このチュートリアルでは、 AWS アカウントを持ち、AWS Glue にアクセスできることを前提としています。

ステップ 1: クローラの追加

Amazon S3 に保存されている CSV ファイルからメタデータを抽出するクローラを設定および実行するには、以下の手順に従ってください。

Amazon S3 に保存されているファイルを読み込むクローラを作成するには

1. AWS Glue サービスコンソールの左側のメニューで、[Crawlers] (クローラー) を選択します。
2. クローラーページで、[Create crawler] を選択します。これにより、クローラの詳細に関する入力を求める一連のページが表示されます。



3. [クローラ名] フィールドに、「**`Flights Data Crawler`**」を入力して、 [Next] (次へ) を選択します。

   クローラは分類子を呼び出して、データのスキーマを推測します。このチュートリアルでは、デフォルトの CSV 用の組み込みの分類子を使用します。
4. [クローラソースタイプ] で、[Data stores] (データストア) を選択し、[Next] (次へ) を選択します。

5. それでは、クローラをデータに指定します。[Add a data store] (データストアの追加) ページで、[Amazon S3 data store] (Amazon S3 データストア) を選択します。このチュートリアルでは接続を使用しないため、[Connection] (接続) フィールドが表示されている場合は、そのフィールドを空白のままにしてください。

   [Crawl data in] (クロールするデータの場所) で、[Specified path in another account] (別のアカウントで指定されたパス) を選択します。次に、[Include path] (インクルードパス) に、クローラがフライトデータ (**`s3://crawler-public-us-east-1/flight/2016/csv`**) に対するパスを入力します。パスを入力すると、このフィールドのタイトルが [Include path] (インクルードパス) に変わります。[Next] を選択します。

6. 単一のクローラで複数のデータストアをクロールできます。ただし、このチュートリアルでは 1 つのデータストアのみを使用しているため、[No] (なし) を選択して、次に [Next] (次へ) を選択します。

7. クローラーには、データストアにアクセスして AWS Glue Data Catalog でオブジェクトを作成するためのアクセス許可が必要です。これらのアクセス許可を設定するために、[Create an IAM role] (IAM ロールを作成する) を選択します。IAM ロール名は 「`AWSGlueServiceRole-`」で始まります。フィールドにロール名の後ろの部分を入力します。 「**`CrawlerTutorial`**」と入力し、[Next] (次へ) を選択します。

   > **Note**
   >
   > IAM ロールを作成するには、AWS ユーザーは `CreateRole`、`CreatePolicy`、および `AttachRolePolicy` アクセス許可が必要です。

   ウィザードが「`AWSGlueServiceRole-CrawlerTutorial`」という名前の IAM ロールを作成し、AWS 管理ポリシー `AWSGlueServiceRole` をこのロールにアタッチし、Amazon S3 ロケーション `s3://crawler-public-us-east-1/flight/2016/csv` への読み取りアクセスを許可するインラインポリシーを追加します。

8. クローラのスケジュールを作成します。[Frequency] (頻度) で、[Run on demand (オンデマンドで実行する) を選択してから、[Next] (次へ) を選択します。

9. クローラは Data Catalog 内にテーブルを作成します。テーブルは、Data Catalog 内のデータベースに格納されます。まず、Add database (データベースを追加する) を選択してデータベースを作成します。ポップアップウィンドウで、データベース名に 「**`test-flights-db`**」と入力し、[Create] (作成) を選択します。

次に、[Prefix added to tables] (テーブルに追加されたプレフィックス) に「**flights**」を入力します。残りのフィールドにはデフォルト値を使用し、[Next] (次へ) を選択します。

10. 選択した内容を確認するには、[Add crawler] (クローラの追加) ウィザードを使用します。誤りを見つけた場合は、Back (戻る) をクリックして前のページに戻り、修正します。

    情報を確認したら、[Finish] (完了) をクリックしてクローラを作成します。

ステップ 2: クローラを実行する

クローラを作成した後、ウィザードがクローラビューページを表示します。オンデマンドのスケジュールでクローラを作成するため、クローラを実行するオプションを使用できます。

クローラを実行するには

1. このページの上部にあるバナーでは、クローラが作成されたことが表示され、それを実行するかどうかを尋ねられます。[Run it now?] (今すぐ実行しますか？) をクリックしてクローラを実行します。

   バナーでは、クローラの「Attempting to run (実行しようとしています)」および「Running (実行中)」というメッセージの表示に変わります。クローラの実行を開始すると、バナーが消え、クローラの表示が更新され、クローラの [Starting] (起動) ステータスが表示されます。1 分後、[Refresh] (更新) アイコンをクリックして、テーブルに表示されるクローラのステータスを更新できます。
2. クローラが完了すると、クローラの変更を説明する新しいバナーが表示されます。test-flights-db リンクを選択して、Data Catalog オブジェクトを表示できます。

ステップ 3: AWS Glue Data Catalog オブジェクトを表示する

クローラがソースの場所からデータを読み込み、Data Catalog にテーブルを作成します。テーブルは、スキーマを含むデータを表すメタデータ定義です。Data Catalog のテーブルにはデータが含まれていません。代わりに、ジョブ定義のソースまたはターゲットとしてこれらのテーブルを使用します。

クローラが作成した Data Catalog オブジェクトを表示する方法

1. 左側のナビゲーションペインの [Data catalog] (データカタログ) で、[Databases] (データベース) を選択します。ここでは、クローラーによって作成された `flights-db` データベースを表示できます。

2. 左側のナビゲーションの [Data catalog] (データカタログ) で、 [Databases] (データベース) の下の Tables (テーブル) を選択します。ここでは、クローラーが作成した `flightscsv` テーブルを表示できます。テーブル名を選択すると、テーブルの設定、パラメータ、およびプロパティを表示できます。このビューで下にスクロールすると、スキーマを表示できます。スキーマとは、テーブルの列とデータ型に関する情報です。
3. テーブルビューページで [View partitions] (パーティションの表示) を選択すると、データ用に作成されたパーティションを表示できます。最初の列はパーティションキーです。

メタデータの手動定義

AWS Glue データカタログは、データソースとデータセットに関するメタデータを保存する中央リポジトリです。クローラーは、サポートされているデータソースのメタデータを自動的にクロールして入力できますが、データカタログでメタデータを手動で定義する必要があるシナリオがいくつかあります。

- サポートされていないデータ形式 – クローラーでサポートされていないデータソースがある場合は、データカタログでそれらのデータソースのメタデータを手動で定義する必要があります。
- カスタムメタデータ要件 — AWS Glue クローラー は、事前定義されたルールと規則に基づいてメタデータを推測します。AWS Glue クローラー で推測されたメタデータでカバーされない特定のメタデータ要件がある場合は、ニーズに合わせてメタデータを手動で定義できます。
- データガバナンスと標準化 — データガバナンス、コンプライアンス、またはセキュリティ上の理由から、メタデータ定義をより細かくコントロールする必要がある場合があります。メタデータを手動で定義することで、そのメタデータが組織の標準とポリシーに準拠していることを保証できます。
- 将来のデータインジェストのためのプレースホルダー – すぐに使用またはアクセスできないデータソースがある場合は、プレースホルダーとして空のスキーマテーブルを作成できます。データソースが利用可能になったら、事前定義された構造を維持しながら、テーブルに実際のデータを入力できます。

メタデータを手動で定義するには、AWS Glue コンソール、Lake Formation コンソール、AWS Glue API、または AWS Command Line Interface (AWS CLI) を使用できます。データベース、テーブル、パーティションを作成し、列名、データ型、説明、その他の属性などのメタデータプロパティを指定できます。

データベースの作成

データベースは、メタデータテーブルを AWS Glue で組織化するために使用されます。AWS Glue Data Catalog でテーブルを定義すると、データベースに追加します。1 つのテーブルは、1 つのデータベースにしか含まれません。

データベースには、数多くのさまざまなデータストアからのデータを定義するテーブルを含めることができます。このデータには、Amazon Simple Storage Service (Amazon S3) のオブジェクトと、Amazon Relational Database Service のリレーショナルテーブルを含めることができます。

> **Note**
>
> データベースを AWS Glue Data Catalog から削除すると、データベース内のすべてのテーブルも削除されます。

データベースのリストを表示するには、AWS Management Console にサインインして、AWS Glue コンソール (https://console.aws.amazon.com/glue/) を開きます。[Databases] (データベース) を選択し、リスト内のデータベース名を選択して詳細を表示します。

コンソールの [DatabasesAWS Glue] (データベース) タブからデータベースの追加、編集、削除ができます。

- 新しいデータベースを作成するには、[Add database] (データベースを追加) を選択し名前と説明を指定します。Apache Hive など、その他のメタデータストアとの互換性を考慮して、名前は小文字に変換されます。

  > **Note**
  >
  > Amazon Athena からデータベースにアクセスする場合は、英数字とアンダースコア文字のみを使用して名前を指定してください。詳細については、Athena での名前を参照してください。

- データベースの説明を編集するには、データベース名の横にあるチェックボックスを選択し、[Edit] (編集) を選択します。
- データベースを削除するには、データベース名の横にあるチェックボックスを選択し、[Remove] (削除) を選択します。
- データベースに含まれるテーブルのリストを表示するには、データベース名を選択します。そうすると、データベースプロパティにデータベース内のすべてのテーブルが表示されます。

クローラーで書き込みが行われたデータベースを変更するには、クローラー定義を変更する必要があります。詳細については、「クローラーを使用したデータカタログへの入力」を参照してください。

データベースリソースリンク

> AWS Glue コンソールの更新が行われました。コンソールの現行バージョンでは、データベースリソースリンクはサポートされません。

Data Catalog にはデータベースへのリソースリンクを含めることもできます。データベースリソースリンクは、ローカルデータベースまたは共有データベースへのリンクです。現在、AWS Lake Formation のみでリソースリンクを作成できます。データベースへのリソースリンクを作成した後、データベース名を使用する任意の場所で、リソースリンク名を使用できます。所有しているデータベースまたは共有しているデータベースとともに、データベースリソースリンクは `glue:GetDatabases()` から返され、AWS Glue コンソールの [Databases] (データベース) ページにエントリとして表示されます。

Data Catalog には、テーブルリソースリンクを含めることもできます。

リソースリンクの詳細については、AWS Lake Formation デベロッパーガイドの「Creating Resource Links」を参照してください。

テーブルの作成

データストア内のデータのインベントリを作成するには、クローラーの実行が推奨されますが、メタデータテーブルを手動で AWS Glue Data Catalog に追加できます。このアプローチにより、メタデータ定義をより詳細にコントロールし、特定の要件に応じてカスタマイズできます。

以下の方法で、データカタログにテーブルを手動で追加することもできます。

- AWS Glue コンソールを使用して AWS Glue Data Catalog に手動でテーブルを作成します。詳細については、「コンソールを使用したテーブルの作成」を参照してください。
- AWS Glue API の `CreateTable` オペレーションを使用し、AWS Glue Data Catalog にテーブルを作成します。詳細については、「CreateTable アクション (Python: create_table)」を参照してください。
- AWS CloudFormation テンプレートを使用します。詳細については、「AWS Glue の場合は AWS CloudFormation」を参照してください。

コンソールまたは API を使用して手動でテーブルを定義する場合、テーブルスキーマおよびデータソースでデータのタイプとフォーマットを示す分類フィールドの値を指定します、クローラーでテーブルを作成する場合、データ形式とスキーマは、組み込み分類子またはカスタム分類子のいずれかによって決定されます。AWS Glue コンソールを使用してテーブルを作成する方法の詳細については、コンソールを使用したテーブルの作成 を参照してください。

トピック

- テーブルパーティション
- テーブルリソースリンク
- コンソールを使用したテーブルの作成
- パーティションインデックスの作成
- クローラーを使用して手動で作成されたデータカタログテーブルを更新する
- Data Catalog テーブルのプロパティ

テーブルパーティション

Amazon Simple Storage Service (Amazon S3) フォルダの AWS Glue テーブル定義によって、パーティションテーブルを記述できます。たとえば、クエリのパフォーマンスを向上させるために、パーティションテーブルでは月の名前をキーとして毎月のデータを別のファイルに分割する場合があります。AWS Glue では、テーブル定義にテーブルのパーティションキーが含まれています。AWS Glue は、Amazon S3 フォルダのデータを評価してテーブルをカタログ化するとき、個々のテーブルまたはパーティション化されたテーブルを追加するかどうかを決定します。

テーブル内のすべてのパーティションをロードする代わりに、テーブル上にパーティションインデックスを作成して、パーティションのサブセットを取得できます。パーティションインデックスの使用方法については、「パーティションインデックスの作成 」を参照してください。

AWS Glue によって Amazon S3 フォルダのパーティションテーブルを作成するには、次の条件がすべて満たされている必要があります。

- ファイルのスキーマは、AWS Glue によって決定されるものと似ている。
- ファイルのデータ形式が同じである。
- ファイルの圧縮形式が同じである。

例えば、iOS と Android 両方のアプリケーションの販売データを保存する `my-app-bucket` という名前の Amazon S3 バケットを所有しているとします。データは、年、月、日で分割されます。iOS

および Android の販売に関するデータファイルは、同じスキーマ、データ形式、および圧縮形式です。AWS Glue Data Catalog では、AWS Glue クローラーが、年、月、日のパーティションキーを使用して 1 つのテーブル定義を作成します。

次の `my-app-bucket` の Amazon S3 リストでは、パーティションのいくつかが示されています。= シンボルは、パーティションキー値の割り当てに使用されます。

```
my-app-bucket/Sales/year=2010/month=feb/day=1/iOS.csv
my-app-bucket/Sales/year=2010/month=feb/day=1/Android.csv
my-app-bucket/Sales/year=2010/month=feb/day=2/iOS.csv
my-app-bucket/Sales/year=2010/month=feb/day=2/Android.csv
...
my-app-bucket/Sales/year=2017/month=feb/day=4/iOS.csv
my-app-bucket/Sales/year=2017/month=feb/day=4/Android.csv
```

テーブルリソースリンク

> AWS Glue コンソールの更新が行われました。コンソールの現行バージョンでは、テーブルリソースリンクはサポートされません。

Data Catalog には、テーブルへのリソースリンクを含めることができます。テーブルリソースリンクは、ローカルまたは共有テーブルへのリンクです。現在、AWS Lake Formation のみでリソースリンクを作成できます。テーブルへのリソースリンクを作成すると、テーブル名を使用する任意の場所で、リソースリンク名を使用できます。自分が所有している、または自分と共有しているテーブルに加えて、テーブルリソースリンクは `glue:GetTables()` から返され、AWS Glue コンソールの [Tables] (テーブル) ページにエントリとして表示されます。

Data Catalog には、データベースリソースリンクを含めることもできます。

リソースリンクの詳細については、AWS Lake Formation デベロッパーガイドの「Creating Resource Links」を参照してください。

コンソールを使用したテーブルの作成

AWS Glue Data Catalog のテーブルは、データストア内のデータを表すメタデータ定義です。クローラの実行時にテーブルを作成するか、または、AWS Glue コンソールで手動でテーブルを作成で

きます。 コンソールの [TablesAWS Glue] (テーブル) リストに、テーブルのメタデータの値が表示されます。ETL (抽出、変換、ロード) ジョブを作成するときに、テーブル定義を使用してソースとターゲットを指定します。

> **Note**
>
> AWS マネジメントコンソールの最新の変更に伴い、既存の IAM ロール を `SearchTables` 許可を持つように変更することが必要になる場合があります。新しいロールを作成する場合、`SearchTables` API 許可は既にデフォルトとして追加されています。

開始するには、AWS Management Console にサインインし、AWS Glue コンソール (https://console.aws.amazon.com/glue/) を開きます。[Tables] (テーブル) タブをクリックし、[Add tables] (テーブルの追加) ボタンを使用して、クローラで、または属性を手動で入力してテーブルを作成します。

コンソールでテーブルを追加する

クローラを使用してテーブルを追加するには、[Add tables] (テーブルの追加)、[Add tables using a crawler] (クローラを使用してテーブルを追加) の順に選択します。次に、[Add crawler] (クローラの追加) ウィザードの手順に従います。クローラが実行されると、テーブルが AWS Glue Data Catalog に追加されます。詳細については、クローラーを使用したデータカタログへの入力 を参照してください。

Data Catalog の Amazon Simple Storage Service (Amazon S3) テーブル定義の作成に必要な属性が分かっている場合は、テーブルウィザードで作成できます。[Add tables] (テーブルの追加)、[Add table manually] (手動でのテーブルを追加) の順に選択し、[Add table] (テーブルの追加) ウィザードの手順に従います。

コンソールで手動でテーブルを追加するときは、以下の点を考慮します。

- Amazon Athena からテーブルにアクセスする場合は、英数字とアンダースコア文字のみを使用して名前を指定してください。詳細については、「テーブル、データベース、および列の名前」を参照してください。
- ソースデータの場所は Amazon S3 パスにする必要があります。
- データのデータ形式は、ウィザードに表示されているいずれかの形式と一致する必要があります。対応する分類、SerDe、およびその他のテーブルのプロパティは、選択された形式に基づいて自動的に入力されます。次の形式でテーブルを定義できます。

Avro

Apache Avro JSON バイナリ形式。

CSV

文字で区切られた値。また、区切り文字として、カンマ、パイプ、セミコロン、タブ、または Ctrl-A を指定します。

JSON

JavaScript Object Notation。

XML

Extensible Markup Language 形式。データの行を定義する XML タグを指定します。列は行のタグ内で定義されます。

Parquet

Apache Parquet 列指向ストレージ。

ORC

Optimized Row Columnar (ORC) ファイル形式。Hive データを効率的に保存するために設計された形式。

- テーブルのパーティションキーを定義できます。
- 現在、コンソールで作成した分割されたテーブルは、ETL ジョブで使用することはできません。

テーブル属性

以下に重要なテーブル属性を示します。

名前

名前は、テーブルの作成時に決定され、変更することはできません。多くの AWS Glue オペレーションでテーブル名を参照します。

データベース

テーブルが存在するコンテナオブジェクト。このオブジェクトには、AWS Glue Data Catalog 内に存在するテーブルの組織名が含まれ、データストアの組織名とは異なる場合があります。データベースを削除すると、データベースに含まれるすべてのテーブルも Data Catalog から削除されます。

説明

テーブルの説明。テーブルの内容を理解しやすくするために説明を記入できます。

テーブル形式

標準 AWS Glue テーブル、または Apache Iceberg 形式のテーブルの作成を指定します。

データカタログには、テーブルストレージを管理し、Iceberg テーブルのクエリパフォーマンスを向上させるための次のテーブル最適化オプションが用意されています。

- 圧縮 – 古いデータを削除して、フラグメント化されたデータをより大規模で効率的なファイルに統合するために、データファイルはマージされ、書き換えられます。
- スナップショット保持 — スナップショットは、Iceberg テーブルのタイムスタンプ付きバージョンです。スナップショット保持設定を使用すると、スナップショットを保持する期間と保持するスナップショットの数を強制できます。スナップショット保持オプティマイザーを設定すると、古い不要なスナップショットとその基となる関連付けされたファイルを削除して、ストレージのオーバーヘッドを管理するのに役立ちます。
- 孤立ファイルの削除 — 孤立ファイルは、Iceberg テーブルメタデータによって参照されなくなったファイルです。これらのファイルは、特にテーブルの削除や ETL ジョブの失敗などのオペレーションの後、時間の経過と共に蓄積される可能性があります。孤立ファイルの削除を有効にすると、AWS Glue はこれらの不要なファイルを定期的に特定して削除できるため、ストレージが解放されます。

詳細については、「Iceberg テーブルの最適化」を参照してください。

最適化設定

デフォルト設定を使用するか、テーブルオプティマイザを有効にするための設定をカスタマイズできます。

IAM ロール

テーブル最適化を実行するために、サービスはユーザーに代わって IAM ロールを引き受けます。IAM ロールは、ドロップダウンを使用して選択できます。圧縮を有効にするために必要な許可がロールに付与されているようにしてください。

IAM ロールに必要な許可の詳細については、「 テーブル最適化の前提条件 」を参照してください。

ロケーション

このテーブル定義が表すデータストア内のデータの場所へのポインタ。

分類

テーブルの作成時に指定された分類の値。通常、これはクローラが実行されてソースデータの形式を指定するときに書き込まれます。

最終更新日

Data Catalog でこのテーブルが更新された日付と時刻 (UTC)。

追加された日付

Data Catalog にこのテーブルが追加された日付と時刻 (UTC)。

廃止済み

AWS Glue により、Data Catalog のテーブルは元のデータストアに存在しなくなったことが分かると、そのテーブルは廃止されたとしてデータカタログにマークされます。廃止されたテーブルを参照するジョブを実行する場合、ジョブは失敗する可能性があります。廃止されたテーブルを参照するジョブを編集し、ソースおよびターゲットとして削除します。廃止されたテーブルが不要になったら削除することをお勧めします。

Connection

AWS Glue でデータストアへの接続が必要な場合は、接続の名前がテーブルに関連付けられます。

テーブルの詳細の表示と管理

既存のテーブルの詳細を表示するには、リスト内のテーブル名を選択し、[Action, View details] (アクション、詳細を表示) を選択します。

テーブルの詳細にはテーブルのプロパティとスキーマが含まれます。このビューには、テーブルに定義された順序の列名、データ型、およびパーティションのキー列を含む、テーブルのスキーマが表示されます。列が複合型の場合は、以下の例に示すように、[View properties] (プロパティの表示) を選択して、そのフィールドの構造の詳細を表示します。

```
{
"StorageDescriptor":
    {
      "cols": {
         "FieldSchema": [
           {
```

```
              "name": "primary-1",
              "type": "CHAR",
              "comment": ""
            },
            {
              "name": "second ",
              "type": "STRING",
              "comment": ""
            }
          ]
      },
      "location": "s3://aws-logs-111122223333-us-east-1",
      "inputFormat": "",
      "outputFormat": "org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat",
      "compressed": "false",
      "numBuckets": "0",
      "SerDeInfo": {
           "name": "",
           "serializationLib": "org.apache.hadoop.hive.serde2.OpenCSVSerde",
           "parameters": {
               "separatorChar": "|"
            }
      },
      "bucketCols": [],
      "sortCols": [],
      "parameters": {},
      "SkewedInfo": {},
      "storedAsSubDirectories": "false"
    },
    "parameters": {
       "classification": "csv"
    }
}
```

StorageDescriptor などのテーブルのプロパティの詳細については、「StorageDescriptor 構造」を参照してください。

テーブルのスキーマを変更するには、[Edit schema] (スキーマの編集) を選択し、列の追加および削除、列名の変更、データ型の変更をします。

スキーマを含め、テーブルの異なるバージョンを比較するには、[Compare versions] (バージョンの比較) を選択し、テーブルの 2 つのバージョンのスキーマを並べて比較します。詳細については、「テーブルスキーマのバージョンの比較 」を参照してください。

Amazon S3 パーティションを構成するファイルを表示するには、[View partition] (パーティションの表示) を選択します。Amazon S3 のテーブルでは、[Key] (キー) 列に、ソースデータストアでテーブルを分割するために使用されるパーティションキーが表示されます。パーティションは、日付、場所、または部門などのキー列の値に基づいて、テーブルを関連する部分に分割する方法です。パーティションの詳細については、インターネットで「hive パーティション」を検索してください。

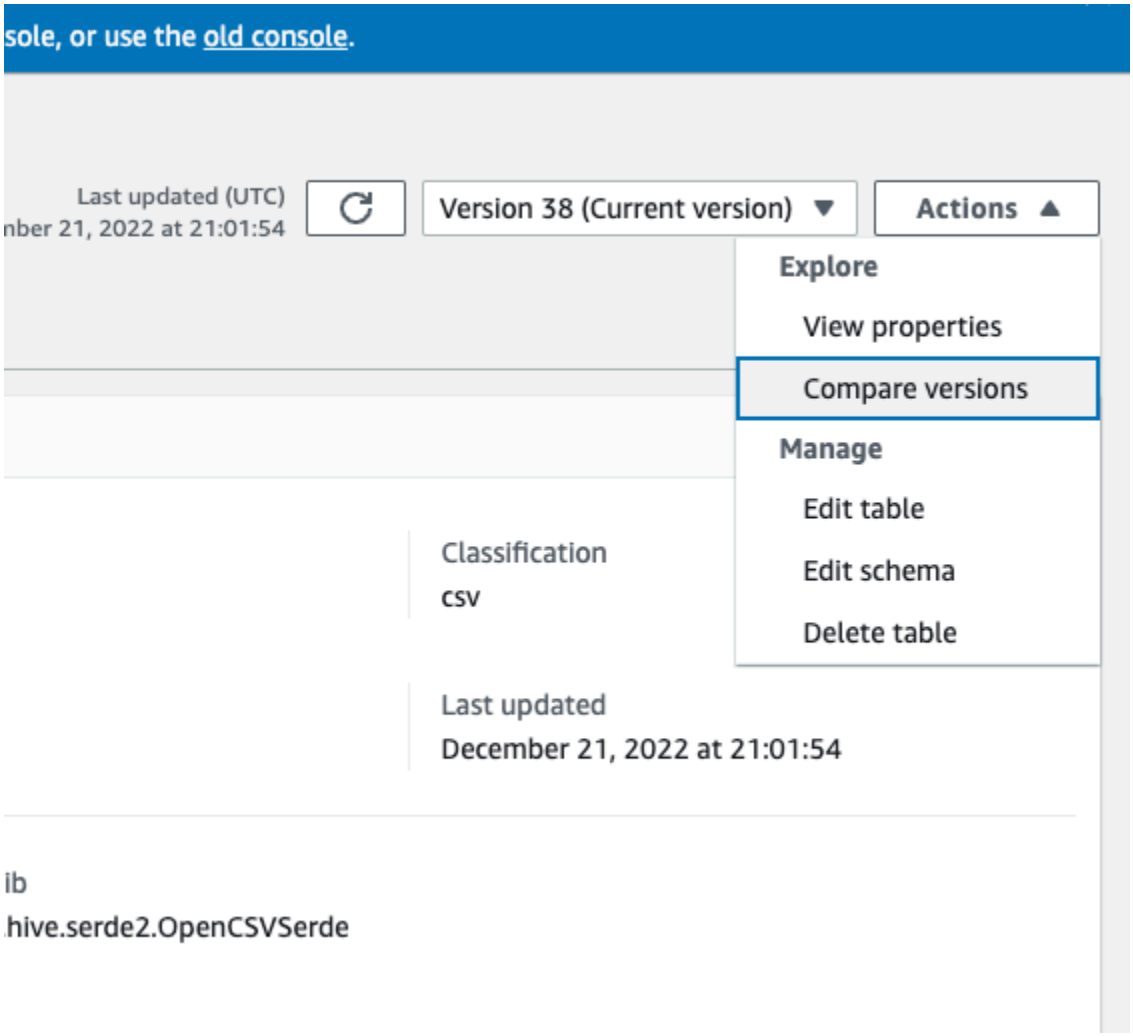
> **Note**
>
> テーブルの詳細を表示するための詳細な手順については、コンソールの [Explore table] (テーブルの確認) チュートリアルを参照してください。

テーブルスキーマのバージョンの比較

テーブルスキーマの 2 つのバージョンを比較する場合、展開・折りたたみによってネストされた行の変更点を比較、2 つのバージョンのスキーマを並べて比較、テーブルプロパティを並べて表示、などが行えます。

バージョンの比較方法

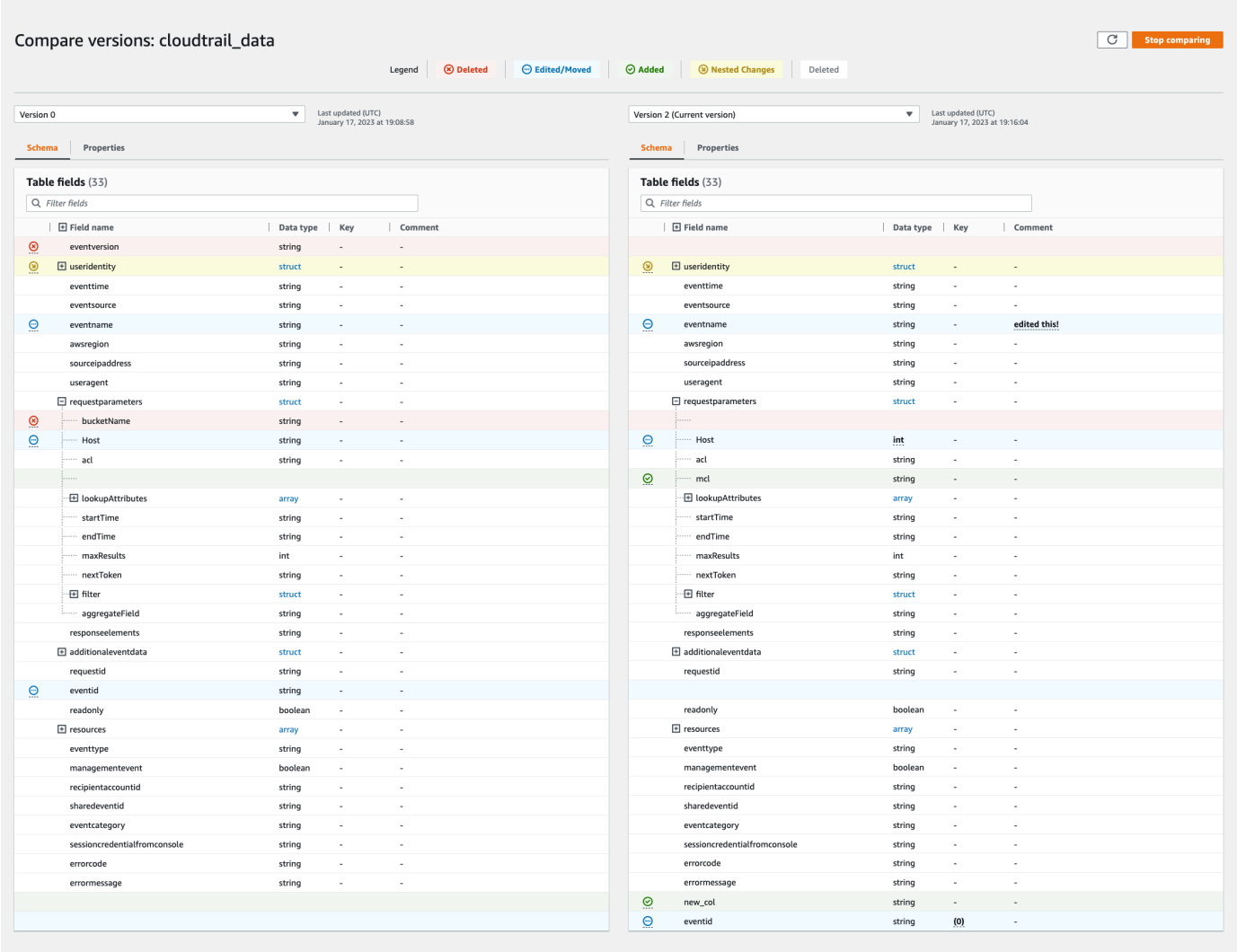
1. AWS Glue コンソールから [テーブル] > [アクション] の順に選択し、[バージョンの比較] を選択します。



2. バージョンのドロップダウンメニューから、比較するバージョンを選択します。スキーマを比較すると、[スキーマ] タブがオレンジ色で強調表示されます。
3. 2 つのバージョン間でテーブルを比較すると、テーブルスキーマが画面の左側と右側に表示されます。列名、データ型、キー、コメントの各フィールドを並べて比較することで、変更点を視覚的に判断できます。変更があると、色付きのアイコンでその変更の種類が表示されます。

   - 削除済み — 赤いアイコンによる表示は、以前のバージョンのテーブルスキーマから列が削除されたことを示します。
   - 編集済みまたは移動済み — 青いアイコンによる表示は、新しいバージョンのテーブルスキーマで列が変更または移動されたことを示します。
   - 追加 — 緑色のアイコンによる表示は、新しいバージョンのテーブルスキーマに列が追加されたことを示します。

- ネストされた変更 — 黄色のアイコンによる表示は、ネストされた列に変更が含まれていることを示します。列を選択して展開すると、削除、編集、移動、追加のいずれかが実行された列が表示されます。



4. 検索バーのフィルタフィールドを使用すると、ここに入力した文字に基づいてフィールドを表示できます。いずれかのテーブルバージョンで列名を入力すると、フィルタリングされたフィールドが両方のテーブルバージョンに表示され、変更が行われた箇所がわかります。
5. プロパティを比較するには、[プロパティ] タブを選択します。
6. バージョンの比較を停止するには、[比較の停止] を選択してテーブルのリストに戻ります。

パーティションインデックスの作成

時間の経過とともに、数十万のパーティションがテーブルに追加されます。GetPartitions API は、テーブル内のパーティションをフェッチするために使用されます。この API は、リクエストで指定された式と一致するパーティションを返します。

「国」、「カテゴリ」、「年」、「月」、「creationDate」のキーで分割された例として、sales_data テーブルを参考にします。2020 年の 2020 年 8 月 15 日以降に「書籍」のカテゴリで販売されたアイテムの売り上げデータを取得する場合、Data Catalog に「Category = 'Books' and creationDate > '2020-08-15'」という式で `GetPartitions` リクエストを行います。

テーブルにパーティションインデックスが存在しない場合、AWS Glue では、テーブルのすべてのパーティションがロードされ、`GetPartitions` リクエストでユーザーが指定したクエリ式を使用してロードされたパーティションがフィルタリングされます。インデックスのないテーブルでパーティション数が増えると、クエリの実行に時間がかかります。インデックスを使用すると、`GetPartitions` クエリでは、テーブル内のすべてのパーティションをロードするのではなく、パーティションのサブセットを取得しようとします。

トピック

- パーティションインデックスについて
- パーティションインデックスを持つテーブルの作成
- パーティションインデックスの既存テーブルへの追加
- テーブル上のパーティションインデックスの説明
- パーティションインデックスの使用に関する制限事項
- GetPartitions 呼び出しを最適化するためのインデックスの使用
- エンジンとの統合

パーティションインデックスについて

パーティションインデックスを作成する際、指定したテーブルに既に存在するパーティションキーのリストを指定します。パーティションインデックスは、テーブルで定義されているパーティションキーのサブリストです。パーティションインデックスは、テーブルで定義されたパーティションキーを任意の順序にして作成できます。上記の sales_data テーブルでは、可能なインデックスには (国、カテゴリ、creationDate)、(国、カテゴリ、年)、(国、カテゴリ)、(国)、(カテゴリ、国、年、月) などがあります。

Data Catalog は、インデックスの作成時に指定された順序でパーティション値を連結します。インデックスは、テーブルに追加されたパーティションと整合して構築されます。インデックスは文字列 (string、char、varchar)、数値 (int、bigint、long、tinyint、smallint)、日付 (yyyy-MM-dd) の列タイプに対して作成できます。

サポートされているデータ型

- Date – YYYY-MM-DD などの ISO 形式の日付。たとえば、2020-08-15 の日付。この形式では、ハイフン (-) を使用して年、月、日を区切ります。インデックス用の日付の許容範囲は、0000-01-01 から 9999-12-31 までです。
- String – 一重引用符または二重引用符で囲まれた文字列リテラルです。
- Char – char(10) など、1 から 255 までの指定された長さを持つ固定長の文字データです。
- Varchar – varchar(10) など、1 から 65535 までの指定された長さを持つ可変長の文字データです。
- Numeric — int、bigint、long、tinyint、smallint

「Numeric」、「String」、「Date」データ型のインデックスは、=、>、>=、<、<=、演算子間をサポートします。インデックス作成ソリューションは現在、AND 論理演算子のみをサポートしています。「LIKE」、「IN」、「OR」、および「NOT」の演算子のある部分式は、インデックスを使用してフィルタリングする場合、この式では無視されます。無視された部分式のフィルタリングは、インデックスフィルタリングを適用した後にフェッチされたパーティションに対して行われます。

テーブルに追加されたパーティションごとに、対応するインデックス項目が作成されます。「n」個のパーティションを持つテーブルの場合、1 つのパーティションインデックスは「n」個のパーティションインデックス項目になります。同じテーブルの「m」個のパーティションインデックスは、「m*n」個のパーティションインデックス項目になります。各パーティションインデックス項目は、現在のデータカタログストレージの AWS Glue 料金ポリシーに従って課金されます。ストレージオブジェクトの料金の詳細については、「AWS Glue の料金」を参照してください。

パーティションインデックスを持つテーブルの作成

テーブルの作成中にパーティションインデックスを作成できます。CreateTable リクエストは、PartitionIndex オブジェクトのリストを入力として指定します。1 つのテーブルには、最大 3 つのパーティションインデックスを作成できます。各パーティションインデックスには、テーブルに対して定義された名前と partitionKeys のリストが必要です。テーブル上に作成されたインデックスは、GetPartitionIndexes API を使用してフェッチできます。

パーティションインデックスの既存テーブルへの追加

既存のテーブルにパーティションインデックスを追加するには、CreatePartitionIndex オペレーションを使用します。CreatePartitionIndex オペレーションごとに、1 つの PartitionIndex を作成できます。インデックスを追加しても、テーブルの可用性には影響しません。これは、インデックスの作成中もテーブルが使用可能になるためです。

追加されたパーティションのインデックスステータスが CREATING に設定され、インデックスデータの作成が開始されます。インデックスの作成プロセスが正常に終了すると、indexStatus は ACTIVE に更新され、プロセスが正常に終了しなかった場合、インデックスステータスは FAILED に更新されます。インデックスが作成できない理由は、複数あり得ます。GetPartitionIndexes オペレーションを使用して、障害の詳細を取得できます。考えられる障害は次のとおりです。

- ENCRYPTED_PARTITION_ERROR – 暗号化されたパーティションを持つテーブルでのインデックスの作成はサポートされていません。
- INVALID_PARTITION_TYPE_DATA_ERROR – partitionKey の値が対応する partitionKey データ型に対して有効な値でない場合に発生します。例えば、「int」データ型の partitionKey の値が「foo」の場合などです。
- MISSING_PARTITION_VALUE_ERROR – indexedKey の partitionValue がない場合に発生します。これは、テーブルのパーティション化に整合性がない場合に発生します。
- UNSUPPORTED_PARTITION_CHARACTER_ERROR – インデックス付きパーティションキーの値に「\u0000」、「\u0001」、または「\u0002」という文字が含まれている場合に発生します。
- INTERNAL_ERROR – インデックスの作成中に内部エラーが発生しました。

テーブル上のパーティションインデックスの説明

テーブル上に作成されたパーティションインデックスを取得するには、GetPartitionIndexes オペレーションを使用します。レスポンスとして、テーブル上のすべてのインデックスと、各インデックスの現在のステータス (IndexStatus) が返ります。

パーティションインデックスの IndexStatus は、次のいずれかになります。

- CREATING – インデックスは現在作成されていますが、まだ使用することはできません。
- ACTIVE – インデックスは、すぐに使用できます。リクエストにインデックスを使用して最適化されたクエリを実行できます。

- DELETING – インデックスは現在削除されているため、使用できなくなっています。アクティブ状態のインデックスは、DeletePartitionIndex リクエストを使用してステータスを ACTIVE から DELETING に移行することによって削除できます。
- FAILED – 既存のテーブルでインデックスの作成に失敗しました。失敗したインデックスは、直近の 10 個まで各テーブルに保存されます。

既存のテーブルで作成されたインデックスには、次のような状態遷移があり得ます。

- CREATING → ACTIVE → DELETING
- CREATING → FAILED

パーティションインデックスの使用に関する制限事項

パーティションインデックスを作成したら、テーブルとパーティションの機能に対する次の変更点に注意してください。

新規パーティションの作成 (インデックス追加後)

テーブルにパーティションインデックスが作成されると、テーブルに追加されたすべての新しいパーティションは、インデックス付きキーのデータ型チェックのために検証されます。インデックス付きキーのパーティション値は、データ型形式について検証されます。データ型のチェックが失敗すると、パーティションの作成操作は失敗します。sales_data テーブルに対して、キー (カテゴリ、年) のインデックスが作成され、カテゴリの型が string、年の型が int の場合、YEAR の値が「foo」であれば、新しいパーティションの作成は失敗します。

インデックスを有効にすると、U+0000、U+00001、U+0002 という文字を含むインデックス付きキー値のパーティションの追加が失敗するようになります。

テーブルの更新

テーブル上にパーティションインデックスを作成すると、既存のパーティションキーのパーティションキー名を変更することはできません。また、インデックスに登録されているキーのタイプまたは順序を変更することはできません。

GetPartitions 呼び出しを最適化するためのインデックスの使用

インデックス付きテーブルで GetPartitions を呼び出すとき、式を含めることができます。また、適用可能な場合は、Data Catalogで、可能であればインデックスが使用されます。インデックスの最初のキーは、フィルタリングに使用されるインデックスの式に含めて渡す必要があります。

フィルタリングのインデックスの最適化は、ベストエフォートとして適用されます。Data Catalog では、可能な限りインデックスの最適化を使用しようとしますが、インデックスが見つからない場合、またはサポートされていない演算子の場合は、すべてのパーティションをロードする既存の実装に戻します。

上記の sales_data テーブルには、[Country, Category, Year] (国、カテゴリ、年) のインデックスを追加できます。式に「Country」が渡されない場合、登録されたインデックスは、インデックスを使用したパーティションのフィルタリングができません。最大 3 つのインデックスを追加して、さまざまなクエリパターンをサポートできます。

いくつかの式を例に取り、そこでインデックスが機能する様子を示します。

表現	インデックスの使用方法
Country = 'US'	インデックスは、パーティションをフィルタリングするために使用されます。
Country = 'US' and Category = 'Shoes'	インデックスは、パーティションをフィルタリングするために使用されます。
Category = 'Shoes'	式に「country」が指定されていないため、インデックスは使用されません。レスポンスを返すため、すべてのパーティションがロードされます。
Country = 'US' and Category = 'Shoes' and Year > '2018'	インデックスは、パーティションをフィルタリングするために使用されます。
Country = 'US' and Category = 'Shoes' and Year > '2018' and month = 2	インデックスを使用して、「country = "US" and category = "shoes" and year > 2018」に該当するすべてのパーティションがフェッチされます。その後、月に関する式によるフィルタリングが実行されます。
Country = 'US' AND Category = 'Shoes' OR Year > '2018'	OR 演算子が式に存在するため、インデックスは使用されません。
Country = 'US' AND Category = 'Shoes' AND (Year = 2017 OR Year = '2018')	インデックスを使用して、「country = "US" and category = "shoes"」に該当するのすべて

表現	インデックスの使用方法
	のパーティションがフェッチされ、その後、年に関する式によるフィルタリングが実行されます。
Country in ('US', 'UK') AND Category = 'Shoes'	IN 演算子は現在サポートされていないため、インデックスはフィルタリングに使用されません。
Country = 'US' AND Category in ('Shoes', 'Books')	インデックスを使用して、「country = "US"」に該当するのすべてのパーティションが取得され、その後、カテゴリに関する式によるフィルタリングが実行されます。
Country = 'US' AND Category in ('Shoes', 'Books') AND (creationDate > '2023-9-01'	「creationDate > '2023-9-01'」に設定し、インデックスを使用して「country = "US"」に該当するすべてのパーティションをフェッチしたら、カテゴリの式にフィルタリングが実行されます。

エンジンとの統合

Redshift Spectrum、Amazon EMR、および AWS Glue ETL Spark DataFrames は、インデックスが AWS Glue で ACTIVE 状態になった後にパーティションを取得するためにインデックスを利用できます。Athena と AWS Glue ETL DynamicFrame では、クエリを向上させるためにインデックスを利用するには追加のステップに従う必要があります。

パーティションのフィルタリングの有効化

Athena でパーティションフィルタリングを有効にするには、テーブルのプロパティを次のように更新する必要があります。

1. AWS Glue コンソールで、[データカタログ] の [テーブル] を選択します。
2. テーブルを選択します。
3. [アクション] で [テーブルの編集] を選択します。
4. [テーブルのプロパティ] で、次の内容を追加します。
   - キー – `partition_filtering.enabled`

- 値 – `true`

5. [Apply] を選択します。

または、Athena で ALTER TABLE SET PROPERTIES クエリを実行してこのパラメータを設定することもできます。

```
ALTER TABLE partition_index.table_with_index
SET TBLPROPERTIES ('partition_filtering.enabled' = 'true')
```

クローラーを使用して手動で作成されたデータカタログテーブルを更新する

手動で AWS Glue Data Catalog テーブルを作成し、AWS Glue クローラーを使用して更新された状態を維持することもできます。スケジュールで実行されているクローラーは、新しいパーティションを追加して、スキーマの変更によりテーブルを更新できます。これは Apache Hive メタストアから移行されたテーブルにも適用されます。

これを行うには、クローラーを定義するときに、1 つ以上のデータストアをクロールのソースとして指定する代わりに、既存のデータカタログテーブルを 1 つ以上指定します。クローラーは、カタログテーブルで指定されたデータストアをクロールします。この場合、新しいテーブルは作成されず、手動で作成されたテーブルが更新されます。

以下に、カタログテーブルを手動で作成し、クローラーソースとしてカタログテーブルを指定するその他の理由を示します。

- カタログテーブル名命名アルゴリズムに依存せず、カタログテーブル名を選択することをお勧めします。
- パーティションの検出を中断する可能性がある形式のファイルが誤ってデータソースパスに保存されないように、新しいテーブルが作成されないようにする必要があります。

詳細については、「ステップ 2: データソースと分類子を選択する」を参照してください。

Data Catalog テーブルのプロパティ

AWS CLI で知られているテーブルのプロパティ、つまりパラメータは無効なキーと値の文字列です。AWS Glue の外部での Data Catalog の使用をサポートする場合は、テーブルに独自のプロパティを設定します。この操作は、Data Catalog を使用する他のサービスでも行うことができます。AWS Glue がジョブまたはクローラーの実行時に一部のテーブルプロパティを設定します。特に

説明がない限り、これらのプロパティは内部使用のためのものであり、現在の形式での存続や、これらのプロパティを手動で変更した場合の製品の動作についてはサポートされません。

AWS Glue クローラーで設定できるテーブルプロパティの詳細については、「the section called “クローラーによって設定されたデータカタログテーブルのパラメータ”」を参照してください。

他の AWS のサービスとの統合

AWS Glue クローラー を使用して AWS Glue Data Catalog にデータを入力することもできますが、いくつかの AWS サービスではカタログに自動的に統合し、データを入力することができます。以下のセクションでは、データカタログに入力できる AWS サービスでサポートされる特定のユースケースについて詳しく説明します。

トピック

- AWS Lake Formation
- Amazon Athena

AWS Lake Formation

AWS Lake Formation は、AWS で簡単にセキュアなデータレイクを構築できるサービスです。Lake Formation は AWS Glue 上に構築され、Lake Formation と AWS Glue は同じ AWS Glue Data Catalog を共有します。Amazon S3 データロケーションを Lake Formation に登録し、Lake Formation コンソールを使用して、AWS Glue データカタログにデータベースとテーブルを作成し、データアクセスポリシーを定義し、データレイク全体のデータアクセスを一元的に監査できます。Lake Formation の細粒度のアクセスコントロールを使用して、既存のデータカタログリソースと Amazon S3 データロケーションを管理できます。

Lake Formation に登録されたデータを使用すると、IAM プリンシパル、AWS アカウント、AWS 組織、組織単位間でデータカタログリソースを安全に共有できます。

Lake Formation を使用したデータカタログリソースの作成の詳細については、AWS Lake Formation デベロッパーガイドの「データカタログのテーブルとデータベースの作成」を参照してください。

Amazon Athena

Amazon Athena は、AWS アカウントの Amazon S3 データに関するテーブルメタデータの保存と取得にデータカタログを使用します。テーブルメタデータは、Athena クエリエンジンがクエリするデータの検索、読み込み、および処理方法を把握できるようにします。

Athena CREATE TABLE ステートメントを直接使用して、AWS Glue Data Catalog にデータを入力できます。クローラーを実行することなく、データカタログでスキーマとパーティションメタデータを手動で定義してデータを入力できます。

1. Athena コンソールで、テーブルメタデータをデータカタログに保存するデータベースを作成します。
2. CREATE EXTERNAL TABLE ステートメントを使用して、データソースのスキーマを定義します。
3. データがパーティション化されている場合は、PARTITIONED BY 句を使用してパーティションキーを定義します。
4. LOCATION 句を使用して、実際のデータファイルを保存する Amazon S3 パスを指定します。
5. CREATE TABLE ステートメントを実行します。

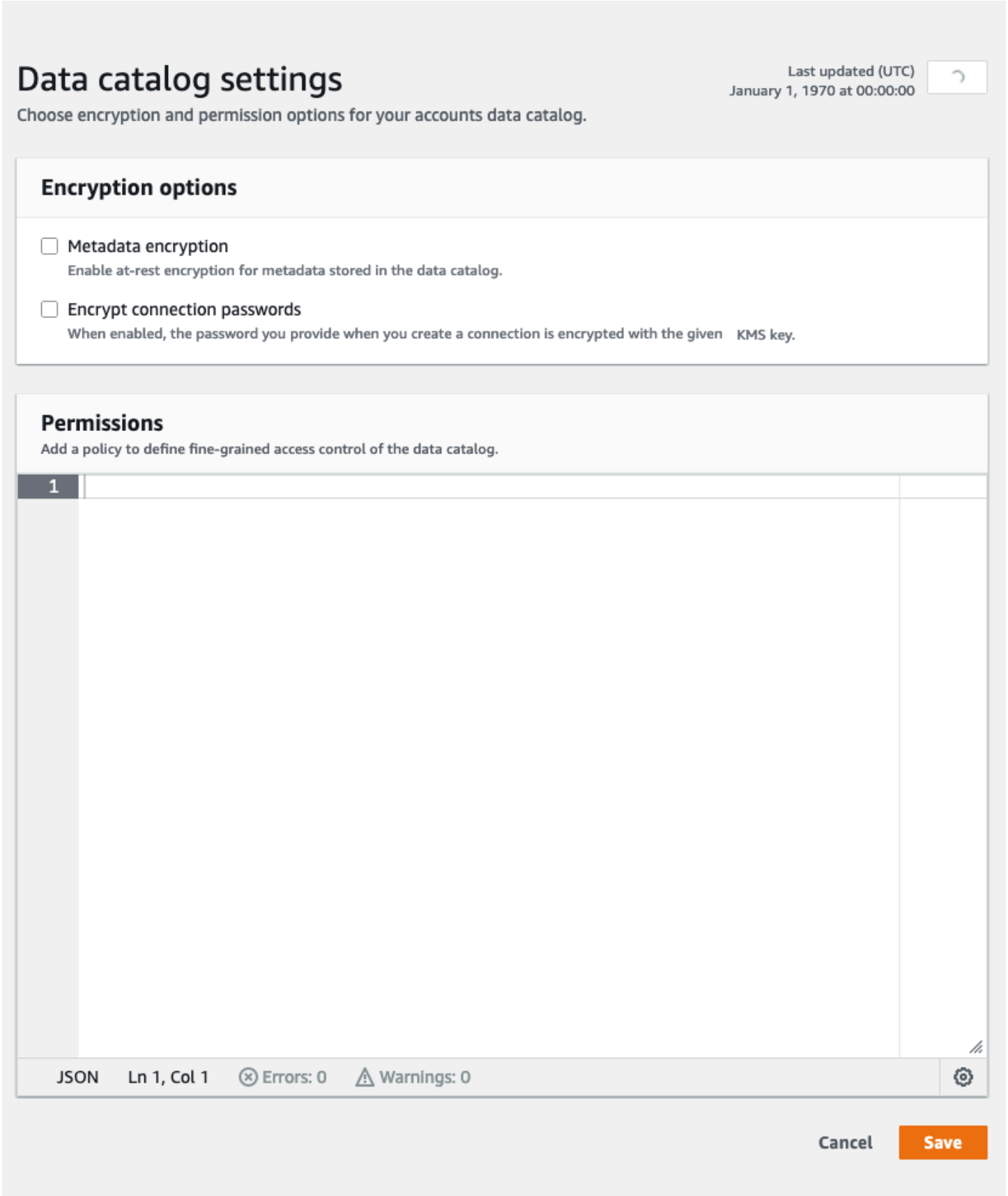
   このクエリは、実際にデータをクロールすることなく、定義したスキーマとパーティションに基づいてデータカタログにテーブルメタデータを作成します。

Athena でテーブルにクエリを実行すると、データカタログのメタデータを使用して Amazon S3 のデータファイルにアクセスしてクエリを実行できます。

詳細については、Amazon Athena ユーザーガイドの「データベースとテーブルの作成」を参照してください。

データカタログの設定

データカタログの設定には、アカウントのデータカタログに対する暗号化およびアクセス許可のオプションが含まれています。


Data catalog settings
Choose encryption and permission options for your accounts data catalog.
Last updated (UTC)
January 1, 1970 at 00:00:00
Encryption options
Metadata encryption
Enable at-rest encryption for metadata stored in the data catalog.
Encrypt connection passwords
When enabled, the password you provide when you create a connection is encrypted with the given KMS key.
Permissions
Add a policy to define fine-grained access control of the data catalog.
1
JSON
Ln 1, Col 1
Errors: 0
Warnings: 0
Cancel
Save

Data Catalog のきめ細かなアクセスコントロールを変更するには

1. AWS Management Consoleにサインインし、AWS Glue コンソール (https://console.aws.amazon.com/glue/) を開きます。
2. 暗号化オプションを選択します。

   - メタデータの暗号化 – データカタログのメタデータを暗号化するには、このチェックボックスをオンにします。メタデータは、指定した AWS Key Management Service (AWS KMS) キーを使用して、保存時に暗号化されます。詳細については、「Data Catalog の暗号化」を参照してください。
   - 接続パスワードの暗号化 – 接続を作成または更新する際、AWS Glue 接続オブジェクトのパスワードを暗号化するには、このチェックボックスをオンにします。パスワードは、指定した AWS KMS キーを使用して暗号化されます。パスワードが返ったら、暗号化されます。このオプションは、Data Catalog のすべての AWS Glue 接続のグローバル設定です。このチェックボックスをオフにしても、以前暗号化したパスワードは、作成または更新する際に使用したキーを使用して暗号化された状態に維持されます。AWS Glue 接続の詳細については、「データへの接続」を参照してください。

     このオプションを有効にする場合は、AWS KMS キーを選択するか、[キーの ARN を入力] を選択して、キーの Amazon リソースネーム (ARN) を指定します。`arn:aws:kms:region:account-id:key/key-id` の形式で ARN を入力します。`arn:aws:kms:region:account-id:alias/alias-name` など、キーのエイリアスで ARN を指定することもできます。

     > **Important**
     >
     > このオプションが選択されている場合は、接続を作成または更新するユーザーまたはロールに、指定された KMS キーの `kms:Encrypt` アクセス許可が必要です。

     詳細については、「接続パスワードの暗号化」を参照してください。
3. [Settings] (設定) をクリックして、[Permissions] (アクセス権限) エディタで、アカウントの Data Catalog のきめ細かなアクセスコントロールを変更するポリシーステートメントを追加します。一度に Data Catalog にアタッチできるポリシーは 1 つだけです。JSON リソースポリシーをこのコンソールに貼り付けることができます。詳細については、「AWS Glue 内のリソースベースのポリシー」を参照してください。
4. [Save] (保存) をクリックして、行った変更を含めて Data Catalog を更新します。

また、AWS Glue API オペレーションを使用して、リソースポリシーを格納、取得、削除できます。詳細については、「AWS Glue のセキュリティ API」を参照してください。

トランザクションテーブルへのデータ入力と管理

Apache Iceberg、Apache Hudi、および Linux Foundation Delta Lake は、Apache Spark で大規模なデータ分析とデータレイクワークロードを処理するために設計されたオープンソースのテーブル形式です。

次の方法を使用して、AWS Glue Data Catalog で Iceberg、Hudi、および Delta Lake の各テーブルにデータを入力できます。

- AWS Glue クローラー – AWS Glue クローラー は、Iceberg、Hudi、Delta Lake のテーブルメタデータを自動的に検出してデータカタログに入力できます。詳細については、「クローラーを使用したデータカタログへの入力 」を参照してください。
- AWS Glue ETL ジョブ – ETL ジョブを作成して Iceberg、Hudi、Delta Lake の各テーブルにデータを書き込み、そのメタデータをデータカタログに入力できます。詳細については、「AWS Glue ETL ジョブでのデータレイクフレームワークの使用」を参照してください。
- AWS Glue コンソール、AWS Lake Formationコンソール、AWS CLI または API – AWS Glue コンソール、Lake Formation コンソール、または API を使用して、データカタログに Iceberg テーブル定義を作成し、管理できます。

トピック

- Apache Iceberg テーブルの作成
- Iceberg テーブルの最適化
- Iceberg テーブルのクエリパフォーマンスの最適化

Apache Iceberg テーブルの作成

Amazon S3 に格納されたデータにより、AWS Glue Data Catalog で Apache Parquet データ形式を使用する Apache Iceberg テーブルを作成できます。Data Catalog のテーブルは、データストア内のデータを表すメタデータ定義です。デフォルトでは、AWS Glue は Iceberg v2 テーブルを作成します。v1 テーブルと v2 テーブルの違いについては、Apache Iceberg ドキュメントの「形式バージョンの変更」を参照してください。

Apache Iceberg は、非常に大規模な分析データセット用のオープンテーブル形式です。Iceberg では、スキーマの変更 (スキーマ進化とも呼ばれます) を簡単に行うことができます。つまり、基になるデータを中断することなく、データテーブルの列を追加、名前変更、または削除できます。Iceberg はデータのバージョニングもサポートしているため、データの変更を経時的に追跡できます。これにより、タイムトラベル機能が有効になるため、過去のバージョンのデータにアクセスしてクエリを実行し、更新と削除の間に行われたデータの変更を分析できます。

AWS Glue または Lake Formation コンソールあるいは AWS Glue API の `CreateTable` オペレーションを使用して、データカタログに Iceberg テーブルを作成することができます。詳細については、「CreateTable アクション (Python: create_table)」を参照してください。

Data Catalog に Iceberg テーブルを作成する場合、読み取りと書き込みを実行できるように、Amazon S3 でテーブル形式とメタデータファイルのパスを指定する必要があります。

Amazon S3 のデータロケーションを AWS Lake Formation に登録すると、Lake Formation を使用してきめ細かなアクセス制御の許可により Iceberg テーブルを保護できます。Amazon S3 のソースデータと、Lake Formation に登録されていないメタデータへのアクセスは、Amazon S3 および AWS Glue アクションの IAM アクセス許可ポリシーによって決定されます。詳細については、「アクセス許可の管理」を参照してください。

> **Note**
>
> Data Catalog は、パーティションの作成と Iceberg テーブルプロパティの追加をサポートしていません。

前提条件

Data Catalog に Iceberg テーブルを作成し、Lake Formation のデータアクセス許可を設定するには、次の要件を満たす必要があります。

1. Lake Formation にデータが登録されていない状態で Iceberg テーブルを作成するために必要なアクセス許可。

   Data Catalog にテーブルを作成するために必要なアクセス許可に加えて、テーブル作成者には次のアクセス許可が必要です。

   - リソース arn:aws:s3:::{bucketName} での `s3:PutObject`
   - リソース arn:aws:s3:::{bucketName} での `s3:GetObject`

- リソース arn:aws:s3:::{bucketName} での `s3:DeleteObject`

2. Lake Formation にデータが登録されている状態で Iceberg テーブルを作成するために必要なアクセス許可:

   Lake Formation を使用してデータレイク内のデータを管理および保護するには、テーブルのデータを含む Amazon S3 ロケーションを Lake Formation に登録します。これは、Lake Formation が Athena、Redshift Spectrum、Amazon EMR などの AWS 分析サービスに認証情報を提供して、データにアクセスできるようにするためです。Amazon S3 ロケーションの登録の詳細については、「データレイクへの Amazon S3 ロケーションの追加」を参照してください。

   Lake Formation に登録されている、基盤となるデータを読み書きするプリンシパルには、次のアクセス許可が必要です。

   - `lakeformation:GetDataAccess`
   - `DATA_LOCATION_ACCESS`

     ロケーションに対するデータロケーション許可を持つプリンシパルは、すべての子ロケーションに対するロケーション許可も持っています。

     データロケーション許可の詳細については、「基盤となるデータのアクセスコントロール」を参照してください。

圧縮を有効にするには、Data Catalog 内のテーブルを更新するアクセス許可を持つ IAM ロールを、サービスが引き受ける必要があります。詳細については、「 テーブル最適化の前提条件 」を参照してください。

Iceberg テーブルの作成

このページに記載されているように、AWS Glue または Lake Formation コンソールあるいは AWS Command Line Interface を使用して Iceberg v1 および v2 テーブルを作成できます。AWS Glue クローラー を使用して Iceberg テーブルを作成することもできます。詳細については、「AWS Glue デベロッパーガイド」の「Data Catalog とクローラー」を参照してください。

Iceberg テーブルを作成するには

Console

1. AWS Management Consoleにサインインし、AWS Glue コンソール (https://console.aws.amazon.com/glue/) を開きます。

2. Data Catalog で [テーブル] を選択し、[テーブルの作成] ボタンを使用して次の属性を指定します。
   - テーブル名 - テーブルの名前を入力します。Athena を使用してテーブルにアクセスする場合は、「Amazon Athena ユーザーガイド」の命名に関するヒントを使用します。
   - データベース - 既存のデータベースを選択するか、新しいデータベースを作成します。
   - 説明 - テーブルの説明。テーブルの内容を理解しやすくするために説明を記入できます。
   - テーブル形式 - [テーブル形式] として、[Apache Iceberg] を選択します。
   - 圧縮を有効にする - [圧縮を有効にする] を選択すると、テーブル内の小さな Amazon S3 オブジェクトが圧縮されてより大きなオブジェクトにまとめられます。
   - IAM ロール:- 圧縮を実行する場合、サービスはユーザーに代わって IAM ロールを引き受けます。IAM ロールは、ドロップダウンを使用して選択できます。圧縮を有効にするために必要なアクセス許可がロールにあることを確認します。

     必要なアクセス許可の詳細については、「 テーブル最適化の前提条件 」を参照してください。
   - ロケーション - メタデータテーブルを保存する Amazon S3 内のフォルダへのパスを指定します。Iceberg が読み取りと書き込みを実行するには、メタデータファイルと Data Catalog 内のロケーションが必要です。
   - スキーマ - [列の追加] を選択して、列と、列のデータ型を追加します。空のテーブルを作成して、後でスキーマを更新することもできます。Data Catalog は Hive データ型をサポートしています。詳細については、「Hive データ型」を参照してください。

     Iceberg では、テーブルを作成した後でスキーマとパーティションを進化させることができます。[Athena クエリ] を使用してテーブルスキーマを更新し、[Spark クエリ] を使用してパーティションを更新できます。

AWS CLI

```
aws glue create-table \
    --database-name iceberg-db \
    --region us-west-2 \
    --open-table-format-input '{
      "IcebergInput": {
           "MetadataOperation": "CREATE",
           "Version": "2"
         }
      }' \
```

```
    --table-input '{"Name":"test-iceberg-input-demo",
            "TableType": "EXTERNAL_TABLE",
            "StorageDescriptor":{
               "Columns":[
                   {"Name":"col1", "Type":"int"},
                   {"Name":"col2", "Type":"int"},
                   {"Name":"col3", "Type":"string"}
                ],
               "Location":"s3://DOC_EXAMPLE_BUCKET_ICEBERG/"
            }
        }'
```

Iceberg テーブルの最適化

AWS Glue は、AWS 分析エンジンと ETL ジョブで使用される Apache Iceberg テーブルの管理とパフォーマンスを強化するための複数のテーブル最適化オプションをサポートしています。これらのオプティマイザーは、効率的なストレージの使用量、クエリパフォーマンスの向上、効率的なデータ管理を実現します。AWS Glue では、次の 3 種類のテーブルオプティマイザーを使用できます。

- 圧縮 — データ圧縮では小さなデータファイルを圧縮して、ストレージの使用量を減らし、読み取りパフォーマンスを向上させます。古いデータを削除して、フラグメント化されたデータをより大規模で効率的なファイルに統合するために、データファイルはマージされ、書き換えられます。圧縮を自動的に実行するように設定できます。

  Binpack は Apache Iceberg のデフォルト圧縮戦略です。より小さなデータファイルをより大きなデータファイルに結合し、最適なパフォーマンスを実現します。圧縮は、同様のデータをクラスター化する Sort 戦略と Z-order 戦略もサポートしています。Sort は、指定された列に基づいてデータを整理し、フィルタリングされた操作のクエリパフォーマンスを向上させます。Z-order は、複数の列を同時にクエリするときにクエリパフォーマンスを向上させるソートされたデータセットを作成します。binpack、sort、Z-order の 3 つの圧縮戦略はすべて、クエリエンジンによってスキャンされるデータ量を減らし、クエリ処理コストを削減します。
- スナップショット保持 - スナップショットは、Iceberg テーブルのタイムスタンプ付きバージョンです。スナップショット保持設定を使用すると、スナップショットを保持する期間と保持するスナップショットの数を強制できます。スナップショット保持オプティマイザーを設定すると、古い不要なスナップショットと、その基になる関連付けられたファイルを削除して、ストレージのオーバーヘッドを管理できます。

- 孤立ファイルの削除 — 孤立ファイルは、Iceberg テーブルメタデータによって参照されなくなったファイルです。これらのファイルは、特にテーブルの削除や ETL ジョブの失敗などのオペレーションの後、時間の経過と共に蓄積される可能性があります。孤立ファイルの削除を有効にすると、AWS Glue はこれらの不要なファイルを定期的に特定して削除できるため、ストレージが解放されます。

圧縮、スナップショット保持、孤立ファイル削除の各オプティマイザーは、AWS Glue コンソール、AWS CLI、または AWS Glue API オペレーションを使用して、データカタログ内の個々の Iceberg テーブルに対して有効または無効にすることができます。

トピック

- テーブル最適化の前提条件
- 圧縮の最適化
- スナップショット保持の最適化
- 孤立ファイルの削除
- 最適化の詳細の表示
- Amazon CloudWatch メトリクスの表示
- オプティマイザーの削除
- 考慮事項と制限事項
- テーブルオプティマイザーがサポートされているリージョン

テーブル最適化の前提条件

テーブルオプティマイザーは、テーブルの最適化オプション (圧縮、スナップショット保持、孤立ファイル削除) を有効にする際に指定する AWS Identity and Access Management (IAM) ロールの許可を引き受けます。すべてのオプティマイザーに対して単一のロールを作成することも、オプティマイザーごとに個別のロールを作成することもできます。

> **Note**
>
> 孤立ファイル削除オプティマイザには、`glue:updateTable` または `s3:putObject` アクセス許可は必要ありません。スナップショットの有効期限と圧縮オプティマイザには、同じアクセス許可セットが必要です。

IAM ロールには、データカタログ内のデータを読み取ったり、メタデータを更新したりするための許可が必要です。IAM ロールを作成し、次のインラインポリシーをアタッチできます。

- AWS Lake Formation に登録されていないデータの場所に対する Amazon S3 の読み取り/書き込み許可を付与する次のインラインポリシーを追加します。このポリシーには、データカタログ内のテーブルを更新するための許可、および AWS Glue が Amazon CloudWatch ログにログを追加したり、メトリクスを公開したりするのを許可するための許可も含まれています。Lake Formation に登録されていない Amazon S3 のソースデータへのアクセスは、Amazon S3 および AWS Glue アクションの IAM アクセス許可ポリシーによって決定されます。

  次のインラインポリシーでは、`bucket-name` を Amazon S3 バケット名、`aws-account-id` および `region` をデータカタログの有効な AWS アカウント番号およびリージョン、`database_name` をデータベース名、`table_name` をテーブル名に置き換えます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject",
                "s3:DeleteObject"
            ],
            "Resource": [
                "arn:aws:s3:::<bucket-name>/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::<bucket-name>"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "glue:UpdateTable",
                "glue:GetTable"
```

```
            ],
            "Resource": [
                "arn:aws:glue:<region>:<aws-account-id>:table/<database-name>/<table-
name>",
                "arn:aws:glue:<region>:<aws-account-id>:database/<database-name>",
                "arn:aws:glue:<region>:<aws-account-id>:catalog"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:<region>:<aws-account-id>:log-group:/aws-glue/iceberg-
compaction/logs:*",
                "arn:aws:logs:<region>:<aws-account-id>:log-group:/aws-glue/iceberg-
retention/logs:*",
                "arn:aws:logs:<region>:<aws-account-id>:log-group:/aws-glue/iceberg-
orphan-file-deletion/logs:*"
        }
    ]
}
```

- 次のポリシーを使用して、Lake Formation に登録されたデータの圧縮を有効にします。

  テーブルに対する IAM_ALLOWED_PRINCIPALS グループアクセス許可が最適化ロールに付与されていない場合、ロールにはテーブルに対する Lake Formation の ALTER、DESCRIBE、INSERT、および DELETE アクセス許可が必要です。

  Lake Formation に Amazon S3 バケットを登録する方法の詳細については、「データレイクへの Amazon S3 ロケーションの追加」を参照してください。

```
{
 "Version": "2012-10-17",
 "Statement": [
    {
        "Effect": "Allow",
        "Action": [
            "lakeformation:GetDataAccess"
        ],
```

```
        "Resource": "*"
    },
    {
        "Effect": "Allow",
        "Action": [
        "glue:UpdateTable",
        "glue:GetTable"
        ],

        "Resource": [
            "arn:aws:glue:<region>:<aws-account-
id>:table/<databaseName>/<tableName>",
            "arn:aws:glue:<region>:<aws-account-id>:database/<database-name>",
            "arn:aws:glue:<region>:<aws-account-id>:catalog"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "logs:CreateLogGroup",
            "logs:CreateLogStream",
            "logs:PutLogEvents"
        ],
        "Resource": [
            "arn:aws:logs:<region>:<aws-account-id>:log-group:/aws-glue/iceberg-
compaction/logs:*",
            "arn:aws:logs:<region>:<aws-account-id>:log-group:/aws-glue/iceberg-
retention/logs:*",
            "arn:aws:logs:<region>:<aws-account-id>:log-group:/aws-glue/iceberg-
orphan-file-deletion/logs:*"
        }
    ]
 }
```

- (オプション) サーバー側の暗号化を使用して暗号化された Amazon S3 バケットにデータがある Iceberg テーブルを最適化するには、圧縮ロールに Amazon S3 オブジェクトを復号して、暗号化されたバケットにオブジェクトを書き込むための新しいデータキーを生成するアクセス許可が必要です。次のポリシーを目的の AWS KMS キーに追加します。バケットレベルの暗号化のみがサポートされています。

```
{
    "Effect": "Allow",
```

```
    "Principal": {
        "AWS": "arn:aws:iam::<aws-account-id>:role/<optimizer-role-name>"
    },
    "Action": [
        "kms:Decrypt",
        "kms:GenerateDataKey"
    ],
    "Resource": "*"
}
```

- (オプション) Lake Formation で登録されたデータの場所については、場所を登録するために使用されるロールには、Amazon S3 オブジェクトを復号したり、暗号化されたバケットにオブジェクトを書き込むための新しいデータキーを生成したりする許可が必要です。詳細については、「暗号化された Amazon S3 の場所の登録」を参照してください。
- (オプション) AWS KMS キーが別の AWS アカウントに保存されている場合は、圧縮ロールに次のアクセス許可を含める必要があります。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
                "kms:GenerateDataKey"
            ],
            "Resource": ["arn:aws:kms:<REGION>:<KEY_OWNER_ACCOUNT_ID>:key/<KEY_ID>" ]
        }
    ]
}
```

- 圧縮を実行するために使用するロールには、そのロールに対する `iam:PassRole` アクセス許可が必要です。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
```

```
            ],
            "Resource": [
                "arn:aws:iam::<account-id>:role/<optimizer-role-name>"
            ]
        }
    ]
}
```

- 圧縮プロセスを実行する IAM ロールを引き受けるには、次の信頼ポリシーを AWS Glue サービスのロールに追加します。

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

圧縮の最適化

Apache Iceberg などのオープンテーブル形式を使用する Amazon S3 データレイクは、データを S3 オブジェクトとして保存します。データレイクテーブルに数千の小さな Amazon S3 オブジェクトがある場合、メタデータのオーバーヘッドが増加し、読み取りパフォーマンスに悪影響が及びます。AWS Glue Data Catalog は Iceberg テーブル用のマネージド圧縮を提供し、小さなオブジェクトを圧縮してより大きなオブジェクトにまとめ、Amazon Athena、Amazon EMR、AWS Glue ETL ジョブなどの AWS 分析サービスによる読み取りパフォーマンスを向上させます。データカタログは、同時クエリに支障をきたすことなく圧縮を実行し、Parquet 形式のテーブルに対してのみ圧縮をサポートしています。

テーブルオプティマイザは、テーブルパーティションを継続的にモニタリングして、ファイル数とファイルサイズがしきい値を超えたときに圧縮プロセスを開始します。

データカタログでは、テーブル、またはそのいずれかのパーティションに 100 個を超えるファイルがある場合、圧縮プロセスが開始されます。各ファイルは、ターゲットファイルサイズの 75% 未満である必要があります。ターゲットファイルサイズは、`write.target-file-size-bytes` テーブルプロパティによって定義されます。明示的に設定されていない場合、このプロパティはデフォルトで 512 MB に設定されます。

制限事項については、「マネージドデータ圧縮でサポートされる形式と制限事項」を参照してください。

トピック

- 圧縮オプティマイザの有効化
- 圧縮オプティマイザの無効化

圧縮オプティマイザの有効化

AWS Glue コンソール、AWS CLI、または AWS API を使用して、AWS Glue データカタログ内の Apache Iceberg テーブルの圧縮を有効にすることができます。新しいテーブルの場合は、テーブル形式として Apache Iceberg を選択し、テーブルの作成時に圧縮を有効にすることができます。圧縮は、新しいテーブルのためにデフォルトで無効になっています。

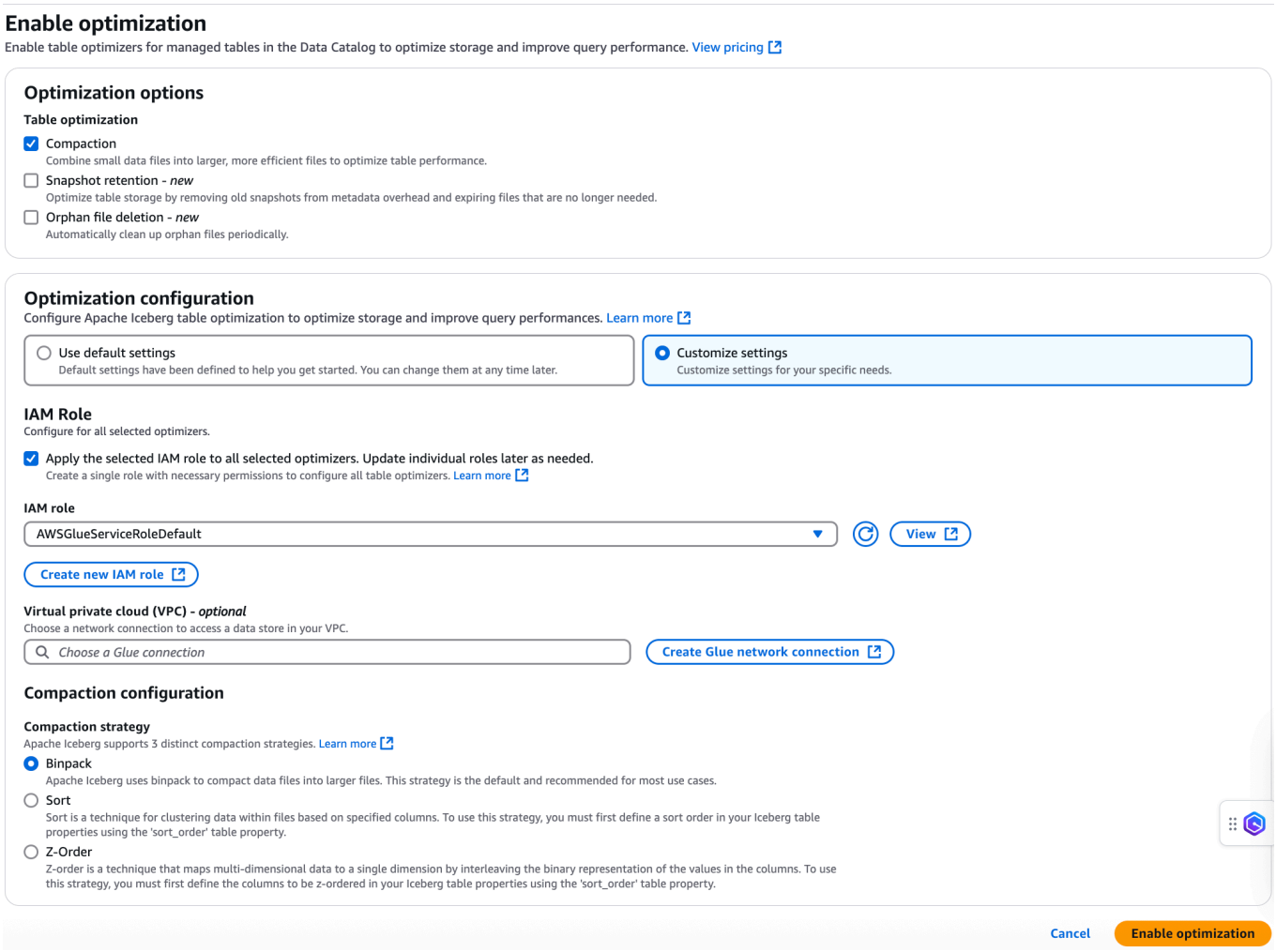
Console

圧縮を有効にするには

1. https://console.aws.amazon.com/glue/ で AWS Glue コンソールを開き、データレイク管理者、テーブル作成者、またはテーブルに対する `glue:UpdateTable` および `lakeformation:GetDataAccess` の許可を付与されたユーザーとしてサインインします。
2. ナビゲーションペインの [データカタログ] で [テーブル] を選択します。
3. [テーブル] ページで、圧縮を有効にするオープンテーブル形式のテーブルを選択し、[アクション] メニューで [最適化]、[有効にする] を選択します。

   [テーブルの詳細] ページで [テーブルの最適化] タブを選択して、圧縮を有効にすることもできます。ページの下部にある [テーブル最適化] タブを選択し、[圧縮を有効にする] を選択します。

   [最適化を有効にする] オプションは、データカタログで新しい Iceberg テーブルを作成するときにも使用できます。
4. [最適化を有効にする] ページで、[テーブル最適化オプション] の [圧縮] を選択します。



5. 次に、テーブル最適化の前提条件 セクションに表示されたアクセス許可を持つ IAM ロールをドロップダウンから選択します。

   また、[新しい IAM ロールを作成] オプションを選択して、圧縮の実行に必要なアクセス許可を持つカスタムロールを作成することもできます。

   以下の手順に従って、既存の IAM ロールを更新します。

   a. IAM ロールの許可ポリシーを更新するには、IAM コンソールで、圧縮の実行に使用されている IAM ロールにアクセスします。

   b. [アクセス許可の追加] セクションで、[ポリシーの作成] を選択します。新しく開いたブラウザウィンドウで、ロールで使用する新しいポリシーを作成します。

   c. [ポリシーの作成] ページで、[JSON] タブを選択します。前提条件に示している JSON コードをポリシーエディタフィールドにコピーします。

6. Iceberg テーブルオプティマイザーが、特定の仮想プライベートクラウド (VPC) から Amazon S3 バケットにアクセスする必要があるセキュリティポリシー設定がある場合は、AWS Glue ネットワーク接続を作成するか、既存のバケットを使用します。

   AWS Glue VPC 接続をまだ設定していない場合は、AWS Glue コンソールまたは AWS CLI/SDK を使用して[コネクタ用の接続を作成する]セクションの手順に従って、新しい接続を作成します。

7. 圧縮戦略を選択します。次のオプションを使用できます。

   - Binpack – Binpack は Apache Iceberg のデフォルト圧縮戦略です。より小さなデータファイルをより大きなデータファイルに結合し、最適なパフォーマンスを実現します。
   - Sort – Apache Iceberg でのソートは、指定された列に基づいてファイル内の情報をクラスター化するデータ整理手法であり、処理する必要があるファイルの数を減らすことでクエリのパフォーマンスを大幅に向上させます。ソート順序フィールドを使用して Iceberg のメタデータでソート順序を定義し、複数の列を指定すると、ソート順序で列が表示される順序でデータがソートされ、同様の値を持つレコードがファイル内にまとめて保存されます。ソート圧縮戦略では、パーティション内のすべてのファイル間でデータをソートすることで、最適化をさらに進めます。
   - Z-order – Z-order は、重要度が同じ複数の列でソートする必要がある場合にデータを整理する方法です。1 つの列を他の列よりも優先する従来のソートとは異なり、Z-order は各列にバランスの取れた重みを与えるため、データを検索するときにクエリエンジンが読み取るファイルが少なくなります。

     この手法は、異なる列の値の二進桁を組み合わせることによって機能します。例えば、2 つの列に数字 3 と 4 がある場合、Z-order は最初にそれらを二進数 (3 が 011 になり、4 が 100 になります) に変換し、これらの数字をインターリーブして新しい値 011010 を作成します。このインターリーブにより、関連するデータを物理的に近づけるパターンが作成されます。

     Z-order は、多次元クエリに特に効果的です。例えば、収入、州、郵便番号で Z-order に並べられた顧客テーブルは、複数のディメンションをまたいでクエリする場合の階層ソートと比較して、優れたパフォーマンスを実現できます。このように整理することで、収入と地理的位置の特定の組み合わせをターゲットとするクエリを使用して、不要なファイルスキャンを最小限に抑えながら、関連するデータをすばやく見つけることができます。

8. [最適化を有効にする] を選択します。

AWS CLI

次の例は、圧縮を有効にする方法を示しています。アカウント ID を有効な AWS アカウント ID に置き換えます。データベース名とテーブル名を実際の Iceberg テーブル名とデータベース名に置き換えます。roleArn を、圧縮の実行に必要なアクセス許可を持つ IAM ロールの ARN (AWSリソースネーム) と IAM ロールの名前に置き換えます。圧縮戦略 sort は、z-order や binpack などサポートされている他の戦略に置き換えることができます。

要件に応じて順序を設定します。

```
aws glue create-table-optimizer \
  --catalog-id 123456789012 \
  --database-name iceberg_db \
  --table-name iceberg_table \
  --table-optimizer-configuration '{
    "roleArn": "arn:aws:iam::123456789012:role/optimizer_role",
    "enabled": true,
    "vpcConfiguration": {"glueConnectionName": "glue_connection_name"},
    "compactionConfiguration": {
      "icebergConfiguration": {"strategy": "sort"}
    }
  }'\
--type compaction
```

AWS API

CreateTableOptimizer オペレーションを呼び出して、テーブルの圧縮を有効にします。

圧縮を有効にすると、圧縮の実行が完了後、[テーブルの最適化] タブに次の圧縮の詳細が表示されます。

開始時間

データカタログ内で圧縮プロセスが開始された時刻。値は UTC 時間のタイムスタンプです。

終了時刻

Data Catalog で圧縮処理が終了した時刻。値は UTC 時間のタイムスタンプです。

ステータス

圧縮実行のステータス。値は成功または失敗です。

圧縮ファイル数

圧縮したファイルの総数。

圧縮バイト数

圧縮したバイトの総数。

圧縮オプティマイザの無効化

AWS Glue コンソールまたは AWS CLI を使用して、特定の Apache Iceberg テーブルの自動圧縮を無効にできます。

Console

1. AWS Management Consoleにサインインし、AWS Glue コンソール (https://console.aws.amazon.com/glue/) を開きます。
2. 左のナビゲーションで、[Data Catalog] の [テーブル] を選択します。
3. テーブルリストから、圧縮を無効にする Iceberg テーブルを選択します。
4. [テーブルの詳細] ページの下部にある、[テーブル最適化] タブを選択します。
5. [アクション] から、[無効にする] を選択し、[圧縮] を選択します。
6. 確認メッセージで [圧縮を無効にする] を選択します。圧縮は後で再度有効にすることができます。

   確認すると、圧縮が無効になり、テーブルの圧縮ステータスが `Disabled` に戻ります。

AWS CLI

次の例で、アカウント ID を、有効な AWS アカウント ID に置き換えます。データベース名とテーブル名を、実際の Iceberg テーブル名とデータベース名に置き換えます。`roleArn` を、圧縮の実行に必要なアクセス許可を持つ IAM ロールの AWS リソースネーム (ARN) と IAM ロールの実際の名前に置き換えます。

```
aws glue update-table-optimizer \
  --catalog-id 123456789012 \
  --database-name iceberg_db \
  --table-name iceberg_table \
  --table-optimizer-configuration
 '{"roleArn":"arn:aws:iam::123456789012:role/optimizer_role", "enabled":'false',
 "vpcConfiguration":{"glueConnectionName":"glue_connection_name"}}'\
```

```
  --type compaction
```

AWS API

UpdateTableOptimizer オペレーションを呼び出して、特定のテーブルの圧縮を無効にします。

スナップショット保持の最適化

Apache Iceberg スナップショット保持機能を使用すると、ユーザーは特定の時点で履歴データをクエリし、テーブルへの不要な変更を元に戻すことができます。AWS Glue データカタログでは、スナップショット保持設定により、これらのスナップショット (テーブルデータのバージョン) の有効期限が切れて削除されるまでの保持期間が制御されます。これは、設定された保持期間または保持するスナップショットの最大数に基づいて古いスナップショットを自動的に削除することで、ストレージコストとメタデータのオーバーヘッドを管理するのに役立ちます。

保持期間を日数で設定し、テーブルに保持するスナップショットの最大数を設定できます。AWS Glue は、最新のスナップショットを設定した上限まで保持しながら、指定された保持期間より古いスナップショットをテーブルメタデータから削除します。メタデータから古いスナップショットを削除すると、AWS Glue は、参照されなくなった、期限切れのスナップショットに固有の対応するデータとメタデータファイルを削除します。これにより、有効期限が切れたスナップショットデータで使用されていたストレージ領域を再利用しながら、保持されている残りのスナップショットまでのタイムトラベルクエリのみが可能になります。

トピック

- スナップショット保持オプティマイザーの有効化
- スナップショット保持オプティマイザーの更新
- スナップショット保持オプティマイザーの無効化

スナップショット保持オプティマイザーの有効化

AWS Glue コンソール、AWS CLI、または AWS API を使用して、データカタログ内の Apache Iceberg テーブルのスナップショット保持オプティマイザーを有効にすることができます。新しいテーブルの場合は、テーブル形式として Apache Iceberg を選択し、テーブルの作成時にスナップショット保持オプティマイザーを有効にすることができます。新しいテーブルでは、スナップショットの保持はデフォルトで無効になっています。

Console

スナップショット保持オプティマイザーを有効にするには

1. https://console.aws.amazon.com/glue/ で AWS Glue コンソールを開き、データレイク管理者、テーブル作成者、またはテーブルに対する `glue:UpdateTable` および `lakeformation:GetDataAccess` の許可を付与されたユーザーとしてサインインします。
2. ナビゲーションペインの [データカタログ] で [テーブル] を選択します。
3. [テーブル] ページで、スナップショット保持オプティマイザーを有効にする Iceberg テーブルを選択してから、[アクション] メニューで、[最適化] の [有効化] を選択します。

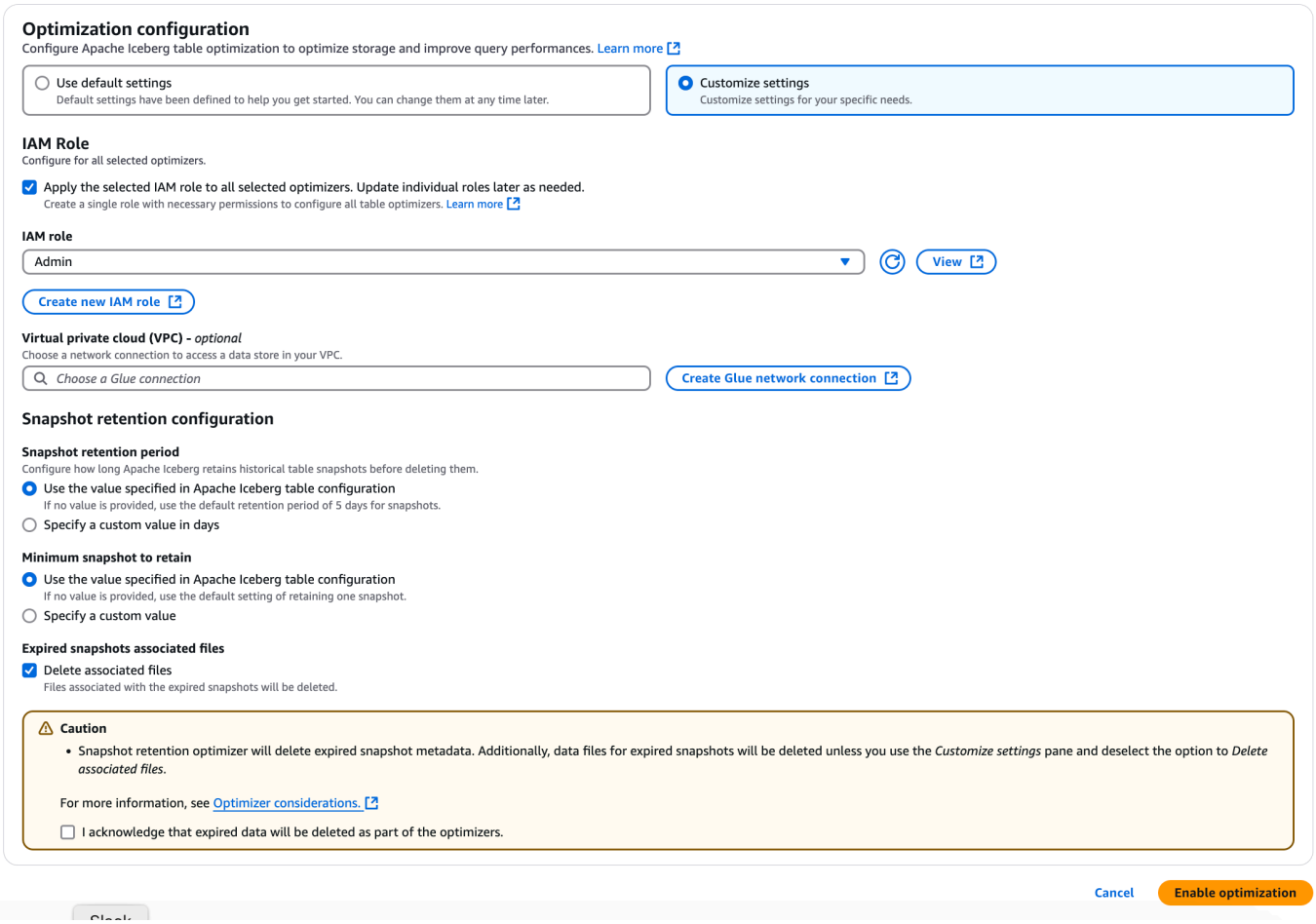
   テーブルを選択して [テーブルの詳細] ページを開いて、最適化を有効にすることもできます。ページの下部にある [テーブル最適化] タブを選択し、[スナップショットの保持を有効にする] を選択します。
4. [最適化を有効にする] ページの [最適化設定] には、[デフォルト設定の使用] と [設定のカスタマイズ] の 2 つのオプションがあります。デフォルト設定を使用する場合、AWS Glue は Iceberg テーブル設定で定義されたプロパティを使用して、スナップショットの保持期間と保持するスナップショットの数を決定します。この設定がない場合、AWS Glue は 1 つのスナップショットを 5 日間保持し、期限切れのスナップショットに関連付けられたファイルを削除します。
5. 次に、オプティマイザーを実行するためにお客様に代わって AWS Glue が引き受けることのできる IAM ロールを選択します。IAM ロールに必要なアクセス許可の詳細については、「テーブル最適化の前提条件」セクションを参照してください。

   以下の手順に従って、既存の IAM ロールを更新します。

   a. IAM ロールの許可ポリシーを更新するには、IAM コンソールで、圧縮の実行に使用されている IAM ロールにアクセスします。

   b. [アクセス許可の追加] セクションで、[ポリシーの作成] を選択します。新しく開いたブラウザウィンドウで、ロールで使用する新しいポリシーを作成します。

   c. [Create policy] (ポリシーの作成) ページで、[JSON] タブを選択します。前提条件に示している JSON コードをポリシーエディタフィールドにコピーします。
6. [スナップショット保持設定] の値を手動で設定する場合は、[設定のカスタマイズ] を選択します。



7. [選択した IAM ロールを選択したオプティマイザーに適用] オプションのボックスを選択して、すべてのオプティマイザーを有効にするために単一の IAM ロールを使用します。
8. Iceberg テーブルオプティマイザーが、特定の仮想プライベートクラウド (VPC) から Amazon S3 バケットにアクセスする必要があるセキュリティポリシー設定がある場合は、AWS Glue ネットワーク接続を作成するか、既存のバケットを使用します。

   AWS Glue VPC 接続をまだ設定していない場合は、AWS Glue コンソールまたは AWS CLI/SDK を使用して[コネクタ用の接続を作成する]セクションの手順に従って、新しい接続を作成します。
9. 次に、[スナップショット保持設定] で、[Iceberg テーブル設定] で指定された値を使用するか、スナップショット保持期間 (history.expire.max-snapshot-age-ms) と保持するスナップショットの最小数 (history.expire.min-snapshots-to-keep) のカスタム値を指定します。
10. [関連するファイルを削除] を選択して、テーブルオプティマイザーがテーブルメタデータから古いスナップショットを削除するときに、基盤となるファイルを削除します。

このオプションを選択しない場合、古いスナップショットがテーブルメタデータから削除されたときに、関連するファイルが孤立ファイルとしてストレージに残ります。

11次に、注意書きを読み、[同意します] を選択して続行します。

> **Note**
>
> データカタログでは、スナップショット保持オプティマイザーにより、ブランチおよびタグレベルの保持ポリシーによって制御されるライフサイクルが尊重されます。詳細については、Iceberg のドキュメントの「Branching and tagging」セクションを参照してください。

12設定を確認し、[最適化を有効にする] を選択します。

保持オプティマイザーが実行され、設定に基づいて古いスナップショットの有効期限が切れるまで数分待ちます。

AWS CLI

AWS Glue で新しい Iceberg テーブルのスナップショット保持を有効にするには、タイプが `retention` のテーブルオプティマイザーを作成し、`table-optimizer-configuration` の `enabled` フィールドを `true` に設定する必要があります。これを行うには、AWS CLI コマンドの `create-table-optimizer` または `update-table-optimizer` を使用します。さらに、要件に基づいて、`snapshotRetentionPeriodInDays` や `numberOfSnapshotsToRetain` などの保持設定フィールドを指定する必要があります。

次の例は、スナップショット保持オプティマイザーを有効にする方法を示しています。アカウント ID を有効な AWS アカウント ID に置き換えます。データベース名とテーブル名を実際の Iceberg テーブル名とデータベース名に置き換えます。`roleArn` を、スナップショット保持オプティマイザーの実行に必要なアクセス許可を持つ IAM ロールの AWS リソースネーム (ARN) と IAM ロールの名前に置き換えます。

```
aws glue create-table-optimizer \
  --catalog-id 123456789012 \
  --database-name iceberg_db \
  --table-name iceberg_table \
  --table-optimizer-configuration
 '{"roleArn":"arn:aws:iam::123456789012:role/optimizer_role","enabled":'true',
 "vpcConfiguration":{
```

```
"glueConnectionName":"glue_connection_name"}, "retentionConfiguration":
{"icebergConfiguration":
{"snapshotRetentionPeriodInDays":7,"numberOfSnapshotsToRetain":3,"cleanExpiredFiles":'true'}
  --type retention
```

このコマンドは、指定されたカタログ、データベース、およびリージョンの、指定された Iceberg テーブルの保持オプティマイザーを作成します。table-optimizer-configuration では、使用する IAM ロール ARN を指定し、オプティマイザーを有効にして、保持設定を設定します。この例では、スナップショットを 7 日間保持し、少なくとも 3 つのスナップショットを保持し、期限切れのファイルをクリーンアップします。

- snapshotRetentionPeriodInDays – 期限切れになるまでのスナップショットを保持する日数。デフォルト値は 5 です。
- numberOfSnapshotsToRetain – 保持期間より古い場合でも保持するスナップショットの最小数。デフォルト値は 1 です。
- cleanExpiredFiles – スナップショットの有効期限が切れた後に、期限切れのデータファイルを削除するかどうかを示すブール値。デフォルト値は `true` です。

  true に設定した場合、古いスナップショットがテーブルメタデータから削除されると、基盤となるファイルが削除されます。このパラメータを false に設定した場合、古いスナップショットがテーブルメタデータから削除されても、基盤となるファイルは孤立ファイルとしてストレージに残ります。

AWS API

CreateTableOptimizer オペレーションを呼び出して、テーブルのスナップショット保持オプティマイザーを有効にします。

圧縮を有効にすると、[テーブル最適化] タブに以下の圧縮の詳細が表示されます (約 15～20 分後)。

開始時刻

スナップショット保持オプティマイザーが開始された時刻。値は UTC 時間のタイムスタンプです。

実行時間

この時間は、オプティマイザーがタスクを完了するためにかかる時間を示します。値は UTC 時間のタイムスタンプです。

ステータス

オプティマイザーの実行のステータス。値は成功または失敗です。

削除されたデータファイル

削除されたファイルの合計数。

削除されたマニフェストファイル

削除されたマニフェストファイルの合計数。

削除されたマニフェストリスト

削除されたマニフェストリストの合計数。

スナップショット保持オプティマイザーの更新

AWS Glue コンソール、AWS CLI、または UpdateTableOptimizer API を使用して、特定の Apache Iceberg テーブルに対するスナップショット保持オプティマイザーの既存の設定を更新できます。

Console

スナップショット保持設定を更新するには

1. AWS Management Consoleにサインインし、AWS Glue コンソール (https://console.aws.amazon.com/glue/) を開きます。
2. [データカタログ]、[テーブル] の順に選択します。Iceberg テーブルリストから、スナップショット保持オプティマイザー設定を更新するテーブルを選択します。
3. [テーブルの詳細] ページの下部セクションで、[テーブルの最適化] タブを選択し、[編集] を選択します。ページの右上隅にある [アクション]メニューから、[最適化] で [編集] を選択することもできます。
4. [最適化の編集] ページで、必要な変更を加えます。
5. [Save] を選択します。

AWS CLI

AWS CLI を使用してスナップショット保持オプティマイザーを更新するときは、次のコマンドを使用できます。

```
aws glue update-table-optimizer \
```

```
 --catalog-id 123456789012 \
 --database-name iceberg_db \
 --table-name iceberg_table \
 --table-optimizer-configuration
 '{"roleArn":"arn:aws:iam::123456789012:role/optimizer_role"","enabled":'true',
 "vpcConfiguration":
{"glueConnectionName":"glue_connection_name"},"retentionConfiguration":
{"icebergConfiguration":
{"snapshotRetentionPeriodInDays":7,"numberOfSnapshotsToRetain":3,"cleanExpiredFiles":'true'}
 \
 --type retention
```

このコマンドは、指定されたカタログ、データベース、およびリージョンの、指定されたテーブルの保持設定を更新します。キーパラメータは次のとおりです。

- snapshotRetentionPeriodInDays – 期限切れになるまでのスナップショットを保持する日数。デフォルト値は 1 です。
- numberOfSnapshotsToRetain – 保持期間より古い場合でも保持するスナップショットの最小数。デフォルト値は 5 です。
- cleanExpiredFiles – スナップショットの有効期限が切れた後に、期限切れのデータファイルを削除するかどうかを示すブール値。デフォルト値は `true` です。

  true に設定した場合、古いスナップショットがテーブルメタデータから削除されると、基盤となるファイルが削除されます。このパラメータを false に設定した場合、古いスナップショットがテーブルメタデータから削除されても、基盤となるファイルは孤立ファイルとしてストレージに残ります。

API

テーブルオプティマイザーを更新するときは、`UpdateTableOptimizer` API を使用できます。この API を使用すると、圧縮、保持、または孤立ファイル削除のために既存のテーブルオプティマイザーの設定を更新できます。リクエストパラメータには以下が含まれます。

- catalogId (必須): テーブルを含むカタログの ID
- databaseName (オプション): テーブルを含むデータベースの名前
- tableName (オプション): テーブルの名前
- type (必須): テーブルオプティマイザーのタイプ (compaction、retention、または orphan_file_deletion)

- retentionConfiguration (必須): ロールの ARN、有効ステータス、保持設定、孤立ファイル削除設定など、テーブルオプティマイザーの更新された設定。

スナップショット保持オプティマイザーの無効化

AWS Glue コンソールまたは AWS CLI を使用して、特定の Apache Iceberg テーブルのスナップショット保持オプティマイザーを無効にすることができます。

Console

スナップショットの保持を無効にするには

1. AWS Management Consoleにサインインし、AWS Glue コンソール (https://console.aws.amazon.com/glue/) を開きます。
2. [データカタログ]、[テーブル] の順に選択します。テーブルリストから、スナップショットの保持用のオプティマイザーを無効にする Iceberg テーブルを選択します。
3. [テーブルの詳細] ページの下部で、[アクション] の下にある [スナップショットの保持]、[テーブルの最適化]、[無効にする] を選択します。

   ページの右上隅にある [アクション] メニューから、[最適化] で [無効にする] を選択することもできます。
4. 確認メッセージで [無効化] を選択します。スナップショット保持オプティマイザーは後から再度有効にできます。

   確認後、スナップショット保持オプティマイザーが無効になり、スナップショット保持のステータスが `Not enabled` に戻ります。

AWS CLI

次の例で、アカウント ID を、有効な AWS アカウント ID に置き換えます。データベース名とテーブル名を、実際の Iceberg テーブル名とデータベース名に置き換えます。`roleArn` を、保持オプティマイザーの実行に必要なアクセス許可を持つ IAM ロールの AWS リソースネーム (ARN) と IAM ロールの実際の名前に置き換えます。

```
aws glue update-table-optimizer \
  --catalog-id 123456789012 \
  --database-name iceberg_db \
  --table-name iceberg_table \
```

```
  --table-optimizer-configuration
 '{"roleArn":"arn:aws:iam::123456789012:role/optimizer_role", "vpcConfiguration":
{"glueConnectionName":"glue_connection_name"}, "enabled":'false'}'\
  --type retention
```

AWS API

UpdateTableOptimizer オペレーションを呼び出して、特定のテーブルのスナップショット保持オプティマイザーを無効にします。

孤立ファイルの削除

AWS Glue Data Catalog では、Iceberg テーブルから孤立ファイルを削除できます。孤立ファイルは、指定されたテーブルの場所の Amazon S3 データソースに存在し、Iceberg テーブルメタデータによって追跡されず、設定された経過時間制限よりも古いファイルです。これらの孤立ファイルは、圧縮、パーティションの削除、テーブルの書き換えなどのオペレーションにより、時間の経過と共に蓄積され、不要なストレージ領域を占める可能性があります。

AWS Glue の孤立ファイル削除オプティマイザーは、テーブルメタデータと実際のデータファイルをスキャンして孤立ファイルを識別し、それらを削除してストレージ領域を再利用します。

データカタログで孤立ファイル削除テーブルオプティマイザーを作成することで、孤立ファイルの削除を開始できます。

> **Important**
>
> デフォルトでは、孤立ファイルを削除すると、AWS Glue テーブルの場所全体のファイルを評価します。サブプレフィックスを設定して評価の範囲を制限することができますが、テーブルの場所に他のデータソースやテーブルのファイルが含まれていないことを確認する必要があります。テーブルの場所が他のデータソースと重複している場合、関係のないファイルが孤立ファイルとして識別されて削除される可能性があります。

トピック

- 孤立ファイルの削除の有効化
- 孤立ファイル削除オプティマイザーの更新
- 孤立ファイルの削除の無効化

孤立ファイルの削除の有効化

AWS Glue コンソール、AWS CLI、または AWS API を使用して、データカタログ内の Apache Iceberg テーブルの孤立ファイルの削除を有効にすることができます。新しいテーブルの場合は、テーブル形式として Apache Iceberg を選択し、テーブルの作成時に孤立ファイル削除オプティマイザーを有効にすることができます。新しいテーブルでは、スナップショットの保持はデフォルトで無効になっています。

Console

孤立ファイルの削除を有効にするには

1. https://console.aws.amazon.com/glue/ で AWS Glue コンソールを開き、データレイク管理者、テーブル作成者、またはテーブルに対する `glue:UpdateTable` および `lakeformation:GetDataAccess` の許可を付与されたユーザーとしてサインインします。
2. ナビゲーションペインの [データカタログ] で [テーブル] を選択します。
3. [テーブル] ページで、孤立ファイルの削除を有効にする Iceberg テーブルを選択します。

   ページの下部にある [テーブル最適化] タブを選択し、[アクション] から [孤立ファイルの削除]、[有効にする] を選択します。

   ページの右上隅にある [アクション] メニューから、[最適化] で [有効にする] することもできます。
4. [最適化を有効にする] ページで、[最適化オプション] の [孤立ファイルの削除] を選択します。
5. [デフォルト設定] を使用する場合、すべての孤立ファイルは 3 日後に削除されます。孤立ファイルを特定の日数保持する場合は、[設定をカスタマイズ] を選択します。
6. 次に、孤立ファイルを削除するために必要なアクセス許可を持つ IAM ロールを選択します。
7. Iceberg テーブルオプティマイザーが、特定の仮想プライベートクラウド (VPC) から Amazon S3 バケットにアクセスする必要があるセキュリティポリシー設定がある場合は、AWS Glue ネットワーク接続を作成するか、既存のバケットを使用します。

   AWS Glue VPC 接続をまだ設定していない場合は、AWS Glue コンソールまたは AWS CLI/SDK を使用して[コネクタ用の接続を作成する]セクションの手順に従って、新しい接続を作成します。
8. [設定をカスタマイズ] を選択した場合は、[孤立ファイルの削除設定] で、削除前にファイルを保持する日数を入力します。

9. [最適化を有効にする] を選択します。

AWS CLI

AWS Glue で Iceberg テーブルの孤立ファイルの削除を有効にするには、タイプが orphan_file_deletion のテーブルオプティマイザーを作成し、enabled フィールドを true に設定する必要があります。AWS CLI を使用して Iceberg テーブルの孤立ファイル削除オプティマイザーを作成するには、次のコマンドを使用します。

```
aws glue create-table-optimizer \
 --catalog-id 123456789012 \
 --database-name iceberg_db \
 --table-name iceberg_table \
 --table-optimizer-configuration
 '{"roleArn":"arn:aws:iam::123456789012:role/optimizer_role","enabled":true,
 "vpcConfiguration":{
"glueConnectionName":"glue_connection_name"}, "orphanFileDeletionConfiguration":
{"icebergConfiguration":{"orphanFileRetentionPeriodInDays":3, "location":'S3
 location'}}}'\
 --type orphan_file_deletion
```

このコマンドにより、指定した Iceberg テーブルの孤立ファイル削除オプティマイザーが作成されます。キーパラメータは次のとおりです。

- roleArn – S3 バケットと Glue リソースへのアクセス許可を持つ IAM ロールの ARN。
- enabled – オプティマイザーを有効にするには、true に設定します。
- orphanFileRetentionPeriodInDays – 孤立ファイルが削除されるまでの保持日数 (最小で 1 日)。
- type – 孤立ファイル削除オプティマイザーを作成するには、orphan_file_deletion に設定します。

テーブルオプティマイザーを作成した後は、孤立ファイルの削除が定期的に (有効のままにした場合は 1 日に 1 回) 実行されます。list-table-optimizer-runs API を使用して実行を確認できます。孤立ファイルの削除ジョブでは、テーブルの Iceberg メタデータで追跡されていないファイルが識別されて削除されます。

API

CreateTableOptimizer オペレーションを呼び出して、特定のテーブルの孤立ファイル削除オプティマイザーを作成します。

孤立ファイル削除オプティマイザーの更新

AWS Glue コンソール、AWS CLI、または UpdateTableOptimizer オペレーションを使用して、孤立ファイルの保持期間やオプティマイザーが使用する IAM ロールを変更するなど、孤立ファイル削除オプティマイザーの設定を変更できます。

AWS Management Console

孤立ファイル削除オプティマイザーを更新するには

1. [データカタログ]、[テーブル] の順に選択します。テーブルリストから、孤立ファイル削除オプティマイザー設定を更新するテーブルを選択します。
2. [テーブルの詳細] ページの下部セクションで、[テーブルの最適化]、[編集] を選択します。
3. [最適化の編集] ページで、必要な変更を加えます。
4. [Save] を選択します。

AWS CLI

update-table-optimizer 呼び出しを使用して AWS Glue の孤立ファイル削除オプティマイザーを更新できます。呼び出しを使用できます。これにより、icebergConfiguration フィールドで OrphanFileDeletionConfiguration を変更して、更新後の OrphanFileRetentionPeriodInDays で孤立ファイルを保持する日数を設定し、孤立ファイルを削除する Iceberg テーブルの場所を指定できます。

```
aws glue update-table-optimizer \
 --catalog-id 123456789012 \
 --database-name iceberg_db \
 --table-name Iceberg_table \
 --table-optimizer-configuration
 '{"roleArn":"arn:aws:iam::123456789012:role/optimizer_role","enabled":true,
 "vpcConfiguration":
{"glueConnectionName":"glue_connection_name"},"orphanFileDeletionConfiguration":
{"icebergConfiguration":{"orphanFileRetentionPeriodInDays":5}}}' \
 --type orphan_file_deletion
```

API

UpdateTableOptimizer オペレーションを呼び出して、テーブルの孤立ファイルの削除オプティマイザーを更新します。

孤立ファイルの削除の無効化

AWS Glue コンソールまたは AWS CLI を使用して、特定の Apache Iceberg テーブルの孤立ファイル削除オプティマイザーを無効にすることができます。

Console

孤立ファイルの削除を無効にするには

1. [データカタログ]、[テーブル] の順に選択します。テーブルリストから、孤立ファイルの削除用のオプティマイザーを無効にする Iceberg テーブルを選択します。
2. [テーブルの詳細] ページの下部のセクションで、[テーブル最適化] を選択します。
3. [アクション] を選択し、[無効にする]、[孤立ファイルの削除] を選択します。

   また、[アクション] メニューから [最適化] の下部にある [無効化] を選択することもできます。
4. 確認メッセージで [無効化] を選択します。孤立ファイル削除オプティマイザーは後から再度有効にできます。

   確認後、孤立ファイル削除オプティマイザーが無効になり、孤立ファイル削除のステータスが `Not enabled` に戻ります。

AWS CLI

次の例で、アカウント ID を、有効な AWS アカウント ID に置き換えます。データベース名とテーブル名を、実際の Iceberg テーブル名とデータベース名に置き換えます。`roleArn` を、オプティマイザーの無効化に必要なアクセス許可を持つ IAM ロールの AWS リソースネーム (ARN) と IAM ロールの実際の名前に置き換えます。

```
aws glue update-table-optimizer \
  --catalog-id 123456789012 \
  --database-name iceberg_db \
  --table-name iceberg_table \
  --table-optimizer-configuration
 '{"roleArn":"arn:aws:iam::123456789012:role/optimizer_role", "enabled":'false'}'\
  --type orphan_file_deletion
```

API

UpdateTableOptimizer オペレーションを呼び出して、特定のテーブルのスナップショット保持オプティマイザーを無効にします。

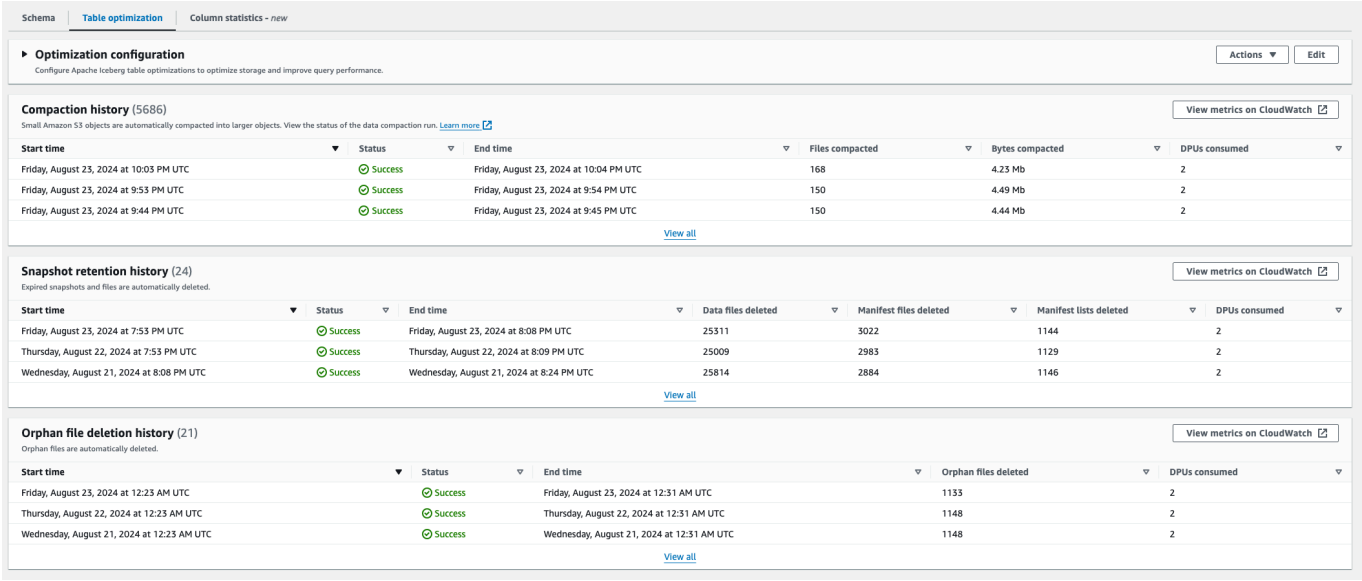
最適化の詳細の表示

Apache Iceberg テーブルの最適化ステータスは、AWS Glue コンソール、AWS CLI、または AWS API オペレーションで表示できます。

Console

Iceberg テーブルの最適化ステータスを表示するには (コンソール)

- [データカタログ] の [テーブル] リストから Iceberg テーブルを選択すると、AWS Glue コンソールで Iceberg テーブルの最適化ステータスを表示できます。[テーブルの最適化] の下。[すべて表示] を選択する



AWS CLI

AWS CLI を使用して最適化の詳細を表示できます。

次の例では、アカウントID を有効な AWS アカウント ID、データベース名、テーブル名を実際の Iceberg テーブル名に置き換えます。`type` には、最適化タイプを指定します。許容値は、`compaction`、`retention`、および `orphan-file-deletion` です。

- テーブルの前回の圧縮実行の詳細を取得するには

```
aws get-table-optimizer \
  --catalog-id 123456789012 \
  --database-name iceberg_db \
  --table-name iceberg_table \
```

```
  --type compaction
```

- 次の例を使用して、特定のテーブルのオプティマイザーの履歴を取得します。

```
aws list-table-optimizer-runs \
  --catalog-id 123456789012 \
  --database-name iceberg_db \
  --table-name iceberg_table \
  --type compaction
```

- 次の例は、複数のオプティマイザーの最適化実行と設定の詳細を取得する方法を示しています。最大 20 個のオプティマイザを指定できます。

```
aws glue batch-get-table-optimizer \
--entries '[{"catalogId":"123456789012", "databaseName":"iceberg_db",
 "tableName":"iceberg_table", "type":"compaction"}]'
```

API

- `GetTableOptimizer` オペレーションを使用して、前回実行したオプティマイザの詳細を取得します。
- 特定のテーブル上の特定のオプティマイザーの履歴を取得するには、`ListTableOptimizerRuns` オペレーションを使用します。1 回の API 呼び出しで 20 個のオプティマイザーを指定できます。
- BatchGetTableOptimizer オペレーションを使用して、アカウント内の複数のオプティマイザーの設定の詳細を取得します。

Amazon CloudWatch メトリクスの表示

テーブルオプティマイザーが正常に実行されると、サービスは最適化ジョブのパフォーマンスに関する Amazon CloudWatch メトリクスを作成します。[CloudWatch メトリクス] に移動し、[メトリクス]、[すべてのメトリクス] の順に選択できます。メトリクスは、特定の名前空間 (AWS Glue など)、テーブル名、またはデータベース名でフィルタリングできます。

詳細については、「Amazon CloudWatch ユーザーガイド」の「使用可能なメトリクスを表示する」を参照してください。

圧縮

- 圧縮されたバイト数
- 圧縮ファイル数
- ジョブに割り当てられた DPU 数
- ジョブの期間 (時間数)

スナップショット保持期限

- 削除されたデータファイルの数
- 削除されたマニフェストファイルの数
- 削除されたマニフェストリストの数
- ジョブの期間 (時間数)

孤立ファイルの削除

- 削除された孤立ファイルの数
- ジョブの期間 (時間数)

オプティマイザーの削除

AWS CLI または AWS API オペレーションを使用して、テーブルのオプティマイザーと関連するメタデータを削除できます。

テーブルの最適化履歴を削除するには、次の AWS CLI コマンドを実行します。オプティマイザーの `type` を、カタログ ID、データベース名、テーブル名と共に指定する必要があります。許容値は、`compaction`、`retention`、および `orphan_file_deletion` です。

```
aws glue delete-table-optimizer \
  --catalog-id 123456789012 \
  --database-name iceberg_db \
  --table-name iceberg_table \
  --type compaction
```

テーブルのオプティマイザーを削除するには、`DeleteTableOptimizer` オペレーションを使用します。

考慮事項と制限事項

このセクションでは、AWS Glue Data Catalog 内でテーブルオプティマイザーを使用する際に考慮すべき点について説明します。

マネージドデータ圧縮でサポートされる形式と制限事項

データ圧縮は、暗号化されたテーブルからのデータの読み取りなど、データの読み書きのためのさまざまなデータ型と圧縮形式をサポートしています。

データ圧縮は次をサポートします。

- 暗号化 - データ圧縮では、デフォルトの Amazon S3 暗号化 (SSE-S3) とサーバー側 KMS 暗号化 (SSE-KMS) のみがサポートされます。
- 圧縮戦略 – ビンパック、ソート、Z オーダーソート
- 基礎となるデータを保存する Amazon S3 バケットが別のアカウントにある場合、データカタログが存在するアカウントから圧縮を実行できます。これを実行するには、圧縮ロールが Amazon S3 バケットにアクセスできる必要があります。

データ圧縮は現在、次をサポートしていません。

- クロスアカウントテーブルでの圧縮 - クロスアカウントテーブルでは圧縮を実行できません。
- クロスリージョンテーブルでの圧縮 - クロスリージョンテーブルでは圧縮を実行できません。
- リソースのリンクでの圧縮の有効化
- Amazon S3 Express One Zone ストレージクラスのテーブル – S3 Express One Zone Iceberg テーブルでは圧縮を実行できません。
- Z オーダー圧縮戦略では、次のデータ型がサポートされていません:
  - 10 進数
  - TimestampWithoutZone

スナップショット保持と孤立ファイル削除オプティマイザに関する考慮事項

スナップショット保持と孤立ファイル削除のオプティマイザーには、次の考慮事項が適用されます。

- スナップショットの保持と孤立ファイルの削除プロセスでは、実行ごとに最大 1,000,000 個のファイルを削除できます。期限切れのスナップショットを削除するときに、削除の対象となるファイル

の数が 1,000,000 を超えると、そのしきい値を超える残りのファイルは、孤立ファイルとしてテーブルストレージに引き続き存在します。

- スナップショットは、保持するスナップショットの最小数と指定された保持期間という両方の基準が満たされた場合にのみ、スナップショット保持オプティマイザによって保持されます。
- スナップショット保持オプティマイザは、Apache Iceberg から期限切れのスナップショットメタデータを削除し、期限切れのスナップショットのタイムトラベルクエリを防ぎ、オプションで関連するデータファイルを削除します。
- オーファンファイル削除オプティマイザは、作成時間がオプティマイザ実行時からオーファンファイル削除保持期間より前である場合、Iceberg メタデータによって参照されなくなった孤立したデータとメタデータファイルを削除します。
- Apache Iceberg は、特定のスナップショット状態へのポインターという名前のブランチとタグを使用してバージョン管理を容易にします。各ブランチとタグは、それぞれのレベルで定義された保持ポリシーによって管理される、独自の独立したライフサイクルに従います。AWS Glue Data Catalog オプティマイザは、これらのライフサイクルポリシーを考慮し、指定された保持ルールを確実に遵守します。ブランチおよびタグレベルの保持ポリシーは、オプティマイザ設定よりも優先されます。

  詳細については、Apache Iceberg のドキュメントの「分岐とタグ付け」を参照してください。
- スナップショット保持と孤立ファイル削除オプティマイザは、設定されたパラメータに従ってクリーンアップの対象となるファイルを削除します。適切なバケットに S3 バージョニングポリシーとライフサイクルポリシーを実装することで、ファイル削除の制御を強化します。

  バージョニングの設定とライフサイクルルールの作成に関する詳細な手順については、「https://docs.aws.amazon.com/AmazonS3/latest/userguide/Versioning.html」を参照してください。
- 孤立ファイルについて適切な判断を行うには、指定されたテーブルの場所とサブパスが他のテーブルやデータソースと重複したり、他のテーブルやデータソースのデータを含んでいないことを確認してください。パスが重複すると、ファイルの意図しない削除によって回復不可能なデータ損失が発生する可能性があります。

テーブルオプティマイザーがサポートされているリージョン

AWS Glue Data Catalog のテーブル最適化機能 (圧縮、スナップショット保持、孤立ファイルの削除) は、次の AWS リージョン で使用できます。

- アジアパシフィック (東京)
- アジアパシフィック (ソウル)

- アジアパシフィック (ムンバイ)
- アジアパシフィック (シンガポール)
- アジアパシフィック (シドニー)
- アジアパシフィック (ジャカルタ)
- カナダ (中部)
- 欧州 (アイルランド)
- 欧州 (ロンドン)
- 欧州 (フランクフルト)
- 欧州 (ストックホルム)
- 米国東部 (バージニア北部)
- 米国東部 (オハイオ)
- 米国西部 (オレゴン)
- 南米 (サンパウロ)

Iceberg テーブルのクエリパフォーマンスの最適化

Apache Iceberg は、膨大な分析データセット用の高性能なオープンテーブル形式です。AWS Glue は、Iceberg テーブルの各列に対する個別の値の数 (NDV) の計算と更新をサポートしています。これらの統計により、大規模データセットを扱うデータエンジニアやサイエンティストにとって、クエリの最適化、データ管理、パフォーマンス効率が改善されます。

AWS Glue は、Iceberg テーブルの各列の個別の値の数を推定し、Iceberg テーブルスナップショットに関連付けられた Amazon S3 の Puffin ファイルに保存します。Puffin は、インデックス、統計、スケッチなどのメタデータを保存するように設計された Iceberg ファイル形式です。スナップショットに関連付けられた Puffin ファイルにスケッチを保存すると、トランザクションの一貫性と NDV 統計の鮮度が確保されます。

AWS Glue コンソールまたは AWS CLI を使用して列統計生成タスクを実行するように設定できます。プロセスを開始すると、AWS Glue はバックグラウンドで Spark ジョブを開始し、データカタログ内の AWS Glue テーブルメタデータを更新します。列統計は、AWS Glue コンソールもしくは AWS CLI を使用して、または GetColumnStatisticsForTable API オペレーションを呼び出すことによって表示できます。

> **Note**
>
> AWS Lake Formation の許可を使用してテーブルに対するアクセスを制御している場合、列統計タスクによって引き受けられるロールには、統計を生成するための完全なテーブルアクセスが必要です。

トピック

- 列統計を生成するための前提条件
- Iceberg テーブルの列統計の生成
- 以下も参照してください。

列統計を生成するための前提条件

Iceberg テーブルの列統計を生成または更新するために、統計生成タスクはユーザーに代わって AWS Identity and Access Management (IAM) ロールを引き受けます。ロールに付与された許可に基づいて、列統計生成タスクは Amazon S3 データストアからデータを読み取ることができます。

AWS Glue では、列統計生成タスクを設定するときに、`AWSGlueServiceRole` AWS 管理ポリシーと指定されたデータソースに必要なインラインポリシーを含むロールを作成できます。

列統計の生成に既存のロールを指定する場合は、そのロールに `AWSGlueServiceRole` ポリシーまたは同等のポリシー (またはこのポリシーのスコープダウンバージョン) と、必要なインラインポリシーが含まれてることを確認します。

必要な許可の詳細については、「列統計を生成するための前提条件」を参照してください。

Iceberg テーブルの列統計の生成

以下の手順に従って、 AWS Glue コンソールまたは AWS CLI または を使用して Data Catalog で統計を生成するスケジュールを設定します。または、 StartColumnStatisticsTaskRunオペレーションを実行します。

列統計を生成するには

1. で AWS Glue コンソールにサインインしますhttps://console.aws.amazon.com/glue/。
2. [データカタログ] の下の [テーブル] を選択します。

3. リストから Iceberg テーブルを選択します。
4. [アクション] メニューで、[列の統計] の [オンデマンドで生成] を選択します。

   [テーブル] ページの下のセクションにある [列統計] タブで [統計を生成] ボタンを選択することもできます。
5. [統計を生成] ページで、統計の生成の詳細を入力します。スケジュールでの列統計の生成 セクションのステップ 6～11 に従って、Iceberg テーブルの統計生成のスケジュールを設定します。

   また、「」の手順に従って、列統計をオンデマンドで生成することもできます。 オンデマンドでの列統計の生成

   > **Note**
   >
   > [サンプリング] オプションは Iceberg テーブルでは使用できません。

   AWS Glue は、Amazon S3 ロケーションで指定されたスナップショット ID にコミットされた新しい Puffin ファイルに対して Iceberg テーブルの各列の個別の値の数を計算します。

以下も参照してください。

- 列統計の表示
- 列統計タスクの実行の表示
- 列統計タスクの実行の停止
- 列統計の削除

データカタログの管理

AWS Glue Data Catalog は、Amazon S3 データセットの構造メタデータと運用メタデータを保存する中央メタデータリポジトリです。データカタログを効果的に管理することは、データ品質、パフォーマンス、セキュリティ、ガバナンスを維持する上で重要です。

これらのデータカタログ管理プラクティスを理解して適用することで、データランドスケープの進化に合わせて、メタデータが正確でパフォーマンスが高く、安全で、十分に管理されるようにすることができます。

このセクションでは、データカタログ管理の以下の側面について説明します。

- テーブルスキーマとパーティションの更新: データの進化に伴い、データカタログで定義されているテーブルスキーマまたはパーティション構造を更新する必要が生じる場合があります。AWS Glue ETL を使用してプログラムでこれらの更新を行う方法の詳細については、「AWS Glue ETL ジョブを使用してデータカタログのスキーマを更新し、新規パーティションを追加する」を参照してください。
- 列統計の管理: 正確な列統計は、クエリプランの最適化とパフォーマンスの向上に役立ちます。列統計を生成、更新、管理する方法については、「列統計を使用したクエリのパフォーマンスの最適化」を参照してください。
- データカタログの暗号化: 機密メタデータを保護するために、AWS Key Management Service (AWS KMS) を使用してデータカタログを暗号化できます。このセクションでは、データカタログの暗号化を有効にして管理する方法について説明します。
- AWS Lake Formation を使用したデータカタログの保護: Lake Formation には、データレイクのセキュリティとアクセスコントロールに対する包括的なアプローチが用意されています。Lake Formation を使用して、データカタログおよび基盤となるデータへのアクセスを保護および管理できます。

トピック

- AWS Glue ETL ジョブを使用してデータカタログのスキーマを更新し、新規パーティションを追加する
- 列統計を使用したクエリのパフォーマンスの最適化
- Data Catalog の暗号化
- Lake Formation を使用したデータカタログの保護
- AWS Glue での AWS Glue Data Catalog ビューの使用

AWS Glue ETL ジョブを使用してデータカタログのスキーマを更新し、新規パーティションを追加する

抽出、変換、ロード (ETL) ジョブによって、ターゲットデータストアに新しいテーブルパーティションが作成される場合があります。データセットスキーマは、時間の経過とともに AWS Glue Data Catalog スキーマから進化し、拡散する可能性があります。AWS GlueETL ジョブには、ETL スクリプト内で使用できるいくつかの機能が用意され、Data Catalog でスキーマおよびパーティションを更新できるようになりました。これらの機能を使用すると、クローラを再実行することなく、Data Catalog での ETL 作業の結果を確認できます。

新しいパーティション

AWS Glue Data Catalog で新しいパーティションを表示するには、次のいずれかを実行します。

- ジョブが終了したら、クローラを再実行し、クローラの完了時にコンソールで新しいパーティションを表示します。
- ジョブが終了すると、クローラを再実行することなく、コンソールで新しいパーティションがすぐに表示されます。この機能は、次の例に示すように、ETL スクリプトに数行のコードを追加して有効にすることができます。このコードは enableUpdateCatalog 引数を使用して、新しいパーティションの作成時のジョブ実行中に Data Catalog を更新する必要があることを示します。

方式 1

オプション引数に enableUpdateCatalog と partitionKeys を渡します。

Python

```
additionalOptions = {"enableUpdateCatalog": True}
additionalOptions["partitionKeys"] = ["region", "year", "month", "day"]


sink = glueContext.write_dynamic_frame_from_catalog(frame=last_transform,
 database=<target_db_name>,

 table_name=<target_table_name>, transformation_ctx="write_sink",

 additional_options=additionalOptions)
```

Scala

```
val options = JsonOptions(Map(
    "path" -> <S3_output_path>,
    "partitionKeys" -> Seq("region", "year", "month", "day"),
    "enableUpdateCatalog" -> true))
val sink = glueContext.getCatalogSink(
    database = <target_db_name>,
    tableName = <target_table_name>,
    additionalOptions = options)sink.writeDynamicFrame(df)
```

方式 2

enableUpdateCatalog に partitionKeys と getSink() を渡して、DataSink オブジェクトで setCatalogInfo() を呼び出します。

Python

```
sink = glueContext.getSink(
    connection_type="s3",
    path="<S3_output_path>",
    enableUpdateCatalog=True,
    partitionKeys=["region", "year", "month", "day"])
sink.setFormat("json")
sink.setCatalogInfo(catalogDatabase=<target_db_name>,
 catalogTableName=<target_table_name>)
sink.writeFrame(last_transform)
```

Scala

```
val options = JsonOptions(
   Map("path" -> <S3_output_path>,
       "partitionKeys" -> Seq("region", "year", "month", "day"),
       "enableUpdateCatalog" -> true))
val sink = glueContext.getSink("s3", options).withFormat("json")
sink.setCatalogInfo(<target_db_name>, <target_table_name>)
sink.writeDynamicFrame(df)
```

クローラを再実行することなく、AWS Glue ETLジョブ自体を使用して、Data Catalog での新しいカタログテーブルの作成、変更されたスキーマによる既存のテーブルの更新、および新しいテーブルパーティションの追加が可能になりました。

テーブルスキーマの更新

Data Catalog テーブルのスキーマを上書きする場合は、次のいずれかを実行します。

- ジョブが終了したら、クローラを再実行し、テーブル定義も更新するようにクローラが設定されていることを確認します。クローラが終了したら、新しいパーティションをスキーマの更新とともにコンソールに表示します。詳細については、「API を使用したクローラの設定」を参照してください。
- ジョブが終了すると、クローラを再実行することなく、コンソールで変更済みスキーマがすぐに表示されます。この機能は、次の例に示すように、ETL スクリプトに数行のコードを追

加して有効にすることができます。このコードでは enableUpdateCatalog を true に設定し、updateBehavior を UPDATE_IN_DATABASE に設定しています。これは、ジョブ実行中にスキーマを上書きし、Data Catalog に新しいパーティションを追加することを示します。

Python

```
additionalOptions = {
    "enableUpdateCatalog": True,
    "updateBehavior": "UPDATE_IN_DATABASE"}
additionalOptions["partitionKeys"] = ["partition_key0", "partition_key1"]

sink = glueContext.write_dynamic_frame_from_catalog(frame=last_transform,
 database=<dst_db_name>,
    table_name=<dst_tbl_name>, transformation_ctx="write_sink",
    additional_options=additionalOptions)
job.commit()
```

Scala

```
val options = JsonOptions(Map(
    "path" -> outputPath,
    "partitionKeys" -> Seq("partition_0", "partition_1"),
    "enableUpdateCatalog" -> true))
val sink = glueContext.getCatalogSink(database = nameSpace, tableName = tableName,
 additionalOptions = options)
sink.writeDynamicFrame(df)
```

テーブルスキーマが上書きされないようにして、新しいパーティションを追加する場合は、updateBehavior 値を LOG に設定することもできます。updateBehavior のデフォルト値は UPDATE_IN_DATABASE です。そのため、明示的に定義しない場合、テーブルスキーマは上書きされます。

enableUpdateCatalog が true に設定されていない場合、updateBehavior で選択したオプションに関係なく、ETL ジョブは Data Catalog 内のテーブルを更新しません。

新しいテーブルの作成

同じオプションを使用して、Data Catalog で新しいテーブルを作成することもできます。setCatalogInfo を使用して、データベースと新しいテーブル名を指定できます。

Python

```
sink = glueContext.getSink(connection_type="s3", path="s3://path/to/data",
    enableUpdateCatalog=True, updateBehavior="UPDATE_IN_DATABASE",
    partitionKeys=["partition_key0", "partition_key1"])
sink.setFormat("<format>")
sink.setCatalogInfo(catalogDatabase=<dst_db_name>, catalogTableName=<dst_tbl_name>)
sink.writeFrame(last_transform)
```

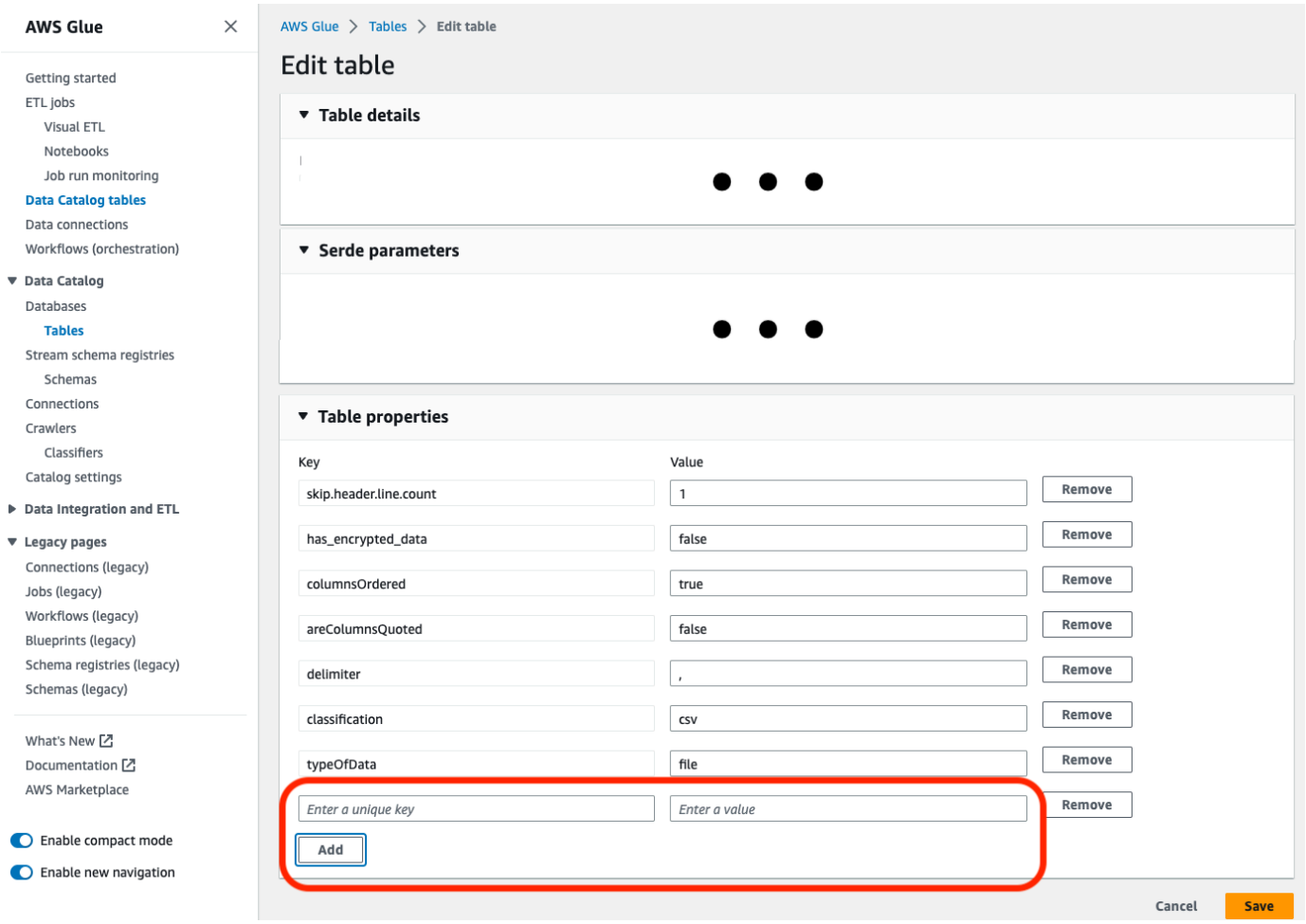
Scala

```
val options = JsonOptions(Map(
    "path" -> outputPath,
    "partitionKeys" -> Seq("<partition_1>", "<partition_2>"),
    "enableUpdateCatalog" -> true,
    "updateBehavior" -> "UPDATE_IN_DATABASE"))
val sink = glueContext.getSink(connectionType = "s3", connectionOptions =
 options).withFormat("<format>")
sink.setCatalogInfo(catalogDatabase = “<dst_db_name>”, catalogTableName =
 “<dst_tbl_name>”)
sink.writeDynamicFrame(df)
```

制限事項

次の制限事項に注意してください。

- Amazon Simple Storage Service (Amazon S3) のターゲットのみがサポートされています。
- `enableUpdateCatalog` 機能は、管理対象テーブルでは、サポートされていません。
- `json`、`csv`、`avro`、および `parquet` の形式のみがサポートされます。
- `parquet` 分類でテーブルを作成または更新するには、AWS Glue に最適化された DynamicFrames 用 parquet ライターを使用する必要があります。これは、次のいずれかの方法で実現できます。
  - `parquet` 分類を使用してカタログ内の既存のテーブルを更新する場合は、更新前にテーブルでは `"useGlueParquetWriter"` テーブルプロパティを `true` に設定する必要があります。このプロパティは、AWS Glue API/SDK、コンソール、または Athena DDL ステートメントから設定できます。



カタログテーブルプロパティを設定したら、次のコードスニペットを使用して新しいデータでカタログテーブルを更新します。

```
glueContext.write_dynamic_frame.from_catalog(
    frame=frameToWrite,
    database="dbName",
    table_name="tableName",
    additional_options={
        "enableUpdateCatalog": True,
        "updateBehavior": "UPDATE_IN_DATABASE"
    }
)
```

- テーブルがカタログ内にまだ存在しない場合は、connection_type="s3" を用いた getSink() メソッドをスクリプトで使用して、データを Amazon S3 に書き込むとともに、テーブルとそのパーティションをカタログに追加します。ワークフローに適切な partitionKeys と compression を指定します。

```
s3sink = glueContext.getSink(
    path="s3://bucket/folder/",
    connection_type="s3",
    updateBehavior="UPDATE_IN_DATABASE",
    partitionKeys=[],
    compression="snappy",
    enableUpdateCatalog=True
)

s3sink.setCatalogInfo(
    catalogDatabase="dbName", catalogTableName="tableName"
)

s3sink.setFormat("parquet", useGlueParquetWriter=True)
s3sink.writeFrame(frameToWrite)
```

  - glueparquet フォーマットの値は、AWS Glue parquet ライターを有効にする従来のメソッドです。

- updateBehavior を LOG に設定した場合、DynamicFrame スキーマが Data Catalog テーブルのスキーマに定義されている列のサブセットと同等であるか、またはそのサブセットを含んでいる場合にのみ、新しいパーティションが追加されます。
- スキーマの更新は、パーティション化されていないテーブル (「partitionKeys」オプションを使用していない) ではサポートされていません。
- partitionKeys は、ETL スクリプトで渡されるパラメータと Data Catalog テーブルスキーマの partitionKeys で同等で、同じ順序でなければなりません。
- この機能は、現在、更新スキーマがネストされているテーブルの更新/作成をサポートしていません (例えば、構造体の内部の配列)。

詳細については、the section called “AWS Glue for Spark” を参照してください。

ETL ジョブでの MongoDB 接続の操作

MongoDB の接続を作成し、その接続を AWS Glue ジョブで使用します。詳細については、「AWS Glue プログラミングガイド」の「the section called “MongoDB 接続”」を参照してください。接続の url、username、および password は MongoDB 接続に保存されます。ETL ジョブスクリプトで glueContext.getCatalogSource という additionalOptions パラメータを使用して、その他のオプションを指定できます。その他のオプションには次のようなものがあります。

- `database`: (必須) 読み込み元の MongoDB データベース。
- `collection`: (必須) 読み込み元の MongoDB コレクション。

ETL ジョブスクリプト内に `database` および `collection` 情報を書き込むと、複数のジョブに同じ接続を使用できます。

1. AWS Glue Data Catalog 接続を MongoDB データソース用に作成します。接続パラメータの説明については、「"connectionType": "mongodb"」を参照してください。接続は、コンソール、API、または CLI を使用して作成できます。
2. AWS Glue Data Catalog でデータベースを作成して MongoDB データのテーブル定義を格納します。詳細については、「データベースの作成」を参照してください。
3. MongoDB に接続するための接続内の情報を使用して、MongoDB 内のデータをクロールするクローラを作成します。クローラによって、ジョブで使用する MongoDB データベース内のテーブルを記述するテーブルが AWS Glue Data Catalog に作成されます。詳細については、「クローラーを使用したデータカタログへの入力 」を参照してください。
4. カスタムスクリプトを使用してジョブを作成します。ジョブは、コンソール、API、または CLI を使用して作成できます。詳細については、「AWS Glue でジョブを追加する」を参照してください。
5. ジョブ用のデーターゲットを選択します。データターゲットを表すテーブルは、Data Catalog で定義することも、ジョブを実行するときに作成することもできます。ジョブを作成するときに、ターゲットの場所を選択します。ターゲットに接続が必要な場合、接続もジョブ内で参照されます。ジョブに複数のデーターゲットが必要な場合、後でスクリプトを編集して追加できます。
6. ジョブと生成されたスクリプトの引数を指定することで、ジョブ処理環境をカスタマイズします。

   Data Catalog で定義されたテーブル構造に基づいて、MongoDB データベースから `DynamicFrame` を作成する例を次に示します。このコードは `additionalOptions` を使用して、追加のデータソースの情報を指定しています。

   Scala

```
val resultFrame: DynamicFrame = glueContext.getCatalogSource(
        database = catalogDB,
        tableName = catalogTable,
        additionalOptions = JsonOptions(Map("database" -> DATABASE_NAME,
                "collection" -> COLLECTION_NAME))
```

```
      ).getDynamicFrame()
```

Python

```
glue_context.create_dynamic_frame_from_catalog(
        database = catalogDB,
        table_name = catalogTable,
        additional_options = {"database":"database_name",
            "collection":"collection_name"})
```

7. オンデマンドで、またはトリガーを使用してジョブを実行します。

列統計を使用したクエリのパフォーマンスの最適化

追加のデータパイプラインを設定することなく、Parquet、ORC、JSON、ION、CSV、XML などのデータ形式で AWS Glue Data Catalog テーブルの列レベルの統計を計算できます。列統計は、列内の値に関するインサイトを得ることで、データプロファイルを理解するのに役立ちます。

データカタログは、列値 (最小値、最大値、null 値の合計、個別の値の合計、値の平均長、真値の合計出現数など) の統計の生成をサポートします。Amazon Redshift や Amazon Athena などの AWS 分析サービスでは、これらの列統計を使用してクエリの実行プランを生成でき、クエリのパフォーマンスを向上させる最適なプランを選択できます。

列統計の生成には次の 3 つのシナリオがあります。

Auto

AWS Glue は、カタログレベルで列統計の自動生成をサポートしているため、AWS Glue Data Catalog で新しいテーブルの統計を自動的に生成できます。

Scheduled (スケジュール済み)

AWS Glue は、列統計の生成を定期的なスケジュールで自動的に実行できるようにサポートします。

スケジュールされた統計計算では、列統計タスクは、最小、最大、平均などのテーブルレベルの統計全体を新しい統計で更新し、クエリエンジンに正確かつ最新の統計を提供して、クエリの実行を最適化します。

[オンデマンド]

このオプションを使用して、必要に応じて列統計をオンデマンドで生成します。これは、アドホック分析や統計をすぐに計算する必要がある場合に役立ちます。

AWS Glue コンソール、AWS CLI、AWS Glue API オペレーションを使用して列統計の生成タスクを実行するように設定できます。プロセスを開始すると、AWS Glue はバックグラウンドで Spark ジョブを開始し、データカタログ内の AWS Glue テーブルメタデータを更新します。列統計は、AWS Glue コンソールもしくは AWS CLI を使用して、または GetColumnStatisticsForTable API オペレーションを呼び出すことによって表示できます。

> **Note**
>
> Lake Formation の許可を使用してテーブルに対するアクセスを制御している場合、列統計タスクによって引き受けられるロールには、統計を生成するための完全なテーブルアクセスが必要です。

トピック

- 列統計を生成するための前提条件
- 列統計の自動生成
- スケジュールでの列統計の生成
- オンデマンドでの列統計の生成
- 列統計の表示
- 列統計タスクの実行の表示
- 列統計タスクの実行の停止
- 列統計の削除
- 考慮事項と制限

列統計を生成するための前提条件

列統計を生成または更新するために、統計生成タスクはユーザーに代わって AWS Identity and Access Management (IAM) ロールを引き受けます。ロールに付与された許可に基づいて、列統計生成タスクは Amazon S3 データストアからデータを読み取ることができます。

AWS Glue では、列統計生成タスクを設定するときに、AWSGlueServiceRole AWS 管理ポリシーと指定されたデータソースに必要なインラインポリシーを含むロールを作成できます。

列統計の生成に既存のロールを指定する場合は、そのロールに AWSGlueServiceRole ポリシーまたは同等のポリシー (またはこのポリシーのスコープダウンバージョン)、および必要なインラインポリシーが含まれてることを確認します。次のステップに従って、新しい IAM ロールを作成します。

> **Note**
>
> Lake Formation によって管理されるテーブルの統計を生成するには、統計の生成に使用される IAM ロールに完全なテーブルアクセスが必要です。

AWS Glue では、列統計生成タスクを設定するときに、AWSGlueServiceRole AWS 管理ポリシーと指定されたデータソースに必要なインラインポリシーを含むロールを作成できます。また、ロールを作成して、以下のポリシーにリストされているアクセス許可をアタッチし、そのロールを列統計生成タスクに追加することもできます。

列統計を生成するための IAM ロールを作成するには

1. IAM ロールを作成するには、「AWS Glue の IAM ロールを作成する」を参照してください。
2. 既存のロールを更新するには、IAM コンソールで、列統計の生成プロセスで使用されている IAM ロールにアクセスします。
3. [許可を追加] セクションで、[ポリシーをアタッチ] を選択します。新しく開いたブラウザウィンドウで、AWSGlueServiceRole AWS マネージドポリシーを選択します。
4. Amazon S3 のデータの場所からデータを読み取るための許可も含める必要があります。

   [Add permissions] (アクセス許可の追加) セクションで、[Create policy] (ポリシーの作成) を選択します。新しく開いたブラウザウィンドウで、ロールで使用する新しいポリシーを作成します。
5. [ポリシーを作成] ページで、[JSON] タブを選択します。次の JSON コードをポリシーエディタフィールドにコピーします。

   > **Note**
   >
   > 次のポリシーでは、アカウント ID を有効な AWS アカウント に置き換え、region をテーブルのリージョンに置き換え、bucket-name を Amazon S3 バケット名に置き換えます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "S3BucketAccess",
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket",
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::<bucket-name>/*",
                "arn:aws:s3:::<bucket-name>"
            ]
        }
     ]
}
```

6. (オプション) Lake Formation の許可を使用してデータに対するアクセスを提供している場合、IAM ロールには `lakeformation:GetDataAccess` 許可が必要です。

```
{
    "Version": "2012-10-17",
    "Statement": [
       {
            "Sid": "LakeFormationDataAccess",
            "Effect": "Allow",
            "Action": "lakeformation:GetDataAccess",
            "Resource": [
                "*"
            ]
        }
    ]
}
```

Amazon S3 のデータの場所が Lake Formation に登録されており、テーブルに対する `IAM_ALLOWED_PRINCIPALS` グループ許可が、列統計生成タスクによって引き受けられる IAM ロールに付与されていない場合、そのロールには、テーブルに対する Lake Formation `ALTER` お

よび DESCRIBE 許可が必要です。Amazon S3 バケットの登録に使用されるロールには、テーブルに対する Lake Formation INSERT および DELETE 許可が必要です。

Amazon S3 のデータの場所が Lake Formation に登録されておらず、テーブルに対する IAM_ALLOWED_PRINCIPALS グループ許可が IAM ロールに付与されていない場合、そのロールには、テーブルに対する Lake Formation ALTER、DESCRIBE、INSERT、および DELETE 許可が必要です。

7. カタログレベルの Automatic statistics generation オプションを有効にしている場合、IAM ロールには、デフォルトのデータカタログに対する glue:UpdateCatalog アクセス許可または Lake Formation ALTER CATALOG アクセス許可が必要になります。カタログのプロパティは、GetCatalog オペレーションを使用して確認できます。

8. (オプション) 暗号化された Amazon CloudWatch Logs を書き込む列統計生成タスクには、キーポリシーで次の許可が必要です。

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Sid": "CWLogsKmsPermissions",
        "Effect":  "Allow",
        "Action": [
            "logs:CreateLogGroup",
            "logs:CreateLogStream",
            "logs:PutLogEvents",
            "logs:AssociateKmsKey"
        ],
        "Resource": [
            "arn:aws:logs:<region>:111122223333:log-group:/aws-glue:*"
        ]
    },
    {
        "Sid": "KmsPermissions",
        "Effect": "Allow",
        "Action": [
            "kms:GenerateDataKey",
            "kms:Decrypt",
            "kms:Encrypt"
        ],
        "Resource": [
        "arn:aws:kms:<region>:111122223333:key/"arn of key used for ETL cloudwatch
 encryption"
        ],
```

```
        "Condition": {
            "StringEquals": {
                "kms:ViaService": ["glue.<region>.amazonaws.com"]
            }
        }
    }
    ]
}
```

9. 列統計の実行に使用するロールには、そのロールに対する iam:PassRole のアクセス許可が必要です。

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": [
            "iam:PassRole"
        ],
        "Resource": [
            "arn:aws:iam::111122223333:role/<columnstats-role-name>"
        ]
    }]
}
```

10. 列統計を生成するための IAM ロールを作成する場合、そのロールには、サービスがそのロールを引き受けることを可能にする次の信頼ポリシーも必要です。

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

```
}
```

列統計の自動生成

列統計を自動で生成することにより、AWS Glue Data Catalog の新しいテーブルで、統計をスケジュールして自動的に計算することができます。統計の自動生成を有効にすると、データカタログが、特定のデータ形式 (Parquet、JSON、CSV、XML、ORC、ION、Apache Iceberg など) を持つ新しいテーブルを、それぞれのバケットパスとともに検出します。1 回限りのカタログ設定を使用すると、データカタログがこれらのテーブルの統計を生成します。

データレイクの管理者は、Lake Formation コンソールでデフォルトのカタログを選択し、`Optimization configuration` オプションを使用してテーブル統計を有効にすることで、統計の生成を設定できます。データカタログで新しいテーブルを作成したり既存のテーブルを更新したりすると、データカタログが Apache Iceberg テーブルの個別値の数 (NDV) と、サポートされている他のファイル形式の NULL、最大長、最小長、平均長など、追加の統計を毎週収集します。

テーブルレベルで統計生成を設定している場合、またはテーブルの統計生成の設定を以前に削除している場合、これらテーブル固有の設定は、列統計の自動生成のデフォルトのカタログ設定より優先されます。

統計の自動生成タスクは、テーブル内のレコードの 50% を分析して統計を計算します。列統計の自動生成により、データカタログは Amazon Athena や Amazon Redshift Spectrum などのクエリエンジンで使用できるメトリクスを週単位で保守でき、クエリのパフォーマンスを向上させ潜在的なコストを削減することができます。AWS Glue API またはコンソールを使用して統計の生成をスケジュールできるため、手動による介入なしにプロセスを自動的に実行できます。

トピック

- カタログレベルでの統計の自動生成を有効化
- 自動化されたテーブルレベルの設定を表示
- カタログレベルの列統計の生成の無効化

カタログレベルでの統計の自動生成を有効化

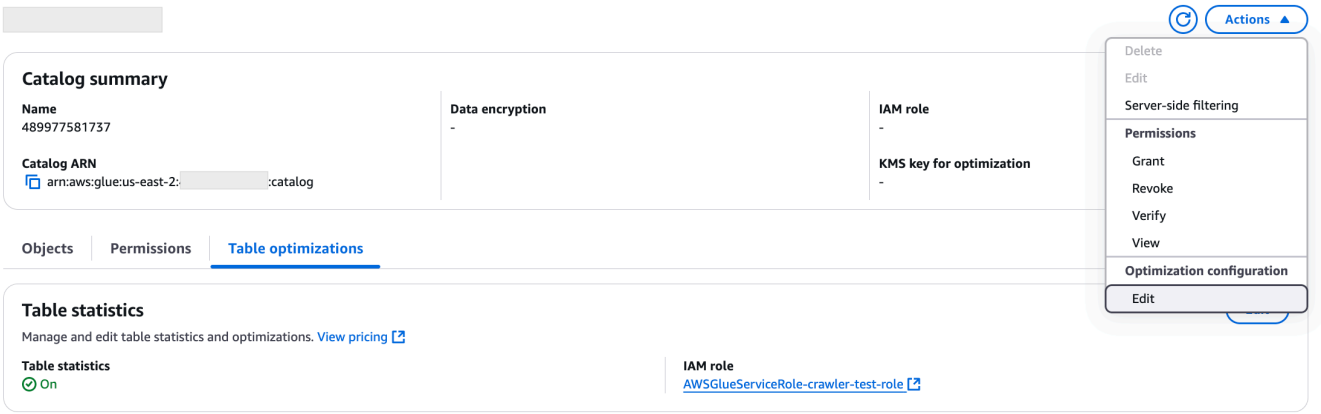
データカタログのすべての新しい Apache Iceberg テーブルと、OTF 以外のテーブル (Parquet、JSON、CSV、XML、ORC、ION) 形式のテーブルに対して、列統計の自動生成を有効にできます。テーブルを作成すると、列統計の設定を手動で明示的に更新することもできます。

データカタログの設定を更新してカタログレベルを有効にするには、使用する IAM ロールに、ルートカタログに対する `glue:UpdateCatalog` アクセス許可または AWS Lake Formation `ALTER CATALOG` アクセス許可が必要になります。カタログのプロパティは、`GetCatalog` API を使用して確認できます。

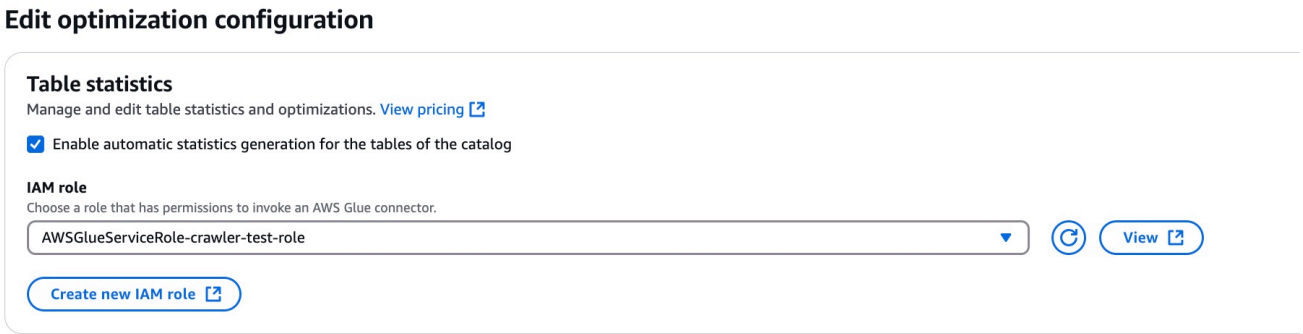
AWS Management Console

アカウントレベルで列統計の自動生成を有効にするには

1. Lake Formation コンソール (https://console.aws.amazon.com/lakeformation/) を開きます。
2. 左のナビゲーションペインで [カタログ] を選択します。
3. [カタログの概要] ページの [最適化の設定] で [編集] を選択します。



4. [テーブル最適化の設定] ページで、[カタログのテーブルの自動統計生成を有効にする] を選択します。



5. 既存の IAM ロールを選択するか、列統計タスクを実行するための必要なアクセス許可を持つ新しいロールを作成します。
6. [送信] を選択します。

AWS CLI

AWS CLI を使ってカタログレベルの統計収集を有効にすることもできます。AWS CLI を使用してテーブルレベルの統計収集を設定するには、次のコマンドを実行します。

```
aws glue update-catalog --cli-input-json '{
    "name": "123456789012",
    "catalogInput": {
        "description": "Updating root catalog with role arn",
        "catalogProperties": {
            "customProperties": {
                "ColumnStatistics.RoleArn": "arn:aws:iam::"123456789012":role/
service-role/AWSGlueServiceRole",
                "ColumnStatistics.Enabled": "true"
            }
        }
    }
}'
```

上記のコマンドは AWS Glue の UpdateCatalog オペレーションを呼び出します。これにより、カタログレベルの統計を生成するために、次のキーと値のペアを持つ CatalogProperties 構造が取り込まれます。

- ColumnStatistics.RoleArn – カタログレベルの統計生成でトリガーされるすべてのタスクで使用する IAM ロール ARN
- ColumnStatistics.Enabled – カタログレベルの設定が有効か無効かを示すブール値

自動化されたテーブルレベルの設定を表示

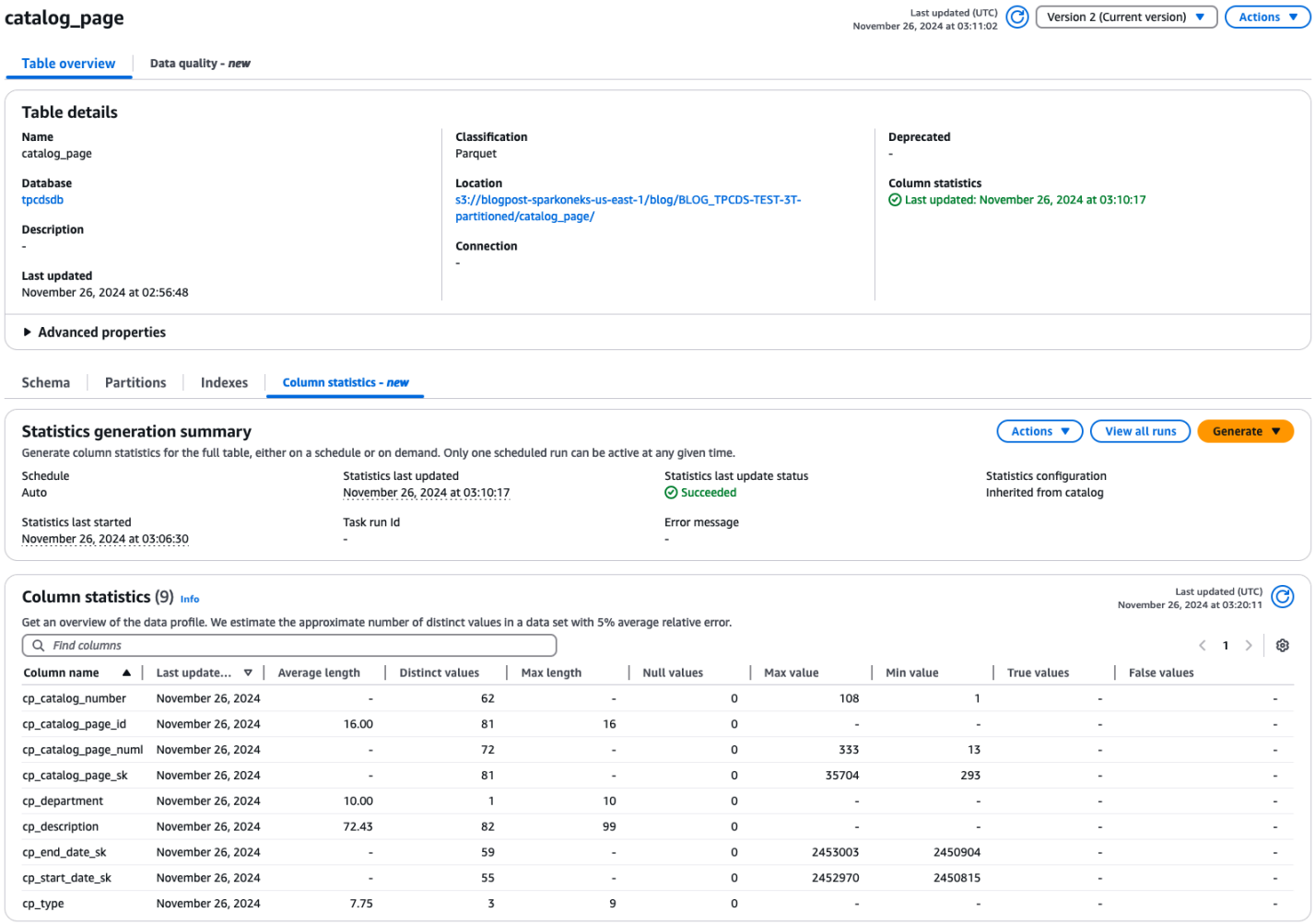
カタログレベルの統計収集が有効になっている場合、Apache Hive テーブルまたは Apache Iceberg テーブルが、AWS Management Console、SDK、AWS Glue クローラー のいずれかで CreateTable または UpdateTable API を介して作成または更新されるたびに、そのテーブルに対して同等のテーブルレベルの設定が作成されます。

統計の自動生成が有効になっているテーブルは、次のいずれかのプロパティに従う必要があります。

- org.apache.hadoop で始まり、TableType が EXTERNAL_TABLE である InputSerdeLibrary を使用する

- com.amazon.ion で始まり、TableType が EXTERNAL_TABLE である InputSerdeLibrary を使用する
- 自身のパラメータ構造に table_type: "ICEBERG" を含める

テーブルを作成または更新したら、テーブルの詳細を検証して統計の生成を確定できます。Statistics generation summary では、Schedule プロパティが AUTO に設定され、Statistics configuration 値が Inherited from catalog を示しています。次の設定を持つテーブル設定は、Glue によって内部で自動的にトリガーされます。



カタログレベルの列統計の生成の無効化

新しいテーブルの列統計の自動生成は、AWS Lake Formation コンソール、glue:UpdateCatalogSettings API、glue:DeleteColumnStatisticsTaskSettings API のいずれかを使用して無効にすることができます。

アカウントレベルで列統計の自動生成を無効にするには

1. Lake Formation コンソール (https://console.aws.amazon.com/lakeformation/) を開きます。
2. 左のナビゲーションペインで [カタログ] を選択します。
3. [カタログの概要] ページの [最適化の設定] で [編集] を選択します。
4. [テーブル最適化の設定] ページで、[カタログのテーブルの自動統計生成を有効にする] の選択を解除します。
5. [Submit] を選択します。

スケジュールでの列統計の生成

AWS Glue コンソール、、 AWS CLIまたは CreateColumnStatisticsTaskSettingsオペレーションを使用して AWS Glue Data Catalog で列統計を生成するスケジュールを設定するには、次の手順に従います。

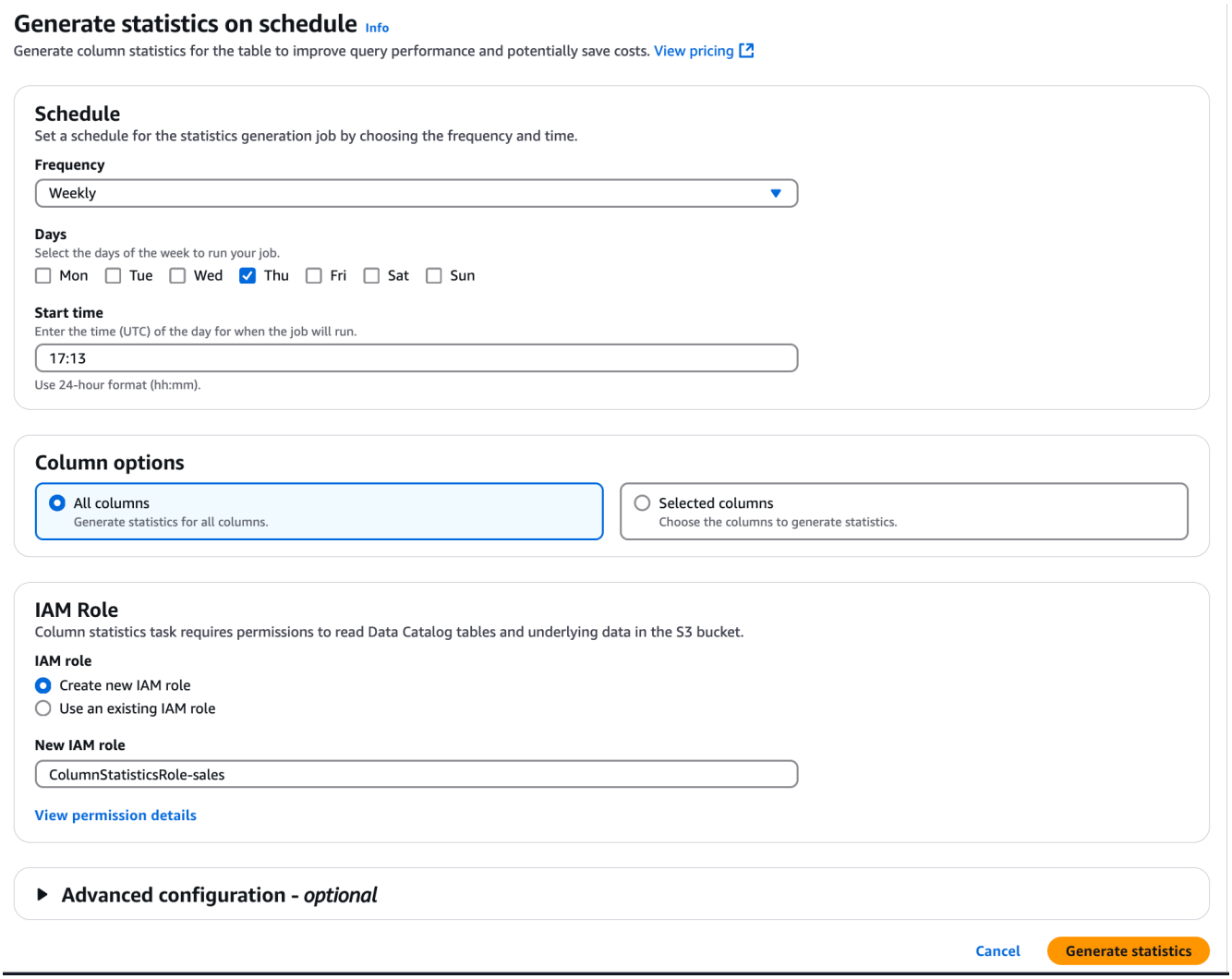
Console

コンソールを使用して列統計を生成するには

1. で AWS Glue コンソールにサインインしますhttps://console.aws.amazon.com/glue/。
2. データカタログテーブルを選択します。
3. リストからテーブルを選択します。
4. Tables ページの下部にある列統計タブを選択します。
5. アクション の列統計でスケジュールどおりに生成を選択することもできます。
6. スケジュールで統計を生成するページで、頻度と開始時刻を選択して、列統計タスクを実行するための定期的なスケジュールを設定します。スケジュールを指定するには、時間単位、日単位、週単位の頻度を選択するか、cron 式を定義できます。

   cron 式は、スペースで区切られた 6 つのフィールドで構成されるスケジュールパターンを表す文字列です。* * * * <minute> <hour> <day of month> <month> <day of week> <year> たとえば、毎日午前 0 時にタスクを実行する場合、cron 式は 0 0 * * ? *

   詳細については、「Cron 式」を参照してください。



7. 次に、列オプションを選択して統計を生成します。

   - すべての列 - テーブル内のすべての列の統計を生成するには、このオプションを選択します。
   - 選択された列 – 特定の列の統計を生成するには、このオプションを選択します。ドロップダウンリストから列を選択できます。

8. IAM ロールを選択するか、統計を生成するアクセス許可を持つ既存のロールを作成します。列統計を生成するには、このロール AWS Glue を前提とします。

   より迅速なアプローチは、 AWS Glue コンソールにロールを作成させることです。作成するロールは、列統計を生成するために特別に用意されており、 `AWSGlueServiceRole` AWS マネージドポリシーと、指定されたデータソースに必要なインラインポリシーが含まれています。

列統計を生成するための既存のロールを指定する場合は、AWSGlueServiceRoleポリシーまたは同等のもの (またはこのポリシーのスコープダウンバージョン) と、必要なインラインポリシーが含まれていることを確認してください。

9. (オプション) 次に、ログの保管中の暗号化を有効にするセキュリティ設定を選択します。
10. (オプション) サンプルサイズは、統計を生成するためにテーブルから特定の割合の行のみを指定することで選択できます。デフォルトは、すべての行です。上矢印と下矢印を使用してパーセント (%) の値を増減します。

    正確な統計を計算するには、テーブル内のすべての行を含めることをお勧めします。近似値が許容される場合にのみ、サンプル行を使用して列統計を生成します。
11. 統計の生成を選択して、列統計の生成タスクを実行します。

AWS CLI

次の AWS CLI 例を使用して、列統計の生成スケジュールを作成できます。database-name、table-name、および role は必須パラメータであり、オプションのパラメータはスケジュール column-name-list、catlate-id、sample-size、および security-configuration です。

```
aws glue create-column-statistics-task-settings \
 --database-name 'database_name' \
 --table-name table_name \
 --role 'arn:aws:iam::123456789012:role/stats-role' \
 --schedule 'cron(0 0-5 14 * * ?)' \
 --column-name-list 'col-1' \
 --catalog-id '123456789012' \
 --sample-size '10.0 ' \
 --security-configuration 'test-security'
```

StartColumnStatisticsTaskRun オペレーションを呼び出すことで、列統計を生成することもできます。

列統計生成のスケジュールの管理

の列統計生成のスケジュールの更新、開始、停止、削除などのスケジューリングオペレーションを管理できます AWS Glue。コンソール、 AWS CLIまたはAWS Glue 列統計APIオペレーションを使用して AWS Glue 、これらのタスクを実行できます。

トピック

- 列統計生成スケジュールの更新
- 列統計生成のスケジュールの停止
- 列統計生成のスケジュールの再開
- 列統計生成スケジュールの削除

列統計生成スケジュールの更新

スケジュールを更新して、列統計生成タスクの作成後にトリガーできます。 AWS Glue コンソールを使用するか AWS CLI、 UpdateColumnStatisticsTaskSettings オペレーションを実行してテーブルのスケジュールを更新できます。スケジュールタイプ (オンデマンド、またはスケジュール済み) やその他のオプションパラメータなど、既存のスケジュールのパラメータを変更できます。

AWS Management Console

列統計生成タスクの設定を更新するには

1. で AWS Glue コンソールにサインインしますhttps://console.aws.amazon.com/glue/。
2. 更新するテーブルをテーブルリストから選択します。
3. テーブルの詳細ページの下部セクションで、列統計 を選択します。
4. アクション で、編集を選択してスケジュールを更新します。
5. スケジュールに必要な変更を行い、 の保存 を選択します。

AWS CLI

コンソールで AWS Glueの統計生成機能を使用していない場合は、 `update-column-statistics-task-settings` コマンドを使用してスケジュールを手動で更新できます。次の例は、 AWS CLIを使用して列統計を更新する方法を示しています。

```
aws glue update-column-statistics-task-settings \
 --database-name 'database_name' \
 --table-name 'table_name' \
 --role arn:aws:iam::123456789012:role/stats_role \
 --schedule 'cron(0 0-5 16 * * ?)' \
 --column-name-list 'col-1' \
 --sample-size '20.0' \
 --catalog-id '123456789012'\
```

```
 --security-configuration 'test-security'
```

列統計生成のスケジュールの停止

増分統計が不要になった場合は、スケジュールされた生成を停止してリソースとコストを節約できます。スケジュールを一時停止しても、以前に生成された統計には影響しません。スケジュールは都合の良いときに再開できます。

AWS Management Console

列統計生成タスクのスケジュールを停止するには

1. AWS Glue コンソールで、データカタログのテーブルを選択します。
2. 列統計を含むテーブルを選択します。
3. [テーブルの詳細] ページで、[列統計] を選択します。
4. アクション で、スケジュールされた生成 、一時停止 を選択します。
5. 確認するには一時停止を選択します。

AWS CLI

を使用して列統計タスク実行スケジュールを停止するには AWS CLI、次のコマンドを使用します。

```
aws glue stop-column-statistics-task-run-schedule \
 --database-name ''database_name' \
 --table-name 'table_name'
```

database_name と をtable_name、列統計タスク実行スケジュールを停止するデータベースとテーブルの実際の名前に置き換えます。

列統計生成のスケジュールの再開

統計生成スケジュールを一時停止 AWS Glue した場合は、都合の良いときにスケジュールを再開できます。コンソール AWS CLIまたは StartColumnStatisticsTaskRunScheduleオペレーションを使用して AWS Glue スケジュールを再開できます。

AWS Management Console

列統計生成のスケジュールを再開するには

1. AWS Glue コンソールで、データカタログのテーブルを選択します。
2. 列統計を含むテーブルを選択します。
3. [テーブルの詳細] ページで、[列統計] を選択します。
4. アクション で、スケジュールされた生成 を選択し、再開 を選択します。
5. 再開を選択して確認します。

AWS CLI

database_name と をtable_name、列統計タスク実行スケジュールを停止するデータベースとテーブルの実際の名前に置き換えます。

```
aws glue start-column-statistics-task-run-schedule \
 --database-name 'database_name' \
 --table-name 'table_name'
```

列統計生成スケジュールの削除

最適なクエリパフォーマンスのために統計を維持すること up-to-dateが一般的に推奨されますが、自動生成スケジュールの削除が有益な特定のユースケースがあります。

- データが比較的静的なままの場合、既存の列統計は長期間にわたって正確であり続け、頻繁な更新の必要性が軽減される可能性があります。スケジュールを削除すると、変更されていないデータに関する統計の再生成に関連する不要なリソース消費とオーバーヘッドを防ぐことができます。
- 統計の生成を手動で制御する場合。自動スケジュールを削除することで、管理者は特定の間隔で、または重要なデータ変更後に列統計を選択的に更新し、メンテナンス戦略とリソース配分のニーズに合わせてプロセスを調整できます。

AWS Management Console

列統計生成のスケジュールを削除するには

1. AWS Glue コンソールで、データカタログのテーブルを選択します。
2. 列統計を含むテーブルを選択します。

3. [テーブルの詳細] ページで、[列統計] を選択します。
4. アクション で、スケジュールされた生成 を選択し、 を削除します。
5. 削除を選択して確認します。

AWS CLI

`database_name` と をtable_name、列統計タスク実行スケジュールを停止するデータベースとテーブルの実際の名前に置き換えます。

列統計スケジュールは、 DeleteColumnStatisticsTaskSettingsAPIオペレーションまたは を使用して削除できます AWS CLI。次の例は、 AWS Command Line Interface () を使用して列統計を生成するためのスケジュールを削除する方法を示していますAWS CLI。

```
aws glue delete-column-statistics-task-settings \
    --database-name 'database_name' \
    --table-name 'table_name'
```

オンデマンドでの列統計の生成

スケジュールを設定せずに AWS Glue Data Catalog 、テーブルタスクの列統計タスクをオンデマンドで実行できます。このオプションは、アドホック分析や、統計をすぐに計算する必要がある場合に役立ちます。

AWS Glue コンソールまたは を使用して Data Catalog テーブルの列統計をオンデマンドで生成するには、次の手順に従います AWS CLI。

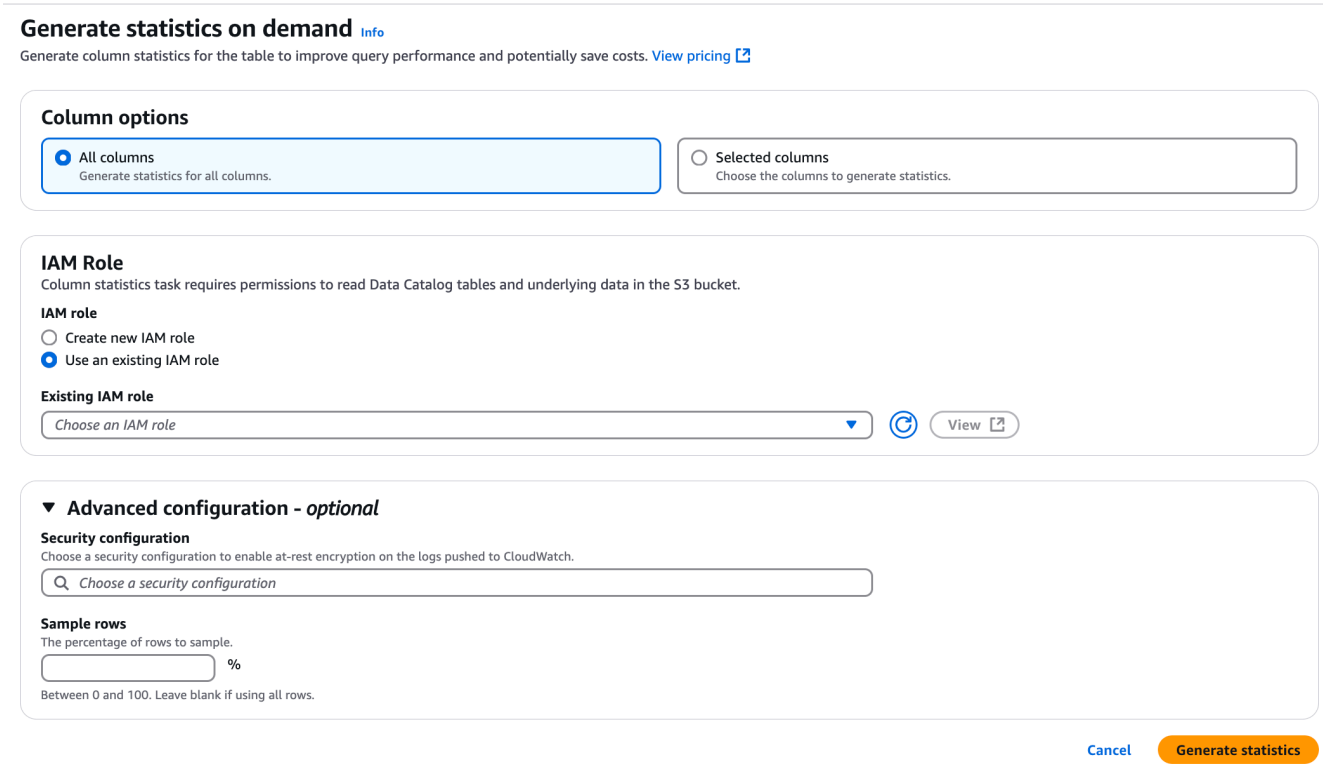
AWS Management Console

コンソールを使用して列統計を生成するには

1. で AWS Glue コンソールにサインインしますhttps://console.aws.amazon.com/glue/。
2. データカタログテーブルを選択します。
3. リストからテーブルを選択します。
4. [アクション] メニューで [統計を生成] を選択します。

   また、テーブルページの下部にある列統計タブで、 の生成、オンデマンドの生成オプションを選択することもできます。

5. のステップ 7 ~ 11 に従ってスケジュールでの列統計の生成、テーブルの列統計を生成します。
6. [統計を生成] ページで、次のオプションを指定します。



- すべての列 - テーブル内のすべての列の統計を生成するには、このオプションを選択します。
- 選択された列 – 特定の列の統計を生成するには、このオプションを選択します。ドロップダウンリストから列を選択できます。
- IAM role – 列統計生成タスクを実行するために必要なアクセス許可ポリシーを持つ新しいIAMロールを作成するを選択します。[アクセス許可の詳細を表示] を選択して、ポリシーステートメントを確認します。リストからIAMロールを選択することもできます。必要な許可の詳細については、「列統計を生成するための前提条件」を参照してください。

  AWS Glue は、統計を生成するために指定したロールのアクセス許可を引き受けます。

  にロールを提供する方法の詳細については AWS Glue、「 のアイデンティティベースのポリシー」を参照してください AWS Glue。
- (オプション) 次に、ログの保管中の暗号化を有効にするセキュリティ設定を選択します。
- サンプル行 – テーブルから特定の割合の行のみを選択して統計を生成します。デフォルトは、すべての行です。上矢印と下矢印を使用してパーセント (%) の値を増減します。

> **Note**
>
> 正確な統計を計算するには、テーブル内のすべての行を含めることをお勧めします。近似値が許容される場合にのみ、サンプル行を使用して列統計を生成します。

[統計を生成] を選択してタスクを実行します。

AWS CLI

このコマンドは、指定されたテーブルの列統計タスク実行をトリガーします。データベース名、テーブル名、統計を生成するアクセス許可を持つIAMロールを指定し、必要に応じて統計計算の列名とサンプルサイズの割合を指定する必要があります。

```
aws glue start-column-statistics-task-run \
    --database-name 'database_name \
    --table-name 'table_name' \
    --role 'arn:aws:iam::123456789012:role/stats-role' \
    --column-name 'col1','col2'  \
    --sample-size 10.0
```

このコマンドは、指定したテーブルの列統計を生成するタスクを開始します。

オンデマンドでの列統計の更新

列統計を維持すること up-to-dateは、クエリオプティマイザが効率的な実行計画を生成し、クエリパフォーマンスの向上、リソース消費の削減、システム全体のパフォーマンスの向上を実現する上で不可欠です。このプロセスは、一括ロードや大規模な変更など、既存の統計が時代遅れになる可能性のある重大なデータ変更の後、特に重要です。

列統計を更新するには、コンソールから統計を生成タスクを明示的に実行する必要があります。AWS Glue データカタログは統計を自動的に更新しません。

コンソールで AWS Glueの統計生成機能を使用していない場合は、UpdateColumnStatisticsForTableAPIオペレーションまたは を使用して列統計を手動で更新できますAWS CLI。次の例は、 AWS CLIを使用して列統計を更新する方法を示しています。

```
aws glue update-column-statistics-for-table --cli-input-json:
```

```
{
    "CatalogId": "111122223333",
    "DatabaseName": "database_name",
    "TableName": "table_name",
    "ColumnStatisticsList": [
        {
            "ColumnName": "col1",
            "ColumnType": "Boolean",
            "AnalyzedTime": "1970-01-01T00:00:00",
            "StatisticsData": {
                "Type": "BOOLEAN",
                "BooleanColumnStatisticsData": {
                    "NumberOfTrues": 5,
                    "NumberOfFalses": 5,
                    "NumberOfNulls": 0
                }
            }
        }
    ]
}
```

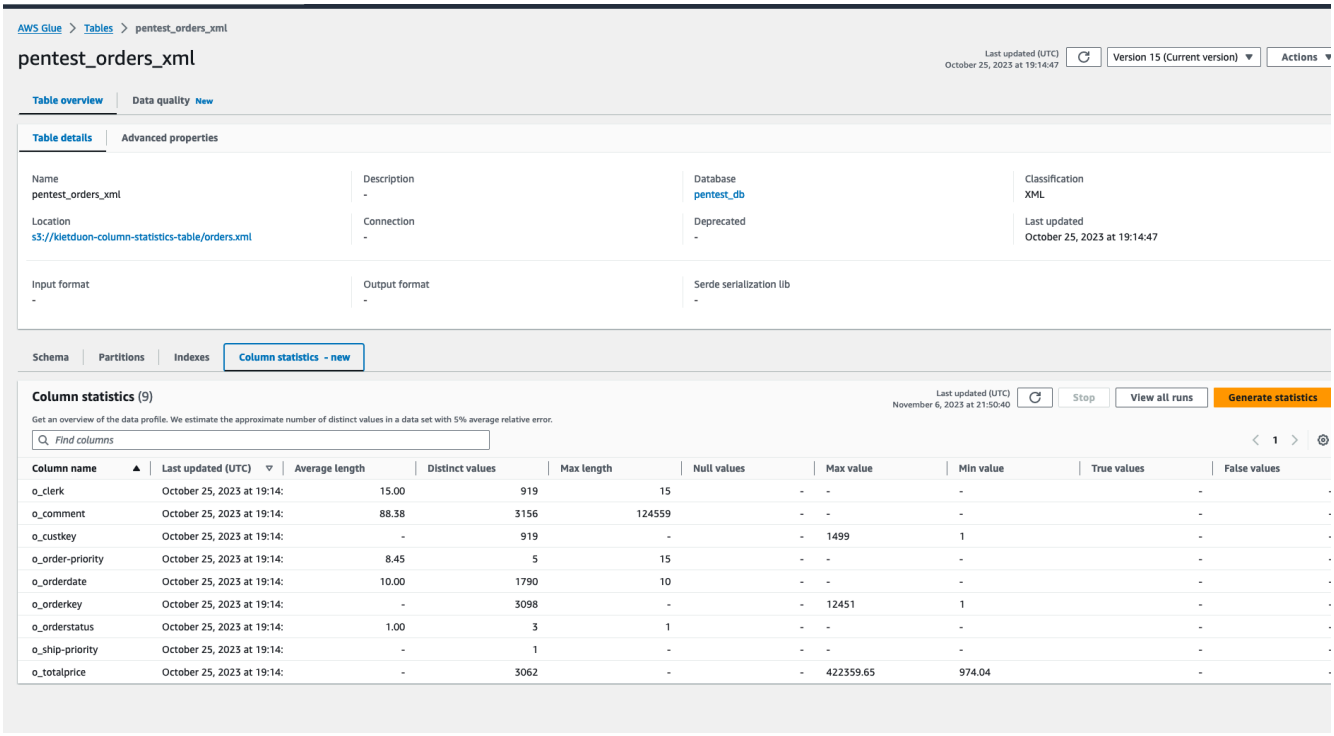
列統計の表示

統計が正常に生成された後、Data Catalog はこの情報をコストベースのオプティマイザ用に Amazon Athena および Amazon Redshift に保存し、クエリの実行時に最適な選択を行います。統計は列のタイプによって異なります。

AWS Management Console

テーブルの列統計を表示するには

- 列統計タスクを実行すると、[テーブルの詳細] ページの [列統計] タブにテーブルの統計が表示されます。



次の統計を使用できます。

- [列名]: 統計を生成するために使用される列名
- [最終更新日時]: 統計が生成された日時
- [平均長]: 列内の値の平均長
- [個別の値]: 列内の個別の値の合計数。列内の個別の値の数を 5% の相対誤差で推定します。
- [最大値]: 列内の最大値。
- [最小値]: 列内の最小値。
- [最大長]: 列内の最大値の長さ。
- [Null 値]: 列内の null 値の合計数。
- [True 値]: 列内の true 値の合計数。
- [False 値]: 列内の false 値の合計数。
- numFiles: テーブル内のファイルの合計数。この値は、[詳細プロパティ] タブで使用できます。

AWS CLI

次の例は、 AWS CLIを使用して列統計を取得する方法を示しています。

```
aws glue get-column-statistics-for-table \
    --database-name database_name \
    --table-name table_name \
    --column-names <column_name>
```

GetColumnStatisticsForTable API オペレーションを使用して列統計を表示することもできます。

列統計タスクの実行の表示

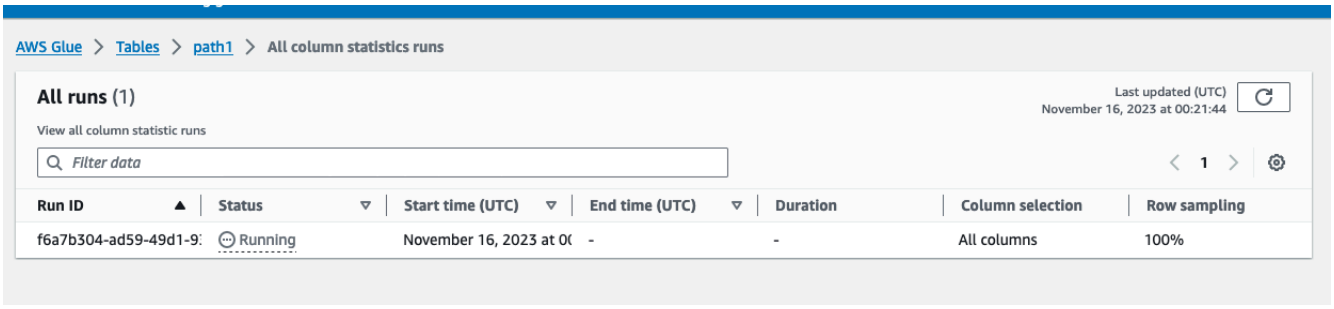
列統計タスクを実行したら、 AWS Glue コンソール AWS CLI または GetColumnStatisticsTaskRunsオペレーションを使用してテーブルのタスク実行の詳細を確認できます。

Console

列統計タスクの実行の詳細を表示するには

1. AWS Glue コンソールで、データカタログのテーブルを選択します。
2. 列統計を含むテーブルを選択します。
3. [テーブルの詳細] ページで、[列統計] を選択します。
4. [実行を表示] を選択します。

   指定されたテーブルに関連付けられたすべての実行に関する情報を確認できます。



AWS CLI

次の例では、DatabaseName と TableName の値を、実際のデータベースとテーブルの名前に置き換えます。

```
aws glue get-column-statistics-task-runs --input-cli-json file://input.json
{
```

```
    "DatabaseName": "database_name",
    "TableName": "table_name"
}
```

列統計タスクの実行の停止

AWS Glue コンソール AWS CLI または StopColumnStatisticsTaskRunオペレーションを使用して、テーブルの列統計タスクの実行を停止できます。

Console

列統計タスクの実行を停止するには

1. AWS Glue コンソールで、データカタログのテーブルを選択します。
2. 列統計タスクの実行が進行中のテーブルを選択します。
3. [テーブルの詳細] ページで、[列統計] を選択します。
4. [Stop] (停止) を選択します。

   実行が完了する前にタスクを停止すると、テーブルの列統計は生成されません。

AWS CLI

次の例では、DatabaseName と TableName の値を、実際のデータベースとテーブルの名前に置き換えます。

```
aws glue stop-column-statistics-task-run --input-cli-json file://input.json
{
    "DatabaseName": "database_name",
    "TableName": "table_name"
}
```

列統計の削除

列統計は、 DeleteColumnStatisticsForTableAPIオペレーションまたは を使用して削除できます AWS CLI。次の例は、 AWS Command Line Interface () を使用して列統計を削除する方法を示していますAWS CLI。

```
aws glue delete-column-statistics-for-table \
    --database-name 'database_name' \
    --table-name 'table_name' \
    --column-name 'column_name'
```

考慮事項と制限

列統計の生成には、次の考慮事項と制限事項が適用されます。

考慮事項

- サンプリングを使用して統計を生成すると、実行時間は短縮されますが、不正確な統計が生成される可能性があります。
- データカタログは、異なるバージョンの統計を保存しません。
- テーブルごとに一度に 1 つの統計生成タスクのみを実行できます。
- データカタログに登録されたカスタマー AWS KMS キーを使用してテーブルが暗号化されている場合、 は同じキー AWS Glue を使用して統計を暗号化します。

列統計タスクは、次の場合に統計の生成をサポートします。

- IAM ロールに完全なテーブルアクセス許可 (IAM または Lake Formation) がある場合。
- Lake Formation ハイブリッドアクセスモードを使用してIAM、ロールにテーブルに対するアクセス許可がある場合。

列統計タスクは、次についての統計の生成をサポートしていません。

- Lake Formation セルベースのアクセスコントロールを持つテーブル
- トランザクションデータレイク - Linux 基盤 Delta Lake、Apache Hudi
- フェデレーテッドデータベース内のテーブル - Hive メタストア、Amazon Redshift データ共有
- ネストされた列、配列、および構造体のデータ型。
- 別のアカウントから共有されるテーブル

Data Catalog の暗号化

AWS Key Management Service (AWS KMS) によって管理される暗号化キーを使用して、保管中の AWS Glue Data Catalog に保存されているメタデータを保護できます。データカタログ設定を使用して、新しいデータカタログのデータカタログ暗号化を有効にできます。必要に応じて、既存のデータカタログの暗号化を有効または無効にできます。有効にすると、AWS Glue はカタログに書き込まれたすべての新しいメタデータを暗号化しますが、既存のメタデータは暗号化されません。

データカタログの暗号化の詳細については、「Data Catalog の暗号化」を参照してください。

Lake Formation を使用したデータカタログの保護

AWS Lake Formation は、AWS で簡単にセキュアなデータレイクを構築できるサービスです。きめ細かなアクセスコントロール許可を定義することで、データレイクを作成および安全に管理するための一元的な場所を提供します。Lake Formation は、データカタログを使用して、テーブル定義、スキーマ情報、データアクセスコントロール設定など、データレイクに関するメタデータを保存および取得します。

メタデータテーブルまたはデータベースの Amazon S3 データロケーションを Lake Formation に登録し、それを使用してデータカタログリソースに対するメタデータレベルのアクセス許可を定義できます。また、Lake Formation を使用して、統合された分析エンジンに代わって、Amazon S3 に保存されている基盤となるデータへのストレージアクセス許可も管理できます。

詳細については、「AWS Lake Formation とは」を参照してください。

AWS Glue での AWS Glue Data Catalog ビューの使用

AWS Glue Data Catalog ではビューを作成および管理できます。これは一般的に AWS Glue Data Catalog ビューと呼ばれます。これらのビューは複数の SQL クエリエンジンをサポートしているため、Amazon Athena、Amazon Redshift、AWS Glue などのさまざまな AWS サービス間で同じビューにアクセスできるため便利です。Apache Iceberg、Apache Hudi、Delta Lake に基づくビューを使用できます。

データカタログでビューを作成することで、AWS Lake Formation でのリソース許可とタグベースのアクセスコントロールを使用して、データカタログビューへのアクセスを許可できます。このアクセスコントロール方法を使用すれば、ビューの作成時に参照したテーブルへの追加のアクセスを設定する必要はありません。アクセス許可を付与するこの方法は、definer セマンティクスと呼ばれ、これらのビューは definer ビューと呼ばれます。AWS Lake Formation でのアクセスコントロールの詳細は、「AWS Lake Formation デベロッパーガイド」の「Granting and revoking permissions on Data Catalog resources」を参照してください。

データカタログビューは、次のユースケースに役立ちます。

- 細やかなアクセス制御 – ユーザーが必要とするアクセス許可に基づいてデータアクセスを制限するビューを作成できます。例えば、データカタログのビューを使用して、人事 (HR) 部門に属さない従業員が個人を特定できる情報 (PII) を表示できないようにすることができます。
- 完全なビュー定義 – データカタログのビューに特定のフィルターを適用することで、ビュー内のデータレコードを常に完全な状態にすることができます。
- セキュリティの強化 – ビューの作成に使用するクエリ定義は完全である必要があります。これにより、データカタログのビューが有害なアクターによる SQL コマンドの影響を受けにくくなります。
- シンプルなデータ共有 – AWS Lake Formation でのアカウント間データ共有により、データを移動せずに他の AWS アカウントとデータを共有することができます。

データカタログビューの作成

Spark SQL による AWS CLI および AWS Glue ETL スクリプトを使用して、データカタログビューを作成できます。データカタログビューを作成するための構文には、ビュータイプを `MULTI DIALECT` に、`SECURITY` 述語を `DEFINER` に指定して定義者ビューを示すことが含まれます。

データカタログビューを作成する SQL ステートメントの例:

```
CREATE PROTECTED MULTI DIALECT VIEW database_name.catalog_view SECURITY DEFINER
AS SELECT order_date, sum(totalprice) AS price
FROM source_table
GROUP BY order_date;
```

データカタログビューを作成したら、当該ビューに対する AWS Lake Formation SELECT アクセス許可を持つ IAM ロールを使用して、Amazon Athena、Amazon Redshift、または AWS Glue ETL ジョブなどのサービスからクエリを実行できます。ビューで参照される基盤となるテーブルへのアクセスを許可する必要はありません。

データカタログビューの作成と設定の詳細については、「AWS Lake Formation デベロッパーガイド」の「Building AWS Glue Data Catalog views」を参照してください。

サポートされているビュー操作

以下のコマンドフラグメントは、データカタログビューを操作するさまざまな方法を示します。

CREATE VIEW

データカタログビューを作成します。以下は、既存のテーブルからビューを作成する例です。

```
CREATE PROTECTED MULTI DIALECT VIEW catalog_view
SECURITY DEFINER AS SELECT * FROM my_catalog.my_database.source_table
```

ALTER VIEW

使用可能な構文:

```
ALTER VIEW view_name [FORCE] ADD DIALECT AS query
ALTER VIEW view_name [FORCE] UPDATE DIALECT AS query
ALTER VIEW view_name DROP DIALECT
```

FORCE ADD DIALECT オプションを使用して、新しいエンジンダイアレクトに従ってスキーマとサブオブジェクトを強制的に更新できます。ただし、これを行うと、他のエンジンダイアレクトの更新にも FORCE を使用しないと、クエリエラーが発生する可能性があることに注意してください。サンプルを以下に示します。

```
ALTER VIEW catalog_view FORCE ADD DIALECTAS
SELECT order_date, sum(totalprice) AS priceFROM source_tableGROUP BY orderdate;
```

以下は、ダイアレクトを更新するためにビューを変更する方法を示します。

```
ALTER VIEW catalog_view UPDATE DIALECT AS
SELECT count(*) FROM my_catalog.my_database.source_table;
```

DESCRIBE VIEW

ビューを記述するために使用できる構文:

SHOW COLUMNS {FROM|IN} view_name [{FROM|IN} database_name] – ユーザーがビューの記述に必要な AWS Glue および AWS Lake Formation のアクセス許可を持っている場合、列を一覧表示できます。列を表示するためのいくつかのサンプルコマンドを以下に示します。

```
SHOW COLUMNS FROM my_database.source_table;
SHOW COLUMNS IN my_database.source_table;
```

DESCRIBE view_name – ユーザーがビューの記述に必要な AWS Glue および AWS Lake Formation のアクセス許可を持っている場合、列およびメタデータをビュー内で一覧表示できます。

DROP VIEW

使用可能な構文:

```
DROP VIEW [ IF EXISTS ] view_name
```

次のサンプルは、ビューを削除する前にビューが存在するかどうかをテストする DROP ステートメントを示します。

```
DROP VIEW IF EXISTS catalog_view;
```

SHOW CREATE VIEW view_name – 指定したビューを作成する SQL ステートメントを示します。以下は、データカタログビューの作成を示すサンプルです。

```
SHOW CREATE TABLE my_database.catalog_view;CREATE PROTECTED MULTI DIALECT VIEW
 my_catalog.my_database.catalog_view (
  net_profit,
  customer_id,
  item_id,
  sold_date)
TBLPROPERTIES (
  'transient_lastDdlTime' = '1736267222')
SECURITY DEFINER AS SELECT * FROM
my_database.store_sales_partitioned_lf WHERE customer_id IN (SELECT customer_id from
 source_table limit 10)
```

SHOW VIEWS

通常のビュー、マルチダイアレクトビュー (MDV)、Spark ダイアレクトのない MDV など、カタログ内のすべてのビューを一覧表示します。使用可能な構文は次のとおりです。

```
SHOW VIEWS [{ FROM | IN } database_name] [LIKE regex_pattern]:
```

ビューを表示するサンプルコマンドを以下に示します。

```
SHOW VIEWS IN marketing_analytics LIKE 'catalog_view*';
```

データカタログビューの作成と設定の詳細については、「AWS Lake Formation デベロッパーガイド」の「Building AWS Glue Data Catalog views」を参照してください。

データカタログビューをクエリする

データカタログビューの作成後は、ビューをクエリできるようになります。AWS Glue ジョブで設定された IAM ロールには、データカタログビューに対する Lake Formation SELECT アクセス許可が必要です。ビューで参照される基盤となるテーブルへのアクセスを許可する必要はありません。

すべてをセットアップしたら、ビューをクエリできます。たとえば、次のクエリを実行してビューにアクセスできます。

```
SELECT * from my_database.catalog_view LIMIT 10;
```

制限

データカタログビューを使用する場合は、次の制限を考慮してください。

- AWS Glue 5.0 以上では、データカタログビューのみを作成できます。
- Data Catalog ビュー定義者は、ビューによってアクセスされる基盤となるベーステーブルに対する SELECT アクセスを持っている必要があります。特定のベーステーブルに定義者ロールに対する Lake Formation フィルターが適用されている場合、データカタログビューの作成は失敗します。
- ベーステーブルには AWS Lake Formation 内の `IAMAllowedPrincipals` データレイクのアクセス許可を付与してはいけません。付与されている場合は、Multi Dialect views may only reference tables without IAMAllowedPrincipals permissions というエラーが表示されます。
- テーブルの Amazon S3 の場所は、AWS Lake Formation データレイクの場所として登録する必要があります。テーブルが登録されていない場合、エラー `Multi Dialect views may only reference AWS Lake Formation managed tables` が発生します。AWS Lake Formation で Amazon S3 の場所を登録する方法については、「AWS Lake Formation デベロッパーガイド」の「Registering an Amazon S3 location」を参照してください。
- PROTECTED データカタログビューのみを作成できます。UNPROTECTED ビューはサポートされていません。
- データカタログビュー定義では別の AWS アカウント内のテーブルを参照することはできません。また、別のリージョンにあるのと同じアカウントのテーブルを参照することはできません。
- アカウントまたはリージョン間でデータを共有するには、AWS Lake Formation リソースリンクを使用してビュー全体をクロスアカウントおよびクロスリージョンで共有する必要があります。

- ユーザー定義関数 (UDF) はサポートされていません。
- データカタログビューで他のビューを参照できません。

データカタログにアクセスする

AWS Glue Data Catalog (データカタログ) を使用すると、データを検出してこれを理解することができます。データカタログは、スキーマの定義、データ型、場所などのメタデータを一貫して維持する方法を提供します。次の方法を使用してデータカタログにアクセスできます。

- AWS Glue コンソール – ウェブベースのユーザーインターフェイスである AWS Glue コンソールからデータカタログにアクセスして管理できます。コンソールでは、データベース、テーブル、および関連するメタデータを参照および検索したり、メタデータ定義を作成、更新、削除したりできます。
- AWS Glue クローラー – クローラーは、データソースを自動的にスキャンし、データカタログにメタデータを入力するプログラムです。クローラーを作成して実行し、Amazon S3、Amazon RDS、Amazon DynamoDB、Amazon CloudWatch、および MySQL や PostgreSQL などの JDBC 準拠のリレーショナルデータベース、Snowflake や Google BigQuery などのいくつかの非 AWS ソースからのデータを検出してカタログ化できます。
- AWS Glue API - AWS Glue API を使用してプログラムでデータカタログにアクセスできます。これらの API を使用すると、プログラムでデータカタログを操作し、自動化と他のアプリケーションやサービスとの統合が可能になります。
- AWS Command Line Interface (AWS CLI) – AWS CLI を使用して、コマンドラインからデータカタログにアクセスして管理できます。CLI には、メタデータ定義の作成、更新、削除、メタデータ情報のクエリと取得のためのコマンドが用意されています。
- 他の AWS サービスとの統合 – データカタログは他のさまざまな AWS サービスと統合されるため、カタログに保存されているメタデータにアクセスして利用できます。たとえば、Amazon Athena を使用してデータカタログのメタデータを使用してデータソースをクエリし、AWS Lake Formation を使用してデータカタログリソースのデータアクセスとガバナンスを管理できます。

トピック

- AWS Glue Iceberg REST エンドポイントを使用したデータカタログへの接続方法
- AWS Glue Iceberg REST 拡張エンドポイントを使用したデータカタログへの接続
- AWS Glue Apache Iceberg 用の REST API の仕様
- スタンドアロンの Spark アプリケーションからデータカタログへの接続

- Amazon Redshift と Apache Iceberg 間のデータマッピング
- AWS Glue Iceberg REST Catalog API を使用する際の考慮事項と制限事項

AWS Glue Iceberg REST エンドポイントを使用したデータカタログへの接続方法

AWS Glue の Iceberg REST エンドポイントは、Apache Iceberg REST 仕様で指定された API オペレーションに対応しています。Iceberg REST クライアントを使用すると、分析エンジンで実行されているアプリケーションをデータカタログでホストされている REST カタログに接続することができます。

エンドポイントは、Apache Iceberg テーブル仕様の両方 (v1 と v2) をサポートしており、デフォルトでは v2 に設定されています。Iceberg テーブル v1 仕様を使用する場合は、API コールで v1 を指定する必要があります。API オペレーションを使用すると、Amazon S3 オブジェクトストレージと Amazon S3 テーブルストレージの両方に保存されている Iceberg テーブルにアクセスできます。

エンドポイント設定

サービスエンドポイントを使用して AWS Glue Iceberg REST カタログにアクセスできます。リージョン固有のエンドポイントについては、「AWS Glue サービスエンドポイントのリファレンスガイド」を参照してください。例えば、us-east-1 リージョンの AWS Glue に接続する場合、エンドポイント URI プロパティを次のように設定する必要があります。

```
Endpoint : https://glue.us-east-1.amazonaws.com/iceberg
```

追加の設定プロパティ – Iceberg クライアントを使用して Spark などの分析エンジンをサービスエンドポイントに接続する場合は、次のアプリケーション設定プロパティを指定する必要があります。

```
catalog_name = "mydatacatalog"
aws_account_id = "123456789012"
aws_region = "us-east-1"
spark = SparkSession.builder \
    ... \
    .config("spark.sql.defaultCatalog", catalog_name) \
    .config(f"spark.sql.catalog.{catalog_name}",
 "org.apache.iceberg.spark.SparkCatalog") \
    .config(f"spark.sql.catalog.{catalog_name}.type", "rest") \
    .config(f"spark.sql.catalog.{catalog_name}.uri", "https://glue.
{aws_region}.amazonaws.com/iceberg") \
```

```
    .config(f"spark.sql.catalog.{catalog_name}.warehouse", "{aws_account_id}") \
    .config(f"spark.sql.catalog.{catalog_name}.rest.sigv4-enabled", "true") \
    .config(f"spark.sql.catalog.{catalog_name}.rest.signing-name", "glue") \

.config("spark.sql.extensions","org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensi
\
    .getOrCreate()
```

AWS Glue Iceberg エンドポイント `https://glue.us-east-1.amazonaws.com/iceberg` は、次の Iceberg REST API をサポートします。

- GetConfig
- ListNamespaces
- CreateNamespace
- LoadNamespaceMetadata
- UpdateNamespaceProperties
- DeleteNamespace
- ListTables
- CreateTable
- LoadTable
- TableExists
- UpdateTable
- DeleteTable

プレフィックスとカタログパスパラメータ

Iceberg REST Catalog API のリクエスト URL には、自由形式のプレフィックスが含まれます。例えば、`ListNamespaces` API コールは `GET/v1/{prefix}/namespaces` の URL 形式を使用します。AWS Glue プレフィックスは、REST パスが AWS Glue マルチカタログ階層に確実に一致するように常に `/catalogs/{catalog}` 構造に従います。`{catalog}` パスパラメータは、次のルールに基づいて派生できます。

アクセスパターン	Glue カタログ ID スタイル	プレフィックススタイル	デフォルトのカタログ ID の例	REST ルートの例
現在のアカウントのデフォルトカタログにアクセスする	不要	:	該当なし	GET /v1/catalogs/:/namespaces
特定のアカウントのデフォルトカタログにアクセスする	accountID	accountID	111122223333	GET /v1/catalogs/111122223333/namespaces
現在のアカウントのネストされたカタログにアクセスする	catalog1/catalog2	catalog1/catalog2	rmscatalog1:db1	GET /v1/catalogs/rmscatalog1:db1/namespaces
特定のアカウントのネストされたカタログにアクセスする	accountId:catalog1/catalog2	accountId:catalog1/catalog2	123456789012/rmscatalog1:db1	GET /v1/catalogs/123456789012:rmscatalog1:db1/namespaces

このカタログ ID からプレフィックスへのマッピングは、REST API を直接呼び出す場合にのみ必要です。エンジンを介して AWS Glue Iceberg REST Catalog API を使用する場合、Iceberg REST Catalog API 設定の `warehouse` パラメータまたは AWS Glue 拡張機能 API 設定の `glue.id` パラメータで AWS Glue カタログ ID を指定する必要があります。例えば、EMR Spark での使用方法については、「Use an Iceberg cluster with Spark」を参照してください。

名前空間パスパラメータ

Iceberg REST Catalog API パスの名前空間には、複数のレベルを含めることができます。ただし、AWS Glue は単一レベルの名前空間のみをサポートします。マルチレベルカタログ階層の名前空間にアクセスするには、名前空間の上にあるマルチレベルカタログに接続して名前空間を参照します。これにより、`catalog.namespace.table` の 3 つの部分で構成される表記をサポートするク

エリエンジンが AWS Glue の複数レベルカタログ階層内のオブジェクトにアクセスできます。マルチレベルの名前空間を使用する場合とは異なり、互換性の問題はありません。

AWS Glue Iceberg REST 拡張エンドポイントを使用したデータカタログへの接続

AWS Glue Iceberg REST 拡張エンドポイントは、Apache Iceberg REST 仕様には含まれていない追加の API と、サーバー側のスキャン計画機能を提供します。これらの追加の API は、Amazon Redshift マネージドストレージに保存されているテーブルにアクセスするときに使用します。エンドポイントには、Apache Iceberg AWS Glue Data Catalog 拡張機能を使用して、アプリケーションからアクセスすることができます。

エンドポイント設定 – Redshift マネージドストレージ内のテーブルを含むカタログには、サービスエンドポイントを使用してアクセスできます。リージョン固有のエンドポイントについては、「AWS Glue サービスエンドポイントのリファレンスガイド」を参照してください。例えば、us-east-1 リージョンの AWS Glue に接続する場合、エンドポイント URI プロパティを次のように設定する必要があります。

```
Endpoint : https://glue.us-east-1.amazonaws.com/extensions
```

```
catalog_name = "myredshiftcatalog"
aws_account_id = "123456789012"
aws_region = "us-east-1"
spark = SparkSession.builder \
    .config("spark.sql.defaultCatalog", catalog_name) \
    .config(f"spark.sql.catalog.{catalog_name}",
 "org.apache.iceberg.spark.SparkCatalog") \
    .config(f"spark.sql.catalog.{catalog_name}.type", "glue") \
    .config(f"spark.sql.catalog.{catalog_name}.glue.id",
 "{123456789012}:redshiftnamespacecatalog/redshiftdb") \

 .config("spark.sql.extensions","org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensi
 \
    .getOrCreate()
```

AWS Glue Apache Iceberg 用の REST API の仕様

このセクションでは、AWS Glue Iceberg REST カタログと AWS Glue 拡張 API に関する仕様と、これらの API を使用する際の考慮事項について説明します。

AWS Glue Data Catalog エンドポイントへの API リクエストは、AWS Signature Version 4 (SigV4) を使用して認証されます。AWS SigV4 の詳細については「API リクエストに対する AWS Signature Version 4」のセクションを参照してください。

AWS Glue サービスエンドポイントと AWS Glue メタデータにアクセスする場合、アプリケーションは `glue:getCatalog` IAM アクションを必要とする IAM ロールを引き受けます。

データカタログとそのオブジェクトへのアクセスは、IAM、Lake Formation、または Lake Formation ハイブリッドモードのアクセス許可を使用して管理できます。

データカタログのフェデレーティッドカタログには、Lake Formation に登録されたデータロケーションがあります。Lake Formation は Data Catalog と連動して、データカタログに対するユーザーアクセスを管理するためのデータベーススタイルのアクセス許可を提供します。

IAM、AWS Lake Formation、Lake Formation のいずれかのハイブリッドモードのアクセス許可を使用することで、デフォルトのカタログデータとそのオブジェクトへのアクセスを管理できます。

Lake Formation のマネージド型オブジェクトでデータを作成、挿入、または削除するには、IAM ユーザーまたはロールに特定のアクセス許可を設定する必要があります。

- CREATE_CATALOG – カタログの作成に必要
- CREATE_DATABASE – データベースの作成に必要
- CREATE_TABLE – テーブルの作成に必要
- DELETE – テーブルからのデータの削除に必要
- DESCRIBE – メタデータの読み取りに必要
- DROP – テーブルまたはデータベースのドロップ/削除に必要
- INSERT – プリンシパルがデータをテーブルに挿入する必要がある場合に必要
- SELECT – プリンシパルがテーブルからデータを選択する必要がある場合に必要

詳細については、「AWS Lake Formation デベロッパーガイド」の「Lake Formation permissions reference」を参照してください。

GetConfig

一般情報

オペレーション名	GetConfig

Type	Iceberg REST Catalog API
REST パス	GET /iceberg/v1/config
IAM アクション	glue:GetCatalog
Lake Formation 許可	該当しない
CloudTrail のイベント	glue:GetCatalog
Open API の定義	https://github.com/apache/iceberg/blob/apache-iceberg-1.6.1/open-api/rest-catalog-open-api.yaml#L67

考慮事項と制限事項

- `warehouse` クエリパラメータはカタログ AWS Glue ID に設定する必要があります。設定されていない場合、現在のアカウントのルートカタログを使用してレスポンスが返されます。詳細については、「the section called “プレフィックスとカタログパスパラメータ”」を参照してください。

GetCatalog

一般情報

オペレーション名	GetCatalog
Type	AWS Glue 拡張機能 API
REST パス	GET/extensions/v1/catalogs/{catalog}
IAM アクション	glue:GetCatalog
Lake Formation 許可	DESCRIBE
CloudTrail のイベント	glue:GetCatalog
Open API の定義	https://github.com/awslabs/glue-extensions-for-iceberg/blob/main/glue-extensions-api.yaml#L40

考慮事項と制限事項

- カタログパスパラメータは、「the section called “プレフィックスとカタログパスパラメータ”」セクションで説明されているスタイルに従う必要があります。

ListNamespaces

一般情報

オペレーション名	ListNamespaces
Type	Iceberg REST Catalog API
REST パス	GET/iceberg/v1/catalogs/{catalog}/namespaces
IAM アクション	glue:GetDatabase
Lake Formation 許可	ALL, DESCRIBE, SELECT
CloudTrail のイベント	glue:GetDatabase
Open API の定義	https://github.com/apache/iceberg/blob/apache-iceberg-1.6.1/open-api/rest-catalog-open-api.yaml#L205

考慮事項と制限事項

- カタログパスパラメータは、「the section called “プレフィックスとカタログパスパラメータ”」セクションで説明されているスタイルに従う必要があります。
- 次のレベルの名前空間のみが表示されます。より深いレベルで名前空間を一覧表示するには、ネストされたカタログ ID をカタログパスパラメータで指定します。

CreateNamespace

一般情報

オペレーション名	CreateNamespace
Type	Iceberg REST Catalog API

REST パス	POST/iceberg/v1/catalogs/{catalog}/namespaces
IAM アクション	glue:CreateDatabase
Lake Formation 許可	ALL, DESCRIBE, SELECT
CloudTrail のイベント	glue:CreateDatabase
Open API の定義	https://github.com/apache/iceberg/blob/apache-iceberg-1.6.1/open-api/rest-catalog-open-api.yaml#L256

考慮事項と制限事項

- カタログパスパラメータは、「the section called “プレフィックスとカタログパスパラメータ”」セクションで説明されているスタイルに従う必要があります。
- 作成できるのは単一レベルの名前空間のみです。マルチレベルの名前空間を作成するには、各レベルを繰り返し作成して、カタログパスパラメータを使用してレベルに接続する必要があります。

StartCreateNamespaceTransaction

一般情報

オペレーション名	StartCreateNamespaceTransaction
Type	AWS Glue 拡張機能 API
REST パス	POST/extensions/v1/catalogs/{catalog}/namespaces
IAM アクション	glue:CreateDatabase
Lake Formation 許可	ALL, DESCRIBE, SELECT
CloudTrail のイベント	glue:CreateDatabase
Open API の定義	https://github.com/apache/iceberg/blob/apache-iceberg-1.6.1/open-api/rest-catalog-open-api.yaml#L256

考慮事項と制限事項

- カタログパスパラメータは、「the section called “プレフィックスとカタログパスパラメータ”」セクションで説明されているスタイルに従う必要があります。
- 作成できるのは単一レベルの名前空間のみです。マルチレベル名前空間を作成するには、各レベルを繰り返し作成し、カタログパスパラメータを使用してレベルに接続する必要があります。
- API は非同期で、`CheckTransactionStatus` API コールを使用して追跡するために使用できるトランザクション ID を返します。
- この API を呼び出すことができるのは、`GetCatalog` API コールにレスポンスの `use-extensions=true` パラメータが含まれている場合のみです。

LoadNamespaceMetadata

一般情報

オペレーション名	LoadNamespaceMetadata
Type	Iceberg REST Catalog API
REST パス	GET/iceberg/v1/catalogs/{catalog}/namespaces/{ns}
IAM アクション	glue:GetDatabase
Lake Formation 許可	ALL, DESCRIBE, SELECT
CloudTrail のイベント	glue:GetDatabase
Open API の定義	https://github.com/apache/iceberg/blob/apache-iceberg-1.6.1/open-api/rest-catalog-open-api.yaml#L302

考慮事項と制限事項

- カタログパスパラメータは、「the section called “プレフィックスとカタログパスパラメータ”」セクションで説明されているスタイルに従う必要があります。
- REST Path パラメータでは、単一レベルの名前空間のみを指定できます。詳細については「the section called “名前空間パスパラメータ”」セクションを参照してください。

UpdateNamespaceProperties

一般情報

オペレーション名	UpdateNamespaceProperties
Type	Iceberg REST Catalog API
REST パス	POST /iceberg/v1/catalogs/{catalog}/namespaces/{ns}/properties
IAM アクション	glue:UpdateDatabase
Lake Formation 許可	ALL, ALTER
CloudTrail のイベント	glue:UpdateDatabase
Open API の定義	https://github.com/apache/iceberg/blob/apache-iceberg-1.6.1/open-api/rest-catalog-open-api.yaml#L400

考慮事項と制限事項

- カタログパスパラメータは、「the section called “プレフィックスとカタログパスパラメータ”」セクションで説明されているスタイルに従う必要があります。
- REST Path パラメータでは、単一レベルの名前空間のみを指定できます。詳細については「the section called “名前空間パスパラメータ”」セクションを参照してください。

DeleteNamespace

一般情報

オペレーション名	DeleteNamespace
Type	Iceberg REST Catalog API
REST パス	DELETE/iceberg/v1/catalogs/{catalog}/namespces/{ns}
IAM アクション	glue:DeleteDatabase
Lake Formation 許可	ALL, DROP

CloudTrail のイベント	glue:DeleteDatabase
Open API の定義	https://github.com/apache/iceberg/blob/apache-iceberg-1.6.1/open-api/rest-catalog-open-api.yaml#L365

考慮事項と制限事項

- カタログパスパラメータは、「the section called “プレフィックスとカタログパスパラメータ”」セクションで説明されているスタイルに従う必要があります。
- REST Path パラメータでは、単一レベルの名前空間のみを指定できます。詳細については「the section called “名前空間パスパラメータ”」セクションを参照してください。
- データベースにオブジェクトがある場合、オペレーションは失敗します。
- API は非同期で、`CheckTransactionStatus` API コールを使用して追跡するために使用できるトランザクション ID を返します。
- API は、`GetCatalog` API コールがレスポンスで `use-extensions=true` を示す場合にのみ使用できます。

StartDeleteNamespaceTransaction

一般情報

オペレーション名	StartDeleteNamespaceTransaction
Type	AWS Glue 拡張機能 API
REST パス	DELETE /extensions/v1/catalogs/{catalog}/namespces/{ns}
IAM アクション	glue:DeleteDatabase
Lake Formation 許可	ALL, DROP
CloudTrail のイベント	glue:DeleteDatabase
Open API の定義	https://github.com/awslabs/glue-extensions-for-iceberg/blob/main/glue-extensions-api.yaml#L85

考慮事項と制限事項

- カタログパスパラメータは、「the section called “プレフィックスとカタログパスパラメータ”」セクションで説明されているスタイルに従う必要があります。
- REST Path パラメータでは、単一レベルの名前空間のみを指定できます。詳細については「the section called “名前空間パスパラメータ”」セクションを参照してください。
- データベースにオブジェクトがある場合、オペレーションは失敗します。
- API は非同期で、`CheckTransactionStatus` API コールを使用して追跡するために使用できるトランザクション ID を返します。
- API は、`GetCatalog` API コールがレスポンスで `use-extensions=true` を示す場合にのみ使用できます。

ListTables

一般情報

オペレーション名	ListTables
Type	Iceberg REST Catalog API
REST パス	GET /iceberg/v1/catalogs/{catalog}/namespaces/{ns}/tables
IAM アクション	glue:GetTables
Lake Formation 許可	ALL, SELECT, DESCRIBE
CloudTrail のイベント	glue:GetTables
Open API の定義	https://github.com/apache/iceberg/blob/apache-iceberg-1.6.1/open-api/rest-catalog-open-api.yaml#L463

考慮事項と制限事項

- カタログパスパラメータは、「the section called “プレフィックスとカタログパスパラメータ”」セクションで説明されているスタイルに従う必要があります。
- REST Path パラメータでは、単一レベルの名前空間のみを指定できます。詳細については「the section called “名前空間パスパラメータ”」セクションを参照してください。

- Iceberg 以外のテーブルを含むすべてのテーブルが一覧表示されます。テーブルを Iceberg テーブルとしてロードできるかどうかを確認するには、 `LoadTable` オペレーションを呼び出します。

CreateTable

一般情報

オペレーション名	CreateTable
Type	Iceberg REST Catalog API
REST パス	GET /iceberg/v1/catalogs/{catalog}/namespaces/{ns}/tables
IAM アクション	glue:CreateTable
Lake Formation 許可	ALL, CREATE_TABLE
CloudTrail のイベント	glue:CreateTable
Open API の定義	https://github.com/apache/iceberg/blob/apache-iceberg-1.6.1/open-api/rest-catalog-open-api.yaml#L497

考慮事項と制限事項

- カタログパスパラメータは、「the section called “プレフィックスとカタログパスパラメータ”」セクションで説明されているスタイルに従う必要があります。
- REST Path パラメータでは、単一レベルの名前空間のみを指定できます。詳細については「the section called “名前空間パスパラメータ”」セクションを参照してください。
- ステージングを使用した `CreateTable` はサポートされていません。`stageCreate` クエリパラメータを指定するとオペレーションは失敗します。`CREATE TABLE AS SELECT` のようなオペレーションはサポートされていないため、回避策として `CREATE TABLE` と `INSERT INTO` の組み合わせを使用できます。
- `CreateTable` API オペレーションはオプション `state-create = TRUE` をサポートしていません。

StartCreateTableTransaction

一般情報

オペレーション名	CreateTable
Type	AWS Glue 拡張機能 API
REST パス	POST/extensions/v1/catalogs/{catalog}/namespaces/{ns}/tables
IAM アクション	glue:CreateTable
Lake Formation 許可	ALL, CREATE_TABLE
CloudTrail のイベント	glue:CreateTable
Open API の定義	https://github.com/awslabs/glue-extensions-for-iceberg/blob/main/glue-extensions-api.yaml#L107

考慮事項と制限事項

- カタログパスパラメータは、「the section called “プレフィックスとカタログパスパラメータ”」セクションで説明されているスタイルに従う必要があります。
- REST Path パラメータでは、単一レベルの名前空間のみを指定できます。詳細については「the section called “名前空間パスパラメータ”」セクションを参照してください。
- ステージングを使用した `CreateTable` はサポートされていません。`stageCreate` クエリパラメータを指定するとオペレーションは失敗します。`CREATE TABLE AS SELECT` のようなオペレーションはサポートされていないため、ユーザーは問題を回避するために `CREATE TABLE` と `INSERT INTO` を組み合わせて使用する必要があります。
- API は非同期で、`CheckTransactionStatus` API コールを使用して追跡するために使用できるトランザクション ID を返します。
- API は、`GetCatalog` API コールがレスポンスで `use-extensions=true` を示す場合にのみ使用できます。

LoadTable

一般情報

オペレーション名	LoadTable
タイプ	Iceberg REST Catalog API
REST パス	GET /iceberg/v1/catalogs/{catalog}/namespaces/{ns}/tables/{table}
IAM アクション	glue:GeTTable
Lake Formation 許可	ALL, SELECT, DESCRIBE
CloudTrail のイベント	glue:GetTable
Open API の定義	https://github.com/apache/iceberg/blob/apache-iceberg-1.6.1/open-api/rest-catalog-open-api.yaml#L616

考慮事項

- カタログパスパラメータは、「the section called “プレフィックスとカタログパスパラメータ”」セクションで説明されているスタイルに従う必要があります。
- REST Path パラメータでは、単一レベルの名前空間のみを指定できます。詳細については「the section called “名前空間パスパラメータ”」セクションを参照してください。
- ステージングを使用した `CreateTable` はサポートされていません。`stageCreate` クエリパラメータを指定するとオペレーションは失敗します。`CREATE TABLE AS SELECT` のようなオペレーションはサポートされていないため、ユーザーは問題を回避するために `CREATE TABLE` と `INSERT INTO` を組み合わせて使用する必要があります。
- API は非同期で、`CheckTransactionStatus` API コールを使用して追跡するために使用できるトランザクション ID を返します。
- API は、`GetCatalog` API コールがレスポンスで `use-extensions=true` を示す場合にのみ使用できます。

ExtendedLoadTable

一般情報

オペレーション名	LoadTable
タイプ	AWS Glue 拡張機能 API
REST パス	GET /extensions/v1/catalogs/{catalog}/namespaces/{ns}/tables/{table}
IAM アクション	glue:GetTable
Lake Formation 許可	ALL, SELECT, DESCRIBE
CloudTrail のイベント	glue:GetTable
Open API の定義	https://github.com/awslabs/glue-extensions-for-iceberg/blob/main/glue-extensions-api.yaml#L134

考慮事項

- カタログパスパラメータは、「the section called “プレフィックスとカタログパスパラメータ”」セクションで説明されているスタイルに従う必要があります。
- REST Path パラメータでは、単一レベルの名前空間のみを指定できます。詳細については「the section called “名前空間パスパラメータ”」セクションを参照してください。
- スナップショットクエリパラメータでは `all` モードのみがサポートされます。
- `LoadTable` API と比較すると、`ExtendedLoadTable` API は次の点が異なります。
  - すべてのフィールドが利用可能になるよう厳密に強制されません。
  - レスポンスの設定フィールドに以下の追加パラメータを提供します。

    追加パラメータ

設定キー	説明
aws.server-side-capabilities.scan-planning	PreparePlan テーブルと PlanTable API を使用してテーブルをスキャンできるかどうかを示します。

設定キー	説明
aws.server-side-capabilities.data-commit	StartUpdateTable トランザクションを使用してテーブルをコミットできるかどうかを示します。
aws.glue.staging.location	サーバー側のスキャン計画またはデータコミットに使用されます。これは、エンジンが一時データファイルを書き込むために使用できるサービスマネージド型のステージング場所です。
aws.glue.staging.access-key-id	サーバー側のスキャン計画またはデータコミットに使用されます。これは、サービスマネージド型のステージング場所にアクセスするための一時的な AWS 認証情報の一部です。
aws.glue.staging.secret-access-key	サーバー側のスキャン計画またはデータコミットに使用されます。これは、サービスマネージド型のステージング場所にアクセスするための一時的な AWS 認証情報の一部です。
aws.glue.staging.session-token	サーバー側のスキャン計画またはデータコミットに使用されます。これは、サービスマネージド型のステージング場所にアクセスするための一時的な AWS 認証情報の一部です。
aws.glue.staging.expiration-ms	サーバー側のスキャン計画またはデータコミットに使用されます。これは、サービスマネージド型のステージング場所にアクセスするための認証情報の有効期限です。
aws.glue.staging.data-transfer-role-arn	サーバー側のスキャン計画またはデータコミットに使用されます。これは、サービスマネージドステージング場所にアクセスするために引き受けることができる IAM ロールです。

PreplanTable

一般情報

オペレーション名	PreplanTable
タイプ	AWS Glue 拡張機能 API
REST パス	POST /extensions/v1/catalogs/{catalog}/namespaces/{ns}/tables/{table}/preplan
IAM アクション	glue:GetTable
Lake Formation 許可	ALL, SELECT, DESCRIBE
CloudTrail のイベント	glue:GetTable
Open API の定義	https://github.com/awslabs/glue-extensions-for-iceberg/blob/main/glue-extensions-api.yaml#L211

考慮事項

- カタログパスパラメータは、「the section called “プレフィックスとカタログパスパラメータ”」セクションで説明されているスタイルに従う必要があります。
- REST Path パラメータでは、単一レベルの名前空間のみを指定できます。詳細については「the section called “名前空間パスパラメータ”」セクションを参照してください。
- この API の呼び出し元は、ページトークンに基づいて、取得する残りの結果があるかどうかを常に判断する必要があります。サーバー側がまだ処理中で指定された応答時間内に結果を生成できない場合、ページ分割トークンを含む空白のページが返されることがあります。
- この API は、API `ExtendedLoadTable` レスポンスに `aws.server-side-capabilities.scan-planning=true` が含まれている場合にのみ使用できます。

PlanTable

一般情報

オペレーション名	PlanTable
タイプ	AWS Glue 拡張機能 API

REST パス	POST /extensions/v1/catalogs/{catalog}/namespaces/{ns}/tables/{table}/plan
IAM アクション	glue:GetTable
Lake Formation 許可	ALL, SELECT, DESCRIBE
CloudTrail のイベント	glue:GetTable
Open API の定義	https://github.com/awslabs/glue-extensions-for-iceberg/blob/main/glue-extensions-api.yaml#L243

考慮事項

- カタログパスパラメータは、「the section called “プレフィックスとカタログパスパラメータ”」セクションで説明されているスタイルに従う必要があります。
- REST Path パラメータでは、単一レベルの名前空間のみを指定できます。詳細については「the section called “名前空間パスパラメータ”」セクションを参照してください。
- この API の呼び出し元は、ページトークンに基づいて、取得する残りの結果があるかどうかを常に判断する必要があります。サーバー側がまだ処理中で指定された応答時間内に結果を生成できない場合、ページ分割トークンを含む空白のページが返されることがあります。
- この API は、API `ExtendedLoadTable` レスポンスに `aws.server-side-capabilities.scan-planning=true` が含まれている場合にのみ使用できます。

TableExists

一般情報

オペレーション名	TableExists
タイプ	Iceberg REST Catalog API
REST パス	HEAD/iceberg/v1/catalogs/{catalog}/namespaces/{ns}/tables/{table}
IAM アクション	glue:GetTable
Lake Formation 許可	ALL, SELECT, DESCRIBE

CloudTrail のイベント	glue:GetTable
Open API の定義	https://github.com/apache/iceberg/blob/apache-iceberg-1.6.1/open-api/rest-catalog-open-api.yaml#L833

考慮事項

- カタログパスパラメータは、「the section called “プレフィックスとカタログパスパラメータ”」セクションで説明されているスタイルに従う必要があります。
- REST Path パラメータでは、単一レベルの名前空間のみを指定できます。詳細については「the section called “名前空間パスパラメータ”」セクションを参照してください。

UpdateTable

一般情報

オペレーション名	UpdateTable
タイプ	Iceberg REST Catalog API
REST パス	POST /iceberg/v1/catalogs/{catalog}/namespaces/{ns}/tables/{table}
IAM アクション	glue:UpdateTable
Lake Formation 許可	ALL, ALTER
CloudTrail のイベント	glue:UpdateTable
Open API の定義	https://github.com/apache/iceberg/blob/apache-iceberg-1.6.1/open-api/rest-catalog-open-api.yaml#L677

考慮事項

- カタログパスパラメータは、「the section called “プレフィックスとカタログパスパラメータ”」セクションで説明されているスタイルに従う必要があります。
- REST Path パラメータでは、単一レベルの名前空間のみを指定できます。詳細については「the section called “名前空間パスパラメータ”」セクションを参照してください。

StartUpdateTableTransaction

一般情報

オペレーション名	StartUpdateTableTransaction
タイプ	AWS Glue 拡張機能 API
REST パス	POST/extensions/v1/catalogs/{catalog}/namespaces/{ns}/tables/{table}
IAM アクション	glue:UpdateTable
Lake Formation 許可	ALL, ALTER
CloudTrail のイベント	glue:UpdateTable
Open API の定義	https://github.com/awslabs/glue-extensions-for-iceberg/blob/main/glue-extensions-api.yaml#L154

考慮事項

- カタログパスパラメータは、「the section called “プレフィックスとカタログパスパラメータ”」セクションで説明されているスタイルに従う必要があります。
- REST Path パラメータでは、単一レベルの名前空間のみを指定できます。詳細については「the section called “名前空間パスパラメータ”」セクションを参照してください。
- API は非同期で、`CheckTransactionStatus` API コールを使用して追跡するために使用できるトランザクション ID を返します。
- この API を使用して `RenamTable` オペレーションを実行することもできます。この場合、呼び出し元は、名前を変更するテーブルの glue:CreateTable または LakeFormation CREATE_TABLE アクセス許可も取得する必要があります。
- この API は、API `ExtendedLoadTable` レスポンスに `aws.server-side-capabilities.scan-planning=true` が含まれている場合にのみ使用できます。

DeleteTable

一般情報

オペレーション名	DeleteTable
タイプ	Iceberg REST Catalog API
REST パス	DELETE/iceberg/v1/catalogs/{catalog}/namespaces/{ns}/tables/{table}
IAM アクション	glue:DeleteTable
Lake Formation 許可	ALL, DROP
CloudTrail のイベント	glue:DeleteTable
Open API の定義	https://github.com/apache/iceberg/blob/apache-iceberg-1.6.1/open-api/rest-catalog-open-api.yaml#L793

考慮事項

- カタログパスパラメータは、「the section called “プレフィックスとカタログパスパラメータ”」セクションで説明されているスタイルに従う必要があります。
- REST Path パラメータでは、単一レベルの名前空間のみを指定できます。詳細については「the section called “名前空間パスパラメータ”」セクションを参照してください。
- `DeleteTable` API オペレーションはパージオプションをサポートします。パージが `true` に設定されている場合、テーブルデータは削除されます。設定されていない場合は削除されません。Amazon S3 のテーブルの場合、このオペレーションでテーブルデータが削除されることはありません。テーブルが Amazon S3 と `purge = TRUE`, に保存されている場合、このオペレーションは失敗します。

  テーブルが Amazon Redshift マネージドストレージに保存されている場合、Amazon Redshift での `DROP TABLE` の動作と同様に、このオペレーションによりテーブルデータは削除されます。テーブルが Amazon Redshift と `purge = FALSE` に保存されている場合、このオペレーションは失敗します。
- `purgeRequest=true` はサポートされていません。

StartDeleteTableTransaction

一般情報

オペレーション名	StartDeleteTableTransaction
タイプ	AWS Glue 拡張機能 API
REST パス	DELETE /extensions/v1/catalogs/{catalog}/namespaces/{ns}/tables/{table}
IAM アクション	glue:DeleteTable
Lake Formation 許可	ALL, DROP
CloudTrail のイベント	glue:DeleteTable
Open API の定義	https://github.com/apache/iceberg/blob/apache-iceberg-1.6.1/open-api/rest-catalog-open-api.yaml#L793

考慮事項

- カタログパスパラメータは、「the section called “プレフィックスとカタログパスパラメータ”」セクションで説明されているスタイルに従う必要があります。
- REST Path パラメータでは、単一レベルの名前空間のみを指定できます。詳細については「the section called “名前空間パスパラメータ”」セクションを参照してください。
- `purgeRequest=false` はサポートされていません。
- API は非同期で、`CheckTransactionStatus` を介して追跡できるトランザクション ID を返します。

CheckTransactionStatus

一般情報

オペレーション名	CheckTransactionStatus
タイプ	AWS Glue 拡張機能 API
REST パス	POST/extensions/v1/transactions/status

IAM アクション	トランザクションを開始するアクションと同じアクセス許可
Lake Formation 許可	トランザクションを開始するアクションと同じアクセス許可
Open API の定義	https://github.com/awslabs/glue-extensions-for-iceberg/blob/main/glue-extensions-api.yaml#L273

考慮事項

- カタログパスパラメータは、「the section called “プレフィックスとカタログパスパラメータ”」セクションで説明されているスタイルに従う必要があります。

スタンドアロンの Spark アプリケーションからデータカタログへの接続

Apache Iceberg コネクタを使用することで、スタンドアロンのアプリケーションからデータカタログに接続することができます。

1. Spark アプリケーションの IAM ロールを作成します。
2. Iceberg コネクタを使用して AWS Glue Iceberg Rest エンドポイントに接続します。

```
# configure your application. Refer to https://docs.aws.amazon.com/cli/latest/
userguide/cli-configure-envvars.html for best practices on configuring environment
 variables.
export AWS_ACCESS_KEY_ID=$(aws configure get appUser.aws_access_key_id)
export AWS_SECRET_ACCESS_KEY=$(aws configure get appUser.aws_secret_access_key)
export AWS_SESSION_TOKEN=$(aws configure get appUser.aws_secret_token)

export AWS_REGION=us-east-1
export REGION=us-east-1
export AWS_ACCOUNT_ID = {specify your aws account id here}

~/spark-3.5.3-bin-hadoop3/bin/spark-shell \
    --packages org.apache.iceberg:iceberg-spark-runtime-3.4_2.12:1.6.0 \
    --conf
 "spark.sql.extensions=org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions"
 \
    --conf "spark.sql.defaultCatalog=spark_catalog" \
```

```
    --conf "spark.sql.catalog.spark_catalog=org.apache.iceberg.spark.SparkCatalog" \
    --conf "spark.sql.catalog.spark_catalog.type=rest" \
    --conf "spark.sql.catalog.spark_catalog.uri=https://glue.us-east-1.amazonaws.com/
iceberg" \
    --conf "spark.sql.catalog.spark_catalog.warehouse = {AWS_ACCOUNT_ID}" \
    --conf "spark.sql.catalog.spark_catalog.rest.sigv4-enabled=true" \
    --conf "spark.sql.catalog.spark_catalog.rest.signing-name=glue" \
    --conf "spark.sql.catalog.spark_catalog.rest.signing-region=us-east-1" \
    --conf "spark.sql.catalog.spark_catalog.io-
impl=org.apache.iceberg.aws.s3.S3FileIO" \
    --conf
 "spark.hadoop.fs.s3a.aws.credentials.provider=org.apache.hadoop.fs.s3a.SimpleAWSCredentialPr
```

3. データカタログのデータをクエリします。

```
spark.sql("create database myicebergdb").show()
spark.sql("""CREATE TABLE myicebergdb.mytbl (name string) USING iceberg location
 's3://bucket_name/mytbl'""")
spark.sql("insert into myicebergdb.mytbl values('demo') ").show()
```

Amazon Redshift と Apache Iceberg 間のデータマッピング

Redshift と Iceberg は、さまざまなデータ型をサポートしています。次の互換性マトリクスは、この 2 つのデータシステム間でデータをマッピングする際のサポートと制限の概要を示しています。各データシステムでサポートされているデータ型の詳細については、「Amazon Redshift データ型」と「Apache Iceberg Table Specifications」を参照してください。

Redshift データ型	エイリアス	Iceberg データ型
SMALLINT	INT2	int
INTEGER	INT、INT4	int
BIGINT	INT8	long
DECIMAL	NUMERIC	decimal
REAL	FLOAT4	フロート

Redshift データ型	エイリアス	Iceberg データ型
REAL	FLOAT4	フロート
DOUBLE PRECISION	FLOAT8、FLOAT	double
CHAR	CHARACTER, NCHAR	文字列
VARCHAR	CHARACTER VARYING, NVARCHAR	文字列
BPCHAR		文字列
TEXT		文字列
DATE		date
TIME	TIME WITHOUT TIMEZONE	時間
TIME	TIME WITH TIMEZONE	サポート外
TIMESTAMP	TIMESTAMP WITHOUT TIMEZONE	TIMESTAMP
TIMESTAMPZ	TIMESTAMP WITH TIMEZONE	TIMESTAMPZ
INTERVAL YEAR TO MONTH		サポートされていません
INTERVAL DAY TO SECOND		サポートされていません
BOOLEAN	BOOL	ブール
HLLSKETCH		サポートされていません
SUPER		サポートされていません
VARBYTE	VARBINARY、BINARY VARYING	バイナリ
GEOMETRY		サポートされていません

Redshift データ型	エイリアス	Iceberg データ型
GEOGRAPHY		サポートされていません

AWS Glue Iceberg REST Catalog API を使用する際の考慮事項と制限事項

Apache Iceberg REST Catalog データ定義言語 (DDL) オペレーション動作を使用する際の考慮事項と制限事項は次のとおりです。

考慮事項

- **`RenameTable`** API の動作 – `RenameTable` オペレーションは Amazon Redshift のテーブルではサポートされていますが、Amazon S3 のテーブルではサポートされていません。
- Amazon Redshift の名前空間とテーブルの DDL オペレーション – Amazon Redshift の名前空間とテーブルの作成、更新、削除のオペレーションは、Amazon Redshift のマネージドワークグループがいつ利用可能か、および競合する DDL と DML トランザクションが進行中かどうかに依存し、オペレーションはロックを待機してから変更のコミットを試みる必要があるため、非同期オペレーションとなります。

制限

- Apache Iceberg REST 仕様の View API は、AWS Glue Iceberg REST Catalog ではサポートされていません。

AWS Glue データカタログのベストプラクティス

このセクションでは、AWS Glue Data Catalog を効果的に管理および活用するためのベストプラクティスについて説明します。効率的なクローラーの使用、メタデータの整理、セキュリティ、パフォーマンスの最適化、自動化、データガバナンス、他の AWS サービスとの統合などのプラクティスについて重点的に説明します。

- クローラーを効果的に使用する – クローラーを定期的に実行して、データソースの変更に応じてデータカタログを最新の状態に保ちます。頻繁に変化するデータソースには増分クロールを使用してパフォーマンスを向上させます。変更が検出されたときに新しいパーティションを自動的に追加したり、スキーマを更新したりするようにクローラーを設定します。

- メタデータテーブルの整理と名前付け — データカタログのデータベースとテーブルについて一貫した命名規則を確立します。関連データソースを論理データベースまたはフォルダにグループ化して、より適切に整理します。各テーブルの目的と内容を伝えるわかりやすい名前を使用します。
- スキーマを効果的に管理する — AWS Glue クローラーのスキーマ推論機能を活用します。ダウンストリームアプリケーションが破損しないように、スキーマの変更を適用する前に確認および更新します。スキーマ進化機能を使用して、スキーマの変更を適切に処理します。
- データカタログの保護 — データカタログの保管中および転送中のデータ暗号化を有効にします。きめ細かなアクセスコントロールポリシーを実装し、機密データへのアクセスを制限します。データカタログのアクセス許可とアクティビティログを定期的に監査して確認します。
- 他の AWS サービスとの統合 - データカタログを Amazon Athena、Redshift Spectrum、AWS Lake Formation などのサービスを一元管理できるメタデータレイヤーとして使用します。AWS Glue ETL ジョブを活用して、データカタログのメタデータを維持しながら、データを変換してさまざまなデータストアにロードします。
- パフォーマンスのモニタリングと最適化 - Amazon CloudWatch メトリクスを使用してクローラーと ETL ジョブのパフォーマンスをモニタリングします。大きなデータセットをデータカタログにパーティション分割して、クエリのパフォーマンスを向上します。頻繁にアクセスされるメタデータのパフォーマンス最適化を実装します。
- AWS Glue ドキュメントとベストプラクティスの最新情報を入手 - AWS Glue ドキュメントと AWS Glue リソースで最新の更新、ベストプラクティス、推奨事項を定期的に確認します。AWS Glue ウェビナー、ワークショップ、その他のイベントに参加して、エキスパートから学び、新機能について最新情報を得ます。

Amazon CloudWatch での Data Catalog 使用状況メトリクスのモニタリング

AWS Glue Data Catalog の使用状況メトリクスが Amazon CloudWatch で利用可能になり、Data Catalog でのリソース使用率のモニタリングと理解が簡素化されました。カタログ、データベース、テーブル、パーティション、接続の Glue Catalog API 使用状況がすぐに可視化され、Data Catalog を容易に監視できるようになりました。

Data Catalog メトリクスの概要

AWS Glue Data Catalog は、使用状況メトリクスを Amazon CloudWatch に自動的に発行します。CloudWatch メトリクスの統合を使用すると、次のような重要なパフォーマンス指標を 1 分ごとに追跡できます。

- テーブルリクエスト
- 作成されたパーティションインデックス
- 更新された接続
- 更新された統計

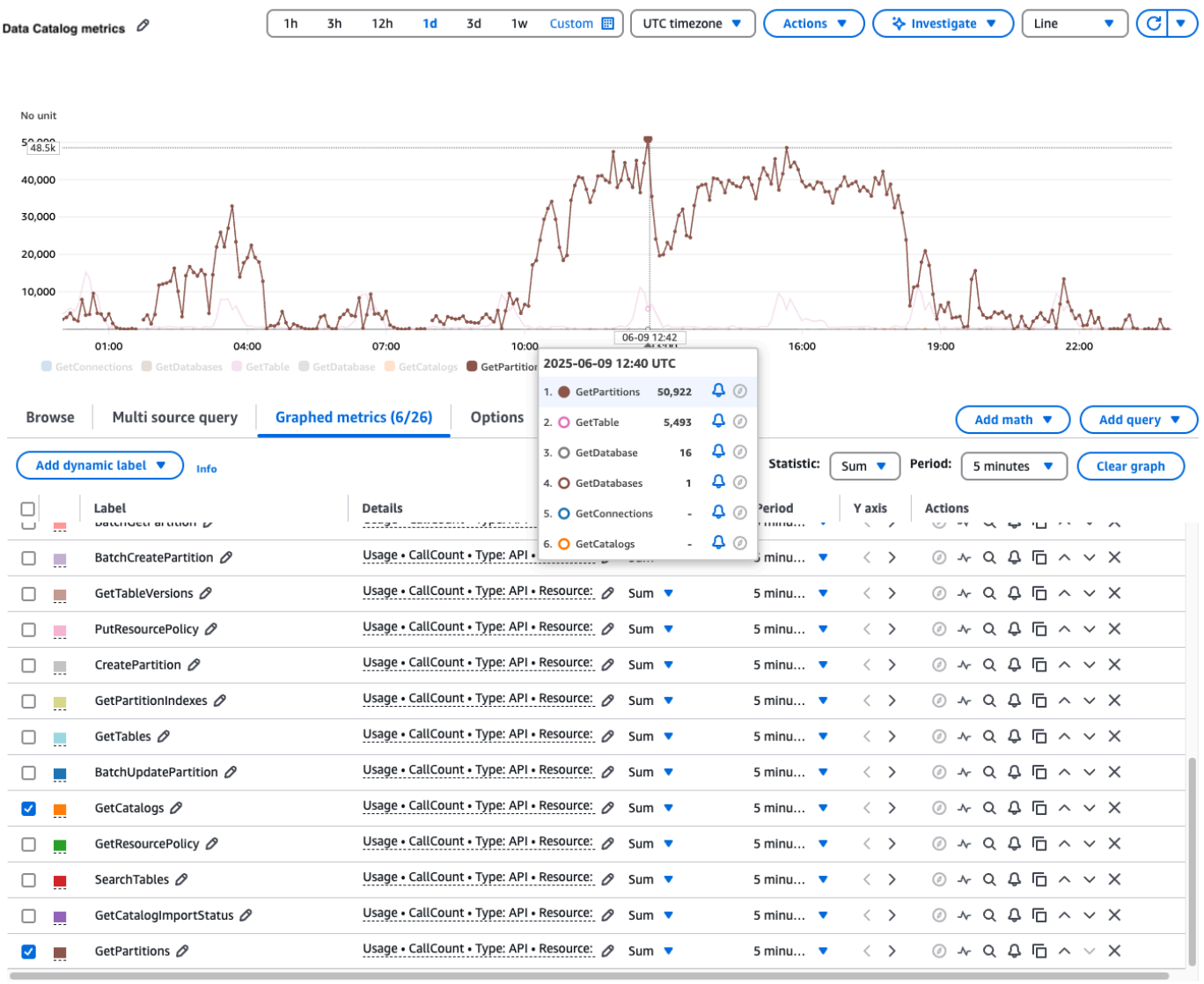
これらのメトリクスを使用して、ボトルネックの特定、異常の検出、データ駆動型の意思決定を行い、データカタログ全体の信頼性を向上させることができます。メトリクスが指定されたしきい値を超えたときに通知を受信するように CloudWatch アラームを設定することもでき、デプロイをプロアクティブに管理できます。

CloudWatch ダッシュボードにメトリクスを追加する

カスタムダッシュボードを作成して AWS Glue Data Catalog リソースをモニタリングし、異常なアクティビティを通知するアラームを設定できます。

以下の手順に従って、CloudWatch ダッシュボードに Data Catalog メトリクスを追加できます。

1. CloudWatch コンソールの https://console.aws.amazon.com/cloudwatch/ を開いてください。
2. ナビゲーションペインで [メトリクス] を選択します。
3. [すべてのメトリクス] を選択します。
4. [使用状況] > AWS[リソース] を選択します。
5. Glue でフィルタリングして、使用可能なメトリクスを表示します。
6. ダッシュボードに追加したいメトリクスを選択します。
7. CloudWatch グラフにカタログ、データベース、テーブル、パーティション、接続のメトリクスを追加します。



API 使用量が定義されたしきい値を超えたときに自動的にトリガーされるカスタムアラームを設定して、データカタログ使用状況の異常を特定できます。

アラームの設定に関する詳細については、「Metrics Insights CloudWatch アラームの作成」を参照してください。

AWS Glue スキーマレジストリ

> **Note**
>
> AWS Glue スキーマレジストリは、アジアパシフィック (ジャカルタ)、欧州 (チューリッヒ)、中東 (UAE) リージョンの AWS Glue コンソールではサポートされていません。。

AWS Glue スキーマレジストリを使用すると、データストリームスキーマを一元的に検出、制御、展開できます。スキーマは、データレコードの構造と形式を定義します。AWS Glue のスキーマレジストリを使用すると、Apache Kafka、Amazon Managed Streaming for Apache Kafka、Amazon Kinesis Data Streams、Amazon Managed Service for Apache Flink、AWS Lambda と便利な統合を使用するデータストリームアプリケーションでスキーマを管理して実施することができます。

スキーマレジストリは、AVRO (v1.11.4) データ形式、Everit ライブラリを使用した JSON スキーマ検証を含むスキーマ用 (Draft-04、Draft-06、Draft-07 仕様) の JSON スキーマ形式の JSON データ形式、`extensions` または `groups` のサポートが含まれないプロトコルバッファ (Protobuf) バージョンの proto2 および proto3、Java 言語対応をサポートしており、今後は他のデータ形式や言語にも対応していきます。サポートされている機能には、互換性、メタデータを介したスキーマのソーシング、スキーマの自動登録、IAM 互換性、ストレージとデータ転送を削減する ZLIB 圧縮のオプションがあります。スキーマレジストリはサーバーレスで無料で使用できます。

プロデューサとコンシューマの間のデータ形式契約としてスキーマを使用すると、データガバナンスおよびデータ品質の向上につながります。さらにデータのコンシューマは、上流で行われる互換性に関する変更に対する復旧力を得ることができます。

スキーマレジストリを使用すると、異なるシステムでシリアル化と非シリアル化用のスキーマを共有できます。例えば、データのプロデューサーとコンシューマーを考えてみます。データの公開時、プロデューサーはスキーマを把握しています。Schema Registry は、Amazon MSK や Apache Kafka などの特定のシステム用に、シリアル化と非シリアル化の機能を提供します。

詳しくは、「スキーマレジストリの仕組み 」を参照してください。

トピック

- スキーマ
- レジストリ
- スキーマのバージョニングと互換性
- オープンソース Serde ライブラリ

- Schema Registry のクォータ
- スキーマレジストリの仕組み
- スキーマレジストリの開始方法
- AWS Glue Schema Registry との統合
- サードパーティー製のスキーマレジストリから AWS Glue Schema Registry への移行

スキーマ

スキーマは、データレコードの構造と形式を定義します。スキーマは、信頼性の高いデータの公開、利用、または保存のための仕様をバージョニングしたものです。

この例の Avro のスキーマでは、形式と構造はレイアウトとフィールド名によって定義され、フィールド名の形式はデータ型 (例:`string`、`int`) により定義されています

```
{
    "type": "record",
    "namespace": "ABC_Organization",
    "name": "Employee",
    "fields": [
        {
            "name": "Name",
            "type": "string"
        },
        {
            "name": "Age",
            "type": "int"
        },
        {
            "name": "address",
            "type": {
                "type": "record",
                "name": "addressRecord",
                "fields": [
                    {
                        "name": "street",
                        "type": "string"
                    },
                    {
                        "name": "zipcode",
                        "type": "int"
                    }
```

```
                ]
            }
        }
    ]
}
```

この例の JSON 用の JSON スキーマ draft-07 では、形式を JSON スキーマ組織が定義しています。

```
{
 "$id": "https://example.com/person.schema.json",
 "$schema": "http://json-schema.org/draft-07/schema#",
 "title": "Person",
 "type": "object",
 "properties": {
  "firstName": {
   "type": "string",
   "description": "The person's first name."
  },
  "lastName": {
   "type": "string",
   "description": "The person's last name."
  },
  "age": {
   "description": "Age in years which must be equal to or greater than zero.",
   "type": "integer",
   "minimum": 0
  }
 }
}
```

この例の Protobuf では、データ形式を Protocol Buffers l言語のバージョン 2 (proto2) で定義しています。

```
syntax = "proto2";

package tutorial;

option java_multiple_files = true;
option java_package = "com.example.tutorial.protos";
option java_outer_classname = "AddressBookProtos";

message Person {
  optional string name = 1;
```

```
  optional int32 id = 2;
  optional string email = 3;

  enum PhoneType {
    MOBILE = 0;
    HOME = 1;
    WORK = 2;
  }

  message PhoneNumber {
    optional string number = 1;
    optional PhoneType type = 2 [default = HOME];
  }

  repeated PhoneNumber phones = 4;
}

message AddressBook {
  repeated Person people = 1;
}
```

レジストリ

レジストリはスキーマの論理コンテナです。レジストリを使用すると、スキーマを整理したり、アプリケーションのアクセス制御を管理したりできます。レジストリは Amazon リソースネーム (ARN) を持っており、これにより、レジストリ内のスキーマ操作に対するさまざまなアクセス許可の整理と設定を可能にしています。

レジストリはデフォルトのまま使用することも可能ですが、必要な数の新しいレジストリを作成することもできます。

AWS Glue Schema Registry での階層

- RegistryName: [string]
  - RegistryArn: [AWS ARN]
  - CreatedTime: [timestamp]
  - UpdatedTime: [timestamp]
- SchemaName: [string]
  - SchemaArn: [AWS ARN]

- DataFormat: [Avro 、Json、Protobuf]
- Compatibility: [例、BACKWARD、BACKWARD_ALL、FORWARD、FORWARD_ALL、FULL、FULL_ALL、NONE、DISABLED]
- Status: [例、PENDING、AVAILABLE、DELETING]
- SchemaCheckpoint: [integer]
- CreatedTime: [timestamp]
- UpdatedTime: [timestamp]

- SchemaVersion: [string]
  - SchemaVersionNumber: [integer]
  - Status: [例、PENDING、AVAILABLE、DELETING、FAILURE]
  - SchemaDefinition: [string, Value: JSON]
  - CreatedTime: [timestamp]

- SchemaVersionMetadata: [list]
  - MetadataKey: [string]
  - MetadataInfo
  - MetadataValue: [string]
  - CreatedTime: [timestamp]

スキーマのバージョニングと互換性

各スキーマは、複数のバージョン作成することができます。バージョニングは、スキーマに適用される互換性ルールによって制御されます。新しいスキーマバージョンの登録に関するリクエストについては、この規則に対するチェックが Schema Registry により行われ後で処理が続行されます。

チェックポイントとしてマークされたスキーマバージョンは、スキーマの新しいバージョンを登録する際の互換性判断のために使用されます。スキーマが最初に作成されると、最初のバージョンがデフォルトのチェックポイントになります。スキーマのバージョンが増加した場合には、CLI/SDK により、一連の制約に従う `UpdateSchema` APIを使用しながら、チェックポイントに使用するスキーマのバージョンを変更できます。デフォルトでは、コンソールでスキーマの定義または互換モードを編集することで、チェックポイントが最新バージョンに変更されます。

互換モードを使用すると、スキーマを時間の経過とともにどのように増加させるか、
あるいは増加させないかを制御できます。これらのモードは、データのプロデューサお

よびコンシューマである、アプリケーション間の契約を形成します。スキーマの新しいバージョンがレジストリに送信されると、そのスキーマ名に適用されている互換性規則を使用して、新しいバージョンの受け付が可能かどうかが判断されます。互換モードには、NONE、DISABLED、BACKWARD、BACKWARD_ALL、FORWARD、FORWARD_ALL、FULL、FULL の 8 種類があります

Avro データ形式では、フィールドはオプションまたは必須の両方の場合があります。フィールドがオプションの場合、Type には null が含まれます。必須フィールドでは、Type は null に設定されません。

Protobuf のデータ形式の場合、 proto2 構文ではフィールドを (繰り返しを含み) オプションまたは必須にすることができます。一方、proto3 構文ではすべてのフィールドが (繰り返しを含み) オプションとなります。すべての互換性ルールは、Protocol Buffers の仕様と、Google の Protocol Buffers ドキュメントを詳細に把握した上で定義されています。

- NONE: 互換モードは適用されません。このモードは、開発向けのシナリオか、スキーマに適用する互換モードがわからない場合に使用できます。新しいバージョンが追加されたとき、互換性チェックを受けずに受け入れられます。
- DISABLED: 互換性についてこの選択をすると、特定のスキーマのバージョン管理が不要になります。新しいバージョンを追加することはできません。
- BACKWARD: これは推奨の互換性モードで、コンシューマは、現在および過去のスキーマバージョンの両方からの読み取りが可能です。このモードを使用すると、フィールドの削除や任意のフィールドの追加を行う際に、以前のスキーマバージョンとの互換性をチェックできます。アプリケーションが最新のスキーマ用に作成された場合などが、BACKWARD の一般的なユースケースです。

  AVRO

  例えば、名前 (必須)、姓 (必須)、E メール (必須)、電話番号 (オプション) で定義されているスキーマがあるとします。

  次のスキーマバージョンで、必須である E メールフィールドが削除されたとしても、これは正常に登録されます。後方互換性のために、コンシューマは、現在および 1 つ前のスキーマバージョンを読み取れることが必要です。古いメッセージの余分な E メールフィールドが無視されるため、コンシューマーは新しいスキーマを読み取ることができます。

  仮に、郵便番号などのフィールドを必須として追加する新しいスキーマバージョンが提案された場合、BACKWARD の互換モードでは正常に登録されません。新しいバージョンのコンシューマーは、必須な郵便番号フィールドを含まない、スキーマが変更される前の古いメッセージを読み取る

ことができないためです。ただし、新しいスキーマで郵便番号フィールドがオプションとして設定されていれば、提案されたバージョンは正常に登録されます。郵便番号フィールドはオプションなので、コンシューマーはこのフィールドなしでも古いスキーマを読み取れるためです。

JSON

例えば、あるスキーマバージョンが、名 (オプション)、姓 (オプション)、E メール (オプション)、電話番号 (オプション)で定義されているとします。

仮に、次に続くスキーマバージョンで電話番号プロパティがオプションとして追加された場合でも、元のスキーマバージョンで `additionalProperties` フィールドを false に設定しプロパティの追加を禁止しているのであれば、この新しいバージョンは正しく登録されます。後方互換性のために、コンシューマは、現在および 1 つ前のスキーマバージョンを読み取れることが必要です。コンシューマは、電話番号プロパティが存在しない元のスキーマで生成されたデータを読み取ることができます。

BACKWARD の互換モードで、オプションの電話番号プロパティを追加する新しいスキーマバージョンが提案された際に、元のスキーマバージョンで `additionalProperties` フィールドを true に設定し任意のプロパティの追加を許可していると、このバージョンは正しく登録されません。新しいバージョンのコンシューマは、スキーマが変更される前の古いメッセージを読み取ることができません。これは、別の型 (数ではなく文字列など) で記述された電話番号プロパティデータを、読み取ることができないためです。

PROTOBUF

例えば、`first name` (必須)、`last name` (必須)、`email` (必須) `phone number` (オプション) フィールドを持つ、proto2 構文の Message `Person` で定義されているスキーマバージョンを考えます。

AVRO での場合と同様に、次に続くスキーマバージョンで、必須である `email` フィールドが削除されたとしても、これは正常に登録されます。後方互換性のために、コンシューマは、現在および 1 つ前のスキーマバージョンを読み取れることが必要です。古いメッセージでは、余分な `email` フィールドが無視されるため、コンシューマーは新しいスキーマを読み取ることができます。

BACKWARD の互換モードでは、仮に `zip code` などのフィールドを必須として追加する新しいスキーマバージョンが提案された場合、これは正常に登録されません。スキーマが変更される前の古いメッセージには、必須な `zip code` フィールドが含まていないため、新しいバージョンのコンシューマーは、そのメッセージを読み取ることができないためです。ただし、新しいスキーマで `zip code` フィールドがオプションとして設定されていれば、提案されたバージョンは正常に登

録されます。zip code フィールドはオプションなので、コンシューマーはこのフィールドなしでも古いスキーマを読み取れます。

gRPC ユースケースの場合、新しい RPC サービスまたは RPC メソッドを追加することで、後方互換性が変更されます。例えば、2 つの RPC メソッド Foo および Bar を使用する RPC サービス MyService で定義されている、スキーマバージョンがあるとします。

次に続くスキーマバージョンで、Baz という新しい RPC メソッドが追加された場合、これは正常に登録されます。新しく追加された RPC メソッド Baz はオプションのため、コンシューマーは元のスキーマで生成されたデータを、BACKWARD 互換性に従って読み取ることができます。

提案された新しいスキーマバージョンで、既存の RPC メソッド Foo を削除する場合、BACKWARD 互換性では正常に登録されません。新しいバージョンのコンシューマーは、スキーマが変更される前の古いメッセージを読み取ることができません。これは、gRPC アプリケーション内に RPC メソッド Foo が存在せず、データを理解して読み取れないためです。

- BACKWARD_ALL: この互換モードでは、コンシューマは現在および過去のすべてのスキーマバージョンの読み取りが可能です。このモードを使用すると、フィールドを削除したり任意のフィールドを追加する際に、以前のすべてのスキーマバージョンとの互換性をチェックできます。
- FORWARD: この互換モードにより、コンシューマは現在とその次のスキーマバージョンの両方を読み取ることができますが、それより後に続くバージョンを必ずしも読み取れる訳ではありません。このオプションを使用すると、フィールドを追加したり、任意のフィールドを削除したりする際に、最新のスキーマバージョンとの互換性をチェックできます。過去のスキーマ用に作成されたアプリケーションが、より新しいスキーマを処理できる必要がある場合などが、FORWARD の一般的なユースケースです。

  AVRO

  例として、名前 (必須)、姓 (必須)、E メール (オプション) で定義されているスキーマバージョンを考えます。

  この場合、電話番号などのフィールドを必須として追加する新しいスキーマバージョンは正常に登録されます。FORWARD 互換モードでは、新しいスキーマで生成されたデータを、コンシューマーが以前のバージョンを使用して読み取れることが必要です。

  必須である名 (ファーストネーム) のフィールドを削除するスキーマバージョンが提案されとしても、FORWARD 互換モードでは正常に登録されません。名の必須フィールドがないので、以前のバージョンのコンシューマーが提案されたスキーマを読み取ることができなくなるためです。しかし、名 (ファーストネーム) のフィールドが本来オプションであれば、オプションの名のフィール

ドを持たない新しいスキーマに基づいたデータをコンシューマーが読み取りできるため、提案された新しいスキーマは正常に登録されます。

JSON

例えば、あるスキーマバージョンが、名 (オプション)、姓 (オプション)、E メール (オプション)、電話番号 (オプション)で定義されているとします。

オプションの電話番号プロパティを削除する新しいスキーマバージョンの場合、このスキーマバージョンで additionalProperties フィールドを false に設定しプロパティの追加を禁止しているのであれば、正しく登録されます。FORWARD 互換モードでは、新しいスキーマで生成されたデータを、コンシューマーが以前のバージョンを使用して読み取れることが必要です。

オプションの電話番号プロパティを削除するスキーマバージョンが提案された際に、この新しいスキーマバージョンで additionalProperties フィールドを true に設定し任意のプロパティの追加を許可していると、FORWARD の互換モードでは正しく登録されません。提案されたスキーマが、異なる型 (数ではなく文字列など) の電話番号プロパティを持つ場合には、以前のバージョンを使用するコンシューマで、このスキーマの読み取りが不可能になるためです。

PROTOBUF

例えば、first name (必須)、last name (必須)、email (オプション) のフィールドを持つ、proto2 構文の Message Person で定義されているスキーマバージョンを考えます。

AVRO の場合と同様に、phone number などの必須フィールドを追加する新しいスキーマバージョンの場合、これは正常に登録されます。FORWARD 互換モードでは、新しいスキーマで生成されたデータを、コンシューマーが以前のバージョンを使用して読み取れることが必要です。

必須である名 first name フィールドを削除するスキーマバージョンが提案されとしても、FORWARD 互換モードでは正常に登録されません。必須である first name フィールドがないので、以前のバージョンのコンシューマーは、提案されたスキーマを読み取ることができなくなるためです。しかし、first name フィールドが本来オプションであれば、新しいスキーマに first name フィールドがないとしてもこれはオプションであるため、コンシューマーはこのスキーマ基づいたデータを読み取ることが可能であり、したがって提案された新しいスキーマは正常に登録されます。

gRPC ユースケースの場合、RPC サービスまたは RPC メソッドを削除すると、前方互換性が変更されます。例えば、2 つの RPC メソッド Foo および Bar を使用する RPC サービス MyService で定義されている、スキーマバージョンがあるとします。

次に続くスキーマバージョンが、Foo という名前の既存の RPC メソッドを削除する場合も、これは FORWARD 互換性に従って正常に登録されます。コンシューマーは、新しいスキーマで生成されたデータを、以前のバージョンを使用して読み取ることが可能です。RPC メソッド Baz を追加するスキーマバージョンが提案され場合、これは FORWARD 互換モードでは正常に登録されません。RPC メソッド Baz がないので、以前のバージョンのコンシューマーは、この提案されたスキーマを読み取ることができないためです。

- FORWARD_ALL: この互換モードでは、任意の新しい登録済みスキーマのプロデューサーによって書き込まれたデータを読み取ることを、コンシューマに許可します。この選択は、フィールドを追加したり、オプションフィールドを削除したり、以前のすべてのスキーマバージョンとの互換性を確認したりする必要がある場合に使用できます。
- FULL: この互換モードにより、コンシューマーは、1 つ前または次のバージョンのスキーマを使用するプロデューサーによって書き込まれたデータを読み取ることができますが、それ以前またはそれ以降のバージョンについては読み取れません。このモードでは、任意のフィールドを追加または削除する際に、最新のスキーマバージョンとの互換性をチェックできます。
- FULL_ALL: この互換モードにより、コンシューマーは、過去の任意のスキーマバージョンを使用するプロデューサーによって書き込まれたデータを読み取ることができます。このモードは、オプションのフィールドを追加または削除する際に、過去のすべてのスキーマバージョンとの互換性をチェックするために使用できます。

オープンソース Serde ライブラリ

AWS では、データをシリアル化および非シリアル化するためのフレームワークとして、オープンソースの Serde ライブラリが用意されています。このオープンソースなライブラリ設計により、一般的なオープンソースのアプリケーションやフレームワークが、それぞれのプロジェクトでこれらのライブラリをサポートできるようにしています。

Serde ライブラリの仕組みの詳細については、「スキーマレジストリの仕組み」を参照してください。

Schema Registry のクォータ

AWS のクォータ (制限とも呼ばれます) は、AWS アカウントのリソース、アクション、および制限の最大値です。以下に、AWS Glue の Schema Registry におけるソフト制限を示します。

スキーマバージョンのメタデータでのキー値ペアの数。

AWS リージョンごとの各 SchemaVersion において、最大 10 個のキーと値のペアを使用できます。

メタデータでのキーと値のペアは、QuerySchemaVersionMetadata アクション (Python: query_schema_version_metadata) または PutSchemaVersionMetadata アクション (Python: put_schema_version_metadata) APIを使用して表示または設定できます。

AWS Glue の Schema Registry でのハード制限は次のとおりです。

レジストリ

このアカウントでは、AWS リージョンごとに最大 100 のレジストリを使用できます。

SchemaVersion

このアカウントでは、AWS リージョンごとに最大 10000 のスキーマバージョンを使用できます。

各新しいスキーマは新しいスキーマバージョンを作成します。したがって、各スキーマに 1 つのバージョンしかない場合、リージョンごとの各アカウントには理論的に最大 10000 のスキーマを使用できます。

スキーマペイロード

スキーマペイロードには、170 KB のサイズ制限があります。

スキーマレジストリの仕組み

このセクションでは、スキーマレジストリでシリアル化および非シリアル化のプロセスの仕組みについて説明します。

1. スキーマの登録: スキーマがレジストリにまだ存在しない場合、スキーマを保存先の名前と同じスキーマ名で登録することができます (test_topic、test_stream、prod_firehose など)。あるいは、プロデューサーがスキーマのカスタム名を指定することも可能です。プロデューサーは、source: msk_kafka_topic_A などのメタデータとしてスキーマにキーと値のペアを追加したり、スキーマ作成時に AWS タグをそのスキーマに追加したりできます。スキーマが登録されると、Schema Registry はスキーマのバージョン ID をシリアライザに返します。スキーマが存在するものの、シリアライザが存在しない新しいバージョンを使用している場合、それを新しいバージョンとして登録する前に、Schema Registry はスキーマ参照で互換性ルールを確認して、新しいバージョンに関する互換性を確保します。

   スキーマの登録には、手動登録と自動登録の 2 つの方法があります。スキーマを手動で登録するには、AWS Glue コンソールまたは CLI/SDK を使用します。

   シリアライザ設定で自動登録が有効化されている場合は、スキーマの自動登録が実行されます。プロデューサ設定で `REGISTRY_NAME` が指定されていない場合、自動登録はデフォルトのレジス

トリ (default-registry) の下に新しいスキーマバージョンを登録します。自動登録プロパティの指定については、「SerDe ライブラリのインストール」を参照してください。

2. シリアライザがスキーマに関してデータレコードを検証: データを生成するアプリケーションがそのスキーマを登録した場合、Schema Registry のシリアライザは、アプリケーションによって生成されるレコードが、登録されたスキーマに一致するフィールドとデータ型で構造化されていることを検証します。レコードのスキーマが登録されたスキーマと一致しない場合、シリアライザは例外を出力し、アプリケーションは、そのレコードを送信先にデリバリすることに失敗します。

   スキーマが存在せず、プロデューサー設定を介してスキーマ名が指定されてもいない場合は、スキーマがトピック名 (Apache Kafka または Amazon MSK の場合) あるいはストリーム名 (Kinesis Data Streams の場合) と同じ名前で作成されます。

   すべてのレコードは、スキーマ定義とデータを含みます。スキーマ定義は、Schema Registry 内の既存のスキーマおよびバージョンに関して照会されます。

   デフォルトでは、プロデューサは、登録済みスキーマのスキーマ定義とスキーマバージョン ID をキャッシュします。レコードのスキーマバージョン定義がキャッシュ内に保存されたものと一致しない場合、プロデューサーは Schema Registry を使用してスキーマの検証を試みます。スキーマのバージョンが有効と認められた場合、そのバージョン ID と定義はプロデューサにローカルにキャッシュされます。

   SerDe ライブラリのインストール のステップ 3 のように、オプションのプロデューサプロパティにより、デフォルトのキャッシュ期間 (24 時間) を調整することが可能です。

3. レコードのシリアル化と配信: レコードがスキーマに準拠している場合、シリアライザは各レコードをスキーマのバージョン ID で装飾し、選択したデータ形式 (AVRO、JSON、Protobuf 他の形式は追加予定) に基づいてシリアル化した上で、そのレコードを (プロデューサーのオプション設定により) 圧縮し送信先に送ります。

4. コンシューマーがデータを非シリアル化: このデータを読み取るコンシューマーは、レコードペイロードからのスキーマバージョン ID を解析する、Schema Registry のデシリアライザ用ライブラリを使用します。

5. デシリアライザが Schema Registry からのスキーマをリクエスト: デシリアライザが特定のスキーマバージョン ID を持つレコードを初めて認識すると、そのデシリアライザは、スキーマバージョン ID を使用してスキーマレジストリからスキーマを要求します。その後、そのスキーマをコンシューマ上でローカルにキャッシュします。Schema Registry でレコードの非シリアル化が不可能な場合、コンシューマはレコードからのデータをログに記録し、処理を続行するかアプリケーションを停止できます。

6. デシリアライザがスキーマを使用してレコードを非シリアル化: デシリアライザは、Schema Registry からスキーマバージョン ID を取得する際に (プロデューサーによって送信されたレコードが圧縮されている場合は) レコードを解凍し、スキーマを使用してレコードを非シリアル化します。これで、アプリケーションはレコードを処理できます。

> **Note**
>
> 暗号化: クライアントは、HTTPS 上での TLS 暗号化を使用して転送中のデータを暗号化する API 呼び出しを介して Schema Registry と通信します。Schema Registry に保存されているスキーマは、サービス管理のAWS Key Management Service(AWS KMS) キーにより常に暗号化されます。

> **Note**
>
> ユーザー認可: Schema Registry は、アイデンティティベースの IAM ポリシーをサポートします。

スキーマレジストリの開始方法

以下のセクションでは、Schema Registry のセットアップと使用に関する概要とチュートリアルを示します。スキーマレジストリの概念および各コンポーネントの詳細については、「AWS Glue スキーマレジストリ」を参照してください。

トピック

- SerDe ライブラリのインストール
- レジストリを作成する
- スキーマの作成
- スキーマまたはレジストリの更新
- スキーマまたはレジストリの削除
- シリアライザ用の IAM の例
- デシリアライザー用の IAM の例
- AWS PrivateLink を使用したプライベート接続
- Amazon CloudWatch メトリクスへのアクセス

- スキーマレジストリの AWS CloudFormation テンプレート例

SerDe ライブラリのインストール

> **Note**
>
> 前提条件: 次のステップを実行する前に、Amazon Managed Streaming for Apache Kafka (Amazon MSK) または Apache Kafka クラスターを起動しておく必要があります。使用するプロデューサーとコンシューマーは、Java 8 以上で実行する必要があります。

SerDe ライブラリは、データのシリアル化と非シリアル化のためのフレームワークを提供します。

データを生成するアプリケーション (総称してシリアライザ) 用の、オープンソースのシリアライザをインストールします。シリアライザは、シリアル化、圧縮、および Schema Registry とのやり取りを処理します。シリアライザは、Schema Registry 対応の送信先 (Amazon MSK など) に書き込まれるレコードから、スキーマを自動的に抽出します。同様に、データを利用するアプリケーションには、オープンソースのデシリアライザをインストールします。

プロデューサとコンシューマにライブラリをインストールするには

1. プロデューサとコンシューマ両方の pom.xml ファイルの中で、以下のコードにより依存関係を追加します。

   ```
   <dependency>
       <groupId>software.amazon.glue</groupId>
       <artifactId>schema-registry-serde</artifactId>
       <version>1.1.5</version>
   </dependency>
   ```

   または、AWS Glue Schema Registry GitHub リポジトリからクローンを作成することもできます。
2. 次の必須プロパティを使用してプロデューサをセットアップします。

   ```
   props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
    StringSerializer.class.getName()); // Can replace StringSerializer.class.getName())
    with any other key serializer that you may use
   props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
    GlueSchemaRegistryKafkaSerializer.class.getName());
   ```

```
props.put(AWSSchemaRegistryConstants.AWS_REGION, "us-east-2");
properties.put(AWSSchemaRegistryConstants.DATA_FORMAT, "JSON"); // OR "AVRO"
```

既存のスキーマがない場合は、自動登録を有効にします (次のステップ)。適用できるスキーマがある場合は、「my-schema」をそのスキーマ名に置き換えます。また、スキーマの自動登録が無効になっている場合は、「registry-name」を指定する必要もあります。スキーマが「default-registry」の下に作成されている場合は、レジストリ名を省略できます。

3. (オプション) これらのオプションのプロデューサープロパティのいずれかを設定します。プロパティの詳細については、ReadMe ファイルを参照してください。

```
props.put(AWSSchemaRegistryConstants.SCHEMA_AUTO_REGISTRATION_SETTING, "true"); // If
 not passed, uses "false"
props.put(AWSSchemaRegistryConstants.SCHEMA_NAME, "my-schema"); // If not passed,
 uses transport name (topic name in case of Kafka, or stream name in case of Kinesis
 Data Streams)
props.put(AWSSchemaRegistryConstants.REGISTRY_NAME, "my-registry"); // If not passed,
 uses "default-registry"
props.put(AWSSchemaRegistryConstants.CACHE_TIME_TO_LIVE_MILLIS, "86400000"); // If
 not passed, uses 86400000 (24 Hours)
props.put(AWSSchemaRegistryConstants.CACHE_SIZE, "10"); // default value is 200
props.put(AWSSchemaRegistryConstants.COMPATIBILITY_SETTING, Compatibility.FULL); //
 Pass a compatibility mode. If not passed, uses Compatibility.BACKWARD
props.put(AWSSchemaRegistryConstants.DESCRIPTION, "This registry is used for several
 purposes."); // If not passed, constructs a description
props.put(AWSSchemaRegistryConstants.COMPRESSION_TYPE,
 AWSSchemaRegistryConstants.COMPRESSION.ZLIB); // If not passed, records are sent
 uncompressed
```

自動登録では、スキーマのバージョンがデフォルトのレジストリ (default-registry) に登録されます。SCHEMA_NAME が前のステップで指定されていない場合、トピック名は SCHEMA_NAME として推定されます。

互換モードの詳細については、「スキーマのバージョニングと互換性」を参照してください。

4. 以下の必須プロパティを使用してコンシューマを設定します。

```
props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,
 StringDeserializer.class.getName());
props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,
 GlueSchemaRegistryKafkaDeserializer.class.getName());
props.put(AWSSchemaRegistryConstants.AWS_REGION, "us-east-2"); // Pass an AWS #####
```

```
props.put(AWSSchemaRegistryConstants.AVRO_RECORD_TYPE,
 AvroRecordType.GENERIC_RECORD.getName()); // Only required for AVRO data format
```

5. (オプション) これらのオプションのコンシューマプロパティを設定します。プロパティの詳細については、ReadMe ファイルを参照してください。

```
properties.put(AWSSchemaRegistryConstants.CACHE_TIME_TO_LIVE_MILLIS, "86400000"); //
 If not passed, uses 86400000
props.put(AWSSchemaRegistryConstants.CACHE_SIZE, "10"); // default value is 200
props.put(AWSSchemaRegistryConstants.SECONDARY_DESERIALIZER,
 "com.amazonaws.services.schemaregistry.deserializers.external.ThirdPartyDeserializer"); //
 For migration fall back scenario
```

レジストリを作成する

AWS Glue API または AWS Glue コンソールを使用して、デフォルトのレジストリを使用することも、必要な数の新しいレジストリを作成することもできます。

AWS Glue API

ここでの手順により、AWS Glue API を使用しながら対象のタスクを実行できます。

AWS Glue Schema Registry API で AWS CLI を使用するには、最新バージョンの AWS CLI 使用する必要が有ります。

新しいレジストリを追加するには、CreateRegistry アクション (Python: create_registry) APIを使用します。RegistryName では、作成するレジストリの名前を指定します。この文字数は最大 255 までで、文字、数字、ハイフン、アンダースコア、ドル記号、およびハッシュ記号のみ使用できます。

2,048 バイト以下で「URI アドレスの複数行の文字列パターン」に一致する文字列として Description を指定します。

オプションで、キーと値のペアのマップ配列として、1 つ以上の Tags をレジストリに指定します。

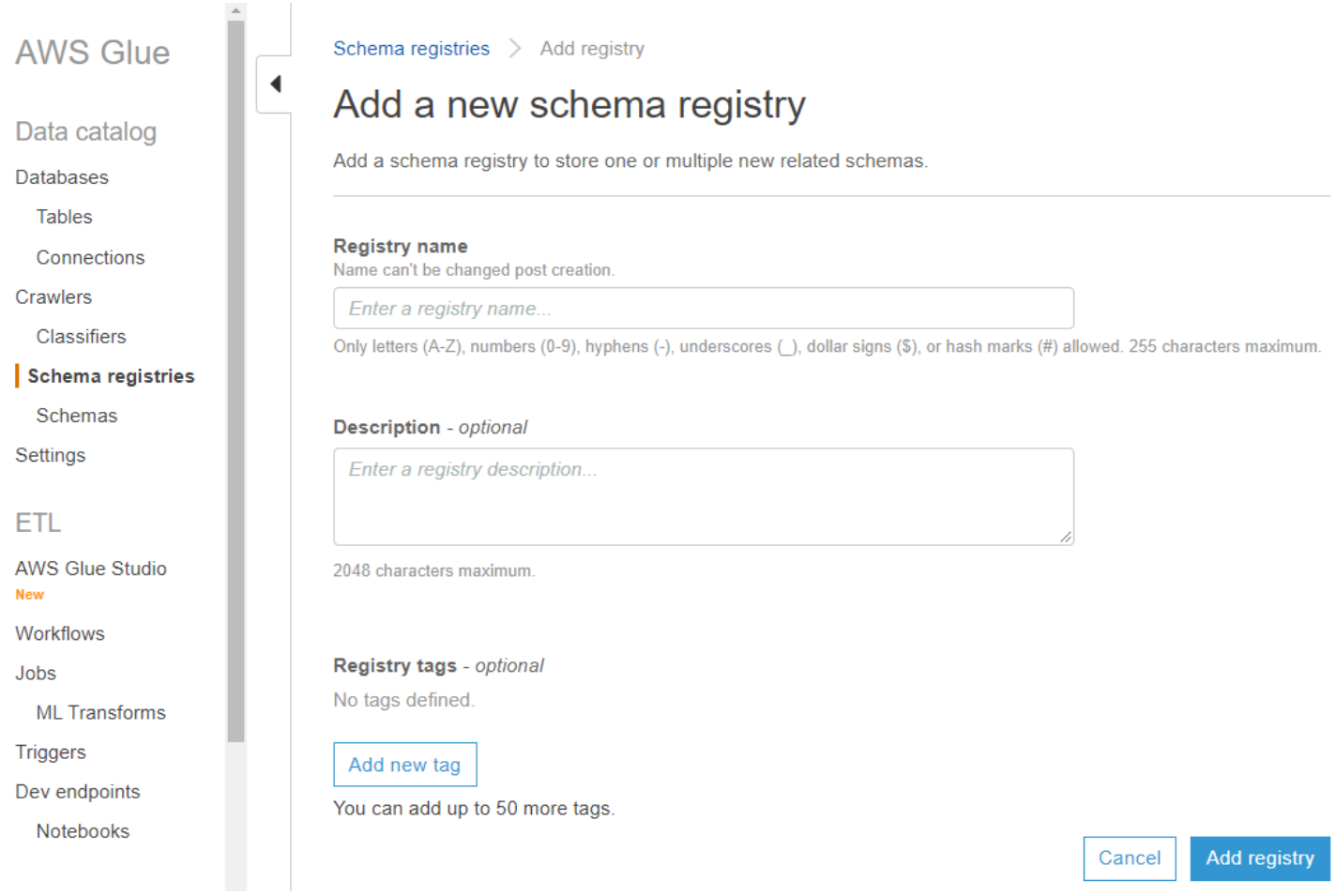
```
aws glue create-registry --registry-name registryName1 --description description
```

作成されたレジストリには、Amazon リソースネーム (ARN) が割り当てられます。これは、RegistryArn API 応答により表示することが可能です。この段階でレジストリ作成が完了しているので、そのレジストリのために 1 つ以上のスキーマを作成します。

AWS Glue コンソール

AWS Glue コンソールで新しいレジストリを追加するには

1. AWS Management Consoleにサインインし、AWS Glue コンソール (https://console.aws.amazon.com/glue/) を開きます。
2. ナビゲーションペインの [Data catalog] (データカタログ) で、[Schema registries] (スキーマレジストリ) をクリックします。
3. [Add registry] (レジストリを追加) をクリックします。
4. [Registry name] (レジストリ名) に、文字、数字、ハイフン、アンダースコアを含むレジストリの名前を入力します。この名前は変更できません。
5. レジストリの [Description] (説明) を入力します　(オプション)。
6. オプションで、1 つ以上のタグをレジストリに適用します。[Add new tag] (新しいタグを追加) を選択し、[Tag key] (タグキー) とオプションの [Tag value] (タグ値) を指定します。
7. [Add registry] (レジストリを追加) をクリックします。



作成されたレジストリには、Amazon リソースネーム (ARN) が割り当てられます。これは、[Schema registries] (スキーマレジストリ) のリストから選択して表示することが可能です。この段階でレジストリ作成が完了しているので、そのレジストリのために 1 つ以上のスキーマを作成します。

JSON の特定のレコード (JAVA POJO) 処理する

従来の単純な Java オブジェクト (POJO) を使用して、オブジェクトをレコードとして渡すことができます。これは、AVRO における特定のレコードの概念と類似しています。mbknor-jackson-jsonschema により、渡された POJO の JSON スキーマを生成できます。また、このライブラリでは、JSON スキーマに追加情報を挿入することもできます。

AWS Glue Schema Registry ライブラリは、スキーマに注入された「className」フィールドを使用して、完全に分類されたクラス名を提供します。「className」フィールドは、そのクラスのオブジェクト内での非シリアル化のために、デシリアライザによって使用されます。

```
 Example class :

@JsonSchemaDescription("This is a car")
@JsonSchemaTitle("Simple Car Schema")
@Builder
@AllArgsConstructor
@EqualsAndHashCode
// Fully qualified class name to be added to an additionally injected property
// called className for deserializer to determine which class to deserialize
// the bytes into
@JsonSchemaInject(
        strings = {@JsonSchemaString(path = "className",
                value =
 "com.amazonaws.services.schemaregistry.integrationtests.generators.Car")}
)
// List of annotations to help infer JSON Schema are defined by https://github.com/
mbknor/mbknor-jackson-jsonSchema
public class Car {
    @JsonProperty(required = true)
    private String make;

    @JsonProperty(required = true)
    private String model;

    @JsonSchemaDefault("true")
    @JsonProperty
```

```
    public boolean used;

    @JsonSchemaInject(ints = {@JsonSchemaInt(path = "multipleOf", value = 1000)})
    @Max(200000)
    @JsonProperty
    private int miles;

    @Min(2000)
    @JsonProperty
    private int year;

    @JsonProperty
    private Date purchaseDate;

    @JsonProperty
    @JsonFormat(shape = JsonFormat.Shape.NUMBER)
    private Date listedDate;

    @JsonProperty
    private String[] owners;

    @JsonProperty
    private Collection<Float> serviceChecks;

    // Empty constructor is required by Jackson to deserialize bytes
    // into an Object of this class
    public Car() {}
}
```

スキーマの作成

スキーマは、AWS Glue API または AWS Glue コンソールを使用して作成できます。

AWS Glue API

ここでの手順により、AWS Glue API を使用しながら対象のタスクを実行できます。

新しいスキーマを追加するには CreateSchema アクション (Python: create_schema) API を使用します。

`RegistryId` 構造体を使用して、スキーマのレジストリを指定します。または、`RegistryId` を省略すると、デフォルトのレジストリが使用されます。

文字、数字、ハイフン、アンダースコアを使用し、DataFormat として **AVRO** または **JSON** を設定しながら SchemaName を指定します。一度スキーマに設定された DataFormat は変更できません。

Compatibility モードを指定する。

- Backward (推奨) – コンシューマは、現在のバージョンと 1 つ前のバージョンの両方を読み取ることができます。
- Backward all – コンシューマは、現在のバージョンと過去のすべてのバージョンを読み取ることができます。
- Forward – コンシューマは、現在のバージョンと次に続くバージョンの両方を読み取ることができます。
- Forward all – コンシューマは、現在のバージョンと後続のすべてのバージョンの両方を読み取ることができます。
- Full – Backward all と Forward all を組み合わせたモードです。
- Full all – Backward all と Forward all を組み合わせたモードです。
- None – 互換性チェックは実行されません。
- Disabled – このスキーマのバージョニングを抑止します。

オプションで、スキーマに Tags を指定します。

SchemaDefinition を指定することで、スキーマのデータ形式を Avro、JSON、もしくは Protobuf として定義します。例を参照してください。

Avro データ形式の場合:

```
aws glue create-schema --registry-id RegistryName="registryName1" --schema-name
 testschema --compatibility NONE --data-format AVRO --schema-definition "{\"type\":
 \"record\", \"name\": \"r1\", \"fields\": [ {\"name\": \"f1\", \"type\": \"int\"},
 {\"name\": \"f2\", \"type\": \"string\"} ]}"
```

```
aws glue create-schema --registry-id RegistryArn="arn:aws:glue:us-
east-2:901234567890:registry/registryName1" --schema-name testschema --compatibility
 NONE --data-format AVRO  --schema-definition "{\"type\": \"record\", \"name\": \"r1\",
 \"fields\": [ {\"name\": \"f1\", \"type\": \"int\"}, {\"name\": \"f2\", \"type\":
 \"string\"} ]}"
```

JSON データ形式の場合:

```
aws glue create-schema --registry-id RegistryName="registryName" --schema-name
 testSchemaJson --compatibility NONE --data-format JSON --schema-definition "{\"$schema
\": \"http://json-schema.org/draft-07/schema#\",\"type\":\"object\",\"properties\":
{\"f1\":{\"type\":\"string\"}}}"
```

```
aws glue create-schema --registry-id RegistryArn="arn:aws:glue:us-
east-2:901234567890:registry/registryName" --schema-name testSchemaJson --compatibility
 NONE --data-format JSON --schema-definition "{\"$schema\": \"http://json-schema.org/
draft-07/schema#\",\"type\":\"object\",\"properties\":{\"f1\":{\"type\":\"string\"}}}"
```

Protobuf データ形式の場合:

```
aws glue create-schema --registry-id RegistryName="registryName" --schema-name
 testSchemaProtobuf --compatibility NONE --data-format PROTOBUF --schema-definition
 "syntax = \"proto2\";package org.test;message Basic { optional int32 basic = 1;}"
```

```
aws glue create-schema --registry-id RegistryArn="arn:aws:glue:us-
east-2:901234567890:registry/registryName" --schema-name testSchemaProtobuf
 --compatibility NONE --data-format PROTOBUF --schema-definition "syntax =
 \"proto2\";package org.test;message Basic { optional int32 basic = 1;}"
```

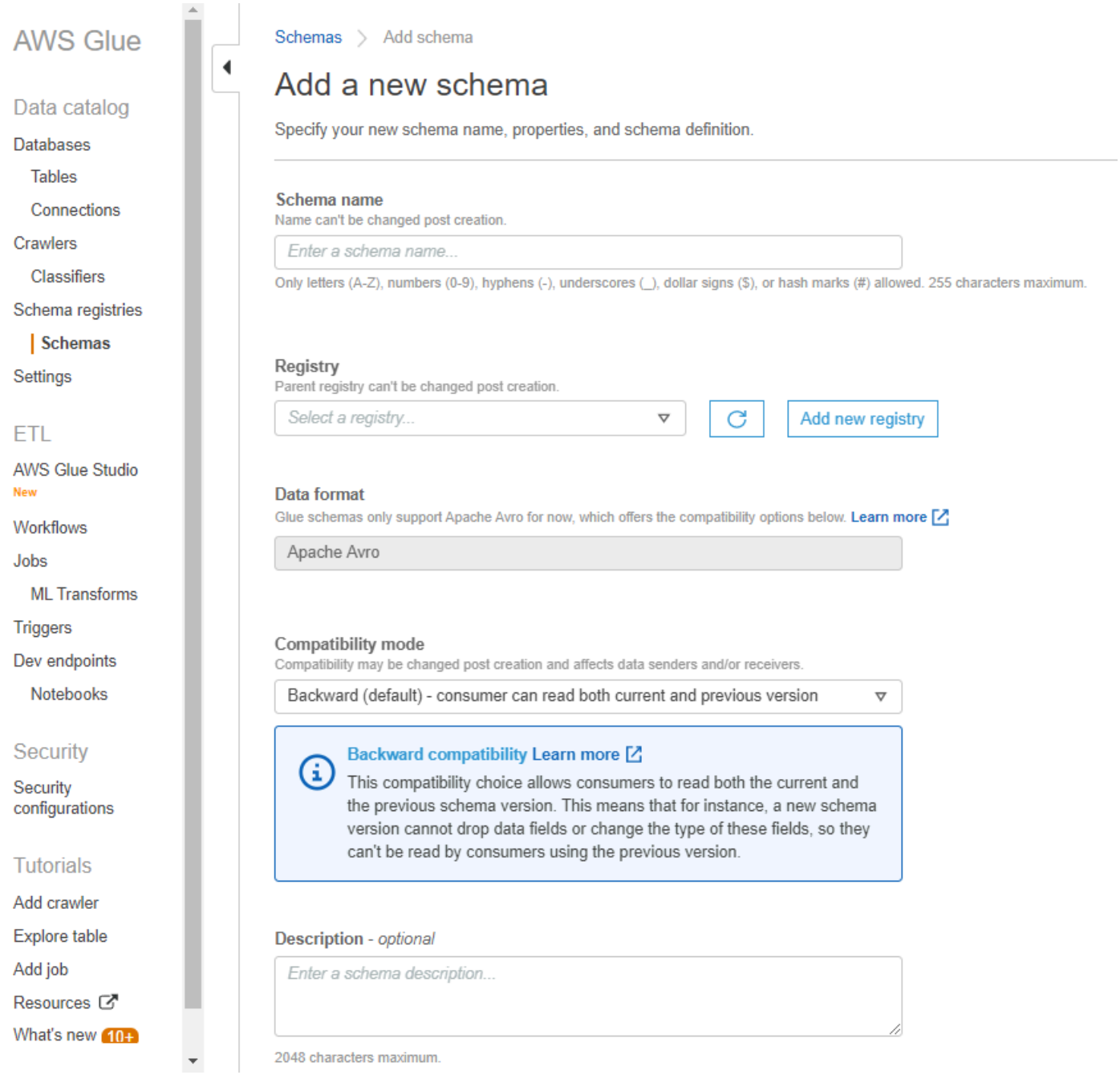
AWS Glue コンソール

AWS Glue コンソールを使用して新しいスキーマを追加するには

1. AWS マネジメントコンソールにサインインし、AWS Glue コンソール (https://console.aws.amazon.com/glue/) を開きます。
2. ナビゲーションペインの [Data catalog] (データカタログ) で、[Schemas] (スキーマ) をクリックします。
3. [Add schema] (スキーマを追加) をクリックします。
4. [Schema name] (スキーマ名) に、文字、数字、ハイフン、アンダースコア、ドル記号、ハッシュマークで構成された名前を入力します。この名前は変更できません。
5. [Registry] (レジストリ) ドロップダウンメニューから、スキーマの保存先となるレジストリを選択します。作成後は、親レジストリを変更することはできません。
6. [Data format] (データ形式) は、「Apache Avro」または「JSON」のままにしておきます。この形式は、このスキーマのすべてのバージョンに適用されます。
7. [Compatibility mode] (互換モード) をクリックします。

- Backward (推奨) – レシーバーは、現在のバージョンと 1 つ前のバージョンの両方を読み取ることができます。
- Backward All – レシーバーは、現在および過去のすべてのバージョンを読み取ることができます。
- Forward – センダーは、現在のバージョンと 1 つ前のバージョンの両方に書き込むことができます。
- Forward All – センダーは、現在のバージョンと過去のすべてのバージョンに書き込むことができます。
- Full – Backward と Forward を組み合わせたモードです。
- Full All – Backward All と Forward All を組み合わせたモードです。
- None – 互換性チェックは実行されません。
- Disabled – このスキーマのバージョニングを抑止します。

8. [Description] (説明) に、レジストリのための説明 (オプション) を、最大 250 文字で入力します。



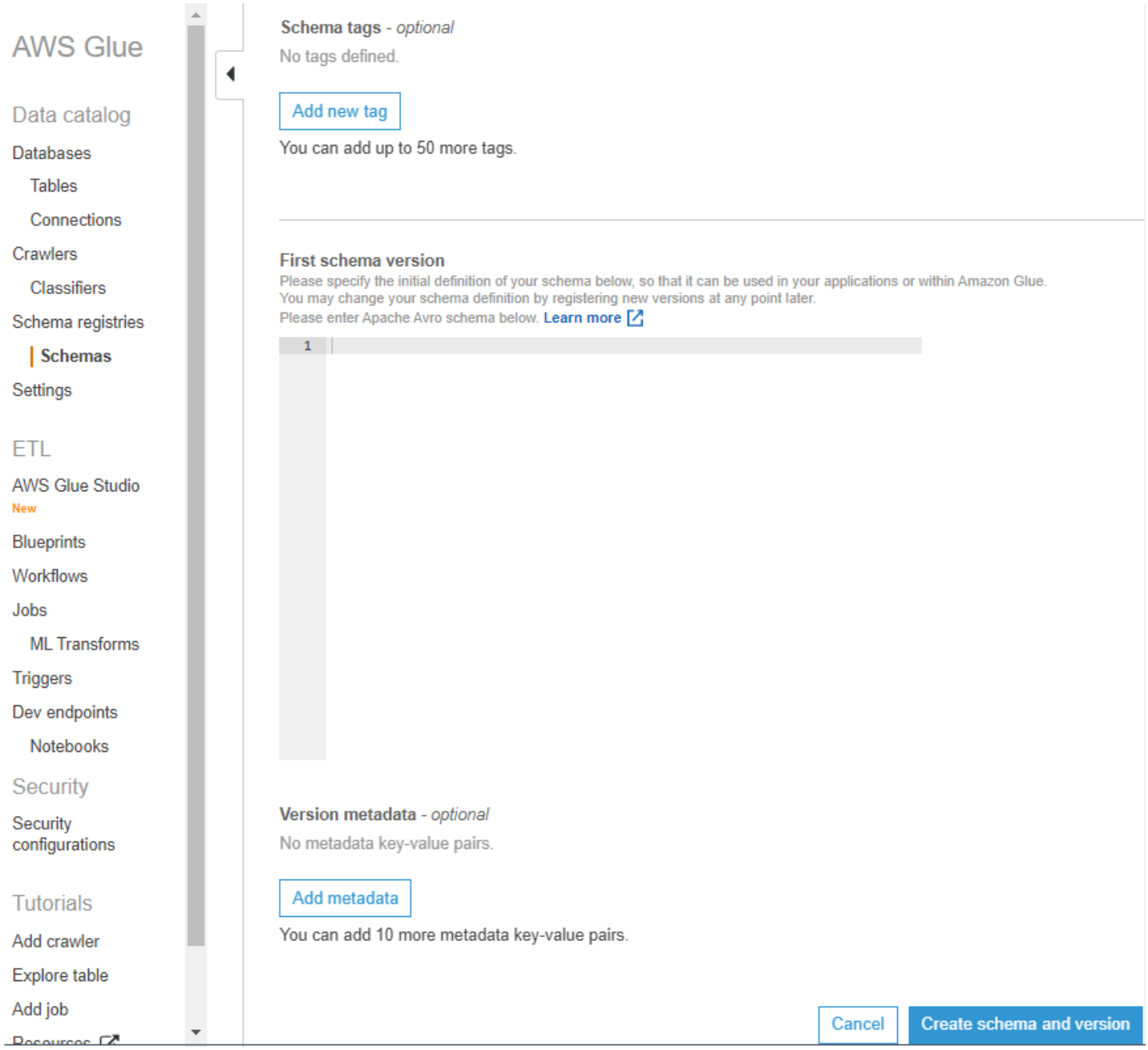
9. オプションで、1 つ以上のタグをスキーマに適用します。[Add new tag] (新しいタグを追加) を選択し、[Tag key] (タグキー) とオプションの [Tag value] (タグ値) を指定します。
10. [First schema バージョニング] (最初のスキーマバージョン) ボックスに、初期スキーマを入力するか貼り付けます。

    Avro 形式については「Avro データ形式での作業」を参照

    JSON 形式については「JSON データ形式での操作」を参照

11必要に応じて、[Add metadata] (メタデータを追加) をクリックして、スキーマバージョンの注釈付けや分類を行うためのバージョンメタデータを追加します。

12[Create schema and version] (スキーマとバージョンを作成する) をクリックします。



スキーマが作成され、[Schemas] (スキーマ) の下に一覧表示されます。

Avro データ形式での作業

Avro では、データのシリアル化とデータ交換サービスが利用できます。Avro は、可読性と解釈しやすさのために、データ定義が JSON 形式で格納されます。データ自体はバイナリ形式で保存されます。

Apache Avro スキーマの定義については、「Apache Avro specification」を参照してください。

JSON データ形式での操作

JSON 形式では、データをシリアル化できます。JSON スキーマ形式の標準は、「JSON Schema format」で定義されています。

スキーマまたはレジストリの更新

作成したスキーマ、スキーマバージョン、またはレジストリは、編集することができます。

レジストリの更新

レジストリは、AWS Glue API または AWS Glue コンソールを使用して更新できます。既存のレジストリの名前は変更できません。レジストリの説明は編集が可能です。

AWS Glue API

既存のレジストリを更新するには、UpdateRegistry アクション (Python: update_registry) API を使用します。

`RegistryId` 構造体を使用して、更新するレジストリを指定します。レジストリの説明を変更するには、`Description` を渡します。

```
aws glue update-registry --description updatedDescription --registry-id
 RegistryArn="arn:aws:glue:us-east-2:901234567890:registry/registryName1"
```

AWS Glue コンソール

AWS Glue コンソールを使用してレジストリを更新するには

1. AWS Management Consoleにサインインし、AWS Glue コンソール (https://console.aws.amazon.com/glue/) を開きます。
2. ナビゲーションペインの [Data catalog] (データカタログ) で、[Schema registries] (スキーマレジストリ) をクリックします。

3. レジストリの一覧から、そのチェックボックスをオンにして、レジストリを選択します。
4. [Action] (アクション) メニューで、[Edit registry] (レジストリの編集) を選択します。

スキーマの更新

スキーマでは、説明または互換性設定の更新が行えます。

既存のスキーマを更新するには、UpdateSchema アクション (Python: update_schema) API を使用します。

SchemaId 構造体を使用して、更新するスキーマを指定します。VersionNumber または Compatibility のいずれかを指定する必要があります。

コード例 11:

```
aws glue update-schema --description testDescription --schema-id
 SchemaName="testSchema1",RegistryName="registryName1" --schema-version-number
 LatestVersion=true --compatibility NONE
```

```
aws glue update-schema --description testDescription --schema-id
 SchemaArn="arn:aws:glue:us-east-2:901234567890:schema/registryName1/testSchema1" --
schema-version-number LatestVersion=true --compatibility NONE
```

スキーマバージョンの追加。

スキーマバージョンを追加する際は、そのバージョンを比較して、新しいスキーマが受け入れられることを確認する必要があります。

既存のスキーマに新しいバージョンを追加するには、RegisterSchemaVersion アクション (Python: register_schema_version) API を使用します。

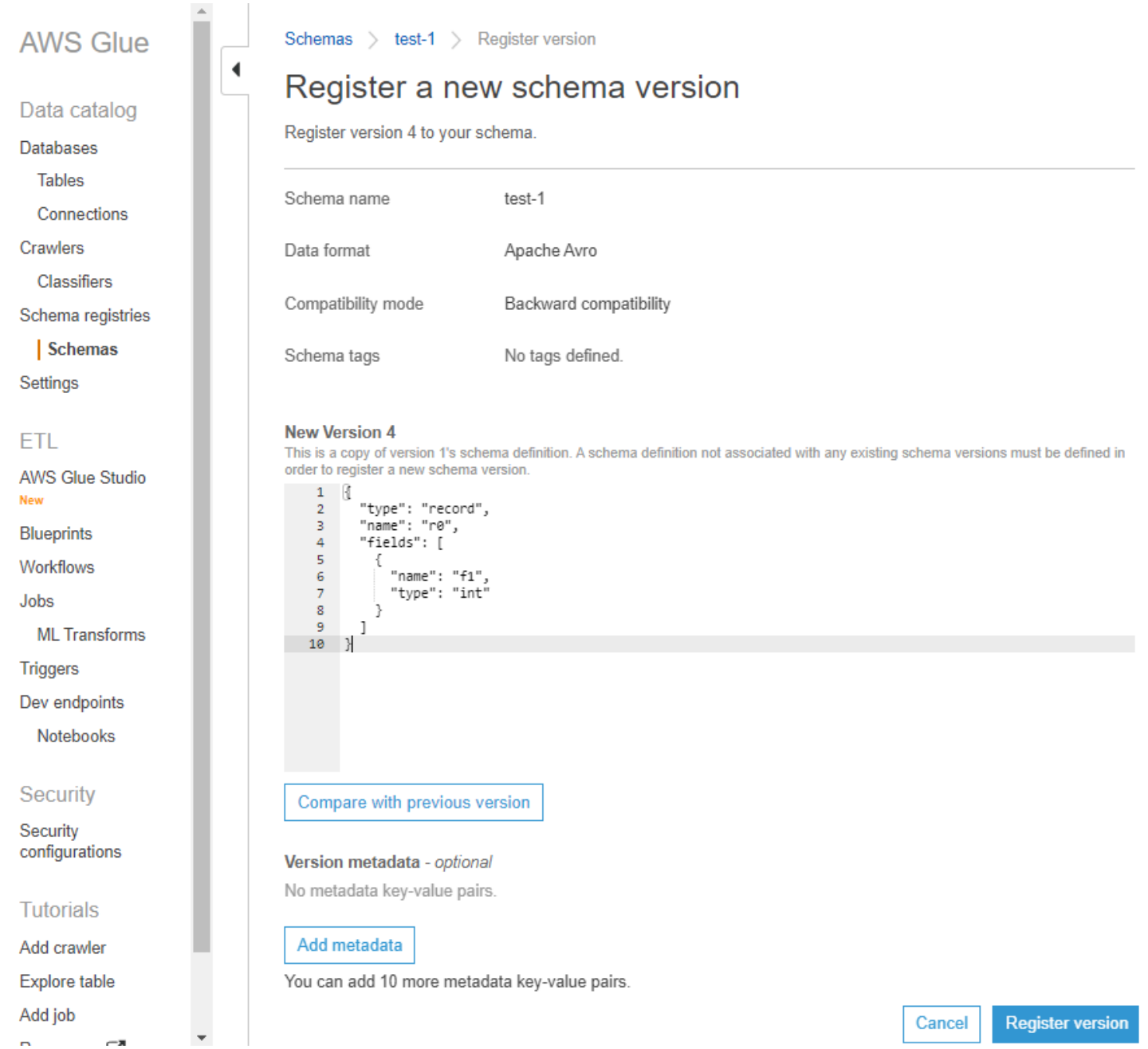
SchemaId 構造体を使用して、バージョンを追加するスキーマを指定し、SchemaDefinition によりスキーマを定義します。

コード例 12:

```
aws glue register-schema-version --schema-definition "{\"type\": \"record\", \"name\":
 \"r1\", \"fields\": [ {\"name\": \"f1\", \"type\": \"int\"}, {\"name\": \"f2\", \"type
\": \"string\"} ]}" --schema-id SchemaArn="arn:aws:glue:us-east-1:901234567890:schema/
registryName/testschema"
```

```
aws glue register-schema-version --schema-definition "{\"type\": \"record\", \"name\":
 \"r1\", \"fields\": [ {\"name\": \"f1\", \"type\": \"int\"}, {\"name\": \"f2\", \"type
\": \"string\"} ]}" --schema-id SchemaName="testschema",RegistryName="testregistry"
```

1. AWS Management Consoleにサインインし、AWS Glue コンソール (https://console.aws.amazon.com/glue/) を開きます。
2. ナビゲーションペインの [Data catalog] (データカタログ) で、[Schemas] (スキーマ) をクリックします。
3. チェックボックスをオンにして、スキーマのリストからスキーマを選択します。
4. チェックボックスをクリックして、リストから 1 つ以上のスキーマを選択します。
5. [Action] (アクションメニュー) から、[Register new version] (新しいバージョンを登録) を選択します。
6. [New version] (新しいバージョン) ボックスに、新しいスキーマを入力または貼り付けます。
7. [Compare with previous version] (過去のバージョンと比較) をクリックして、以前のスキーマのバージョンとの違いを確認します。
8. 必要に応じて、[Add metadata] (メタデータを追加) をクリックして、スキーマバージョンの注釈付けや分類を行うためのバージョンメタデータを追加します。[Key] (キー) および (オプションの) [Value] (値) を入力します。
9. [Register version] (バージョンの登録) をクリックします。



バージョンの一覧の中に、スキーマのバージョンが表示されます。バージョンで互換モードが変更された場合、そのバージョンはチェックポイントとしてマークされます。

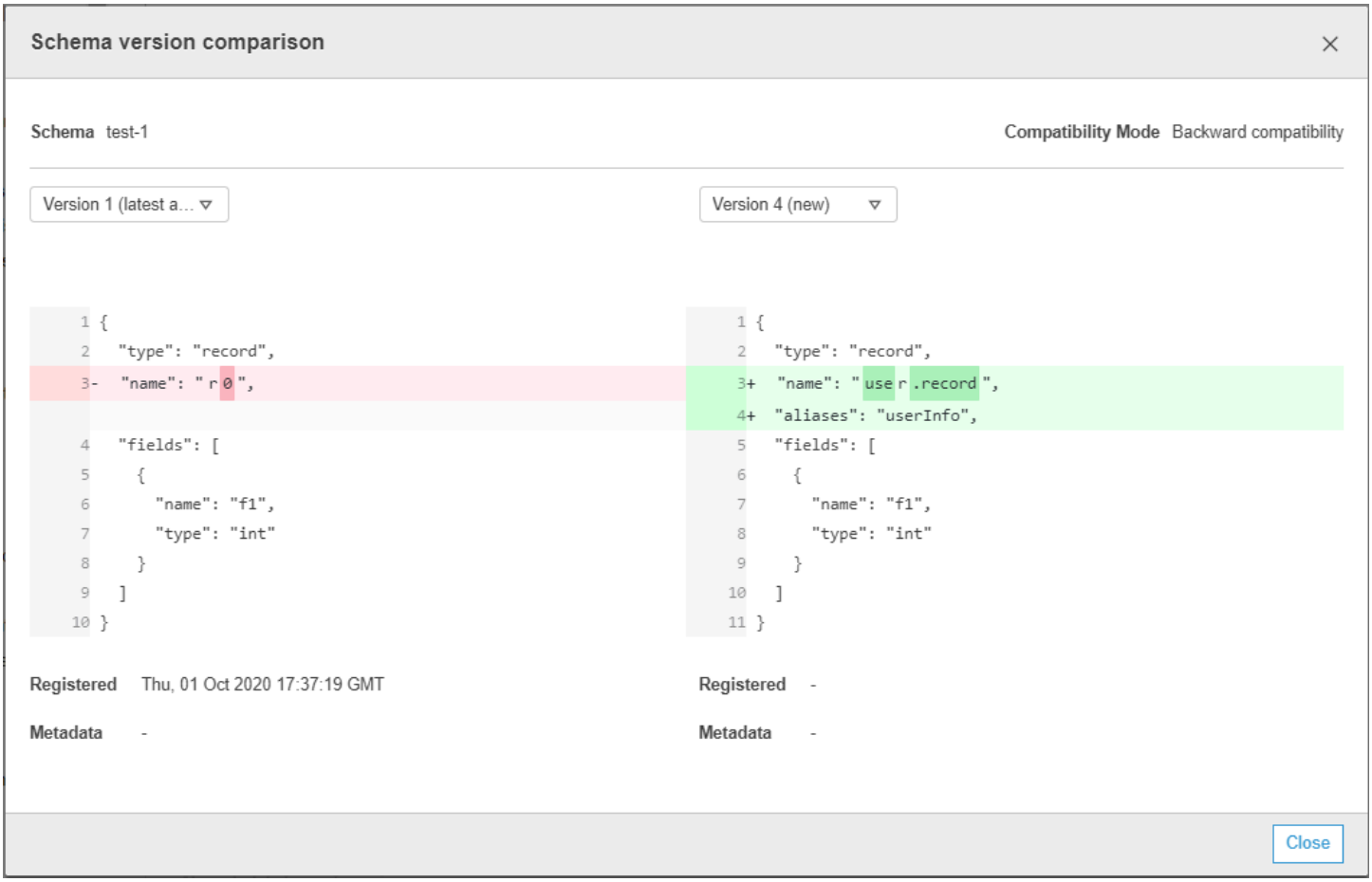
スキーマのバージョン比較の例。

[Compare with previous version] (過去のバージョンと比較) をクリックすると、以前のバージョンと新しいバージョンが一緒に表示されます。変更された情報は、次のように強調表示されます。

- 黄色: 変更された情報を示します。

- 緑: 最新バージョンで追加されたコンテンツを示します。
- 赤: 最新バージョンで削除されたコンテンツを示します。

より古いバージョンと比較することも可能です。



スキーマまたはレジストリの削除

スキーマ、スキーマバージョン、またはレジストリの削除は永続的な操作であり、元に戻すことはできません。

スキーマの削除

レジストリ内で使用する必要がなくなったスキーマは、AWS Management Console または DeleteSchema アクション (Python: delete_schema) API を使用して削除することができます。。

1 つ以上のスキーマを削除することは永続的なアクションであり、元に戻すことはできません。(1 つあるいは複数の) スキーマが不要になったことを確認します。

レジストリからスキーマを削除するには、SchemaId 構造体により対象のスキーマを特定しながら DeleteSchema アクション (Python: delete_schema) API を呼び出します。

例:

```
aws glue delete-schema --schema-id SchemaArn="arn:aws:glue:us-
east-2:901234567890:schema/registryName1/schemaname"
```

```
aws glue delete-schema --schema-id SchemaName="TestSchema6-
deleteschemabyname",RegistryName="default-registry"
```

AWS Glue コンソール

AWS Glue コンソールからスキーマを削除するには

1. AWS Management Consoleにサインインし、AWS Glue コンソール (https://console.aws.amazon.com/glue/) を開きます。
2. ナビゲーションペインの [Data catalog] (データカタログ) で、[Schema registries] (スキーマレジストリ) をクリックします。
3. レジストリのリストから、自分のスキーマを含むレジストリを選択します。
4. チェックボックスをクリックして、リストから 1 つ以上のスキーマを選択します。
5. [Action] (アクション) メニューで、[Delete schema] (スキーマの削除) をクリックします。
6. フィールドに「**Delete**」というテキストを入力して、削除を確定します。
7. [削除] を選択します。

指定した (1 つ以上の) スキーマがレジストリから削除されます。

スキーマバージョンの削除

スキーマはレジストリに蓄積されるので、不要なスキーマバージョンは、AWS Management Console または DeleteSchemaVersions アクション (Python: delete_schema_versions) API を使用して削除できます。1 つ以上のスキーマバージョンを削除することは永続的なアクションであり、元に戻すことはできません。そのスキーマバージョンが不要であることを確認します。

スキーマのバージョンを削除する場合は、以下の制約に注意してください。

- チェックポイントとなっているバージョンを削除することはできません。

- 25 を超えて連続するバージョンの範囲を削除することはできません。
- 最新のスキーマバージョンが保留状態にある場合は、削除は行えません。

SchemaId 構造体を使用してスキーマを指定し、削除するバージョンの範囲を Versions で指定します。バージョンまたはバージョンの範囲の指定の詳細については、「DeleteRegistry アクション (Python: delete_registry)」を参照してください。指定したスキーマバージョンがレジストリから削除されます。

この呼び出しの後に ListSchemaVersions アクション (Python: list_schema_versions) API を呼び出すと、削除されたバージョンのステータスが一覧表示されます。

例:

```
aws glue delete-schema-versions --schema-id
 SchemaName="TestSchema6",RegistryName="default-registry" --versions "1-1"
```

```
aws glue delete-schema-versions --schema-id SchemaArn="arn:aws:glue:us-
east-2:901234567890:schema/default-registry/TestSchema6-NON-Existent" --versions "1-1"
```

1. AWS Management Consoleにサインインし、AWS Glue コンソール (https://console.aws.amazon.com/glue/) を開きます。
2. ナビゲーションペインの [Data catalog] (データカタログ) で、[Schema registries] (スキーマレジストリ) をクリックします。
3. レジストリのリストから、自分のスキーマを含むレジストリを選択します。
4. チェックボックスをクリックして、リストから 1 つ以上のスキーマを選択します。
5. [Action] (アクション) メニューで、[Delete schema] (スキーマの削除) をクリックします。
6. フィールドに「**Delete**」というテキストを入力して、削除を確定します。
7. [削除] を選択します。

指定したスキーマバージョンがレジストリから削除されます。

レジストリの削除

レジストリに含まれるスキーマの整理の必要性がなくなった場合は、そのレジストリを削除することができます。これらのスキーマは、別のレジストリに再割り当てする必要があります。

1 つ以上のレジストリを削除することは永続的なアクションであり、元に戻すことはできません。(1 つもしくは複数の) レジストリが不要であることを確認します。

デフォルトのレジストリは、AWS CLI を使用して削除できます。

AWS Glue API

レジストリ全体を、登録されたスキーマとそのすべてのバージョンとともに削除するには、DeleteRegistry アクション (Python: delete_registry) API を呼び出します。`RegistryId` 構造体を使用し、レジストリを特定します。

例:

```
aws glue delete-registry --registry-id RegistryArn="arn:aws:glue:us-
east-2:901234567890:registry/registryName1"
```

```
aws glue delete-registry --registry-id RegistryName="TestRegistry-deletebyname"
```

削除オペレーションのステータスを取得するには、非同期呼び出し後に `GetRegistry` API を呼び出します。

AWS Glue コンソール

AWS Glue コンソールからレジストリの削除を行うには

1. AWS Management Consoleにサインインし、AWS Glue コンソール (https://console.aws.amazon.com/glue/) を開きます。
2. ナビゲーションペインの [Data catalog] (データカタログ) で、[Schema registries] (スキーマレジストリ) をクリックします。
3. チェックボックスをオンにして、リストからレジストリを選択します。
4. [Action] (アクション) メニューで、[Delete registry] (レジストリの削除) を選択します。
5. フィールドに「**`Delete`**」というテキストを入力して、削除を確定します。
6. [削除] を選択します。

選択したレジストリが AWS Glue から削除されます。

シリアライザ用の IAM の例

> **Note**
>
> AWS 管理ポリシーは、一般的ユースケースに必要なアクセス許可を付与します。管理ポリシーを使用してスキーマのレジストリを管理する方法については、「AWS Glue の AWS マネージド (事前定義) ポリシー」を参照してください。

シリアライザの場合、以下と同様の最小限のポリシーを作成して、特定のスキーマ定義のために schemaVersionId を検索できるようにします。レジストリ内のスキーマを読み取るには、そのレジストリに対する読み取り許可が必要であることに注意してください。読み取り可能なレジストリは、Resource 句を使用して制限できます。

コード例 13:

```
{
    "Sid" : "GetSchemaByDefinition",
    "Effect" : "Allow",
    "Action" :
 [
        "glue:GetSchemaByDefinition"
    ],
        "Resource" : ["arn:aws:glue:us-east-2:012345678:registry/registryname-1",
                      "arn:aws:glue:us-east-2:012345678:schema/registryname-1/
schemaname-1",
                      "arn:aws:glue:us-east-2:012345678:schema/registryname-1/
schemaname-2"
                     ]
}
```

さらに、以下の新しいメソッドを追加して、プロデューサに対し新しいスキーマとバージョンの作成を許可することもできます。レジストリ内でスキーマを追加/削除/拡大させるためには、そのレジストリを調査できることが必要です。調査することが可能なレジストリは、Resource 句を使用して制限できます。

コード例 14:

```
{
    "Sid" : "RegisterSchemaWithMetadata",
```

```
    "Effect" : "Allow",
    "Action" :
 [
        "glue:GetSchemaByDefinition",
        "glue:CreateSchema",
        "glue:RegisterSchemaVersion",
        "glue:PutSchemaVersionMetadata",
    ],
    "Resource" : ["arn:aws:glue:aws-region:123456789012:registry/registryname-1",
                  "arn:aws:glue:aws-region:123456789012:schema/registryname-1/
schemaname-1",
                  "arn:aws:glue:aws-region:123456789012:schema/registryname-1/
schemaname-2"
                ]
}
```

デシリアライザー用の IAM の例

デシリアライザ (コンシューマ側) の場合、以下のようなポリシーを作成する必要があります。これにより、非シリアル化のために Schema Registry からスキーマを取得することを、デシリアライザに対し許可します。レジストリ内のスキーマを取得するためには、そのレジストリを調査することが許可されている必要があります。

コード例 15:

```
{
    "Sid" : "GetSchemaVersion",
    "Effect" : "Allow",
    "Action" :
 [
        "glue:GetSchemaVersion"
    ],
    "Resource" : ["*"]
}
```

AWS PrivateLink を使用したプライベート接続

AWS Glue でインターフェイス VPC エンドポイントを定義しながら AWS PrivateLinkを使用すると、データのプロデューサの VPC を AWS Glue に接続することができます。VPC インターフェイスエンドポイントにより、AWS ネットワーク内全体で VPC と AWS Glue 間の通信を処理します。詳細については、「Using AWS Glue with VPC Endpoints」を参照してください。

Amazon CloudWatch メトリクスへのアクセス

Amazon CloudWatch メトリクスは、CloudWatch の無料利用枠の一部として利用できます。これらのメトリクスは CloudWatch コンソールでアクセスできます。APIレベルのメトリクスとしては、CreateSchema (Success および Latency)、GetSchemaByDefinition (Success および Latency)、GetSchemaVersion (Success および Latency)、RegisterSchemaVersion (Success および Latency)、PutSchemaVersionMetadata (Success および Latency) があります。リソースレベルのメトリクスには、Registry.ThrottledByLimit、SchemaVersion.ThrottledByLimit、SchemaVersion.Size があります。

スキーマレジストリの AWS CloudFormation テンプレート例

以下に、AWS CloudFormation で Schema Registry リソースを作成するためのテンプレート例を示します。アカウントにこのスタックを作成するには、上記のテンプレートを `SampleTemplate.yaml` ファイルにコピーした上で、次のコマンドを実行します。

```
aws cloudformation create-stack --stack-name ABCSchemaRegistryStack --template-body
 "'cat SampleTemplate.yaml'"
```

この例では、レジストリを作成するために `AWS::Glue::Registry` を、スキーマを作成するために `AWS::Glue::Schema` を、スキーマバージョンを作成するために `AWS::Glue::SchemaVersion` を使用し、`AWS::Glue::SchemaVersionMetadata` によりスキーマバージョンのメタデータを記述しています。

```
Description: "A sample CloudFormation template for creating Schema Registry resources."
Resources:
  ABCRegistry:
    Type: "AWS::Glue::Registry"
    Properties:
      Name: "ABCSchemaRegistry"
      Description: "ABC Corp. Schema Registry"
      Tags:
        - Key: "Project"
          Value: "Foo"
  ABCSchema:
    Type: "AWS::Glue::Schema"
    Properties:
      Registry:
        Arn: !Ref ABCRegistry
      Name: "TestSchema"
      Compatibility: "NONE"
```

```
      DataFormat: "AVRO"
      SchemaDefinition: >
        {"namespace":"foo.avro","type":"record","name":"user","fields":
[{"name":"name","type":"string"},{"name":"favorite_number","type":"int"}]}
      Tags:
        - Key: "Project"
          Value: "Foo"
  SecondSchemaVersion:
    Type: "AWS::Glue::SchemaVersion"
    Properties:
      Schema:
        SchemaArn: !Ref ABCSchema
      SchemaDefinition: >
        {"namespace":"foo.avro","type":"record","name":"user","fields":
[{"name":"status","type":"string", "default":"ON"}, {"name":"name","type":"string"},
{"name":"favorite_number","type":"int"}]}
  FirstSchemaVersionMetadata:
    Type: "AWS::Glue::SchemaVersionMetadata"
    Properties:
      SchemaVersionId: !GetAtt ABCSchema.InitialSchemaVersionId
      Key: "Application"
      Value: "Kinesis"
  SecondSchemaVersionMetadata:
    Type: "AWS::Glue::SchemaVersionMetadata"
    Properties:
      SchemaVersionId: !Ref SecondSchemaVersion
      Key: "Application"
      Value: "Kinesis"
```

AWS Glue Schema Registry との統合

これらのセクションでは、AWS Glue スキーマレジストリとの統合について説明します。このセクションの例では、AVRO データ形式のスキーマを使用します。JSON データ形式のスキーマなど、その他の例については、「AWS Glue Schema Registry open source repository」で、統合テストと ReadMe に関する情報をご覧ください。

トピック

- ユースケース: Schema Registry を Amazon MSK または Apache Kafka に接続する
- ユースケース: Amazon Kinesis Data Streams と AWS Glue Schema Registry との統合
- Amazon Managed Service for Apache Flink のユースケース

- ユースケース: AWS Lambda との統合
- ユースケース: AWS Glue Data Catalog
- ユースケース: AWS Glue ストリーミング
- ユースケース: Apache Kafka ストリーム

ユースケース: Schema Registry を Amazon MSK または Apache Kafka に接続する

Apache Kafka トピックにデータを書き込む場合には、以下の手順に従い作業を開始します。

1. Amazon Managed Streaming for Apache Kafka(Amazon MSK) または Apache Kafka のクラスターを作成し、少なくとも 1 つのトピックを含めます。Amazon MSK クラスターを作成する場合は、AWS Management Console を使用します。Amazon Managed Streaming for Apache Kafka デベロッパーガイドの「Getting Started Using Amazon MSK」にある手順に従います。
2. 上記の SerDe ライブラリのインストール ステップを実行します。
3. スキーマのレジストリ、スキーマ、またはスキーマバージョンを作成するには、このドキュメントにある スキーマレジストリの開始方法 セクションの手順に従います。
4. Amazon MSK または Apache Kafka のトピックとの間で、レコードの書き込みや読み取りを行うために、Schema Registry を使用してプロデューサとコンシューマを起動します。プロデューサとコンシューマのコード例は、Serdeライブラリの ReadMe ファイルから入手できます。プロデューサの Schema Registry ライブラリは、レコードを自動的にシリアル化し、スキーマバージョン ID でそのレコードを修飾します。
5. このレコードにスキーマが入力済みの場合、または自動登録が有効になっている場合には、スキーマが Schema Registry に登録されます。
6. AWS Glue Schema Registry ライブラリを使用して、Amazon MSK または Apache Kafka のトピックからスキーマの読み取りを行うコンシューマは、自動的に Schema Registry からスキーマを検索します。

ユースケース: Amazon Kinesis Data Streams と AWS Glue Schema Registry との統合

この統合には、既存の Amazon Kinesis データストリーム が必要です。詳細については、Amazon Kinesis Data Streams デベロッパーガイドの「Getting Started with Amazon Kinesis Data Streams」を参照してください。

Kinesis データストリームでは、データの操作用に以下の 2 つの方法があります。

- Java の Kinesis Producer Library (KPL) および Kinesis Client Library (KCL) ライブラリを使用します。多言語サポートは提供されていません。
- AWS SDK for Java に用意されている `PutRecords`、`PutRecord`、および `GetRecords` Kinesis Data Streams API を使用します。

現在、KPL/KCL ライブラリを使用中であれば、そのメソッドを引き続き使用することをお勧めします。ここでの例に示すように、Schema Registry が統合済みの、更新された KCL および KPL バージョンを使用できます。それ以外で、KDS API を直接使用している場合には、サンプルコードを通じて AWS Glue Schema Registry を利用します。

Schema Registry との統合は、KPL v0.14.2 以降と KCL v2.3 以降でのみ使用できます。JSON データ形式による Schema Registry との統合は、KPL v0.14.8 以降および KCL v2.3.6 以降で使用できます。

Kinesis SDK V2 を使用したデータの操作

このセクションでは、Kinesis SDK V2 による Kinesis の操作について説明します。

```
// Example JSON Record, you can construct a AVRO record also
private static final JsonDataWithSchema record =
 JsonDataWithSchema.builder(schemaString, payloadString);
private static final DataFormat dataFormat = DataFormat.JSON;

//Configurations for Schema Registry
GlueSchemaRegistryConfiguration gsrConfig = new GlueSchemaRegistryConfiguration("us-
east-1");

GlueSchemaRegistrySerializer glueSchemaRegistrySerializer =
        new GlueSchemaRegistrySerializerImpl(awsCredentialsProvider, gsrConfig);
GlueSchemaRegistryDataFormatSerializer dataFormatSerializer =
        new GlueSchemaRegistrySerializerFactory().getInstance(dataFormat, gsrConfig);

Schema gsrSchema =
        new Schema(dataFormatSerializer.getSchemaDefinition(record), dataFormat.name(),
 "MySchema");

byte[] serializedBytes = dataFormatSerializer.serialize(record);

byte[] gsrEncodedBytes = glueSchemaRegistrySerializer.encode(streamName, gsrSchema,
 serializedBytes);
```

```
PutRecordRequest putRecordRequest = PutRecordRequest.builder()
        .streamName(streamName)
        .partitionKey("partitionKey")
        .data(SdkBytes.fromByteArray(gsrEncodedBytes))
        .build();
shardId = kinesisClient.putRecord(putRecordRequest)
        .get()
        .shardId();

GlueSchemaRegistryDeserializer glueSchemaRegistryDeserializer = new
 GlueSchemaRegistryDeserializerImpl(awsCredentialsProvider, gsrConfig);

GlueSchemaRegistryDataFormatDeserializer gsrDataFormatDeserializer =
        glueSchemaRegistryDeserializerFactory.getInstance(dataFormat, gsrConfig);

GetShardIteratorRequest getShardIteratorRequest = GetShardIteratorRequest.builder()
        .streamName(streamName)
        .shardId(shardId)
        .shardIteratorType(ShardIteratorType.TRIM_HORIZON)
        .build();

String shardIterator = kinesisClient.getShardIterator(getShardIteratorRequest)
        .get()
        .shardIterator();

GetRecordsRequest getRecordRequest = GetRecordsRequest.builder()
        .shardIterator(shardIterator)
        .build();
GetRecordsResponse recordsResponse = kinesisClient.getRecords(getRecordRequest)
        .get();

List<Object> consumerRecords = new ArrayList<>();
List<Record> recordsFromKinesis = recordsResponse.records();

for (int i = 0; i < recordsFromKinesis.size(); i++) {
    byte[] consumedBytes = recordsFromKinesis.get(i)
             .data()
             .asByteArray();

    Schema gsrSchema = glueSchemaRegistryDeserializer.getSchema(consumedBytes);
    Object decodedRecord =
 gsrDataFormatDeserializer.deserialize(ByteBuffer.wrap(consumedBytes),

 gsrSchema.getSchemaDefinition());
```

```
    consumerRecords.add(decodedRecord);
}
```

KPL/KCL ライブラリを使用したデータの操作

このセクションでは、KPL/KCL ライブラリを使用しての Kinesis Data Streams と Schema Registry の統合について説明します。KPL/KCL の使用方法については、Amazon Kinesis Data Streams デベロッパーガイドの「Developing Producers Using the Amazon Kinesis Producer Library」を参照してください。

KPL で Schema Registry を設定する

1. AWS Glue Schema Registryで作成したデータ、データ形式、スキーマ名のスキーマ定義を行います。
2. 必要に応じて、GlueSchemaRegistryConfiguration オブジェクトも構成します。
3. addUserRecord API にスキーマオブジェクトを渡します。

```
private static final String SCHEMA_DEFINITION = "{"namespace": "example.avro",\n"
+ " "type": "record",\n"
+ " "name": "User",\n"
+ " "fields": [\n"
+ " {"name": "name", "type": "string"},\n"
+ " {"name": "favorite_number", "type": ["int", "null"]},\n"
+ " {"name": "favorite_color", "type": ["string", "null"]}\n"
+ " ]\n"
+ "}";

KinesisProducerConfiguration config = new KinesisProducerConfiguration();
config.setRegion("us-west-1")

//[Optional] configuration for Schema Registry.

GlueSchemaRegistryConfiguration schemaRegistryConfig =
new GlueSchemaRegistryConfiguration("us-west-1");

schemaRegistryConfig.setCompression(true);

config.setGlueSchemaRegistryConfiguration(schemaRegistryConfig);

///Optional configuration ends.

final KinesisProducer producer =
```

```
      new KinesisProducer(config);

final ByteBuffer data = getDataToSend();

com.amazonaws.services.schemaregistry.common.Schema gsrSchema =
    new Schema(SCHEMA_DEFINITION, DataFormat.AVRO.toString(), "demoSchema");

ListenableFuture<UserRecordResult> f = producer.addUserRecord(
config.getStreamName(), TIMESTAMP, Utils.randomExplicitHashKey(), data, gsrSchema);

private static ByteBuffer getDataToSend() {
      org.apache.avro.Schema avroSchema =
        new org.apache.avro.Schema.Parser().parse(SCHEMA_DEFINITION);

      GenericRecord user = new GenericData.Record(avroSchema);
      user.put("name", "Emily");
      user.put("favorite_number", 32);
      user.put("favorite_color", "green");

      ByteArrayOutputStream outBytes = new ByteArrayOutputStream();
      Encoder encoder = EncoderFactory.get().directBinaryEncoder(outBytes, null);
      new GenericDatumWriter<>(avroSchema).write(user, encoder);
      encoder.flush();
      return ByteBuffer.wrap(outBytes.toByteArray());
 }
```

Kinesis Client Library のセットアップ

Kinesis Client Library コンシューマーを、Java により構築します。詳細については、Amazon Kinesis Data Streams デベロッパーガイドの「Developing a Kinesis Client Library Consumer in Java」を参照してください。

1. `GlueSchemaRegistryConfiguration` オブジェクトを渡すことで `GlueSchemaRegistryDeserializer` インスタンスを作成します。
2. `GlueSchemaRegistryDeserializer` を `retrievalConfig.glueSchemaRegistryDeserializer` に渡します。
3. `kinesisClientRecord.getSchema()` を呼び出して、受信メッセージのスキーマにアクセスします。

```
GlueSchemaRegistryConfiguration schemaRegistryConfig =
    new GlueSchemaRegistryConfiguration(this.region.toString());
```

```
GlueSchemaRegistryDeserializer glueSchemaRegistryDeserializer =
   new
GlueSchemaRegistryDeserializerImpl(DefaultCredentialsProvider.builder().build(),
schemaRegistryConfig);

RetrievalConfig retrievalConfig =
configsBuilder.retrievalConfig().retrievalSpecificConfig(new
PollingConfig(streamName, kinesisClient));
retrievalConfig.glueSchemaRegistryDeserializer(glueSchemaRegistryDeserializer);

 Scheduler scheduler = new Scheduler(
           configsBuilder.checkpointConfig(),
           configsBuilder.coordinatorConfig(),
           configsBuilder.leaseManagementConfig(),
           configsBuilder.lifecycleConfig(),
           configsBuilder.metricsConfig(),
           configsBuilder.processorConfig(),
           retrievalConfig
       );

public void processRecords(ProcessRecordsInput processRecordsInput) {
           MDC.put(SHARD_ID_MDC_KEY, shardId);
           try {
               log.info("Processing {} record(s)",
               processRecordsInput.records().size());
               processRecordsInput.records()
               .forEach(
                   r ->
                       log.info("Processed record pk: {} -- Seq: {} : data {} with
schema: {}",
                       r.partitionKey(),
r.sequenceNumber(), recordToAvroObj(r).toString(), r.getSchema()));
           } catch (Throwable t) {
               log.error("Caught throwable while processing records. Aborting.");
               Runtime.getRuntime().halt(1);
           } finally {
               MDC.remove(SHARD_ID_MDC_KEY);
           }
}

private GenericRecord recordToAvroObj(KinesisClientRecord r) {
   byte[] data = new byte[r.data().remaining()];
   r.data().get(data, 0, data.length);
```

```
    org.apache.avro.Schema schema = new
 org.apache.avro.Schema.Parser().parse(r.schema().getSchemaDefinition());
    DatumReader datumReader = new GenericDatumReader<>(schema);

    BinaryDecoder binaryDecoder = DecoderFactory.get().binaryDecoder(data, 0,
 data.length, null);
    return (GenericRecord) datumReader.read(null, binaryDecoder);
 }
```

Kinesis Data Streams API を使用したデータの操作

このセクションでは、Kinesis Data Streams API を使用しての、Kinesis Data Streams と Schema Registry の統合について説明します。

1. Maven の以下の依存関係を更新します。

```
<dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>com.amazonaws</groupId>
                <artifactId>aws-java-sdk-bom</artifactId>
                <version>1.11.884</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
        </dependencies>
    </dependencyManagement>

    <dependencies>
        <dependency>
            <groupId>com.amazonaws</groupId>
            <artifactId>aws-java-sdk-kinesis</artifactId>
        </dependency>

        <dependency>
            <groupId>software.amazon.glue</groupId>
            <artifactId>schema-registry-serde</artifactId>
            <version>1.1.5</version>
        </dependency>

        <dependency>
            <groupId>com.fasterxml.jackson.dataformat</groupId>
```

```
            <artifactId>jackson-dataformat-cbor</artifactId>
            <version>2.11.3</version>
        </dependency>
    </dependencies>
```

2. PutRecords または Kinesis Data Streams の PutRecord API を使用しながら、プロデューサ内にスキーマヘッダー情報を追加します。

```
//The following lines add a Schema Header to the record
        com.amazonaws.services.schemaregistry.common.Schema awsSchema =
            new com.amazonaws.services.schemaregistry.common.Schema(schemaDefinition,
 DataFormat.AVRO.name(),
                schemaName);
        GlueSchemaRegistrySerializerImpl glueSchemaRegistrySerializer =
            new
 GlueSchemaRegistrySerializerImpl(DefaultCredentialsProvider.builder().build(), new
 GlueSchemaRegistryConfiguration(getConfigs()));
        byte[] recordWithSchemaHeader =
            glueSchemaRegistrySerializer.encode(streamName, awsSchema,
 recordAsBytes);
```

3. プロデューサ内で PutRecords または PutRecord API を使用して、レコードをデータストリームに配置します。

4. コンシューマ内で、ヘッダーからスキーマレコードを削除し、Avro スキーマレコードをシリアル化します。

```
//The following lines remove Schema Header from record
        GlueSchemaRegistryDeserializerImpl glueSchemaRegistryDeserializer =
            new
 GlueSchemaRegistryDeserializerImpl(DefaultCredentialsProvider.builder().build(),
 getConfigs());
        byte[] recordWithSchemaHeaderBytes = new
 byte[recordWithSchemaHeader.remaining()];
        recordWithSchemaHeader.get(recordWithSchemaHeaderBytes, 0,
 recordWithSchemaHeaderBytes.length);
        com.amazonaws.services.schemaregistry.common.Schema awsSchema =
            glueSchemaRegistryDeserializer.getSchema(recordWithSchemaHeaderBytes);
        byte[] record =
 glueSchemaRegistryDeserializer.getData(recordWithSchemaHeaderBytes);

        //The following lines serialize an AVRO schema record
        if (DataFormat.AVRO.name().equals(awsSchema.getDataFormat())) {
```

```
            Schema avroSchema = new
  org.apache.avro.Schema.Parser().parse(awsSchema.getSchemaDefinition());
            Object genericRecord = convertBytesToRecord(avroSchema, record);
            System.out.println(genericRecord);
        }
```

Kinesis Data Streams API を使用したデータの操作

以下に、PutRecords および GetRecords API を使用するコード例を示します。

```
//Full sample code
import
 com.amazonaws.services.schemaregistry.deserializers.GlueSchemaRegistryDeserializerImpl;
import
 com.amazonaws.services.schemaregistry.serializers.GlueSchemaRegistrySerializerImpl;
import com.amazonaws.services.schemaregistry.utils.AVROUtils;
import com.amazonaws.services.schemaregistry.utils.AWSSchemaRegistryConstants;
import org.apache.avro.Schema;
import org.apache.avro.generic.GenericData;
import org.apache.avro.generic.GenericDatumReader;
import org.apache.avro.generic.GenericDatumWriter;
import org.apache.avro.generic.GenericRecord;
import org.apache.avro.io.Decoder;
import org.apache.avro.io.DecoderFactory;
import org.apache.avro.io.Encoder;
import org.apache.avro.io.EncoderFactory;
import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.services.glue.model.DataFormat;

import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.IOException;
import java.nio.ByteBuffer;
import java.util.Collections;
import java.util.HashMap;
import java.util.Map;


public class PutAndGetExampleWithEncodedData {
    static final String regionName = "us-east-2";
    static final String streamName = "testStream1";
    static final String schemaName = "User-Topic";
    static final String AVRO_USER_SCHEMA_FILE = "src/main/resources/user.avsc";
```

```
    KinesisApi kinesisApi = new KinesisApi();

    void runSampleForPutRecord() throws IOException {
        Object testRecord = getTestRecord();
        byte[] recordAsBytes = convertRecordToBytes(testRecord);
        String schemaDefinition =
AVROUtils.getInstance().getSchemaDefinition(testRecord);

        //The following lines add a Schema Header to a record
        com.amazonaws.services.schemaregistry.common.Schema awsSchema =
            new com.amazonaws.services.schemaregistry.common.Schema(schemaDefinition,
DataFormat.AVRO.name(),
                schemaName);
        GlueSchemaRegistrySerializerImpl glueSchemaRegistrySerializer =
            new
GlueSchemaRegistrySerializerImpl(DefaultCredentialsProvider.builder().build(), new
GlueSchemaRegistryConfiguration(regionName));
        byte[] recordWithSchemaHeader =
            glueSchemaRegistrySerializer.encode(streamName, awsSchema, recordAsBytes);

        //Use PutRecords api to pass a list of records
        kinesisApi.putRecords(Collections.singletonList(recordWithSchemaHeader),
streamName, regionName);

        //OR
        //Use PutRecord api to pass single record
        //kinesisApi.putRecord(recordWithSchemaHeader, streamName, regionName);
    }

    byte[] runSampleForGetRecord() throws IOException {
        ByteBuffer recordWithSchemaHeader = kinesisApi.getRecords(streamName,
regionName);

        //The following lines remove the schema registry header
        GlueSchemaRegistryDeserializerImpl glueSchemaRegistryDeserializer =
            new
GlueSchemaRegistryDeserializerImpl(DefaultCredentialsProvider.builder().build(), new
GlueSchemaRegistryConfiguration(regionName));
        byte[] recordWithSchemaHeaderBytes = new
byte[recordWithSchemaHeader.remaining()];
        recordWithSchemaHeader.get(recordWithSchemaHeaderBytes, 0,
recordWithSchemaHeaderBytes.length);

        com.amazonaws.services.schemaregistry.common.Schema awsSchema =
```

```
            glueSchemaRegistryDeserializer.getSchema(recordWithSchemaHeaderBytes);

        byte[] record =
glueSchemaRegistryDeserializer.getData(recordWithSchemaHeaderBytes);

        //The following lines serialize an AVRO schema record
        if (DataFormat.AVRO.name().equals(awsSchema.getDataFormat())) {
            Schema avroSchema = new
org.apache.avro.Schema.Parser().parse(awsSchema.getSchemaDefinition());
            Object genericRecord = convertBytesToRecord(avroSchema, record);
            System.out.println(genericRecord);
        }

        return record;
    }

    private byte[] convertRecordToBytes(final Object record) throws IOException {
        ByteArrayOutputStream recordAsBytes = new ByteArrayOutputStream();
        Encoder encoder = EncoderFactory.get().directBinaryEncoder(recordAsBytes,
null);
        GenericDatumWriter datumWriter = new
GenericDatumWriter<>(AVROUtils.getInstance().getSchema(record));
        datumWriter.write(record, encoder);
        encoder.flush();
        return recordAsBytes.toByteArray();
    }

    private GenericRecord convertBytesToRecord(Schema avroSchema, byte[] record) throws
IOException {
        final GenericDatumReader<GenericRecord> datumReader = new
GenericDatumReader<>(avroSchema);
        Decoder decoder = DecoderFactory.get().binaryDecoder(record, null);
        GenericRecord genericRecord = datumReader.read(null, decoder);
        return genericRecord;
    }

    private Map<String, String> getMetadata() {
        Map<String, String> metadata = new HashMap<>();
        metadata.put("event-source-1", "topic1");
        metadata.put("event-source-2", "topic2");
        metadata.put("event-source-3", "topic3");
        metadata.put("event-source-4", "topic4");
        metadata.put("event-source-5", "topic5");
        return metadata;
```

```
    }

    private GlueSchemaRegistryConfiguration getConfigs() {
        GlueSchemaRegistryConfiguration configs = new
 GlueSchemaRegistryConfiguration(regionName);
        configs.setSchemaName(schemaName);
        configs.setAutoRegistration(true);
        configs.setMetadata(getMetadata());
        return configs;
    }

    private Object getTestRecord() throws IOException {
        GenericRecord genericRecord;
        Schema.Parser parser = new Schema.Parser();
        Schema avroSchema = parser.parse(new File(AVRO_USER_SCHEMA_FILE));

        genericRecord = new GenericData.Record(avroSchema);
        genericRecord.put("name", "testName");
        genericRecord.put("favorite_number", 99);
        genericRecord.put("favorite_color", "red");

        return genericRecord;
    }
}
```

Amazon Managed Service for Apache Flink のユースケース

Apache Flinkは、無制限および制限付きのデータストリームに対するステートフルな計算に広く使用されている、オープンソースフレームワークの分散処理エンジンです。Amazon Managed Service for Apache Flink は、ストリーミングデータを処理するため、Apache Flink アプリケーションを構築して管理できるようにする完全マネージド型の AWS サービスです。

オープンソースの Apache Flink では、多数のソースとシンクを利用できます。例えば、事前定義済みのデータソースには、ファイル、ディレクトリ、およびソケットからの読み込みや、コレクションとイテレータからのデータの取り込みなどが含まれています。Apache Flink DataStream Connector は、Apache Flinkが、Apache Kafka や Kinesis などの各種サードパーティー製システムと、ソースおよび/またはシンクとしてインターフェースするためのコードを提供します。

詳細については、Amazon Kinesis Data Analytics デベロッパーガイドを参照してください。

Apache Flink Kafka Connector

Apache Flinkは、Kafka のトピックに対するデータの読み取りおよび書き込みを、正確に一度で行えるようにするための、Apache Kafka データストリームのコネクタを提供します。Flink の Kafka コンシューマ `FlinkKafkaConsumer` では、1 つ以上の Kafka トピックから読み取りを行うアクセスが提供されます。Apache Flink の Kafka プロデューサ `FlinkKafkaProducer` では、1 つ以上の Kafka トピックに対しレコードのストリームを書き込むことができます。詳細については、「Apache Kafka Connector」を参照してください。

Apache Flink Kinesis Streams Connector

Kinesis データストリームのコネクタは、Amazon Kinesis Data Streams へのアクセスを提供します。並列ストリーミングデータソース `FlinkKinesisConsumer` は、同じAWS のサービスリージョン内で複数の Kinesis ストリームにサブスクライブされ、ジョブの実行中にストリームの再シャーディングを透過的に (確実に 1 回で) 処理できます。コンシューマーの各サブタスクが、複数の Kinesis シャードからのデータレコードの取得を受け持ちます。各サブタスクによって取得されるシャードの数は、Kinesis によってシャードが閉じられ、また作成されるたびに変化します。`FlinkKinesisProducer` は Kinesis Producer Library (KPL) を使用して、Kinesis ストリーム内に Apache Flink ストリームからのデータを配置します。詳細については、「Amazon Kinesis Streams Connector」を参照してください。

詳細については、AWS Glue のGitHub リポジトリを参照してください。

Apache Flink との統合

Schema Registry で提供されている SerDes ライブラリは、Apache Flink と統合されています。Apache Flink を使用するには、`SerializationSchema` および `DeserializationSchema` のインターフェイス (`GlueSchemaRegistryAvroSerializationSchema` および `GlueSchemaRegistryAvroDeserializationSchema`) を実装する必要があります。これらは、Apache Flink コネクタにプラグインして使用します。

Apache Flink アプリケーションへの AWS Glue Schema Registry の依存関係の追加

Apache Flink アプリケーション内で AWS Glue Schema Registry との統合の依存関係をセットアップするには

1. `pom.xml` ファイルに依存関係を追加します。

```
<dependency>
```

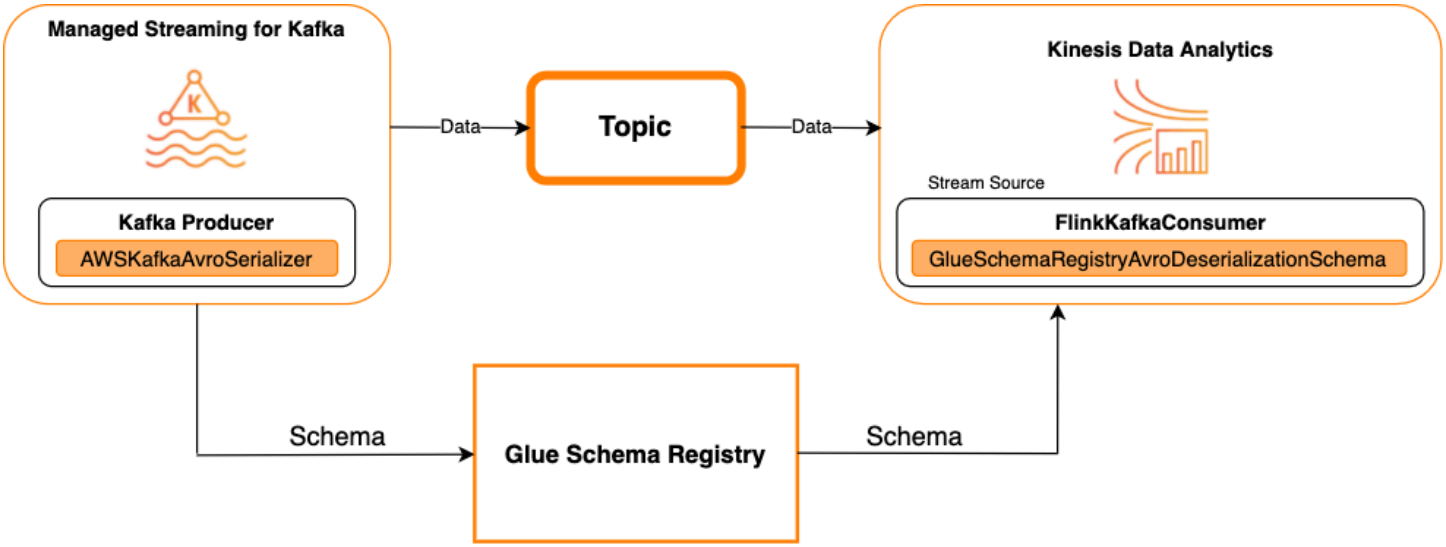
```
    <groupId>software.amazon.glue</groupId>
    <artifactId>schema-registry-flink-serde</artifactId>
    <version>1.0.0</version>
</dependency>
```

Kafka または Amazon MSK を Apache Flink と統合する

Kafka をソースまたはシンクとしながら、Apache Flink 対応の Managed Service for Apache Flink を使用できます。

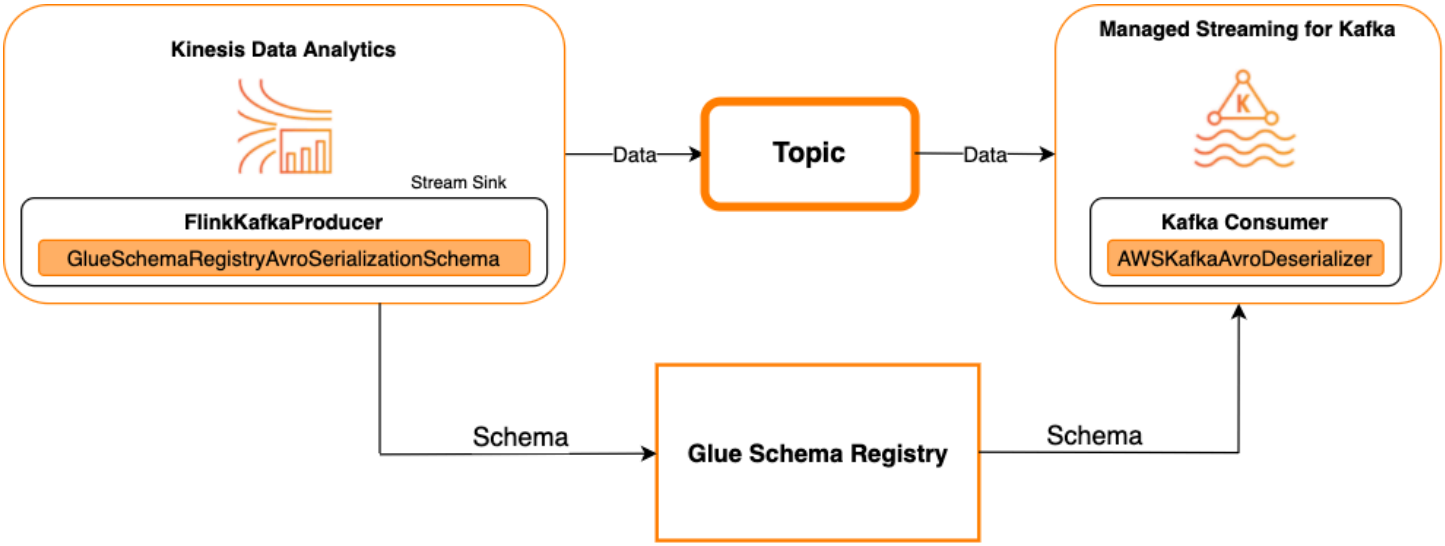
Kafka をソースとする場合

次の図は、Kafka をソースとしながら、Kinesis Data Streams と Apache Flink 対応の Managed Service for Apache Flink を統合した様子です。



Kafka をシンクとする場合

次の図は、Kafka をシンクとしながら、Kinesis Data Streams と Apache Flink 対応の Managed Service for Apache Flink を統合した様子です。



Kafka をソースまたはシンクとしながら、Kafka (または Amazon MSK) を Apache Flink 対応の Managed Service for Apache Flink と統合するには、以下のコード変更を行います。太字で示されたコードブロックを、類似するセクション内の対応するコードにそれぞれ追加します。

Kafka をソースとする場合は、デシリアライザ用コード (ブロック 2) を使用します。Kafka をシンクとする場合は、シリアライザコード (ブロック 3) を使用します。

```
StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();

String topic = "topic";
Properties properties = new Properties();
properties.setProperty("bootstrap.servers", "localhost:9092");
properties.setProperty("group.id", "test");

// block 1
Map<String, Object> configs = new HashMap<>();
configs.put(AWSSchemaRegistryConstants.AWS_REGION, "aws-region");
configs.put(AWSSchemaRegistryConstants.SCHEMA_AUTO_REGISTRATION_SETTING, true);
configs.put(AWSSchemaRegistryConstants.AVRO_RECORD_TYPE,
 AvroRecordType.GENERIC_RECORD.getName());

FlinkKafkaConsumer<GenericRecord> consumer = new FlinkKafkaConsumer<>(
    topic,
    // block 2
    GlueSchemaRegistryAvroDeserializationSchema.forGeneric(schema, configs),
    properties);

FlinkKafkaProducer<GenericRecord> producer = new FlinkKafkaProducer<>(
```

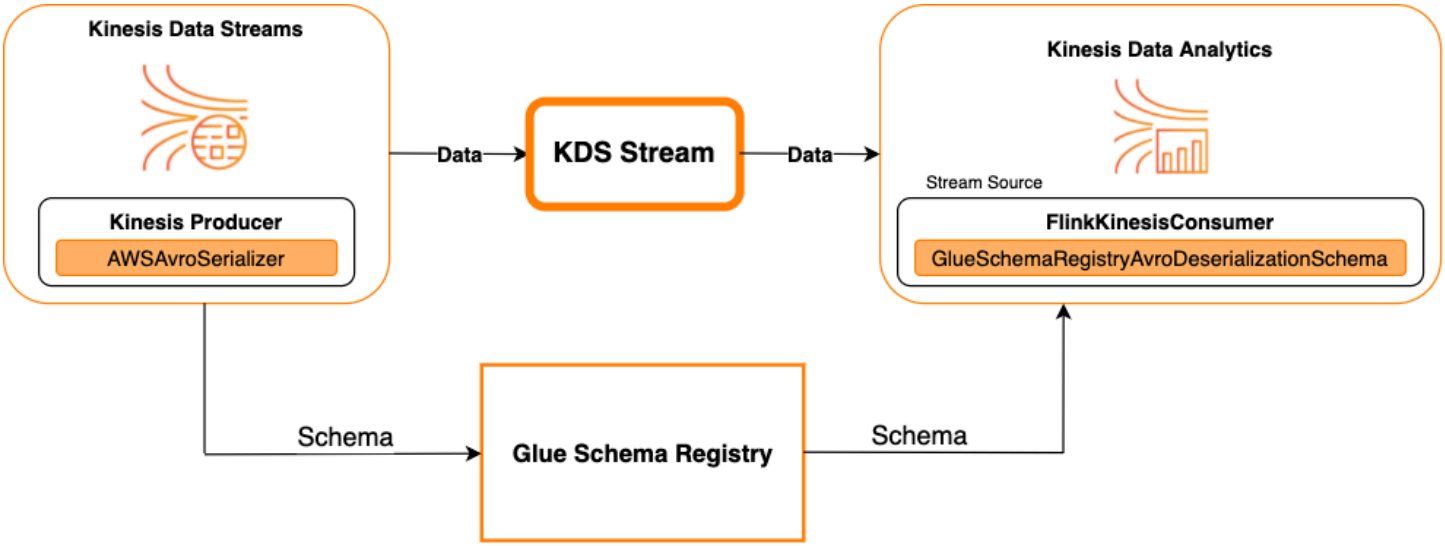
```
    topic,
    // block 3
    GlueSchemaRegistryAvroSerializationSchema.forGeneric(schema, topic, configs),
    properties);

DataStream<GenericRecord> stream = env.addSource(consumer);
stream.addSink(producer);
env.execute();
```

Kinesis Data Streams と Apache Flink との統合

Kinesis Data Streams をソースまたはシンクとしながら、Apache Flink 対応の Managed Service for Apache Flink を使用できます。

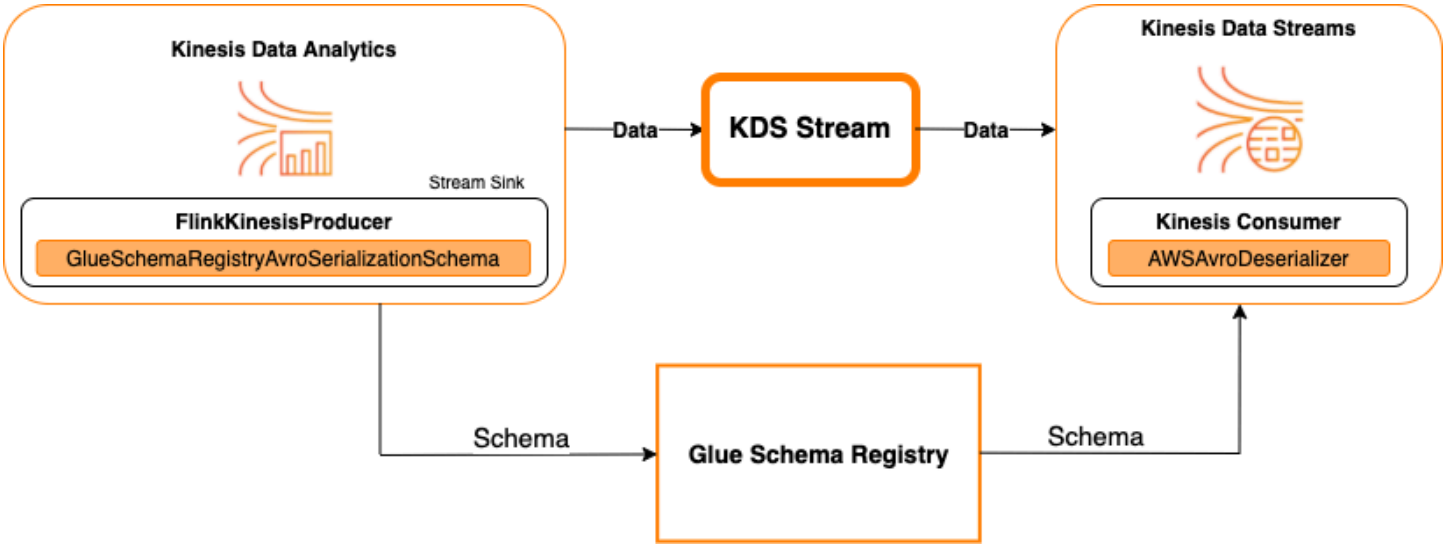
Kinesis Data Streams をソースとする場合

次の図は、Kinesis Data Streams をソースとしながら、Kinesis Data Streams と Apache Flink 対応の Managed Service for Apache Flink を統合した様子です。



Kinesis Data Streams をシンクとする場合

次の図は、Kinesis Data Streams をシンクとしながら、Kinesis Data Streams と Apache Flink 対応の Managed Service for Apache Flink を統合した様子です。



Kinesis Data Streams をソースまたはシンクとしながら、Kinesis Data Streams と Apache Flink 対応の Managed Service for Apache Flink を統合するには、以下のコード変更を行います。太字で示されたコードブロックを、類似するセクション内の対応するコードにそれぞれ追加します。

Kinesis Data Streams をソースとする場合は、デシリアライザコード (ブロック 2) を使用します。Kinesis Data Streams をシンクとする場合は、シリアライザコード (ブロック 3) を使用します。

```
StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();

String streamName = "stream";
Properties consumerConfig = new Properties();
consumerConfig.put(AWSConfigConstants.AWS_REGION, "aws-region");
consumerConfig.put(AWSConfigConstants.AWS_ACCESS_KEY_ID, "aws_access_key_id");
consumerConfig.put(AWSConfigConstants.AWS_SECRET_ACCESS_KEY, "aws_secret_access_key");
consumerConfig.put(ConsumerConfigConstants.STREAM_INITIAL_POSITION, "LATEST");

// block 1
Map<String, Object> configs = new HashMap<>();
configs.put(AWSSchemaRegistryConstants.AWS_REGION, "aws-region");
configs.put(AWSSchemaRegistryConstants.SCHEMA_AUTO_REGISTRATION_SETTING, true);
configs.put(AWSSchemaRegistryConstants.AVRO_RECORD_TYPE,
 AvroRecordType.GENERIC_RECORD.getName());

FlinkKinesisConsumer<GenericRecord> consumer = new FlinkKinesisConsumer<>(
    streamName,
    // block 2
    GlueSchemaRegistryAvroDeserializationSchema.forGeneric(schema, configs),
    properties);
```

```
FlinkKinesisProducer<GenericRecord> producer = new FlinkKinesisProducer<>(
    // block 3
    GlueSchemaRegistryAvroSerializationSchema.forGeneric(schema, topic, configs),
    properties);
producer.setDefaultStream(streamName);
producer.setDefaultPartition("0");

DataStream<GenericRecord> stream = env.addSource(consumer);
stream.addSink(producer);
env.execute();
```

ユースケース: AWS Lambda との統合

AWS Lambda 関数を Apache Kafka/Amazon MSK のコンシューマーとして使用し、Avro でエンコードされたメッセージを AWS Glue Schema Registry により非シリアル化するには、MSK ラボのページをご覧ください。

ユースケース: AWS Glue Data Catalog

AWS Glue テーブルは、手動による指定、または AWS Glue Schema Registry への参照によって指定できる、スキーマをサポートしています。AWS Glue テーブルまたは Data Catalog のパーティションを作成または更新する際に、オプションで Schema Registry に格納されているスキーマを使用できるように、Schema Registry には Data Catalog が統合されています。Schema Registry のスキーマ定義を特定するには、少なくとも、対象となるスキーマの ARN を知る必要があります。スキーマ定義を含むスキーマのスキーマバージョンは、UUID またはバージョン番号により参照が可能です。スキーマバージョンの中でも「最新」バージョンについては、バージョン番号または UUID を把握しなくても常に参照することができます。

`CreateTable` または `UpdateTable` オペレーションの呼び出し時は、Schema Registry 内の既存のスキーマに対する `TableInput` を指定するために `SchemaReference` 構造体 (`StorageDescriptor` を含む) を渡します。同様に、`GetTable` または `GetPartition` API を呼び出す場合は、そのレスポンスにスキーマと `SchemaReference` が含まれます。スキーマ参照を使用してテーブルまたはパーティションが作成されると、Data Catalog はこのスキーマ参照のスキーマ取得を試みます。Schema Registry 内にスキーマが見つからない場合は、`GetTable` レスポンスで空のスキーマを返します。それ以外では、このレスポンスにスキーマとスキーマ参照の両方が出力されます。

また、AWS Glue コンソールからアクションを実行することも可能です。

これらのオペレーションを実行し、スキーマ情報を作成、更新、表示するには、呼び出しユーザーに、GetSchemaVersion API へのアクセス権限を付与する、IAM ロールを付与する必要があります。

テーブルの追加またはテーブルのスキーマの更新

既存のスキーマから新しいテーブルを追加すると、そのテーブルは特定のスキーマバージョンにバインドされます。新しいスキーマバージョンの登録が完了すると、このテーブル定義が、AWS Glue コンソールの [View table] (テーブルの表示) ページ、もしくは UpdateTable アクション (Python: update_table) API を使用して更新できるようになります。

既存のスキーマからのテーブルの追加

AWS Glue コンソールまたは CreateTable API を使用して、レジストリ内のスキーマバージョンから AWS Glue を作成できます。

AWS Glue API

CreateTable API を呼び出す際に、(StorageDescriptor にSchemaReference が指定されている) TableInput を、スキーマレジストリの既存のスキーマに追加します。

AWS Glue コンソール

AWS Glue コンソールを使用してテーブルを作成するには

1. AWS Management Consoleにサインインし、AWS Glue コンソール (https://console.aws.amazon.com/glue/) を開きます。
2. ナビゲーションペインの [Data catalog] (データカタログ) で、[Tables] (テーブル) をクリックします。
3. [Add Tables] (テーブルの追加) メニューで、[Add table from existing schema] (既存のスキーマからテーブルを追加する) をクリックします。
4. テーブルのプロパティとデータストアを、AWS Glue デベロッパーガイド に沿って設定します。
5. [Choose a Glue schema] (Glue スキーマの選択) ページで、スキーマが置かれている [Registry] (レジストリ) を選択します。
6. [Schema name] (スキーマ名) をクリックし、適用するスキーマの [Version] (バージョン) を選択します。
7. スキーマのプレビューを確認し、[Next] (次へ) をクリックします。

8. テーブルを確認し、作成します。

作成したテーブルに適用されたスキーマとバージョンは、テーブルの一覧内で [Glue schema] (Glue スキーマ) 列に表示されます。テーブルを表示すると、さらに詳細を確認できます。

テーブルのスキーマの更新

新しいスキーマバージョンが使用可能になったら、テーブルのスキーマを UpdateTable アクション (Python: update_table) API または AWS Glue コンソールにより更新することができます

> **Important**
>
> 手動で指定された AWS Glue スキーマを含む既存のテーブル用にスキーマを更新する場合、Schema Registry で参照される新しいスキーマは互換性を持たない可能性があります。この場合、ジョブが失敗することがあります。

AWS Glue API

`UpdateTable` API を呼び出す際に、(`StorageDescriptor` に`SchemaReference` が指定されている) `TableInput` を、スキーマレジストリの既存のスキーマに追加します。

AWS Glue コンソール

AWS Glue コンソールからテーブルのスキーマを更新するには

1. AWS Management Consoleにサインインし、AWS Glue コンソール (https://console.aws.amazon.com/glue/) を開きます。
2. ナビゲーションペインの [Data catalog] (データカタログ) で、[Tables] (テーブル) をクリックします。
3. テーブルの一覧でテーブルを表示します。
4. 新しいバージョンの情報が表示されたボックスで、[Update schema] (スキーマの更新) をクリックします。
5. 現在のスキーマと更新後のスキーマの違いを確認します。
6. さらに詳細を表示するには、[Show all schema differences] (スキーマの違いをすべて表示) をクリックします。
7. [Save table] (テーブルを保存) をクリックし、新しいバージョンを受け入れます。

ユースケース: AWS Glue ストリーミング

AWS Glue ストリーミングは、ストリーミングソースからのデータを消費し、出力シンクに書き込む前に ETL オペレーションを実行します。入力ストリーミングソースは、データテーブルを使用して指定するか、ソース構成を指定して直接指定することができます。

AWS Glue ストリーミングは、AWS Glue スキーマレジストリに存在するスキーマで作成されたストリーミングソースのデータカタログテーブルをサポートします。AWS Glue スキーマレジストリにスキーマを作成し、そのスキーマを使用してストリーミングソースで AWS Glue テーブルを作成できます。この AWS Glue テーブルは、 AWS Glue ストリーミングジョブへの入力として使用し、入力ストリーミングのデータを逆シリアル化することができます。

注意すべき点は、AWS Glue スキーマレジスト内のスキーマが変化した場合、AWS Glue ストリーミングジョブを再度開始して、スキーマの変更を反映させる必要があることです。

ユースケース: Apache Kafka ストリーム

Apache Kafka Streams APIは、Apache Kafka に格納されているデータを処理・分析するためのクライアントライブラリです。このセクションでは、Apache Kafka Streams と AWS Glue Schema Registry の統合について説明します。これにより、データストリーミングアプリケーションのスキーマを、管理および適用できるようになります。Apache Kafka Streams の詳細については、「Apache Kafka Streams」を参照してください。

SerDes ライブラリとの統合

`GlueSchemaRegistryKafkaStreamsSerde` クラスにより、Streams のアプリケーションを設定できます。

Kafka Streams アプリケーションのコード例

Apache Kafka Streams アプリケーション内で AWS Glue Schema Registry を使用するには

1. Kafka Streams アプリケーションを設定します。

```
final Properties props = new Properties();
    props.put(StreamsConfig.APPLICATION_ID_CONFIG, "avro-streams");
    props.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092");
    props.put(StreamsConfig.CACHE_MAX_BYTES_BUFFERING_CONFIG, 0);
    props.put(StreamsConfig.DEFAULT_KEY_SERDE_CLASS_CONFIG,
 Serdes.String().getClass().getName());
    props.put(StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_CONFIG,
 AWSKafkaAvroSerDe.class.getName());
```

```
    props.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "earliest");

    props.put(AWSSchemaRegistryConstants.AWS_REGION, "aws-region");
    props.put(AWSSchemaRegistryConstants.SCHEMA_AUTO_REGISTRATION_SETTING, true);
    props.put(AWSSchemaRegistryConstants.AVRO_RECORD_TYPE,
 AvroRecordType.GENERIC_RECORD.getName());
 props.put(AWSSchemaRegistryConstants.DATA_FORMAT, DataFormat.AVRO.name());
```

2. トピック avro-input からストリームを作成します。

```
StreamsBuilder builder = new StreamsBuilder();
final KStream<String, GenericRecord> source = builder.stream("avro-input");
```

3. データレコードを処理します (favorite_color の値がピンクであるか、値が 15 となっているレコードの除外など)。

```
final KStream<String, GenericRecord> result = source
    .filter((key, value) -
> !"pink".equals(String.valueOf(value.get("favorite_color"))));
    .filter((key, value) -> !"15.0".equals(String.valueOf(value.get("amount"))));
```

4. トピック avro-output に結果を書き込みます。

```
result.to("avro-output");
```

5. Apache Kafka Streams アプリケーションを起動します。

```
KafkaStreams streams = new KafkaStreams(builder.build(), props);
streams.start();
```

実装結果

以下の結果は、ステップ 3 において (favorite_color が「pink」であるか値が「15.0」であるために) 除外されたレコードに関するフィルタリング処理を示しています。

フィルタリング前のレコード:

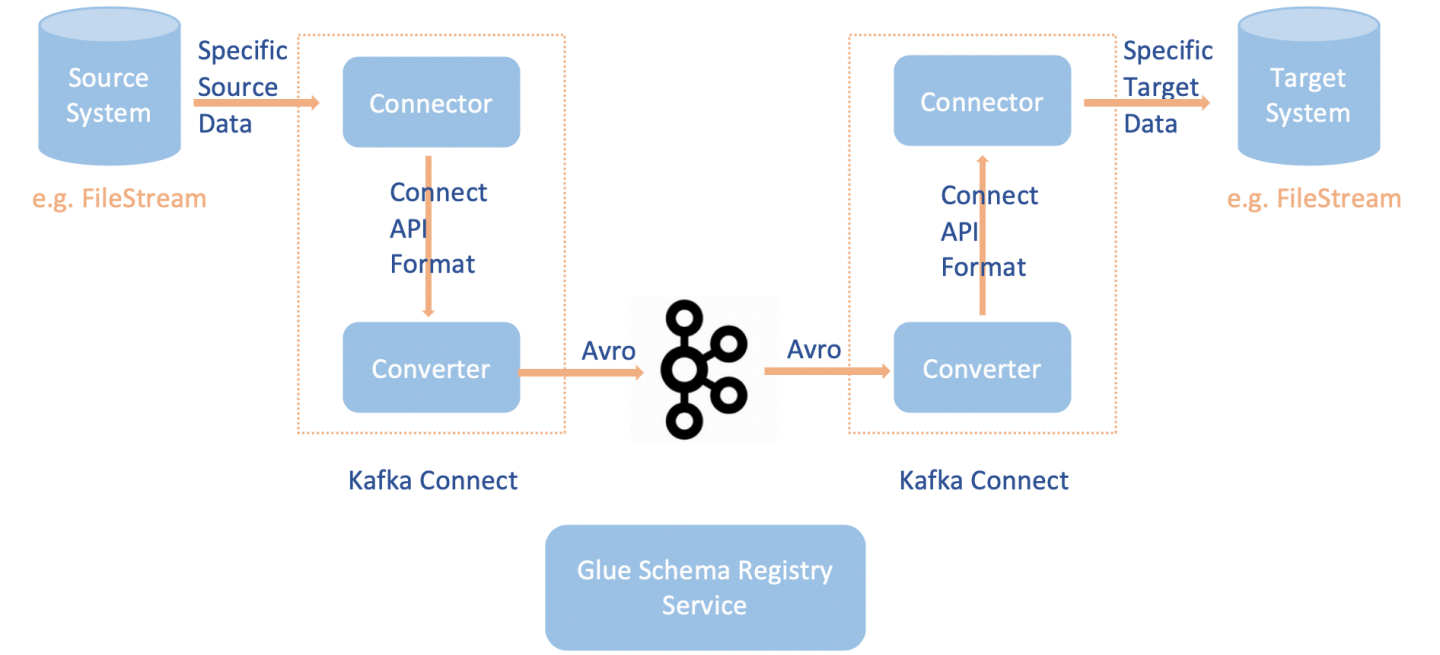
```
{"name": "Sansa", "favorite_number": 99, "favorite_color": "white"}
{"name": "Harry", "favorite_number": 10, "favorite_color": "black"}
{"name": "Hermione", "favorite_number": 1, "favorite_color": "red"}
{"name": "Ron", "favorite_number": 0, "favorite_color": "pink"}
{"name": "Jay", "favorite_number": 0, "favorite_color": "pink"}

{"id": "commute_1","amount": 3.5}
{"id": "grocery_1","amount": 25.5}
{"id": "entertainment_1","amount": 19.2}
{"id": "entertainment_2","amount": 105}
 {"id": "commute_1","amount": 15}
```

フィルタリング後のレコード:

```
{"name": "Sansa", "favorite_number": 99, "favorite_color": "white"}
{"name": "Harry", "favorite_number": 10, "favorite_color": "black"}
{"name": "Hermione", "favorite_number": 1, "favorite_color": "red"}
{"name": "Ron", "favorite_number": 0, "favorite_color": "pink"}

{"id": "commute_1","amount": 3.5}
{"id": "grocery_1","amount": 25.5}
{"id": "entertainment_1","amount": 19.2}
{"id": "entertainment_2","amount": 105}
```

ユースケース: Apache Kafka Connect

Apache Kafka Connect と AWS Glue Schema Registry を統合することで、コネクタからスキーマ情報を取得できるようになります。Apache Kafka のコンバータにより、Apache Kafka 内のデータ形式と、Apache Kafka Connect データへの変換方法を指定します。すべての Apache Kafka Connect ユーザーは、これらのコンバータを Apache Kafka との間でロードまたは保存する際に、データに適用する形式に基づいた設定を行う必要があります。これにより、Apache Kafka Connect データを AWS Glue Schema Registry で使用する型 (例: Avro) に変換する独自のコンバータを定義し、さらにシリアライザを使用してスキーマを登録しシリアル化を実行します。その後コンバータはデシリアライザを使用して、Apache Kafka から受信したデータを逆シリアル化し、元の Apache Kafka Connect データに変換することができます。ワークフローの例を以下の図に示します。



1. AWS Glue Schema Registry 用 Githubリポジトリをクローンして、aws-glue-schema-registry プロジェクトをインストールします。

```
git clone git@github.com:awslabs/aws-glue-schema-registry.git
cd aws-glue-schema-registry
mvn clean install
mvn dependency:copy-dependencies
```

2. Apache Kafka Connect を Standalone モードで使用する予定の場合、このステップで以下に示した手順を使用して、connect-standalone.properties を更新します。Apache Kafka Connect を Distributed モードで使用する予定の場合は、同じ手順により connect-avro-distributed.properties を更新します。

   a. Apache Kafka の接続プロパティファイルにも、これらのプロパティを追加します。

```
key.converter.region=aws-region
value.converter.region=aws-region
key.converter.schemaAutoRegistrationEnabled=true
value.converter.schemaAutoRegistrationEnabled=true
key.converter.avroRecordType=GENERIC_RECORD
value.converter.avroRecordType=GENERIC_RECORD
```

b. 以下のコマンドを、[kafka-run-class.sh] の下の [Launch mode] (起動モード) セクションに追加します。

```
-cp $CLASSPATH:"<your AWS GlueSchema Registry base directory>/target/dependency/*"
```

3. [kafka-run-class.sh] の下の [Launch mode] (起動モード) セクションに以下のコマンドを追加する

```
-cp $CLASSPATH:"<your AWS GlueSchema Registry base directory>/target/dependency/*"
```

プリンシパルは以下のようになります。

```
# Launch mode
if [ "x$DAEMON_MODE" = "xtrue" ]; then
  nohup "$JAVA" $KAFKA_HEAP_OPTS $KAFKA_JVM_PERFORMANCE_OPTS $KAFKA_GC_LOG_OPTS
 $KAFKA_JMX_OPTS $KAFKA_LOG4J_OPTS -cp $CLASSPATH:"/Users/johndoe/aws-glue-schema-
registry/target/dependency/*" $KAFKA_OPTS "$@" > "$CONSOLE_OUTPUT_FILE" 2>&1 < /dev/
null &
else
  exec "$JAVA" $KAFKA_HEAP_OPTS $KAFKA_JVM_PERFORMANCE_OPTS $KAFKA_GC_LOG_OPTS
 $KAFKA_JMX_OPTS $KAFKA_LOG4J_OPTS -cp $CLASSPATH:"/Users/johndoe/aws-glue-schema-
registry/target/dependency/*" $KAFKA_OPTS "$@"
fi
```

4. bash を使用している場合は、以下のコマンドを実行して、bash_profile で CLASSPATH を設定します。他のシェルの場合は、それに応じて環境を更新します。

```
echo 'export GSR_LIB_BASE_DIR=<>' >>~/.bash_profile
echo 'export GSR_LIB_VERSION=1.0.0' >>~/.bash_profile
echo 'export KAFKA_HOME=<your Apache Kafka installation directory>' >>~/.bash_profile
echo 'export CLASSPATH=$CLASSPATH:$GSR_LIB_BASE_DIR/avro-kafkaconnect-converter/
target/schema-registry-kafkaconnect-converter-$GSR_LIB_VERSION.jar:$GSR_LIB_BASE_DIR/
common/target/schema-registry-common-$GSR_LIB_VERSION.jar:$GSR_LIB_BASE_DIR/
avro-serializer-deserializer/target/schema-registry-serde-$GSR_LIB_VERSION.jar'
 >>~/.bash_profile
source ~/.bash_profile
```

5. (オプション) 単純なファイルをソースとして使用しテストを行う場合は、ファイルソースコネクタのクローンを作成します。

```
git clone https://github.com/mmolimar/kafka-connect-fs.git
```

```
cd kafka-connect-fs/
```

a. ソースコネクタの設定で、データ形式を Avro に、ファイルリーダーを AvroFileReader に変更します。さらに、読み込んでいるファイルパスからサンプルの Avro オブジェクトを更新します。例:

```
vim config/kafka-connect-fs.properties
```

```
fs.uris=<path to a sample avro object>
policy.regexp=^.*\.avro$
file_reader.class=com.github.mmolimar.kafka.connect.fs.file.reader.AvroFileReader
```

b. ソースコネクタをインストールします。

```
mvn clean package
echo "export CLASSPATH=\$CLASSPATH:\"\$(find target/ -type f -name '*.jar'| grep
 '\-package' | tr '\n' ':')\"" >>~/.bash_profile
source ~/.bash_profile
```

c. *<your Apache Kafka installation directory>*/config/connect-file-sink.properties のシンクのプロパティを更新し、トピック名と出力ファイル名を更新します。

```
file=<output file full path>
topics=<my topic>
```

6. Source Connector (この例では、ソースファイルのコネクタ) を起動します。

```
$KAFKA_HOME/bin/connect-standalone.sh $KAFKA_HOME/config/connect-
standalone.properties config/kafka-connect-fs.properties
```

7. Sink Connector (この例では、シンクファイルのコネクタ) を実行します。

```
$KAFKA_HOME/bin/connect-standalone.sh $KAFKA_HOME/config/connect-
standalone.properties $KAFKA_HOME/config/connect-file-sink.properties
```

Kafka Connect の使用例については、AWS Glue Schema Registry 用 Githubリポジトリの、integration-tests フォルダにある run-local-tests.sh スクリプトでご確認ください。

サードパーティー製のスキーマレジストリから AWS Glue Schema Registry への移行

サードパーティー製スキーマレジストリから AWS Glue Schema Registry への移行作業は、その時点での既存のサードパーティー製スキーマレジストリにより異なります。サードパーティーのスキーマレジストリを使用して送信された Apache Kafka トピック内にレコードがある場合、コンシューマはサードパーティーのスキーマレジストリを使用して、それらのレコードを非シリアル化する必要があります。`AWSKafkaAvroDeserializer` には、セカンダリのデシリアライザクラスを定義する機能があります。このクラスは、サードパーティーのデシリアライザを特定し、対象のレコードを非シリアル化するために使用できます、

サードパーティー製スキーマには、使用停止に関して 2 つの基準があります。1 つ目の基準では、サードパーティー製スキーマレジストリを使用する Apache Kafka トピックのレコードを必要とするコンシューマが、なくなった後にのみ使用停止となります。2 つ目の基準では、トピックに指定された保持期間に応じ、Apache Kafka トピックの試用期間が終了することで使用停止となります。無制限の保持期間を持つトピックの場合は、AWS Glue Schema Registry への移行は可能ですが、サードパーティー製スキーマレジストリを使用停止にすることはできません。この回避策としては、アプリケーションまたは Mirror Maker 2 により現在のトピックを読み取り、AWS Glue Schema Registry を使用する新しいトピックとして作成し直すことができます。

サードパーティー製スキーマレジストリから AWS Glue Schema Registry へ移行するには

1. AWS Glue Schema Registry にレジストリを作成するか、デフォルトのレジストリを使用します。
2. コンシューマを停止します。AWS Glue Schema Registry をプライマリのデシリアライザとして含めるように、コンシューマを変更します、サードパーティーのスキーマレジストリをセカンダリに変更します。
   - コンシューマのプロパティを設定します。この例では、secondary_decerializer は別のデシリアライザに設定されています。動作は次のとおりです。コンシューマは Amazon MSK からレコードを取得し、最初に `AWSKafkaAvroDeserializer` の使用を試みます。AWS Glue Schema Registry のスキーマの、Avro スキーマ ID を含むマジックバイトを読み取ることができない場合、`AWSKafkaAvroDeserializer`は、secondary_deserializer で指定されるデシリアライザクラスの使用を試みます。セカンダリのデシリアライザに固有のプロパティは、以下に示すように、schema_registry_url_config や specific_avro_reader_config などのコンシューマプロパティでも指定する必要があります。

```
consumerProps.setProperty(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,
 StringDeserializer.class.getName());
consumerProps.setProperty(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,
 AWSKafkaAvroDeserializer.class.getName());
consumerProps.setProperty(AWSSchemaRegistryConstants.AWS_REGION,
 KafkaClickstreamConsumer.gsrRegion);
consumerProps.setProperty(AWSSchemaRegistryConstants.SECONDARY_DESERIALIZER,
 KafkaAvroDeserializer.class.getName());
consumerProps.setProperty(KafkaAvroDeserializerConfig.SCHEMA_REGISTRY_URL_CONFIG,
 "URL for third-party schema registry");
consumerProps.setProperty(KafkaAvroDeserializerConfig.SPECIFIC_AVRO_READER_CONFIG,
 "true");
```

3. コンシューマを再起動します。
4. プロデューサを停止し、そのプロデューサで AWS Glue Schema Registry を指定します。
   a. プロデューサのプロパティを設定します。この例のプロデューサでは、デフォルトのレジストリと自動登録スキーマバージョンを使用しています。

```
producerProps.setProperty(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
 StringSerializer.class.getName());
producerProps.setProperty(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
 AWSKafkaAvroSerializer.class.getName());
producerProps.setProperty(AWSSchemaRegistryConstants.AWS_REGION, "us-east-2");
producerProps.setProperty(AWSSchemaRegistryConstants.AVRO_RECORD_TYPE,
 AvroRecordType.SPECIFIC_RECORD.getName());
producerProps.setProperty(AWSSchemaRegistryConstants.SCHEMA_AUTO_REGISTRATION_SETTING,
 "true");
```

5. (オプション) 既存のスキーマおよびスキーマバージョンを、現在のサードパーティー製スキーマレジストリから AWS Glue Schema Registry (AWS Glue Schema Registry のデフォルトレジストリ、または AWS Glue Schema Registry 内の特定の非デフォルトレジストリ) に手動で移行します。これには、サードパーティーのスキーマレジストリから JSON 形式でスキーマをエクスポートし、AWS Management Console または AWS CLI を使用しながら AWS Glue Schema Registry の新しいスキーマを作成します。

   このステップは、AWS CLI および AWS Management Console を使用して、新しく作成されたスキーマバージョンについて、以前のスキーマバージョンとの互換性チェックを有効にする必要がある場合、またはスキーマバージョンの自動登録を有効にした新しいスキーマで、プロデューサがメッセージを送信する場合に重要です。

6. プロデューサを起動します。

データへの接続

AWS Glue 接続は、特定のデータストアのログイン認証情報、URI 文字列、仮想プライベートクラウド (VPC) 情報などを保存するデータカタログオブジェクトです。AWS Glue クローラー、ジョブ、および開発エンドポイントは、特定のタイプのデータストアにアクセスするために接続を使用します。ソースとターゲットの両方に接続を使用したり、複数のクローラーまたは抽出、変換、ロード (ETL) ジョブで同じ接続を再利用したりできます。

AWS Glue 接続スキーマの最新バージョンでは、AWS Glue、Amazon Athena、Amazon SageMaker AI Unified Studio などの AWS サービスとアプリケーションとの間のデータ接続を統一された方法で管理できます。

コネクタと接続の使用に関する概要

接続には、特定のデータストアに接続するために必要なプロパティが含まれます。作成した接続は AWS Glue Data Catalog に保存されます。コネクタを選択し、そのコネクタに基づいて接続を作成します。

ネイティブにサポートされていないデータストアへのコネクタは、AWS Marketplace 内でサブスクライブすることができます。その後、接続を作成するときにこれらのコネクタを使用します。さらに、デベロッパーは独自のコネクタを作成し、それを接続の作成に使用することもできます。

> **Note**
>
> カスタムのコネクタ、または AWS Marketplace からのコネクタを使用して作成された接続は、タイプを AWS Glue Studio として、AWS Glue コンソール内の `UNKNOWN` に表示されます。

以下のステップで、AWS Glue Studio でコネクタを使用するための全体的なプロセスを説明します。

1. AWS Marketplace 内でコネクタをサブスクライブするか、独自のコネクタを開発して、AWS Glue Studio にアップロードします。詳しくは、「AWS Glue Studio にコネクタを追加する 」を参照してください。
2. コネクタの使用方法に関する情報を確認します。この情報は、コネクタ製品ページの [Usage] (使用方法) タブに表示されます。例えば、[Usage] (使用状況) タブで、「AWS Glue Connector for

Google BigQuery」をクリックした場合、[Additional Resources] (その他のリソース) セクションには、このコネクタの使用に関するブログへのリンクが表示されます。

3. 接続を作成します。使用するコネクタを選択し、ログイン認証情報、URI 文字列、仮想プライベートクラウド (VPC) 情報など、接続に関する追加情報を提供します。詳しくは、「コネクタ用の接続を作成する 」を参照してください。
4. ジョブ用に IAM ロールを作成します。ジョブは、作成時に指定する [IAM role] (IAM ロール) のアクセス許可があることを想定します。この IAM ロールには、データストアを承認し、そこからのデータ抽出、およびデータを書き込むために必要なアクセス許可を有する必要があります。
5. ETL ジョブを作成し、その ETL ジョブのためにデータソースプロパティを設定します。カスタムコネクタプロバイダーの指示に従って、接続オプションと認証情報を指定します。詳しくは、「カスタムコネクタを使用したジョブのオーサリング 」を参照してください。
6. AWS Glue Studio でビジュアル ETL ジョブを開始する での説明を参考に、変換を追加するか新しいデータストアを追加しながら、ETL ジョブをカスタマイズします。
7. データターゲットにコネクタを使用している場合は、ETL ジョブ用のデータターゲットプロパティを設定します。カスタムコネクタプロバイダーの指示に従って、接続オプションと認証情報を指定します。詳しくは、「the section called “カスタムコネクタを使用したジョブのオーサリング” 」を参照してください。
8. ジョブのプロパティを変更する での説明のように、ジョブのプロパティを構成して、ジョブの実行環境をカスタマイズします。
9. ジョブを実行します。

統合接続

AWS は最近、「SageMaker LakeHouse 接続」または「AWS Glue 統合接続」と呼ばれる新機能を導入しました。この機能を使用すると、AWS Glue や Amazon Athena などの複数の AWS サービスで使用できる接続を作成できます。Amazon Athena でデータソースを作成すると、AWS Glue 接続入力を参照するセクションが表示されます。この場合、Amazon Athena は、接続の「AthenaProperties」セクションにある Amazon Athena 固有のプロパティを含む AWS Glue 接続を作成します。

一方、AWS Glue で直接接続を作成する場合、AWS Glue および Apache Spark に固有のプロパティを入力するように求められます。これは、当該接続の「ConnectionProperties」セクションと「SparkProperties」セクションに保存されます。

いずれのシナリオでも「統合接続」が作成されますが、Amazon Athena で作成された接続は Amazon Athena 内でのみ使用するように設定され、AWS Glue で作成された接続は AWS Glue 内

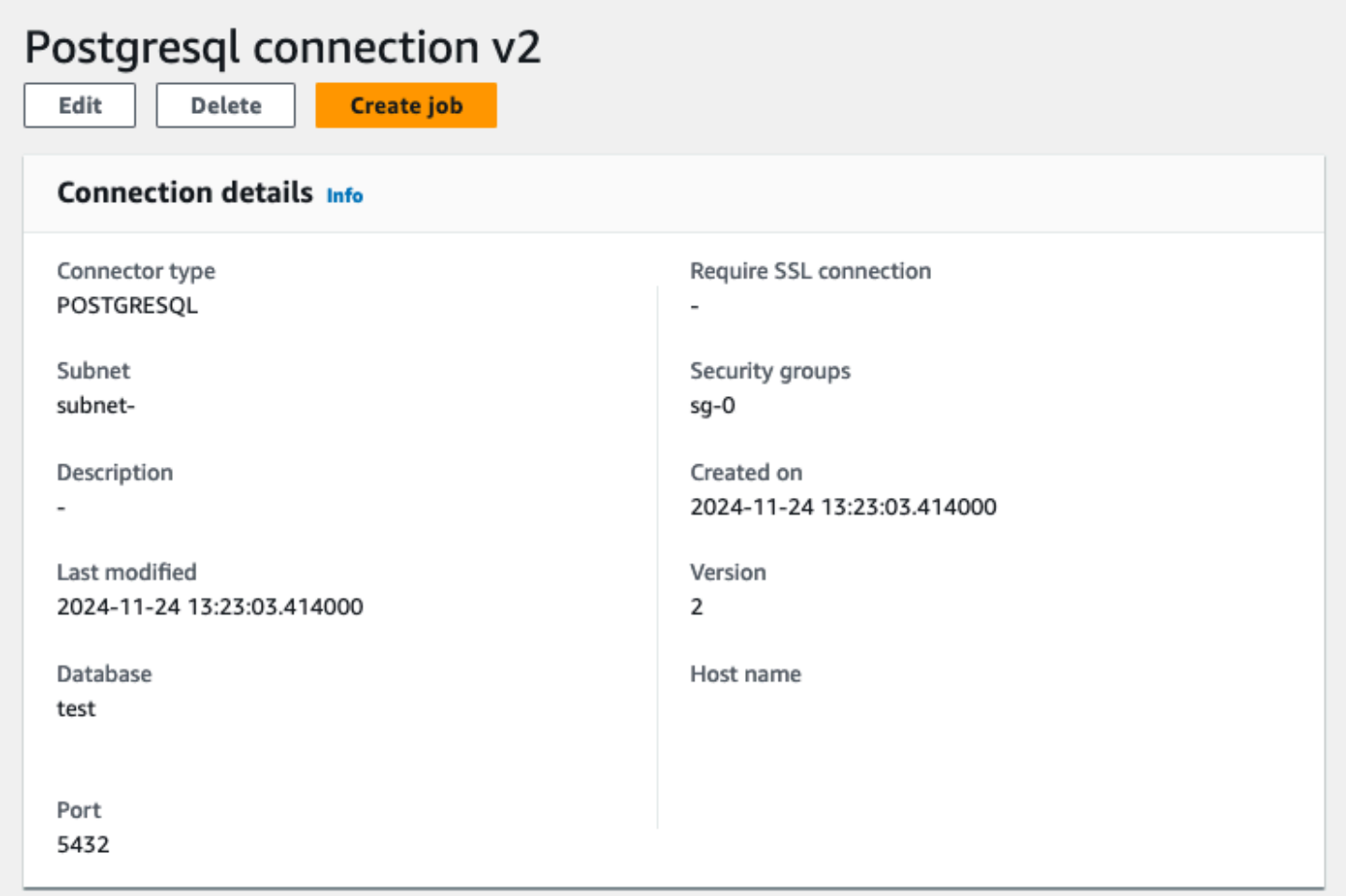
でのみ使用するように設定されます。ただし、これらのプロパティ (Amazon Athena または Spark プロパティ) を欠落している接続を更新して、両方のサービスで使用できるようにすることは可能です。Amazon SageMaker AIUnified Studio は、AWS Glue の接続に必要なすべてのプロパティ (「ConnectionProperties」、「AthenaProperties」、「SparkProperties」) を入力してこの処理を自動的に実行し、AWS Glue と Amazon Athena の両方で接続を使用できるようにします。

これらを「統合接続」と呼びますが、AWS Glue または Amazon Athena で個別に作成した接続は、両方のサービスで使用できるように適切に設定しない限り、真の統合接続とはならない点に注意してさい。SageMaker Unified Studio で作成された接続のみが、追加設定なしで複数のサービスですぐ使用可能な真の統合接続となります。

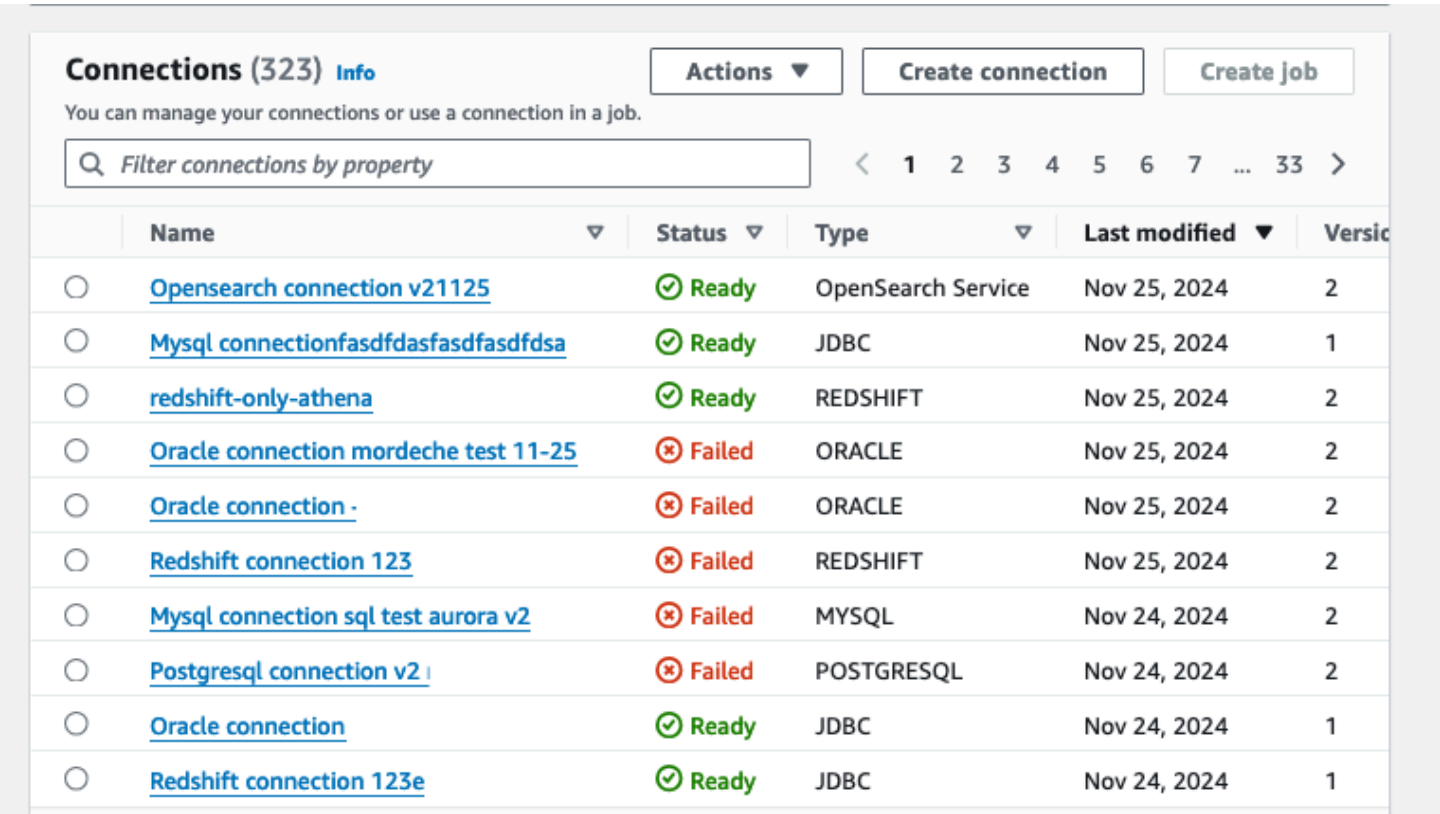
さらに、AWS Glue で作成された接続は Amazon Athena には表示されません。これは Amazon Athena に表示されるデータソースは AWS Glue 接続への参照を含むが AWS Glue 接続そのものではないためです。同様に、Amazon Athena で作成された接続は AWS Glue Studio には表示されません。これは、AWS Glue に必要な設定で設定されていない接続が AWS Glue Studio により除外されるためです。

AWS Glue Studio は、デフォルトで統合接続を作成します。AWS Glue コンソールでは、接続のバージョンを、接続ページの接続テーブル、接続の詳細ページの接続テーブル、ジョブの詳細ページの接続テーブルで、それぞれ確認できます。

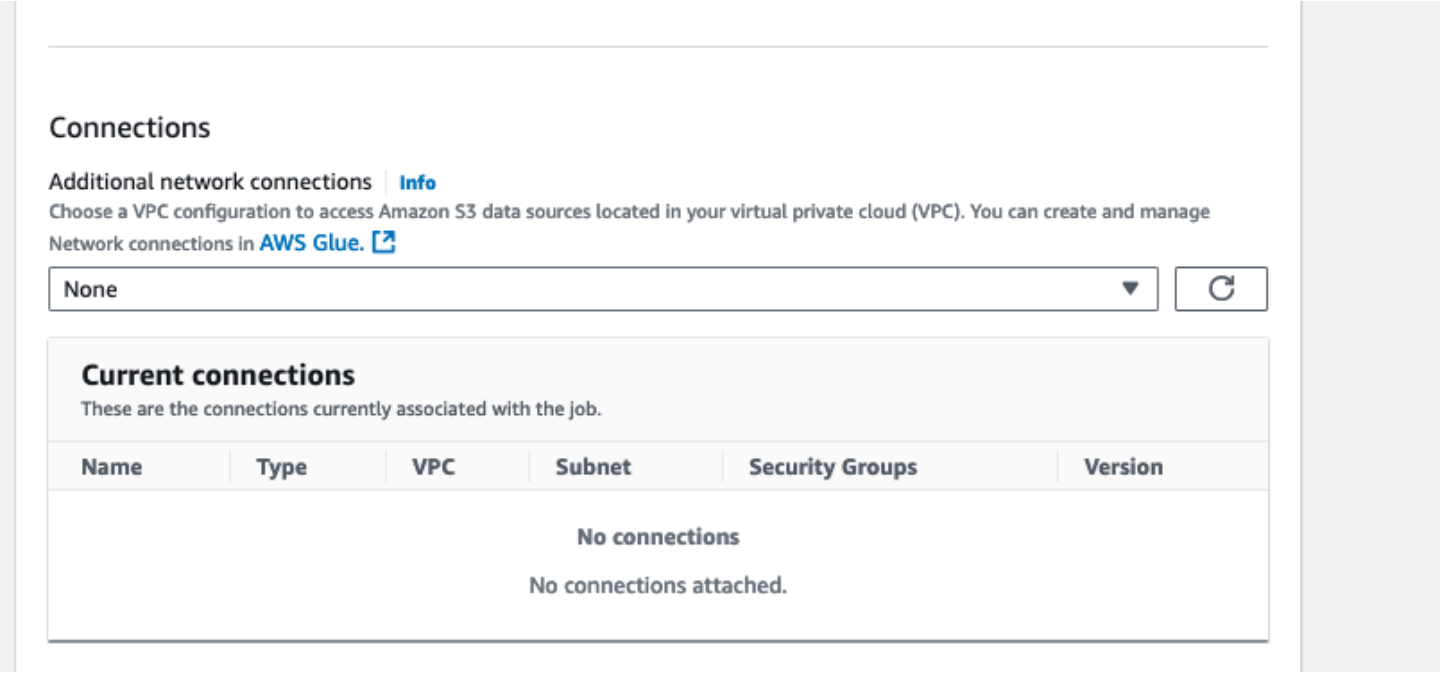
接続のバージョンは [接続の詳細] で確認できます。



接続バージョンは、すべての接続を表示するときにも確認できます。



最後に、接続バージョンはジョブの [ジョブの詳細] タブに表示されます。



バージョン 2 接続では、次の拡張されたデータ接続の機能を使用できます。

- 接続タイプの検出: 標準化されたテンプレートを使用する接続の作成をサポート。ユーザーがアクセス可能な接続タイプと、特定の接続タイプの必須および任意の入力を AWS Glue が自動的に検出します。
- 再利用性: AWS データ処理エンジンや AWS Glue、Amazon Athena、Amazon SageMaker AI などのツール全体で再利用可能な接続定義。接続に AthenaProperties、SparkProperties、PythonProperties が追加されました。これにより、ConnectionProperties に保存されている一般的なプロパティに加え、コンピューティング環境/サービスに固有の接続プロパティを指定することができます。Athena は、AthenaProperties プロパティマップで Athena 固有のプロパティを指定して、AWS Glue で接続を作成するようになりました。
- データプレビュー: 接続されたソースからメタデータを参照しデータをプレビューする機能。
- コネクタメタデータ: テーブルメタデータを検出するために再利用可能な接続を使用できます。
- サービスにリンクされたシークレット: ユーザーは、`CreateConnection` リクエストに必要な OAuth、基本またはカスタムの認証情報を提供できます。CreateConnection API は、アカウントに Service Linked Secret を作成し、ユーザーに代わって認証情報を保存します。

サポートされている認可タイプ

統合接続では、次の認可タイプがサポートされています。

- BASIC – ほとんどのデータベース接続タイプと既存の AWS Glue 接続タイプでは、基本認可 (ユーザー名とパスワード) がサポートされています。以前は、SecretsManager のキーの名前はコネクタ固有であり、例えば、user、username、userName、opensearch.net.http.auth.user などでした。ここで、統合接続によって USERNAME キーと PASSWORD キーで基本認可接続タイプが標準化されました。
- OAUTH2 – 新規に起動される SaaS 接続タイプのほとんどは、OAuth2 プロトコルをサポートしています。
- CUSTOM – 一部の接続タイプには、Google BigQuery などの他の認可メカニズムがあります。Google BigQuery の場合、ユーザーは Google BigQuery から取得する JSON を提供することになります。

考慮事項

データソースの統合接続を作成するときは、次の違いを考慮してください。

- AWS Glue Studio を介して統合接続を作成する場合、ユーザー認証情報は接続自体ではなく AWS Secrets Manager に保存されます。つまり、ジョブは Secrets Manager にアクセスする必要があります。
- ジョブが VPC で実行されている場合、AWS Secrets Manager および Secure Token Service (STS) にアクセスするために VPC エンドポイントまたは NAT ゲートウェイのいずれかが必要です。これにより、追加コストが発生します。
- 特定のデータソース (Redshift、SQL Server、MySQL、Oracle、PostgreSQL) では、AWS Glue Studio を介して統合接続を作成するには、AWS STS と AWS Secrets Manager にアクセスする必要があります。これは、安全な接続を確立し、仮想プライベートクラウド (VPC) 内のこれらのデータソースにアクセスするために必要な認証情報を取得するために必要です。
- AWS Glue Studio 経由で統合接続を作成するには、AWS Secrets Manager にアクセスして VPC リソースを管理するためのアクセス許可を持つ IAM ロールが必要です (VPC を使用している場合)。
  - secretsmanager:GetSecretValue
  - secretsmanager:PutSecretValue
  - secretsmanager:DescribeSecret
  - ec2:CreateNetworkInterface
  - ec2:DeleteNetworkInterface
  - ec2:DescribeNetworkInterfaces
  - ec2:DescribeSubnets

使用可能な接続

AWS Glue では次の接続タイプをサポートしています。

- Adobe Analytics
- Adobe Marketo Engage
- Amazon Aurora (ネイティブ JDBC ドライバーを使用している場合はサポートされます。すべてのドライバー機能を利用できるわけではありません)
- Amazon DocumentDB
- Amazon DynamoDB
- Amazon OpenSearch Service (AWS Glue for Spark で使用)。
- Amazon Redshift

- Asana
- Azure Cosmos (AWS Glue ETL ジョブで Azure Cosmos DB for NoSQL を使用する場合)
- Azure SQL (AWS Glue for Spark で使用)。
- Blackbaud
- CircleCI
- Datadog
- Docusign Monitor
- Domo
- Dynatrace
- Facebook 広告
- Facebook Page Insights
- Freshdesk
- Freshsales
- Google 広告
- Google Analytics 4
- Google BigQuery (AWS Glue for Spark で使用)。
- Google 検索コンソール
- Google Sheets
- HubSpot
- Instagram 広告
- Intercom
- JDBC
- Jira Cloud
- Kafka
- Kustomer
- LinkedIn
- Mailchimp
- Microsoft Teams
- Mixpanel
- 月曜日

- MongoDB
- MongoDB Atlas
- Okta
- Oracle NetSuite
- Paypal
- Pendo
- Pipedrive
- Productboard
- QuickBooks
- Salesforce
- Salesforce Commerce Cloud
- Salesforce Marketing Cloud
- Salesforce Marketing Cloud Account Engagement (旧 Salesforce Pardot)
- SAP HANA (AWS Glue for Spark で使用)。
- SAP OData
- SendGrid
- ServiceNow
- Slack
- Smartsheet
- Snapchat 広告
- Stripe
- Snowflake (AWS Glue for Spark で使用)。
- Teradata Vantage (AWS Glue for Spark を使用する場合)。
- Twilio
- Vertica (AWS Glue for Spark で使用)。
- WooCommerce
- Zendesk
- Zoho CRM
- Zoom Meetings
- さまざまな Amazon Relational Database Service (Amazon RDS) オファリング。

- Network (Amazon Virtual Private Cloud (Amazon VPC) 内のデータソースへの接続を指定します)

AWS Glue Studio を使用して、コネクタの接続を作成することもできます。コネクタとは AWS Glue Studio 内でデータストアに対するアクセスを支援するための、オプションのコードパッケージです。詳細については、「Using connectors and connections with AWS Glue Studio」を参照してください。

オンプレミスのデータベースに接続する方法の詳細については、AWS ビッグデータブログ記事「AWS Glue を使用することによってオンプレミスデータストアにアクセスして分析する方法」を参照してください。

カスタム IAM ロールの使用中に VPC 設定を使用した接続を作成するには、次の VPC アクセスのアクションが必要です。

- secretsmanager:GetSecretValue
- secretsmanager:PutSecretValue
- secretsmanager:DescribeSecret
- ec2:CreateNetworkInterface
- ec2:DeleteNetworkInterface
- ec2:DescribeNetworkInterfaces
- ec2:DescribeSubnets

制限

- AWS Glue API を使用して v2 接続を作成した場合は、AWS Glue コンソールから接続を編集することはできません。
  - Amazon DocumentDB
  - Amazon Aurora
  - MariaDB
  - MongoDB Atlas
  - MongoDB

AWS Glue 接続プロパティ

このトピックでは、AWS Glue 接続のプロパティについて説明します。

トピック

- 必要な接続プロパティ
- AWS Glue JDBC 接続プロパティ
- AWS Glue MongoDB と MongoDB Atlas の接続プロパティ
- Salesforce 接続プロパティ
- Snowflake 接続
- Vertica 接続
- SAP HANA 接続
- Azure SQL 接続
- Teradata Vantage 接続
- OpenSearch Service 接続
- Azure Cosmos 接続
- AWS Glue SSL 接続プロパティ
- クライアント認証用の Apache Kafka 接続プロパティ
- Google BigQuery 接続
- Vertica 接続

必要な接続プロパティ

AWS Glue コンソールで設定を定義する場合は、次のプロパティに値を指定する必要があります。

接続名

接続用の一意の名前を入力します。

接続タイプ

JDBC またはいずれかの接続タイプを選択します。

JDBC 接続タイプの詳細については、「the section called “JDBC 接続プロパティ”」を参照してください。

Amazon Virtual Private Cloud 環境 (Amazon VPC) 内のデータソースへ接続するには、Network を選択します。

選択したタイプに応じて、AWS Glue コンソールにその他の必須フィールドが表示されます。例えば、Amazon RDS を選択した場合は、データベースエンジンを選択する必要があります。

要求される SSL 接続

このオプションを選択すると、AWS Glue では、データストア接続が信頼できる Secure Sockets Layer (SSL) 経由であることの確認が必要です。

このオプションを選択したときに利用できる追加オプションなどの詳細については、「the section called “SSL 接続プロパティ”」を参照してください。

MSK クラスターを選択 (Amazon Managed Streaming for Apache Kafka (MSK) のみ)

別の AWS アカウントからの MSK クラスターを指定します。

Kafka ブートストラップサーバーの URL (Kafka のみ)

ブートストラップサーバーの URL のカンマ区切りリストを指定します。ポート番号を含めます。例: b-1.vpc-test-2.o4q88o.c6.kafka.us-east-1.amazonaws.com:9094, b-2.vpc-test-2.o4q88o.c6.kafka.us-east-1.amazonaws.com:9094, b-3.vpc-test-2.o4q88o.c6.kafka.us-east-1.amazonaws.com:9094

AWS Glue JDBC 接続プロパティ

AWS Glue Studio は、MySQL、Oracle、PostgresSQL、Redshift、および SQL Server データソースの統合接続を作成するようになりました。これには、Secrets Manager および VPC リソースにアクセスするための追加のステップが必要で、追加コストが発生する可能性があります。AWS Glue Studio でこれらの接続にアクセスするには、それぞれの接続の接続名を選択します。

詳細については、「考慮事項」を参照してください。

AWS Glue は、JDBC 接続を介して次のデータストアに接続できます。

- Amazon Redshift
- Amazon Aurora
- Microsoft SQL Server
- MySQL
- Oracle
- PostgreSQL

- Snowflake (AWS Glue クローラーを使用する場合)
- Aurora (ネイティブ JDBC ドライバーを使用している場合はサポートされます。すべてのドライバー機能を利用できるわけではありません)
- Amazon RDS for MariaDB

> **Important**
>
> 現在、ETL ジョブは 1 つのサブネット内でしか JDBC 接続を使用できません。1 つのジョブに複数のデータストアがある場合は、同じサブネットにあるか、サブネットからアクセス可能である必要があります。
> AWS Glue クローラー用に独自の JDBC ドライバーバージョンを導入する場合、クローラーは AWS Glue ジョブと Amazon S3 のリソースを使用して、用意したドライバーが自分の環境で実行されるようにします。リソースの追加使用量はアカウントに反映されます。さらに、独自の JDBC ドライバーを用意しても、クローラーがドライバーの機能をすべて活用できるわけではありません。ドライバーは、「Data Catalog での接続の定義」に記載されているプロパティに限定されます。

JDBC 接続タイプの追加プロパティを次に示します。

JDBC URL

JDBC データストアの URL を入力します。ほとんどのデータベースエンジンの場合、このフィールドは次の形式になります。この形式では、*protocol*、*host*、*port*、および *db_name* を独自の情報に置き換えます。

```
jdbc:protocol://host:port/db_name
```

データベースエンジンに応じて、別の JDBC URL の形式が必要な場合があります。この形式では、コロン (:) とスラッシュ (/) の使用方法が若干異なるか、データベースを指定するためのキーワードが異なる場合があります。

JDBC をデータストアに接続するためには、データストアの db_name が必要です。db_name は、指定された username と password を使用してネットワーク接続を確立するために使用されます。接続すると、AWS Glue はデータストア内の他のデータベースにアクセスして、クローラーを実行したり ETL ジョブを実行したりできます。

次の JDBC URL の例は、いくつかのデータベースエンジンの構文を示しています。

- dev データベースを使用して Amazon Redshift クラスターデータストアに接続するには

  ```
  jdbc:redshift://xxx.us-east-1.redshift.amazonaws.com:8192/dev
  ```

- employee データベースを使用して Amazon RDS for MySQL データストアに接続するには

  ```
  jdbc:mysql://xxx-cluster.cluster-xxx.us-east-1.rds.amazonaws.com:3306/
  employee
  ```

- employee データベースを使用して Amazon RDS for PostgreSQL データストアに接続するには

  ```
  jdbc:postgresql://xxx-cluster.cluster-xxx.us-
  east-1.rds.amazonaws.com:5432/employee
  ```

- employee サービス名を使用して Amazon RDS for Oracle データストアに接続するには

  ```
  jdbc:oracle:thin://@xxx-cluster.cluster-xxx.us-
  east-1.rds.amazonaws.com:1521/employee
  ```

  Amazon RDS for Oracle の構文は次のパターンに従います。これらのパターンでは、*host*、*port*、*service_name*、*SID* を独自の情報に置き換えます。

  - `jdbc:oracle:thin://@`*host*`:`*port*`/`*service_name*
  - `jdbc:oracle:thin://@`*host*`:`*port*`:`*SID*

- employee データベースを使用して Amazon RDS for Microsoft SQL Server データストアに接続するには

  ```
  jdbc:sqlserver://xxx-cluster.cluster-xxx.us-
  east-1.rds.amazonaws.com:1433;databaseName=employee
  ```

  Amazon RDS for SQL Server の構文は次のパターンに従います。これらのパターンでは、*server_name*、*port*、および *db_name* を独自の情報に置き換えます。

  - `jdbc:sqlserver://`*server_name*`:`*port*`;database=`*db_name*
  - `jdbc:sqlserver://`*server_name*`:`*port*`;databaseName=`*db_name*

- employee データベースの Amazon Aurora PostgreSQL インスタンスに接続するには、データベースインスタンスのエンドポイント、ポート、およびデータベース名を次のように指定します。

  ```
  jdbc:postgresql://employee_instance_1.xxxxxxxxxxxx.us-
  east-2.rds.amazonaws.com:5432/employee
  ```

- Amazon RDS for MariaDB データストアに接続する (employee データベースを持つ) には、データベースインスタンスのエンドポイント、ポート、およびデータベース名を指定します。

  jdbc:mysql://*xxx*-cluster.cluster-*xxx*.*aws-region*.rds.amazonaws.com:3306/employee

- > **Warning**
  >
  > Snowflake JDBC 接続は、AWS Glue クローラーによってのみサポートされます。AWS Glue ジョブで Snowflake コネクタを使用する場合は、Snowflake 接続タイプを使用します。

  sample データベースのインスタンスに接続するには、Snowflake インスタンスのエンドポイント、ユーザー、データベース名、およびロール名を指定します。必要に応じて warehouse パラメータを追加できます。

  jdbc:snowflake://*account_name*.snowflakecomputing.com/?user=*user_name*&db=sample&role=*role_name*&warehouse=*warehouse_name*

  > **Important**
  >
  > JDBC 経由の Snowflake 接続では、URL 内のパラメータは、user、db、role_name、warehouse の順序にする必要があります。

- AWS プライベートリンクで sample データベースのインスタンスに接続するには、Snowflake JDBC URL を次のように指定します。

  jdbc:snowflake://*account_name*.*region*.privatelink.snowflakecomputing.com/?user=*user_name*&db=sample&role=*role_name*&warehouse=*warehouse_name*

ユーザーネーム

> **Note**
>
> ユーザー名とパスワードを直接入力するのではなく、AWS シークレットを使用して接続認証情報を保存することをお勧めします。詳細については、「での接続認証情報の保存 AWS Secrets Manager」を参照してください。

JDBC データストアにアクセスする権限を持つユーザー名を指定します。

パスワード

JDBC データストアへのアクセス権限を持つユーザー名のパスワードを入力します。

ポート

JDBC URL で使用するポートを入力し Amazon RDS Oracle インスタンスに接続します。このフィールドは、Amazon RDS Oracle インスタンスに対して [Require SSL connection] (SSL 接続が必要) を選択したときにしか表示されません。

VPC

データストアを含むVirtual Private Cloud (VPC) の名前を選択します。AWS Glue コンソールには、現在のリージョンの VPC がすべて表示されます。

> **Important**
>
> Snowflake からのデータなど、AWS の外部でホストされている JDBC 接続を使用する場合、VPC にはトラフィックをパブリックサブネットとプライベートサブネットに分割する NAT ゲートウェイが必要です。パブリックサブネットは外部ソースへの接続に使用され、内部サブネットは AWS Glue による処理に使用されます。Amazon VPC を外部接続用に設定する方法については、「NAT デバイスを使用してインターネットまたは他のネットワークに接続する」と「AWS Glue から Amazon RDS データストアに JDBC 接続するための Amazon VPC の設定」を参照してください。

サブネット

データストアを含む VPC 内のサブネットを選択します。AWS Glue コンソールには VPC 内のデータストアのすべてのサブネットが一覧表示されます。

セキュリティグループ

データストアに関連付けられているセキュリティグループを選択します。AWS Glue には、AWS Glue の接続を許可するインバウンドソースルールを持つ 1 つ以上のセキュリティグループが必要です。AWS Glue コンソールには、VPC へのインバウンドアクセスが許可されているすべてのセキュリティグループが一覧表示されます。AWS Glue は、VPC サブネットにアタッチされている Elastic Network Interface にこれらのセキュリティグループを関連付けます。

JDBC ドライバークラス名 - オプション

次のカスタム JDBC ドライバークラス名を指定します。

- Postgres – org.postgresql.Driver
- MySQL – com.mysql.jdbc.Driver、com.mysql.cj.jdbc.Driver
- Redshift – com.amazon.redshift.jdbc.Driver、com.amazon.redshift.jdbc42.Driver
- Oracle – oracle.jdbc.driver.OracleDriver
- SQL Server – com.microsoft.sqlserver.jdbc.SQLServerDriver

JDBC ドライバー S3 パス - オプション

カスタム JDBC ドライバーに対して Amazon S3 の場所を指定します。これは.jar ファイルへの絶対パスです。クローラーがサポートするデータベースのデータソースに接続するための独自の JDBC ドライバーを用意したい場合は、`customJdbcDriverS3Path` パラメータと `customJdbcDriverClassName` パラメータの値を指定できます。
お客様が用意した JDBC ドライバーの使用は、必要な 必要な接続プロパティ に限られます。

AWS Glue MongoDB と MongoDB Atlas の接続プロパティ

MongoDB または MongoDB Atlas 接続タイプの追加のプロパティを次に示します。

MongoDB URL

お使いの MongoDB または MongoDB Atlas データストアの URL を入力します。

- MongoDB の場合: mongodb://host:port/database ホストは、ホスト名、IP アドレス、UNIX ドメインソケットのいずれかにすることができます。接続文字列でポートが指定されていない場合は、デフォルトの MongoDB ポート、27017 が使用されます。
- MongoDB Atlas: mongodb+srv://server.example.com/database ホストは、DNS SRV レコードに対応するホスト名にすることができます。SRV 形式ではポートは不要であり、デフォルトの MongoDB ポート、27017 が使用されます。

ユーザーネーム

> **Note**
>
> ユーザー名とパスワードを直接入力するのではなく、AWS シークレットを使用して接続認証情報を保存することをお勧めします。詳細については、「での接続認証情報の保存 AWS Secrets Manager」を参照してください。

JDBC データストアにアクセスする権限を持つユーザー名を指定します。

パスワード

MongoDB または MongoDB Atlas へのアクセス権限を持つユーザー名のパスワードを入力します。

Salesforce 接続プロパティ

Salesforce 接続タイプの追加プロパティを次に示します。

- ENTITY_NAME(文字列) – (必須) 読み取り / 書き込みに使用されます。Salesforce のオブジェクトの名前。
- API_VERSION(文字列) – (必須) 読み取り / 書き込みに使用されます。使用する Salesforce Rest API バージョン。
- SELECTED_FIELDS(List<String>) - Default: empty(SELECT *). 読み込みに使用されます。オブジェクトに選択する列。
- FILTER_PREDICATE(文字列) – デフォルト: 空 読み込みに使用されます。Spark SQL 形式である必要があります。
- QUERY(文字列) – デフォルト: 空 読み込みに使用されます。完全な Spark SQL クエリ。
- PARTITION_FIELD(文字列) – 読み取りに使用されます。クエリをパーティション化するために使用するフィールド。
- LOWER_BOUND(文字列) - 読み取りに使用されます。選択したパーティションフィールドの包括的な下限値。
- UPPER_BOUND(文字列) – 読み取りに使用されます。選択したパーティションフィールドの排他的上限値。
- NUM_PARTITIONS(整数) – デフォルト: 1。読み込みに使用されます。読み取り用のパーティションの数。

- IMPORT_DELETED_RECORDS (文字列) - デフォルト: FALSE。読み込みに使用されます。クエリ中に削除レコードを取得するには。
- WRITE_OPERATION(文字列) - デフォルト: INSERT。書き込みに使用されます。値は INSERT、UPDATE、UPSERT、DELETE である必要があります。
- ID_FIELD_NAMES(文字列) - デフォルト: null。UPSERT にのみ使用されます。

Snowflake 接続

次のプロパティは、AWS Glue ETL ジョブで用いる Snowflake 接続を設定するために使用します。Snowflake をクローリングする場合は、JDBC 接続を使用します。

Snowflake URL

Snowflake エンドポイントの URL。Snowflake エンドポイントの URL の詳細については、Snowflake ドキュメントの「アカウントへの接続」を参照してください。

AWS シークレット

AWS Secrets Manager シークレットのシークレット名。AWS Glue は、シークレットの sfUser キーと sfPassword キーを使用して Snowflake に接続します。

Snowflake ロール (オプション)

AWS Glue が接続時に使用する Snowflake セキュリティロール。

AWS PrivateLink を使用して、Amazon VPC でホストされている Snowflake エンドポイントへの接続を設定する場合は、次のプロパティを使用します。

VPC

データストアを含むVirtual Private Cloud (VPC) の名前を選択します。AWS Glue コンソールには、現在のリージョンの VPC がすべて表示されます。

サブネット

データストアを含む VPC 内のサブネットを選択します。AWS Glue コンソールには VPC 内のデータストアのすべてのサブネットが一覧表示されます。

セキュリティグループ

データストアに関連付けられているセキュリティグループを選択します。AWS Glue には、AWS Glue の接続を許可するインバウンドソースルールを持つ 1 つ以上のセキュリティグループが必要

です。AWS Glue コンソールには、VPC へのインバウンドアクセスが許可されているすべてのセキュリティグループが一覧表示されます。AWS Glue は、VPC サブネットにアタッチされている Elastic Network Interface にこれらのセキュリティグループを関連付けます。

Vertica 接続

AWS Glue ETL ジョブのために Vertica 接続を設定するには、次のプロパティを使用します。

Vertica ホスト

Vertica インストールのホスト名。

Vertica ポート

Vertica インストールに使用できるポート。

AWS シークレット

AWS Secrets Manager シークレットの [シークレット名]。AWS Glue は、シークレットのキーを使用して Vertica に接続します。

Amazon VPC でホストされている Vertica エンドポイントへの接続を設定する場合は、次のプロパティを使用します。

VPC

データストアを含むVirtual Private Cloud (VPC) の名前を選択します。AWS Glue コンソールには、現在のリージョンの VPC がすべて表示されます。

サブネット

データストアを含む VPC 内のサブネットを選択します。AWS Glue コンソールには VPC 内のデータストアのすべてのサブネットが一覧表示されます。

セキュリティグループ

データストアに関連付けられているセキュリティグループを選択します。AWS Glue には、AWS Glue の接続を許可するインバウンドソースルールを持つ 1 つ以上のセキュリティグループが必要です。AWS Glue コンソールには、VPC へのインバウンドアクセスが許可されているすべてのセキュリティグループが一覧表示されます。AWS Glue は、VPC サブネットにアタッチされている Elastic Network Interface にこれらのセキュリティグループを関連付けます。

SAP HANA 接続

AWS Glue ETL ジョブのために SAP HANA 接続を設定するには、次のプロパティを使用します。

SAP HANA URL

SAP JDBC URL。

SAP HANA JDBC URL の形式は `jdbc:sap://saphanaHostname:saphanaPort/?databaseName=saphanaDBname,ParameterNa` です

AWS Glue には次の JDBC URL パラメータが必要です。

- `databaseName` – 接続先の SAP HANA のデフォルトデータベース。

AWS シークレット

AWS Secrets Manager シークレットの [シークレット名]。AWS Glue は、シークレットのキーを使用して SAP HANA に接続します。

Amazon VPC でホストされている SAP HANA エンドポイントへの接続を設定する場合は、次のプロパティを使用します。

VPC

データストアを含むVirtual Private Cloud (VPC) の名前を選択します。AWS Glue コンソールには、現在のリージョンの VPC がすべて表示されます。

サブネット

データストアを含む VPC 内のサブネットを選択します。AWS Glue コンソールには VPC 内のデータストアのすべてのサブネットが一覧表示されます。

セキュリティグループ

データストアに関連付けられているセキュリティグループを選択します。AWS Glue には、AWS Glue の接続を許可するインバウンドソースルールを持つ 1 つ以上のセキュリティグループが必要です。AWS Glue コンソールには、VPC へのインバウンドアクセスが許可されているすべてのセキュリティグループが一覧表示されます。AWS Glue は、VPC サブネットにアタッチされている Elastic Network Interface にこれらのセキュリティグループを関連付けます。

Azure SQL 接続

AWS Glue ETL ジョブのために Azure SQL 接続を設定するには、次のプロパティを使用します。

Azure SQL URL

Azure SQL エンドポイントの JDBC URL。

URL は、次のような形式になります:
`jdbc:sqlserver://databaseServerName:databasePort;databaseName=azuresqlDBname;`。

AWS Glue には次の URL プロパティが必要です。

- `databaseName` – 接続先の Azure SQL のデフォルトデータベース。

Azure SQL Managed Instances の JDBC URL の詳細については、Microsoft のドキュメントを参照してください。

AWS シークレット

AWS Secrets Manager シークレットの [シークレット名]。AWS Glue は、シークレットのキーを使用して Azure SQL に接続します。

Teradata Vantage 接続

AWS Glue ETL ジョブのために Teradata Vantage 接続を設定するには、次のプロパティを使用します。

Teradata URL

Teradata インスタンスに接続するには、データベースインスタンスのホスト名と、関連する Teradata パラメータを指定します。

`jdbc:teradata://teradataHostname/ParameterName=ParameterValue,ParameterName=Pa`

AWS Glue では、次の JDBC URL パラメータをサポートしています。

- `DATABASE_NAME` – 接続先の Teradata のデフォルトデータベース。
- `DBS_PORT` – 標準でない場合は、Teradata ポートを指定します。

AWS シークレット

AWS Secrets Manager シークレットの [シークレット名]。AWS Glue は、シークレットのキーを使用して Teradata Vantage に接続します。

Amazon VPC でホストされている Teradata Vantage エンドポイントへの接続を設定する場合は、次のプロパティを使用します。

VPC

データストアを含むVirtual Private Cloud (VPC) の名前を選択します。AWS Glue コンソールには、現在のリージョンの VPC がすべて表示されます。

サブネット

データストアを含む VPC 内のサブネットを選択します。AWS Glue コンソールには VPC 内のデータストアのすべてのサブネットが一覧表示されます。

セキュリティグループ

データストアに関連付けられているセキュリティグループを選択します。AWS Glue には、AWS Glue の接続を許可するインバウンドソースルールを持つ 1 つ以上のセキュリティグループが必要です。AWS Glue コンソールには、VPC へのインバウンドアクセスが許可されているすべてのセキュリティグループが一覧表示されます。AWS Glue は、VPC サブネットにアタッチされている Elastic Network Interface にこれらのセキュリティグループを関連付けます。

OpenSearch Service 接続

AWS Glue ETL ジョブのために OpenSearch Service 接続を設定するには、次のプロパティを使用します。

ドメインエンドポイント

Amazon OpenSearch Service ドメインエンドポイントのデフォルト形式は、https://search-*domainName-unstructuredIdContent*.*region*.es.amazonaws.com です。ドメインエンドポイントの識別の詳細については、Amazon OpenSearch Service ドキュメントの「Amazon OpenSearch Service ドメインの作成と管理」を参照してください。

ポート

エンドポイントでポートが開きます。

AWS シークレット

AWS Secrets Manager シークレットの [シークレット名]。AWS Glue は、シークレットのキーを使用して OpenSearch Service に接続します。

Amazon VPC でホストされている OpenSearch Service エンドポイントへの接続を設定する場合は、次のプロパティを使用します。

VPC

データストアを含むVirtual Private Cloud (VPC) の名前を選択します。AWS Glue コンソールには、現在のリージョンの VPC がすべて表示されます。

サブネット

データストアを含む VPC 内のサブネットを選択します。AWS Glue コンソールには VPC 内のデータストアのすべてのサブネットが一覧表示されます。

セキュリティグループ

データストアに関連付けられているセキュリティグループを選択します。AWS Glue には、AWS Glue の接続を許可するインバウンドソースルールを持つ 1 つ以上のセキュリティグループが必要です。AWS Glue コンソールには、VPC へのインバウンドアクセスが許可されているすべてのセキュリティグループが一覧表示されます。AWS Glue は、VPC サブネットにアタッチされている Elastic Network Interface にこれらのセキュリティグループを関連付けます。

Azure Cosmos 接続

AWS Glue ETL ジョブのために Azure Cosmos 接続を設定するには、次のプロパティを使用します。

Azure Cosmos DB アカウントのエンドポイント URI

Azure Cosmos への接続に使用するエンドポイント。詳細については、Azure のドキュメントを参照してください。

AWS シークレット

AWS Secrets Manager シークレットの [シークレット名]。AWS Glue は、シークレットのキーを使用して Azure Cosmos に接続します。

AWS Glue SSL 接続プロパティ

[Require SSL connection] (SSL 接続が必要) プロパティの詳細は、次の通りです。

SSL 接続が必要でない場合、AWS Glue では SSL を使用してデータストアへの接続を暗号化できなくても無視されます。設定の手順については、データストアのドキュメントを参照してください。こ

のオプションを選択すると、AWS Glue が接続できない場合、開発エンドポイントでのジョブ実行、クローラー、または ETL のステートメントは失敗します。

> **Note**
>
> Snowflake はデフォルトで SSL 接続をサポートしているため、このプロパティは Snowflake には適用されません。

このオプションは、AWS Glue クライアント側で検証されます。JDBC 接続の場合、AWS Glue は、証明書とホスト名の検証を使用する SSL 経由でのみ接続を行います。SSL 接続サポートは、次の場合に利用できます。

- Oracle Database
- Microsoft SQL Server
- PostgreSQL
- Amazon Redshift
- MySQL (Amazon RDS インスタンスのみ)
- Amazon Aurora MySQL (Amazon RDS インスタンスのみ)
- Amazon Aurora PostgreSQL (Amazon RDS インスタンスのみ)
- Kafka (Amazon Managed Streaming for Apache Kafka を含む)
- MongoDB

> **Note**
>
> Amazon RDS Oracle データストアが [Require SSL connection] (SSL 接続が必要) を使用できるようにするためには、Oracle インスタンスへのオプショングループを作成してアタッチする必要があります。
>
> 1. AWS Management Console にサインインし、Amazon RDS コンソール https://console.aws.amazon.com/rds/ を開きます。
> 2. オプショングループを Amazon RDS Oracle インスタンスに追加します。Amazon RDS コンソールで、オプショングループを追加する方法についての詳細は、「オプショングループを作成する」を参照してください。

3. オプションを SSL のオプショングループに追加します。SSL に指定したポートは、後で Amazon RDS Oracle インスタンスの AWS Glue JDBC 接続 URL を作成するときに使用されます。Amazon RDS コンソールでオプションを追加する方法の詳細については、Amazon RDS ユーザーガイドの「オプショングループにオプションを追加する」を参照してください。の。Oracle SSL のオプションの詳細については、Amazon RDS ユーザーガイドの「Oracle SSL」を参照してください。
4. AWS Glue コンソールで、Amazon RDS Oracle インスタンスへの接続を作成します。接続定義で、[Require SSL connection] を選択します。Amazon RDS Oracle SSL オプションで使用しているポートを、求められれば入力します。

接続に [Require SSL connection] (SSL 接続が必要) が選択されている場合は、次の追加のオプションプロパティを使用できます。

S3 のカスタム JDBC 証明書

オンプレミスまたはクラウドデータベースとの SSL 通信に現在使用している証明書がある場合は、この証明書を AWS Glue データソースまたはターゲットへの SSL 接続に使用できます。カスタムルート証明書が含まれている Amazon Simple Storage Service (Amazon S3) の場所を入力します。AWS Glue はこの証明書を使用してデータベースへの SSL 接続を確立します。AWS Glue は X.509 証明書のみを処理します。証明書は DER エンコードし、Base64 エンコード PEM 形式で指定する必要があります。

このフィールドを空白のままにすると、デフォルトの証明書が使用されます。

カスタム JDBC 証明書文字列

JDBC データベース固有の証明書情報を入力します。これは、ドメインの一致または識別名 (DN) の一致に使用される文字列です。Oracle Database の場合、この文字列は `tnsnames.ora` ファイルの security セクションの `SSL_SERVER_CERT_DN` パラメータにマッピングされます。Microsoft SQL Server の場合、この文字列は `hostNameInCertificate` として使用されます。

Oracle Database の `SSL_SERVER_CERT_DN` パラメータの例を次に示します。

```
cn=sales,cn=OracleContext,dc=us,dc=example,dc=com
```

Kafka のプライベート CA 証明書の場所

Kafka データストアとの SSL 通信に現在使用している証明書がある場合は、この証明書を AWS Glue接続に使用できます。このオプションは Kafka データストアには必須で、Amazon Managed Streaming for Apache Kafka データストアにはオプションです。カスタムルート証明書が含まれている Amazon Simple Storage Service (Amazon S3) の場所を入力します。AWS Glue はこの証明書を使用して Kafka データストアへの SSL 接続を確立します。AWS Glue は X.509 証明書のみを処理します。証明書は DER エンコードし、Base64 エンコード PEM 形式で指定する必要があります。

証明書の検証をスキップ

「証明書の検証をスキップ」 チェックボックスをオンにして、AWS Glue によるカスタム証明書の検証をスキップします。検証することを選択すると、AWS Glue は証明書の署名アルゴリズムとサブジェクトパブリックキーアルゴリズムを検証します。証明書の検証に失敗すると、この接続を使用する ETL ジョブまたはクローラーはすべて失敗します。

許可される署名アルゴリズムは、SHA256withRSA、SHA384withRSA、または SHA512withRSA のみです。サブジェクトパブリックキーアルゴリズムの場合、キーの長さは 2048 以上にする必要があります。

Kafka クライアントのキーストアの場所

Kafka クライアント側認証用のクライアントキーストアファイルの Amazon S3 の場所。パスは s3://bucket/prefix/filename.jks の形式である必要があります。ファイル名と .jks 拡張子で終わる必要があります。

Kafka クライアントキーストアのパスワード (オプション)

指定されたキーストアにアクセスするためのパスワード。

Kafka クライアントキーのパスワード (オプション)

キーストアは、複数のキーで構成することができるので、これは、Kafkaサーバー側のキーで使用されるクライアントキーにアクセスするためのパスワードです。

クライアント認証用の Apache Kafka 接続プロパティ

AWS Glue は、Apache Kafka 接続を作成するときの認証用の Simple Authentication and Security Layer (SASL) フレームワークをサポートしています。SASL フレームワークはさまざまな認証メカニズムをサポートしており、AWS Glue は SCRAM プロトコル (ユーザー名およびパスワード)、GSSAPI プロトコル (Kerberos プロトコル)、PLAIN プロトコルを提供します。

AWS Glue Studio を使用して、次のクライアント認証方法のいずれかを設定します。詳細については、「AWS Glue Studio ユーザーガイド」の「コネクタ用の接続を作成する」を参照してください。

- None - 認証なし。これは、テスト目的で接続する場合に便利です。
- SASL/SCRAM-SHA-512 - この認証方法を選択すると、認証情報を指定することができます。2 つのオプションがあります。
  - AWS Secrets Manager を使用 (推奨) - このオプションを選択すると、ユーザー名とパスワードを AWS Secrets Manager に保存して、必要なときに AWS Glue がアクセスできるようにできます。SSL または SASL 認証の認証情報を格納するシークレットを指定します。詳細については、「での接続認証情報の保存 AWS Secrets Manager」を参照してください。
  - 有効なユーザー名とパスワードを直接指定します。
- SASL/GSSAPI (Kerberos) - このオプションを選択すると、キータブファイル、krb5.conf ファイルの場所を選択して、Kerberos プリンシパル名と Kerberos サービス名を入力することができます。キータブファイルと krb5.conf ファイルの場所は、Simple Storage Service (Amazon S3) がある場所の中にする必要があります。MSK は SALS/GSSAPI をまだサポートしていないため、このオプションは Customer Managed Apache Kafka クラスターでのみ使用できます。詳細については、「MIT Kerberos ドキュメント: キータブ」を参照してください。
- SASL/PLAIN - 認証用の認証情報を指定するためにこの認証方法を選択します。2 つのオプションがあります。
  - AWS Secrets Manager を使用 (推奨) - このオプションを選択すると、認証情報が AWS Secrets Manager に保存され、必要になった際に AWS Glue からその情報へアクセスすることを許可できます。SSL または SASL 認証の認証情報を格納するシークレットを指定します。
  - ユーザー名とパスワードを直接指定します。
- SSL クライアント認証 - このオプションを選択すると、Simple Storage Service (Amazon S3) を参照することで Kafka クライアントキーストアの場所を選択できます。オプションで、Kafka クライアントキーストアのパスワードと Kafka クライアントキーのパスワードを入力できます。

Google BigQuery 接続

次のプロパティは、AWS Glue ETL ジョブで用いる Google BigQuery 接続を設定するために使用します。詳細については、「the section called “BigQuery 接続”」を参照してください。

AWS シークレット

AWS Secrets Manager にあるシークレットの[シークレット名]。 AWS GlueETL ジョブは、ユーザーのシークレットの `credentials` キーを使用して Google BigQuery に接続します。

Vertica 接続

次のプロパティは、AWS Glue ETL ジョブで使用される Vertica 接続を設定するために使用します。詳細については、「the section called “Vertica 接続”」を参照してください。

での接続認証情報の保存 AWS Secrets Manager

データストアの接続認証情報を指定 AWS Secrets Manager するために を使用することをお勧めします。この方法で Secrets Manager を使用すると、 はETLジョブとクローラー実行のランタイムにシークレット AWS Glue にアクセスし、認証情報を安全に保つことができます。

前提条件

で Secrets Manager を使用するには AWS Glue、シークレット値を取得するアクセス許可IAMをロール AWS Glueに付与する必要があります。 AWS マネージドポリシーには AWS Secrets Manager アクセス許可`AWSGlueServiceRole`は含まれていません。IAM ポリシーの例については、AWS Secrets Manager「 ユーザーガイド」の「例: シークレット値を取得するアクセス許可」を参照してください。

ネットワーク設定によっては、 VPCと Secrets Manager 間のプライベート接続を確立するためにVPCエンドポイントを作成する必要がある場合もあります。詳細については、VPC「エンドポイントの使用 AWS Secrets Manager」を参照してください。

のシークレットを作成するには AWS Glue

1. 「AWS Secrets Manager ユーザーガイド」の「Create and manage secrets」(シークレットの作成と管理) の手順に従ってください。次の例は、 のシークレットを作成するときに、プレーンテキストタブで認証情報を指定する方法JSONを示しています AWS Glue。

```
{
  "username": "EXAMPLE-USERNAME",
  "password": "EXAMPLE-PASSWORD"
}
```

2. AWS Glue Studio インターフェイスを使用してシークレットを接続に関連付けます。手順の詳細については、「AWS Glue Studio ユーザーガイド」の「コネクタ用の接続を作成する」を参照してください。

AWS Glue 接続の追加

AWS Glue for Spark のデータソースにはプログラム的に接続できます。詳細については、AWS Glue for Spark での ETL の接続タイプとオプションを参照してください。

また、AWS Glue コンソールを使用して、接続の追加、編集、削除、およびテストを行うことができます。AWS Glue 接続の詳細については、「データへの接続」を参照してください。

トピック

- Adobe Analytics に対する接続
- Adobe Marketo Engage への接続
- AWS Glue Studio の Amazon Redshift に接続する
- Asana に対する接続
- AWS Glue Studio での Azure Cosmos DB に対する接続
- AWS Glue Studio での Azure SQL に対する接続
- Blackbaud Raiser's Edge NXT への接続
- CircleCI への接続
- Datadog に対する接続
- Docusign Monitor への接続
- Domo への接続
- Dynatrace への接続
- Facebook 広告への接続
- Facebook Page Insights に対する接続
- Freshdesk に対する接続
- Freshsales に対する接続
- Google 広告に対する接続
- Google Analytics 4 への接続

- AWS Glue Studio で Google BigQuery に接続する
- Google 検索コンソールに対する接続
- Google Sheets への接続
- への接続 HubSpot
- Instagram 広告への接続
- AWS Glue Studio での Intercom への接続
- Jira Cloud への接続
- Kustomer への接続
- LinkedIn に対する接続
- Mailchimp への接続
- Microsoft Teams への接続
- Mixpanel に対する接続
- Monday への接続
- AWS Glue Studio での MongoDB に対する接続
- Oracle への接続 NetSuite
- AWS Glue Studio での OpenSearch Service に対する接続
- Okta への接続
- PayPal に対する接続
- Pendo への接続
- Pipedrive への接続
- Productboard への接続
- QuickBooks に対する接続
- Salesforce への接続
- Salesforce Marketing Cloud への接続
- Salesforce Commerce Cloud への接続
- Salesforce Marketing Cloud Account Engagement への接続
- AWS Glue Studio での SAP HANA に対する接続
- SAP OData への接続
- SendGrid に対する接続

- への接続 ServiceNow
- AWS Glue Studio での Slack への接続
- Smartsheet に対する接続
- AWS Glue Studio での Snapchat 広告への接続
- AWS Glue Studio での Snowflake への接続
- AWS Glue Studio での Stripe への接続
- AWS Glue Studio での Teradata Vantage に対する接続
- Twilio に対する接続
- AWS Glue Studio での Vertica に対する接続
- WooCommerce に対する接続
- Zendesk への接続
- Zoho CRM への接続
- Zoom Meetings に対する接続
- 独自の JDBC ドライバーを使用した JDBC 接続の追加

Adobe Analytics に対する接続

Adobe Analytics は、カスタマージャーニーをサポートするマルチチャネルデジタルエクスペリエンスからデータを収集し、データを分析するためのツールを提供する堅牢なデータ分析プラットフォームです。これは、マーケティング担当者やビジネスアナリストがビジネス分析の目的で一般的に使用するプラットフォームです。Adobe Analytics ユーザーの場合は、Adobe Analytics アカウントに AWS Glue を接続できます。その後、Adobe Analytics を ETL ジョブのデータソースとして使用できます。これらのジョブを実行して、Adobe Analytics と AWS サービス、またはその他のサポートされているアプリケーション間でデータを転送します。

トピック

- Adobe Analytics の AWS Glue サポート
- 接続を作成および使用するための API オペレーションを含むポリシー
- Adobe Analytics の設定
- Adobe Analytics 接続の設定
- Adobe Analytics エンティティからの読み取り

- Adobe Analytics の接続オプション
- Adobe Analytics アカウントの作成
- 制限

Adobe Analytics の AWS Glue サポート

AWS Glue は、次のように Adobe Analytics をサポートします。

ソースとしてサポートされていますか?

はい。AWS Glue ETL ジョブを使用して、Adobe Analytics からデータをクエリできます。

ターゲットとしてサポートされていますか?

いいえ。

サポートされている Adobe Analytics API バージョン

v2.0

接続を作成および使用するための API オペレーションを含むポリシー

次のサンプル ポリシーで、接続の作成と使用に必要な AWS の権限について説明します。新しいロールを作成する場合は、以下を含むポリシーを作成します。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:ListConnectionTypes",
                "glue:DescribeConnectionType",
                "glue:RefreshOAuth2Tokens",
                "glue:ListEntities",
                "glue:DescribeEntity"
            ],
            "Resource": "*"
        }
    ]
```

```
}
```

上記の方法を使用しない場合は、代わりに次のマネージド IAM ポリシーを使用します。

- AWSGlueServiceRole – さまざまな AWS Glue プロセスを実行するために必要なリソースへのアクセス権をユーザーに代わって付与します。これらのリソースには、AWS Glue、Amazon S3、IAM、CloudWatch Logs、および Amazon EC2 が含まれます。このポリシーで指定されたリソースの命名規則に従った場合、AWS Glue プロセスは必要なアクセス権限を使用できます。このポリシーは、通常、クローラ、ジョブ、開発エンドポイントを定義するときに指定されたロールにアタッチされます。
- AWSGlueConsoleFullAccess — ポリシーがアタッチされているアイデンティティが AWS マネージメントコンソールを使用するときは、AWS Glue リソースへのフルアクセスを許可します。このポリシーで指定されたリソースの命名規則に従った場合、ユーザーは完全なコンソール機能を使用できます。このポリシーは、通常 AWS Glue コンソールのユーザーにアタッチされています。

Adobe Analytics の設定

AWS Glue を使用して Adobe Analytics からデータを転送できるようにするには、次の要件を満たす必要があります。

最小要件

- E メールとパスワードを持つ Adobe Analytics アカウントがある。アカウントの作成の詳細については、「Adobe Analytics アカウントの作成」を参照してください。
- Adobe Analytics アカウントで API アクセスが有効になっている。Select、Prime、Ultimate エディションでは、API アクセスがデフォルトで有効になっています。

これらの要件を満たすと、Adobe Analytics アカウントに AWS Glue を接続する準備が整います。一般的な接続では、Adobe Analytics で他に何もする必要はありません。

Adobe Analytics 接続の設定

Adobe Analytics は、`OAuth2` の `AUTHORIZATION_CODE` グラントタイプをサポートしています。

このグラントタイプは、ユーザーを認証するためにサードパーティーの認証サーバーにユーザーをリダイレクトすることから、「three-legged」の `OAuth` と見なされます。ユーザーは、Adobe Analytics で独自の接続アプリを作成し、AWS Glue コンソールを介して接続を作成するときに独自

のクライアント ID とクライアントシークレットを指定することを選択できます。このシナリオでは、引き続き Adobe Analytics にリダイレクトされてログインし、リソースへアクセスするために AWS Glue を承認します。

このグラントタイプは、更新トークンとアクセストークンになります。アクセストークンの有効期間は短く、更新トークンを使用してユーザーとやり取りすることなく自動的に更新される場合があります。

AUTHORIZATION_CODE OAuth フローの接続アプリケーションの作成に関する Adobe Analytics の公開ドキュメントについては、「Adobe Analytics API」を参照してください。

Adobe Analytics 接続を設定するには：

1. AWS Secrets Manager で、次の詳細を含むシークレットを作成します。

   カスタマーマネージド接続アプリケーションの場合 – シークレットには、`USER_MANAGED_CLIENT_APPLICATION_CLIENT_SECRET` を使用して接続されたアプリケーションのコンシューマーシークレットをキーとして含める必要があります。

   > **Note**
   >
   > AWS Glue で接続ごとにシークレットを作成する必要があります。

2. AWS Glue Studio で、以下の手順に従って [データ接続] の下に接続を作成します。
   a. [接続タイプ] を選択する際に、[Adobe Analytics] を選択します。
   b. 接続先の Adobe Analytics の `x_api_key, instanceUrl` を指定します。
   c. AWS Glue が次のアクションを担い、その権限を持つことができる IAM ロールを選択します。

```
{
    "Version": "2012-10-17",
        "Statement": [
            {
                "Effect": "Allow",
                "Action": [
                    "secretsmanager:DescribeSecret",
                    "secretsmanager:GetSecretValue",
                    "secretsmanager:PutSecretValue",
                    "ec2:CreateNetworkInterface",
                    "ec2:DescribeNetworkInterface",
                    "ec2:DeleteNetworkInterface",
                ],
```

```
                "Resource": "*"
            }
        ]
    }
```

d. AWS Glue でこの接続に使用する `secretName` を選択して、トークンを配置します。

e. ネットワークを使用する場合は、ネットワークオプションを選択します。

3. AWS Glue ジョブに関連付けられている IAM ロールに `secretName` を読み取るアクセス許可を付与します。

Adobe Analytics エンティティからの読み取り

前提条件

読み取り元の Adobe Analytics オブジェクト。使用可能なエンティティを確認するには、以下のサポートされているエンティティの表を参照してください。

サポートされているエンティティ

エンティティ	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
注釈	はい	はい	はい	はい	いいえ
計算メトリクス	はい	はい	はい	はい	いいえ
計算メトリクス関数	はい	いいえ	いいえ	はい	いいえ
コンポーネントのメタデータ共有	はい	はい	いいえ	はい	いいえ
日付範囲	はい	はい	いいえ	はい	いいえ
ディメンション	はい	いいえ	いいえ	はい	いいえ

エンティティ	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
メトリクス	はい	いいえ	いいえ	はい	いいえ
プロジェクト	はい	いいえ	いいえ	はい	いいえ
レポート上位項目	はい	はい	いいえ	はい	いいえ
セグメント	はい	はい	はい	はい	いいえ
使用状況ログ	はい	はい	いいえ	はい	いいえ

例

```
adobeAnalytics_read = glueContext.create_dynamic_frame.from_options(
     connection_type="adobeanalytics",
     connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "annotation/ex*****",
        "API_VERSION": "v2.0"
 })
```

Adobe Analytics エンティティとフィールドの詳細

- 注釈
- 計算メトリクス
- コンポーネントのメタデータ
- 日付範囲
- ディメンション
- メトリクス
- プロジェクト
- レポート
- セグメント
- ユーザー

- 使用状況ログ

Adobe Analytics の接続オプション

Adobe Analytics の接続オプションは次のとおりです。

- `ENTITY_NAME`(文字列) – (必須) 読み取り / 書き込みに使用されます。Adobe Analytics のオブジェクトの名前。
- `API_VERSION`(文字列) – (必須) 読み取り / 書き込みに使用されます。使用する Adobe Analytics Rest API バージョン。例: v2.0。
- `X_API_KEY`(文字列) – (必須) 読み取り / 書き込みに使用されます。API にリクエストを行う開発者またはアプリケーションを認証する必要があります。
- `SELECTED_FIELDS`(List<String>) – Default: empty(SELECT *). 読み込みに使用されます。オブジェクトに選択する列。
- `FILTER_PREDICATE`(文字列) – デフォルト: 空 読み込みに使用されます。Spark SQL 形式である必要があります。
- `QUERY`(文字列) – デフォルト: 空 読み込みに使用されます。完全な Spark SQL クエリ。

Adobe Analytics アカウントの作成

1. Adobe パートナープログラムにアクセスして、交換パートナープログラムに登録します。
2. [Join Exchange Program] を選択します。
3. 会社の E メールアドレスを使用してアカウントを登録または作成します。
4. 提案ボックスから、Adobe Analytics 製品サブスクリプションを持つ適切な会社を選択します。
5. アカウントが、アクティブな Adobe Analytics サブスクリプションを持つ有効な組織 (利用可能なリストから) に登録されていることを確認します。
6. 会社管理者の承認後、承認 E メールのリンクをクリックしてアカウントをアクティブ化します。

作成したアカウントが Adobe Analytics サービスにアクセスできるかどうかの確認

1. Adobe 管理コンソールにログインします。
2. ページの右上隅にある組織名をチェックして、正しい会社にログインしていることを確認します。

3. [製品] を選択し、Adobe Analytics が使用可能かどうかを確認します。

> **Note**
>
> 利用可能な組織がない場合、または Adobe Analytics 製品がグレー表示または使用できない場合は、アカウントが組織に関連付けられていないか、アクティブな Adobe Analytics サブスクリプションがない可能性があります。システム管理者に連絡して、アカウントのこのサービスのアクセスをリクエストしてください。

プロジェクトと **OAuth2.0** 認証情報の作成

1. OAuth 2.0 アプリを作成する Adobe Analytics アカウントにログインします。
2. [プロジェクト] を選択して、[プロジェクトの新規作成] を選択します。
3. プロジェクトを追加するには、[プロジェクトに追加] を選択し、[API] を選択します。
4. [Adobe Analytics API] を選択します。
5. ユーザー認証として [OAUTH] を選択します。
6. OAUTH として [ウェブ] を選択し、リダイレクト URI を指定します。

   リダイレクト URI とそのパターンについては、以下を参照してください。

   - OAuth 2.0 デフォルトのリダイレクト URI – デフォルトのリダイレクト URI は、認証プロセス中に Adobe がアクセスするページの URL です。例えば、`https://ap-southeast-2.console.aws.amazon.com/appflow/oauth`
   - OAuth 2.0 リダイレクト URI パターン – リダイレクト URI パターンは、ログインフローが完了したときに Adobe がリダイレクト (リクエストされた場合) できる URI パス (またはカンマ区切りのパスのリスト) です。例えば、`https://ap-southeast-2\\.console\\.aws\\.amazon\\.com`
7. 以下のスコープを追加します。

   - `openid`
   - `read_organizations`
   - `additional_info.projectedProductContext`
   - `additional_info.job_function`
8. [認証情報を保存] を選択します。

9. アプリケーションを作成したら、Client ID および Client Secret の値をテキストファイルにコピーします。

制限

Adobe Analytics コネクタの制限事項は次のとおりです:

- Adobe Analytics は、フィールドベースおよびレコードベースのパーティショニングをサポートしていません。フィールドベースのパーティショニングは、パーティション化するフィールドをクエリできないため、サポートされていません。ページ分割の「オフセット」を取得するプロビジョニングがないため、レコードベースのパーティショニングはサポートできません。
- Report Top Item エンティティで、startDate および endDate クエリパラメータが期待どおりに機能していません。これらのパラメータに基づいてレスポンスがフィルタリングされていないため、このエンティティのフィルターと増分フローに問題が発生しています。
- Annotation、Calculated Metrics、Calculated Metrics Function、Date Ranges、Dimension、Metric、Project、Report Top Items、および Segment エンティティの場合、locale クエリパラメータは、レスポンスのローカライズされたセクションに使用する言語を指定し、レコードをフィルタリングしません。たとえば、locale="ja_JP" はデータを日本語で表示します。
- Report Top Item エンティティ – dateRange フィールドと lookupNoneValues フィールドのフィルターは現在機能していません。
- Segment エンティティ: フィルター値 includeType="templates" の場合、他のフィールドのフィルターは機能しません。
- Date Range エンティティ – curatedRsid フィールドのフィルターが機能していません。
- Metric entity エンティティ –「false」値を持つセグメント化可能なフィールドでフィルタリングすると、true 値と false 値の両方の結果が得られます。

Adobe Marketo Engage への接続

Adobe Marketo Engage は、マーケティング担当者が見込み客や顧客向けにパーソナライズされたマルチチャネルプログラムやキャンペーンを管理できるようにするマーケティング自動化プラットフォームです。

トピック

- Adobe Marketo Engage の AWS Glue サポート

- 接続を作成および使用するためのAPIオペレーションを含むポリシー
- Adobe Marketo Engage の設定
- Adobe Marketo Engage 接続の設定
- Adobe Marketo Engage エンティティからの読み取り
- Adobe Marketo Engage 接続オプション
- Adobe Marketo Engage コネクタの制限事項と注意事項

Adobe Marketo Engage の AWS Glue サポート

AWS Glue は、次のように Adobe Marketo Engage をサポートします。

ソースとしてサポートされていますか?

はい。AWS Glue ETL ジョブを使用して、Adobe Marketo Engage からデータをクエリできます。

ターゲットとしてサポートされていますか?

いいえ。

サポートされている Adobe Marketo Engage API バージョン

次の Adobe Marketo Engage API バージョンがサポートされています。

- v1

バージョン固有のエンティティサポートについては、「ソースに対応するエンティティ」を参照してください。

接続を作成および使用するためのAPIオペレーションを含むポリシー

次のサンプルポリシーでは、接続の作成と使用に必要な AWS IAMアクセス許可について説明します。新しいロールを作成する場合は、以下を含むポリシーを作成します。

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

```
            "Action": [
                "glue:ListConnectionTypes",
                "glue:DescribeConnectionType",
                "glue:RefreshOAuth2Tokens"
                "glue:ListEntities",
                "glue:DescribeEntity"
            ],
            "Resource": "*"
        }
    ]
}
```

上記の方法を使用しない場合は、次の管理IAMポリシーを使用します。

- AWSGlueServiceRole – さまざまな AWS Glue プロセスがユーザーに代わって実行するために必要なリソースへのアクセスを許可します。これらのリソースには AWS Glue、、Amazon S3、IAM、 CloudWatch Logs、Amazon が含まれますEC2。このポリシーで指定されたリソースの命名規則に従うと、 AWS Glue プロセスには必要なアクセス許可があります。このポリシーは、通常、クローラ、ジョブ、開発エンドポイントを定義するときに指定されたロールにアタッチされます。
- AWSGlueConsoleFullAccess - ポリシーがアタッチされている ID が AWS マネジメントコンソールを使用する場合、 AWS Glue リソースへのフルアクセスを許可します。このポリシーで指定されたリソースの命名規則に従った場合、ユーザーは完全なコンソール機能を使用できます。このポリシーは通常、 AWS Glue コンソールのユーザーにアタッチされます。

Adobe Marketo Engage の設定

AWS Glue を使用して Adobe Marketo Engage からデータを転送できるようにするには、以下の要件を満たす必要があります。

最小要件

以下に、最小要件を示します。

- クライアント認証情報を持つ Adobe Marketo Engage アカウントがあります。
- Adobe Marketo Engage アカウントは、有効なライセンスで API にアクセスできます。

これらの要件を満たしている場合は、Adobe Marketo Engage アカウント AWS Glue に接続できます。一般的な接続では、Adobe Marketo Engage で他に何もする必要はありません。

2.0 OAuth 認証情報の取得

インスタンスに認証された呼び出しを行えるようにAPI認証情報を取得するには、Adobe Marketo Engage デベロッパーガイドRESTAPIの「」を参照してください。

Adobe Marketo Engage 接続の設定

Adobe Marketo Engage は、OAuth2 の CLIENT CREDENTIALS グラントタイプをサポートしています。

- このグラントタイプは、クライアントがユーザーのコンテキスト外でアクセストークンを取得するために使用するため、2-legged の OAuth 2.0 と見なされます。AWS Glue は、クライアント ID とクライアントシークレットを使用して、定義したカスタムサービスによって提供される Adobe Marketo Engage API を認証できます。
- 各カスタムサービスは、特定のアクションを実行することをサービスに許可する一連のロールとアクセス許可を持つ API 専用ユーザーによって所有されます。アクセストークンは 1 つのカスタムサービスに関連付けられています。
- このグラントタイプは、有効期間が短いアクセストークンになり、ID エンドポイントを呼び出すことで更新される場合があります。
- クライアント認証情報を使用した OAuth 2.0 の Adobe Marketo Engage の公開ドキュメントについては、Adobe Marketo Engage デベロッパーガイドの「認証」を参照してください。

Adobe Marketo Engage 接続を設定するには：

1. AWS Secrets Manager で、次の詳細を含むシークレットを作成します。
   a. カスタマーマネージド接続アプリケーションの場合、シークレットには、`USER_MANAGED_CLIENT_APPLICATION_CLIENT_SECRET` を使用して接続されたアプリケーションのコンシューマーシークレットをキーとして含める必要があります。
   b. 注: AWS Glue で接続ごとにシークレットを作成する必要があります。
2. AWS Glue Glue Studio で、以下の手順に従って [データ接続] の下に接続を作成します。
   a. [接続タイプ] を選択する際に、[Adobe Marketo Engage] を選択します。
   b. 接続先の Adobe Marketo Engage インスタンスの `INSTANCE_URL` を指定します。
   c. AWS Glue が次のアクションを引き受けて、その権限を持つことができる AWS IAM ロールを選択します。

```
{
```

```
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:DescribeSecret",
                "secretsmanager:GetSecretValue",
                "secretsmanager:PutSecretValue",
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterface",
                "ec2:DeleteNetworkInterface",
            ],
            "Resource": "*"
        }
    ]
}
```

d. AWS Glue でこの接続に使用する `secretName` を選択して、トークンを配置します。

e. ネットワークを使用する場合は、ネットワークオプションを選択します。

3. AWS Glue ジョブに関連付けられている IAM ロールに `secretName` を読み取るアクセス許可を付与します。

Adobe Marketo Engage エンティティからの読み取り

前提条件

読み取り元の Adobe Marketo Engage オブジェクト。leads、activities、customobjects などのオブジェクト名が必要になります。次の表は、サポートされているエンティティを示しています。

ソース (同期) でサポートされているエンティティ:

エンティティ	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
leads	はい	はい	なし	あり	いいえ
アクティビティ	はい	はい	なし	あり	いいえ

エンティティ	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
customobjects	はい	はい	なし	あり	いいえ

ソース (非同期) でサポートされているエンティティ:

エンティティ	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
leads	はい	いいえ	いいえ	はい	はい
アクティビティ	はい	いいえ	いいえ	あり	いいえ
customobjects	はい	いいえ	いいえ	はい	はい

例:

```
adobe-marketo-engage_read = glueContext.create_dynamic_frame.from_options(
    connection_type="adobe-marketo-engage",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "leads",
        "API_VERSION": "v2",
        "INSTANCE_URL": "https://539-t**-6**.mktorest.com"
    }
```

Adobe Marketo Engage エンティティとフィールドの詳細:

静的メタデータを持つエンティティ:

エンティティ	フィールド	データ型	サポートされている演算子
アクティビティ	sinceDatetime (同期でのみサポート)	DateTime	>= (同期モードの場合のみ)
	createdAt (非同期でのみサポート)	DateTime	between (非同期モードのみ)
	activitiesTypeId	整数	=
	adobe-marketo-engageGUID	Long	= (同期モードの場合のみ)
	leadId	Long	該当なし
	activityDate	DateTime	該当なし
	campaignId	Long	該当なし
	primaryAttributeValueId	整数	該当なし
	primaryAttributeValue	String	該当なし
	属性	String	該当なし

動的メタデータを持つエンティティ:

以下のエンティティの場合、Adobe Marketo Engage はメタデータを動的にフェッチするエンドポイントを備えているため、演算子のサポートは各エンティティのデータ型レベルでキャプチャされます。

エンティティ	データ型	サポートされている演算子
leads	整数	= (同期モードの場合のみ)
	DateTime	between (非同期モードのみ)

エンティティ	データ型	サポートされている演算子
	String	= (同期モードの場合のみ)
	Long	該当なし
	ブール値	該当なし
	日付	該当なし
	浮動小数点数	該当なし
customobjects	整数	該当なし
	DateTime	between (非同期モードのみ)
	String	= (同期モードの場合のみ)
	日付	該当なし
	Long	該当なし
	ブール値	該当なし
	浮動小数点数	該当なし

パーティショニングクエリ

Spark で同時実行を使用する場合は、追加の Spark オプション `PARTITION_FIELD`、`LOWER_BOUND`、`UPPER_BOUND`、および `NUM_PARTITIONS` を指定できます。これらのパラメータを使用すると、元のクエリは Spark タスクで同時に実行できるサブクエリの `NUM_PARTITIONS` の数に分割されます。

- `PARTITION_FIELD`: クエリのパーティション化に使用するフィールドの名前。
- `LOWER_BOUND`: 選択したパーティションフィールドの包括的な下限値。

  DateTime フィールドでは、ISO 形式の値を受け入れます。

  有効な値の例:

```
"2024-07-01T00:00:00.000Z"
```

- UPPER_BOUND: 選択したパーティションフィールドの排他的上限値。
- NUM_PARTITIONS: パーティション数。

次の表に、エンティティパーティション分割フィールドのサポートの詳細を示します:

エンティティ名	パーティション分割フィールド	データ型
leads	createdAt	DateTime
	updateAt	DateTime
customobjects	updatedAt	DateTime

例:

```
adobe-marketo-engage_read = glueContext.create_dynamic_frame.from_options(
    connection_type="adobe-marketo-engage",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "leads",
        "API_VERSION": "v1",
        "PARTITION_FIELD": "createdAt"
        "LOWER_BOUND": "2024-07-01T00:00:00.000Z"
        "UPPER_BOUND": "2024-07-02T00:00:00.000Z"
        "NUM_PARTITIONS": "10"
    }
```

Adobe Marketo Engage 接続オプション

Adobe Marketo Engage の接続オプションは次のとおりです。

- ENTITY_NAME(文字列) – (必須) 読み取りに使用されます。Adobe Marketo Engage のオブジェクトの名前。
- API_VERSION(文字列) – (必須) 読み取りに使用されます。使用する Adobe Marketo Engage Rest API バージョン。例: v1。

- SELECTED_FIELDS(List<String>) – Default: empty(SELECT *). 読み込みに使用されます。オブジェクトに選択する列。
- FILTER_PREDICATE(文字列) – デフォルト: 空 読み込みに使用されます。Spark SQL 形式である必要があります。
- QUERY(文字列) - デフォルト: 空 読み込みに使用されます。完全な Spark SQL クエリ。
- PARTITION_FIELD(文字列) – 読み取りに使用されます。クエリをパーティション化するために使用するフィールド。
- LOWER_BOUND(文字列) - 読み取りに使用されます。選択したパーティションフィールドの包括的な下限値。
- UPPER_BOUND (文字列) - 読み取りに使用されます。選択したパーティションフィールドの排他的上限値。
- NUM_PARTITIONS(整数) – デフォルト: 1。読み込みに使用されます。読み取り用のパーティションの数。
- TRANSFER_MODE (文字列) – デフォルト: SYNC。非同期読み取りに使用されます。

Adobe Marketo Engage コネクタの制限事項と注意事項

Adobe Marketo Engage コネクタの制限事項または注意事項は次のとおりです。

- 'sinceDatetime' と 'activityTypeId' は、同期アクティビティエンティティの必須フィルターパラメータです。
- サブスクリプションには、1 日あたり 50,000 件の API コールが割り当てられます (中部標準時で毎日午前 12:00 にリセットされます)。Adobe Marketo Engage サブスクリプションの一部として、1 日あたりの容量を追加購入できます。
- 日付範囲フィルター (createdAt または updatedAt) の最大期間は 31 日間です。
- サブスクリプションには、いつでもキューに最大 10 個の一括抽出ジョブが割り当てられます。
- デフォルトでは、抽出ジョブは 1 日あたり 500 MB に制限されています (中部標準時で毎日午前 12:00 にリセットされます)。Adobe Marketo Engage サブスクリプションの一部として、1 日あたりの容量を追加購入できます。
- 同時エクスポートジョブの最大数は 2 です。
- キューに入れられたエクスポートジョブ (現在エクスポート中のジョブを含む) の最大数は 10 です。
- 一括ジョブから抽出できる最大許容ファイルサイズは 1 GB です。
- 非同期ジョブが作成されると、ファイル保持期間は有効期限の 7 日前になります。

- `createdAt` または `updatedAt` は、非同期リードエンティティの必須フィルターパラメータです。
- `createdAt` は、非同期アクティビティエンティティの必須フィルターパラメータです。
- `updatedAt` は、非同期カスタムオブジェクトエンティティの必須フィルターパラメータです。

詳細については、「Adobe Marketo Engage 統合ベストプラクティス」および「一括抽出」を参照してください。

AWS Glue Studio の Amazon Redshift に接続する

> **Note**
>
> AWS Glue for Spark を使用して AWS Glue Studio の Amazon Redshift データベース内のテーブルに対する読み取りと書き込みを行うことができます。AWS Glue ジョブで Amazon Redshift をプログラム的に設定する方法については、「Redshift 接続」を参照してください。

AWS Glue には、Amazon Redshift に対するサポートが組み込まれています。AWS Glue Studio には、Amazon Redshift に接続してデータ統合ジョブをオーサリングし、AWS Glue Studio の Spark ランタイム上でジョブをサーバーレスで実行できる、ビジュアルインターフェイスが用意されています。

トピック

- Amazon Redshift 接続の作成
- Amazon Redshift ソースノードの作成
- Amazon Redshift ターゲットノードの作成
- 詳細オプション

Amazon Redshift 接続の作成

必要となる許可

Amazon Redshift クラスターおよび Amazon Redshift サーバーレス環境を使用するには、追加の許可が必要です。ETL ジョブに許可を追加する方法の詳細については、「ETL ジョブに必要な IAM アクセス許可を確認する」を参照してください。

- redshift:DescribeClusters
- redshift-serverless:ListWorkgroups
- redshift-serverless:ListNamespaces

概要

Amazon Redshift 接続を追加するには、既存の Amazon Redshift 接続を選択するか、AWS Glue Studio でデータソース - Redshift ノードを追加するときに、新しい接続を作成します。

AWS Glue は、Amazon Redshift クラスターと Amazon Redshift サーバーレス環境の両方をサポートします。接続を作成すると、Amazon Redshift サーバーレス環境では、接続オプションの横に [serverless] ラベルが表示されます。

Amazon Redshift 接続を作成する方法について詳しくは、「Amazon Redshift との間でのデータの移動」を参照してください。

Amazon Redshift ソースノードの作成

必要となる許可

Amazon Redshift データソースを使用する AWS Glue Studio ジョブには追加の許可が必要です。ETL ジョブに許可を追加する方法の詳細については、「ETL ジョブに必要な IAM アクセス許可を確認する」を参照してください。

Amazon Redshift 接続を使用するには、次の許可が必要です。

- redshift-data:ListSchemas
- redshift-data:ListTables
- redshift-data:DescribeTable
- redshift-data:ExecuteStatement
- redshift-data:DescribeStatement
- redshift-data:GetStatementResult

Amazon Redshift データソースの追加

データソース — Amazon Redshift ノードを追加するには

1. Amazon Redshift アクセスタイプを選択します。

- 直接データ接続 (推奨) — Amazon Redshift データに直接アクセスする場合は、このオプションを選択してください。これは推奨されるオプションで、デフォルトでもあります。
- Data Catalog tables — 使用したいデータカタログテーブルがある場合は、このオプションを選択してください。

2. [直接データ接続] を選択した場合は、Amazon Redshift データソースへの接続を選択します。既に接続が存在していて、それらの接続から選択できることが前提です。接続の作成が必要な場合は、[Redshift 接続の作成] を選択します。詳細については、「コネクタと接続の使用に関する概要」を参照してください。

   接続を選択したあとは、[プロパティを表示] をクリックすると、接続のプロパティを表示できます。URL、セキュリティグループ、サブネット、アベイラビリティーゾーン、説明、作成時刻 (UTC) と最終更新時刻 (UTC) のタイムスタンプなど、接続についての情報が表示されます。

3. Amazon Redshift のソースオプションを選択してください。

   - [Choose a single table] - これは、単一の Amazon Redshift テーブルからアクセスするデータを含むテーブルです。
   - カスタムクエリを入力 — カスタムクエリに基づき、Amazon Redshift の複数のテーブルからデータセットにアクセスできます。

4. 単一のテーブルを選択した場合は、Amazon Redshift のスキーマを選択します。選択したテーブルによって、使えるスキーマのリストが決まります。

   または、[カスタムクエリを入力] を選択します。このオプションを選択すると、Amazon Redshift の複数のテーブルから、カスタムデータセットにアクセスできます。このオプションを選択した場合は、Amazon Redshift クエリを入力します。

   Amazon Redshift サーバーレス環境に接続するときに、カスタムクエリに次の許可を追加します。

   ```
   GRANT SELECT ON ALL TABLES IN <schema> TO PUBLIC
   ```

   [スキーマを推測] を選択すると、入力したクエリに基づいてスキーマを読み取ることができます。[Redshift クエリエディタを開く] を選択して、Amazon Redshift クエリを入力することもできます。詳細については、「クエリエディタを使用してデータベースのクエリを実行する」を参照してください。

5. [パフォーマンスとセキュリティ] で、Amazon S3 のステージングディレクトリと IAM ロールを選択します。

   - Amazon S3 ステージングディレクトリ — データを一時的にステージングする、Amazon S3 の場所を選択します。
   - IAM ロール — 選択した Amazon S3 に書き込める IAM ロールを選択します。

6. [カスタム Redshift パラメータ - オプション] に、パラメータと値を入力します。

Amazon Redshift ターゲットノードの作成

必要となる許可

Amazon Redshift データターゲットを使用する AWS Glue Studio ジョブには追加の許可が必要です。ETL ジョブに許可を追加する方法の詳細については、「ETL ジョブに必要な IAM アクセス許可を確認する」を参照してください。

Amazon Redshift 接続を使用するには、次の許可が必要です。

- redshift-data:ListSchemas
- redshift-data:ListTables

Amazon Redshift ターゲットノードの追加

Amazon Redshift ターゲットノードを作成するには

1. 既存の Amazon Redshift テーブルをターゲットとして選択するか、新しいテーブル名を入力します。
2. データターゲット - Redshift ターゲットノードを使用する場合、次のオプションから選択できます。

   - APPEND — テーブルがすでに存在している場合は、新しいデータをテーブルにすべて挿入してダンプします。テーブルが存在しない場合は、新規にテーブルを作成し、そこに新しいデータをすべて挿入します。

     またターゲットのテーブルで、既存のレコードを更新 (UPSERT) する場合は、このチェックボックスにチェックを入れてください。まずテーブルが存在している必要があります。存在しない場合には、操作は失敗します。

- MERGE — 指定した条件に基づいて、AWS Glue がターゲットとなるテーブルのデータを更新するか、またはデータを追加します。

  > **Note**
  >
  > AWS Glue でマージアクションを使用するには、Amazon Redshift のマージ機能を有効にする必要があります。Amazon Redshift インスタンスのマージを有効にする方法については、「MERGE (プレビュー)」を参照してください。

  [オプション] を選択します。

  - キーと簡単なアクションの選択 — ソースデータとターゲットデータセットとの、マッチングキーとして使用する列を選択します。

    一致した場合、次のオプションを指定します。

    - ターゲットのデータセットにあるレコードを、ソースのデータで更新します。
    - ターゲットのデータセットにあるレコードを削除します。

    一致しない場合、次のオプションを指定します。

    - ターゲットのデータセットに、新しい行としてソースデータを挿入します。
    - 何もしない。
  - カスタム MERGE ステートメントの入力 — その後 [MERGE ステートメントの検証] を選択し、ステートメントが有効か、無効かを検証できます。
- TRUNCATE — 既にテーブルが存在している場合は、ターゲットのテーブルの内容を削除してから、ターゲットのテーブルを削除します。削除が成功してから、すべてのデータを挿入します。テーブルが存在していない場合、テーブルを作成し、すべてのデータを挿入します。削除が成功しなかった場合、操作は失敗します。
- DROP — 既にテーブルが存在している場合は、テーブルのメタデータとデータを削除します。削除が成功してから、すべてのデータを挿入します。テーブルが存在していない場合、テーブルを作成し、すべてのデータを挿入します。削除が成功しなかった場合、操作は失敗します。
- CREATE — 新しいテーブルを、デフォルトの名前を使用して作成します。既にテーブル名が存在する場合は、接尾辞として `job_datetime` を名前につけたうえで新しいテーブルを作成し、一意性を保ちます。これで、すべてのデータが新しいテーブルに挿入されます。テーブ

ルが存在する場合、最終的なテーブル名には接尾辞が付きます。テーブルが存在しない場合、テーブルが作成されます。いずれの場合も、テーブルが新しく作成されます。

詳細オプション

「AWS Glue で Amazon Redshift Spark コネクターを使用する」を参照してください。

Asana に対する接続

Asana は、チームがタスクやプロジェクトを編成、計画、完了するのに役立つクラウドベースのチームコラボレーションソリューションです。Asana ユーザーの場合、アカウントにはワークスペース、プロジェクト、タスク、チームなどに関するデータが含まれます。Asana から特定の AWS サービスやその他のサポートされているアプリケーションにデータを転送できます。

トピック

- Asana の AWS Glue サポート
- 接続を作成および使用するための API オペレーションを含むポリシー
- Asana の設定
- Asana 接続の設定
- Asana エンティティからの読み取り
- Asana 接続オプション
- Asana アカウントの作成
- 制限

Asana の AWS Glue サポート

AWS Glue は、次のように Asana をサポートします。

ソースとしてサポートされていますか?

はい。AWS Glue ETL ジョブを使用して、Asana からデータをクエリできます。

ターゲットとしてサポートされていますか?

いいえ。

サポートされている Asana API バージョン

1.0

接続を作成および使用するための API オペレーションを含むポリシー

次のサンプル ポリシーで、接続の作成と使用に必要な AWS の権限について説明します。新しいロールを作成する場合は、以下を含むポリシーを作成します。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:ListConnectionTypes",
                "glue:DescribeConnectionType",
                "glue:RefreshOAuth2Tokens",
                "glue:ListEntities",
                "glue:DescribeEntity"
            ],
            "Resource": "*"
        }
    ]
}
```

上記の方法を使用しない場合は、代わりに次のマネージド IAM ポリシーを使用します。

- AWSGlueServiceRole – さまざまな AWS Glue プロセスを実行するために必要なリソースへのアクセス権をユーザーに代わって付与します。これらのリソースには、AWS Glue、Amazon S3、IAM、CloudWatch Logs、および Amazon EC2 が含まれます。このポリシーで指定されたリソースの命名規則に従った場合、AWS Glue プロセスは必要なアクセス権限を使用できます。このポリシーは、通常、クローラ、ジョブ、開発エンドポイントを定義するときに指定されたロールにアタッチされます。
- AWSGlueConsoleFullAccess — ポリシーがアタッチされているアイデンティティが AWS マネージメントコンソールを使用するときは、AWS Glue リソースへのフルアクセスを許可します。このポリシーで指定されたリソースの命名規則に従った場合、ユーザーは完全なコンソール機能を使用できます。このポリシーは、通常 AWS Glue コンソールのユーザーにアタッチされています。

Asana の設定

AWS Glue を使用して Asana から転送できるようにするには、次の要件を満たす必要があります。

最小要件

- E メールとパスワードが設定された Asana アカウントがある。アカウントの作成の詳細については、「Asana アカウントの作成」を参照してください。
- AWS Glue へのサービスアクセスで AWS アカウントを作成する必要があります。
- Asana アカウントに次のいずれかのリソースが作成されていることを確認します。
  - OAuth 2.0 認証をサポートするデベロッパーアプリ。詳細については、Asana 開発者ドキュメントの「OAuth」を参照してください。または「the section called “Asana アカウントの作成”」を参照してください。
  - 個人アクセストークン。詳細については、Asana 開発者ドキュメントの「個人用アクセストークンhttps://developers.asana.com/docs/personal-access-token」を参照してください。

これらの要件を満たすと、Adobe Analytics アカウントに AWS Glue を接続する準備が整います。一般的な接続では、Adobe Analytics で他に何もする必要はありません。

Asana 接続の設定

Asana は、OAuth2 の AUTHORIZATION_CODE グラントタイプをサポートしています。

このグラントタイプは、ユーザーを認証するためにサードパーティーの認証サーバーにユーザーをリダイレクトすることから、「three-legged」の OAuth と見なされます。ユーザーは、Asana で独自の接続アプリを作成し、AWS Glue コンソールを介して接続を作成するときに独自のクライアント ID とクライアントシークレットを指定することを選択できます。このシナリオでは、引き続き Asana にリダイレクトされてログインし、リソースへアクセスするために AWS Glue を承認します。

このグラントタイプは、更新トークンとアクセストークンになります。アクセストークンの有効期間は短く、更新トークンを使用してユーザーとやり取りすることなく自動的に更新される場合があります。

AUTHORIZATION_CODE OAuth フロー用の接続アプリケーションの作成に関する Asana の公開ドキュメントについては、「Asana API」を参照してください。

Asana 接続を設定するには：

1. AWS Secrets Manager で、次の詳細を含むシークレットを作成します。

- カスタマーマネージド接続アプリケーションの場合 – シークレットには、USER_MANAGED_CLIENT_APPLICATION_CLIENT_SECRET を使用して接続されたアプリケーションのコンシューマーシークレットをキーとして含める必要があります。
-

> **Note**
>
> AWS Glue で接続ごとにシークレットを作成する必要があります。

2. AWS Glue Studio で、以下の手順に従って [データ接続] の下に接続を作成します。
   a. [接続タイプ] を選択する際に、[Asana] を選択します。
   b. Asana 環境を指定します。
   c. AWS Glue が次のアクションを担い、その権限を持つことができる IAM ロールを選択します。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:DescribeSecret",
                "secretsmanager:GetSecretValue",
                "secretsmanager:PutSecretValue",
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterface",
                "ec2:DeleteNetworkInterface",
            ],
            "Resource": "*"
        }
    ]
}
```

   d. AWS Glue でこの接続に使用する secretName を選択して、トークンを配置します。
   e. ネットワークを使用する場合は、ネットワークオプションを選択します。
3. AWS Glue ジョブに関連付けられている IAM ロールに secretName を読み取るアクセス許可を付与します。

Asana エンティティからの読み取り

前提条件

読み取り元の Asana オブジェクト。使用可能なエンティティを確認するには、以下のサポートされているエンティティの表を参照してください。

ソースに対応するエンティティ

エンティティ	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
ワークスペース	いいえ	はい	いいえ	はい	いいえ
タグ	いいえ	はい	いいえ	はい	いいえ
ユーザー	いいえ	はい	いいえ	はい	いいえ
ポートフォリオ	いいえ	はい	いいえ	はい	いいえ
Team	いいえ	はい	いいえ	はい	いいえ
プロジェクト	はい	はい	いいえ	はい	いいえ
セクション	いいえ	はい	いいえ	はい	いいえ
タスク	はい	いいえ	いいえ	はい	はい
目標	はい	はい	いいえ	はい	いいえ
AuditLogEvent	はい	はい	いいえ	はい	いいえ
ステータスの更新	はい	はい	いいえ	はい	いいえ
カスタムフィールド	いいえ	はい	いいえ	はい	いいえ
プロジェクトの概要	はい	いいえ	いいえ	はい	はい

例

```
read_read = glueContext.create_dynamic_frame.from_options(
    connection_type="Asana",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "task/workspace:xxxx",
        "API_VERSION": "1.0",
        "PARTITION_FIELD": "created_at",
        "LOWER_BOUND": "2024-02-05T14:09:30.115Z",
        "UPPER_BOUND": "2024-06-07T13:30:00.134Z",
        "NUM_PARTITIONS": "3"
    }
```

Asana エンティティとフィールドの詳細

- ワークスペース
- タグ
- ユーザー
- ポートフォリオ
- チーム
- プロジェクト
- セクション
- タスク
- 目標
- AuditLogEvent
- ステータスの更新
- カスタムフィールド
- プロジェクトの概要

パーティショニングクエリ

Spark で同時実行を使用する場合は、追加の Spark オプション PARTITION_FIELD、LOWER_BOUND、UPPER_BOUND、および NUM_PARTITIONS を指定できます。これらのパラメータを使用すると、元のクエリは Spark タスクで同時に実行できるサブクエリの NUM_PARTITIONS の数に分割されます。

- PARTITION_FIELD: クエリのパーティション化に使用するフィールドの名前。
- LOWER_BOUND: 選択したパーティションフィールドの包括的な下限値。

  日付については、Spark SQL クエリで使用される Spark の日付形式を受け入れます。有効な値の例: 2024-06-07T13:30:00.134Z。
- UPPER_BOUND: 選択したパーティションフィールドの排他的上限値。
- NUM_PARTITIONS: パーティション数。

エンティティごとのパーティション分割フィールドのサポートの詳細は、次の表にまとめられています。

エンティティ名	Partitioning Field	データタイプ
タスク	created_at	DateTime
タスク	modified_at	DateTime

例

```
read_read = glueContext.create_dynamic_frame.from_options(
    connection_type="Asana",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "task/workspace:xxxx",
        "API_VERSION": "1.0",
        "PARTITION_FIELD": "created_at",
        "LOWER_BOUND": "2024-02-05T14:09:30.115Z",
        "UPPER_BOUND": "2024-06-07T13:30:00.134Z",
        "NUM_PARTITIONS": "3"
    }
```

Asana 接続オプション

Asana の接続オプションは次のとおりです。

- ENTITY_NAME(文字列) – (必須) 読み取り / 書き込みに使用されます。Asana のオブジェクトの名前。

- API_VERSION(文字列) – (必須) 読み取り / 書き込みに使用されます。使用する Asana Rest API バージョン。(例: 1.0)。
- SELECTED_FIELDS(List<String>) – Default: empty(SELECT *). 読み込みに使用されます。オブジェクトに選択する列。
- FILTER_PREDICATE(文字列) – デフォルト: 空 読み込みに使用されます。Spark SQL 形式である必要があります。
- QUERY(文字列) – デフォルト: 空 読み込みに使用されます。完全な Spark SQL クエリ。
- PARTITION_FIELD(文字列) – 読み取りに使用されます。クエリをパーティション化するために使用するフィールド。
- LOWER_BOUND(文字列) - 読み取りに使用されます。選択したパーティションフィールドの包括的な下限値。
- UPPER_BOUND(文字列) – 読み取りに使用されます。選択したパーティションフィールドの排他的上限値。
- NUM_PARTITIONS(整数) – デフォルト: 1。読み込みに使用されます。読み取り用のパーティションの数。

Asana アカウントの作成

1. Asana アカウントにサインアップし、[サインアップ] を選択します。
2. ログインすると、[アカウントのセットアップ] ページにリダイレクトされます。以下のステップを実行します。
   - アカウント設定フォームを確認します。
   - 関連するすべての詳細を入力して、Asana アカウントを作成します。
   - 情報の精度を再確認します。
3. [アカウントを作成] または [送信] (正確なボタンのテキストは異なる場合があります) を選択して、アカウントの設定を確定します。

OAuth2.0 の Asana でのアプリの作成

1. Asana 顧客認証情報を使用して Asana アカウントにログインします。
2. 右上隅にあるユーザープロファイルアイコンを選択し、ドロップダウンメニューから [マイ設定] を選択します。
3. [アプリ] タブを選択し、[開発者アプリを管理] を選択します。

4. [アプリの新規作成] を選択し、関連する詳細を入力します。
5. [アプリを作成] を選択します。
6. [マイアプリ] ページで、次の操作を行います。
   a. [OAuth] を選択し、[アプリ認証情報] セクションで、クライアント ID とクライアントシークレットを書き留めます。
   b. [リダイレクト URL] セクションで、必要なリダイレクト URL (複数可) を追加します。

> **Note**
>
> 次の形式を使用してリダイレクト URI を入力します: `https://{aws-region-code}.console.aws.amazon.com/gluestudio/oauth`。たとえば、米国東部 (バージニア北部) の場合は次を使用します: `https://us-east-1.console.aws.amazon.com/gluestudio/oauth`

Asana での **PAT** トークン用アプリの作成

1. Asana 顧客認証情報を使用して Asana アカウントにログインします。
2. 右上隅のユーザープロファイルアイコンを選択し、ドロップダウンメニューから [マイプロファイル設定] を選択します。
3. [アプリ] タブを選択し、[サービスアカウント] を選択します。
4. [アプリの新規作成] を選択し、関連する詳細を入力します。
5. [サービスアカウントを追加] を選択します。
6. 次のページにはトークンが表示されます。トークンをコピーして安全に保管します。

> **Important**
>
> このトークンは 1 回だけ表示されます。必ずコピーして安全に保管してください。

制限

Asana コネクタの制限は次のとおりです:

- エンタープライズドメインのサービスアカウントは、監査ログ API エンドポイントにのみアクセスできます。これらのエンドポイントにアクセスするには、サービスアカウントの個人アクセストークンによる認証が必要です。
- Goal エンティティは、Premium プラン以上のユーザーアカウントでのみアクセスできます。
- `Audit Log Event Entity` – コネクタでは、`start_at` フィールドと `end_at` フィールドが 1 つのフィールド「start_end_at」に結合され、フィルタリングと増分転送をサポートします。
- パーティショニングは、以上の演算子と以下の演算子をサポートしている場合でも、`Date` フィールドではサポートできません。シナリオ: `partitionField` を `due_on` (データ型: 日付)、`lowerBound` を `2019-09-14`、`upperBound` を `2019-09-16`、`numPartition` を 2 としてジョブを作成しました。エンドポイント URL のフィルター部分は次のように作成されます。
  - partition1: due_on.before=2019-09-14&due_on.after=2019-09-14
  - partition2: due_on.before=2019-09-15&due_on.after=2019-09-15 出力：
  - partition1 では、due_date が 2019-09-14 および 2019-09-15 のデータを取得します。
  - partition2 では、due_date が (partition1 にあった) 2019-09-15 と同じデータが他のデータとともに取得され、データの重複が発生します。
- SaaS エンドから不正なリクエストエラーがスローされるため、同じフィールドでフィルタリングとパーティショニングをサポートすることはできません。
- Task エンティティには、フィルター条件に少なくとも 1 つのフィールドが必要です。Asana には、時間ベースのフィールドに基づいてレコードをソートしないとページ分割が識別されないという制限があります。したがって、Created_at field は、次のレコードセットを区別するためにページ分割とともに使用されます。Created_at field はフィルターで必須としてマークされ、指定されていない場合、デフォルト値は 2000-01-01T00:00:00Z です。ページ分割の詳細については、「ワークスペースのタスク」を参照してください。

AWS Glue Studio での Azure Cosmos DB に対する接続

AWS Glue は、Azure Cosmos DB のための組み込みサポートを提供します。AWS Glue Studio は、Azure Cosmos DB for NoSQL に接続してデータ統合ジョブをオーサリングし、AWS Glue Studio サーバーレス Spark ランタイム上でそれらのジョブを実行するためのビジュアルインターフェイスを提供します。

トピック

- Azure Cosmos DB 接続の作成
- Azure Cosmos DB ソースノードの作成

- Azure Cosmos DB ターゲットノードの作成
- 詳細オプション

Azure Cosmos DB 接続の作成

前提条件:

- Azure では、AWS Glue で使用する Azure Cosmos DB キー (cosmosKey) を特定または生成する必要があります。詳細については、Azure ドキュメントの「Azure Cosmos DB のデータへのアクセスをセキュリティで保護する」を参照してください。

Azure Cosmos DB に対する接続を設定するには:

1. AWS Secrets Manager で、Azure Cosmos DB キーを使用してシークレットを作成します。Secrets Manager でシークレットを作成するには、AWS Secrets Manager ドキュメントの「AWS Secrets Manager シークレットを作成する」にあるチュートリアルに従ってください。シークレットを作成したら、次のステップのためにシークレット名 *secretName* を保存しておきます。
   - [key/value ペア] を選択する際に、*cosmosKey* という値を持つキー spark.cosmos.accountKey のペアを作成します。
2. AWS Glue コンソールで、「the section called “AWS Glue 接続の追加”」にあるステップに従って接続を作成します。接続を作成したら、将来的に AWS Glue で使用するために、接続名 *connectionName* を維持します。
   - [接続タイプ] を選択する際に、[Azure Cosmos DB] を選択します。
   - [AWS Secret] をクリックして、*secretName* を入力します。

Azure Cosmos DB ソースノードの作成

必要な前提条件

- 前のセクション the section called “Azure Cosmos DB 接続の作成” で説明したように、AWS Secrets Manager シークレットを使用して設定された AWS Glue Azure Cosmos DB 接続。
- 接続で使用されるシークレットを読み取るためのジョブに対する適切なアクセス許可。
- 読み取り元とする Azure Cosmos DB for NoSQL コンテナ。コンテナの識別情報が必要になります。

Azure Cosmos for NoSQL コンテナは、データベースとコンテナによって識別されます。Azure Cosmos for NoSQL API に接続する際には、データベース *cosmosDBName* とコンテナ *cosmosContainerName* の名前を指定する必要があります。

Azure Cosmos DB データソースの追加

[データソース — Azure Cosmos DB] ノードを追加するには:

1. Azure Cosmos DB データソース用の接続を選択します。すでに作成したので、ドロップダウンに表示されているはずです。接続を作成する必要がある場合は、[Azure Cosmos DB 接続を作成] を選択します。詳細については、前の「the section called “Azure Cosmos DB 接続の作成”」セクションを参照してください。

   接続を選択したあとは、[プロパティを表示] をクリックすると、接続のプロパティを表示できます。
2. [Cosmos DB データベース名] を選択します。読み取り元のデータベースの名前である *cosmosDBName* を指定します。
3. [Azure Cosmos DB コンテナ] を選択します。読み取り元のコンテナの名前である *cosmosContainerName* を指定します。
4. 必要に応じて、[Azure Cosmos DB カスタムクエリ] を選択します。Azure Cosmos DB から特定の情報を取得するには、SQL SELECT クエリを指定します。
5. [Azure Cosmos のカスタムプロパティ] で、必要に応じてパラメータと値を入力します。

Azure Cosmos DB ターゲットノードの作成

必要な前提条件

- 前のセクション the section called “Azure Cosmos DB 接続の作成” で説明したように、AWS Secrets Manager シークレットを使用して設定された AWS Glue Azure Cosmos DB 接続。
- 接続で使用されるシークレットを読み取るためのジョブに対する適切なアクセス許可。
- 書き込み先とする Azure Cosmos DB テーブル。コンテナの識別情報が必要になります。接続メソッドを呼び出す前にコンテナを作成する必要があります。

  Azure Cosmos for NoSQL コンテナは、データベースとコンテナによって識別されます。Azure Cosmos for NoSQL API に接続する際には、データベース *cosmosDBName* とコンテナ *cosmosContainerName* の名前を指定する必要があります。

Azure Cosmos DB データターゲットの追加

[データターゲット — Azure Cosmos DB] ノードを追加するには:

1. Azure Cosmos DB データソース用の接続を選択します。すでに作成したので、ドロップダウンに表示されているはずです。接続を作成する必要がある場合は、[Azure Cosmos DB 接続を作成] を選択します。詳細については、前の「the section called “Azure Cosmos DB 接続の作成”」セクションを参照してください。

   接続を選択したあとは、[プロパティを表示] をクリックすると、接続のプロパティを表示できます。
2. [Cosmos DB データベース名] を選択します。読み取り元のデータベースの名前である *cosmosDBName* を指定します。
3. [Azure Cosmos DB コンテナ] を選択します。読み取り元のコンテナの名前である *cosmosContainerName* を指定します。
4. [Azure Cosmos のカスタムプロパティ] で、必要に応じてパラメータと値を入力します。

詳細オプション

Azure Cosmos DB ノードを作成する際に、高度なオプションを指定できます。これらのオプションは Spark AWS Glue スクリプトのプログラミング時に使用できるオプションと同じです。

「the section called “Azure Cosmos DB 接続”」を参照してください。

AWS Glue Studio での Azure SQL に対する接続

AWS Glue は、Azure SQL のための組み込みサポートを提供します。AWS Glue Studio は、Azure SQL に接続してデータ統合ジョブをオーサリングし、AWS Glue Studio サーバーレス Spark ランタイム上でそれらのジョブを実行するためのビジュアルインターフェイスを提供します。

トピック

- Azure SQL 接続の作成
- Azure SQL ソースノードの作成
- Azure SQL ターゲットノードの作成
- 詳細オプション

Azure SQL 接続の作成

AWS Glue から Azure SQL に接続するには、Azure SQL 認証情報を作成して AWS Secrets Manager シークレットに保存し、そのシークレットを Azure SQL AWS Glue 接続に関連付ける必要があります。

Azure SQL に対する接続を設定するには:

1. AWS Secrets Manager で、Azure SQL 認証情報を使用してシークレットを作成します。Secrets Manager でシークレットを作成するには、AWS Secrets Manager ドキュメントの「AWS Secrets Manager シークレットを作成する」にあるチュートリアルに従ってください。シークレットを作成したら、次のステップのためにシークレット名 *secretName* を保存しておきます。

   - [key/value ペア] を選択する際に、*azuresqlUsername* という値を持つキー `user` のペアを作成します。
   - [key/value ペア] を選択する際に、*azuresqlPassword* という値を持つキー `password` のペアを作成します。

2. AWS Glue コンソールで、「the section called “AWS Glue 接続の追加”」にあるステップに従って接続を作成します。接続を作成したら、将来的に AWS Glue で使用するために、接続名 *connectionName* を維持します。

   - [接続タイプ] を選択する際に、[Azure SQL] を選択します。
   - Azure SQL URL を指定する場合は、JDBC エンドポイント URL を入力します。

     URL は、次のような形式になります:
     `jdbc:sqlserver://`*databaseServerName*`:`*databasePort*`;databaseName=`*azuresqlDBnam*

     AWS Glue には次の URL プロパティが必要です。

     - `databaseName` – 接続先の Azure SQL のデフォルトデータベース。

     Azure SQL Managed Instances の JDBC URL の詳細については、Microsoft のドキュメントを参照してください。

   - [AWS Secret] をクリックして、*secretName* を入力します。

Azure SQL ソースノードの作成

必要な前提条件

- 前のセクション the section called “Azure SQL 接続の作成” で説明したように、AWS Secrets Manager シークレットを使用して設定された AWS Glue Azure SQL 接続。
- 接続で使用されるシークレットを読み取るためのジョブに対する適切なアクセス許可。
- 読み取り元とする Azure SQL テーブル (*tableName*)。

  Azure SQL テーブルは、データベース、スキーマ、テーブル名によって識別されます。Azure SQL に接続する際には、データベース名とテーブル名を指定する必要があります。スキーマがデフォルトの「public」でない場合は、スキーマも指定する必要があります。データベースは、*connectionName* の URL プロパティ、dbtable を通じたスキーマおよびテーブル名を介して指定されます。

Azure SQL データソースの追加

[データソース — Azure SQL] ノードを追加するには:

1. Azure SQL データソース用の接続を選択します。すでに作成したので、ドロップダウンに表示されているはずです。接続を作成する必要がある場合は、[Azure SQL 接続を作成] を選択します。詳細については、前の「the section called “Azure SQL 接続の作成”」セクションを参照してください。

   接続を選択したあとは、[プロパティを表示] をクリックすると、接続のプロパティを表示できます。
2. [Azure SQL ソース] オプションを選択します。
   - [単一のテーブルを選択] – 単一のテーブルからすべてのデータにアクセスします。
   - [カスタムクエリを入力] — カスタムクエリに基づいて、複数のテーブルからデータセットにアクセスします。
3. 単一のテーブルを選択した場合は、*tableName* を入力します。

   [カスタムクエリを入力] を選択した場合は、TransactSQL SELECT クエリを入力します。
4. [Azure SQL のカスタムプロパティ] で、必要に応じてパラメータと値を入力します。

Azure SQL ターゲットノードの作成

必要な前提条件

- 前のセクション the section called “Azure SQL 接続の作成” で説明したように、AWS Secrets Manager シークレットを使用して設定された AWS Glue Azure SQL 接続。
- 接続で使用されるシークレットを読み取るためのジョブに対する適切なアクセス許可。
- 書き込み先とする Azure SQL テーブル (*tableName*)。

  Azure SQL テーブルは、データベース、スキーマ、テーブル名によって識別されます。Azure SQL に接続する際には、データベース名とテーブル名を指定する必要があります。スキーマがデフォルトの「public」でない場合は、スキーマも指定する必要があります。データベースは、*connectionName* の URL プロパティ、dbtable を通じたスキーマおよびテーブル名を介して指定されます。

Azure SQL データターゲットの追加

[データターゲット — Azure SQL] ノードを追加するには:

1. Azure SQL データソース用の接続を選択します。すでに作成したので、ドロップダウンに表示されているはずです。接続を作成する必要がある場合は、[Azure SQL 接続を作成] を選択します。詳細については、前の「the section called “Azure SQL 接続の作成”」セクションを参照してください。

   接続を選択したあとは、[プロパティを表示] をクリックすると、接続のプロパティを表示できます。
2. *tableName* を指定して [テーブル名] を設定します。
3. [Azure SQL のカスタムプロパティ] で、必要に応じてパラメータと値を入力します。

詳細オプション

Azure SQL ノードを作成する際に、高度なオプションを指定できます。これらのオプションは Spark AWS Glue スクリプトのプログラミング時に使用できるオプションと同じです。

「the section called “Azure SQL 接続”」を参照してください。

Blackbaud Raiser's Edge NXT への接続

Blackbaud Raiser's Edge NXT は、非営利団体やソーシャルグッドコミュニティ全体向けに特別に構築された、包括的なクラウドベースの資金集めおよび寄付者管理ソフトウェアソリューションです。このコネクタは、Blackbaud Raiser's Edge NXT の SKY API 上に構築されており、Rayers Edge NXT 内のエンティティの管理に役立つオペレーションを提供します。

トピック

- Blackbaud Raiser's Edge NXT の AWS Glue サポート
- 接続を作成および使用するための API オペレーションを含むポリシー
- Blackbaud Raiser's Edge NXT の設定
- Blackbaud Raiser's Edge NXT 接続の設定
- Blackbaud Raiser's Edge NXT エンティティからの読み取り
- Blackbaud Raiser's Edge NXT 接続オプション
- Blackbaud Raiser's Edge NXT の制限事項

Blackbaud Raiser's Edge NXT の AWS Glue サポート

AWS Glue は、次のように Blackbaud Raiser's Edge NXT をサポートします。

ソースとしてサポートされていますか?

はい。AWS Glue ETL ジョブを使用して、Blackbaud Raiser's Edge NXT からデータをクエリできます。

ターゲットとしてサポートされていますか?

いいえ。

サポートされている Blackbaud Raiser's Edge NXT API バージョン

次の Blackbaud Raiser's Edge NXT API バージョンがサポートされています。

- v1

接続を作成および使用するための API オペレーションを含むポリシー

次のサンプル ポリシーで、接続の作成と使用に必要な AWS IAM 権限について説明します。新しいロールを作成する場合は、以下を含むポリシーを作成します。

```
{
 "Version": "2012-10-17",
    "Statement": [
     {
            "Effect": "Allow",
            "Action": [
                "glue:ListConnectionTypes",
                "glue:DescribeConnectionType",
                "glue:RefreshOAuth2Tokens",
                "glue:ListEntities",
                "glue:DescribeEntity"
            ],
            "Resource": "*"
        }
    ]
 }
```

上記の方法を使用しない場合は、代わりに次のマネージド IAM ポリシーを使用します:

- AWSGlueServiceRole – さまざまな AWS Glue プロセスを実行するために必要なリソースへのアクセス権をユーザーに代わって付与します。これらのリソースには AWS Glue 、Amazon S3、IAM、CloudWatch Logs、および Amazon EC2 が含まれます。このポリシーで指定されたリソースの命名規則に従った場合、AWS Glue プロセスは必要なアクセス権限を使用できます。このポリシーは、通常、クローラ、ジョブ、開発エンドポイントを定義するときに指定されたロールにアタッチされます。
- AWSGlueConsoleFullAccess — ポリシーがアタッチされているアイデンティティが AWS マネージメントコンソールを使用するときは、AWS Glue リソースへのフルアクセスを許可します。このポリシーで指定されたリソースの命名規則に従った場合、ユーザーは完全なコンソール機能を使用できます。このポリシーは、通常 AWS Glue コンソールのユーザーにアタッチされています。

Blackbaud Raiser's Edge NXT の設定

AWS Glue を使用して Blackbaud Raiser's Edge NXT からデータを転送できるようにするには、次の要件を満たす必要があります。

最小要件

以下に、最小要件を示します。

- Blackbaud Raiser's Edge NXT アカウントを持っている。

- API にアクセスするために割り当てられた適切な読み取り/書き込みスコープを使用して、Blackbaud Raiser's Edge NXT アカウントにアクセストークンを生成した。詳細については、「Authorization」を参照してください。

これらの要件を満たすと、Blackbaud Raiser's Edge NXT アカウントに AWS Glue を接続する準備が整います。

Blackbaud Raiser's Edge NXT 接続の設定

Blackbaud Raiser's Edge NXT は、OAuth2 の AUTHORIZATION_CODE グラントタイプをサポートしています。

- このグラントタイプは、ユーザーを認証するためにサードパーティーの認証サーバーにユーザーをリダイレクトすることから、「three-legged」の OAuth と見なされます。AWS Glue コンソール経由で接続を作成するときに使用されます。AWS Glue コンソールでは、ユーザーが Blackbaud Raiser's Edge NXT にリダイレクトされます。ここではユーザーがログインし、Blackbaud Raiser's Edge NXT インスタンスにアクセスするためのリクエストされたアクセス許可を AWS Glue に許可する必要があります。
- ユーザーは、Blackbaud Raiser's Edge NXT で独自の接続アプリを作成し、AWS Glue コンソールを介して接続を作成するときに独自のクライアント ID、サブスクリプションキー、およびインスタンス URL を指定することを選択できます。このシナリオでは、引き続き Blackbaud Raiser's Edge NXT にリダイレクトされてログインし、リソースへアクセスするために AWS Glue を承認します。
- このグラントタイプは、更新トークンとアクセストークンになります。アクセストークンの有効期間は短く、更新トークンを使用してユーザーとやり取りすることなく自動的に更新される場合があります。
- 認可コード OAuth フロー用の接続アプリの作成に関する Blackbaud Raiser's Edge NXT の公開ドキュメントについては、「Authorization」を参照してください。

Blackbaud Raiser's Edge NXT 接続を設定するには:

1. AWS Secrets Manager で、次の詳細を含むシークレットを作成します。
   a. カスタマーマネージド接続アプリケーションの場合、シークレットには、*USER_MANAGED_CLIENT_APPLICATION_CLIENT_SECRET* を使用して接続されたアプリケーションの API キーをキーとして含める必要があります。
   b. 注: AWS Glue で接続にシークレットを作成する必要があります。

1. AWS Glue Glue Studio で、以下の手順に従って [データ接続] の下に接続を作成します。
   a. [データソース] を選択する際に、[Blackbaud Raiser's Edge NXT] を選択します。
   b. 接続する Blackbaud Raiser's Edge NXT アカウントの `INSTANCE_URL` を指定します。
   c. ユーザーマネージドクライアントアプリケーション `clientId` を指定します。
   d. アカウントに関連付けられたサブスクリプションキーを指定します。
   e. 次のアクションを実行でき、AWS Glue がその権限を持つ AWS IAM ロールを選択します。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:DescribeSecret",
                "secretsmanager:GetSecretValue",
                "secretsmanager:PutSecretValue",
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterface",
                "ec2:DeleteNetworkInterface",
            ],
            "Resource": "*"
        }
    ]
}
```

   f. AWS Glue でこの接続に使用する `secretName` を選択して、トークンを配置します。
   g. ネットワークを使用する場合は、ネットワークオプションを選択します。
2. AWS Glue ジョブに関連付けられている IAM ロールに `secretName` を読み取るアクセス許可を付与します。

Blackbaud Raiser's Edge NXT エンティティからの読み取り

前提条件

読み取り元の Blackbaud Raiser's Edge NXT オブジェクト。オブジェクト名が必要になります。

ソースに対応するエンティティ:

エンティティ	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
構成員アドレス	あり	はい	なし	あり	あり
構成員教育	あり	はい	なし	あり	あり
構成員 E メールアドレス	あり	はい	なし	あり	あり
構成員電話	あり	はい	なし	あり	あり
構成員ノート	あり	はい	なし	あり	あり
構成員関係	あり	はい	なし	あり	あり
構成員オンラインプレゼンス	あり	はい	なし	あり	あり
機会	あり	はい	なし	あり	あり
アピール	あり	はい	なし	あり	あり
キャンペーン	あり	はい	なし	あり	あり
資金	あり	はい	なし	あり	あり
パッケージ	あり	はい	なし	あり	あり
ギフトバッチ	あり	はい	なし	あり	不可
イベント参加者	あり	あり	あり	あり	あり

エンティティ	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
構成員ファンドレイザー割り当て	いいえ	いいえ	なし	あり	不可
ギフト	あり	あり	あり	あり	あり
メンバーシップ	あり	はい	なし	あり	あり
アクション	あり	はい	なし	あり	不可
構成員	あり	あり	あり	あり	あり
構成員物品	あり	はい	なし	あり	あり
イベント	あり	あり	あり	あり	あり
ギフトカスタムフィールド	あり	はい	なし	あり	あり

例:

```
blackbaud_read = glueContext.create_dynamic_frame.from_options(
    connection_type="BLACKBAUD",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "entityName",
        "API_VERSION": "v1",
        "SUBSCRIPTION_KEY": <Subscription key associated with one's developer account>
    }
```

Blackbaud Raiser's Edge NXT エンティティとフィールドの詳細

エンティティとフィールドの詳細については、以下を参照してください:

- アクション

- 構成員
- 構成員アドレス
- 構成員メンバーシップ
- 構成員ファンドレイザー割り当て
- 構成員教育
- 構成員 E メールアドレス
- 構成員電話
- 構成員ノート
- 構成員オンラインプレゼンス
- 構成員関係
- イベント
- イベント参加者
- アピール
- Campaign
- 資金
- パッケージ
- ギフト
- ギフトカスタムフィールド
- ギフトバッチ
- 機会
- 構成員コード

> **Note**
>
> 構造体データ型とリストデータ型は、コネクタの応答で文字列データ型に変換され、DateTime データ型は Timestamp に変換されます。

パーティショニングクエリ

フィールドベースのパーティション分割:

Blackbaud Raiser's Edge NXT は、フィールドベースまたはレコードベースのパーティショニングをサポートしていません。

レコードベースのパーティション分割:

Spark で並行処理を使用する場合は、追加の Spark オプション NUM_PARTITIONS を指定できます。これらのパラメータを使用すると、元のクエリは Spark タスクで同時に実行できるサブクエリの NUM_PARTITIONS の数に分割されます。

レコードベースのパーティショニングでは、存在するレコードの合計数が Blackbaud Raiser's Edge NXT API からクエリされ、指定された NUM_PARTITIONS 数で割られます。その後、結果のレコード数は、各サブクエリによって同時に取得されます。

- NUM_PARTITIONS: パーティション数。

例:

```
blackbaud_read = glueContext.create_dynamic_frame.from_options(
    connection_type="BLACKBAUD",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "entityName",
        "API_VERSION": "v1",
        "NUM_PARTITIONS": "2",
        "SUBSCRIPTION_KEY": <Subscription key associated with one's developer account>
    }
```

Blackbaud Raiser's Edge NXT 接続オプション

Blackbaud Raiser's Edge NXT の接続オプションは次のとおりです。

- ENTITY_NAME(文字列) – (必須) 読み取りに使用されます。Blackbaud Raiser's Edge NXT のオブジェクトの名前。
- API_VERSION(文字列) – (必須) 読み取りに使用されます。使用する Blackbaud Raiser's Edge NXT Rest API バージョン。
- SELECTED_FIELDS(List<String>) – Default: empty(SELECT *). 読み込みに使用されます。オブジェクトに選択する列。
- FILTER_PREDICATE(文字列) – デフォルト: 空 読み込みに使用されます。Spark SQL 形式である必要があります。

- QUERY(文字列) - デフォルト: 空 読み込みに使用されます。完全な Spark SQL クエリ。
- NUM_PARTITIONS(整数) – デフォルト: 1。読み込みに使用されます。読み取り用のパーティションの数。値の例: 10。
- SUBSCRIPTION_KEY(文字列) - (必須) デフォルト: 空。読み込みに使用されます。デベロッパーアカウントに関連付けられたサブスクリプションキー。

Blackbaud Raiser's Edge NXT の制限事項

Blackbaud Raiser's Edge NXT の制限事項または注意事項は次のとおりです。

- SaaS は EQUAL_TO 演算子のみをサポートし、指定された日付以降に作成または変更された結果を返します。さらに、「id」フィールドは文字列データ型です。また、null 非許容フィールドの識別も行われません。したがって、フィールドベースのパーティショニングはサポートされていません。
- 増分プルは、毎日、毎月、毎週の頻度で Event エンティティによってのみサポートされます。
- 構成員ファンドレイザー割り当てエンティティは、最大 20 個のレコードを返します。
- レコードベースのパーティション分割:
  - Action、Constituent Fundraiser Assignment または Gift Batch エンティティではサポートされていません。
  - フィルター述語を使用したレコードベースのパーティショニングは、Event および Event Participant エンティティでのみサポートされます。フィルター述語が他のレコードベースのサポートされているエンティティと共に使用されている場合、例外がスローされます。
- Gift Custom Field エンティティでは、フィールド「value」をフィールド「category」と組み合わせて使用する必要があります。そうしないと、フィルタリングされていないレスポンスが発生します。したがって、「value」フィールドでフィルタリングしながら「category」フィールドをプラグインするようにユーザーに強制するために、前述の要件に従わない場合、例外がスローされます。
- 該当するすべてのエンティティの date_added および last_modified フィールドは、比較演算子をサポートしていません。これらは、等値演算子のみをサポートします。また、前述のフィールドと組み合わせてレコードの範囲を指定できるフィールドはありません。したがって、これらのフィールドはクエリのみ可能で、増分転送をサポートすることはできません。
- Gift Batch エンティティの added_by フィールドは、正しい結果を出力しない可能性があるため、フィルタリング可能とは見なされません。

- `Gift` エンティティにデータを挿入すると、`/GET Gift List` エンドポイントを介してレコードが取得されるまでに約 30 分のレイテンシーがあります。
- データソース側の制限により、ギフトエンティティの増分転送のサポートは廃止されました。
- 機会エンティティのステータスフィールドには 10 分のレイテンシーがあります。
- `Fundraiser Assignment` エンティティには、依存エンティティとして `Constituent` があります。コネクタは、最大 5,000 ID をロードして、レスポンスサイズが最大許容ペイロードサイズを超えないようにします。

CircleCI への接続

CircleCI は、継続的インテグレーションおよび継続的デリバリープラットフォームです。CircleCI アカウントには、プロジェクト、パイプライン、ワークフローなどに関するデータが含まれています。CircleCI ユーザーの場合は、CircleCI アカウントに AWS Glue を接続できます。その後、CircleCI を ETL ジョブのデータソースとして使用できます。これらのジョブを実行して、CircleCI と AWS サービス、またはその他のサポートされているアプリケーション間でデータを転送します。

トピック

- CircleCI の AWS Glue サポート
- 接続を作成および使用するための API オペレーションを含むポリシー
- CircleCI の設定
- CircleCI 接続の設定
- CircleCI エンティティからの読み取り
- CircleCI 接続オプション
- CircleCI の制限事項

CircleCI の AWS Glue サポート

AWS Glue は、次のように CircleCI をサポートします。

ソースとしてサポートされていますか?

はい。AWS Glue ETL ジョブを使用して、CircleCI からデータをクエリできます。

ターゲットとしてサポートされていますか?

いいえ。

サポートされている CircleCI API バージョン

次の CircleCI API バージョンがサポートされています。

- v2

接続を作成および使用するための API オペレーションを含むポリシー

次のサンプル ポリシーで、接続の作成と使用に必要な AWS IAM 権限について説明します。新しいロールを作成する場合は、以下を含むポリシーを作成します。

```
{
 "Version": "2012-10-17",
     "Statement": [
      {
             "Effect": "Allow",
             "Action": [
                 "glue:ListConnectionTypes",
                 "glue:DescribeConnectionType",
                 "glue:RefreshOAuth2Tokens",
                 "glue:ListEntities",
                 "glue:DescribeEntity"
             ],
             "Resource": "*"
         }
     ]
 }
```

上記の方法を使用しない場合は、代わりに次のマネージド IAM ポリシーを使用します:

- AWSGlueServiceRole – さまざまな AWS Glue プロセスを実行するために必要なリソースへのアクセス権をユーザーに代わって付与します。これらのリソースには AWS Glue 、Amazon S3、IAM、CloudWatch Logs、および Amazon EC2 が含まれます。このポリシーで指定されたリソースの命名規則に従った場合、AWS Glue プロセスは必要なアクセス権限を使用できます。このポリシーは、通常、クローラ、ジョブ、開発エンドポイントを定義するときに指定されたロールにアタッチされます。
- AWSGlueConsoleFullAccess — ポリシーがアタッチされているアイデンティティが AWS マネージメントコンソールを使用するときは、AWS Glue リソースへのフルアクセスを許可します。この

ポリシーで指定されたリソースの命名規則に従った場合、ユーザーは完全なコンソール機能を使用できます。このポリシーは、通常 AWS Glue コンソールのユーザーにアタッチされています。

CircleCI の設定

AWS Glue を使用して CircleCI からデータを転送できるようにするには、次の要件を満たす必要があります。

最小要件

以下に、最小要件を示します。

- 転送するデータを含む CircleCI のアカウントを持っている。
- アカウントのユーザー設定で、個人用 API トークンを作成した。詳細については、「Creating a personal API token」を参照してください。
- 接続の作成時に、個人用 API トークンを AWS Glue に提供します。

これらの要件を満たすと、CircleCI アカウントに AWS Glue を接続する準備が整います。

CircleCI 接続の設定

CircleCI はカスタム認可をサポートしています。

CircleCI 接続を設定するには:

1. AWS Secrets Manager で、次の詳細を含むシークレットを作成します。
   a. カスタマーマネージド接続アプリケーションの場合、シークレットには、`Circle-Token` を使用して接続されたアプリケーションの API キーをキーとして含める必要があります。
   b. 注: AWS Glue で接続にシークレットを作成する必要があります。

1. AWS Glue Glue Studio で、以下の手順に従って [データ接続] の下に接続を作成します。
   a. [データソース] を選択する際に、[CircleCI] を選択します。
   b. 次のアクションを実行でき、AWS Glue がその権限を持つ AWS IAM ロールを選択します。

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

```
            "Effect": "Allow",
            "Action": [
                "secretsmanager:DescribeSecret",
                "secretsmanager:GetSecretValue",
                "secretsmanager:PutSecretValue",
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterface",
                "ec2:DeleteNetworkInterface",
            ],
            "Resource": "*"
        }
    ]
}
```

c. AWS Glue でこの接続に使用する `secretName` を選択して、トークンを配置します。

d. ネットワークを使用する場合は、ネットワークオプションを選択します。

2. AWS Glue ジョブに関連付けられている IAM ロールに `secretName` を読み取るアクセス許可を付与します。

CircleCI エンティティからの読み取り

前提条件

読み取り元の CircleCI オブジェクト。オブジェクト名が必要になります。

ソースに対応するエンティティ:

エンティティ	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
Context	あり	いいえ	なし	あり	不可
組織概要メトリクス	あり	いいえ	なし	あり	不可
パイプライン	いいえ	いいえ	なし	あり	不可
パイプラインワークフロー	あり	いいえ	なし	あり	不可

エンティティ	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
プロジェクトブランチ	あり	いいえ	なし	あり	不可
プロジェクトフレーキーテスト	いいえ	いいえ	なし	あり	不可
プロジェクト概要メトリクス	あり	いいえ	なし	あり	不可
スケジュール	いいえ	いいえ	なし	あり	不可
ワークフロージョブの時系列	あり	いいえ	なし	あり	不可
ワークフローのメトリクスと傾向	あり	いいえ	なし	あり	不可
ワークフローの最近の実行	あり	いいえ	なし	あり	不可
ワークフロー概要メトリクス	あり	いいえ	なし	あり	不可
ワークフローテストメトリクス	あり	いいえ	なし	あり	不可

例:

```
circleci_read = glueContext.create_dynamic_frame.from_options(
    connection_type="circleci",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "context/e7ea2945-dccb-4205-b673-8391fe1b3a4c",
        "API_VERSION": "v2"
    }
```

CircleCI エンティティとフィールドの詳細

エンティティとフィールドの詳細については、以下を参照してください:

- コンテキスト
- プロジェクト概要メトリクス
- ワークフロージョブの時系列
- 組織概要メトリクス
- プロジェクトブランチ
- プロジェクトフレーキーテスト
- ワークフローの最近の実行
- ワークフロー概要メトリクス
- ワークフローのメトリクスと傾向
- ワークフローテストメトリクス
- Pipelines
- パイプラインワークフロー
- スケジュール

静的メタデータを持つエンティティ:

エンティティ	フィールド	データ型	サポートされている演算子
Context	作成時刻	String	
	ID	String	
	名前	String	

エンティティ	フィールド	データ型	サポートされている演算子
	所有者タイプ	String	EQUAL_TO
組織概要メトリクス	すべてのプロジェクト	リスト	
	組織データ	Struct	
	組織プロジェクトデータ	リスト	
	プロジェクト名	String	EQUAL_TO
	レポートウィンドウ	String	EQUAL_TO
パイプライン	ブランチ	String	EQUAL_TO
	作成時刻	String	
	エラー	リスト	
	ID	String	
	数値	整数	
	プロジェクトスラグ	String	
	都道府県	String	
	Trigger トリガー)	Struct	
	トリガーパラメータ	Struct	
	更新時刻	String	
	VCS	Struct	
パイプラインワークフロー	キャンセルしたユーザー	String	

エンティティ	フィールド	データ型	サポートされている演算子
	作成時刻	String	
	エラー発生元	String	
	ID	String	
	名前	String	
	パイプライン ID	String	
	パイプライン番号	整数	
	プロジェクトスラグ	String	
	開始したユーザー	String	
	ステータス	String	
	停止時刻	String	
	タグ	String	
プロジェクトブランチ	ブランチ	リスト	
	組織 ID	String	
	プロジェクト ID	String	
	ワークフローの名前	String	EQUAL_TO
プロジェクトフレーキーテスト	Classname	String	
	ファイル	String	
	ジョブ名	String	
	ジョブ番号	整数	
	パイプライン番号	整数	

エンティティ	フィールド	データ型	サポートされている演算子
	ソース	String	
	テスト名	String	
	消費時間	整数	
	フレークされた回数	整数	
	ワークフローの作成時刻	String	
	ワークフロー ID	String	
	ワークフローの名前	String	
プロジェクト概要メトリクス	すべてのブランチ	リスト	
	すべてのワークフロー	リスト	
	ブランチ	String	EQUAL_TO
	組織 ID	String	
	プロジェクトデータ	Struct	
	プロジェクト ID	String	
	プロジェクトのワークフローブランチデータ	リスト	
	プロジェクトのワークフローデータ	リスト	
	レポートウィンドウ	String	EQUAL_TO
	ワークフロー名	String	EQUAL_TO

エンティティ	フィールド	データ型	サポートされている演算子
スケジュール	[Actor] (アクター)	Struct	
	作成時刻	String	
	説明	文字列	
	ID	String	
	名前	String	
	パラメータ	Struct	
	プロジェクトスラグ	String	
	タイムテーブル	Struct	
	更新時刻	String	
ワークフロージョブの時系列	ブランチ	String	EQUAL_TO
	詳細度	String	EQUAL_TO
	最大終了時刻	String	
	メトリクス	Struct	
	最小開始時刻	String	
	名前	String	
	開始終了日	DateTime	EQUAL_TO、BETWEEN
	Timestamp	String	
ワークフローのメトリクスと傾向	すべてのブランチ	ブール値	EQUAL_TO
	ブランチ	String	EQUAL_TO

エンティティ	フィールド	データ型	サポートされている演算子
	メトリクス	Struct	
	トレンド	Struct	
	ワークフロー名	リスト	
ワークフローの最近の実行	すべてのブランチ	ブール値	EQUAL_TO
	ブランチ	String	EQUAL_TO
	作成時刻	String	
	使用されたクレジット	整数	
	期間	整数	
	ID	String	
	承認されているかどうか	ブール値	
	開始終了日	DateTime	EQUAL_TO、BETWEEN
	ステータス	String	
	停止時刻	String	
ワークフロー概要メトリクス	すべてのブランチ	ブール値	EQUAL_TO
	ブランチ	String	EQUAL_TO
	メトリクス	Struct	
	名前	String	
	プロジェクト ID	String	

エンティティ	フィールド	データ型	サポートされている演算子
ワークフローテストメトリクス	レポートウィンドウ	String	EQUAL_TO
	ウィンドウの終了	String	
	ウィンドウの開始	String	
	平均テスト数	整数	
	ブランチ	String	EQUAL_TO
	最も失敗したテスト	リスト	
	最も失敗したテストの追加分	整数	
	最も遅いテスト	リスト	
	最も遅いテストの追加分	整数	
	テスト実行	リスト	
	テスト実行の合計数	整数	

> **Note**
>
> 構造体データ型とリストデータ型は、コネクタの応答で文字列データ型に変換されます。

パーティショニングクエリ

CircleCI は、フィールドベースまたはレコードベースのパーティショニングをサポートしていません。

CircleCI 接続オプション

CircleCI の接続オプションは次のとおりです。

- ENTITY_NAME(文字列) – (必須) 読み取りに使用されます。CircleCI のオブジェクトの名前。
- API_VERSION(文字列) – (必須) 読み取りに使用されます。使用する CircleCI Rest API バージョン。
- SELECTED_FIELDS(List<String>) – Default: empty(SELECT *). 読み込みに使用されます。オブジェクトに選択する列。
- FILTER_PREDICATE(文字列) – デフォルト: 空 読み込みに使用されます。Spark SQL 形式である必要があります。
- QUERY(文字列) - デフォルト: 空 読み込みに使用されます。完全な Spark SQL クエリ。

CircleCI の制限事項

CircleCI の制限事項または注意事項は次のとおりです。

- CircleCI は、フィールドベースまたはレコードベースのパーティション分割をサポートしていません。
- '-' (ハイフン) を含むフィルターフィールドは、バックティックでラップされている場合にのみ機能します。例: `workflow-name` = "abc"
- GitLab VCS エンティティパスに必要な「プロジェクト ID」を取得するプログラム的な方法がないため、GitLab VCS タイプをサポートできません。

Datadog に対する接続

Datadog は、インフラストラクチャ、アプリケーション、サービス、ツールなど、クラウド規模のアプリケーション向けのモニタリングおよび分析プラットフォームです。

トピック

- Datadog 向けの AWS Glue サポート
- 接続を作成および使用するための API オペレーションを含むポリシー
- Datadog の設定
- Datadog 接続の設定
- Datadog エンティティからの読み取り
- Datadog 接続オプション
- Datadog アカウントの作成

- 制限

Datadog 向けの AWS Glue サポート

AWS Glue は、次のように Datadog をサポートします。

ソースとしてサポートされていますか?

はい。AWS Glue ETL ジョブを使用して、Datadog からデータをクエリできます。

ターゲットとしてサポートされていますか?

いいえ。

サポートされている Datadog API バージョン

- v1
- v2

接続を作成および使用するための API オペレーションを含むポリシー

次のサンプル ポリシーで、接続の作成と使用に必要な AWS の権限について説明します。新しいロールを作成する場合は、以下を含むポリシーを作成します。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:ListConnectionTypes",
                "glue:DescribeConnectionType",
                "glue:RefreshOAuth2Tokens",
                "glue:ListEntities",
                "glue:DescribeEntity"
            ],
            "Resource": "*"
        }
    ]
```

```
}
```

上記の方法を使用しない場合は、代わりに次のマネージド IAM ポリシーを使用します。

- AWSGlueServiceRole – さまざまな AWS Glue プロセスを実行するために必要なリソースへのアクセス権をユーザーに代わって付与します。これらのリソースには、AWS Glue、Amazon S3、IAM、CloudWatch Logs、および Amazon EC2 が含まれます。このポリシーで指定されたリソースの命名規則に従った場合、AWS Glue プロセスは必要なアクセス権限を使用できます。このポリシーは、通常、クローラ、ジョブ、開発エンドポイントを定義するときに指定されたロールにアタッチされます。
- AWSGlueConsoleFullAccess — ポリシーがアタッチされているアイデンティティが AWS マネージメントコンソールを使用するときは、AWS Glue リソースへのフルアクセスを許可します。このポリシーで指定されたリソースの命名規則に従った場合、ユーザーは完全なコンソール機能を使用できます。このポリシーは、通常 AWS Glue コンソールのユーザーにアタッチされています。

Datadog の設定

AWS Glue を使用して Datadog からデータを転送できるようにするには、次の要件を満たす必要があります。

最小要件

- DD-API-KEY と DD-APPLICATION-KEY を持つ Datadog アカウントがある。アカウントの作成の詳細については、「Datadog アカウントの作成」を参照してください。
- Datadog アカウントに、有効なライセンスを持つ API アクセスがある。

Datadog は、次の 6 つの URL をサポートしている。すべての Datadog API クライアントは、デフォルトで Datadog US1 サイト API を使用するように設定されている。Datadog EU サイトを使用している場合、API にアクセスするには、Datadog EU サイトの `DD-API-KEY` と `DD-APPLICATION-KEY` を含む https://api.datadoghq.eu の URL を選択する必要があります。同様に、他のサイトの場合は、それぞれのサイトの `DD-API-KEY and DD-APPLICATION-KEY` でそれぞれの URL を選択する必要があります。

- US1 API URL — https://api.datadoghq.comhttps://api.datadoghq.com
- EU API URL — https://api.datadoghq.eu
- US3 API URL — https://api.us3.datadoghq.com

- US5 API URL — https://api.us5.datadoghq.com
- S1-FED API URL – https://api.ddog-gov.com
- Japan API URL — https://api.ap1.datadoghq.com

これらの要件を満たすと、Datadog アカウントに AWS Glue を接続する準備が整います。

Datadog 接続の設定

Datadog はカスタム認証をサポートしています。Datadog 接続を設定する手順は次のとおりです。

Datadog 接続を設定するには:

1. AWS Secrets Manager で、次の詳細を含むシークレットを作成します。

   カスタマーマネージド接続アプリケーションの場合 – シークレットには、**API_KEY** と **APPLICATION_KEY** を使用して接続されたアプリケーションのコンシューマーシークレットをキーとして含める必要があります。

   > **Note**
   >
   > AWS Glue で接続ごとにシークレットを作成する必要があります。

2. AWS Glue Studio で、以下の手順に従って [データ接続] の下に接続を作成します。
   a. [接続タイプ] を選択する際に、[Datadog] を選択します。
   b. 接続先の Datadog の `Instance_Url` を指定します。
   c. AWS Glue が次のアクションを担い、その権限を持つことができる IAM ロールを選択します。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:DescribeSecret",
                "secretsmanager:GetSecretValue",
                "secretsmanager:PutSecretValue",
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterface",
                "ec2:DeleteNetworkInterface",
            ],
```

```
            "Resource": "*"
        }
    ]
}
```

d. AWS Glue でこの接続に使用する secretName を選択して、トークンを配置します。

e. ネットワークを使用する場合は、ネットワークオプションを選択します。

3. AWS Glue ジョブに関連付けられている IAM ロールに secretName を読み取るアクセス許可を付与します。

Datadog エンティティからの読み取り

前提条件

読み取り元の Datadog オブジェクト。使用可能なエンティティを確認するには、以下のサポートされているエンティティの表を参照してください。

サポートされているエンティティ

エンティティ	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
メトリクスの時系列	はい	いいえ	いいえ	はい	いいえ
ログクエリ	はい	はい	はい	はい	いいえ

例

```
Datadog_read = glueContext.create_dynamic_frame.from_options(
    connection_type="datadog",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "log-queries",
        "API_VERSION": "v2",
        "INSTANCE_URL": "https://api.datadoghq.com",
        "FILTER_PREDICATE": "from = `2023-10-03T09:00:26Z`"
    }
```

Datadog エンティティとフィールドの詳細

エンティティ	フィールド	データタイプ	サポートされている演算子
メトリクスの時系列	エラー	String	NA
	aggr	String	NA
	属性	Struct	NA
	display_name	String	NA
	end	DateTime	NA
	expression	String	NA
	interval	整数	NA
	length	整数	NA
	metric	String	NA
	pointlist	リスト	NA
	query_index	整数	NA
	scope	String	NA
	start	DateTime	NA
	tag_set	リスト	NA
	単位	Struct	NA
	from_to_date	DateTime	BETWEEN
	query	String	EQUAL_TO
	ステータス	String	NA
	type	String	NA

エンティティ	フィールド	データタイプ	サポートされている演算子
	host	String	NA
ログクエリ	id	String	NA
	属性	Struct	NA
	timestamp	DateTime	NA
	type	String	NA
	送信元	DateTime	BETWEEN,EQUAL_TO
	indexes	リスト	EQUAL_TO
	storage_tier	String	EQUAL_TO
	query	String	EQUAL_TO

Datadog 接続オプション

Datadog の接続オプションは次のとおりです。

- ENTITY_NAME(文字列) – (必須) 読み取り / 書き込みに使用されます。Datadog のオブジェクトの名前。
- API_VERSION(文字列) – (必須) 読み取り / 書き込みに使用されます。使用する Datadog Rest API バージョン。v1 バージョンは metrics-timeseries エンティティをサポートし、v2 バージョンは log-queries エンティティをサポートします。
- INSTANCE_URL(文字列) – (必須) 読み取りに使用されます。Datadog インスタンス URL。Datadog インスタンスの URL はリージョンによって異なります。
- SELECTED_FIELDS(List<String>) – Default: empty(SELECT *). 読み込みに使用されます。オブジェクトに選択する列。
- FILTER_PREDICATE(文字列) – デフォルト: 空 読み込みに使用されます。Spark SQL 形式である必要があります。
- QUERY(文字列) – デフォルト: 空 読み込みに使用されます。完全な Spark SQL クエリ。

Datadog アカウントの作成

1. https://www.datadoghq.com/ に移動します。
2. [無料で開始する] を選択します。
3. 必須事項を入力し、サインアップします。
4. 提案に従って [Datadog エージェントインストーラ] をインストールします。
5. アカウントが、アクティブな Datadog サブスクリプションを持つ有効な組織 (利用可能なリストから) に登録されていることを確認します。
6. Datadog アカウントにログインしたら、右上隅のユーザー名にカーソルを合わせると、[キー] の詳細が表示されます。

   a. API キーを取得するには、[API キー] を選択します。

   b. アプリケーションキーを取得するには、[アプリケーションキー] を選択します。

制限

Datadog コネクタの制限は次のとおりです:

- Datadog は、フィールドベースまたはレコードベースのパーティション分割をサポートしていません。
- `from` は `Log Queries` エンティティの必須フィルターパラメータです。
- `from_to_date` および `query` は `Metrics Timeseries` エンティティの必須フィルターパラメータです。

Docusign Monitor への接続

Docusign Monitor は、24 時間体制でアクティビティを追跡し、組織が契約を保護するのを支援します。Monitor API は、このアクティビティ追跡情報を既存のセキュリティスタックまたはデータ可視化ツールに直接配信します。これにより、チームは不正なアクティビティを検出し、インシデントを調査し、検証済みの脅威に迅速に対応できます。また、セキュリティチームが特定のビジネスニーズに合わせてダッシュボードとアラートをカスタマイズするために必要な柔軟性も提供します。

トピック

- Docusign Monitor の AWS Glue サポート
- 接続を作成および使用するための API オペレーションを含むポリシー

- Docusign Monitor の設定
- Docusign Monitor 接続の設定
- Docusign Monitor エンティティからの読み取り
- Docusign Monitor 接続オプション
- Docusign Monitor の制限事項

Docusign Monitor の AWS Glue サポート

AWS Glue は、次のように Docusign Monitor をサポートします。

ソースとしてサポートされていますか?

はい。AWS Glue ETL ジョブを使用して、Docusign Monitor からデータをクエリできます。

ターゲットとしてサポートされていますか?

いいえ。

サポートされている Docusign Monitor API バージョン

次の Docusign Monitor API バージョンがサポートされています。

- v2.0

接続を作成および使用するための API オペレーションを含むポリシー

次のサンプル ポリシーで、接続の作成と使用に必要な AWS IAM 権限について説明します。新しいロールを作成する場合は、以下を含むポリシーを作成します。

```
{
 "Version": "2012-10-17",
    "Statement": [
     {
            "Effect": "Allow",
            "Action": [
                "glue:ListConnectionTypes",
                "glue:DescribeConnectionType",
                "glue:RefreshOAuth2Tokens",
                "glue:ListEntities",
```

```
                "glue:DescribeEntity"
            ],
            "Resource": "*"
        }
    ]
}
```

上記の方法を使用しない場合は、代わりに次のマネージド IAM ポリシーを使用します:

- AWSGlueServiceRole – さまざまな AWS Glue プロセスを実行するために必要なリソースへのアクセス権をユーザーに代わって付与します。これらのリソースには AWS Glue 、Amazon S3、IAM、CloudWatch Logs、および Amazon EC2 が含まれます。このポリシーで指定されたリソースの命名規則に従った場合、AWS Glue プロセスは必要なアクセス権限を使用できます。このポリシーは、通常、クローラ、ジョブ、開発エンドポイントを定義するときに指定されたロールにアタッチされます。
- AWSGlueConsoleFullAccess — ポリシーがアタッチされているアイデンティティが AWS マネージメントコンソールを使用するときは、AWS Glue リソースへのフルアクセスを許可します。このポリシーで指定されたリソースの命名規則に従った場合、ユーザーは完全なコンソール機能を使用できます。このポリシーは、通常 AWS Glue コンソールのユーザーにアタッチされています。

Docusign Monitor の設定

AWS Glue を使用して Docusign Monitor からサポートされている送信先にデータを転送できるようにするには、次の要件を満たす必要があります。

最小要件

以下に、最小要件を示します。

- Docusign Monitor で Docusign Software 製品を使用する Docusign アカウントを持っている。
- Docusign アカウントのデベロッパーコンソールで、AWS Glue 用の OAuth 2.0 統合アプリケーションを作成した。

  このアプリにより、AWS Glue がお客様のアカウントに対して認可呼び出しを行うときに、お客様のデータに安全にアクセスするために使用するクライアントの認証情報が提供されます。詳細については、Docusign Monitor ドキュメントの「OAuth 2.0」を参照してください。

これらの要件を満たすと、Docusign Monitor アカウントに AWS Glue を接続する準備が整います。

Docusign Monitor 接続の設定

Docusign Monitor は、AUTHORIZATION_CODE グラントタイプをサポートしています。

- このグラントタイプは、ユーザーを認証するためにサードパーティーの認証サーバーにユーザーをリダイレクトすることから、three-legged の OAuth と見なされます。AWS Glue コンソール経由で接続を作成するときに使用されます。
- ユーザーは、Docusign Monitor で独自の接続アプリを作成し、AWS Glue コンソールを介して接続を作成するときに独自のクライアント ID とクライアントシークレットを指定することを選択できます。このシナリオでは、引き続き Docusign Monitor にリダイレクトされてログインし、リソースへアクセスするために AWS Glue を認可します。
- このグラントタイプは、更新トークンとアクセストークンになります。アクセストークンの有効期間は短く、更新トークンを使用してユーザーとやり取りすることなく自動的に更新される場合があります。
- 認可コード OAuth フロー用の接続アプリの作成に関する Docusign Monitor の公開ドキュメントについては、「OAuth for Docusign Connect」を参照してください。

Docusign Monitor 接続を設定するには:

1. AWS Secrets Manager で、次の詳細を含むシークレットを作成します。
   a. カスタマーマネージド接続アプリケーションの場合、シークレットには、USER_MANAGED_CLIENT_APPLICATION_CLIENT_SECRET を使用して接続されたアプリケーションの API キーをキーとして含める必要があります。
   b. 注: AWS Glue で接続にシークレットを作成する必要があります。

1. AWS Glue Glue Studio で、以下の手順に従って [データ接続] の下に接続を作成します。
   a. [接続] で [接続を作成] を選択します。
   b. [データソース] を選択する際に、[Docusign Monitor] を選択します。
   c. 次のアクションを実行でき、AWS Glue がその権限を持つ AWS IAM ロールを選択します。

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

```
                "secretsmanager:DescribeSecret",
                "secretsmanager:GetSecretValue",
                "secretsmanager:PutSecretValue",
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterface",
                "ec2:DeleteNetworkInterface",
            ],
            "Resource": "*"
        }
    ]
}
```

d. Docusign Monitor アプリの [ユーザー管理クライアントアプリケーション ClientId] を指定します。

e. AWS Glue でこの接続に使用する `secretName` を選択して、トークンを配置します。

f. ネットワークを使用する場合は、ネットワークオプションを選択します。

2. AWS Glue ジョブに関連付けられている IAM ロールに `secretName` を読み取るアクセス許可を付与します。

Docusign Monitor エンティティからの読み取り

前提条件

読み取り元の Docusign Monitor オブジェクト。

ソースに対応するエンティティ:

エンティティ	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
データのモニタリング	あり	はい	なし	あり	不可

例:

```
docusignmonitor_read = glueContext.create_dynamic_frame.from_options(
    connection_type="docusign_monitor",
    connection_options={
```

```
        "connectionName": "connectionName",
        "ENTITY_NAME": "monitoring-data",
        "API_VERSION": "v2.0"
    }
```

Docusign Monitor エンティティとフィールドの詳細

静的メタデータを持つエンティティ:

エンティティ	フィールド	データ型	サポートされている演算子
データのモニタリング	timestamp	DateTime	該当なし
	eventId	String	該当なし
	アプリケーション	String	該当なし
	環境	String	該当なし
	site	String	該当なし
	traceToken	String	該当なし
	organizationId	String	該当なし
	accountId	String	該当なし
	userId	String	該当なし
	オブジェクト	String	該当なし
	アクション	String	該当なし
	property	String	該当なし
	field	String	該当なし
	result	String	該当なし
	IntegratorKey	String	該当なし

エンティティ	フィールド	データ型	サポートされている演算子
	customerVisible	String	該当なし
	version	String	該当なし
	userAgent	String	該当なし
	userAgentClientInfo	Struct	該当なし
	ipAddress	String	該当なし
	ipAddressLocation	Struct	該当なし
	データ	String	該当なし
	ソース	String	該当なし
	latitude	倍精度	該当なし
	longitude	倍精度	該当なし
	city	String	該当なし
	state	String	該当なし
	country	String	該当なし
	usUserMemberOfDomain	ブール値	該当なし
	affectedUserIsMemberOfDomain	ブール値	該当なし
	proxyStatus	String	該当なし
	proxyType	String	該当なし
	proxyLevel	String	該当なし
	referencedUserId	String	該当なし

エンティティ	フィールド	データ型	サポートされている演算子
	デバイス	String	該当なし
	ブラウザ	String	該当なし
	cursor	DateTime	EQUAL_TO

パーティショニングクエリ

Docusign Monitor は、フィールドベースまたはレコードベースのパーティショニングをサポートしていません。

Docusign Monitor 接続オプション

Docusign Monitor の接続オプションは次のとおりです。

- `ENTITY_NAME`(文字列) – (必須) 読み取りに使用されます。Docusign Monitor のオブジェクトの名前。
- `API_VERSION`(文字列) – (必須) 読み取りに使用されます。使用する Docusign Monitor Rest API バージョン。
- `SELECTED_FIELDS`(List<String>) – Default: empty(SELECT *). 読み込みに使用されます。オブジェクトに選択する列。
- `QUERY`(文字列) – デフォルト: 空 読み込みに使用されます。完全な Spark SQL クエリ。
- `FILTER_PREDICATE`(文字列) – デフォルト: 空 読み込みに使用されます。Spark SQL 形式である必要があります。

Docusign Monitor の制限事項

Docusign Monitor の制限または注意事項は次のとおりです。

- `cursor` フィールドを使用してフィルターを適用すると、API は指定された日付から 7 日間のレコードを取得します。
- フィルターが指定されていない場合、API は API リクエストの現在の日付から過去 7 日間のレコードを取得します。

- Docusign Monitor は、フィールドベースまたはレコードベースのパーティショニングをサポートしていません。
- Docusign Monitor は Order By 機能をサポートしていません。

Domo への接続

Domo はクラウドベースのダッシュボードツールです。Domo のエンタープライズアプリケーションプラットフォームでは、Domo を拡張するために必要な基盤が整っているため、カスタムソリューションをより迅速に構築できます。

トピック

- Domo の AWS Glue サポート
- 接続を作成および使用するための API オペレーションを含むポリシー
- Domo の設定
- Domo 接続の設定
- Domo エンティティからの読み取り
- Domo 接続オプション
- Domo の制限事項

Domo の AWS Glue サポート

AWS Glue は、次のように Domo をサポートします。

ソースとしてサポートされていますか?

はい。AWS Glue ETL ジョブを使用して、Domo からデータをクエリできます。

ターゲットとしてサポートされていますか?

いいえ。

サポートされている Domo API バージョン

次の Domo API バージョンがサポートされています。

- v1

接続を作成および使用するための API オペレーションを含むポリシー

次のサンプル ポリシーで、接続の作成と使用に必要な AWS IAM 権限について説明します。新しいロールを作成する場合は、以下を含むポリシーを作成します。

```
{
 "Version": "2012-10-17",
    "Statement": [
     {
            "Effect": "Allow",
            "Action": [
                "glue:ListConnectionTypes",
                "glue:DescribeConnectionType",
                "glue:RefreshOAuth2Tokens",
                "glue:ListEntities",
                "glue:DescribeEntity"
            ],
            "Resource": "*"
        }
    ]
 }
```

上記の方法を使用しない場合は、代わりに次のマネージド IAM ポリシーを使用します:

- AWSGlueServiceRole – さまざまな AWS Glue プロセスを実行するために必要なリソースへのアクセス権をユーザーに代わって付与します。これらのリソースには AWS Glue 、Amazon S3、IAM、CloudWatch Logs、および Amazon EC2 が含まれます。このポリシーで指定されたリソースの命名規則に従った場合、AWS Glue プロセスは必要なアクセス権限を使用できます。このポリシーは、通常、クローラ、ジョブ、開発エンドポイントを定義するときに指定されたロールにアタッチされます。
- AWSGlueConsoleFullAccess — ポリシーがアタッチされているアイデンティティが AWS マネージメントコンソールを使用するときは、AWS Glue リソースへのフルアクセスを許可します。このポリシーで指定されたリソースの命名規則に従った場合、ユーザーは完全なコンソール機能を使用できます。このポリシーは、通常 AWS Glue コンソールのユーザーにアタッチされています。

Domo の設定

AWS Glue を使用して Domo からサポートされている送信先にデータを転送できるようにするには、次の要件を満たす必要があります。

最小要件

以下に、最小要件を示します。

- API アクセスが有効になっている Domo アカウントがある。
- Domo デベロッパーアカウントの下には、AWS Glue がアカウントに対して認可呼び出しを行うときにデータに安全にアクセスするために使用するクライアント認証情報を提供するアプリケーションがある。詳細については、「Domo デベロッパーアプリケーションの作成」を参照してください。

これらの要件を満たすと、Domo アカウントに AWS Glue を接続する準備が整います。

Domo デベロッパーアプリケーションの作成

[クライアント ID] と [クライアントシークレット] を取得するには、デベロッパーアカウントを作成します。

1. Domo デベロッパーログインページに移動します。
2. [ログイン] を選択します。
3. ドメイン名を指定し、[Continue] をクリックします。
4. [My Account] にカーソルを合わせ、[New Client] を選択します。
5. 名前と説明を入力し、スコープ (「データ」) を選択して、[Create] を選択します。
6. 作成された新しいクライアントから、生成された [クライアント ID] と [クライアントシークレット] を取得します。

Domo 接続の設定

Domo は、OAuth2 の CLIENT_CREDENTIALS グラントタイプをサポートしています。

- このグラントタイプは、クライアントアプリケーションのみがサーバーに対して自身を認可し、ユーザーには関与しないため、2 レッグの OAuth と見なされます。
- ユーザーは、Domo で独自の接続アプリを作成し、AWS Glue コンソールを介して接続を作成するときに独自のクライアント ID とクライアントシークレットを指定することを選択できます。
- 認可コード OAuth フロー用の接続アプリの作成に関する Domo の公開ドキュメントについては、「OAuth Authentication」を参照してください。

Domo 接続を設定するには:

1. AWS Secrets Manager で、次の詳細を含むシークレットを作成します。
   a. カスタマーマネージド接続アプリの場合、シークレットには接続アプリのアクセストークン、client_id、および client_secret を含める必要があります。
   b. 注: AWS Glue で接続にシークレットを作成する必要があります。

1. AWS Glue Glue Studio で、以下の手順に従って [データ接続] の下に接続を作成します。
   a. [接続] で [接続を作成] を選択します。
   b. [データソース] を選択する際に、[Domo] を選択します。
   c. 次のアクションを実行でき、AWS Glue がその権限を持つ AWS IAM ロールを選択します。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:DescribeSecret",
                "secretsmanager:GetSecretValue",
                "secretsmanager:PutSecretValue",
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterface",
                "ec2:DeleteNetworkInterface",
            ],
            "Resource": "*"
        }
    ]
}
```

   d. AWS Glue でこの接続に使用する secretName を選択して、トークンを配置します。
   e. ネットワークを使用する場合は、ネットワークオプションを選択します。
2. AWS Glue ジョブに関連付けられている IAM ロールに secretName を読み取るアクセス許可を付与します。

Domo エンティティからの読み取り

前提条件

読み取り元の Domo オブジェクト。Data Set や Data Permission Policies などのオブジェクト名が必要です。次の表に、サポートされているエンティティを示します。

ソースに対応するエンティティ:

エンティティ	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
データセット	あり	あり	あり	あり	あり
Data Permission Policies	いいえ	いいえ	なし	あり	不可

例:

```
Domo_read = glueContext.create_dynamic_frame.from_options(
    connection_type="domo",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "dataset",
        "API_VERSION": "v1"
    }
```

Domo エンティティとフィールドの詳細

静的メタデータを持つエンティティ:

エンティティ	フィールド	データ型	サポートされている演算子
Data Permission Policies	id	Long	該当なし
	type	String	該当なし
	名前	String	該当なし
	フィルター	リスト	該当なし

エンティティ	フィールド	データ型	サポートされている演算子
	ユーザー	リスト	該当なし
	virtualUsers	リスト	該当なし
	グループ	リスト	該当なし

次のエンティティについて、Domo はメタデータを動的に取得するエンドポイントを提供するため、演算子のサポートはエンティティのデータ型レベルでキャプチャされます。

エンティティ	データ型	サポートされている演算子
データセット	整数	=, !=, <, >, >=, <=
	Long	=, !=, <, >, >=, <=
	String	=、!=、CONTAINS
	日付	=、>、>=、<、<=、BETWEEN
	DateTime	=、>、>=、<、<=、BETWEEN
	ブール値	=, !=
	倍精度	=, !=, <, >, >=, <=
	リスト	該当なし
	Struct	該当なし

パーティショニングクエリ

フィールドベースのパーティション分割

Spark で同時実行を使用する場合は、追加の Spark オプション PARTITION_FIELD、LOWER_BOUND、UPPER_BOUND、および NUM_PARTITIONS を指定できます。これらのパラメータを使用すると、元のクエリは Spark タスクで同時に実行できるサブクエリの NUM_PARTITIONS の数に分割されます。

- PARTITION_FIELD: クエリのパーティション化に使用するフィールドの名前。
- LOWER_BOUND: 選択したパーティションフィールドの包括的な下限値。

  DateTime フィールドでは、ISO 形式の値を受け入れます。

  有効な値の例:

  ```
  "2023-01-15T11:18:39.205Z"
  ```

  Date フィールドでは、ISO 形式の値を受け入れます。

  有効な値の例:

  ```
  "2023-01-15"
  ```
- UPPER_BOUND: 選択したパーティションフィールドの排他的上限値。

  有効な値の例:

  ```
  "2023-02-15T11:18:39.205Z"
  ```
- NUM_PARTITIONS: パーティション数。

エンティティごとのパーティション分割フィールドのサポートの詳細は、次の表にまとめられています。

エンティティ名	パーティション分割フィールド	データ型
データセット	任意の日付/時刻ベースのフィールド [動的メタデータ]	DateTime
	任意の日付ベースのフィールド [動的メタデータ]	日付

例:

```
Domo_read = glueContext.create_dynamic_frame.from_options(
    connection_type="domo",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "dataset",
        "API_VERSION": "v1",
        "PARTITION_FIELD": "permissionTime"
        "LOWER_BOUND": "2023-01-15T11:18:39.205Z"
        "UPPER_BOUND": "2023-02-15T11:18:39.205Z"
        "NUM_PARTITIONS": "2"
    }
```

レコードベースのパーティション分割

Spark で並行処理を使用する場合は、追加の Spark オプション NUM_PARTITIONS を指定できます。これらのパラメータを使用すると、元のクエリは Spark タスクで同時に実行できるサブクエリの NUM_PARTITIONS の数に分割されます。

レコードベースのパーティショニングでは、存在するレコードの合計数が Domo からクエリされ、指定された NUM_PARTITIONS 数で割られます。その後、結果のレコード数は、各サブクエリによって同時に取得されます。

例:

```
Domo_read = glueContext.create_dynamic_frame.from_options(
    connection_type="domo",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "dataset",
        "API_VERSION": "v1",
        "NUM_PARTITIONS": "2"
    }
```

Domo 接続オプション

Domo の接続オプションは次のとおりです。

- ENTITY_NAME(文字列) – (必須) 読み取りに使用されます。Domo のオブジェクトの名前。
- API_VERSION(文字列) – (必須) 読み取りに使用されます。使用する Domo Rest API バージョン。

- SELECTED_FIELDS(List<String>) – Default: empty(SELECT *). 読み込みに使用されます。オブジェクトに選択する列。
- FILTER_PREDICATE(文字列) – デフォルト: 空 読み込みに使用されます。Spark SQL 形式である必要があります。
- QUERY(文字列) - デフォルト: 空 読み込みに使用されます。完全な Spark SQL クエリ。
- PARTITION_FIELD (文字列) – 読み取りに使用されます。クエリをパーティション化するために使用するフィールド。
- LOWER_BOUND (文字列)– 読み取りに使用されます。選択したパーティションフィールドの包括的な下限値。
- UPPER_BOUND (文字列) – 読み取りに使用されます。選択したパーティションフィールドの排他的上限値。
- NUM_PARTITIONS(整数) – デフォルト: 1。読み込みに使用されます。読み取り用のパーティションの数。

Domo の制限事項

Domo の制限事項または注意事項は次のとおりです。

- SDK の制限により、「_」で始まるクエリ可能なフィールド (_BATCH_ID など) では、フィルタリングが期待どおりに機能しません。
- API の制限により、フィルタリングは指定された日付より前の日付で機能します。これは増分プルにも影響します。この制限を回避するには、UTC を基準にしたタイムゾーンに従って日付を選択し、必要な日付のデータを取得します。

Dynatrace への接続

Dynatrace は、包括的なオブザーバビリティとセキュリティのための分析と自動化を提供するプラットフォームです。アプリケーションのパフォーマンス、インフラストラクチャ、ユーザーエクスペリエンスのモニタリングと最適化に特化しています。

トピック

- Dynatrace の AWS Glue サポート
- 接続を作成および使用するための API オペレーションを含むポリシー
- Dynatrace の設定
- Dynatrace 接続の設定

- Dynatrace エンティティからの読み取り
- Dynatrace の接続オプション
- Dynatrace の制限事項

Dynatrace の AWS Glue サポート

AWS Glue は、次のように Dynatrace をサポートします。

ソースとしてサポートされていますか?

はい。AWS Glue ETL ジョブを使用して、Dynatrace からデータをクエリできます。

ターゲットとしてサポートされていますか?

いいえ。

サポートされている Dynatrace API バージョン

次の Dynatrace API バージョンがサポートされています。

- v2

接続を作成および使用するための API オペレーションを含むポリシー

次のサンプル ポリシーで、接続の作成と使用に必要な AWS IAM 権限について説明します。新しいロールを作成する場合は、以下を含むポリシーを作成します。

```
{
 "Version": "2012-10-17",
     "Statement": [
      {
             "Effect": "Allow",
             "Action": [
                 "glue:ListConnectionTypes",
                 "glue:DescribeConnectionType",
                 "glue:RefreshOAuth2Tokens",
                 "glue:ListEntities",
                 "glue:DescribeEntity"
             ],
             "Resource": "*"
         }
```

```
    ]
}
```

上記の方法を使用しない場合は、代わりに次のマネージド IAM ポリシーを使用します:

- AWSGlueServiceRole – さまざまな AWS Glue プロセスを実行するために必要なリソースへのアクセス権をユーザーに代わって付与します。これらのリソースには AWS Glue 、Amazon S3、IAM、CloudWatch Logs、および Amazon EC2 が含まれます。このポリシーで指定されたリソースの命名規則に従った場合、AWS Glue プロセスは必要なアクセス権限を使用できます。このポリシーは、通常、クローラ、ジョブ、開発エンドポイントを定義するときに指定されたロールにアタッチされます。
- AWSGlueConsoleFullAccess — ポリシーがアタッチされているアイデンティティが AWS マネージメントコンソールを使用するときは、AWS Glue リソースへのフルアクセスを許可します。このポリシーで指定されたリソースの命名規則に従った場合、ユーザーは完全なコンソール機能を使用できます。このポリシーは、通常 AWS Glue コンソールのユーザーにアタッチされています。

Dynatrace の設定

AWS Glue を使用して Dynatrace からデータを転送できるようにするには、次の要件を満たす必要があります。

最小要件

以下に、最小要件を示します。

- Dynatrace アカウントがある。
- API にアクセスするために割り当てられた適切な読み取り/書き込みスコープを使用して、Dynatrace アカウントにアクセストークンを生成している。詳細については、「Generate a token」を参照してください。

これらの要件を満たすと、Dynatrace アカウントに AWS Glue を接続する準備が整います。

Dynatrace 接続の設定

Dynatrace はカスタム認可をサポートしています。

Dynatrace 接続を設定するには

1. AWS Secrets Manager で、次の詳細を含むシークレットを作成します。

a. カスタマーマネージド接続アプリケーションの場合、シークレットには、*apiToken* を使用して接続されたアプリケーションの API キーをキーとして含める必要があります。

b. 注: AWS Glue で接続にシークレットを作成する必要があります。

1. AWS Glue Glue Studio で、以下の手順に従って [データ接続] の下に接続を作成します。
   a. [データソース] を選択するときは、[Dynatrace] を選択します。
   b. 接続先の Dynatrace アカウントの `INSTANCE_URL` を指定します。
   c. 次のアクションを実行でき、AWS Glue がその権限を持つ AWS IAM ロールを選択します。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:DescribeSecret",
                "secretsmanager:GetSecretValue",
                "secretsmanager:PutSecretValue",
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterface",
                "ec2:DeleteNetworkInterface",
            ],
            "Resource": "*"
        }
    ]
}
```

   d. AWS Glue でこの接続に使用する `secretName` を選択して、トークンを配置します。
   e. ネットワークを使用する場合は、ネットワークオプションを選択します。
2. AWS Glue ジョブに関連付けられている IAM ロールに `secretName` を読み取るアクセス許可を付与します。

Dynatrace エンティティからの読み取り

前提条件

読み取り元の Dynatrace オブジェクト。「problem」などのオブジェクト名が必要です。

ソースに対応するエンティティ:

エンティティ	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
問題	あり	あり	あり	はい	不可

例:

```
Dynatrace_read = glueContext.create_dynamic_frame.from_options(
    connection_type="Dynatrace",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "problem",
        "API_VERSION": "v2",
        "INSTANCE_URL": "https://[instanceName].live.dynatrace.com"
    }
```

Dynatrace エンティティとフィールドの詳細:

Dynatrace では、サポートされているエンティティのメタデータを動的に取得するためのエンドポイントが提供されます。したがって、オペレータのサポートはデータ型レベルでキャプチャされます。

エンティティ	フィールド	データ型	サポートされている演算子
問題	affectedEntities	リスト	EQUAL_TO
	displayId	String	EQUAL_TO
	endTime	DateTime	
	entityTags	リスト	
	evidenceDetails	Struct	
	impactAnalysis	Struct	
	impactLevel	String	EQUAL_TO

エンティティ	フィールド	データ型	サポートされている演算子
	impactedEntities	リスト	EQUAL_TO
	linkedProblemInfo	Struct	
	managementZones	リスト	EQUAL_TO
	problemFilters	リスト	
	recentComments	Struct	
	rootCauseEntity	Struct	EQUAL_TO
	problemId	String	EQUAL_TO
	severityLevel	String	EQUAL_TO
	startTime	DateTime	BETWEEN
	ステータス	String	EQUAL_TO
	title	String	
	送信元	DateTime	EQUAL_TO、BETWEEN
	problemFilterIds	String	EQUAL_TO
	problemFilterNames	String	EQUAL_TO
	managementZoneIds	String	EQUAL_TO
	text	String	EQUAL_TO
	underMaintenance	ブール値	EQUAL_TO
	message	String	

パーティショニングクエリ

Dynatrace は、フィールドベースまたはレコードベースのパーティショニングをサポートしていません。

Dynatrace の接続オプション

Dynatrace の接続オプションは次のとおりです。

- ENTITY_NAME(文字列) – (必須) 読み取りに使用されます。Dynatrace のオブジェクトの名前。
- API_VERSION(文字列) – (必須) 読み取りに使用されます。使用する Dynatrace Rest API バージョン。
- INSTANCE_URL(文字列) – 読み取りに使用されます。有効な Dynatrace インスタンス URL。
- SELECTED_FIELDS(List<String>) – Default: empty(SELECT *). 読み込みに使用されます。オブジェクトに選択する列。
- FILTER_PREDICATE(文字列) – デフォルト: 空 読み込みに使用されます。Spark SQL 形式である必要があります。
- QUERY(文字列) - デフォルト: 空 読み込みに使用されます。完全な Spark SQL クエリ。

Dynatrace の制限事項

Dynatrace の制限事項または注意事項は次のとおりです。

- Dynatrace は、フィールドベースまたはレコードベースのパーティション化をサポートしていません。
- Select All 機能では、フィルターに「field」を指定した場合、レコード数はページあたり 10 を超えることはできません。
- サポートされる最大ページサイズは 500 です。フローの作成時に [evidenceDetails, impactAnalysis, recentComments] フィールドのいずれかを選択すると、ページあたりのレコード数はデフォルトで 10 になります。

Facebook 広告への接続

Facebook 広告は、あらゆる規模の企業がターゲットオーディエンスに到達し、さまざまなマーケティング目標を達成するために使用する強力なデジタル広告プラットフォームです。このプラットフォームにより、広告主は、Facebook や Messenger などのFacebook のアプリやサービス群全体で

表示できるカスタマイズされた広告を作成できます。Facebook 広告は高度なターゲティング機能を備えているため、企業は特定のターゲット層、関心、行動、場所にまでリーチすることができます。

トピック

- AWS Glue による Facebook 広告のサポート
- 接続を作成および使用するための API オペレーションを含むポリシー
- Facebook 広告の設定
- Facebook 広告接続の設定
- Facebook 広告エンティティからの読み取り
- Facebook 広告の接続オプション
- Facebook 広告コネクタの制限と注意事項

AWS Glue による Facebook 広告のサポート

AWS Glue は、次のように Facebook 広告をサポートします:

ソースとしてサポートされていますか?

はい。AWS Glue ETL ジョブを使用して、Facebook 広告からデータをクエリできます。

ターゲットとしてサポートされているかどうか

いいえ。

サポートされている Facebook Ads API バージョン

以下の Facebook Ads API バージョンがサポートされています:

- v17.0
- v18.0
- v19.0
- v20.0

接続を作成および使用するための API オペレーションを含むポリシー

次のサンプル ポリシーで、接続の作成と使用に必要な AWS IAM 権限について説明します。新しいロールを作成する場合は、以下を含むポリシーを作成します。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:ListConnectionTypes",
                "glue:DescribeConnectionType",
                "glue:RefreshOAuth2Tokens"
                "glue:ListEntities",
                "glue:DescribeEntity"
            ],
            "Resource": "*"
        }
    ]
}
```

上記の方法を使用しない場合は、代わりに次のマネージド IAM ポリシーを使用します:

- AWSGlueServiceRole – さまざまな AWS Glue プロセスを実行するために必要なリソースへのアクセス権をユーザーに代わって付与します。これらのリソースには AWS Glue 、Amazon S3、IAM、CloudWatch Logs、および Amazon EC2 が含まれます。このポリシーで指定されたリソースの命名規則に従った場合、AWS Glue プロセスは必要なアクセス権限を使用できます。このポリシーは、通常、クローラ、ジョブ、開発エンドポイントを定義するときに指定されたロールにアタッチされます。
- AWSGlueConsoleFullAccess — ポリシーがアタッチされているアイデンティティが AWS マネージメントコンソールを使用するときは、AWS Glue リソースへのフルアクセスを許可します。このポリシーで指定されたリソースの命名規則に従った場合、ユーザーは完全なコンソール機能を使用できます。このポリシーは、通常 AWS Glue コンソールのユーザーにアタッチされています。

Facebook 広告の設定

AWS Glue を使用して Facebook 広告からデータを転送できるようにするには、次の要件を満たす必要があります:

最小要件

以下に、最小要件を示します。

- Facebook スタンダードアカウントは Facebook から直接アクセスできること。

- アクセストークンを生成するには、ユーザー認証が必要です。
- Facebook 広告の SDK コネクタは、ユーザーアクセストークン OAuth フローを実装すること。
- Facebook 広告への API リクエストの認証には OAuth2.0 を使用すること。このウェブベースの認証は、2FA 上位セットである多要素認証 (MFA) アーキテクチャに該当します。
- ユーザーは、エンドポイントにアクセスするためのアクセス許可を付与する必要があります。ユーザーのデータにアクセスするために、エンドポイント認可はアクセス許可と機能を通じて処理されます。

OAuth 2.0 認証情報の取得

API 認証情報を取得してインスタンスに対して認証呼び出しを行うには、Facebook 広告デベロッパーガイドの「REST API」を参照してください。

Facebook 広告接続の設定

Facebook 広告は、OAuth2 の AUTHORIZATION_CODE のグラントタイプをサポートしています。

- このグラントタイプは、ユーザーを認証するためにサードパーティーの認証サーバーにユーザーをリダイレクトすることから、three-legged の OAuth と見なされます。AWS Glue コンソール経由で接続を作成するときに使用されます。
- ユーザーは、Facebook 広告で独自の接続アプリを作成し、AWS Glue コンソールを介して接続を作成するときに独自のクライアント ID とクライアントシークレットを指定することを選択できます。このシナリオでは、引き続き Facebook 広告にリダイレクトされてログインし、リソースへアクセスするために AWS Glue を承認します。
- このグラントタイプは、アクセストークンになります。期限切れのシステムユーザートークンは、生成または更新された日付から 60 日間有効です。継続性を作成するには、開発者は 60 日以内にアクセストークンを更新する必要があります。そうしないと、アクセストークンは失われ、開発者は API アクセスを回復するために新しいトークンを取得する必要があります。「Refresh Access Token」を参照してください。
- 認可コード OAuth フロー用の接続アプリの作成に関する Facebook 広告の公開ドキュメントについては、Google の開発者向けガイドの「OAuth 2.0 を使用した Google API へのアクセス」を参照してください。

Facebook 広告接続を設定するには:

1. AWS Glue Glue Studio で、以下の手順に従って [データ接続] の下に接続を作成します。

a. [接続タイプ] を選択する際に、[Facebook 広告] を選択します。

b. 接続先の Facebook 広告インスタンスの `INSTANCE_URL` を指定します。

c. AWS Glue が次のアクションを引き受けて、その権限を持つことができる AWS IAM ロールを選択します。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:DescribeSecret",
                "secretsmanager:GetSecretValue",
                "secretsmanager:PutSecretValue",
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterface",
                "ec2:DeleteNetworkInterface",
            ],
            "Resource": "*"
        }
    ]
}
```

d. AWS Glue でこの接続に使用する `secretName` を選択して、トークンを配置します。

e. ネットワークを使用する場合は、ネットワークオプションを選択します。

2. AWS Glue ジョブに関連付けられている IAM ロールに `secretName` を読み取るアクセス許可を付与します。

Facebook 広告エンティティからの読み取り

前提条件

読み取り元の Facebook 広告オブジェクト。オブジェクト名が必要になります。次の表は、サポートされているエンティティを示しています。

ソースに対応するエンティティ:

エンティティ	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
Campaign	あり	はい	いいえ	はい	はい
Ad Set	あり	はい	いいえ	はい	はい
Ads	あり	はい	いいえ	はい	はい
Ad Creative	いいえ	はい	いいえ	はい	なし
Insights - Account	いいえ	はい	いいえ	はい	なし
Adaccounts	あり	はい	いいえ	はい	なし
Insights - Ad	あり	はい	いいえ	はい	はい
Insights - AdSet	あり	はい	いいえ	はい	はい
Insights - Campaign	あり	はい	いいえ	はい	はい

例:

```
FacebookAds_read = glueContext.create_dynamic_frame.from_options(
    connection_type="FacebookAds",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "entityName",
        "API_VERSION": "v20.0"
    }
```

Facebook 広告エンティティとフィールドの詳細

エンティティとフィールドの詳細については、以下を参照してください:

- Ad Account

- Campaign
- Ad Set
- Ad
- Ad Creative
- Insight 広告アカウント
- Insights 広告
- Insights AdSets
- Insights キャンペーン

詳細については、「Marketing API」を参照してください。

> **Note**
>
> 構造体データ型とリストデータ型は、コネクタの応答で文字列データ型に変換されます。

パーティショニングクエリ

Spark で同時実行を使用する場合は、追加の Spark オプション PARTITION_FIELD、LOWER_BOUND、UPPER_BOUND、および NUM_PARTITIONS を指定できます。これらのパラメータを使用すると、元のクエリは Spark タスクで同時に実行できるサブクエリの NUM_PARTITIONS の数に分割されます。

- PARTITION_FIELD: クエリのパーティション化に使用するフィールドの名前。
- LOWER_BOUND: 選択したパーティションフィールドの包括的な下限値。

  DateTime フィールドでは、Spark SQL クエリで使用される Spark タイムスタンプ形式を受け入れます。

  有効な値の例:

  ```
  "2022-01-01"
  ```

- UPPER_BOUND: 選択したパーティションフィールドの排他的上限値。
- NUM_PARTITIONS: パーティション数。

例：

```
FacebookADs_read = glueContext.create_dynamic_frame.from_options(
    connection_type="FacebookAds",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "entityName",
        "API_VERSION": "v20.0",
        "PARTITION_FIELD": "created_time"
        "LOWER_BOUND": "2022-01-01"
        "UPPER_BOUND": "2024-01-02"
        "NUM_PARTITIONS": "10"
    }
```

Facebook 広告の接続オプション

Facebook 広告の接続オプションは次のとおりです:

- ENTITY_NAME(文字列) – (必須) 読み取りに使用されます。Facebook 広告内のオブジェクトの名前。
- API_VERSION(文字列) – (必須) 読み取りに使用されます。使用する Facebook 広告 Rest API バージョン。例: v1。
- SELECTED_FIELDS(List<String>) - Default: empty(SELECT *). 読み込みに使用されます。オブジェクトに選択する列。
- FILTER_PREDICATE(文字列) - デフォルト: 空 読み込みに使用されます。Spark SQL 形式である必要があります。
- QUERY(文字列) - デフォルト: 空 読み込みに使用されます。完全な Spark SQL クエリ。
- PARTITION_FIELD (文字列) – 読み取りに使用されます。クエリをパーティション化するために使用するフィールド。
- LOWER_BOUND (文字列)– 読み取りに使用されます。選択したパーティションフィールドの包括的な下限値。
- UPPER_BOUND (文字列) – 読み取りに使用されます。選択したパーティションフィールドの排他的上限値。
- NUM_PARTITIONS(整数) - デフォルト: 1。読み込みに使用されます。読み取り用のパーティションの数。
- TRANSFER_MODE (文字列) – デフォルト: SYNC。非同期読み取りに使用されます。

Facebook 広告コネクタの制限と注意事項

Facebook 広告コネクタの制限事項または注意事項は次のとおりです:

- Facebook 広告は動的メタデータをサポートしているため、すべてのフィールドをクエリできます。すべてのフィールドがフィルタリングをサポートしており、データが利用可能な場合はレコードが取得されます。それ以外の場合、Facebook は適切なエラーメッセージとともに Bad リクエスト (400) レスポンスを返します。
- アプリの呼び出し回数は、1 時間のウィンドウ 200 中にユーザーが実行できる呼び出しの回数に、ユーザー数を掛けたものです。レート制限の詳細については、「Rate Limits」と「Business Use Case Rate Limits」を参照してください。

Facebook Page Insights に対する接続

Facebook ページを使用すると、企業やその他の利害関係者グループは Facebook.com ソーシャルネットワークのページを作成できます。企業はこれらのページを使用して、営業時間の共有、公表、オンラインによる顧客とのやりとりを行います。Facebook Page Insights ユーザーの場合は、AWS Glue を Facebook Page Insights アカウントに接続できます。Facebook Page Insights は、ETL ジョブのデータソースとして使用できます。これらのジョブを実行して、Facebook Page Insights から AWS サービスまたはその他のサポートされているアプリケーションにデータを転送します。

トピック

- Facebook Page Insights の AWS Glue サポート
- 接続を作成および使用するための API オペレーションを含むポリシー
- Facebook Page Insights の設定
- Facebook Page Insights 接続の設定
- Facebook Page Insights エンティティからの読み取り
- Facebook Page Insights 接続オプション
- Facebook Page Insights コネクタの制限事項と注意事項

Facebook Page Insights の AWS Glue サポート

AWS Glue は、次のように Facebook Page Insights をサポートします。

ソースとしてサポートされていますか?

はい。AWS Glue ETL ジョブを使用して、Facebook Page Insights からデータをクエリできます。

ターゲットとしてサポートされていますか?

いいえ。

サポートされている Facebook Page Insights API バージョン

次の Facebook Page Insights API バージョンがサポートされています。

- v17
- v18
- v19
- v20
- v21

接続を作成および使用するための API オペレーションを含むポリシー

次のサンプル ポリシーで、接続の作成と使用に必要な AWS IAM 権限について説明します。新しいロールを作成する場合は、以下を含むポリシーを作成します。

```
{
 "Version": "2012-10-17",
     "Statement": [
      {
             "Effect": "Allow",
             "Action": [
                 "glue:ListConnectionTypes",
                 "glue:DescribeConnectionType",
                 "glue:RefreshOAuth2Tokens",
                 "glue:ListEntities",
                 "glue:DescribeEntity"
             ],
             "Resource": "*"
         }
     ]
 }
```

上記の方法を使用しない場合は、代わりに次のマネージド IAM ポリシーを使用します:

- AWSGlueServiceRole – さまざまな AWS Glue プロセスを実行するために必要なリソースへのアクセス権をユーザーに代わって付与します。これらのリソースには AWS Glue 、Amazon S3、IAM、CloudWatch Logs、および Amazon EC2 が含まれます。このポリシーで指定されたリソースの命名規則に従った場合、AWS Glue プロセスは必要なアクセス権限を使用できます。このポリシーは、通常、クローラ、ジョブ、開発エンドポイントを定義するときに指定されたロールにアタッチされます。
- AWSGlueConsoleFullAccess — ポリシーがアタッチされているアイデンティティが AWS マネージメントコンソールを使用するときは、AWS Glue リソースへのフルアクセスを許可します。このポリシーで指定されたリソースの命名規則に従った場合、ユーザーは完全なコンソール機能を使用できます。このポリシーは、通常 AWS Glue コンソールのユーザーにアタッチされています。

Facebook Page Insights の設定

AWS Glue を使用して Facebook Page Insights からデータを転送できるようにするには、次の要件を満たす必要があります。

最小要件

以下に、最小要件を示します。

- Facebook スタンダードアカウントは Facebook から直接アクセスできること。
- アクセストークンを生成するには、ユーザー認証が必要です。
- Facebook Page Insights コネクタは、ユーザーアクセストークン OAuth フローを実装しています。
- コネクタは OAuth2.0 を使用して Facebook Page Insights への API リクエストを認証します。これは、2FA 上位セットである多要素認証 (MFA) アーキテクチャに該当します。これはウェブベースの認証です。
- ユーザーは、エンドポイントにアクセスするためのアクセス許可を付与する必要があります。ユーザーのデータにアクセスするために、エンドポイント認可はアクセス許可と機能を通じて処理されます。

Facebook Page Insights 接続の設定

Facebook Page Insights 接続を設定するには:

1. AWS Glue Glue Studio で、以下の手順に従って [データ接続] の下に接続を作成します。
   a. [接続タイプ] を選択する際に、[Facebook Page Insights] を選択します。

b. 次のアクションを実行でき、AWS Glue がその権限を持つ AWS IAM ロールを選択します。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:DescribeSecret",
                "secretsmanager:GetSecretValue",
                "secretsmanager:PutSecretValue",
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterface",
                "ec2:DeleteNetworkInterface",
            ],
            "Resource": "*"
        }
    ]
}
```

c. 認可コード URL を選択します。

d. AWS Glue でこの接続に使用する `secretName` を選択して、トークンを配置します。

e. ネットワークを使用する場合は、ネットワークオプションを選択します。

2. AWS Glue ジョブに関連付けられている IAM ロールに `secretName` を読み取るアクセス許可を付与します。

Facebook Page Insights エンティティからの読み取り

前提条件

読み取り元の Facebook Page Insights オブジェクト。オブジェクト名が必要になります。

ソースに対応するエンティティ:

エンティティ	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
ページの内容	はい	いいえ	はい	はい	はい

エンティティ	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
ページ CTA クリック	はい	いいえ	いいえ	はい	はい
ページエンゲージメント	はい	いいえ	いいえ	はい	はい
ページインプレッション	はい	いいえ	いいえ	はい	はい
ページの投稿	はい	いいえ	いいえ	はい	はい
ページ投稿のエンゲージメント	いいえ	いいえ	いいえ	はい	いいえ
ページ投稿のリアクション	いいえ	いいえ	いいえ	はい	いいえ
ページのリアクション	はい	いいえ	いいえ	はい	はい
ストーリー	はい	いいえ	いいえ	はい	はい
ページユーザー属性	はい	いいえ	いいえ	はい	はい
ページビデオビュー	はい	いいえ	いいえ	はい	はい
ページビュー	はい	いいえ	いいえ	はい	はい
ページの動画投稿	はい	いいえ	いいえ	はい	はい
ページ	いいえ	はい	いいえ	はい	いいえ

エンティティ	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
フィード	はい	はい	いいえ	はい	はい

例:

```
facebookPageInsights_read = glueContext.create_dynamic_frame. from options(
    connection_type="facebookpageinsights",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "entityName",
        "API_VERSION": "v21"
   }
```

Facebook Page Insights フィールドの詳細:

エンティティ	フィールド	データ型	サポートされている演算子
ページの内容	名前	String	該当なし
	期間	期間	EQUAL_TO
	Since	DateTime	EQUAL_TO
	値	リスト	該当なし
	タイトル	String	該当なし
	説明	String	該当なし
	description_from_api_doc	String	該当なし
	ID	String	該当なし
ページ CTA クリック	名前	String	該当なし

エンティティ	フィールド	データ型	サポートされている演算子
	期間	期間	EQUAL_TO
	Since	DateTime	EQUAL_TO
	値	リスト	該当なし
	タイトル	String	該当なし
	説明	String	該当なし
	description_from_api_doc	String	該当なし
	ID	String	該当なし
ページエンゲージメント	名前	String	該当なし
	期間	期間	EQUAL_TO
	Since	DateTime	EQUAL_TO
	値	リスト	該当なし
	タイトル	String	該当なし
	説明	String	該当なし
	description_from_api_doc	String	該当なし
	ID	String	該当なし
ページインプレッション	名前	String	該当なし
	期間	期間	EQUAL_TO
	Since	DateTime	EQUAL_TO
	値	リスト	該当なし

エンティティ	フィールド	データ型	サポートされている演算子
	タイトル	String	該当なし
	説明	String	該当なし
	description_from_api_doc	String	該当なし
	ID	String	該当なし
ページの投稿	名前	String	該当なし
	期間	期間	EQUAL_TO
	Since	DateTime	EQUAL_TO
	値	リスト	該当なし
	タイトル	String	該当なし
	説明	String	該当なし
	description_from_api_doc	String	該当なし
	ID	String	該当なし
ページ投稿のエンゲージメント	名前	String	該当なし
	期間	期間	EQUAL_TO
	値	リスト	該当なし
	タイトル	String	該当なし
	説明	String	該当なし
	description_from_api_doc	String	該当なし

エンティティ	フィールド	データ型	サポートされている演算子
	ID	String	該当なし
ページ投稿のリアクション	名前	String	該当なし
	期間	期間	EQUAL_TO
	値	リスト	該当なし
	タイトル	String	該当なし
	説明	String	該当なし
	description_from_api_doc	String	該当なし
	ID	String	該当なし
ページユーザー属性	名前	String	該当なし
	期間	期間	EQUAL_TO
	Since	DateTime	EQUAL_TO
	値	リスト	該当なし
	タイトル	String	該当なし
	説明	String	該当なし
	description_from_api_doc	String	該当なし
	ID	String	該当なし
ページビデオビュー	名前	String	該当なし
	期間	期間	EQUAL_TO
	Since	DateTime	EQUAL_TO

エンティティ	フィールド	データ型	サポートされている演算子
	値	リスト	該当なし
	タイトル	String	該当なし
	説明	String	該当なし
	description_from_api_doc	String	該当なし
	ID	String	該当なし
ページビュー	名前	String	該当なし
	期間	期間	EQUAL_TO
	Since	DateTime	EQUAL_TO
	値	リスト	該当なし
	タイトル	String	該当なし
	説明	String	該当なし
	description_from_api_doc	String	該当なし
	ID	String	該当なし
ページの動画投稿	名前	String	該当なし
	期間	期間	EQUAL_TO
	Since	DateTime	EQUAL_TO
	値	リスト	該当なし
	タイトル	String	該当なし
	説明	String	該当なし

エンティティ	フィールド	データ型	サポートされている演算子
	description_from_api_doc	String	該当なし
	ID	String	該当なし
ページ	名前	String	該当なし
	概要	String	該当なし
	access_token	String	該当なし
	ad_campaign	String	該当なし
	所属	String	該当なし
	app_id	String	該当なし
	artists_we_like	String	該当なし
	Attire	String	該当なし
	特典	String	該当なし
	band_interests	String	該当なし
	band_members	String	該当なし
	best_page	String	該当なし
	Bio	String	該当なし
	誕生日	String	該当なし
	booking_agent	String	該当なし
	Built	String	該当なし
	can_checkin	String	該当なし

エンティティ	フィールド	データ型	サポートされている演算子
	can_post	String	該当なし
	カテゴリ	String	該当なし
	category_list	リスト	該当なし
	Checkins	整数	該当なし
	company_overview	String	該当なし
	connected_instagram_account	String	該当なし
	contact_address	String	該当なし
	country_page_likes	整数	該当なし
	カバー	Struct	該当なし
	culinary_team	String	該当なし
	current_location	String	該当なし
	delivery_and_pickup_option_info	リスト	該当なし
	説明	String	該当なし
	description_html	String	該当なし
	differently_open_offerings	リスト	該当なし
	directed_by	String	該当なし
	display_subtext	String	該当なし

エンティティ	フィールド	データ型	サポートされている演算子
	displayed_message_response_time	String	該当なし
	E メール	String	該当なし
	エンゲージメント	String	該当なし
	fan_count	整数	該当なし
	featured_video	String	該当なし
	機能	String	該当なし
	followers_count	整数	該当なし
	food_styles	リスト	該当なし
	設立済み	String	該当なし
	general_info	String	該当なし
	general_manager	String	該当なし
	Genre	String	該当なし
	global_brand_page_name	String	該当なし
	global_brand_root_id	String	該当なし
	has_added_app	ブール値	該当なし
	has_transitioned_to_new_page_experience	ブール値	該当なし
	has_whatsapp_business_number	ブール値	該当なし

エンティティ	フィールド	データ型	サポートされている演算子
	has_whatsapp_number	ブール値	該当なし
	ホームタウン	String	該当なし
	時間	Struct	該当なし
	Impressum	String	該当なし
	影響	String	該当なし
	instagram_business_account	String	該当なし
	is_always_open	ブール値	該当なし
	is_chain	ブール値	該当なし
	is_community_page	ブール値	該当なし
	is_eligible_for_branded_content	ブール値	該当なし
	is_messenger_bot_get_started_enabled	ブール値	該当なし
	is_messenger_platform_bot	ブール値	該当なし
	is_owned	ブール値	該当なし
	is_permanently_closed	ブール値	該当なし
	is_published	ブール値	該当なし
	名前	String	該当なし

エンティティ	フィールド	データ型	サポートされている演算子
	タスク	リスト	該当なし
	is_unclaimed	ブール値	該当なし
	is_webhooks_subscribed	ブール値	該当なし
	leadgen_tos_acceptance_time	DateTime	該当なし
	leadgen_tos_accepted	ブール値	該当なし
	leadgen_tos_accepting_user	String	該当なし
	leadgen_tos_accepting_user	Struct	該当なし
	リンク	リンク	該当なし
	場所	Struct	該当なし
	メンバー	String	該当なし
	merchant_review_status	String	該当なし
	messenger_ads_default_icebreakers	リスト	該当なし
	messenger_ads_default_page_welcome_message	Struct	該当なし
	messenger_ads_default_quick_replies	リスト	該当なし

エンティティ	フィールド	データ型	サポートされている演算子
	messenger_ads_quick_replies_type	String	該当なし
	ミッション	String	該当なし
	Mpg	String	該当なし
	name_with_location_descriptor	String	該当なし
	ネットワーク	String	該当なし
	new_like_count	整数	該当なし
	offer_eligible	ブール値	該当なし
	overall_star_rating	浮動小数点数	該当なし
	page_token	String	該当なし
	parent_page	String	該当なし
	Parking	String	該当なし
	payment_options	Struct	該当なし
	personal_info	String	該当なし
	personal_interests	String	該当なし
	pharma_safety_info	String	該当なし
	電話	String	該当なし
	pickup_options	リスト	該当なし
	place_type	String	該当なし
	plot_outline	String	該当なし

エンティティ	フィールド	データ型	サポートされている演算子
	press_contact	String	該当なし
	price_range	String	該当なし
	privacy_info_url	String	該当なし
	produced_by	String	該当なし
	製品	String	該当なし
	promotion_eligible	ブール値	該当なし
	promotion_ineligible_reason	String	該当なし
	public_transit	String	該当なし
	rating_count	整数	該当なし
	record_label	String	該当なし
	release_date	String	該当なし
	restaurant_services	Struct	該当なし
	restaurant_specialties	Struct	該当なし
	スケジュール	String	該当なし
	screenplay_by	String	該当なし
	シーズン	String	該当なし
	single_line_address	String	該当なし
	スターリング	String	該当なし
	start_info	Struct	該当なし

エンティティ	フィールド	データ型	サポートされている演算子
	store_code	String	該当なし
	store_location_descriptor	String	該当なし
	store_number	整数	該当なし
	Studio	String	該当なし
	supports_donate_button_in_live_video	ブール値	該当なし
	talking_about_count	整数	該当なし
	temporary_status	String	該当なし
	unread_message_count	整数	該当なし
	unread_notif_count	整数	該当なし
	unseen_message_count	整数	該当なし
	ユーザーネーム	String	該当なし
	verification_status	String	該当なし
	voip_info	Struct	該当なし
	ウェブサイト	String	該当なし
	were_here_count	整数	該当なし
	whatsapp_number	String	該当なし
written_by	String	該当なし	

エンティティ	フィールド	データ型	サポートされている演算子
フィード	ID	String	該当なし
	アクション	リスト	該当なし
	admin_creator	オブジェクト	該当なし
	アプリケーション	オブジェクト	該当なし
	添付ファイル	オブジェクト	該当なし
	backdated_time	DateTime	該当なし
	call_to_action	オブジェクト	該当なし
	can_reply_privately	ブール値	該当なし
	child_attachments	リスト	該当なし
	座標	Struct	該当なし
	created_time	DateTime	該当なし
	イベント	Struct	該当なし
	expanded_height	整数	該当なし
	expanded_width	整数	該当なし
	feed_targeting	オブジェクト	該当なし
	From	オブジェクト	該当なし
	full_picture	String	該当なし
	[Height] (高さ)	整数	該当なし
	アイコン	String	該当なし
	instagram_eligibility	String	該当なし

エンティティ	フィールド	データ型	サポートされている演算子
	is_eligible_for_promotion	ブール値	該当なし
	is_expired	ブール値	該当なし
	is_hidden	ブール値	該当なし
	is_inline_created	ブール値	該当なし
	is_instagram_eligible	ブール値	該当なし
	is_popular	ブール値	該当なし
	is_published	ブール値	該当なし
	is_spherical	ブール値	該当なし
	メッセージ	String	該当なし
	message_tags	リスト	該当なし
	multi_share_end_card	ブール値	該当なし
	multi_share_optimized	ブール値	該当なし
	parent_id	String	該当なし
	permalink_url	String	該当なし
	Place	String	該当なし
	プライバシー	オブジェクト	該当なし
	promotable_id	String	該当なし
	promotion_status	String	該当なし
	プロパティ	リスト	該当なし

エンティティ	フィールド	データ型	サポートされている演算子
	scheduled_publish_time	浮動小数点数	該当なし
	共有	オブジェクト	該当なし
	status_type	String	該当なし
	ストーリー	String	該当なし
	story_tags	リスト	該当なし
	サブスクライブ済み	ブール値	該当なし
	Target	Struct	該当なし
	ターゲティング	オブジェクト	該当なし
	To	オブジェクト	該当なし
	timeline_visibility	文字列	該当なし
	updated_time	DateTime	該当なし
	Via	Struct	該当なし
	video_buying_eligibility	リスト	該当なし
	幅	整数	該当なし
	Since	DateTime	EQUAL_TO

パーティショニングクエリ

フィルターベースのパーティション分割:

Spark で同時実行を使用する場合は、追加の Spark オプション `PARTITION_FIELD`、`LOWER_BOUND`、`UPPER_BOUND`、および `NUM_PARTITIONS` を指定できま

す。これらのパラメータを使用すると、元のクエリは Spark タスクで同時に実行できるサブクエリの NUM_PARTITIONS の数に分割されます。

- PARTITION_FIELD: クエリのパーティション化に使用するフィールドの名前。
- LOWER_BOUND: 選択したパーティションフィールドの包括的な下限値。

  Datetime フィールドでは、Spark SQL クエリで使用される Spark タイムスタンプ形式を受け入れます。

  有効な値の例は次のとおりです:

  ```
  "2024-09-30T01:01:01.000Z"
  ```

- UPPER_BOUND: 選択したパーティションフィールドの排他的上限値。
- NUM_PARTITIONS: パーティション数。

例:

```
facebookPageInsights_read = glueContext.create_dynamic_frame.from_options(
     connection_type="facebookpageinsights",
     connection_options={
         "connectionName": "connectionName",
         "ENTITY_NAME": "entityName",
         "API_VERSION": "v21",
         "PARTITION_FIELD": "created_Time"
         "LOWER_BOUND": "2024-10-27T07:00:00+0000"
         "UPPER_BOUND": "2024-10-27T07:00:00+0000"
         "NUM_PARTITIONS": "10"
     }
```

Facebook Page Insights 接続オプション

Facebook Page Insights の接続オプションは次のとおりです。

- ENTITY_NAME(文字列) – (必須) 読み取りに使用されます。Facebook Page Insights のオブジェクトの名前。
- API_VERSION(文字列) – (必須) 読み取りに使用されます。使用する Facebook Page Insights Rest API バージョン。

- SELECTED_FIELDS(List<String>) – Default: empty(SELECT *). 読み込みに使用されます。オブジェクトに選択する列。
- FILTER_PREDICATE(文字列) – デフォルト: 空 読み込みに使用されます。Spark SQL 形式である必要があります。
- QUERY(文字列) – デフォルト: 空 読み込みに使用されます。完全な Spark SQL クエリ。
- PARTITION_FIELD(文字列) – 読み取りに使用されます。クエリをパーティション化するために使用するフィールド。
- LOWER_BOUND(文字列) - 読み取りに使用されます。選択したパーティションフィールドの包括的な下限値。
- UPPER_BOUND(文字列) – 読み取りに使用されます。選択したパーティションフィールドの排他的上限値。
- NUM_PARTITIONS(整数) – デフォルト: 1。読み込みに使用されます。読み取り用のパーティションの数。
- INSTANCE_URL(文字列) – (必須) 読み取りに使用されます。有効な Facebook Page Insights インスタンス URL。

Facebook Page Insights コネクタの制限事項と注意事項

Facebook Page Insights コネクタの制限事項または注意事項は次のとおりです。

- ほとんどのメトリクスは 24 時間ごとに 1 回更新されます。
- 過去 2 年間のインサイトデータのみを使用できます。
- `since` および `until` パラメータを使用する場合、一度に表示できるのは 90 日間のインサイトのみです。

Freshdesk に対する接続

Freshdesk は、機能が豊富で使いやすいクラウドベースのカスタマーサポートソフトウェアです。ライブチャット、E メール、電話、ソーシャルメディアなど、複数のサポートチャネルを利用できるため、お客様が希望する通信方法を通じてサポートできます。Freshdesk ユーザーの場合は、Freshdesk アカウントに AWS Glue を接続できます。Freshdesk を ETL ジョブのデータソースとして使用できます。これらのジョブを実行して、Freshdesk から AWS サービス、またはその他のサポートされているアプリケーションにデータを転送します。

トピック

- Freshdesk 向けの AWS Glue サポート
- 接続を作成および使用するための API オペレーションを含むポリシー
- Freshdesk の設定
- Freshdesk 接続の設定
- Freshdesk エンティティからの読み取り
- Freshdesk 接続オプション
- Freshdesk コネクタの制限事項と注意事項

Freshdesk 向けの AWS Glue サポート

AWS Glue は、次のように、Freshdesk をサポートします。

ソースとしてサポートされていますか?

はい – Sync と Async。AWS Glue ETL ジョブを使用して、Freshdesk からデータをクエリできます。

ターゲットとしてサポートされていますか?

いいえ。

サポートされる Freshdesk API バージョン

次の Freshdesk API バージョンがサポートされています。

- v2

接続を作成および使用するための API オペレーションを含むポリシー

次のサンプル ポリシーで、接続の作成と使用に必要な AWS IAM 権限について説明します。新しいロールを作成する場合は、以下を含むポリシーを作成します。

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

```
                "glue:ListConnectionTypes",
                "glue:DescribeConnectionType",
                "glue:RefreshOAuth2Tokens",
                "glue:ListEntities",
                "glue:DescribeEntity"
            ],
            "Resource": "*"
        }
    ]
}
```

上記の方法を使用しない場合は、代わりに次のマネージド IAM ポリシーを使用します:

- AWSGlueServiceRole – さまざまな AWS Glue プロセスを実行するために必要なリソースへのアクセス権をユーザーに代わって付与します。これらのリソースには AWS Glue 、Amazon S3、IAM、CloudWatch Logs、および Amazon EC2 が含まれます。このポリシーで指定されたリソースの命名規則に従った場合、AWS Glue プロセスは必要なアクセス権限を使用できます。このポリシーは、通常、クローラ、ジョブ、開発エンドポイントを定義するときに指定されたロールにアタッチされます。
- AWSGlueConsoleFullAccess — ポリシーがアタッチされているアイデンティティが AWS マネージメントコンソールを使用するときは、AWS Glue リソースへのフルアクセスを許可します。このポリシーで指定されたリソースの命名規則に従った場合、ユーザーは完全なコンソール機能を使用できます。このポリシーは、通常 AWS Glue コンソールのユーザーにアタッチされています。

Freshdesk の設定

AWS Glue を使用して Freshdesk からデータを転送できるようにするには、次の要件を満たす必要があります。

最小要件

以下に、最小要件を示します。

- Freshdesk アカウント。Free、Growth、Pro、Enterprise エディションから選択できます。
- Freshdesk ユーザーの API キー。

Freshdesk 接続の設定

Freshdesk はカスタム認証をサポートしています。

カスタム認証に必要な API キーの生成に関する Freshdesk のパブリックドキュメントについては、「Freshdesk authentication」を参照してください。

Freshdesk 接続を設定する手順は次のとおりです。

- AWS Secrets Manager で、次の詳細を含むシークレットを作成します。
  - カスタマーマネージド接続アプリケーションの場合 - シークレットには、apiKey を使用して接続されたアプリケーションの API キーをキーとして含める必要があります。AWS Glue で接続ごとにシークレットを作成する必要があることに注意してください。
- AWS Glue Studio で、以下の手順に従って [データ接続] の下に接続を作成します。
  - [データソース] を選択する際に、[Freshdesk] を選択します。
  - 接続先の Freshdesk インスタンスの INSTANCE_URL を指定します。
  - 次のアクションを実行でき、AWS Glue がその権限を持つ AWS IAM ロールを選択します。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:DescribeSecret",
                "secretsmanager:GetSecretValue",
                "secretsmanager:PutSecretValue",
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterface",
                "ec2:DeleteNetworkInterface",
            ],
            "Resource": "*"
        }
    ]
}
```

  - AWS Glue でこの接続に使用する secretName を選択して、トークンを配置します。
  - ネットワークを使用する場合は、ネットワークオプションを選択します。
- AWS Glue ジョブに関連付けられている IAM ロールに secretName を読み取るアクセス許可を付与します。
- AWS Glue ジョブ設定で、[追加のネットワーク接続] として connectionName を指定します。

Freshdesk エンティティからの読み取り

前提条件

読み取り元の Freshdesk オブジェクト。オブジェクト名が必要になります。

Sync ソースに対応するエンティティ:

エンティティ	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
エージェント	はい	はい	なし	はい	はい
営業時間	いいえ	あり	なし	はい	はい
Company	はい	はい	なし	はい	はい
問い合わせ	はい	はい	なし	はい	はい
Conversations	いいえ	あり	なし	あり	いいえ
E メール設定	いいえ	あり	なし	あり	いいえ
E メール受信トレイ	はい	あり	あり	はい	いいえ
フォーラムカテゴリ	いいえ	あり	なし	あり	いいえ
フォーラム	いいえ	あり	なし	あり	いいえ
グループ	いいえ	あり	なし	あり	いいえ
製品	いいえ	あり	なし	あり	いいえ
ロール	いいえ	あり	なし	あり	いいえ
満足度評価	はい	はい	なし	あり	いいえ

エンティティ	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
スキル	いいえ	あり	なし	あり	いいえ
解決方法	はい	はい	なし	あり	いいえ
アンケート	いいえ	あり	なし	あり	いいえ
チケット	はい	あり	あり	あり	はい
時間エントリ	はい	はい	なし	あり	いいえ
トピック	いいえ	あり	なし	あり	いいえ
トピックのコメント	いいえ	あり	なし	あり	いいえ

Async ソースに対応するエンティティ:

エンティティ	API バージョン	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
Companies	v2	いいえ	いいえ	いいえ	いいえ	いいえ
問い合わせ	v2	いいえ	いいえ	いいえ	いいえ	いいえ

例:

```
freshdesk_read = glueContext.create_dynamic_frame.from_options(
    connection_type="freshdesk",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "entityName",
        "API_VERSION": "v2"
```

```
    }
```

Freshdesk エンティティとフィールドの詳細

エンティティ	フィールド
エージェント	https://developers.freshdesk.com/api/#list_all_agents
営業時間	https://developers.freshdesk.com/api/#list_all_business_hours
コメント	https://developers.freshdesk.com/api/#comment_attributess
Company	https://developers.freshdesk.com/api/#companies
問い合わせ	https://developers.freshdesk.com/api/#list_all_contacts
Conversations	https://developers.freshdesk.com/api/#list_all_ticket_notes
Email-configs	https://developers.freshdesk.com/api/#list_all_email_configs
Email-inboxes	https://developers.freshdesk.com/api/#list_all_email_mailboxes
Forum-categories	https://developers.freshdesk.com/api/#category_attributes
フォーラム	https://developers.freshdesk.com/api/#forum_attributes
グループ	https://developers.freshdesk.com/api/#list_all_groups

エンティティ	フィールド
製品	https://developers.freshdesk.com/api/#list_all_products
ロール	https://developers.freshdesk.com/api/#list_all_roles
Satisfaction-rating	https://developers.freshdesk.com/api/#view_all_satisfaction_ratingss
スキル	https://developers.freshdesk.com/api/#list_all_skills
解決方法	https://developers.freshdesk.com/api/#solution_content
アンケート	https://developers.freshdesk.com/api/#list_all_survey
チケット	https://developers.freshdesk.com/api/#list_all_tickets
Time-entries	https://developers.freshdesk.com/api/#list_all_time_entries
トピック	https://developers.freshdesk.com/api/#topic_attributes

パーティショニングクエリ

フィルターベースのパーティション分割:

Spark で同時実行を使用する場合は、追加の Spark オプション `PARTITION_FIELD`、`LOWER_BOUND`、`UPPER_BOUND`、および `NUM_PARTITIONS` を指定できます。これらのパラメータを使用すると、元のクエリは Spark タスクで同時に実行できるサブクエリの `NUM_PARTITIONS` の数に分割されます。

- `PARTITION_FIELD`: クエリのパーティション化に使用するフィールドの名前。

- LOWER_BOUND: 選択したパーティションフィールドの包括的な下限値。

  Datetime フィールドでは、Spark SQL クエリで使用される Spark タイムスタンプ形式を受け入れます。

  有効な値の例は次のとおりです:

  ```
  "2024-09-30T01:01:01.000Z"
  ```

- UPPER_BOUND: 選択したパーティションフィールドの排他的上限値。
- NUM_PARTITIONS: パーティション数。

例:

```
freshDesk_read = glueContext.create_dynamic_frame.from_options(
     connection_type="freshdesk",
     connection_options={
         "connectionName": "connectionName",
         "ENTITY_NAME": "entityName",
         "API_VERSION": "v2",
         "PARTITION_FIELD": "Created_Time"
         "LOWER_BOUND": " 2024-10-27T23:16:08Z“
         "UPPER_BOUND": " 2024-10-27T23:16:08Z"
         "NUM_PARTITIONS": "10"
     }
```

Freshdesk 接続オプション

Freshdesk の接続オプションは次のとおりです。

- ENTITY_NAME(文字列) – (必須) 読み取りに使用されます。Freshdesk のオブジェクトの名前。
- API_VERSION(文字列) – (必須) 読み取りに使用されます。使用する Freshdesk Rest API バージョン。
- SELECTED_FIELDS(List<String>) – Default: empty(SELECT *). 読み込みに使用されます。オブジェクトに選択する列。
- FILTER_PREDICATE(文字列) – デフォルト: 空 読み込みに使用されます。Spark SQL 形式である必要があります。
- QUERY(文字列) - デフォルト: 空 読み込みに使用されます。完全な Spark SQL クエリ。

- PARTITION_FIELD(文字列) – 読み取りに使用されます。クエリをパーティション化するために使用するフィールド。
- LOWER_BOUND(文字列) - 読み取りに使用されます。選択したパーティションフィールドの包括的な下限値。
- UPPER_BOUND (文字列) - 読み取りに使用されます。選択したパーティションフィールドの排他的上限値。
- NUM_PARTITIONS(整数) – デフォルト: 1。読み込みに使用されます。読み取り用のパーティションの数。
- INSTANCE_URL (文字列) – (必須) 読み取りに使用されます。有効な Freshdesk インスタンス URL。
- TRANSFER_MODE(文字列) - SYNC または ASYNC のような処理のタイプがデフォルトで SYNC に設定されているかどうかを示すために使用されます。(オプション)

Freshdesk コネクタの制限事項と注意事項

Freshdesk コネクタの制限事項または注意事項は次のとおりです。

- Company、Contacts、および Tickets のフィルタリングされたエンティティには、ページ分割の制限があります。1 ページあたり 30 レコードのみが返され、ページ値は最大 10 個まで設定できます (最大 300 レコードを取得)。
- Tickets エンティティは 30 日以上経過したレコードを取得しません。
- Company、Contacts、および Tickets エンティティは、フィルタリングで「Date」データ型をサポートします。これら 3 つのエンティティの「毎日」以降のトリガー頻度を選択する必要があります。「分」または「時間単位」を選択すると、データが重複する可能性があります。また、フィルタリングのためにこれらのフィールドを選択するときは、選択したタイムスタンプの日付部分のみを考慮するため、日付値のみを選択する必要があります。
- 1 分あたりの API コール数は、プランに基づきます。この制限は、通話に使用されるエージェントや IP アドレスの数などの要因に関係なく、アカウント全体に適用されます。すべてのトライアルユーザーについて、デフォルトの API 制限は 50 コール/分です。詳細については、「Freshdesk」を参照してください
- どのエンティティでも、一度に処理される Export/Async ジョブは 1 つだけです。新しいジョブは、既存のジョブが正常に完了したか失敗した場合にのみ処理されます。詳細については、「Freshdesk」を参照してください
- 以下のフィールドは Sync API コールでサポートされていますが、Async API リクエストボディで渡すことはサポート/許可されていません。

- id
- created_at
- updated_at
- updated_since
- ACTIVE
- company_id
- other_companies
- avatar
- view_all_tickets
- 削除済み
- other_emails
- state
- タグ
- tags

Freshsales に対する接続

Freshsales は直感的に使用できる CRM のため、販売担当者は営業の手間を省くことができます。電話と E メール、タスク、予約、メモが組み込まれているため、営業担当者は見込み客をフォローアップするためにタブを切り替える必要はありません。パイプラインビューを使用して取引をより適切に管理し、より多くの取引を成約に導くことができます。Freshsales ユーザーの場合は、AWS Glue を Freshsales アカウントに接続できます。その後、Freshsales を ETL ジョブのデータソースとして使用できます。これらのジョブを実行して、Freshsales から AWS サービス、またはその他のサポートされているアプリケーションにデータを転送します。

トピック

- Freshsales の AWS Glue サポート
- 接続を作成および使用するための API オペレーションを含むポリシー
- Freshsales の設定
- Freshsales 接続の設定
- Freshsales エンティティからの読み取り
- Freshsales 接続オプション

- Freshsales の制限事項

Freshsales の AWS Glue サポート

AWS Glue は、次のように Freshsales をサポートします。

ソースとしてサポートされていますか?

はい。AWS Glue ETL ジョブを使用して、Freshsales からデータをクエリできます。

ターゲットとしてサポートされていますか?

いいえ。

サポートされる Freshsales API バージョン

次の Freshsales API バージョンがサポートされています。

- v1.0

接続を作成および使用するための API オペレーションを含むポリシー

次のサンプル ポリシーで、接続の作成と使用に必要な AWS IAM 権限について説明します。新しいロールを作成する場合は、以下を含むポリシーを作成します。

```
{
 "Version": "2012-10-17",
     "Statement": [
      {
             "Effect": "Allow",
             "Action": [
                 "glue:ListConnectionTypes",
                 "glue:DescribeConnectionType",
                 "glue:RefreshOAuth2Tokens",
                 "glue:ListEntities",
                 "glue:DescribeEntity"
             ],
             "Resource": "*"
         }
     ]
 }
```

上記の方法を使用しない場合は、代わりに次のマネージド IAM ポリシーを使用します:

- AWSGlueServiceRole – さまざまな AWS Glue プロセスを実行するために必要なリソースへのアクセス権をユーザーに代わって付与します。これらのリソースには AWS Glue 、Amazon S3、IAM、CloudWatch Logs、および Amazon EC2 が含まれます。このポリシーで指定されたリソースの命名規則に従った場合、AWS Glue プロセスは必要なアクセス権限を使用できます。このポリシーは、通常、クローラ、ジョブ、開発エンドポイントを定義するときに指定されたロールにアタッチされます。
- AWSGlueConsoleFullAccess — ポリシーがアタッチされているアイデンティティが AWS マネージメントコンソールを使用するときは、AWS Glue リソースへのフルアクセスを許可します。このポリシーで指定されたリソースの命名規則に従った場合、ユーザーは完全なコンソール機能を使用できます。このポリシーは、通常 AWS Glue コンソールのユーザーにアタッチされています。

Freshsales の設定

AWS Glue を使用して Freshsales からデータを転送できるようにするには、次の要件を満たす必要があります。

最小要件

以下に、最小要件を示します。

- Freshsales アカウントがある。
- ユーザー API キーがある。

これらの要件を満たすと、Freshsales アカウントに AWS Glue を接続する準備が整います。一般的な接続では、Freshsales で他に何もする必要はありません。

Freshsales 接続の設定

Freshsales はカスタム認証をサポートしています。

カスタム認証に必要な API キーの生成に関する Freshsales のパブリックドキュメントについては、「認証」を参照してください。

Freshsales 接続を設定するには:

1. AWS Secrets Manager で、次の詳細を含むシークレットを作成します。

a. カスタマーマネージド接続アプリケーションの場合、シークレットには、*apiSecretKey* を使用して接続されたアプリケーションの API キーをキーとして含める必要があります。また、シークレットには、*apiKey* をキー、*token* を値とする別のキーと値のペアを含める必要があります。

b. 注: AWS Glue で接続にシークレットを作成する必要があります。

1. AWS Glue Glue Studio で、以下の手順に従って [データ接続] の下に接続を作成します。

   a. [データソース] を選択する際に、[Freshsales] を選択します。

   b. 接続先の Freshsales アカウントの `INSTANCE_URL` を指定します。

   c. 次のアクションを実行でき、AWS Glue がその権限を持つ AWS IAM ロールを選択します。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:DescribeSecret",
                "secretsmanager:GetSecretValue",
                "secretsmanager:PutSecretValue",
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterface",
                "ec2:DeleteNetworkInterface",
            ],
            "Resource": "*"
        }
    ]
}
```

   d. AWS Glue でこの接続に使用する `secretName` を選択して、トークンを配置します。

   e. ネットワークを使用する場合は、ネットワークオプションを選択します。

2. AWS Glue ジョブに関連付けられている IAM ロールに `secretName` を読み取るアクセス許可を付与します。

Freshsales エンティティからの読み取り

前提条件

読み取り元の Freshsales オブジェクト。オブジェクト名が必要になります。

ソースに対応するエンティティ:

エンティティ	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
アカウント	はい	はい	はい	はい	はい
連絡先	はい	はい	はい	はい	はい

例:

```
freshSales_read = glueContext.create_dynamic_frame.from_options(
     connection_type="freshsales",
     connection_options={
         "connectionName": "connectionName",
         "ENTITY_NAME": "entityName",
         "API_VERSION": "v1.0"
     }
```

Freshsales エンティティとフィールドの詳細 :

Freshsales では、サポートされているエンティティのメタデータを動的に取得するためのエンドポイントが提供されます。したがって、オペレータのサポートはデータ型レベルでキャプチャされます。

エンティティ	データ型	サポートされている演算子
Freshsales エンティティ (すべて)	整数	!=,=,<,<=,>,>=,BETWEEN
	String	Like, =, !=
	BigInteger	!=,=,<,<=,>,>=,BETWEEN
	ブール値	=
	倍精度	!=,=,<,<=,>,>=,BETWEEN

エンティティ	データ型	サポートされている演算子
	BigDecimal	!=,=,<,<=,>,>=,BETWEEN
	日付	!=,=,<,<=,>,>=,BETWEEN
	DateTime	!=,=,<,<=,>,>=,BETWEEN
	Struct	該当なし
	リスト	該当なし

パーティショニングクエリ

フィルターベースのパーティション分割:

Spark で同時実行を使用する場合は、追加の Spark オプション PARTITION_FIELD、LOWER_BOUND、UPPER_BOUND、および NUM_PARTITIONS を指定できます。これらのパラメータを使用すると、元のクエリは Spark タスクで同時に実行できるサブクエリの NUM_PARTITIONS の数に分割されます。

- PARTITION_FIELD: クエリのパーティション化に使用するフィールドの名前。
- LOWER_BOUND: 選択したパーティションフィールドの包括的な下限値。

  Datetime フィールドでは、ISO 形式の値を受け入れます。

  有効な値の例は次のとおりです:

  ```
  "2024-09-30T01:01:01.000Z"
  ```
- UPPER_BOUND: 選択したパーティションフィールドの排他的上限値。
- NUM_PARTITIONS: パーティション数。

例:

```
freshSales_read = glueContext.create_dynamic_frame.from_options(
     connection_type="freshsales",
     connection_options={
         "connectionName": "connectionName",
```

```
        "ENTITY_NAME": "entityName",
        "API_VERSION": "v1",
        "PARTITION_FIELD": "Created_Time"
        "LOWER_BOUND": " 2024-10-15T21:16:25Z"
        "UPPER_BOUND": " 2024-10-20T21:25:50Z"
        "NUM_PARTITIONS": "10"
    }
```

Freshsales 接続オプション

Freshsales の接続オプションは次のとおりです。

- ENTITY_NAME(文字列) – (必須) 読み取りに使用されます。Freshsales のオブジェクトの名前。
- API_VERSION(文字列) – (必須) 読み取りに使用されます。使用する Freshsales Rest API バージョン。
- SELECTED_FIELDS(List<String>) – Default: empty(SELECT *). 読み込みに使用されます。オブジェクトに選択する列。
- FILTER_PREDICATE(文字列) – デフォルト: 空 読み込みに使用されます。Spark SQL 形式である必要があります。
- QUERY(文字列) – デフォルト: 空 読み込みに使用されます。完全な Spark SQL クエリ。
- PARTITION_FIELD(文字列) – 読み取りに使用されます。クエリをパーティション化するために使用するフィールド。
- LOWER_BOUND(文字列) - 読み取りに使用されます。選択したパーティションフィールドの包括的な下限値。
- UPPER_BOUND(文字列) – 読み取りに使用されます。選択したパーティションフィールドの排他的上限値。
- NUM_PARTITIONS(整数) – デフォルト: 1。読み込みに使用されます。読み取り用のパーティションの数。
- INSTANCE_URL(文字列) – 読み取りに使用されます。有効な Freshsales インスタンス URL。

Freshsales の制限事項

Freshsales の制限事項または注意事項は次のとおりです。

- Freshsales では、API レート制限はアカウントあたり 1 時間ごとに 1000 件の API リクエストです (「エラー」を参照。ただし、この制限はエンタープライズサブスクリプションプランで拡張できます (「プランの比較」を参照)。

Google 広告に対する接続

Google 広告 API は、Google 広告へのプログラムによるインターフェイスであり、大規模または複雑な Google 広告アカウントやキャンペーンの管理に使用されます。Google 広告ユーザーの場合は、 AWS Glue を Google 広告アカウントに接続できます。その後、Google 広告を ETL ジョブのデータソースとして使用できます。これらのジョブを実行して、Google 広告と AWS サービス、またはその他のサポートされているアプリケーション間でデータを転送します。

トピック

- Google 広告用の AWS Glue サポート
- 接続を作成および使用するためのAPIオペレーションを含むポリシー
- Google Ads の設定
- Google 広告接続の設定
- Google 広告エンティティからの読み取り
- Google 広告の接続オプション
- Google Ads アカウントの作成
- 制限

Google 広告用の AWS Glue サポート

AWS Glue は次のように Google 広告をサポートします。

ソースとしてサポートされていますか?

はい。AWS Glue ETL ジョブを使用し、Google 広告のデータをクエリできます。

ターゲットとしてサポートされていますか?

いいえ。

サポートされている Google 広告 の API バージョン

v18

接続を作成および使用するためのAPIオペレーションを含むポリシー

次のサンプルポリシーでは、接続の作成と使用に必要な AWS アクセス許可について説明します。新しいロールを作成する場合は、以下を含むポリシーを作成します。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:ListConnectionTypes",
                "glue:DescribeConnectionType",
                "glue:RefreshOAuth2Tokens"
                "glue:ListEntities",
                "glue:DescribeEntity"
            ],
            "Resource": "*"
        }
    ]
}
```

以下の管理IAMポリシーを使用してアクセスを許可することもできます。

- AWSGlueServiceRole – さまざまな AWS Glue プロセスがユーザーに代わって実行するために必要なリソースへのアクセスを許可します。これらのリソースには AWS Glue、、Amazon S3、IAM、 CloudWatch Logs、Amazon が含まれますEC2。このポリシーで指定されたリソースの命名規則に従うと、 AWS Glue プロセスには必要なアクセス許可があります。このポリシーは、通常、クローラ、ジョブ、開発エンドポイントを定義するときに指定されたロールにアタッチされます。
- AWSGlueConsoleFullAccess — ポリシーがアタッチされている ID が マネジメントコンソールを使用する場合 AWS 、 AWS Glue リソースへのフルアクセスを許可します。このポリシーで指定されたリソースの命名規則に従った場合、ユーザーは完全なコンソール機能を使用できます。このポリシーは通常、 AWS Glue コンソールのユーザーにアタッチされます。

Google Ads の設定

AWS Glue を使用して Google Ads から転送する前に、次の要件を満たす必要があります。

最小要件

- E メールとパスワードを持つ Google Ads アカウントがあります。アカウントの作成の詳細については、「Google Ads アカウントの作成」を参照してください。

- Google Ads アカウントへのアクセスが有効になっていますAPI。Google Ads のすべての使用API は、追加料金なしで利用できます。
- Google Ads アカウントでは、接続されたアプリケーションをインストールできます。この機能にアクセスできない場合は、Google Ads 管理者にお問い合わせください。

これらの要件を満たすと、Google Ads アカウント AWS Glue に接続できます。

Google 広告接続の設定

Google 広告は OAuth2 の AUTHORIZATION_CODE グラントタイプをサポートしています。

このグラントタイプは、ユーザーを認証するためにサードパーティーの認証サーバーにユーザーをリダイレクトすることから、「three-legged」の OAuth と見なされます。AWS Glue コンソール経由で接続を作成するときに使用されます。AWS Glue コンソールでは、ユーザーが Google 広告にリダイレクトされます。ここではユーザーがログインし、Google 広告インスタンスにアクセスするためのリクエストされた権限を AWS Glue に許可する必要があります。

ユーザーは、Google 広告で独自の接続アプリを作成し、AWS Glue コンソールを介して接続を作成するときに独自のクライアント ID とクライアントシークレットを指定することを選択できます。このシナリオでは、引き続き Google 広告にリダイレクトされてログインし、リソースへアクセスするために AWS Glue を承認します。

このグラントタイプは、更新トークンとアクセストークンになります。アクセストークンの有効期間は短く、更新トークンを使用してユーザーとやり取りすることなく自動的に更新される場合があります。

詳細については、「OAuth 認証コードフローでの接続されたアプリケーションの作成に関する Google 広告の公開ドキュメント」を参照してください。

Google 広告の接続を設定するには:

1. AWS Secrets Manager で、次の詳細を含むシークレットを作成します。AWS Glue で接続ごとにシークレットを作成する必要があります。
   a. AuthorizationCode グラントタイプの場合:
      - カスタマーマネージド接続アプリケーションの場合 – シークレットには、USER_MANAGED_CLIENT_APPLICATION_CLIENT_SECRET を使用して接続されたアプリケーションのコンシューマーシークレットをキーとして含める必要があります。
2. AWS Glue Glue Studio で、以下の手順に従って [データ接続] の下に接続を作成します。

a. [接続タイプ] を選択する際に、[Google 広告] を選択します。

b. 接続先の Google 広告の `developer token` を指定します。

c. マネージャーとしてログインする場合は、Google 広告の `MANAGER ID` を指定します。

d. AWS Glue が次のアクションを担い、その権限を持つことができる IAM ロールを選択します。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:DescribeSecret",
                "secretsmanager:GetSecretValue",
                "secretsmanager:PutSecretValue",
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterface",
                "ec2:DeleteNetworkInterface",
            ],
            "Resource": "*"
        }
    ]
}
```

e. AWS Glue でこの接続に使用する `secretName` を選択して、トークンを配置します。

f. ネットワークを使用する場合は、ネットワークオプションを選択します。

3. AWS Glue ジョブに関連付けられている IAM ロールに `secretName` を読み取るアクセス許可を付与します。

Google 広告エンティティからの読み取り

前提条件

- 読み取り元の Google 広告オブジェクト。使用可能なエンティティを確認するには、以下のサポートされているエンティティの表を参照してください。

サポートされているエンティティ

エンティティ	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
Ad Group Ad	はい	可能	はい	なし	あり
Ad Group	可能	可能	はい	なし	あり
Campaign Budget	はい	可能	可能	可能	あり
Account Budget	あり	なし	あり	はい	いいえ
キャンペーン	可能	可能	可能	可能	あり
アカウント	あり	なし	あり	いいえ	いいえ

例

```
googleAds_read = glueContext.create_dynamic_frame.from_options(
    connection_type="googleads",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "campaign-3467***",
        "API_VERSION": "v16"
    }
```

Google 広告エンティティとフィールドの詳細

エンティティ	フィールド	データタイプ	サポートされる演算子
アカウント	resourceName	文字列	!=, =
アカウント	callReportingEnabled	ブール値	!=, =
アカウント	callConversionReportingEnabled	ブール値	!=, =

エンティティ	フィールド	データタイプ	サポートされる演算子
アカウント	callConversionAction	文字列	!=, =
アカウント	conversionTrackingId	BigInteger	BETWEEN, =, !=, <, >, <=, >=
アカウント	crossAccountConversionTrackingId	BigInteger	BETWEEN, =, !=, <, >, <=, >=
アカウント	payPerConversionEligibilityFailureReasons	リスト	
アカウント	id	BigInteger	BETWEEN, =, !=, <, >, <=, >=
アカウント	currencyCode	文字列	!=, =, LIKE
アカウント	timeZone	文字列	!=, =, LIKE
アカウント	autoTaggingEnabled	ブール値	!=, =
アカウント	hasPartnersBadge	ブール値	!=, =
アカウント	manager	ブール値	!=, =
アカウント	testAccount	ブール値	!=, =
アカウント	date	日付	BETWEEN, =, <, >, <=, >=
アカウント	costMicros	BigInteger	BETWEEN, =, !=, <, >, <=, >=
アカウント	acceptedCustomerDataTerms	ブール値	

エンティティ	フィールド	データタイプ	サポートされる演算子
アカウント	conversionTrackingStatus	文字列	!=, =, LIKE
アカウント	enhancedConversionsForLeadsEnabled	ブール値	
アカウント	googleAdsConversionCustomer	文字列	
アカウント	ステータス	文字列	!=, =
アカウント	allConversionsByConversionDate	倍精度	!=, =, <, >
アカウント	allConversionsValueByConversionDate	倍精度	!=, =, <, >
アカウント	conversionsByConversionDate	倍精度	!=, =, <, >
アカウント	conversionsValueByConversionDate	倍精度	!=, =, <, >
アカウント	valuePerAllConversionsByConversionDate	倍精度	!=, =, <, >
アカウント	videoViews	BigInteger	BETWEEN, =, !=, <, >, <=, >=
アカウント	clicks	BigInteger	BETWEEN, =, !=, <, >, <=, >=
アカウント	invalidClicks	BigInteger	BETWEEN, =, !=, <, >, <=, >=

エンティティ	フィールド	データタイプ	サポートされる演算子
アカウント	costPerAllConversions	倍精度	!=, =, <, >
アカウント	costPerConversion	倍精度	!=, =, <, >
アカウント	conversions	倍精度	!=, =, <, >
アカウント	absoluteTopImpressionPercentage	倍精度	!=, =, <, >
アカウント	IMPRESSION	BigInteger	BETWEEN, =, !=, <, >, <=, >=
アカウント	topImpressionPercentage	倍精度	!=, =, <, >
アカウント	averageCpc	倍精度	!=, =, <, >
アカウント	activeViewMeasurableCostMicros	BigInteger	BETWEEN, =, !=, <, >, <=, >=
アカウント	averageCost	倍精度	!=, =, <, >
アカウント	ctr	倍精度	!=, =, <, >
アカウント	activeViewCtr	倍精度	!=, =, <, >
アカウント	searchImpressionShare	倍精度	!=, =, <, >
アカウント	conversionAction	文字列	!=, =
アカウント	conversionActionCategory	文字列	!=, =
アカウント	conversionActionName	文字列	!=, =, LIKE

エンティティ	フィールド	データタイプ	サポートされる演算子
Account Budget	resourceName	文字列	!=, =
Account Budget	ステータス	文字列	!=, =
Account Budget	proposedEndTimeType	文字列	!=, =
Account Budget	approvedEndTimeType	文字列	!=, =
Account Budget	id	BigInteger	BETWEEN, =, !=, <, >, <=, >=
Account Budget	billingSetup	文字列	!=, =
Account Budget	名前	文字列	!=, =, LIKE
Account Budget	approvedStartDateTime	DateTime	BETWEEN, =, <, >, <=, >=
Account Budget	proposedSpendingLimitMicros	BigInteger	BETWEEN, =, !=, <, >, <=, >=
Account Budget	approvedSpendingLimitMicros	BigInteger	BETWEEN, =, !=, <, >, <=, >=
Account Budget	adjustedSpendingLimitMicros	BigInteger	BETWEEN, =, !=, <, >, <=, >=
Account Budget	amountServedMicros	BigInteger	BETWEEN, =, !=, <, >, <=, >=
Ad Group	resourceName	文字列	!=, =, LIKE
Ad Group	ステータス	文字列	!=, =, LIKE
Ad Group	type	文字列	!=, =, LIKE

エンティティ	フィールド	データタイプ	サポートされる演算子
Ad Group	id	BigInteger	BETWEEN, =, !=, <, >, <=, >=
Ad Group	名前	文字列	!=, =, LIKE
Ad Group	キャンペーン	文字列	!=, =
Ad Group	cpcBidMicros	BigInteger	BETWEEN, =, !=, <, >, <=, >=
Ad Group	targetCpaMicros	BigInteger	BETWEEN, =, !=, <, >, <=, >=
Ad Group	cpmBidMicros	BigInteger	BETWEEN, =, !=, <, >, <=, >=
Ad Group	cpvBidMicros	BigInteger	BETWEEN, =, !=, <, >, <=, >=
Ad Group	targetCpmMicros	BigInteger	BETWEEN, =, !=, <, >, <=, >=
Ad Group	effectiveTargetCpaMicros	BigInteger	BETWEEN, =, !=, <, >, <=, >=
Ad Group	date	日付	BETWEEN, =, <, >, <=, >=
Ad Group	costMicros	BigInteger	BETWEEN, =, !=, <, >, <=, >=
Ad Group	useAudienceGrouped	ブール値	!=, =
Ad Group	effectiveCpcBidMicros	BigInteger	BETWEEN, =, !=, <, >, <=, >=

エンティティ	フィールド	データタイプ	サポートされる演算子
Ad Group	allConversionsByConversionDate	倍精度	!=, =, <, >
Ad Group	allConversionsValueByConversionDate	倍精度	!=, =, <, >
Ad Group	conversionsByConversionDate	倍精度	!=, =, <, >
Ad Group	conversionsValueByConversionDate	倍精度	!=, =, <, >
Ad Group	valuePerAllConversionsByConversionDate	倍精度	!=, =, <, >
Ad Group	valuePerConversionsByConversionDate	倍精度	!=, =, <, >
Ad Group	averageCost	倍精度	!=, =, <, >
Ad Group	costPerAllConversions	倍精度	!=, =, <, >
Ad Group	costPerConversion	倍精度	!=, =, <, >
Ad Group	averagePageViews	倍精度	!=, =, <, >
Ad Group	videoViews	BigInteger	BETWEEN, =, !=, <, >, <=, >=
Ad Group	clicks	BigInteger	BETWEEN, =, !=, <, >, <=, >=
Ad Group	allConversions	倍精度	!=, =, <, >
Ad Group	averageCpc	倍精度	!=, =, <, >

エンティティ	フィールド	データタイプ	サポートされる演算子
Ad Group	absoluteTopImpressionPercentage	倍精度	!=, =, <, >
Ad Group	IMPRESSION	BigInteger	BETWEEN, =, !=, <, >, <=, >=
Ad Group	topImpressionPercentage	倍精度	!=, =, <, >
Ad Group	activeViewCtr	倍精度	!=, =, <, >
Ad Group	ctr	倍精度	!=, =, <, >
Ad Group	searchTopImpressionShare	倍精度	!=, =, <, >
Ad Group	searchImpressionShare	倍精度	!=, =, <, >
Ad Group	searchAbsoluteTopImpressionShare	倍精度	!=, =, <, >
Ad Group	relativeCtr	倍精度	!=, =, <, >
Ad Group	conversionAction	文字列	!=, =
Ad Group	conversionActionCategory	文字列	!=, =
Ad Group	conversionActionName	文字列	!=, =, LIKE
Ad Group	updateMask	文字列	
Ad Group	作成	Struct	
Ad Group	更新	Struct	

エンティティ	フィールド	データタイプ	サポートされる演算子
Ad Group	primaryStatus	文字列	!=, =
Ad Group	primaryStatusReasons	リスト	
Ad Group Ad	resourceName	文字列	!=, =
Ad Group Ad	id	BigInteger	BETWEEN, =, !=, <, >, <=, >=
Ad Group Ad	ステータス	文字列	!=, =
Ad Group Ad	ラベル	リスト	
Ad Group Ad	adGroup	文字列	!=, =
Ad Group Ad	costMicros	BigInteger	BETWEEN, =, !=, <, >, <=, >=
Ad Group Ad	approvalStatus	文字列	!=, =
Ad Group Ad	reviewStatus	文字列	!=, =
Ad Group Ad	adStrength	文字列	!=, =
Ad Group Ad	type	文字列	!=, =
Ad Group Ad	businessName	文字列	!=, =, LIKE
Ad Group Ad	date	日付	BETWEEN, =, <, >, <=, >=
Ad Group Ad	allConversionsByConversionDate	倍精度	!=, =, <, >
Ad Group Ad	allConversionsValueByConversionDate	倍精度	!=, =, <, >

エンティティ	フィールド	データタイプ	サポートされる演算子
Ad Group Ad	conversionsByConversionDate	倍精度	!=, =, <, >
Ad Group Ad	conversionsValueByConversionDate	倍精度	!=, =, <, >
Ad Group Ad	valuePerAllConversionsByConversionDate	倍精度	!=, =, <, >
Ad Group Ad	valuePerConversionsByConversionDate	倍精度	!=, =, <, >
Ad Group Ad	activeViewMeasurableCostMicros	BigInteger	BETWEEN, =, !=, <, >, <=, >=
Ad Group Ad	averageCost	倍精度	!=, =, <, >
Ad Group Ad	costPerAllConversions	倍精度	!=, =, <, >
Ad Group Ad	costPerConversion	倍精度	!=, =, <, >
Ad Group Ad	clicks	BigInteger	BETWEEN, =, !=, <, >, <=, >=
Ad Group Ad	averagePageViews	倍精度	!=, =, <, >
Ad Group Ad	videoViews	BigInteger	BETWEEN, =, !=, <, >, <=, >=
Ad Group Ad	allConversions	倍精度	!=, =, <, >
Ad Group Ad	averageCpc	倍精度	!=, =, <, >
Ad Group Ad	topImpressionPercentage	倍精度	!=, =, <, >

エンティティ	フィールド	データタイプ	サポートされる演算子
Ad Group Ad	IMPRESSION	BigInteger	BETWEEN, =, !=, <, >, <=, >=
Ad Group Ad	absoluteTopImpressionPercentage	倍精度	!=, =, <, >
Ad Group Ad	activeViewCtr	倍精度	!=, =, <, >
Ad Group Ad	ctr	倍精度	!=, =, <, >
Ad Group Ad	conversionAction	文字列	!=, =
Ad Group Ad	conversionActionCategory	文字列	!=, =
Ad Group Ad	conversionActionName	文字列	!=, =, LIKE
Ad Group Ad	updateMask	文字列	
Ad Group Ad	作成	Struct	
Ad Group Ad	更新	Struct	
Ad Group Ad	policyValidationParameter	Struct	
Ad Group Ad	primaryStatus	文字列	!=, =
Ad Group Ad	primaryStatusReasons	リスト	
キャンペーン	resourceName	文字列	!=, =
キャンペーン	ステータス	文字列	!=, =
キャンペーン	baseCampaign	文字列	!=, =

エンティティ	フィールド	データタイプ	サポートされる演算子
キャンペーン	名前	文字列	!=, =, LIKE
キャンペーン	id	BigInteger	BETWEEN, =, !=, <, >, <=, >=
キャンペーン	campaignBudget	文字列	!=, =, LIKE
キャンペーン	startDate	日付	BETWEEN, =, <, >, <=, >=
キャンペーン	endDate	日付	BETWEEN, =, <, >, <=, >=
キャンペーン	adServingOptimizationStatus	文字列	!=, =
キャンペーン	advertisingChannelType	文字列	!=, =
キャンペーン	advertisingChannelSubType	文字列	!=, =
キャンペーン	experimentType	文字列	!=, =
キャンペーン	servingStatus	文字列	!=, =
キャンペーン	biddingStrategyType	文字列	!=, =
キャンペーン	domainName	文字列	!=, =, LIKE
キャンペーン	languageCode	文字列	!=, =, LIKE
キャンペーン	useSuppliedUrlsOnly	ブール値	!=, =
キャンペーン	positiveGeoTargetType	文字列	!=, =

エンティティ	フィールド	データタイプ	サポートされる演算子
キャンペーン	negativeGeoTargetType	文字列	!=, =
キャンペーン	paymentMode	文字列	!=, =
キャンペーン	optimizationGoalTypes	リスト	
キャンペーン	date	日付	BETWEEN, =, <, >, <=, >=
キャンペーン	averageCost	倍精度	
キャンペーン	clicks	BigInteger	BETWEEN, =, !=, <, >, <=, >=
キャンペーン	costMicros	BigInteger	BETWEEN, =, !=, <, >, <=, >=
キャンペーン	IMPRESSION	BigInteger	BETWEEN, =, !=, <, >, <=, >=
キャンペーン	useAudienceGrouped	ブール値	!=, =
キャンペーン	activeViewMeasurableCostMicros	BigInteger	BETWEEN, =, !=, <, >, <=, >=
キャンペーン	costPerAllConversions	倍精度	!=, =, <, >
キャンペーン	costPerConversion	倍精度	!=, =, <, >
キャンペーン	invalidClicks	BigInteger	BETWEEN, =, !=, <, >, <=, >=
キャンペーン	publisherPurchasedClicks	BigInteger	BETWEEN, =, !=, <, >, <=, >=

エンティティ	フィールド	データタイプ	サポートされる演算子
キャンペーン	averagePageViews	倍精度	!=, =, <, >
キャンペーン	videoViews	BigInteger	BETWEEN, =, !=, <, >, <=, >=
キャンペーン	allConversionsByConversionDate	倍精度	!=, =, <, >
キャンペーン	allConversionsValueByConversionDate	倍精度	!=, =, <, >
キャンペーン	conversionsByConversionDate	倍精度	!=, =, <, >
キャンペーン	conversionsValueByConversionDate	倍精度	!=, =, <, >
キャンペーン	valuePerAllConversionsByConversionDate	倍精度	!=, =, <, >
キャンペーン	valuePerConversionsByConversionDate	倍精度	!=, =, <, >
キャンペーン	allConversions	倍精度	!=, =, <, >
キャンペーン	absoluteTopImpressionPercentage	倍精度	!=, =, <, >
キャンペーン	searchAbsoluteTopImpressionShare	倍精度	!=, =, <, >
キャンペーン	averageCpc	倍精度	!=, =, <, >
キャンペーン	searchImpressionShare	倍精度	!=, =, <, >

エンティティ	フィールド	データタイプ	サポートされる演算子
キャンペーン	searchTopImpressionShare	倍精度	!=, =, <, >
キャンペーン	activeViewCtr	倍精度	!=, =, <, >
キャンペーン	ctr	倍精度	!=, =, <, >
キャンペーン	relativeCtr	倍精度	!=, =, <, >
キャンペーン	updateMask	文字列	
キャンペーン	作成	Struct	
キャンペーン	更新	Struct	
Campaign Budget	resourceName	文字列	!=, =
Campaign Budget	id	BigInteger	BETWEEN, =, !=, <, >, <=, >=
Campaign Budget	ステータス	文字列	!=, =
Campaign Budget	deliveryMethod	文字列	!=, =
Campaign Budget	period	文字列	!=, =
Campaign Budget	type	文字列	!=, =
Campaign Budget	名前	文字列	!=, =, LIKE
Campaign Budget	amountMicros	BigInteger	BETWEEN, =, !=, <, >, <=, >=
Campaign Budget	explicitlyShared	ブール値	!=, =
Campaign Budget	referenceCount	BigInteger	BETWEEN, =, !=, <, >, <=, >=

エンティティ	フィールド	データタイプ	サポートされる演算子
Campaign Budget	hasRecommendedBudget	ブール値	!=, =
Campaign Budget	date	日付	BETWEEN, =, <, >, <=, >=
Campaign Budget	costMicros	BigInteger	BETWEEN, =, !=, <, >, <=, >=
Campaign Budget	startDate	日付	BETWEEN, =, <, >, <=, >=
Campaign Budget	endDate	日付	BETWEEN, =, <, >, <=, >=
Campaign Budget	maximizeConversionValueTargetRoas	倍精度	!=, =, <, >
Campaign Budget	maximizeConversionsTargetCpaMicros	BigInteger	BETWEEN, =, !=, <, >, <=, >=
Campaign Budget	selectiveOptimizationConversionActions	文字列	
Campaign Budget	averageCost	倍精度	!=, =, <, >
Campaign Budget	costPerAllConversions	倍精度	!=, =, <, >
Campaign Budget	costPerConversion	倍精度	!=, =, <, >
Campaign Budget	videoViews	BigInteger	BETWEEN, =, !=, <, >, <=, >=
Campaign Budget	clicks	BigInteger	BETWEEN, =, !=, <, >, <=, >=

エンティティ	フィールド	データタイプ	サポートされる演算子
Campaign Budget	allConversions	倍精度	!=, =, <, >
Campaign Budget	valuePerAllConversions	倍精度	!=, =, <, >
Campaign Budget	averageCpc	倍精度	!=, =, <, >
Campaign Budget	IMPRESSION	BigInteger	BETWEEN, =, !=, <, >, <=, >=
Campaign Budget	ctr	倍精度	!=, =, <, >
Campaign Budget	updateMask	文字列	
Campaign Budget	作成	Struct	
Campaign Budget	更新	Struct	

パーティショニングクエリ

Spark で同時実行を使用する場合は、追加の Spark オプション `PARTITION_FIELD`、`LOWER_BOUND`、`UPPER_BOUND`、および `NUM_PARTITIONS` を指定できます。これらのパラメータを使用すると、元のクエリは Spark タスクで同時に実行できるサブクエリの `NUM_PARTITIONS` の数に分割されます。

- `PARTITION_FIELD`: クエリのパーティション化に使用するフィールドの名前。
- `LOWER_BOUND`: 選択したパーティションフィールドの包括的な下限値。

  日付については、Spark SQL クエリで使用される Spark の日付形式を受け入れます。有効な値の例: `"2024-02-06"`。
- `UPPER_BOUND`: 選択したパーティションフィールドの排他的上限値。
- `NUM_PARTITIONS`: パーティション数。

エンティティごとのパーティション分割フィールドのサポートの詳細は、次の表にまとめられています。

エンティティ名	Partitioning Field	データタイプ
Ad Group Ad	date	日付
Ad Group	date	日付
キャンペーン	date	日付
Campaign Budget	date	日付

例

```
googleads_read = glueContext.create_dynamic_frame.from_options(
    connection_type="googleads",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "campaign-3467***",
        "API_VERSION": "v16",
        "PARTITION_FIELD": "date"
        "LOWER_BOUND": "2024-01-01"
        "UPPER_BOUND": "2024-06-05"
        "NUM_PARTITIONS": "10"
    }
)
```

Google 広告の接続オプション

Google 広告の接続オプションは次のとおりです。

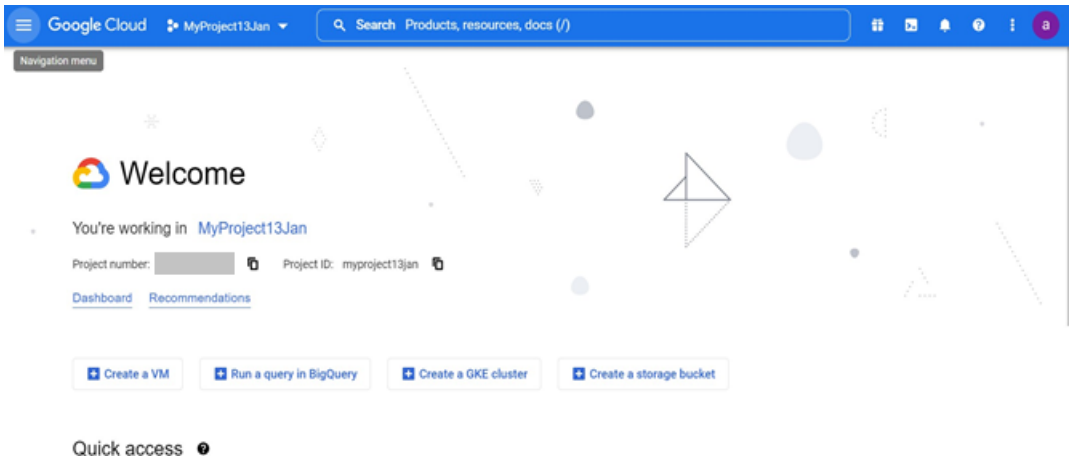
- ENTITY_NAME(文字列) – (必須) 読み取り / 書き込みに使用されます。Google 広告内のオブジェクトの名前。
- API_VERSION(文字列) – (必須) 読み取り / 書き込みに使用されます。使用する Google 広告の Rest API バージョン。例: v16。
- DEVELOPER_TOKEN(文字列) – (必須) 読み取り / 書き込みに使用されます。API にリクエストを行う開発者またはアプリケーションを認証するために必要です。
- MANAGER_ID(文字列) – 読み取り / 書き込みに使用されます。複数の Google 広告アカウントを管理できる一意の識別子。これは、承認されたマネージャーの顧客 ID です。顧客アカウントへの

アクセスがマネージャーアカウント経由である場合は、MANAGER_ID が必要です。詳細については、「login-customer-id」を参照してください。

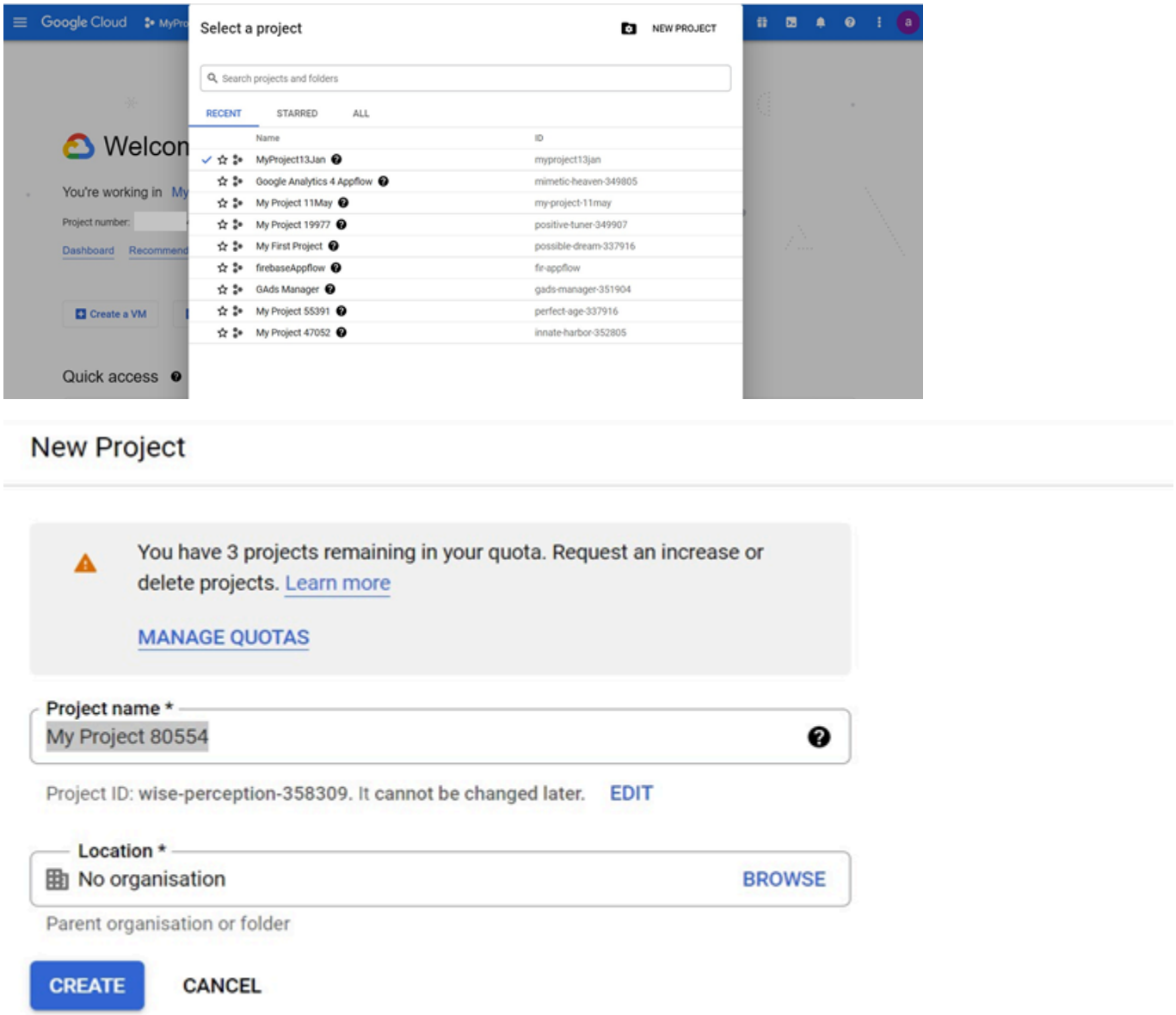
- SELECTED_FIELDS(List<String>) - Default: empty(SELECT *). 読み込みに使用されます。オブジェクトに選択する列。
- FILTER_PREDICATE(文字列) – デフォルト: 空 読み込みに使用されます。Spark SQL 形式である必要があります。
- QUERY(文字列) – デフォルト: 空 読み込みに使用されます。完全な Spark SQL クエリ。
- PARTITION_FIELD(文字列) – 読み取りに使用されます。クエリをパーティション化するために使用するフィールド。
- LOWER_BOUND(文字列) - 読み取りに使用されます。選択したパーティションフィールドの包括的な下限値。
- UPPER_BOUND(文字列) – 読み取りに使用されます。選択したパーティションフィールドの排他的上限値。
- NUM_PARTITIONS(整数) – デフォルト: 1。読み込みに使用されます。読み取り用のパーティションの数。

Google Ads アカウントの作成

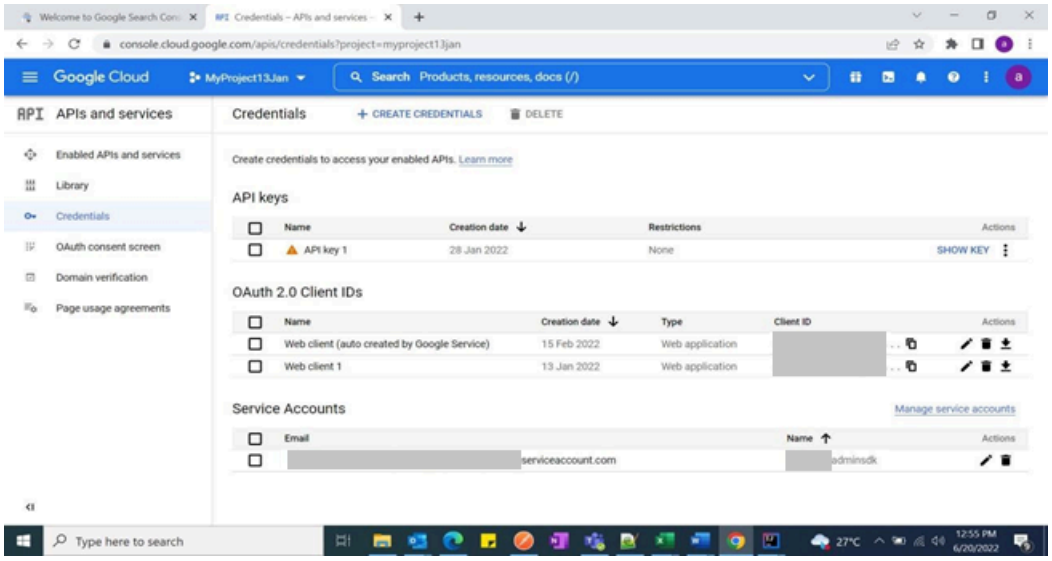
1. 認証情報を使用して Google Ads デベロッパーアカウントにログインし、* に移動します MyProject。



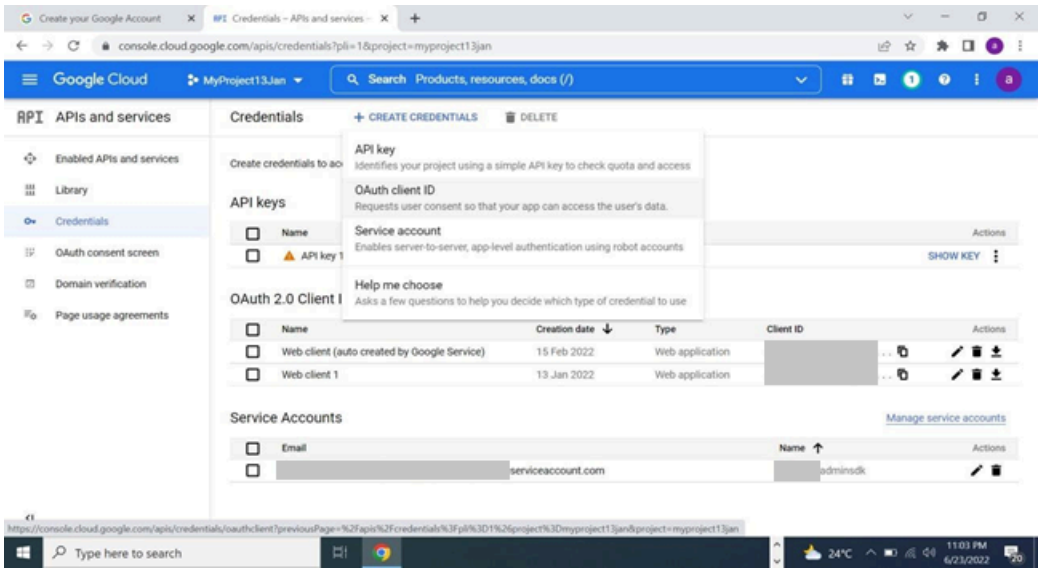
2. New Project を選択し、Google プロジェクトに登録されたアプリケーションがない場合は、Google プロジェクトの作成に必要な情報を入力します。



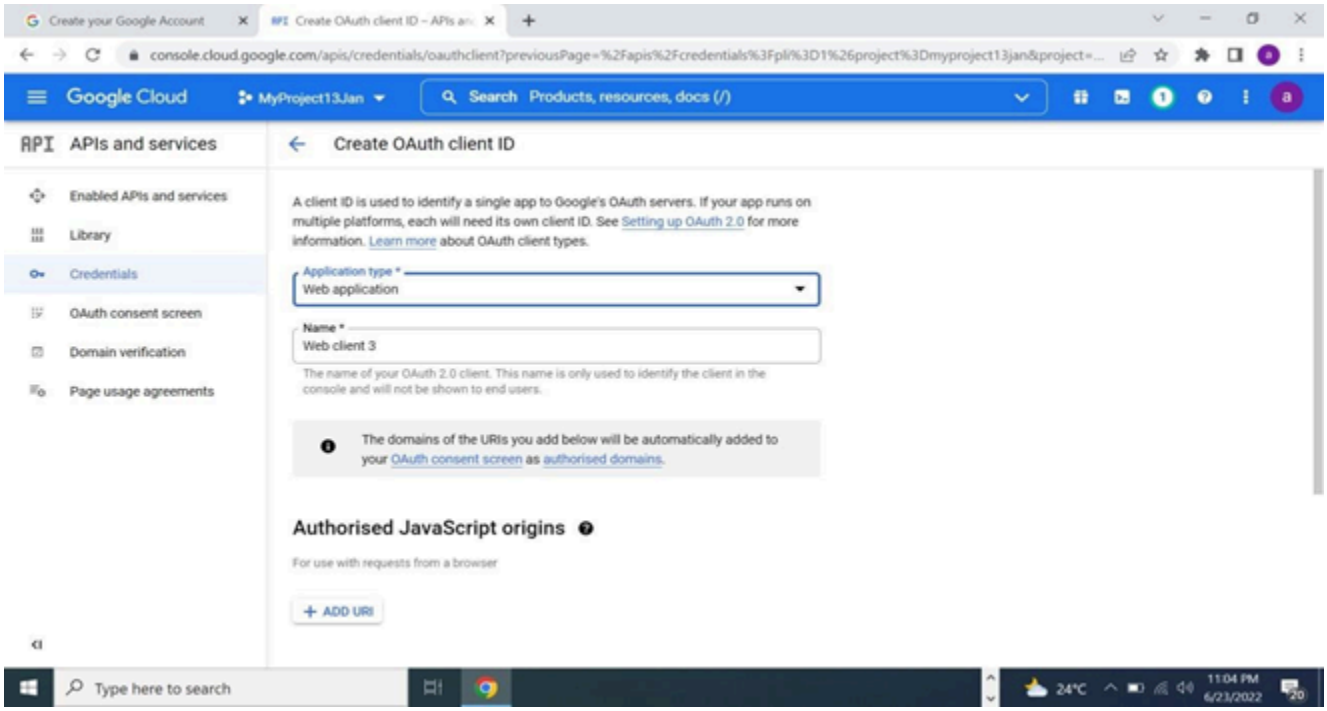
3. ナビゲーションタブ 、次にAPI設定 、クライアント ID の作成 および を選択します。ClientSecretこれにより、 AWS Glue と 間の接続を作成するための設定がさらに必要になります GoogleAds。詳細については、「 API認証情報」を参照してください。



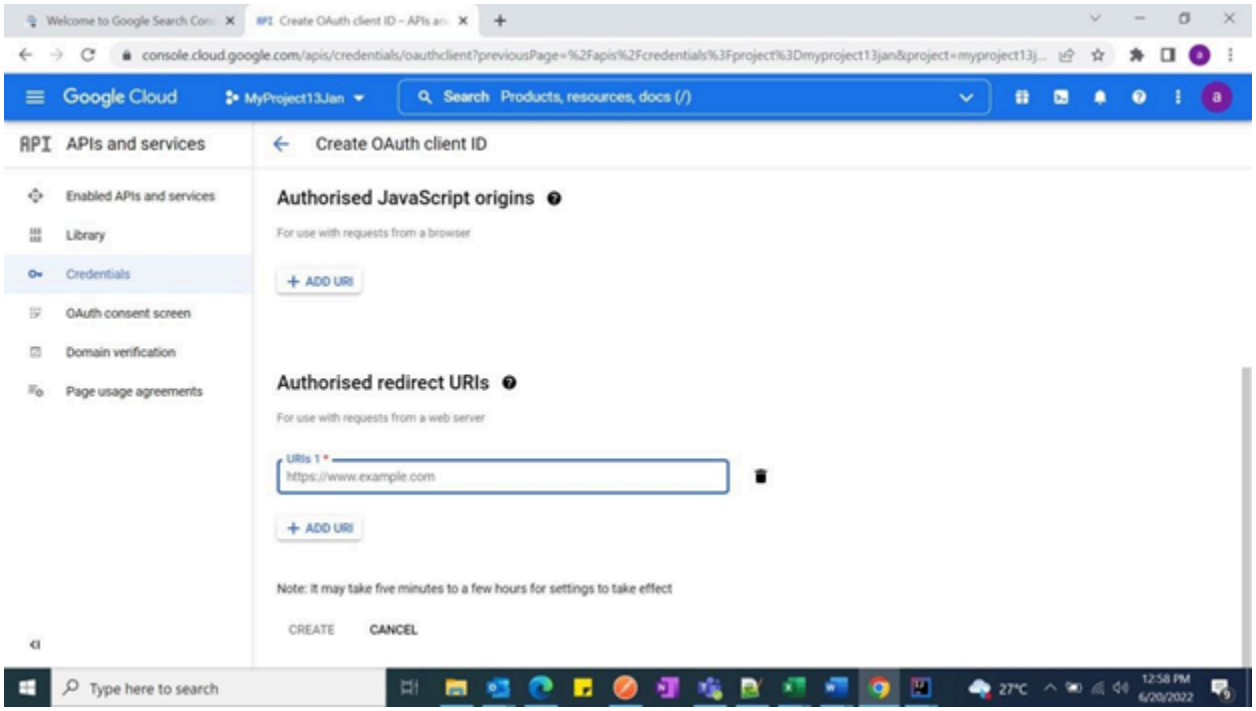
4. OAuth クライアント ID を選択してCREATECREDENTIALS選択します。



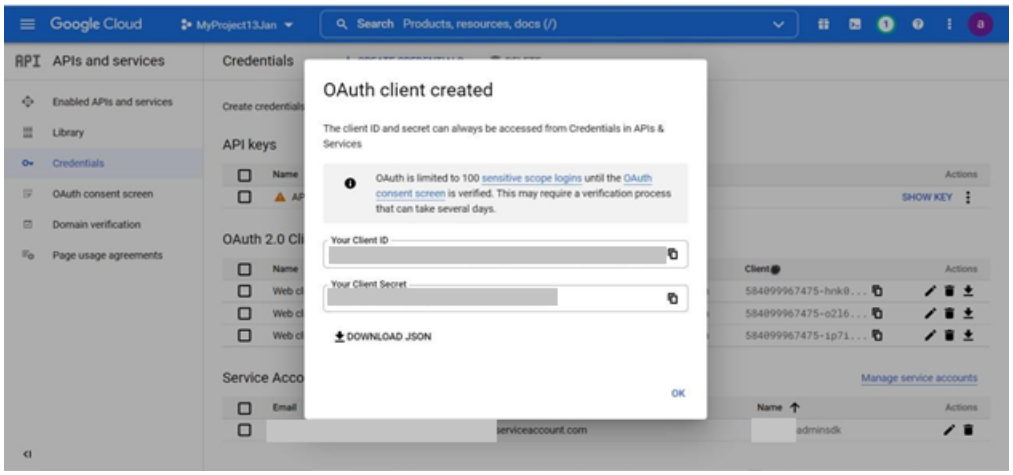
5. アプリケーションタイプをウェブアプリケーション として選択します。



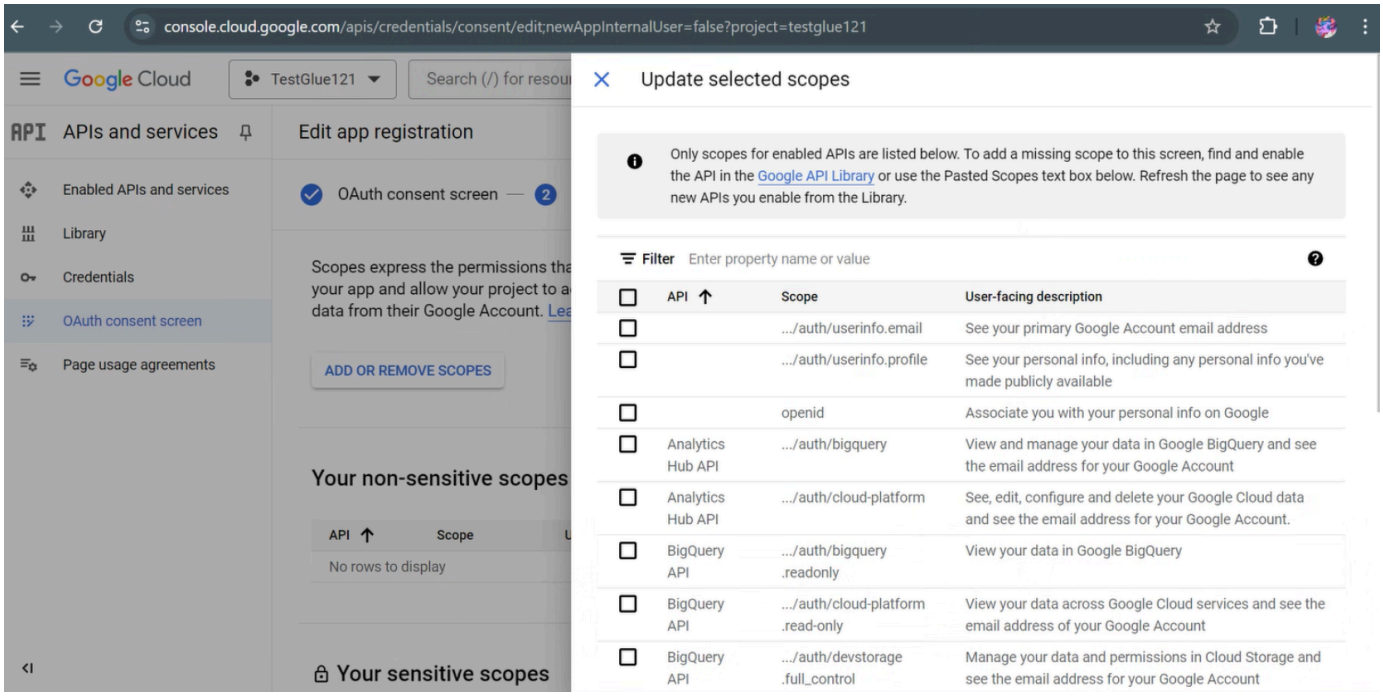
6. 「許可されたリダイレクトURIs」で、OAuthリダイレクトを追加URIsし、「 の作成」を選択します。URIs 必要に応じて、複数のリダイレクトを追加できます。



7. クライアント ID とクライアントシークレットは、 AWS Glue と Google Ads 間の接続を作成するときに生成されます。



8. アプリケーションのニーズに応じてスコープを追加し、OAuth同意画面を選択し、必要な情報を提供し、要件に基づいてスコープを追加します。



制限

Google 広告コネクタの制限は次のとおりです:

- MANAGER_ID は、接続を作成するときのオプションの入力です。ただし、特定のマネージャーの基盤となる顧客にアクセスする場合、MANAGER_ID は必須の入力です。以下の表では、MANAGER_ID が接続に含まれるかどうかに基づくアクセス制限について説明します。

接続の作成時に MANAGER_ID は提供されていますか?	アクセス可能な顧客
はい	指定された MANAGER_ID の下にリストされている顧客を、MANAGER_ID とともにリストします。
いいえ	すべての顧客が一覧表示されますが、マネージャーの基盤となる顧客はアクセスできません。

- マネージャーアカウントがオブジェクトとして選択されると、Account のみがサブオブジェクトとして表示されます。Google 広告コネクタでは、キャンペーン、広告などのエンティティは、マネージャーアカウントではなく個々のクライアントアカウントに基づいて取得されます。
- マネージャーアカウントのメトリクスを取得することはできません。代わりに、個々のクライアントアカウントのメトリクスを取得できます。
- 各アカウントには、アクティブなキャンペーンと一時停止されたキャンペーンの両方を含め、最大 10,000 のキャンペーンを設定できます。詳細については、「アカウントあたりのキャンペーン」を参照してください。
- レポートを作成するときに、表示する特定のメトリクスを選択した場合、選択したメトリクスがすべてゼロである行は返されません。詳細については、「ゼロメトリクス」を参照してください。
- 以下のフィールドでは、フルマッピングフローは、アカウント、広告グループや広告グループ広告エンティティ、特に conversionAction、conversionActionCategory、conversionActionName では機能しません。詳細については、「セグメントとメトリクス」を参照してください。
- segments.date フィールドが選択されている場合、日付範囲フィルターは必須です。

Google Analytics 4 への接続

Google Analytics 4 は、アプリやウェブサイトとの訪問者のやり取りに関するメトリクスを追跡および報告する分析サービスです。これらのメトリクスには、ページビュー、アクティブユーザー、イベントが含まれます。Google Analytics 4 ユーザーの場合は、Google Analytics 4 アカウント AWS Glue に接続できます。Google Analytics 4 をETLジョブのデータソースとして使用できます。これらのジョブを実行して、Google Analytics 4 から AWS サービスまたはその他のサポートされているアプリケーションにデータを転送します。

トピック

- AWS Glue Google Analytics 4 のサポート
- 接続を作成および使用するためのAPIオペレーションを含むポリシー
- Google Analytics の設定 4
- Google Analytics 4 接続の設定
- Google Analytics 4 エンティティからの読み取り
- Google Analytics 4 接続オプション
- Google Analytics 4 アカウントの作成
- クライアントアプリケーションと 2.0 OAuth 認証情報を作成する手順
- 制約事項と考慮事項

AWS Glue Google Analytics 4 のサポート

AWS Glue は、次のように Google Analytics 4 をサポートします。

ソースとしてサポートされていますか?

はい。ジョブを使用して AWS Glue ETL、Google Analytics 4 からデータをクエリできます。

ターゲットとしてサポートされているかどうか

いいえ。

サポートされている Google Analytics 4 APIバージョン

v1 ベータ。

接続を作成および使用するためのAPIオペレーションを含むポリシー

次のサンプルポリシーでは、接続の作成と使用に必要な AWS アクセス許可について説明します。新しいロールを作成する場合は、以下を含むポリシーを作成します。

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

```
            "Action": [
                "glue:ListConnectionTypes",
                "glue:DescribeConnectionType",
                "glue:RefreshOAuth2Tokens"
                "glue:ListEntities",
                "glue:DescribeEntity"
            ],
            "Resource": "*"
        }
    ]
}
```

以下の管理IAMポリシーを使用してアクセスを許可することもできます。

- AWSGlueServiceRole – さまざまな AWS Glue プロセスがユーザーに代わって実行するために必要なリソースへのアクセスを許可します。これらのリソースには AWS Glue、、Amazon S3、IAM、 CloudWatch Logs、Amazon が含まれますEC2。このポリシーで指定されたリソースの命名規則に従うと、 AWS Glue プロセスには必要なアクセス許可があります。このポリシーは、通常、クローラ、ジョブ、開発エンドポイントを定義するときに指定されたロールにアタッチされます。
- AWSGlueConsoleFullAccess — ポリシーがアタッチされている ID が マネジメントコンソールを使用する場合 AWS 、 AWS Glue リソースへのフルアクセスを許可します。このポリシーで指定されたリソースの命名規則に従った場合、ユーザーは完全なコンソール機能を使用できます。このポリシーは通常、 AWS Glue コンソールのユーザーにアタッチされます。

Google Analytics の設定 4

AWS Glue を使用して Google Analytics 4 から転送する前に、次の要件を満たす必要があります。

最小要件

- 転送するデータを収集する 1 つ以上のデータストリームを持つ Google Analytics アカウントがあります。
- Google Cloud Platform アカウントと Google Cloud プロジェクトがあります。
- Google Cloud プロジェクトでは、次の が有効になっていますAPIs。
  - Google Analytics API
  - Google Analytics 管理者 API
  - Google Analytics データ API

- Google Cloud プロジェクトでは、外部ユーザーのOAuth同意画面を設定しました。OAuth 同意画面の詳細については、Google Cloud Platform コンソールヘルプのOAuth「同意画面の設定」を参照してください。
- Google Cloud プロジェクトで、2.0 OAuth クライアント ID を設定しました。詳細については、「 2.0 OAuth のセットアップ」を参照してください。

これらの要件を満たすと、Google Analytics 4 アカウント AWS Glue に接続できます。

Google Analytics 4 接続の設定

Google Sheet 接続を設定するには：

1. AWS Secrets Manager で、次の詳細を含むシークレットを作成します。AWS Glue で接続ごとにシークレットを作成する必要があります。
   a. AuthorizationCode グラントタイプの場合:
      - カスタマーマネージド接続アプリケーションの場合 – シークレットには、USER_MANAGED_CLIENT_APPLICATION_CLIENT_SECRET を使用して接続されたアプリケーションのコンシューマーシークレットをキーとして含める必要があります。
2. AWS Glue Glue Studio で、以下の手順に従って [データ接続] の下に接続を作成します。
   a. [接続タイプ] を選択する際に、[Google Analytics 4] を選択します。
   b. 接続先の Google Analytics 4 の INSTANCE_URL を指定します。
   c. AWS Glue が次のアクションを担い、その権限を持つことができる IAM ロールを選択します。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:DescribeSecret",
                "secretsmanager:GetSecretValue",
                "secretsmanager:PutSecretValue",
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterface",
                "ec2:DeleteNetworkInterface",
            ],
            "Resource": "*"
        }
```

```
    ]
}
```

d. AWS Glue でこの接続に使用する `secretName` を選択して、トークンを配置します。

e. ネットワークを使用する場合は、ネットワークオプションを選択します。

3. AWS Glue ジョブに関連付けられている IAM ロールに `secretName` を読み取るアクセス許可を付与します。

`AUTHORIZATION_CODE` 付与タイプ。

このグラントタイプは、ユーザーを認証するためにサードパーティーの認証サーバーにユーザーをリダイレクトすることから、「three-legged」の OAuth と見なされます。AWS Glue コンソール経由で接続を作成するときに使用されます。AWS Glue コンソールでは、ユーザーが Google Analytics 4 にリダイレクトされます。ここではユーザーがログインし、Google Analytics 4 インスタンスにアクセスするためのリクエストされた権限を AWS Glue に許可する必要があります。

ユーザーは、Google Analytics 4 で独自の接続アプリを作成し、AWS Glue コンソールを介して接続を作成するときに独自のクライアント ID とクライアントシークレットを指定することを選択できます。このシナリオでは、引き続き Google Analytics 4 にリダイレクトされてログインし、リソースへアクセスするために AWS Glue を承認します。

このグラントタイプは、更新トークンとアクセストークンになります。アクセストークンの有効期間は短く、更新トークンを使用してユーザーとやり取りすることなく自動的に更新される場合があります。

詳細については、「Auth 2.0 を使用した Google API へのアクセス」を参照してください。

Google Analytics 4 エンティティからの読み取り

前提条件

- 読み取る Google Analytics 4 オブジェクト。使用可能なエンティティを確認するには、以下のサポートされているエンティティテーブルを参照してください。

サポートされているエンティティ

エンティティ	フィルタリング可能	制限をサポート	注文者をサポート	Select * をサポート	パーティショニングをサポート
リアルタイムレポート	あり	はい	はい	はい	なし
コアレポート	あり	はい	はい	はい	はい

例

```
googleAnalytics4_read = glueContext.create_dynamic_frame.from_options(
    connection_type="GoogleAnalytics4",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "entityName",
        "API_VERSION": "v1beta"
    }
```

Google Analytics 4 エンティティとフィールドの詳細

エンティティ	フィールド	データタイプ	サポートされる演算子
コアレポート	動的フィールド		
コアレポート	属性フィールド	文字列	LIKE, =
コアレポート	属性フィールド	日付	LIKE, =
コアレポート	メトリクスフィールド	文字列	>, <, >=, <=, = BETWEEN
コアレポート	カスタムディメンションフィールドとカスタムメトリクスフィールド	文字列	該当なし

エンティティ	フィールド	データタイプ	サポートされる演算子
リアルタイムレポート	appVersion	文字列	LIKE, =
リアルタイムレポート	audienceId	文字列	LIKE, =
リアルタイムレポート	audienceName	文字列	LIKE, =
リアルタイムレポート	city	文字列	LIKE, =
リアルタイムレポート	cityId	文字列	LIKE, =
リアルタイムレポート	country	文字列	LIKE, =
リアルタイムレポート	countryId	文字列	LIKE, =
リアルタイムレポート	deviceCategory	文字列	LIKE, =
リアルタイムレポート	eventName	文字列	LIKE, =
リアルタイムレポート	minutesAgo	文字列	LIKE, =
リアルタイムレポート	platform	文字列	LIKE, =
リアルタイムレポート	streamId	文字列	LIKE, =

エンティティ	フィールド	データタイプ	サポートされる演算子
リアルタイムレポート	streamName	文字列	LIKE, =
リアルタイムレポート	unifiedScreenName	文字列	LIKE, =
リアルタイムレポート	activeUsers	文字列	>, <, >=, <=, = BETWEEN
リアルタイムレポート	変換	文字列	>, <, >=, <=, = BETWEEN
リアルタイムレポート	eventCount	文字列	>, <, >=, <=, = BETWEEN
リアルタイムレポート	screenPageViews	文字列	>, <, >=, <=, = BETWEEN

パーティションクエリ

1. フィルターベースのパーティション

   Spark で同時実行を使用する場合はPARTITION_FIELD、追加のスパークオプション LOWER_BOUND、UPPER_BOUND、 NUM_PARTITIONSを指定できます。これらのパラメータを使用すると、元のクエリは、スパークタスクで同時に実行できるサブクエリNUM_PARTITIONSの数に分割されます。

   - PARTITION_FIELD: クエリをパーティション化するために使用されるフィールドの名前。
   - LOWER_BOUND: 選択したパーティションフィールドの包括的な下限値。

     日付については、Spark SQLクエリで使用される Spark 日付形式を受け入れます。有効な値の例: "2024-02-06"。
   - UPPER_BOUND: 選択したパーティションフィールドの排他的上限値。
   - NUM_PARTITIONS: パーティションの数。

   例

```
googleAnalytics4_read = glueContext.create_dynamic_frame.from_options(
    connection_type="GoogleAnalytics4",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "entityName",
        "API_VERSION": "v1beta",
        "PARTITION_FIELD": "date"
        "LOWER_BOUND": "2022-01-01"
        "UPPER_BOUND": "2024-01-02"
        "NUM_PARTITIONS": "10"
    }
```

2. レコードベースのパーティション

   Spark で同時実行を使用する場合は、追加のスパークオプションNUM_PARTITIONSを使用できます。これらのパラメータを使用すると、元のクエリは、スパークタスクで同時に実行できるサブクエリNUM_PARTITIONSの数に分割されます。

   - NUM_PARTITIONS: パーティションの数。

   例

```
googleAnalytics4_read = glueContext.create_dynamic_frame.from_options(
    connection_type="GoogleAnalytics4",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "entityName",
        "API_VERSION": "v1beta",
        "NUM_PARTITIONS": "10"
    }
```

Google Analytics 4 接続オプション

Google Analytics 4 の接続オプションは次のとおりです。

- ENTITY_NAME (文字列) - (必須) 読み取りに使用されます。Google Analytics 4 のオブジェクトの名前。
- API_VERSION (文字列) - (必須) 読み取りに使用されます。使用する Google Analytics 4 Rest API バージョン。

- SELECTED_FIELDS (リスト < 文字列>) - デフォルト: empty(SELECT *)。読み込みに使用されます。オブジェクトに選択する列。
- FILTER_PREDICATE(文字列) - デフォルト: 空 読み込みに使用されます。Spark SQL形式である必要があります。
- QUERY(文字列) - デフォルト: 空 読み込みに使用されます。フル Spark SQLクエリ。
- PARTITION_FIELD (文字列) - 読み取りに使用されます。クエリをパーティション化するために使用するフィールド。
- LOWER_BOUND(文字列) - 読み取りに使用されます。選択したパーティションフィールドの包括的な下限値。
- UPPER_BOUND (文字列) - 読み取りに使用されます。選択したパーティションフィールドの排他的上限値。
- NUM_PARTITIONS(整数) - デフォルト: 1。読み込みに使用されます。読み取り用のパーティションの数。
- INSTANCE_URL (整数) - 読み取りに使用されます。(オプション)

Google Analytics 4 アカウントの作成

Google Analytics 4 アカウントを作成するには、https://support.google.com/analytics/answer/9304153?hl=en」のステップに従います。

クライアントアプリケーションと 2.0 OAuth 認証情報を作成する手順

詳細については、「Google Analytics4 APIドキュメント」を参照してください。

1. 認証情報を使用して Google Analytics アカウントにログインして、アカウントを作成してセットアップします。次に、Admin > Create Account に移動します。
2. プロパティの作成 を選択して、作成したアカウントのプロパティを作成します。必要な詳細を含むプロパティを設定します。指定されたすべての詳細が、対応するプロパティ ID が生成されます。
3. ドロップダウンから Data Streams > Add Stream > Web を選択して、作成したプロパティのデータストリームを追加します。などのウェブサイトの詳細URLと、その他の必須フィールドを指定します。すべての詳細を入力すると、対応するストリーム ID と測定 ID が生成されます。
4. 測定 ID をコピーしてウェブサイトで Google Analytics を設定し、ウェブサイトの設定に追加します。
5. レポートに移動し、必要なレポートを生成して、Google Analytics からレポートを作成します。

6. console.cloud.google.com に移動して Google Analytics Data を検索しAPI、 を有効にして、アプリを承認しますAPI。

   1. API および サービスページに移動し、認証情報 > セットアップ OAuth 2.0 クライアント IDsを選択します。
   2. Redirect URLを追加して AWS Glue リダイレクトを提供しますURL。

7. 接続の作成にさらに必要となるクライアント ID とクライアントシークレットをコピーします。

制約事項と考慮事項

Google Analytics 4 コネクタの制限は次のとおりです。

- Core Report エンティティの場合、リクエストで送信できるのは 9 つのディメンションフィールドと 10 のメトリクスフィールドのみです。許可されたフィールド数を超えると、リクエストは失敗し、コネクタはエラーメッセージをスローします。
- リアルタイムレポートエンティティの場合、リクエストで送信できるのは 4 つのディメンションフィールドのみです。許可されたフィールド数を超えると、リクエストは失敗し、コネクタはエラーメッセージをスローします。
- Google Analytics 4 はベータ版の無料ツールであるため、新機能、エンティティの強化、新しいフィールドの追加、既存のフィールドの非推奨化を定期的に更新します。
- コアレポートフィールドは動的に入力されるため、フィールドの追加、減価償却、名前の変更、フィールドへの新しい制限の適用をいつでも行うことができます。
- デフォルトの開始日は 30 日で、終了日は昨日 (現在の日付の 1 日前) です。ユーザーが値を設定している場合、またはフローが増分の場合、これらの日付はフィルター式コードで上書きされます。
- ドキュメントによると、リアルタイムレポートエンティティは、リクエストで制限が渡されない場合に 10,000 個のレコードを返します。そうでない場合、リクエストの数に関係なく、 はリクエストごとに最大 250,000 行APIを返します。詳細については、Google Analytics ドキュメントの「メソッド: プロパティrunRealtimeReport」を参照してください。
- リアルタイムレポートエンティティはページ分割をサポートしていないため、レコードベースのパーティションをサポートしていません。また、定義された基準を満たしているフィールドがないため、フィールドベースのパーティションはサポートされていません。
- リクエストで渡すことができるフィールドの数に制限があるため。デフォルトのディメンションフィールドとメトリクスフィールドは、指定された制限内で設定します。「すべて選択」が選択されている場合、これらの事前定義されたフィールドのデータのみが取得されます。
  - コアレポート

- SAAS からの制限により、リクエストは最大 9 つのディメンションと最大 10 個のメトリクスのみが許可されます (つまり、リクエストには最大 19 個のフィールド (メトリクス + ディメンション) を含めることができます。
- 実装に従って - ユーザーが SELECT_ALL または 25 を超える選択したフィールドを使用する場合、デフォルトのフィールドはリクエストで渡されます。
- 次のフィールドは、コアレポート - 「国」、「都市」、eventName「」、cityId「ブラウザ」、「日付」、currencyCode「」、deviceCategory「」、」、transactionId「アクティブ 1DayUsers」、「アクティブ 28DayUsers」、「アクティブ 7DayUsers」、」、activeUsers「」、averagePurchaseRevenue「」、averageRevenuePer「ユーザー」、averageSessionDuration」、「」、engagedSessions「」、「」、eventCount」「」、engagementRate「」」のデフォルトフィールドと見なされます。

- リアルタイムレポート
  - SAAS リクエストの制限により、最大 4 つのディメンションが許可されます。
  - ユーザーが SELECT_ALL または 15 を超えるフィールドを渡すと、デフォルトのフィールドはリクエストで渡されます。
  - 次のフィールドは、 RealTime Report のデフォルトフィールドと見なされます。「国」、deviceCategory「」、「都市」、cityId「」、activeUsers「変換」、eventCount「」、screenPageViews「」。

- Core-Report エンティティで、日付フィールドのパーティションと のフィルター startDate が同時に存在する場合。その場合、 dateRange 値は startDate フィルター値で上書きされますが、パーティションは常に優先順位であるため、日付フィールドのパーティションがすでに存在する場合は startDate フィルターを破棄します。
- 現在 cohortSpecs はコアレポートリクエスト本文の一部でもあるため、現在のコアレポートエンティティを強化して、 cohortSpec 属性のサポートを含めました。 cohortSpecs リクエスト本文では、ほぼすべてのフィールドでユーザー入力が必要です。これに対処するために、これらの属性/フィールドにデフォルト値を設定し、必要に応じてユーザーがこれらの値を上書きできるようにプロビジョニングしました。

FieldName	デフォルト値	デフォルト値を上書きする filterPredicate オプションを渡すサンプルクエリ
startDate	現在の日付から 30 日前	2023-05-09」と2023-05-10startDate 」の間

FieldName	デフォルト値	デフォルト値を上書きする filterPredicate オプションを渡すサンプルクエリ
endDate	現在の日付から 1 日前	2023-05-09」と2023-05-10startDate 」の間
startOffset	0	startOffset=2
endOffset	1	endOffset=10
詳細度	DAILY	粒度WEEKLY「」

- これらのフィルターを一度に渡すことも、他のフィルターと一緒に渡すこともできます。
  - 例 1 - filterPredicate: startDate 2023-05-09」と2023-05-10ANDstartOffset」 =1 AND endOffset=2 AND粒度WEEKLY「」
  - 例 2 - filterPredicate: city="xyz" AND startOffset=1 AND endOffset=2 AND粒度WEEKLY「」
- コホートリクエストの場合：
  - リクエストでcohortNthMonth「」が渡された場合、内部粒度値はMONTHLY「」として設定されます。
  - 同様に、cohortNthWeek「」が渡されると、粒度値はWEEKLY「」として設定されます。
  - また、cohortNthDay「」の粒度値はDAILY「」として設定されます。詳細については、以下を参照してください。
    - https://developers.google.com/analytics/devguides/reporting/data/v1/advanced
    - https://developers.google.com/analytics/devguides/reporting/data/v1/rest/v1beta/CohortSpec
  - ユーザーが dateRange と粒度のデフォルト値を上書きできるようにプロビジョニングが与えられます。上記の表を参照してください。

AWS Glue Studio で Google BigQuery に接続する

> **Note**
>
> AWS Glue for Spark を使用して、AWS Glue 4.0 以降のバージョンの Google BigQuery のテーブルからの読み取りとテーブルへの書き込みを行うことができます。AWS Glue ジョブ

と Google BigQuery をプログラム的に設定する方法については、「BigQuery 接続」を参照してください。

AWS Glue Studio には、BigQuery へ接続し、データ統合ジョブを作成し、AWS Glue Studio サーバーレス Spark ランタイム上でそれらを実行するビジュアルインターフェイスが用意されています。

AWS Glue Studio で Google BigQuery への接続を作成すると、統合接続が作成されます。詳細については、「考慮事項」を参照してください。

特定の形式 `{"credentials": "base64 encoded JSON"}` の認証情報を使用してシークレットを作成する代わりに、Google BigQuery への統合接続を使用して、Golge BigQuery の JSON を直接含むシークレットを作成できます: `{"type": "service-account", ...}`。

トピック

- BigQuery 接続を作成する
- BigQuery ソースノードの作成
- BigQuery ターゲットノードを作成する
- 詳細オプション

BigQuery 接続を作成する

AWS Glue から Google BigQuery へ接続するには、Google Cloud Platform の認証情報を作成して AWS Secrets Manager シークレットに保存し、そのシークレットを Google BigQuery AWS Glue 接続に関連付ける必要があります。

BigQuery への接続を設定するには:

1. Google Cloud Platform で、関連するリソースを作成して特定します。
   - 接続したい BigQuery テーブルを含む GCP プロジェクトを作成または特定します。
   - BigQuery API を有効にします。詳細については、「Use the BigQuery Storage Read API to read table data」を参照してください。
2. Google Cloud Platform で、サービスアカウントの認証情報を作成してエクスポートします。

   BigQuery 認証情報ウィザードを使用すると、「認証情報の作成」というステップを迅速に実行できます。

GCP でサービスアカウントを作成するには、「サービス アカウントを作成する」にあるチュートリアルに従ってください。

- プロジェクトを選択するときは、BigQuery テーブルを含むプロジェクトを選択します。
- サービスアカウントの GCP IAM ロールを選択するときは、BigQuery テーブルの読み取り、書き込み、作成を行う BigQuery ジョブを実行するための適切な権限を付与するロールを追加または作成します。

サービスアカウントの認証情報を作成するには、「サービス アカウント キーを作成する」にあるチュートリアルに従ってください。

- キータイプを選択するときは、[JSON] を選択します。

これで、サービスアカウントの認証情報が記載された JSON ファイルがダウンロードされたはずです。これは次のように表示されます。

```
{
  "type": "service_account",
  "project_id": "*****",
  "private_key_id": "*****",
  "private_key": "*****",
  "client_email": "*****",
  "client_id": "*****",
  "auth_uri": "https://accounts.google.com/o/oauth2/auth",
  "token_uri": "https://oauth2.googleapis.com/token",
  "auth_provider_x509_cert_url": "https://www.googleapis.com/oauth2/v1/certs",
  "client_x509_cert_url": "*****",
  "universe_domain": "googleapis.com"
}
```

3. AWS Secrets Manager で、ダウンロードした認証情報ファイルを使用してシークレットを作成します。[プレーンテキスト] タブを選択し、JSON 形式のファイルコンテンツを貼り付けることができます。Secrets Manager でシークレットを作成するには、AWS Secrets Manager ドキュメントの「AWS Secrets Manager シークレットを作成する」にあるチュートリアルに従ってください。シークレットを作成したら、次のステップのためにシークレット名 *secretName* を保存しておきます。

4. AWS Glue データカタログで、https://docs.aws.amazon.com/glue/latest/dg/console-connections.html にある手順に従って接続を作成します。接続を作成したら、次のステップのために接続名 *connectionName* を保存しておきます。

   - [接続タイプ] を選択するときは、Google BigQuery を選択してください。
   - [AWS Secret] をクリックして、*secretName* を入力します。

5. AWS Glue ジョブに関連付けられている IAM ロールに *secretName* を読み取るアクセス許可を付与します。
6. AWS Glue ジョブ設定で、[Additional network connection] として *[connectionName]* を指定します。

BigQuery ソースノードの作成

必要な前提条件

- BigQuery タイプの AWS Glue データカタログ接続
- 接続時に使用される Google BigQuery 認証情報の AWS Secrets Manager シークレット。
- 接続で使用されるシークレットを読み取るためのジョブに対する適切なアクセス許可。
- 読み取る対象のテーブルおよび対応する Google Cloud プロジェクトの名前とデータセット。

BigQuery データソースを追加する

データソース — BigQuery ノードを追加するには

1. BigQuery データソースの接続を選択します。すでに作成したので、ドロップダウンに表示されているはずです。接続を作成する必要がある場合は、[Create BigQuery connection] を選択します。詳細については、「コネクタと接続の使用に関する概要」を参照してください。

   接続を選択したあとは、[プロパティを表示] をクリックすると、接続のプロパティを表示できます。
2. 読み取る BigQuery データを特定し、[BigQuery Source] オプションを選択します。

   - Choose a single table - テーブルからすべてのデータを取得できます。
   - Enter a custom query — クエリを指定することで、取得するデータをカスタマイズできます。

3. 読み取るデータの説明を記述します。

(必須) テーブルを含むプロジェクト、または該当する場合は、請求元の親プロジェクトに[親プロジェクト]を設定します。

1 つのテーブルを選択した場合は、[テーブル] を次の形式で Google BigQuery テーブルの名前に設定します。`[dataset].[table]`

クエリを選択した場合は、それを [クエリ] に提供します。クエリでは、`[project].[dataset].[tableName]` の形式の完全修飾テーブル名の付いたテーブルを参照します。

4. BigQuery プロパティを指定します。

   1 つのテーブルを選択した場合は、追加のプロパティを指定する必要はありません。

   クエリを選択した場合は、以下の [Custom Google BigQuery properties] を指定する必要があります。

   - `viewsEnabled` を true に設定します。
   - `materializationDataset` をデータセットに設定します。AWS Glue 接続を通じて提供された認証情報で認証された GCP プリンシパルは、このデータセットにテーブルを作成できる必要があります。

BigQuery ターゲットノードを作成する

必要な前提条件

- BigQuery タイプの AWS Glue データカタログ接続
- 接続時に使用される Google BigQuery 認証情報の AWS Secrets Manager シークレット。
- 接続で使用されるシークレットを読み取るためのジョブに対する適切なアクセス許可。
- 書き込み対象のテーブルおよび対応する Google Cloud プロジェクトの名前とデータセット。

BigQuery データターゲットを追加する

[Data target – BigQuery] ノードを追加するには:

1. BigQuery データターゲットの接続を選択します。すでに作成したので、ドロップダウンに表示されているはずです。接続を作成する必要がある場合は、[Create BigQuery connection] を選択します。詳細については、「コネクタと接続の使用に関する概要」を参照してください。

接続を選択したあとは、[プロパティを表示] をクリックすると、接続のプロパティを表示できます。

2. どの BigQuery テーブルに書き込むかを特定し、[Write method] を選択します。

   - 直接 — BigQuery Storage Write API を使用して BigQuery に直接書き込みます。
   - 間接 — Google Cloud Storage に書き込んでから BigQuery にコピーします。

   間接的に書き込む場合は、一時 GCS バケットを使用して送信先の GCS ロケーションを指定します。AWS Glue 接続時に追加の設定を行う必要があります。詳細については、「Using indirect write with Google BigQuery」を参照してください。

3. 読み取るデータの説明を記述します。

   (必須) テーブルを含むプロジェクト、または該当する場合は、請求元の親プロジェクトに[親プロジェクト]を設定します。

   1 つのテーブルを選択した場合は、[テーブル] を次の形式で Google BigQuery テーブルの名前に設定します。`[dataset].[table]`

詳細オプション

BigQuery ノードを作成するときに、高度なオプションを指定できます。これらのオプションは Spark AWS Glue スクリプトのプログラミング時に使用できるオプションと同じです。

「AWS Glue デベロッパーガイド」の「BigQuery 接続オプションリファレンス」をご覧ください。

Google 検索コンソールに対する接続

Google 検索コンソールは、Google がサイトをどのように表示し、その存在を最適化するかをモニタリングするために、ウェブサイト所有者が利用できる無料のプラットフォームです。これには、参照元ドメイン、モバイルサイトのパフォーマンス、豊富な検索結果、トラフィックの多いクエリとページの表示が含まれます。Google 検索コンソールユーザーの場合は、Google 検索コンソールアカウントに AWS Glue を接続できます。Google 検索コンソールを ETL ジョブのデータソースとして使用できます。これらのジョブを実行して、Google 検索コンソールから AWS サービスまたはその他のサポートされているアプリケーションにデータを転送します。

トピック

- Google 検索コンソールの AWS Glue サポート

- 接続を作成および使用するための API オペレーションを含むポリシー
- Google 検索コンソールの設定
- Google 検索コンソール接続の設定
- Google 検索コンソールエンティティからの読み取り
- Google 検索コンソールの接続オプション
- Google 検索コンソールの制限

Google 検索コンソールの AWS Glue サポート

AWS Glue は、次のように Google 検索コンソールをサポートします。

ソースとしてサポートされていますか?

はい。AWS Glue ETL ジョブを使用して、Google 検索コンソールからデータをクエリできます。

ターゲットとしてサポートされていますか?

いいえ。

サポートされている Google 検索コンソールの API バージョン

次の Google 検索コンソールの API バージョンがサポートされています。

- v3

接続を作成および使用するための API オペレーションを含むポリシー

次のサンプル ポリシーで、接続の作成と使用に必要な AWS IAM 権限について説明します。新しいロールを作成する場合は、以下を含むポリシーを作成します。

```
{
 "Version": "2012-10-17",
     "Statement": [
      {
             "Effect": "Allow",
             "Action": [
                 "glue:ListConnectionTypes",
                 "glue:DescribeConnectionType",
                 "glue:RefreshOAuth2Tokens",
```

```
                "glue:ListEntities",
                "glue:DescribeEntity"
            ],
            "Resource": "*"
        }
    ]
}
```

上記の方法を使用しない場合は、代わりに次のマネージド IAM ポリシーを使用します:

- AWSGlueServiceRole – さまざまな AWS Glue プロセスを実行するために必要なリソースへのアクセス権をユーザーに代わって付与します。これらのリソースには AWS Glue 、Amazon S3、IAM、CloudWatch Logs、および Amazon EC2 が含まれます。このポリシーで指定されたリソースの命名規則に従った場合、AWS Glue プロセスは必要なアクセス権限を使用できます。このポリシーは、通常、クローラ、ジョブ、開発エンドポイントを定義するときに指定されたロールにアタッチされます。
- AWSGlueConsoleFullAccess — ポリシーがアタッチされているアイデンティティが AWS マネージメントコンソールを使用するときは、AWS Glue リソースへのフルアクセスを許可します。このポリシーで指定されたリソースの命名規則に従った場合、ユーザーは完全なコンソール機能を使用できます。このポリシーは、通常 AWS Glue コンソールのユーザーにアタッチされています。

Google 検索コンソールの設定

AWS Glue を使用して Google 検索コンソールからデータを転送できるようにするには、次の要件を満たす必要があります。

最小要件

以下に、最小要件を示します。

- Google 検索コンソールアカウントがある。
- Google Cloud Platform のアカウントと Google Cloud プロジェクトがある。
- Google Cloud プロジェクトで、Google 検索コンソール API を有効にしている。
- Google Cloud プロジェクトで、外部ユーザー用の OAuth 同意画面を設定している。詳細については、Google Cloud Platform コンソールヘルプの「OAuth 同意画面の設定」を参照してください。
- Google Cloud プロジェクトで、OAuth 2.0 クライアント ID を設定している。アカウントへの認証呼び出しを行うときに AWS Glue がデータに安全にアクセスするために使用するクライアント認証情報については、「OAuth 2.0 の設定」を参照してください。

これらの要件を満たすと、Google 検索コンソールアカウントに AWS Glue を接続する準備が整います。一般的な接続では、Google 検索コンソールで他に何もする必要はありません。

Google 検索コンソール接続の設定

Google 検索コンソールは、OAuth2 の AUTHORIZATION_CODE グラントタイプをサポートしています。グラントタイプは、AWS Glue が Google 検索コンソールと通信してデータへのアクセスをリクエストする方法を決定します。

- このグラントタイプは、ユーザーを認証するためにサードパーティーの認証サーバーにユーザーをリダイレクトすることから、「three-legged」の OAuth と見なされます。AWS Glue コンソール経由で接続を作成するときに使用されます。
- ユーザーは、Google 検索コンソールで独自の接続アプリを作成し、AWS Glue コンソールを介して接続を作成するときに独自のクライアント ID とクライアントシークレットを指定することを選択できます。このシナリオでは、引き続き Google 検索コンソールにリダイレクトされてログインし、リソースへアクセスするために AWS Glue を承認します。
- このグラントタイプは、更新トークンとアクセストークンになります。アクセストークンの有効期間は短く、更新トークンを使用してユーザーとやり取りすることなく自動的に更新される場合があります。
- 認可コード OAuth フロー用の接続アプリの作成に関する Google 検索コンソールの公開ドキュメントについては、「OAuth 2.0 を使用した Google API へのアクセス」を参照してください。

Google 検索コンソール接続を設定するには:

1. AWS Secrets Manager で、次の詳細を含むシークレットを作成します。
   a. カスタマーマネージド接続アプリケーションの場合、シークレットには、USER_MANAGED_CLIENT_APPLICATION_CLIENT_SECRET を使用して接続されたアプリケーションのコンシューマーシークレットをキーとして含める必要があります。
   b. 注: AWS Glue で接続にシークレットを作成する必要があります。

1. AWS Glue Glue Studio で、以下の手順に従って [データ接続] の下に接続を作成します。
   a. [接続タイプ] を選択する際に、[Google 検索コンソール] を選択します。
   b. 次のアクションを実行でき、AWS Glue がその権限を持つ AWS IAM ロールを選択します。

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

```
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:DescribeSecret",
                "secretsmanager:GetSecretValue",
                "secretsmanager:PutSecretValue",
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterface",
                "ec2:DeleteNetworkInterface",
            ],
            "Resource": "*"
        }
    ]
}
```

c. AWS Glue でこの接続に使用する `secretName` を選択して、トークンを配置します。

d. ネットワークを使用する場合は、ネットワークオプションを選択します。

2. AWS Glue ジョブに関連付けられている IAM ロールに `secretName` を読み取るアクセス許可を付与します。

Google 検索コンソールエンティティからの読み取り

前提条件

読み取り元の Google 検索コンソールオブジェクト。オブジェクト名が必要になります。

ソースに対応するエンティティ:

エンティティ	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
[分析] を選択する	はい	はい	いいえ	はい	いいえ
サイト	いいえ	いいえ	いいえ	はい	いいえ
サイトマップ	いいえ	いいえ	いいえ	はい	いいえ

例:

```
googleSearchConsole_read = glueContext.create_dynamic_frame.from_options(
    connection_type="googlesearchconsole",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "entityName",
        "API_VERSION": "v3"
    }
```

Google 検索コンソールのエンティティとフィールドの詳細：

Google 検索コンソールでは、サポートされているエンティティのメタデータを動的に取得するためのエンドポイントが提供されます。したがって、オペレータのサポートはデータ型レベルでキャプチャされます。

エンティティ	フィールド	データ型	サポートされている演算子	メモ
[分析] を選択する	キー	リスト	該当なし	
	clicks	倍精度	該当なし	
	IMPRESSION	倍精度	該当なし	
	ctr	BigDecimal	該当なし	BigDecimal データ型の場合、値「0」は「0E-18」の形式になります。
	position	倍精度	該当なし	
	start_end_date	日付	BETWEEN	`start_end_date` のデフォルト値は、<現在の日付から 30 日前> と <昨日: つまり、現在

エンティティ	フィールド	データ型	サポートされている演算子	メモ
				の日付から 1 日前> の間 注: UTC 日付値を渡すことを想定しています。 例: start_end_date が '2022-01-01' と '2024-09-09' の間
	country	String	EQUAL_TO、NOT_EQUAL_TO、CONTAINS	有効な値は「IND」、「CAN」などです。
	type	String	EQUAL_TO、NOT_EQUAL_TO	有効な値は、「discover」、「googleNews」、「news」、「image」、「video」、「web」です。
	searchAppearance	String	EQUAL_TO、NOT_EQUAL_TO、CONTAINS	有効な値のリストについては、「検索表示」を参照してください。

エンティティ	フィールド	データ型	サポートされている演算子	メモ
	デバイス	String	EQUAL_TO、NOT_EQUAL_TO、CONTAINS	有効な値は「DESKTOP」、「MOBILE」、「TABLET」です
	ディメンション	String	EQUAL_TO	有効な値は「国」、「デバイス」です
	page	String	EQUAL_TO、NOT_EQUAL_TO、CONTAINS	
	query	String	EQUAL_TO、NOT_EQUAL_TO、CONTAINS	
	dataState	String	EQUAL_TO	有効な値は「all」と「final」です
サイト	siteUrl	String	該当なし	
	permissionLevel	String	該当なし	
サイトマップ	パス	String	該当なし	
	type	String	該当なし	
	lastSubmitted	DateTime	該当なし	
	isPending	ブール値	該当なし	

エンティティ	フィールド	データ型	サポートされている演算子	メモ
	isSitemapsIndex	ブール値	該当なし	
	lastDownloaded	DateTime	該当なし	
	warnings	Long	該当なし	
	エラー	Long	該当なし	
	コンテンツ	リスト	該当なし	

> **Note**
>
> フィルターの有効な値の更新リストについては、「Google 検索コンソール」の API ドキュメントを参照してください。
> フィールド `start_end_date` は `start_date` と `end_date` の組み合わせです。

パーティショニングクエリ

フィルターベースのパーティショニングとレコードベースのパーティション分割はサポートされていません。

Google 検索コンソールの接続オプション

Google 検索コンソールの接続オプションは次のとおりです。

- `ENTITY_NAME`(文字列) – (必須) 読み取りに使用されます。Google 検索コンソールのオブジェクトの名前。
- `API_VERSION`(文字列) – (必須) 読み取りに使用されます。使用する Google 検索コンソールの Rest API バージョン。
- `SELECTED_FIELDS`(List<String>) – Default: empty(SELECT *). 読み込みに使用されます。オブジェクトに選択する列。
- `FILTER_PREDICATE`(文字列) – デフォルト:「start_end_date が <現在の日付から 30 日前> と <昨日: つまり、現在の日付から 1 日前> の間」。読み込みに使用されます。Spark SQL 形式である必要があります。

- QUERY(文字列) – デフォルト:「start_end_date が <現在の日付から 30 日前> と <昨日: つまり、現在の日付から 1 日前> の間」。読み取りに使用されます。完全な Spark SQL クエリ。
- INSTANCE_URL(文字列) – 読み取りに使用されます。有効な Google 検索コンソールインスタンス URL。

Google 検索コンソールの制限

Google 検索コンソールの制限事項または注意事項は次のとおりです。

- Google 検索コンソールでは、API の使用制限が適用されます。詳細については、「使用制限」を参照してください。
- Search Analytics エンティティにフィルタが渡されない場合、API は、指定されたデフォルトの日付範囲内のサイト全体のすべてのクリック、インプレッション、CTR、およびその他のデータを合計し、単一のレコードとして表示します。
- データを小さなセグメントに分割するには、クエリにディメンションを導入する必要があります。ディメンションは、データをセグメント化する方法を API に伝えます。
  - たとえば、filterPredicate: dimensions="country" を追加すると、指定した期間中にサイトがトラフィックを受信した国ごとに 1 つのレコードを取得します。
  - 複数のディメンションを渡す例: filterPredicate: dimensions="country" AND dimensions="device" AND dimensions="page"。この場合、これら 3 つのディメンションの一意の組み合わせごとに 1 行がレスポンスに表示されます。
- デフォルト値は、start_end_date および dataState フィールドに設定されます。

フィールド名	デフォルトのフィルター式	デフォルト値を上書きする式
start_end_date	start_end_date が <現在の日付から 30 日前> と <昨日: つまり、現在の日付から 1 日前> の間	start_end_date が「2024-01-01」と「2024-05-05」の間
dateState	dataState="all"	dataState="final"

Google Sheets への接続

Google Sheets は、大量のデータを整理したり、カスタムレポートを作成したり、計算を自動化したり、他のユーザーとコラボレーションしたりできるオンラインスプレッドシートソフトウェアです。Google Sheets ユーザーの場合は、Google Sheets アカウント AWS Glue に接続できます。その後、ETLジョブのデータソースとして Google Sheets を使用できます。これらのジョブを実行して、Google Sheets と AWS サービス、またはその他のサポートされているアプリケーション間でデータを転送します。

トピック

- AWS Glue Google Sheets のサポート
- 接続を作成および使用するための API オペレーションを含むポリシー
- Google Sheets の設定
- Google Sheet 接続の設定
- Google Sheets エンティティからの読み取り
- Google Sheets 接続オプション
- Google Sheets の承認コードOAuthフローを設定する
- Google Sheets コネクタの制限事項

AWS Glue Google Sheets のサポート

AWS Glue は、次のように Google Sheets をサポートしています。

ソースとしてサポートされていますか?

はい。ジョブを使用して AWS Glue ETL、Google Sheets からデータをクエリできます。

ターゲットとしてサポートされているかどうか

いいえ。

サポートされている Google Sheets APIバージョン

Google Sheets API v4 と Google Drive API v3

接続を作成および使用するための API オペレーションを含むポリシー

次のサンプル ポリシーで、接続の作成と使用に必要な AWS の権限について説明します。新しいロールを作成する場合は、以下を含むポリシーを作成します。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:ListConnectionTypes",
                "glue:DescribeConnectionType",
                "glue:RefreshOAuth2Tokens",
                "glue:ListEntities",
                "glue:DescribeEntity"
            ],
            "Resource": "*"
        }
    ]
}
```

また、次のマネージド IAM ポリシーを使用して、アクセスを許可できます。

- AWSGlueServiceRole – さまざまな AWS Glue プロセスを実行するために必要なリソースへのアクセス権をユーザーに代わって付与します。これらのリソースには AWS Glue 、Amazon S3、IAM、CloudWatch Logs、および Amazon EC2 が含まれます。このポリシーで指定されたリソースの命名規則に従った場合、AWS Glue プロセスは必要なアクセス権限を使用できます。このポリシーは、通常、クローラ、ジョブ、開発エンドポイントを定義するときに指定されたロールにアタッチされます。
- AWSGlueConsoleFullAccess — ポリシーがアタッチされているアイデンティティが AWS マネージメントコンソールを使用するときは、AWS Glue リソースへのフルアクセスを許可します。このポリシーで指定されたリソースの命名規則に従った場合、ユーザーは完全なコンソール機能を使用できます。このポリシーは、通常 AWS Glue コンソールのユーザーにアタッチされています。

Google Sheets の設定

AWS Glue を使用して Google Sheets から転送する前に、次の要件を満たす必要があります。

最小要件

- E メールとパスワードを持つ Google Sheets アカウントがあります。
- Google Sheets アカウントへのアクセスが有効になっていますAPI。Google Sheets のすべての使用APIは、追加料金なしで利用できます。

- Google Sheets アカウントでは、接続されたアプリケーションをインストールできます。この機能にアクセスできない場合は、Google Sheets 管理者にお問い合わせください。

これらの要件を満たすと、Google Sheets アカウント AWS Glue に接続できます。

Google Sheet 接続の設定

Google Sheet 接続を設定するには：

1. AWS Secrets Manager で、次の詳細を含むシークレットを作成します。
   a. AuthorizationCode グラントタイプの場合:
      - カスタマーマネージド接続アプリケーションの場合 – シークレットには、USER_MANAGED_CLIENT_APPLICATION_CLIENT_SECRET を使用して接続されたアプリケーションのコンシューマーシークレットをキーとして含める必要があります。
2. AWS Glue Glue Studio で、以下の手順に従って [データ接続] の下に接続を作成します。
   a. [データソース] を選択する際に、[Google Sheets] を選択します。
   b. Google Sheet 環境を指定します。
      i. AWS Glue でこの接続に使用する secretName を選択して、トークンを配置します。
      ii. ネットワークを使用する場合は、ネットワークオプションを選択します。
3. AWS Glue ジョブに関連付けられている IAM ロールに secretName を読み取るアクセス許可を付与します。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:DescribeSecret",
                "secretsmanager:GetSecretValue",
                "secretsmanager:PutSecretValue",
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterface",
                "ec2:DeleteNetworkInterface",
            ],
            "Resource": "*"
        }
    ]
```

```
}
```

AUTHORIZATION_CODE のグラントタイプ

このグラントタイプは、ユーザーを認証するためにサードパーティーの認証サーバーにユーザーをリダイレクトすることから、「three-legged」の OAuth と見なされます。AWS Glue コンソール経由で接続を作成するときに使用されます。AWS Glue コンソールでは、ユーザーが Google Sheets にリダイレクトされます。ここではユーザーがログインし、Google Sheets インスタンスにアクセスするためのリクエストされた権限を AWS Glue に許可する必要があります。

ユーザーは、Google Sheets で独自の接続アプリを作成し、AWS Glue コンソールを介して接続を作成するときに独自のクライアント ID とクライアントシークレットを指定することを選択できます。このシナリオでは、引き続き Google Sheets にリダイレクトされてログインし、リソースへアクセスするために AWS Glue を承認します。

このグラントタイプは、更新トークンとアクセストークンになります。アクセストークンの有効期間は短く、更新トークンを使用してユーザーとやり取りすることなく自動的に更新される場合があります。

詳細については、「OAuth 認証コードフローでの接続されたアプリケーションの作成に関する Google Sheets の公開ドキュメント」を参照してください。

Google Sheets エンティティからの読み取り

前提条件

- 読み取り元の Google スプレッドシート。スプレッドシートの SpreadSheet ID と tabName が必要です。

Google Sheets エンティティとフィールドの詳細

エンティティ	データタイプ	サポートされる演算子
スプレッドシート	String	該当なし (フィルタリングはサポートされていません)

例

```
googleSheets_read = glueContext.create_dynamic_frame.from_options(
    connection_type="googlesheets",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "{SpreadSheetID}#{SheetTabName}",
        "API_VERSION": "v4"
    }
```

パーティショニングクエリ

レコードベースのパーティション分割の場合のみ、Spark で同時実行を使用する場合は、追加の Spark オプションとして NUM_PARTITIONS を指定できます。これらのパラメータを使用すると、元のクエリは spark タスクで同時に実行できるサブクエリの NUM_PARTITIONS の数に分割されます。

NUM_PARTITIONS を使用した例

```
googlesheets_read = glueContext.create_dynamic_frame.from_options(
    connection_type="googlesheets",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "{SpreadSheetID}#{SheetTabName}",
        "API_VERSION": "v4",
        "NUM_PARTITIONS": "10"
    }
```

Google Sheets 接続オプション

Google Sheets の接続オプションは次のとおりです。

- ENTITY_NAME (文字列) - (必須) 読み取りに使用されます。Google Sheets sheetTabNameの SpreadSheet IDと 。例えば、{SpreadSheetID}#{SheetTabName} などです。
- API_VERSION (文字列) - (必須) 読み取りに使用されます。使用する Google Sheets Rest APIバージョン。
- SELECTED_FIELDS (リスト < 文字列>) - デフォルト: empty(SELECT *)。読み込みに使用されます。オブジェクトに選択する列。
- FILTER_PREDICATE(文字列) - デフォルト: 空 読み込みに使用されます。Spark SQL形式である必要があります。
- QUERY(文字列) - デフォルト: 空 読み込みに使用されます。フル Spark SQLクエリ。

- NUM_PARTITIONS(整数) - デフォルト: 1。読み込みに使用されます。読み取り用のパーティションの数。

Google Sheets の承認コードOAuthフローを設定する

前提条件

- Google Sheets アプリを使用するためにサインインできる Google アカウント。Google アカウントでは、Google Sheets には転送するデータが含まれています。
- Google Cloud Platform アカウントと Google Cloud プロジェクト。詳細については、「Google Cloud Project の作成」を参照してください。

Google アカウントを設定し、2.0 OAuth 認証情報を取得するには：

1. Google Cloud プロジェクトのセットアップが完了したら、プロジェクトAPIsで Google Sheets APIと Google Drive を有効にします。有効にする手順については、Google Cloud Platform のAPI コンソールヘルプAPIsの「有効と無効」を参照してください。
2. 次に、外部ユーザーのOAuth同意画面を設定します。OAuth 同意画面の詳細については、「Google Cloud Platform コンソールヘルプ」のOAuth「同意画面の設定」を参照してください。
3. OAuth 同意画面で、次のスコープを追加します。

   - Google Sheets API の読み取り専用スコープ
   - Google Drive API の読み取り専用スコープ

   これらのスコープの詳細については、Google Identity OAuth ドキュメントAPIsの「2.0 Scopes for Google」を参照してください。
4. 2.0 OAuth クライアント ID とシークレットを生成します。このクライアント ID を作成する手順については、Google Cloud Platform OAuth コンソールヘルプの「Setting up 2.0」を参照してください。

   2.0 OAuth クライアント ID には、1 つ以上の承認されたリダイレクト が必要ですURLs。

   Redirect URLsの形式は次のとおりです。

   - https://<aws-region>.console.aws.amazon。com/gluestudio/oauth

5. 2.0 クライアント ID の設定からクライアント ID OAuth とクライアントシークレットを書き留めます。

Google Sheets コネクタの制限事項

Google Sheets コネクタの制限事項は次のとおりです:

- Google Sheets コネクタはフィルターをサポートしていません。したがって、フィルターベースのパーティション分割はサポートされません。
- レコードベースのパーティション分割には、SAAS によって正確なレコード数を返すプロビジョニングはありません。その結果、空のレコードを持つファイルが作成されるシナリオが発生する可能性があります。
- Google Sheets コネクタはフィルターベースのパーティション分割をサポートしていないため、`partitionField`、`lowerbound`、および `upperbound` は有効な接続オプションではありません。これらのオプションが指定されている場合、AWS Glue ジョブは失敗することが予想されます。
- データ処理の問題を回避するには、シートの最初の行をヘッダー行として指定することが重要です。
  - ヘッダー行が提供されない場合、シートの最初の行が空でデータが含まれている場合、ヘッダー行は `Unnamed:1`、`Unnamed:2`、`Unnamed:3`...に置き換えられます。
  - ヘッダー行が指定されている場合、空の列名は `Unnamed:<number of column>` に置き換えられます。例えば、ヘッダー行が `['ColumnName1', 'ColumnName2', '', '', 'ColumnName5', 'ColumnName6']` の場合、`['ColumnName1', 'ColumnName2', 'Unnamed:3', 'Unnamed:4', 'ColumnName5', 'ColumnName6'].` になります。
- Google Sheets コネクタは増分転送をサポートしていません。
- Google Sheets コネクタは、文字列データ型のみをサポートしています。
- シート内の重複ヘッダーは、数値サフィックスで繰り返し名前が変更されます。ユーザーによって指定されたヘッダー名は、重複するヘッダーの名前を変更する際に優先されます。たとえば、ヘッダー行が["Name", "", "Name", null, "Unnamed:6", ""] の場合、["Name", "Unnamed:2", "Name1", "Unnamed:4", "Unnamed:6", "Unnamed:61"] に変更されます。
- Google Sheets コネクタは tabName のスペースをサポートしていません。
- フォルダ名に次の特殊文字を含めることはできません:
  - #
  - /

への接続 HubSpot

HubSpotのCRMプラットフォームには、マーケティング、販売、コンテンツ管理、カスタマーサービスに必要なすべてのツールと統合があります。

- Marketing Hub - トラフィックを増やし、より多くの訪問者を変換し、完全なインバウンドマーケティングキャンペーンを大規模に実行するのに役立つマーケティングソフトウェア。
- Sales Hub - 見込み客に関するより深いインサイトを得て、タスクを自動化し、より多くの取引をより迅速に締結するのに役立つ販売CRMソフトウェア。
- Service Hub - 顧客とつながり、期待を上回り、ビジネスを成長させる推進者に変えるのに役立つカスタマーサービスソフトウェア。
- Operations Hub - アプリケーションを同期し、顧客データをクリーンアップして作成し、プロセスを自動化するオペレーションソフトウェア。これにより、すべてのシステムとチームが連携しやすくなります。

トピック

- HubSpot の AWS Glue サポート
- 接続を作成および使用するためのAPIオペレーションを含むポリシー
- HubSpot の設定
- HubSpot 接続の設定
- HubSpot エンティティからの読み取り
- HubSpot 接続オプション
- HubSpot コネクタの制限と注意事項

HubSpot の AWS Glue サポート

AWS Glue は、次のように HubSpot をサポートします。

ソースとしてサポートされていますか?

はい – Sync と Async。AWS Glue ETL ジョブを使用して、HubSpot からデータをクエリできます。

ターゲットとしてサポートされていますか?

いいえ。

サポートされている HubSpot API バージョン

次の HubSpot API バージョンがサポートされています。

- v1
- v2
- v3
- v4

バージョン固有ごとのエンティティサポートについては、「Sync ソースに対応するエンティティ」および「Async ソースに対応するエンティティ」を参照してください。

接続を作成および使用するためのAPIオペレーションを含むポリシー

次のサンプルポリシーでは、接続の作成と使用に必要な AWS IAMアクセス許可について説明します。新しいロールを作成する場合は、以下を含むポリシーを作成します。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:ListConnectionTypes",
                "glue:DescribeConnectionType",
                "glue:RefreshOAuth2Tokens"
                "glue:ListEntities",
                "glue:DescribeEntity"
            ],
            "Resource": "*"
        }
    ]
}
```

上記の方法を使用しない場合は、次の管理IAMポリシーを使用します。

- AWSGlueServiceRole – さまざまな AWS Glue プロセスがユーザーに代わって実行するために必要なリソースへのアクセスを許可します。これらのリソースには AWS Glue、、Amazon S3、IAM、 CloudWatch Logs、Amazon が含まれますEC2。このポリシーで指定されたリソースの命名規則に従うと、 AWS Glue プロセスには必要なアクセス許可があります。このポリシー

は、通常、クローラ、ジョブ、開発エンドポイントを定義するときに指定されたロールにアタッチされます。

- AWSGlueConsoleFullAccess — ポリシーがアタッチされている ID が AWS マネジメントコンソールを使用する場合、 AWS Glue リソースへのフルアクセスを許可します。このポリシーで指定されたリソースの命名規則に従った場合、ユーザーは完全なコンソール機能を使用できます。このポリシーは通常、 AWS Glue コンソールのユーザーにアタッチされます。

HubSpot の設定

AWS Glue を使用して HubSpot からデータを転送できるようにするには、次の要件を満たす必要があります。

最小要件

以下に、最小要件を示します。

- HubSpot アカウントを持っている。詳細については、「HubSpot アカウントの作成」を参照してください。
- HubSpot アカウントで API アクセスが有効になっている。
- HubSpot デベロッパーアカウントの下には、AWS Glue がアカウントに対して認可呼び出しを行うときにデータに安全にアクセスするために使用するクライアント認証情報を提供するアプリケーションが必要です。詳細については、「HubSpot 開発者アプリの作成」を参照してください。

これらの要件を満たすと、HubSpot アカウントに AWS Glue を接続する準備が整います。一般的な接続では、HubSpot で他に何もする必要はありません。

HubSpot アカウントの作成

HubSpot アカウントを作成するには:

1. HubSpot CRM SignUp URL に移動します。
2. E メールアドレスを入力し、[E メールを確認] を選択します (または、Google、Microsoft、または Apple アカウントでサインアップすることもできます)。
3. 受信トレイで HubSpot の検証コードを確認してください。
4. 6 桁の検証コードを入力し、[次へ] をクリックします。
5. パスワードを入力し、[次へ] をクリックします。

6. 名と姓を入力し、[次へ] をクリックするか、[Google でサインアップ] リンクを使用してサインアップします。
7. 業界を入力し、[次へ] をクリックします。
8. 職務を入力し、[次へ] をクリックします。
9. 会社名を入力し、[次へ] をクリックします。
10. 会社の規模 (社内の従業員数) を選択し、[次へ] をクリックします。
11. 会社のウェブサイトを入力し、[次へ] をクリックします。
12. データをホストする場所 (米国または欧州) を選択し、[アカウントの作成] をクリックします。
13. アカウント作成の目的を選択し、[次へ] をクリックします。
14. [Google アカウントを接続する] を選択するか、自分で連絡先を追加して、連絡先を HubSpot アカウントにリンクします。
15. [Google アカウントを接続する] オプションを選択した場合は、Google アカウントにログインして連絡先をリンクし、HubSpot アカウントの使用を開始します。

HubSpot 開発者アプリの作成

アプリ開発者アカウントは、アプリ、統合、開発者テストアカウントの作成と管理を目的としています。また、App Marketplace 出品を作成および管理できる場所でもあります。ただし、アプリ開発者アカウントおよび関連するテストアカウントは、標準の HubSpot アカウントに接続されません。別の HubSpot アカウントとの間でデータまたはアセットを同期することはできません。[クライアント ID] と [クライアントシークレット] を取得するには、デベロッパーアカウントを作成します。

1. https://developers.hubspot.com/ に移動します。
2. [開発者アカウントの作成] を選択し、下にスクロールします。
3. アプリ開発者アカウント、プライベートアプリアカウント、または CMS 開発者サンドボックスアカウントを作成するかどうか尋ねられます。[アプリ開発者アカウントの作成] を選択します。
4. HubSpot でアカウントを作成済みであるため、[このユーザーで続行] を選択できます。
5. [サインアップを開始] をクリックします。
6. 職務を入力し、[次へ] をクリックします。
7. デベロッパーアカウントに名前を付け、[次へ]、[スキップ] の順にクリックします。
8. [Create App (アプリの作成)] を選択します。
9. アプリを作成したら、[認証] を選択します。
10. [認証] の下にある [クライアント ID] と [クライアントシークレット] を書き留めます。

11.リージョン固有の [リダイレクト URL] を https://<aws-region>.console.aws.amazon.com/gluestudio/oauth として追加します。例えば、us-east-1 リージョンの場合、https://us-east-1.console.aws.amazon.com/gluestudio/oauth を追加します。

12.下にスクロールしてスコープを見つけます。「CRM」と「標準」の見出しの下に、選択する必要がある 2 種類のスコープがあります。

13.以下のスコープを追加します。

```
content
automation
oauth
crm.objects.owners.read
forms
tickets
crm.objects.contacts.write
e-commerce
crm.schemas.custom.read
crm.objects.custom.read
sales-email-read
crm.objects.custom.write
crm.objects.companies.write
crm.lists.write
crm.objects.companies.read
crm.lists.read
crm.objects.deals.read
crm.objects.deals.write
crm.objects.contacts.read
```

14.[保存] をクリックすると、開発アカウントを使用する準備が整います。

15.上にスクロールして [クライアント ID] を見つけます。

16.同じページで、[表示] をクリックして [クライアントシークレット] を取得します。

HubSpot 開発者テストアカウントの作成

アプリ開発者アカウント内で、実際の HubSpot データに影響を与えることなく、アプリと統合をテストする開発者テストアカウントを作成できます。開発者テストアカウントは本番稼働用アカウントをミラーリングするのではなく、Marketing、Sales、Service、CMS、Operations Hub のエンタープライズバージョンの 90 日間トライアルにアクセスでき、ほとんどの HubSpot ツールと API をテストできます。

1. [ホーム] をクリックします。

2. [テストアカウントの作成] をクリックします。
3. [アプリテストアカウントの作成] をクリックします。
4. 新しいウィンドウが表示されます。アプリのテストアカウント名を入力し、[作成] をクリックします。

   これで、アプリテストアカウントが作成されました。

> **Note**
>
> デベロッパーアカウントは API 統合などの開発アクティビティに関連しており、アプリテストアカウントはデベロッパーアカウントによって作成またはプルされたデータを表示するために使用されます。

HubSpot 接続の設定

HubSpot は、OAuth2 の AUTHORIZATION_CODE のグラントタイプをサポートしています。

- このグラントタイプは、ユーザーを認証するためにサードパーティーの認証サーバーにユーザーをリダイレクトすることから、「three-legged」の OAuth と見なされます。AWS Glue コンソール経由で接続を作成するときに使用されます。接続を作成するユーザーは、HubSpot クライアントアプリケーションのクライアント ID やクライアントシークレットなどの OAuth 関連情報を指定する必要があります。AWS Glue コンソールでは、ユーザーが HubSpot にリダイレクトされます。ここではユーザーがログインし、HubSpot インスタンスにアクセスするためのリクエストされた権限を AWS Glue に許可する必要があります。
- ユーザーは、HubSpot で独自の接続アプリを作成し、AWS Glue コンソールを介して接続を作成するときに独自のクライアント ID とクライアントシークレットを指定することを選択できます。このシナリオでは、引き続き HubSpot にリダイレクトされてログインし、リソースへアクセスするために AWS Glue を承認します。
- このグラントタイプは、更新トークンとアクセストークンになります。アクセストークンの有効期間は短く、更新トークンを使用してユーザーとやり取りすることなく自動的に更新される場合があります。
- 認可コード OAuth フロー用の接続アプリケーションの作成に関する HubSpot の公開ドキュメントについては、「パブリックアプリケーション」を参照してください。

HubSpot 接続を設定するには:

1. AWS Secrets Manager で、次の詳細を含むシークレットを作成します。
   a. カスタマーマネージド接続アプリケーションの場合、シークレットには、USER_MANAGED_CLIENT_APPLICATION_CLIENT_SECRET を使用して接続されたアプリケーションのコンシューマーシークレットをキーとして含める必要があります。
   b. 注: AWS Glue で接続ごとにシークレットを作成する必要があります。
2. AWS Glue Glue Studio で、以下の手順に従って [データ接続] の下に接続を作成します。
   a. [接続タイプ] を選択する際に、[HubSpot] を選択します。
   b. HubSpot 環境を指定します。
   c. 次のアクションを実行でき、AWS Glue がその権限を持つ AWS IAM ロールを選択します。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:DescribeSecret",
                "secretsmanager:GetSecretValue",
                "secretsmanager:PutSecretValue",
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterface",
                "ec2:DeleteNetworkInterface",
            ],
            "Resource": "*"
        }
    ]
}
```

   d. AWS Glue でこの接続に使用する secretName を選択して、トークンを配置します。
   e. ネットワークを使用する場合は、ネットワークオプションを選択します。
3. AWS Glue ジョブに関連付けられている IAM ロールに secretName を読み取るアクセス許可を付与します。
4. AWS Glue ジョブ設定で、[追加のネットワーク接続] として connectionName を指定します。

HubSpot エンティティからの読み取り

前提条件

読み取り元の HubSpot オブジェクト。連絡先またはタスクなどのオブジェクト名が必要です。次の表に、Sync ソースでサポートされているエンティティを示します。

Sync ソースに対応するエンティティ

エンティティ	API バージョン	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティショニングをサポートする
キャンペーン	v1	いいえ	あり	なし	あり	いいえ
Companies	v3	はい	あり	あり	あり	はい
問い合わせ	v3	はい	あり	あり	あり	はい
Contact Lists	v1	いいえ	あり	なし	あり	いいえ
Deals	v3	はい	あり	あり	あり	はい
CRM Pipeline (Deal Pipelines)	v1	いいえ	いいえ	いいえ	あり	いいえ
E メールイベント	v1	いいえ	あり	なし	あり	いいえ
呼び出し	v3	はい	あり	あり	あり	はい
メモ	v3	はい	あり	あり	あり	はい
Emails	v3	はい	あり	あり	あり	はい
会議	v3	はい	あり	あり	あり	はい
タスク	v3	はい	あり	あり	あり	はい

エンティティ	API バージョン	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティショニングをサポートする
Postal Mails	v3	はい	あり	あり	あり	はい
Custom Objects	v3	はい	あり	あり	あり	はい
Forms	v2	いいえ	いいえ	いいえ	あり	いいえ
Owners	v3	いいえ	あり	なし	あり	いいえ
製品	v3	はい	あり	あり	あり	はい
Tickets	v3	はい	あり	あり	あり	はい
ワークフロー	v3	いいえ	いいえ	いいえ	あり	いいえ
関連付け	v4	はい	いいえ	いいえ	あり	いいえ
Associations Labels	v4	いいえ	いいえ	いいえ	あり	いいえ

例:

```
hubspot_read = glueContext.create_dynamic_frame.from_options(
    connection_type="hubspot",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "contact",
        "API_VERSION": "v3"
    }
```

Async ソースに対応するエンティティ

エンティティ	API バージョン	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティショニングをサポートする
Companies	v3	はい	なし	はい	はい	いいえ
問い合わせ	v3	はい	なし	はい	はい	いいえ
Deals	v3	はい	なし	はい	はい	いいえ
呼び出し	v3	はい	なし	はい	はい	いいえ
メモ	v3	はい	なし	はい	はい	いいえ
Emails	v3	はい	なし	はい	はい	いいえ
会議	v3	はい	なし	はい	はい	いいえ
タスク	v3	はい	なし	はい	はい	いいえ
Postal Mails	v3	はい	なし	はい	はい	いいえ
Custom Objects	v3	はい	なし	はい	はい	いいえ
製品	v3	はい	なし	はい	はい	いいえ
Tickets	v3	はい	なし	はい	はい	いいえ

例:

```
hubspot_read = glueContext.create_dynamic_frame.from_options(
    connection_type="hubspot",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "contact",
```

```
        "API_VERSION": "v3",
        "TRANSFER_MODE": "ASYNC"
    }
```

HubSpot エンティティとフィールドの詳細:

HubSpot API v4:

エンティティ	API バージョン	フィールド	データ型	サポートされている演算子
Association Label	v4	category	String	該当なし
		typeId	整数	該当なし
		ラベル	String	該当なし
関連付け		送信元	Struct	該当なし
		id	String	"="
		次のように変更します。	リスト	該当なし

> **Note**
>
> `Associations` オブジェクトの場合、2 つのオブジェクト間の関連付けを取得するには、AWS Glue ジョブの作成時に必須フィルターを介して「from Id」 (最初のオブジェクトの ID) を指定する必要があります。この場合、IDs から複数の の関連付けを取得するには、`where` 句に複数の IDsを指定する必要があります。例えば、問い合わせ ID 「1」と「151」の `Associations`を取得するには、フィルターを `where id=1 AND id=151` として指定する必要があります。

HubSpot API v3:

エンティティ	フィールド	データ型	サポートされている演算子
所有者	firstName	String	該当なし
	lastName	String	該当なし
	createdAt	DateTime	該当なし
	archived	ブール値	該当なし
	teams	リスト	該当なし
	id	String	該当なし
	userId	整数	該当なし
	email	String	該当なし
	updatedAt	DateTime	該当なし
ワークフロー	名前	String	該当なし
	id	整数	該当なし
	type	String	該当なし
	有効	ブール値	該当なし
	insertedAt	Long	該当なし
	updatedAt	Long	該当なし
	contactListIds	Struct	該当なし
	personaTagIds	リスト	該当なし

次のエンティティについて、HubSpot はメタデータを動的に取得するエンドポイントを提供するため、オペレータのサポートは各エンティティのデータ型レベルでキャプチャされます。

> **Note**
>
> DML_STATUS は、実行時にすべてのレコードに追加された仮想フィールドで、Sync モードでのそのステータス (CREATED/UPDATED) を決定します。CONTAINS/LIKE 演算子は Async モードではサポートされていません。

エンティティ	データ型	サポートされている演算子
連絡先	整数	"=, !=, <, >, >=, <="
	Long	"=, !=, <, >, >=, <="
	String	"=, !=, LIKE"
	日付	該当なし
	DateTime	"between"
	ブール値	"="
	リスト	該当なし
	Struct	該当なし
Company	整数	"=, !=, <, >, >=, <="
	Long	"=, !=, <, >, >=, <="
	String	"=, !=, LIKE"
	日付	該当なし
	DateTime	"between"
	ブール値	"="
	リスト	該当なし
	Struct	該当なし

エンティティ	データ型	サポートされている演算子
Deal	整数	"=, !=, <, >, >=, <="
	Long	"=, !=, <, >, >=, <="
	String	"=, !=, LIKE"
	日付	該当なし
	DateTime	"between"
	ブール値	"="
	リスト	該当なし
	Struct	該当なし
Ticket	整数	"=, !=, <, >, >=, <="
	Long	"=, !=, <, >, >=, <="
	String	"=, !=, LIKE"
	日付	該当なし
	DateTime	"between"
	ブール値	"="
	リスト	該当なし
	Struct	該当なし
製品	整数	"=, !=, <, >, >=, <="
	Long	"=, !=, <, >, >=, <="
	String	"=, !=, LIKE"
	日付	該当なし

エンティティ	データ型	サポートされている演算子
	DateTime	"between"
	ブール値	"="
	リスト	該当なし
	Struct	該当なし
Custom Object	整数	"=, !=, <, >, >=, <="
	Long	"=, !=, <, >, >=, <="
	String	"=, !=, LIKE"
	日付	該当なし
	DateTime	"between"
	ブール値	"="
	リスト	該当なし
	Struct	該当なし
電話	整数	"=, !=, <, >, >=, <="
	Long	"=, !=, <, >, >=, <="
	String	"=, !=, LIKE"
	日付	該当なし
	DateTime	"between"
	ブール値	"="
	リスト	該当なし
	Struct	該当なし

エンティティ	データ型	サポートされている演算子
E メール	整数	"=, !=, <, >, >=, <="
	Long	"=, !=, <, >, >=, <="
	String	"=, !=, LIKE"
	日付	該当なし
	DateTime	"between"
	ブール値	"="
	リスト	該当なし
	Struct	該当なし
会議	整数	"=, !=, <, >, >=, <="
	Long	"=, !=, <, >, >=, <="
	String	"=, !=, LIKE"
	日付	該当なし
	DateTime	"between"
	ブール値	"="
	リスト	該当なし
	Struct	該当なし
メモ	整数	"=, !=, <, >, >=, <="
	Long	"=, !=, <, >, >=, <="
	String	"=, !=, LIKE"
	日付	該当なし

エンティティ	データ型	サポートされている演算子
	DateTime	"between"
	ブール値	"="
	リスト	該当なし
	Struct	該当なし
タスク	整数	"=, !=, <, >, >=, <="
	Long	"=, !=, <, >, >=, <="
	String	"=, !=, LIKE"
	日付	該当なし
	DateTime	"between"
	ブール値	"="
	リスト	該当なし
	Struct	該当なし
Postal Mail	整数	"=, !=, <, >, >=, <="
	Long	"=, !=, <, >, >=, <="
	String	"=, !=, LIKE"
	日付	該当なし
	DateTime	"between"
	ブール値	"="
	リスト	該当なし
	Struct	該当なし

HubSpot API v2:

エンティティ	フィールド	データ型	サポートされている演算子
フォーム	portalId	整数	該当なし
	guid	String	該当なし
	名前	String	該当なし
	method	String	該当なし
	cssClass	String	該当なし
	リダイレクト	String	該当なし
	submitText	String	該当なし
	notifyRecipients	String	該当なし
	createdAt	Long	該当なし
	updatedAt	Long	該当なし
	ignoreCurrentValues	ブール値	該当なし
	deletable	ブール値	該当なし
	inlineMessage	ブール値	該当なし
	captchaEnabled	ブール値	該当なし
	cloneable	ブール値	該当なし
	formFieldGroups	リスト	該当なし
	editable	ブール値	該当なし
	deletedAt	整数	該当なし
	themeName	String	該当なし

エンティティ	フィールド	データ型	サポートされている演算子
	parentId	整数	該当なし
	style	String	該当なし
	isPublished	ブール値	該当なし
	publishAt	整数	該当なし
	unpublishAt	整数	該当なし
	publishedAt	整数	該当なし
	kickbackEmailWorkflowId	String	該当なし
	kickbackEmailsJson	整数	該当なし
	customUid	String	該当なし
	createMarketableContact	ブール値	該当なし
	editVersion	整数	該当なし
	thankYouMessageJson	String	該当なし
	themeColor	String	該当なし
	alwaysCreateNewCompany	ブール値	該当なし
	internalUpdatedAt	Long	該当なし
	ビジネスユニット ID	整数	該当なし
	portableKey	String	該当なし

エンティティ	フィールド	データ型	サポートされている演算子
	paymentSessionTemplateIds	リスト	該当なし
	selectedExternalOptions	リスト	該当なし

HubSpot API v1:

エンティティ	フィールド	データ型	サポートされている演算子
キャンペーン	id	整数	該当なし
	appId	整数	該当なし
	appName	String	該当なし
	lastUpdatedTime	Long	該当なし
Contact_List	dynamic	ブール値	該当なし
	名前	String	該当なし
	portalId	整数	該当なし
	createdAt	Long	該当なし
	listId	整数	該当なし
	updatedAt	Long	該当なし
	ListType	String	該当なし
	フィルター	リスト	該当なし
	authorId	整数	該当なし

エンティティ	フィールド	データ型	サポートされている演算子
	metaData	Struct	該当なし
	archived	ブール値	該当なし
	ilsFilterBranch	String	該当なし
	filterIds	リスト	該当なし
	limitExempt	ブール値	該当なし
	internal	ブール値	該当なし
	readOnly	ブール値	該当なし
	parentId	整数	該当なし
Email_Event	id	String	該当なし
	type	String	該当なし
	受取人	String	該当なし
	portalId	整数	該当なし
	appId	整数	該当なし
	appName	String	該当なし
	emailCampaignId	Long	該当なし
	attempt	整数	該当なし
	作成済み	Long	該当なし
	sentBy	Struct	該当なし
	smtpId	String	該当なし
	レスポンス	String	該当なし

エンティティ	フィールド	データ型	サポートされている演算子
	subject	String	該当なし
	cc	リスト	該当なし
	bcc	リスト	該当なし
	replyTo	リスト	該当なし
	送信元	String	該当なし
	dropReason	String	該当なし
	dropMessage	String	該当なし
	ブラウザ	Struct	該当なし
	userAgent	String	該当なし
	duration	Long	該当なし
	location	Struct	該当なし
	filteredEvent	ブール値	該当なし
	deviceType	String	該当なし
	suppressedReason	String	該当なし
	suppressedMessage	String	該当なし
CRM_Pipeline	pipelineId	String	該当なし
	createdAt	Long	該当なし
	updatedAt	Long	該当なし
	objectType	String	該当なし
	ラベル	String	該当なし

エンティティ	フィールド	データ型	サポートされている演算子
	displayOrder	整数	該当なし
	ACTIVE	ブール値	該当なし
	ステージ	リスト	該当なし
	objectTypeId	String	該当なし
	デフォルト	ブール値	該当なし

パーティショニングクエリ

Spark で同時実行を使用する場合は、追加の Spark オプション PARTITION_FIELD、LOWER_BOUND、UPPER_BOUND、および NUM_PARTITIONS を指定できます。これらのパラメータを使用すると、元のクエリは Spark タスクで同時に実行できるサブクエリの NUM_PARTITIONS の数に分割されます。

- PARTITION_FIELD: クエリのパーティション化に使用するフィールドの名前。
- LOWER_BOUND: 選択したパーティションフィールドの包括的な下限値。

  DateTime フィールドでは、ISO 形式の値を受け入れます。

  有効な値の例は次のとおりです:

  ```
  “2024-01-01T10:00:00.115Z"
  ```
- UPPER_BOUND: 選択したパーティションフィールドの排他的上限値。
- NUM_PARTITIONS: パーティション数。

次の表に、エンティティパーティション分割フィールドのサポートの詳細を示します:

エンティティ名	パーティション分割フィールド	データ型
contact	hs_object_id	Long

エンティティ名	パーティション分割フィールド	データ型
	createdate、lastmodifieddate	DateTime
company	hs_object_id	Long
	createate、hs_lastmodifieddate	DateTime
deal	hs_object_id	Long
	createate、hs_createdate、hs_lastmodifieddate	DateTime
ticket	hs_object_id	Long
	createate、hs_lastmodifieddate	DateTime
product	hs_object_id	Long
	createate、hs_lastmodifieddate	DateTime
custom_object	hs_object_id	Long
	createate、hs_lastmodifieddate	DateTime
call	hs_object_id	Long
	createate、hs_lastmodifieddate	DateTime
email	hs_object_id	Long
	createate、hs_lastmodifieddate	DateTime
会議	hs_object_id	Long

エンティティ名	パーティション分割フィールド	データ型
	createate、hs_lastmodifieddate	DateTime
note	hs_object_id	Long
	createate、hs_lastmodifieddate	DateTime
タスク	hs_object_id	Long
	createate、hs_lastmodifieddate	DateTime
postal_mail	hs_object_id	Long
	createate、hs_lastmodifieddate	DateTime

例:

```
hubspot_read = glueContext.create_dynamic_frame.from_options(
    connection_type="hubspot",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "company",
        "API_VERSION": "v3",
        "PARTITION_FIELD": "hs_object_id"
        "LOWER_BOUND": "50"
        "UPPER_BOUND": "16726619290"
        "NUM_PARTITIONS": "10"
    }
```

HubSpot 接続オプション

HubSpot の接続オプションは次のとおりです。

- ENTITY_NAME(文字列) – (必須) 読み取りに使用されます。HubSpot のオブジェクトの名前。

- API_VERSION(文字列) – (必須) 読み取りに使用されます。使用する HubSpot Rest API バージョン。例: v1、v2、v3、v4。
- SELECTED_FIELDS(List<String>) – Default: empty(SELECT *). 読み込みに使用されます。オブジェクトに選択する列。
- FILTER_PREDICATE(文字列) – デフォルト: 空 読み込みに使用されます。Spark SQL 形式である必要があります。
- QUERY(文字列) - デフォルト: 空 読み込みに使用されます。完全な Spark SQL クエリ。
- PARTITION_FIELD(文字列) – 読み取りに使用されます。クエリをパーティション化するために使用するフィールド。
- LOWER_BOUND(文字列) - 読み取りに使用されます。選択したパーティションフィールドの包括的な下限値。
- UPPER_BOUND (文字列) - 読み取りに使用されます。選択したパーティションフィールドの排他的上限値。
- NUM_PARTITIONS(整数) – デフォルト: 1。読み込みに使用されます。読み取り用のパーティションの数。
- TRANSFER_MODE(文字列) - クエリを Async モードで実行するかどうかを示すために使用されます。

HubSpot コネクタの制限と注意事項

HubSpot コネクタの制限事項または注意事項は次のとおりです。

- 検索エンドポイントは、特定のクエリに対して合計 10,000 件の結果に制限されます。10,000 を超えるレコードを持つパーティションは、400 エラーになります。
- コネクタに関するその他の重要な制限については、「制限事項」を参照してください。
- HubSpot では、最大 3 つのフィルタリングステートメントを使用できます。
- 現在、HubSpot は標準 HubSpot オブジェクト (連絡先、会社、取引、チケットなど) とカスタムオブジェクト間の関連付けをサポートしています。
  - 無料アカウントの場合: 各オブジェクトペアの間に作成できる関連付けタイプは最大 10 個です (連絡先や会社など)。
  - スーパー管理者アカウントの場合: 各オブジェクトペアの間に作成できる関連付けタイプは最大 50 個です。
  - 詳細については、「Associations v4」および「Create and use association labels」を参照してください。

- 「引用」および「通信」オブジェクトは、現在コネクタでサポートされていないため、関連付けには存在しません。
- Async の場合、SaaS は値を昇順でのみソートします。
- `Ticket` エンティティの場合、SaaS は Async モードで `hs_object_id` フィールドを返しません。

Instagram 広告への接続

Instagram は写真共有アプリであり、ブランド、有名人、ソートリーダー、友人、家族などとつながることができます。これは、写真共有およびソーシャルネットワーキングサービスです。ユーザーは写真やショート動画を撮り、フォロワーと共有できます。Instagram 広告は、企業が料金を払うことで Instagram ユーザーに出すことができる投稿です。

トピック

- AWS Glue による Instagram 広告のサポート
- 接続を作成および使用するための API オペレーションを含むポリシー
- Instagram 広告の設定
- Instagram 広告接続の設定
- Instagram 広告エンティティからの読み取り
- Instagram 広告接続オプション
- Instagram 広告コネクタの制限と注意事項

AWS Glue による Instagram 広告のサポート

AWS Glue は、次のように Instagram 広告をサポートします:

ソースとしてサポートされていますか?

はい。AWS Glue ETL ジョブを使用して、Instagram 広告からデータをクエリできます。

ターゲットとしてサポートされているかどうか

いいえ。

サポートされている Instagram 広告の API バージョン

以下の Instagram 広告 API バージョンがサポートされています:

- v17.0
- v18.0
- v19.0
- v20.0

接続を作成および使用するための API オペレーションを含むポリシー

次のサンプル ポリシーで、接続の作成と使用に必要な AWS IAM 権限について説明します。新しいロールを作成する場合は、以下を含むポリシーを作成します。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:ListConnectionTypes",
                "glue:DescribeConnectionType",
                "glue:RefreshOAuth2Tokens"
                "glue:ListEntities",
                "glue:DescribeEntity"
            ],
            "Resource": "*"
        }
    ]
}
```

上記の方法を使用しない場合は、代わりに次のマネージド IAM ポリシーを使用します:

- AWSGlueServiceRole – さまざまな AWS Glue プロセスを実行するために必要なリソースへのアクセス権をユーザーに代わって付与します。これらのリソースには AWS Glue 、Amazon S3、IAM、CloudWatch Logs、および Amazon EC2 が含まれます。このポリシーで指定されたリソースの命名規則に従った場合、AWS Glue プロセスは必要なアクセス権限を使用できます。このポリシーは、通常、クローラ、ジョブ、開発エンドポイントを定義するときに指定されたロールにアタッチされます。
- AWSGlueConsoleFullAccess — ポリシーがアタッチされているアイデンティティが AWS マネージメントコンソールを使用するときは、AWS Glue リソースへのフルアクセスを許可します。このポリシーで指定されたリソースの命名規則に従った場合、ユーザーは完全なコンソール機能を使用できます。このポリシーは、通常 AWS Glue コンソールのユーザーにアタッチされています。

Instagram 広告の設定

AWS Glue を使用して Instagram 広告からデータを転送できるようにするには、次の要件を満たす必要があります:

最小要件

以下に、最小要件を示します。

- Instagram のスタンダードアカウントは Facebook 経由で間接的にアクセスできること。
- アクセストークンを生成するには、ユーザー認証が必要です。
- Instagram 広告の SDK コネクタは、ユーザーアクセストークン OAuth フローを実装すること。
- Instagram 広告への API リクエストの認証には OAuth2.0 を使用すること。このウェブベースの認証は、2FA 上位セットである多要素認証 (MFA) アーキテクチャに該当します。
- ユーザーは、エンドポイントにアクセスするためのアクセス許可を付与する必要があります。ユーザーのデータにアクセスするために、エンドポイント認可はアクセス許可と機能を通じて処理されます。

OAuth 2.0 認証情報の取得

インスタンスに対して認証された呼び出しを行えるように API 認証情報を取得するには、「Graph API」を参照してください。

Instagram 広告接続の設定

Instagram 広告は、OAuth2 の AUTHORIZATION_CODE グラントタイプをサポートしています。

- このグラントタイプは、ユーザーを認証するためにサードパーティーの認証サーバーにユーザーをリダイレクトすることから、three-legged　の OAuth と見なされます。AWS Glue コンソール経由で接続を作成するときに使用されます。
- ユーザーは、Instagram 広告で独自の接続アプリケーションを作成し、AWS Glue コンソールを介して接続を作成するときに独自のクライアント ID とクライアントシークレットを提供することを選択できます。このシナリオでは、引き続き Instagram 広告にリダイレクトされてログインし、リソースへアクセスするために AWS Glue を承認します。
- このグラントタイプは、アクセストークンになります。期限切れのシステムユーザートークンは、生成または更新された日付から 60 日間有効です。継続性を作成するには、開発者は 60 日以内にアクセストークンを更新する必要があります。そうしないと、アクセストークンは失われ、開発者

は API アクセスを回復するために新しいトークンを取得する必要があります。「Refresh Access Token」を参照してください。

Instagram 広告接続を設定するには:

1. AWS Glue Glue Studio で、以下の手順に従って [データ接続] の下に接続を作成します。
   a. [接続タイプ] を選択する際に、[Instagram 広告] を選択します。
   b. AWS Glue が次のアクションを引き受けて、その権限を持つことができる AWS IAM ロールを選択します。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:DescribeSecret",
                "secretsmanager:GetSecretValue",
                "secretsmanager:PutSecretValue",
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterface",
                "ec2:DeleteNetworkInterface",
            ],
            "Resource": "*"
        }
    ]
}
```

   c. ユーザーマネージドクライアントアプリケーションクライアント ID を指定します。
   d. AWS Glue でこの接続に使用する `secretName` を選択して、トークンを配置します。選択したシークレットには、接続されたアプリのクライアントシークレットである値を持つ `USER_MANAGED_CLIENT_APPLICATION_CLIENT_SECRET` キーが必要です。
   e. ネットワークを使用する場合は、ネットワークオプションを選択します。
2. AWS Glue ジョブに関連付けられている IAM ロールに `secretName` を読み取るアクセス許可を付与します。

Instagram 広告エンティティからの読み取り

前提条件

読み取り元の Instagram 広告オブジェクト。オブジェクト名が必要になります。次の表は、サポートされているエンティティを示しています。

ソースに対応するエンティティ:

エンティティ	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
Campaign	はい	はい	いいえ	はい	はい
Ad Set	はい	はい	いいえ	はい	はい
Ads	はい	はい	いいえ	はい	はい
Ad Creative	なし	あり	いいえ	あり	不可
Insights - Account	なし	あり	いいえ	あり	不可
Ad Image	はい	はい	いいえ	あり	不可
Insights - Ad	はい	はい	いいえ	はい	はい
Insights - AdSet	はい	はい	いいえ	はい	はい
Insights - Campaign	はい	はい	いいえ	はい	はい

例:

```
instagramAds_read = glueContext.create_dynamic_frame.from_options(
    connection_type="instagramads",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "entityName",
        "API_VERSION": "v20.0"
    }
```

Instagram 広告エンティティとフィールドの詳細

エンティティとフィールドの詳細については、以下を参照してください:

- Campaign
- Ad Set
- Ad
- Ad Creative
- Ad Account Insight
- Ad Image
- Ad Insights
- AdSets Insights
- Campaigns Insights

詳細については、「Marketing API」を参照してください。

> **Note**
>
> 構造体データ型とリストデータ型は、コネクタの応答で文字列データ型に変換されます。

パーティショニングクエリ

Spark で同時実行を使用する場合は、追加の Spark オプション PARTITION_FIELD、LOWER_BOUND、UPPER_BOUND、および NUM_PARTITIONS を指定できます。これらのパラメータを使用すると、元のクエリは Spark タスクで同時に実行できるサブクエリの NUM_PARTITIONS の数に分割されます。

- PARTITION_FIELD: クエリのパーティション化に使用するフィールドの名前。
- LOWER_BOUND: 選択したパーティションフィールドの包括的な下限値。

  DateTime フィールドでは、Spark SQL クエリで使用される Spark タイムスタンプ形式を受け入れます。

  有効な値の例:

```
"2022-01-01T00:00:00.000Z"
```

- UPPER_BOUND: 選択したパーティションフィールドの排他的上限値。

  有効な値の例:

  ```
  "2024-01-02T00:00:00.000Z"
  ```

- NUM_PARTITIONS: パーティション数。

例:

```
instagramADs_read = glueContext.create_dynamic_frame.from_options(
    connection_type="instagramads",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "entityName",
        "API_VERSION": "v20.0",
        "PARTITION_FIELD": "created_time"
        "LOWER_BOUND": "2022-01-01T00:00:00.000Z"
        "UPPER_BOUND": "2024-01-02T00:00:00.000Z"
        "NUM_PARTITIONS": "10"
    }
```

Instagram 広告接続オプション

Instagram 広告の接続オプションは次のとおりです:

- ENTITY_NAME(文字列) – (必須) 読み取りに使用されます。Instagram 広告のオブジェクトの名前。
- API_VERSION(文字列) – (必須) 読み取りに使用されます。使用する Instagram 広告グラフ API バージョン。例: v21。
- SELECTED_FIELDS(List<String>) - Default: empty(SELECT *). 読み込みに使用されます。オブジェクトに選択する列。
- FILTER_PREDICATE(文字列) - デフォルト: 空 読み込みに使用されます。Spark SQL 形式である必要があります。
- QUERY(文字列) - デフォルト: 空 読み込みに使用されます。完全な Spark SQL クエリ。
- PARTITION_FIELD (文字列) – 読み取りに使用されます。クエリをパーティション化するために使用するフィールド。
- LOWER_BOUND (文字列)– 読み取りに使用されます。選択したパーティションフィールドの包括的な下限値。

- UPPER_BOUND (文字列) – 読み取りに使用されます。選択したパーティションフィールドの排他的上限値。
- NUM_PARTITIONS(整数) - デフォルト: 1。読み込みに使用されます。読み取り用のパーティションの数。

Instagram 広告コネクタの制限と注意事項

以下に、Instagram 広告コネクタの制限事項または注意事項を示します:

- アプリの呼び出し回数は、1 時間のウィンドウ 200 中にユーザーが実行できる呼び出しの回数に、ユーザー数を掛けたものです。レート制限の詳細については、「Rate Limits」と「Business Use Case Rate Limits」を参照してください。

AWS Glue Studio での Intercom への接続

Intercom は、エンゲージメント OS であり、製品内、その瞬間、およびその条件においてビジネスと顧客間のオープンチャネンルとして機能します。カスタマージャーニー全体のあらゆるエンゲージメントを最大限に活用できるよう継続的な対話を作成します。

トピック

- AWS Glue による Intercom のサポート
- 接続を作成および使用するための API オペレーションを含むポリシー
- Intercom の設定
- Intercom 接続の設定
- Intercom エンティティからの読み取り
- Intercom 接続オプション
- 制限
- 新しい Intercom アカウントの作成とクライアントアプリの設定

AWS Glue による Intercom のサポート

AWS Glue は、次のように Intercom をサポートします:

ソースとしてサポートされていますか?

はい。AWS Glue ETL ジョブを使用して、Intercom からデータをクエリできます。

ターゲットとしてサポートされているかどうか

いいえ。

サポートされている Intercom API のバージョン

v2.5。バージョン固有ごとのエンティティサポートについては、「Intercom エンティティからの読み取り」を参照してください。

接続を作成および使用するための API オペレーションを含むポリシー

次のサンプル ポリシーで、接続の作成と使用に必要な IAM 権限について説明します。新しいロールを作成する場合は、以下を含むポリシーを作成します。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:ListConnectionTypes",
                "glue:DescribeConnectionType",
                "glue:RefreshOAuth2Tokens"
                "glue:ListEntities",
                "glue:DescribeEntity"
            ],
            "Resource": "*"
        }
    ]
}
```

また、次のマネージド IAM ポリシーを使用して、アクセスを許可できます。

- AWSGlueServiceRole – さまざまな AWS Glue プロセスを実行するために必要なリソースへのアクセス権をユーザーに代わって付与します。これらのリソースには、AWS Glue、Amazon S3、IAM、Amazon CloudWatch Logs および Amazon EC2 が含まれます。このポリシーで指定されたリソースの命名規則に従った場合、AWS Glue プロセスは必要なアクセス権限を使用できます。このポリシーは、通常、クローラ、ジョブ、開発エンドポイントを定義するときに指定されたロールにアタッチされます。
- AWSGlueConsoleFullAccess — ポリシーがアタッチされているアイデンティティが AWS マネージメントコンソールを使用するときは、AWS Glue リソースへのフルアクセスを許可します。この

ポリシーで指定されたリソースの命名規則に従った場合、ユーザーは完全なコンソール機能を使用できます。このポリシーは、通常 AWS Glue コンソールのユーザーにアタッチされています。

Intercom の設定

AWS Glue を使用して Intercom から転送できるようにするには、次の要件を満たす必要があります:

最小要件

- Intercom アカウントを持っていること。詳細については、「新しい Intercom アカウントの作成とクライアントアプリの設定」を参照してください。
- Intercom アカウントで API アクセスが有効になっていること。
- Intercom デベロッパーアカウントの下には、AWS Glue がアカウントに対して認証済みの呼び出しを行うときにデータに安全にアクセスするために使用するクライアント認証情報を提供するアプリが必要です。詳細については、「Intercom- New Account and Client App Creation Steps」を参照してください。

これらの要件を満たすと、Intercom アカウントに AWS Glue を接続する準備が整います。

Intercom 接続の設定

Intercom は OAuth 2 の `AUTHORIZATION_CODE` グラントタイプをサポートしています。

このグラントタイプは、ユーザーを認証するためにサードパーティーの認証サーバーにユーザーをリダイレクトすることから、「three-legged」の OAuth と見なされます。AWS Glue コンソール経由で接続を作成するときに使用されます。AWS Glue コンソールでは、ユーザーが Google 広告にリダイレクトされます。ここではユーザーがログインし、Intercom インスタンスにアクセスするためのリクエストされた権限を AWS Glue に許可する必要があります。

AWS Glue コンソールを介して接続を作成するときは、ユーザーは独自のクライアント ID とクライアントシークレットを指定する必要があります。このシナリオでは、引き続き Intercom にリダイレクトされてログインし、リソースへアクセスするために AWS Glue を承認します。

このグラントタイプは、更新トークンとアクセストークンになります。アクセストークンの有効期間は短く、更新トークンを使用してユーザーとやり取りすることなく自動的に更新される場合があります。

OAuth 認証コードフローでの接続されたアプリケーションの作成の詳細については、「Ads API」を参照してください。

Intercom 接続を設定するには:

1. AWS Secrets Manager で、次の詳細を含むシークレットを作成します。AWS Glue で接続ごとにシークレットを作成する必要があります。
   a. カスタマーマネージド接続アプリケーションの場合 – シークレットには、接続されたアプリケーションアクセストークン、更新トークン、client_id、および client_secret が含まれている必要があります。
2. AWS Glue Glue Studio で、以下の手順に従って [データ接続] の下に接続を作成します。
   a. [接続タイプ] を選択する際に、[Intercom] を選択します。
   b. Intercom 環境を指定します。
   c. AWS Glue が次のアクションを担い、その権限を持つことができる IAM ロールを選択します。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:DescribeSecret",
                "secretsmanager:GetSecretValue",
                "secretsmanager:PutSecretValue",
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterface",
                "ec2:DeleteNetworkInterface",
            ],
            "Resource": "*"
        }
    ]
}
```

   d. AWS Glue でこの接続に使用する `secretName` を選択して、トークンを配置します。
   e. ネットワークを使用する場合は、ネットワークオプションを選択します。
3. AWS Glue ジョブに関連付けられている IAM ロールに `secretName` を読み取るアクセス許可を付与します。

Intercom エンティティからの読み取り

前提条件

- 読み取り元の Intercom オブジェクト。使用可能なエンティティを確認するには、以下のサポートされているエンティティの表を参照してください。

サポートされているエンティティ

エンティティ	API_Version	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
管理者	v2.5	いいえ	いいえ	なし	あり	いいえ
Companies	v2.5	いいえ	あり	なし	あり	不可
会話	v2.5	可能	可能	可能	可能	可能
Data Attributes	v2.5	いいえ	いいえ	なし	あり	いいえ
問い合わせ	v2.5	可能	可能	可能	可能	あり
セグメント	v2.5	いいえ	いいえ	なし	あり	不可
[タグ]	v2.5	いいえ	いいえ	なし	あり	不可
チーム	v2.5	いいえ	いいえ	なし	あり	いいえ

例

```
Intercom_read = glueContext.create_dynamic_frame.from_options(
    connection_type="Intercom",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "company",
        "API_VERSION": "V2.5"
    }
)
```

Intercom エンティティとフィールドの詳細

エンティティ	フィールド	データタイプ	サポートされる演算子
管理者	type	文字列	NA
管理者	id	文字列	NA
管理者	avatar	Struct	NA
管理者	名前	文字列	NA
管理者	email	文字列	NA
管理者	away_mode_enabled	ブール値	NA
管理者	away_mode_reassign	ブール値	NA
管理者	has_inbox_seat	ブール値	NA
管理者	teams_ids	リスト	NA
管理者	job_title	文字列	NA
Companies	type	文字列	NA
Companies	id	文字列	NA
Companies	app_id	文字列	NA
Companies	created_at	DateTime	NA
Companies	remote_created_at	DateTime	NA
Companies	updated_at	DateTime	NA
Companies	last_request_at	DateTime	NA
Companies	計画	Struct	NA
Companies	company_id	文字列	NA
Companies	名前	文字列	NA

エンティティ	フィールド	データタイプ	サポートされる演算子
Companies	custom_attributes	Struct	NA
Companies	session_count	整数	NA
Companies	monthly_spend	整数	NA
Companies	user_count	整数	NA
Companies	industry	文字列	NA
Companies	size	整数	NA
Companies	website	文字列	NA
Companies	tags	Struct	NA
Companies	segments	Struct	NA
問い合わせ	id	文字列	EQUAL_TO. NOT_EQUAL_TO
問い合わせ	type	文字列	NA
問い合わせ	workspace_id	文字列	NA
問い合わせ	external_id	文字列	CONTAINS、EQUAL_TO、NOT_EQUAL_TO
問い合わせ	ロール	文字列	EQUAL_TO. NOT_EQUAL_TO
問い合わせ	email	文字列	CONTAINS、EQUAL_TO、NOT_EQUAL_TO

エンティティ	フィールド	データタイプ	サポートされる演算子
問い合わせ	phone	文字列	CONTAINS、EQUAL_TO、NOT_EQUAL_TO
問い合わせ	名前	文字列	CONTAINS、EQUAL_TO、NOT_EQUAL_TO
問い合わせ	avatar	文字列	NA
問い合わせ	owner_id	整数	EQUAL_TO、NOT_EQUAL_TO、GREATER_THAN、LESS_THAN
問い合わせ	social_profiles	Struct	NA
問い合わせ	has_hard_bounced	ブール値	EQUAL_TO
問い合わせ	marked_email_as_spam	ブール値	EQUAL_TO
問い合わせ	unsubscribed_from_emails	ブール値	EQUAL_TO
問い合わせ	created_at	DateTime	EQUAL_TO、GREATER_THAN、LESS_THAN
問い合わせ	updated_at	DateTime	EQUAL_TO、GREATER_THAN、LESS_THAN

エンティティ	フィールド	データタイプ	サポートされる演算子
問い合わせ	signed_up_at	DateTime	EQUAL_TO、GREATER_THAN、LESS_THAN
問い合わせ	last_seen_at	DateTime	EQUAL_TO、GREATER_THAN、LESS_THAN
問い合わせ	last_replied_at	DateTime	EQUAL_TO、GREATER_THAN、LESS_THAN
問い合わせ	last_contacted_at	DateTime	EQUAL_TO、GREATER_THAN、LESS_THAN
問い合わせ	last_email_opened_at	DateTime	EQUAL_TO、GREATER_THAN、LESS_THAN
問い合わせ	last_email_clicked_at	DateTime	EQUAL_TO、GREATER_THAN、LESS_THAN
問い合わせ	language_override	文字列	CONTAINS、EQUAL_TO、NOT_EQUAL_TO
問い合わせ	ブラウザ	文字列	CONTAINS、EQUAL_TO、NOT_EQUAL_TO
問い合わせ	browser_version	文字列	CONTAINS、EQUAL_TO、NOT_EQUAL_TO

エンティティ	フィールド	データタイプ	サポートされる演算子
問い合わせ	browser_language	文字列	CONTAINS、EQUAL_TO、NOT_EQUAL_TO
問い合わせ	os	文字列	CONTAINS、EQUAL_TO、NOT_EQUAL_TO
問い合わせ	location	Struct	NA
問い合わせ	location_country	文字列	CONTAINS、EQUAL_TO、NOT_EQUAL_TO
問い合わせ	location_region	文字列	CONTAINS、EQUAL_TO、NOT_EQUAL_TO
問い合わせ	location_city	文字列	CONTAINS、EQUAL_TO、NOT_EQUAL_TO
問い合わせ	Android_app_name	文字列	CONTAINS、EQUAL_TO、NOT_EQUAL_TO
問い合わせ	Android_app_version	文字列	NA
問い合わせ	android_device	文字列	NA
問い合わせ	android_os_version	文字列	NA
問い合わせ	android_sdk_version	文字列	CONTAINS、EQUAL_TO、NOT_EQUAL_TO

エンティティ	フィールド	データタイプ	サポートされる演算子
問い合わせ	android_last_seen_at	日付	NA
問い合わせ	ios_app_name	文字列	CONTAINS、EQUAL_TO、NOT_EQUAL_TO
問い合わせ	ios_app_version	文字列	NA
問い合わせ	ios_device	文字列	NA
問い合わせ	ios_os_version	文字列	CONTAINS、EQUAL_TO、NOT_EQUAL_TO
問い合わせ	ios_sdk_version	文字列	CONTAINS、EQUAL_TO、NOT_EQUAL_TO
問い合わせ	ios_last_seen_at	DateTime	NA
問い合わせ	custom_attributes	Struct	NA
問い合わせ	tags	Struct	NA
問い合わせ	注意事項	Struct	NA
問い合わせ	companies	Struct	NA
問い合わせ	unsubscribed_from_sms	ブール値	NA
問い合わせ	sms_consent	ブール値	NA
問い合わせ	opted_out_subscription_types	Struct	NA
問い合わせ	referrer	文字列	NA

エンティティ	フィールド	データタイプ	サポートされる演算子
問い合わせ	utm_campaign	文字列	NA
問い合わせ	utm_content	文字列	NA
問い合わせ	utm_medium	文字列	NA
問い合わせ	utm_source	文字列	NA
問い合わせ	utm_term	文字列	NA
会話	type	文字列	NA
会話	id	整数	EQUAL_TO、NOT_EQUAL_TO、GREATER_THAN、LESS_THAN
会話	created_at	DateTime	EQUAL_TO、NOT_EQUAL_TO、GREATER_THAN、LESS_THAN
会話	updated_at	DateTime	EQUAL_TO、NOT_EQUAL_TO、GREATER_THAN、LESS_THAN
会話	ソース	Struct	NA
会話	source_id	文字列	EQUAL_TO、NOT_EQUAL_TO

エンティティ	フィールド	データタイプ	サポートされる演算子
会話	source_type	文字列	EQUAL_TO、NOT_EQUAL_TO、
会話	source_delivered_as	文字列	EQUAL_TO、NOT_EQUAL_TO、
会話	source_subject	文字列	CONTAINS、EQUAL_TO、NOT_EQUAL_TO
会話	source_body	文字列	CONTAINS、EQUAL_TO、NOT_EQUAL_TO
会話	source_author_id	文字列	CONTAINS、EQUAL_TO、NOT_EQUAL_TO
会話	source_author_type	文字列	CONTAINS、EQUAL_TO、NOT_EQUAL_TO
会話	source_author_name	文字列	CONTAINS、EQUAL_TO、NOT_EQUAL_TO
会話	source_author_email	文字列	CONTAINS、EQUAL_TO、NOT_EQUAL_TO
会話	source_url	文字列	CONTAINS、EQUAL_TO、NOT_EQUAL_TO
会話	contacts	Struct	NA

エンティティ	フィールド	データタイプ	サポートされる演算子
会話	teammates	Struct	NA
会話	title	文字列	NA
会話	admin_assignee_id	整数	EQUAL_TO、NOT_EQUAL_TO、GREATER_THAN、LESS_THAN
会話	team_assignee_id	整数	CONTAINS、EQUAL_TO、NOT_EQUAL_TO
会話	custom_attributes	Struct	NA
会話	open	ブール値	EQUAL_TO
会話	state	文字列	CONTAINS、EQUAL_TO、NOT_EQUAL_TO
会話	read	ブール値	EQUAL_TO
会話	waiting_since	DateTime	EQUAL_TO、NOT_EQUAL_TO、GREATER_THAN、LESS_THAN
会話	snoozed_until	DateTime	EQUAL_TO、NOT_EQUAL_TO、GREATER_THAN、LESS_THAN

エンティティ	フィールド	データタイプ	サポートされる演算子
会話	tags	Struct	NA
会話	first_contact_reply	Struct	NA
会話	priority	文字列	EQUAL_TO、NOT_EQUAL_TO
会話	トピック	Struct	NA
会話	sla_applied	Struct	NA
会話	conversation_rating	Struct	NA
会話	conversation_rating_requested_at	DateTime	EQUAL_TO、NOT_EQUAL_TO、GREATER_THAN、LESS_THAN
会話	conversation_rating_replied_at	DateTime	EQUAL_TO、NOT_EQUAL_TO、GREATER_THAN、LESS_THAN
会話	conversation_rating_score	整数	EQUAL_TO、NOT_EQUAL_TO、GREATER_THAN、LESS_THAN
会話	conversation_rating_remark	文字列	CONTAINS、EQUAL_TO、NOT_EQUAL_TO

エンティティ	フィールド	データタイプ	サポートされる演算子
会話	conversation_rating_contact_id	文字列	CONTAINS、EQUAL_TO、NOT_EQUAL_TO
会話	conversation_rating_admin_id	文字列	CONTAINS、EQUAL_TO、NOT_EQUAL_TO
会話	統計	Struct	NA
会話	statistics_time_to_assignment	整数	EQUAL_TO、NOT_EQUAL_TO、GREATER_THAN、LESS_THAN
会話	statistics_time_to_admin_reply	整数	EQUAL_TO、NOT_EQUAL_TO、GREATER_THAN、LESS_THAN
会話	statistics_time_to_first_close	整数	EQUAL_TO、NOT_EQUAL_TO、GREATER_THAN、LESS_THAN
会話	statistics_time_to_last_close	整数	EQUAL_TO、NOT_EQUAL_TO、GREATER_THAN、LESS_THAN

エンティティ	フィールド	データタイプ	サポートされる演算子
会話	statistics_median_time_to_reply	整数	EQUAL_TO、NOT_EQUAL_TO、GREATER_THAN、LESS_THAN
会話	statistics_first_contact_reply_at	DateTime	EQUAL_TO、NOT_EQUAL_TO、GREATER_THAN、LESS_THAN
会話	statistics_first_assignment_at	DateTime	EQUAL_TO、NOT_EQUAL_TO、GREATER_THAN、LESS_THAN
会話	statistics_first_admin_reply_at	DateTime	EQUAL_TO、NOT_EQUAL_TO、GREATER_THAN、LESS_THAN
会話	statistics_first_close_at	DateTime	EQUAL_TO、NOT_EQUAL_TO、GREATER_THAN、LESS_THAN
会話	statistics_last_assignment_at	DateTime	EQUAL_TO、NOT_EQUAL_TO、GREATER_THAN、LESS_THAN

エンティティ	フィールド	データタイプ	サポートされる演算子
会話	statistics_last_assignment_admin_reply_at	DateTime	EQUAL_TO、NOT_EQUAL_TO、GREATER_THAN、LESS_THAN
会話	statistics_last_contact_reply_at	DateTime	EQUAL_TO、NOT_EQUAL_TO、GREATER_THAN、LESS_THAN
会話	statistics_last_admin_reply_at	DateTime	EQUAL_TO、NOT_EQUAL_TO、GREATER_THAN、LESS_THAN
会話	statistics_last_close_at	DateTime	EQUAL_TO、NOT_EQUAL_TO、GREATER_THAN、LESS_THAN
会話	statistics_last_closed_by_id	文字列	CONTAINS、EQUAL_TO、NOT_EQUAL_TO
会話	statistics_count_reopens	整数	EQUAL_TO、NOT_EQUAL_TO、GREATER_THAN、LESS_THAN

エンティティ	フィールド	データタイプ	サポートされる演算子
会話	statistics_count_assignments	整数	EQUAL_TO、NOT_EQUAL_TO、GREATER_THAN、LESS_THAN
会話	statistics_count_conversation_parts	整数	EQUAL_TO、NOT_EQUAL_TO、GREATER_THAN、LESS_THAN
会話	conversation_parts	リスト	NA
Data Attributes	id	整数	NA
Data Attributes	type	文字列	NA
Data Attributes	モデル	文字列	NA
Data Attributes	名前	文字列	NA
Data Attributes	full_name	文字列	NA
Data Attributes	ラベル	文字列	NA
Data Attributes	description	文字列	NA
Data Attributes	data_type	文字列	NA
Data Attributes	options:	リスト	NA
Data Attributes	api_writable	ブール値	NA
Data Attributes	ui_writable	ブール値	NA
Data Attributes	カスタム	ブール値	NA

エンティティ	フィールド	データタイプ	サポートされる演算子
Data Attributes	archived	ブール値	NA
Data Attributes	created_at	ブール値	NA
Data Attributes	updated_at	DateTime	NA
Data Attributes	admin_id	文字列	NA
セグメント	type	文字列	NA
セグメント	id	文字列	NA
セグメント	名前	文字列	NA
セグメント	created_at	DateTime	NA
セグメント	updated_at	DateTime	NA
セグメント	person_type	文字列	NA
セグメント	count	整数	NA
[タグ]	type	文字列	NA
[タグ]	id	文字列	NA
[タグ]	名前	文字列	NA
チーム	type	文字列	NA
チーム	id	文字列	NA
チーム	名前	文字列	NA
チーム	admin_ids	リスト	NA

パーティショニングクエリ

Spark で同時実行を使用する場合は、追加の Spark オプション PARTITION_FIELD、LOWER_BOUND、UPPER_BOUND、および NUM_PARTITIONS を指定できます。これらのパラメータを使用すると、元のクエリは Spark タスクで同時に実行できるサブクエリの NUM_PARTITIONS の数に分割されます。

- PARTITION_FIELD: クエリのパーティション化に使用するフィールドの名前。
- LOWER_BOUND: 選択したパーティションフィールドの包括的な下限値。

  日付については、Spark SQL クエリで使用される Spark の日付形式を受け入れます。有効な値の例: "2024-02-06"。
- UPPER_BOUND: 選択したパーティションフィールドの排他的上限値。
- NUM_PARTITIONS: パーティション数。

エンティティごとのパーティション分割フィールドのサポートの詳細は、次の表にまとめられています。

エンティティ名	Partitioning Field	データタイプ
問い合わせ	created_at、update_at、last_seen_at	DateTime
会話	id	整数
会話	created_at、update_at	DateTime

例

```
Intercom_read = glueContext.create_dynamic_frame.from_options(
    connection_type="Intercom",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "conversation",
        "API_VERSION": "V2.5",
        "PARTITION_FIELD": "created_at"
        "LOWER_BOUND": "2022-07-13T07:55:27.065Z"
        "UPPER_BOUND": "2022-08-12T07:55:27.065Z"
        "NUM_PARTITIONS": "2"
    }
```

```
)
```

Intercom 接続オプション

Intercom の接続オプションは次のとおりです:

- ENTITY_NAME (文字列) – (必須) 読み取りに使用されます。Intercom 内のオブジェクトの名前。
- API_VERSION (文字列) – (必須) 読み取りに使用されます。使用する Intercom Rest API のバージョン。例: v2.5。
- SELECTED_FIELDS(List<String>) - Default: empty(SELECT *). 読み込みに使用されます。オブジェクトに選択する列。
- FILTER_PREDICATE(文字列) - デフォルト: 空 読み込みに使用されます。Spark SQL 形式である必要があります。
- QUERY(文字列) - デフォルト: 空 読み込みに使用されます。完全な Spark SQL クエリ。
- PARTITION_FIELD (文字列) - 読み取りに使用されます。クエリをパーティション化するために使用するフィールド。
- LOWER_BOUND(文字列) - 読み取りに使用されます。選択したパーティションフィールドの包括的な下限値。
- UPPER_BOUND (文字列) - 読み取りに使用されます。選択したパーティションフィールドの排他的上限値。
- NUM_PARTITIONS(整数) - デフォルト: 1。読み込みに使用されます。読み取り用のパーティションの数。
- INSTANCE_URL (文字列) – ユーザーがオペレーションを実行するインスタンスの URL。例: https://api.intercom.io。

制限

Intercom コネクタの制限は次のとおりです:

- 会社エンティティを使用する場合、返すことができる会社は 10,000 社に制限されています。詳細については、「List all companies API」を参照してください。
- 並び替えを適用する場合、Contact エンティティと Conversation エンティティの両方にフィルターが必須です。
- MCA は SaaS プロバイダーによってサポートされています。ただし、ドキュメントに記載されている API レート制限に基づいて、他のワークロードに影響を与え、リソースの競合によりパ

フォーマンスの問題が発生する可能性があるため、AWS Glue で MCA をホストすることはありません。

新しい Intercom アカウントの作成とクライアントアプリの設定

Intercom アカウントの作成

1. Intercom URL で選択し、ページの右上隅にある [無料トライアルの開始] を選択します。
2. ページの右上隅にある[無料試用ボタン] を選択します。
3. 必要なビジネスタイプを選択します。
4. このページで、必要な情報をすべて入力します。
5. すべての情報を入力したら、[登録] を選択します。

Intercom 開発者アプリの作成

[クライアント ID] と [クライアントシークレット] を取得するには、開発者アカウントを作成します。

1. https://app.intercom.com/ に移動します。
2. Google を使用して E メール ID とパスワード/サインインを入力し、ログインします。
3. 左下隅の [ユーザープロフィール] を選択し、設定を選択します。
4. [アプリと統合] を選択します。
5. [アプリと統合] で [開発者ハブ] タブを選択します。
6. [新規アプリ] を選択し、ここでアプリを作成します。
7. アプリ名を指定し、アプリの [作成] を選択します。
8. アプリ内で、[認証] セクションに移動します。
9. [編集] を選択し、リダイレクト URI を追加します。リージョン固有のリダイレクト URL を `https://<aws-region>.console.aws.amazon.com/gluestudio/oauth` として追加します。例えば、`https://us-east-1.console.aws.amazon.com/gluestudio/oauth for the us-east-1 region` を追加します。
10. 生成された [クライアント ID] と [クライアントシークレット] を [基本情報セクション] で取得します。

Jira Cloud への接続

Jira Cloud は、Atlassian によって開発されたプラットフォームです。このプラットフォームには、チームがアジャイルプロジェクトの計画と追跡に役立つ問題追跡製品が含まれています。Jira Cloud ユーザーの場合、アカウントには、問題、ワークフロー、イベントなどのプロジェクトに関するデータが含まれます。 AWS Glue を使用して、Jira Cloud データを特定の AWS サービスやその他のサポートされているアプリケーションに転送できます。

トピック

- AWS Glue Jira Cloud のサポート
- 接続を作成および使用するためのAPIオペレーションを含むポリシー
- Jira Cloud の設定
- Jira Cloud 接続の設定
- Jira Cloud エンティティからの読み取り
- Jira Cloud 接続オプション
- Jira Cloud コネクタの制限と注意事項

AWS Glue Jira Cloud のサポート

AWS Glue は、次のように Jira Cloud をサポートします。

ソースとしてサポートされていますか?

はい。ジョブを使用して AWS Glue ETL Jira Cloud からデータをクエリできます。

ターゲットとしてサポートされているかどうか

いいえ。

サポートされている Jira Cloud APIバージョン

次の Jira Cloud APIバージョンがサポートされています。

- v2

バージョン固有のエンティティサポートについては、「Source でサポートされているエンティティ」を参照してください。

接続を作成および使用するためのAPIオペレーションを含むポリシー

次のサンプルポリシーでは、接続の作成と使用に必要な AWS IAMアクセス許可について説明します。新しいロールを作成する場合は、以下を含むポリシーを作成します。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:ListConnectionTypes",
                "glue:DescribeConnectionType",
                "glue:RefreshOAuth2Tokens"
                "glue:ListEntities",
                "glue:DescribeEntity"
            ],
            "Resource": "*"
        }
    ]
}
```

上記の方法を使用しない場合は、次の管理IAMポリシーを使用します。

- AWSGlueServiceRole – さまざまな AWS Glue プロセスがユーザーに代わって実行するために必要なリソースへのアクセスを許可します。これらのリソースには AWS Glue、、Amazon S3、IAM、 CloudWatch Logs、Amazon が含まれますEC2。このポリシーで指定されたリソースの命名規則に従うと、 AWS Glue プロセスには必要なアクセス許可があります。このポリシーは、通常、クローラ、ジョブ、開発エンドポイントを定義するときに指定されたロールにアタッチされます。
- AWSGlueConsoleFullAccess — ポリシーがアタッチされている ID が AWS マネジメントコンソールを使用する場合、 AWS Glue リソースへのフルアクセスを許可します。このポリシーで指定されたリソースの命名規則に従った場合、ユーザーは完全なコンソール機能を使用できます。このポリシーは通常、 AWS Glue コンソールのユーザーにアタッチされます。

Jira Cloud の設定

AWS Glue を使用して Jira Cloud からサポートされている送信先にデータを転送する前に、次の要件を満たす必要があります。

最小要件

以下に、最小要件を示します。

- Jira Cloud で Jira ソフトウェア製品を使用する Atlassian アカウントがあります。詳細については、「Jira Cloud アカウントの作成」を参照してください。
- へのサービスアクセスを持つ AWS アカウントを作成する必要があります AWS Glue。
- このアプリは、 がアカウントに対して認証された呼び出しを行うときに、 AWS Glue がデータに安全にアクセスするためのクライアント認証情報を提供します。詳細については、「Atlassian デベロッパードキュメントOAuth」の「Enable 2.0 (3LO)」を参照してください。

これらの要件を満たすと、Jira Cloud アカウント AWS Glue に接続できます。

Jira Cloud アカウントの作成

Jira Cloud アカウントを作成するには：

1. Atlassian サインアップ URLに移動します。
2. 勤務先の E メールアドレスと名前を入力し、同意 を選択します。検証 E メールが届きます。
3. E メールを確認したら、パスワードを作成し、サインアップ を選択します。
4. 名前とパスワードを入力し、サインアップ を選択します。
5. サイトにアクセスする必要があるページにリダイレクトされます。サイト名を入力し、同意 を選択します。

Atlassian Cloud サイトが起動したら、プロジェクトタイプの設定に従っていくつかの質問に答えて Jira を設定できます。

既存のアカウントにログインするには：

1. Atlassian ログインURLに移動し、認証情報を入力します。
2. E メールとパスワードを入力し、 でログインをクリックします。Jira ダッシュボードにリダイレクトされます。

Jira Cloud でのアプリケーションの作成

Jira Cloud でアプリケーションを作成し、マネージドクライアントアプリケーションからクライアント ID とクライアントシークレットを取得するには：

1. Jira Cloud URL に移動し、認証情報を入力します。
2. Create を選択し、OAuth2.0 統合オプションを選択します。
3. アプリケーション名を入力し、T&C をチェックして の作成 を選択します。
4. 左メニューの Distribution セクションに移動し、Edit を選択します。
5. ディストリビューションコントロールの編集セクションで、次の操作を行います。
   a. 共有 DISTRIBUTIONSTATUSとして を選択します。
   b. ベンダー名を入力します。
   c. プライバシーポリシー URLの を入力します。例えば、 https://docs.aws.amazon.com/glue/latest/dg/security-iam-awsmanpol.html
   d. 利用規約URLの を入力します (オプション)。
   e. カスタマーサポート連絡先URLの を入力します (オプション)。
   f. から Yes/No を選択しPERSONALDATADECLARATION、変更の保存 を選択します。
6. 各アプリの左側のメニューのアクセス許可に移動します。
7. Jira の場合はAPI、「 を追加」を選択します。追加したら、Configuration オプションを選択します。
8. Classic scopes > Jira プラットフォームRESTAPIセクションで、Edit Scopes を選択します。すべてのスコープを確認します。[Save] (保存) をクリックします。
9. 「細かなスコープ」で「スコープの編集」を選択し、次のスコープを選択します。
10. 下にスクロールしてスコープを見つけます。見出し「」と「スタンダードCRM」の 2 種類のスコープを選択する必要があります。
11. 次のスコープを追加します。

```
read:application-role:jira
read:audit-log:jira
read:avatar:jira
read:field:jira
read:group:jira
read:instance-configuration:jira
read:issue-details:jira
read:issue-event:jira
read:issue-link-type:jira
read:issue-meta:jira
read:issue-security-level:jira
read:issue-security-scheme:jira
read:issue-type-scheme:jira
read:issue-type-screen-scheme:jira
```

```
read:issue-type:jira
read:issue.time-tracking:jira
read:label:jira
read:notification-scheme:jira
read:permission:jira
read:priority:jira
read:project:jira
read:project-category:jira
read:project-role:jira
read:project-type:jira
read:project-version:jira
read:project.component:jir
read:project.property:jira
read:resolution:jira
read:screen:jira
read:status:jira
read:user:jira
read:workflow-scheme:jira
read:workflow:jira
read:field-configuration:jira
read:issue-type-hierarchy:jira
read:webhook:jira
```

12左のメニューで認証に移動し、 の追加 を選択します。

13https://us-east-1.console.aws.amazon.com/gluestudio/oauth などのコールバックURLを入力する

14左側のメニューの設定に移動し、下にスクロールして認証の詳細を確認します。クライアント ID とシークレットを書き留めます。

Jira Cloud 接続の設定

Jira Cloud は、OAuth2 の AUTHORIZATION_CODE のグラントタイプをサポートしています。

- このグラントタイプは、ユーザーを認証するためにサードパーティーの認証サーバーにユーザーをリダイレクトすることから、「three-legged」の OAuth と見なされます。AWS Glue コンソール経由で接続を作成するときに使用されます。AWS Glue コンソールでは、ユーザーが Jira Cloud にリダイレクトされます。ここではユーザーがログインし、Jira Cloud インスタンスにアクセスするためのリクエストされた権限を AWS Glue に許可する必要があります。
- ユーザーは、Jira Cloud で独自の接続アプリを作成し、AWS Glue コンソールを介して接続を作成するときに独自のクライアント ID とクライアントシークレットを指定することを選択できます。

このシナリオでは、引き続き Jira Cloud にリダイレクトされてログインし、リソースへアクセスするために AWS Glue を承認します。

- このグラントタイプは、更新トークンとアクセストークンになります。アクセストークンの有効期間は短く、更新トークンを使用してユーザーとやり取りすることなく自動的に更新される場合があります。
- 認可コード OAuth フロー用の接続アプリケーションの作成に関する Jira Cloud の公開ドキュメントについては、「OAuth 2.0 (3LO) の有効化」を参照してください。

Jira Cloud 接続を設定するには：

1. AWS Secrets Manager で、次の詳細を含むシークレットを作成します。
   a. カスタマーマネージド接続アプリケーションの場合、シークレットには、USER_MANAGED_CLIENT_APPLICATION_CLIENT_SECRET を使用して接続されたアプリケーションのコンシューマーシークレットをキーとして含める必要があります。
   b. 注: AWS Glue で接続ごとにシークレットを作成する必要があります。
2. AWS Glue Glue Studio で、以下の手順に従って [データ接続] の下に接続を作成します。
   a. [接続タイプ] を選択する際に、[Jira Cloud] を選択します。
   b. Jira Cloud 環境を指定します。
   c. 次のアクションを実行でき、AWS Glue がその権限を持つ AWS IAM ロールを選択します。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:DescribeSecret",
                "secretsmanager:GetSecretValue",
                "secretsmanager:PutSecretValue",
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterface",
                "ec2:DeleteNetworkInterface",
            ],
            "Resource": "*"
        }
    ]
}
```

d. AWS Glue でこの接続に使用する `secretName` を選択して、トークンを配置します。

e. ネットワークを使用する場合は、ネットワークオプションを選択します。

3. AWS Glue ジョブに関連付けられている IAM ロールに `secretName` を読み取るアクセス許可を付与します。

Jira Cloud エンティティからの読み取り

前提条件

読み取り元の Jira Cloud オブジェクト。監査レコードまたは問題などのオブジェクト名が必要です。次の表に、サポートされているエンティティを示します。

ソースに対応するエンティティ:

エンティティ	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
Audit Record	あり	はい	いいえ	はい	はい
問題	あり	はい	いいえ	はい	はい
Issue Field	なし	いいえ	いいえ	はい	なし
Issue Field Configuration	あり	はい	いいえ	はい	はい
Issue Link Type	なし	いいえ	いいえ	はい	なし
Issue Notification Scheme	あり	はい	いいえ	はい	はい
Issue Security Scheme	なし	いいえ	いいえ	はい	なし

エンティティ	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
Issue Type Scheme	あり	はい	はい	はい	はい
Issue Type Screen Scheme	あり	はい	はい	はい	はい
Issue Type	なし	いいえ	いいえ	はい	なし
Jira Setting	はい	いいえ	いいえ	はい	なし
Jira Setting Advanced	なし	いいえ	いいえ	はい	なし
Jira Setting Global	なし	いいえ	いいえ	はい	なし
ラベル	なし	いいえ	いいえ	はい	はい
Myself	はい	いいえ	いいえ	はい	なし
アクセス許可	なし	いいえ	いいえ	はい	いいえ。
プロジェクト	あり	はい	はい	はい	はい
Project Category	なし	いいえ	いいえ	はい	なし
プロジェクトタイプ	なし	いいえ	いいえ	はい	なし
Server Info	なし	いいえ	いいえ	はい	なし
[ユーザー]	なし	なし	いいえ。	はい	なし
ワークフロー	あり	はい	はい	はい	はい

エンティティ	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
Workflow Scheme	いいえ	はい	いいえ	はい	はい
Workflow Scheme Project Association	はい	いいえ	いいえ	はい	なし
ワークフローステータス	なし	いいえ	いいえ	はい	なし
Workflow Status Category	なし	いいえ	いいえ	はい	なし

例:

```
jiracloud_read = glueContext.create_dynamic_frame.from_options(
    connection_type="JiraCloud",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "audit-record",
        "API_VERSION": "v2"
    }
```

Jira Cloud エンティティとフィールドの詳細:

オブジェクト	フィールド	データ型	サポートされているフィルター演算子
Audit Record	フィルター	文字列	"="
	送信元	DateTime	"="

オブジェクト	フィールド	データ型	サポートされているフィルター演算子
	から	DateTime	"="
	id	整数	該当なし
	概要	文字列	該当なし
	remoteAddress	文字列	該当なし
	authorAccountId	文字列	該当なし
	作成済み	文字列	該当なし
	category	文字列	該当なし
	eventSource	文字列	該当なし
	説明	文字列	該当なし
	objectItem	Struct	該当なし
	changedValues	リスト	該当なし
	associatedItems	リスト	該当なし
グループ	groupName	リスト	"="
	名前	文字列	該当なし
	groupId	文字列	"="
問題	affectedVersion	文字列	"=, !="
	assignee	文字列	"=, !="
	category	文字列	"=, !="
	コンポーネント	文字列	"=, !="
	creator	文字列	"=, !="

オブジェクト	フィールド	データ型	サポートされているフィルター演算子
	due	DateTime	該当なし
	epic_link	文字列	"=, !="
	フィルター	文字列	"=, !="
	fixVersion	文字列	"=, !="
	hierarchyLevel	整数	"=, !="
	issueKey	文字列	"=, !=, >, <, >=, <="
	issueLink	文字列	"=, !="
	issueLinkType	文字列	"=, !="
	ラベル	文字列	"=, !="
	lastViewed	DateTime	"=, >, <, >=, <=, between"
	level	文字列	"=, !="
	parent	文字列	"=, !="
	priority	文字列	"=, !="
	プロジェクト	文字列	"=, !="
	projectType	文字列	"=, !="
	reporter	文字列	"=, !="
	resolution	文字列	"=, !="
	resolved	DateTime	"=, >, <, >=, <=, between"
	Sprint	文字列	"=, !="

オブジェクト	フィールド	データ型	サポートされているフィルター演算子
	status	文字列	"=, !="
	type	文字列	"=, !="
	更新済み	DateTime	"=, >, <, >=, <=, between"
	voter	文字列	"=, !="
	votes	整数	"=, !=, <, >, <=, >=, between"
	watcher	文字列	"=, !="
	watchers	整数	"=, !=, <, >, <=, >=, between"
	workRatio	整数	"=, !=, <, >, <=, >=, between"
	validateQuery	文字列	"="
	expand	文字列	"="
	fieldByKeys	ブール値	"="
	id	文字列	該当なし
	self	文字列	該当なし
	キー	文字列	該当なし
	renderedFields	Struct	該当なし
	プロパティ	リスト	"="
	names	Struct	該当なし

オブジェクト	フィールド	データ型	サポートされているフィルター演算子
	スキーマ	Struct	該当なし
	transitions	リスト	該当なし
	オペレーション	Struct	該当なし
	editmeta	Struct	該当なし
	変更ログ	Struct	該当なし
	versionedRepresentations	Struct	該当なし
	fields	リスト	"="
	fieldsToInclude	Struct	該当なし
	warningMessages	リスト	該当なし
	作成済み	DateTime	該当なし
	worklogDate	DateTime	該当なし
IssueEvents	id	整数	該当なし
	名前	文字列	該当なし
Issue Fields	id	文字列	該当なし
	キー	文字列	該当なし
	名前	文字列	該当なし
	カスタム	ブール値	該当なし
	orderable	ブール値	該当なし
	navigable	ブール値	該当なし

オブジェクト	フィールド	データ型	サポートされているフィルター演算子
	searchable	ブール値	該当なし
	clauseNames	リスト	該当なし
	scope	Struct	該当なし
	スキーマ	Struct	該当なし
Issue Field Configurations	isDefault	ブール値	"="
	query	文字列	"="
	id	整数	"="
	名前	文字列	該当なし
	説明	文字列	該当なし
Issue Link Type	id	文字列	該当なし
	名前	文字列	該当なし
	inward	文字列	該当なし
	outward	文字列	該当なし
	self	文字列	該当なし
Issue Notification Schemes	expand	文字列	"="
	self	文字列	該当なし
	id	整数	該当なし
	名前	文字列	該当なし
	説明	文字列	該当なし

オブジェクト	フィールド	データ型	サポートされているフィルター演算子
	notificationSchemeEvents	リスト	該当なし
	scope	Struct	該当なし
Issue Priority	self	文字列	該当なし
	statusColor	文字列	該当なし
	説明	文字列	該当なし
	iconUrl	文字列	該当なし
	名前	文字列	該当なし
	id	文字列	該当なし
	isDefault	ブール値	該当なし
Issue Resolutions	self	文字列	該当なし
	id	文字列	該当なし
	説明	文字列	該当なし
	名前	文字列	該当なし
Issue Security Scheme	self	文字列	該当なし
	id	整数	該当なし
	名前	文字列	該当なし
	説明	文字列	該当なし
	defaultSecurityLevelId	整数	該当なし
	levels	リスト	該当なし

オブジェクト	フィールド	データ型	サポートされている フィルター演算子
Issue Type	self	文字列	該当なし
	id	文字列	該当なし
	説明	文字列	該当なし
	iconUrl	文字列	該当なし
	名前	文字列	該当なし
	subtask	ブール値	該当なし
	avatarId	整数	該当なし
	entityId	文字列	該当なし
	hierarchyLevel	整数	該当なし
	scope	Struct	該当なし
Issue Type Scheme	orderBy	文字列	"="
	expand	文字列	"="
	queryString	文字列	"="
	id	文字列	該当なし
	名前	文字列	該当なし
	説明	文字列	該当なし
	defaultIssueTypeId	文字列	該当なし
	isDefault	ブール値	該当なし
Issue Type Screen Scheme	queryString	文字列	"="
	orderBy	文字列	"="

オブジェクト	フィールド	データ型	サポートされているフィルター演算子
	expand	文字列	"="
	id	文字列	"="
	名前	文字列	該当なし
	説明	文字列	該当なし
Jira Settings	キー	文字列	該当なし
	keyFilter	文字列	"="
	id	文字列	該当なし
	value	文字列	該当なし
	名前	文字列	該当なし
	desc	文字列	該当なし
	type	文字列	該当なし
	defaultValue	文字列	該当なし
	例	文字列	該当なし
	allowedValues	リスト	該当なし
Jira Settings Advanced	id	文字列	該当なし
	キー	文字列	該当なし
	value	文字列	該当なし
	名前	文字列	該当なし
	desc	文字列	該当なし
	type	文字列	該当なし

オブジェクト	フィールド	データ型	サポートされているフィルター演算子
	defaultValue	文字列	該当なし
	例	文字列	該当なし
	allowedValues	リスト	該当なし
Jira Settings Global	votingEnabled	ブール値	該当なし
	watchingEnabled	ブール値	該当なし
	unassignedIssuesAllowed	ブール値	該当なし
	subTasksEnabled	ブール値	該当なし
	issueLinkingEnabled	ブール値	該当なし
	timeTrackingEnabled	ブール値	該当なし
	attachmentsEnabled	ブール値	該当なし
	timeTrackingConfiguration	Struct	該当なし
ラベル	値	リスト	該当なし
Myself	expand	文字列	"="
	self	文字列	該当なし
	accountId	文字列	該当なし
	accountType	文字列	該当なし
	emailAddress	文字列	該当なし
	avatarUrls	文字列	該当なし
	displayName	文字列	該当なし

オブジェクト	フィールド	データ型	サポートされている フィルター演算子
	ACTIVE	ブール値	該当なし
	timeZone	文字列	該当なし
	サイト	文字列	該当なし
	グループ	Struct	該当なし
	applicationRoles	Struct	該当なし
アクセス許可	id	文字列	該当なし
	キー	文字列	該当なし
	名前	文字列	該当なし
	type	文字列	該当なし
	説明	文字列	該当なし
	havePermission	ブール値	該当なし
	deprecatedKey	ブール値	該当なし
プロジェクト	orderBy	文字列	"="
	キー	リスト	"="
	query	文字列	"="
	typeKey	文字列	"="
	categoryId	整数	"="
	アクション	文字列	"="
	expand	文字列	"="
	status	リスト	"="

オブジェクト	フィールド	データ型	サポートされているフィルター演算子
	self	文字列	該当なし
	id	整数	"="
	キー	文字列	該当なし
	説明	文字列	該当なし
	lead	Struct	該当なし
	コンポーネント	リスト	該当なし
	issueTypes	リスト	該当なし
	url	文字列	該当なし
	email	文字列	該当なし
	assigneeType	文字列	該当なし
	バージョン	リスト	該当なし
	名前	文字列	該当なし
	ロール	Struct	該当なし
	avatarUrls	Struct	該当なし
	projectCategory	Struct	該当なし
	projectTypeKey	文字列	該当なし
	簡略化済み	ブール値	該当なし
	style	文字列	該当なし
	favourite	ブール値	該当なし
	IsPrivate	ブール値	該当なし

オブジェクト	フィールド	データ型	サポートされている フィルター演算子
	issueTypeHierarchy	Struct	該当なし
	権限	Struct	該当なし
	プロパティ	リスト	"="
	uuid	文字列	該当なし
	insight	Struct	該当なし
	削除済み	ブール値	該当なし
	retentionTillDate	文字列	該当なし
	deletedDate	文字列	該当なし
	deletedBy	Struct	該当なし
	archived	ブール値	該当なし
	archivedDate	文字列	該当なし
	archivedBy	Struct	該当なし
	landedPageInfo	Struct	該当なし
Project Category	self	文字列	該当なし
	id	文字列	該当なし
	名前	文字列	該当なし
	説明	文字列	該当なし
プロジェクトタイプ	キー	文字列	該当なし
	formattedKey	文字列	該当なし
	説明	文字列	該当なし

オブジェクト	フィールド	データ型	サポートされている フィルター演算子
	descriptionI18nKey	文字列	該当なし
	アイコン	文字列	該当なし
	color	文字列	該当なし
Server Info	baseUrl	文字列	該当なし
	version	文字列	該当なし
	versionNumbers	リスト	該当なし
	deploymentType	文字列	該当なし
	buildNumber	整数	該当なし
	buildDate	DateTime	該当なし
	serverTime	DateTime	該当なし
	scmInfo	文字列	該当なし
	serverTitle	文字列	該当なし
	healthChecks	リスト	該当なし
[ユーザー]	self	文字列	該当なし
	accountId	文字列	該当なし
	accountType	文字列	該当なし
	emailAddress	文字列	該当なし
	avatarUrls	Struct	該当なし
	displayName	文字列	該当なし
	ACTIVE	ブール値	該当なし

オブジェクト	フィールド	データ型	サポートされている フィルター演算子
	timeZone	文字列	該当なし
	サイト	文字列	該当なし
	グループ	Struct	該当なし
	applicationRoles	Struct	該当なし
	expand	文字列	該当なし
ワークフロー	workflowName	文字列	"="
	expand	文字列	"="
	queryString	文字列	"="
	orderBy	文字列	"="
	isActive	ブール値	"="
	id	Struct	該当なし
	説明	文字列	該当なし
	transitions	リスト	該当なし
	statuses	リスト	該当なし
	isDefault	ブール値	該当なし
	schemes	リスト	該当なし
	projects	リスト	該当なし
	hasDraftWorkflow	ブール値	該当なし
	オペレーション	Struct	該当なし
	作成済み	文字列	該当なし

オブジェクト	フィールド	データ型	サポートされているフィルター演算子
	更新済み	文字列	該当なし
Workflow Scheme	self	文字列	該当なし
	id	整数	該当なし
	名前	文字列	該当なし
	説明	文字列	該当なし
	defaultWorkflow	文字列	該当なし
	issueTypeMappings	Struct	該当なし
	originalDefaultWorkflow	文字列	該当なし
	originalIssueTypeMappings	Struct	該当なし
	draft	ブール値	該当なし
	lastModifiedUser	Struct	該当なし
	lastModified	文字列	該当なし
	updateDraftIfNeeded	ブール値	該当なし
	issueTypes	Struct	該当なし
Workflow Scheme Project Association	projectId	整数	"="
	projectIds	リスト	該当なし
	workflowScheme	Struct	該当なし
ワークフローステータス	self	文字列	該当なし
	説明	文字列	該当なし

オブジェクト	フィールド	データ型	サポートされているフィルター演算子
	iconUrl	文字列	該当なし
	名前	文字列	該当なし
	id	文字列	該当なし
	StatusCategory	Struct	該当なし
Workflow Status Category	self	文字列	該当なし
	id	文字列	該当なし
	キー	文字列	該当なし
	colorName	文字列	該当なし
	名前	文字列	該当なし

パーティショニングクエリ

Spark で並行処理を使用する場合は、追加の Spark オプション NUM_PARTITIONS を指定できます。これらのパラメータを使用すると、元のクエリは Spark タスクで同時に実行できるサブクエリの NUM_PARTITIONS の数に分割されます。

- NUM_PARTITIONS: パーティション数。

例：

```
jiraCloud_read = glueContext.create_dynamic_frame.from_options(
    connection_type="JiraCloud",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "issue",
        "API_VERSION": "v2",
        "NUM_PARTITIONS": "10"
    }
```

Jira Cloud 接続オプション

Jira Cloud の接続オプションは次のとおりです。

- ENTITY_NAME (文字列) - (必須) 読み取りに使用されます。Jira Cloud 内のオブジェクトの名前。
- API_VERSION (文字列) - (必須) 読み取りに使用されます。使用する Jira Cloud Rest APIバージョン。例: v2。
- DOMAIN_URL (文字列) - (必須) 使用する Jira Cloud ID。
- SELECTED_FIELDS (リスト < 文字列>) - デフォルト: empty(SELECT *)。読み込みに使用されます。オブジェクトに選択する列。
- FILTER_PREDICATE(文字列) - デフォルト: 空 読み込みに使用されます。Spark SQL形式である必要があります。
- QUERY(文字列) - デフォルト: 空 読み込みに使用されます。フル Spark SQLクエリ。
- NUM_PARTITIONS(整数) - デフォルト: 1。読み込みに使用されます。読み取り用のパーティションの数。

Jira Cloud コネクタの制限と注意事項

Jira Cloud コネクタの制限事項または注意事項は次のとおりです。

- Contains 演算子は、文字列データ型の resourceNameフィールドでは動作しません。

Kustomer への接続

Kustomer は、顧客により良いサービスを提供するために必要なすべてを 1 つの使いやすいツールにまとめた、強力なカスタマーエクスペリエンスプラットフォームです。

トピック

- Kustomer の AWS Glue サポート
- 接続を作成および使用するための API オペレーションを含むポリシー
- Kustomer の設定
- Kustomer 接続の設定
- Kustomer エンティティからの読み取り
- Kustomer 接続のオプション

- Kustomer の制限事項

Kustomer の AWS Glue サポート

AWS Glue は、次のように Kustomer をサポートします。

ソースとしてサポートされていますか?

はい。AWS Glue ETL ジョブを使用して、Kustomer からデータをクエリできます。

ターゲットとしてサポートされていますか?

いいえ。

サポートされている Kustomer API バージョン

次の Kustomer API バージョンがサポートされています。

- v1

接続を作成および使用するための API オペレーションを含むポリシー

次のサンプル ポリシーで、接続の作成と使用に必要な AWS IAM 権限について説明します。新しいロールを作成する場合は、以下を含むポリシーを作成します。

```
{
 "Version": "2012-10-17",
     "Statement": [
      {
             "Effect": "Allow",
             "Action": [
                 "glue:ListConnectionTypes",
                 "glue:DescribeConnectionType",
                 "glue:RefreshOAuth2Tokens",
                 "glue:ListEntities",
                 "glue:DescribeEntity"
             ],
             "Resource": "*"
         }
     ]
 }
```

上記の方法を使用しない場合は、代わりに次のマネージド IAM ポリシーを使用します:

- AWSGlueServiceRole – さまざまな AWS Glue プロセスを実行するために必要なリソースへのアクセス権をユーザーに代わって付与します。これらのリソースには AWS Glue 、Amazon S3、IAM、CloudWatch Logs、および Amazon EC2 が含まれます。このポリシーで指定されたリソースの命名規則に従った場合、AWS Glue プロセスは必要なアクセス権限を使用できます。このポリシーは、通常、クローラ、ジョブ、開発エンドポイントを定義するときに指定されたロールにアタッチされます。
- AWSGlueConsoleFullAccess — ポリシーがアタッチされているアイデンティティが AWS マネージメントコンソールを使用するときは、AWS Glue リソースへのフルアクセスを許可します。このポリシーで指定されたリソースの命名規則に従った場合、ユーザーは完全なコンソール機能を使用できます。このポリシーは、通常 AWS Glue コンソールのユーザーにアタッチされています。

Kustomer の設定

AWS Glue を使用して Kustomer からサポートされている送信先にデータを転送できるようにするには、次の要件を満たす必要があります。

最小要件

以下に、最小要件を示します。

- 転送するデータを含む Kustomer のアカウントがある。
- アカウントの設定で、API キーを作成している。詳細については、「API キーの作成」を参照してください。
- 接続の作成時に AWS Glue に API キーを指定します。

これらの要件を満たすと、Kustomer アカウントに AWS Glue を接続する準備が整います。

API キーの作成

AWS Glue Studio で Kustomer コネクタ用の接続を作成するために使用する API キーを作成するには

1. 認証情報を使用して Kustomer ダッシュボードにログインします。
2. 左側のメニューから、[Settings] アイコンを選択します。
3. [Security] ドロップダウンを展開し、[API Keys] を選択します。
4. API キーの作成ページで、右上隅の [Add an API Key] を選択します。

5. 作成する API キーの必須項目を入力します。
   - Name: API キーの任意の名前。
   - Roles: Kustomer API が機能するには、「org」を選択する必要があります。
   - Expires (in days): API キーを有効にする日数。ユースケースに応じて、[Never expires] にすることもできます。
6. [Create] (作成) を選択します。
7. Kustomer コネクタ用の接続を作成するための API キー (トークン) 値を今後の使用のために AWS Glue Studio に保存します。

Kustomer 接続の設定

Kustomer 接続を設定するには

1. AWS Secrets Manager で、次の詳細を含むシークレットを作成します。
   a. カスタマーマネージド接続アプリケーションの場合、シークレットには、`apiKey` を使用して接続されたアプリケーションのコンシューマーシークレットをキーとして含める必要があります。
   b. 注: AWS Glue で接続にシークレットを作成する必要があります。

1. AWS Glue Glue Studio で、以下の手順に従って [データ接続] の下に接続を作成します。
   a. [接続] で [接続を作成] を選択します。
   b. [データソース] を選択するときは、[Kustomer] を選択します。
   c. 次のアクションを実行でき、AWS Glue がその権限を持つ AWS IAM ロールを選択します。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:DescribeSecret",
                "secretsmanager:GetSecretValue",
                "secretsmanager:PutSecretValue",
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterface",
                "ec2:DeleteNetworkInterface",
            ],
```

```
            "Resource": "*"
        }
    ]
}
```

d. AWS Glue でこの接続に使用する `secretName` を選択して、トークンを配置します。

e. ネットワークを使用する場合は、ネットワークオプションを選択します。

2. AWS Glue ジョブに関連付けられている IAM ロールに `secretName` を読み取るアクセス許可を付与します。

Kustomer エンティティからの読み取り

前提条件

読み取り元の Kustomer オブジェクト。Brands または Cards などのオブジェクト名が必要です。次の表に、サポートされているエンティティを示します。

ソースに対応するエンティティ:

エンティティ	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
Brands	いいえ	あり	なし	あり	不可
[Cards] (カード)	いいえ	あり	なし	あり	不可
Chat Settings	いいえ	いいえ	なし	あり	不可
Companies	あり	あり	あり	あり	あり
会話	あり	あり	あり	あり	あり
顧客	あり	あり	あり	あり	あり
Customer Searches Pinned	いいえ	あり	なし	あり	不可

エンティティ	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
Customer Searches Position	いいえ	いいえ	なし	あり	不可
Email Hooks	いいえ	あり	なし	あり	不可
Web Hooks	いいえ	あり	なし	あり	不可
KB Articles	いいえ	あり	なし	あり	不可
KB Categories	いいえ	あり	なし	あり	不可
KB Forms	いいえ	あり	なし	あり	不可
KB Routes	いいえ	あり	なし	あり	不可
KB Tags	いいえ	あり	なし	あり	不可
KB Templates	いいえ	あり	なし	あり	不可
KB Themes	いいえ	あり	なし	あり	不可
Klasses	いいえ	あり	なし	あり	不可
KViews	いいえ	あり	なし	あり	不可
メッセージ	あり	あり	あり	あり	あり
コメント	あり	あり	あり	あり	あり
通知	いいえ	あり	なし	あり	不可

例:

```
Kustomer_read = glueContext.create_dynamic_frame.from_options(
    connection_type="kustomer",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "brands",
        "API_VERSION": "v1"
    }
```

Kustomer のエンティティとフィールドの詳細

エンティティとフィールドの詳細については、以下を参照してください:

- Brands
- [Cards] (カード)
- Chat Settings
- Companies
- Conversations
- Customers
- Customers Searches Pinned
- Customer Searches Positions
- Hooks Email
- Hooks Web
- KB Articles
- KB Categories
- KB Forms
- KB Routes
- KB Tags
- KB Templates
- KB Themes
- Klasses
- Kviews
- メッセージ
- Notes (メモ)
- 通知

Kustomer API v1

エンティティ	フィールド	データ型	サポートされている演算子
Brands	id	String	該当なし
	名前	String	該当なし
	iconUrl	String	該当なし
	createdAt	DateTime	該当なし
	updatedAt	DateTime	該当なし
	modifiedAt	DateTime	該当なし
	デフォルト	ブール値	該当なし
[Cards] (カード)	id	String	該当なし
	名前	String	該当なし
	createdAt	DateTime	該当なし
	updatedAt	DateTime	該当なし
	modifiedAt	DateTime	該当なし
	description	文字列	該当なし
	url	String	該当なし
	contexts	リスト	該当なし
Chat Settings	id	String	該当なし
	settingsVersion	整数	該当なし
	widgetType	String	該当なし
	version	整数	該当なし

エンティティ	フィールド	データ型	サポートされている演算子
	teamName	String	該当なし
	greeting	String	該当なし
	autoreply	String	該当なし
	embedIconUrl	String	該当なし
	embedIconColor	String	該当なし
	fallbackEmailSubject	String	該当なし
	fallbackEmailIntroduction	String	該当なし
	有効	ブール値	該当なし
	outboundChatEnabled	ブール値	該当なし
	updatedAt	DateTime	該当なし
	modifiedAt	DateTime	該当なし
	offhoursMessage	String	該当なし
	offhoursImageUrl	String	該当なし
	closableChat	ブール値	該当なし
	noHistory	ブール値	該当なし
	disableAttachments	ブール値	該当なし
	volumeControl	Struct	該当なし
	singleSessonChat	ブール値	該当なし
	showTypingIndicatorWeb	ブール値	該当なし

エンティティ	フィールド	データ型	サポートされている演算子
Companies	id	String	該当なし
	名前	String	=、!=、CONTAINS
	createdAt	DateTime	該当なし
	updatedAt	DateTime	該当なし
	modifiedAt	DateTime	=, !=, <, <=, >, >=, BETWEEN
	tags	リスト	該当なし
	domains	リスト	該当なし
	emails	リスト	該当なし
	phones	リスト	該当なし
	whatsapps	リスト	該当なし
	socials	リスト	該当なし
	urls	リスト	該当なし
	ロケーション	リスト	該当なし
	roleGroupVersions	リスト	該当なし
	rev	整数	該当なし
会話	id	String	該当なし
	名前	String	=、!=、CONTAINS
	プレビュー	String	該当なし
	チャンネル	リスト	該当なし

エンティティ	フィールド	データ型	サポートされている演算子
	ステータス	String	=、!=、CONTAINS
	messageCount	整数	=, !=, >, >=, <, <=
	noteCount	整数	=, !=, >, >=, <, <=
	satisfaction	整数	=, !=, >, >=, <, <=
	satisfactionLevel	Struct	該当なし
	createdAt	DateTime	=, !=, <, <=, >, >=, BETWEEN
	updatedAt	DateTime	=, !=, <, <=, >, >=, BETWEEN
	modifiedAt	DateTime	=, !=, <, <=, >, >=, BETWEEN
	lastActivityAt	DateTime	該当なし
	スパム	ブール値	該当なし
	ended	ブール値	=, !=
	endedAt	DateTime	=, !=, <, <=, >, >=, BETWEEN
	endedReason	String	CONTAINS
	endedByType	String	該当なし
	importedAt	String	該当なし
	tags	リスト	該当なし
	suggestedTags	リスト	該当なし

エンティティ	フィールド	データ型	サポートされている演算子
	感情	String	該当なし
	予測	リスト	該当なし
	suggestedShortcuts	リスト	該当なし
	firstMessageIn	Struct	該当なし
	firstMessageOut	Struct	該当なし
	lastMessageIn	Struct	該当なし
	lastMessageOut	Struct	該当なし
	lastMessageAt	DateTime	=, !=, <, <=, >, >=, BETWEEN
	lastMessageUnrespondedTo	Struct	該当なし
	lastMessageUnrespondedToSinceLastDone	Struct	該当なし
	assignedUsers	リスト	該当なし
	assignedTeams	リスト	該当なし
	firstResponse	Struct	該当なし
	firstResponseSinceLastDone	Struct	該当なし
	lastResponse	Struct	該当なし
	firstDone	Struct	該当なし
	lastDone	Struct	該当なし

エンティティ	フィールド	データ型	サポートされている演算子
	direction	String	=、!=、CONTAINS
	lastMessageDirection	String	該当なし
	outboundMessageCount	整数	該当なし
	inboundMessageCount	整数	該当なし
	rev	整数	該当なし
	priority	整数	=, !=, >, >=, <, <=
	roleGroupVersions	リスト	該当なし
	accessOverride	リスト	該当なし
	assistant	Struct	該当なし
	phase	String	該当なし
	スキル	リスト	該当なし
	matchedTimeBasedRules	リスト	該当なし
顧客	id	String	該当なし
	名前	String	=、!=、CONTAINS
	displayName	String	該当なし
	displayColor	String	該当なし
	displayIcon	String	該当なし
	externalId	String	=、!=、CONTAINS

エンティティ	フィールド	データ型	サポートされている演算子
	externalIds	リスト	該当なし
	sharedExternalIds	リスト	該当なし
	emails	リスト	該当なし
	sharedEmails	リスト	該当なし
	phones	リスト	該当なし
	sharedPhones	リスト	該当なし
	whatsapps	リスト	該当なし
	facebookIds	リスト	該当なし
	instagramIds	リスト	該当なし
	socials	リスト	該当なし
	sharedSocials	リスト	該当なし
	urls	リスト	該当なし
	ロケーション	リスト	該当なし
	activeUsers	リスト	該当なし
	watchers	リスト	該当なし
	recentLocation	Struct	該当なし
	サイト	String	=、!=、CONTAINS
	timeZone	String	該当なし
	gender	String	=、!=、CONTAINS

エンティティ	フィールド	データ型	サポートされている演算子
	createdAt	DateTime	=, !=, <, <=, >, >=, BETWEEN
	updatedAt	DateTime	=, !=, <, <=, >, >=, BETWEEN
	modifiedAt	DateTime	=, !=, <, <=, >, >=, BETWEEN
	lastActivityAt	DateTime	該当なし
	削除済み	ブール値	該当なし
	lastConversation	Struct	該当なし
	conversationCounts	Struct	該当なし
	プレビュー	Struct	該当なし
	tags	リスト	該当なし
	progressiveStatus	String	=、!=、CONTAINS
	verified	ブール値	該当なし
	rev	整数	該当なし
	recentItems	リスト	該当なし
	defaultLang	String	=、!=、CONTAINS
	satisfactionLevel	Struct	該当なし
	roleGroupVersions	リスト	該当なし
	accessOverride	リスト	該当なし
	companyName	String	該当なし

エンティティ	フィールド	データ型	サポートされている演算子
	firstName	String	該当なし
	lastName	String	該当なし
Customer Searches Pinned	id	String	該当なし
	検索	String	該当なし
	createdAt	DateTime	該当なし
Customer Searches Positions	id	String	該当なし
	positions	リスト	該当なし
	children	リスト	該当なし
	createdAt	DateTime	該当なし
	updatedAt	DateTime	該当なし
	modifiedAt	DateTime	該当なし
	rev	整数	該当なし
Email Hooks	id	String	該当なし
	description	文字列	該当なし
	デバッグ	ブール値	該当なし
	email	String	該当なし
	eventName	String	該当なし
	title	String	該当なし
	ハッシュ	String	該当なし
	キー	String	該当なし

エンティティ	フィールド	データ型	サポートされている演算子
	createdAt	DateTime	該当なし
	modifiedAt	DateTime	該当なし
	updatedAt	DateTime	該当なし
Web Hooks	id	String	該当なし
	description	文字列	該当なし
	eventName	String	該当なし
	ハッシュ	String	該当なし
	url	String	該当なし
	createdAt	DateTime	該当なし
	modifiedAt	DateTime	該当なし
	updatedAt	DateTime	該当なし
	title	String	該当なし
	version	整数	該当なし
	デバッグ	ブール値	該当なし
KB Articles	id	String	該当なし
	ハッシュ	String	該当なし
	title	String	該当なし
	ソース	String	該当なし
	ステータス	String	該当なし
	scope	String	該当なし

エンティティ	フィールド	データ型	サポートされている演算子
	createdAt	DateTime	該当なし
	updatedAt	DateTime	該当なし
	削除済み	ブール値	該当なし
	deletedAt	DateTime	該当なし
	modifiedAt	DateTime	該当なし
	publishedAt	DateTime	該当なし
	tags	リスト	該当なし
	カテゴリ	リスト	該当なし
	knowledgeBases	リスト	該当なし
	metaTitle	String	該当なし
	metaDescription	String	該当なし
	metaKeywords	リスト	該当なし
	langVersions	Struct	該当なし
	latestLangs	Struct	該当なし
KB Categories	id	String	該当なし
	ハッシュ	String	該当なし
	createdAt	DateTime	該当なし
	modifiedAt	DateTime	該当なし
	updatedAt	DateTime	該当なし
	published	ブール値	該当なし

エンティティ	フィールド	データ型	サポートされている演算子
	positions	リスト	該当なし
	categoryPositions	リスト	該当なし
	ルート	ブール値	該当なし
	langs	Struct	該当なし
KB Forms	id	String	該当なし
	名前	String	該当なし
	slug	String	該当なし
	ハッシュ	String	該当なし
	本文	String	該当なし
	layout	リスト	該当なし
	layoutV2	リスト	該当なし
	componentsV2	Struct	該当なし
	条件	Struct	該当なし
	advanced	ブール値	該当なし
	createdAt	DateTime	該当なし
	updatedAt	DateTime	該当なし
	publishedAt	DateTime	該当なし
	modifiedAt	String	該当なし
	published	ブール値	該当なし
	snippets	リスト	該当なし

エンティティ	フィールド	データ型	サポートされている演算子
	recaptcha	ブール値	該当なし
	klass	String	該当なし
	チャンネル	String	該当なし
	deflection	ブール値	該当なし
	formHookEnabled	ブール値	該当なし
	replyFrom	String	該当なし
	wcag	ブール値	該当なし
KB Routes	id	String	該当なし
	url	String	該当なし
	routableType	String	該当なし
	routableId	String	該当なし
	createdAt	DateTime	該当なし
	updatedAt	DateTime	該当なし
	modifiedAt	DateTime	該当なし
KB Tags	id	String	該当なし
	名前	String	該当なし
	createdAt	DateTime	該当なし
	updatedAt	DateTime	該当なし
	modifiedAt	DateTime	該当なし
KB Templates	id	String	該当なし

エンティティ	フィールド	データ型	サポートされている演算子
	title	String	該当なし
	description	文字列	該当なし
	beta	ブール値	該当なし
	マニフェスト	Struct	該当なし
	jsxSnippets	リスト	該当なし
	画像	リスト	該当なし
	version	String	該当なし
	createdAt	DateTime	該当なし
	updatedAt	DateTime	該当なし
KB Themes	id	String	該当なし
	名前	String	該当なし
	ACTIVE	ブール値	該当なし
	デフォルト	ブール値	該当なし
	lastfileUpdatedAt	DateTime	該当なし
	カスタム	ブール値	該当なし
	ステータス	String	該当なし
	templateVersionId	String	該当なし
	templateTitle	String	該当なし
	templateVersion	String	該当なし
	マニフェスト	Struct	該当なし

エンティティ	フィールド	データ型	サポートされている演算子
	configSnippets	リスト	該当なし
	jsxSnippets	リスト	該当なし
	createdAt	DateTime	該当なし
	updatedAt	DateTime	該当なし
	modifiedAt	DateTime	該当なし
	rev	整数	該当なし
Klasses	id	String	該当なし
	名前	String	該当なし
	icon	String	該当なし
	color	String	該当なし
	appDisabled	ブール値	該当なし
	ステータス	String	該当なし
	updatedAt	DateTime	該当なし
	createdAt	DateTime	該当なし
	s3DataUrl	String	該当なし
KViews	id	String	該当なし
	リソース	String	該当なし
	テンプレート	String	該当なし
	context	String	該当なし
	meta	Struct	該当なし

エンティティ	フィールド	データ型	サポートされている演算子
	appDisabled	ブール値	該当なし
	有効	ブール値	該当なし
	advanced	ブール値	該当なし
	layout	リスト	該当なし
	コンポーネント	Struct	該当なし
	条件	Struct	該当なし
	rev	整数	該当なし
	createdAt	DateTime	該当なし
	modifiedAt	DateTime	該当なし
	updatedAt	DateTime	該当なし
通知	id	String	該当なし
	名前	String	該当なし
	ステータス	String	該当なし
	イベント	Struct	該当なし
	createdAt	DateTime	該当なし
	updatedAt	DateTime	該当なし
メッセージ	id	String	該当なし
	externalId	String	該当なし
	チャンネル	String	=、!=、CONTAINS
	アプリケーション	String	該当なし

エンティティ	フィールド	データ型	サポートされている演算子
	size	整数	=, !=, >, >=, <, <=
	direction	String	=、!=、CONTAINS
	プレビュー	String	該当なし
	subject	String	該当なし
	meta	Struct	該当なし
	ステータス	String	=、!=、CONTAINS
	directionType	String	=、!=、CONTAINS
	assignedTeams	リスト	該当なし
	assignedUsers	リスト	該当なし
	errorAt	DateTime	=, !=, <, <=, >, >=, BETWEEN
	自動	ブール値	=, !=
	sentAt	DateTime	=, !=, <, <=, >, >=, BETWEEN
	createdAt	DateTime	=, !=, <, <=, >, >=, BETWEEN
	updatedAt	DateTime	該当なし
	modifiedAt	DateTime	該当なし
	redacted	ブール値	該当なし
	createdByTeams	リスト	該当なし
	rev	整数	該当なし

エンティティ	フィールド	データ型	サポートされている演算子
	reactions	リスト	該当なし
	intentDetections	リスト	該当なし
メモ	id	String	該当なし
	本文	String	CONTAINS
	createdAt	DateTime	=, !=, <, <=, >, >=, BETWEEN
	updatedAt	DateTime	=, !=, <, <=, >, >=, BETWEEN
	modifiedAt	DateTime	=, !=, <, <=, >, >=, BETWEEN
	createdByTeams	リスト	該当なし

パーティショニングクエリ

フィールドベースのパーティション分割

Spark で同時実行を使用する場合は、追加の Spark オプション `PARTITION_FIELD`、`LOWER_BOUND`、`UPPER_BOUND`、および `NUM_PARTITIONS` を指定できます。これらのパラメータを使用すると、元のクエリは Spark タスクで同時に実行できるサブクエリの `NUM_PARTITIONS` の数に分割されます。

- `PARTITION_FIELD`: クエリのパーティション化に使用するフィールドの名前。
- `LOWER_BOUND`: 選択したパーティションフィールドの包括的な下限値。

  DateTime フィールドでは、ISO 形式の値を受け入れます。

  有効な値の例:

```
"2023-01-15T11:18:39.205Z"
```

- UPPER_BOUND: 選択したパーティションフィールドの排他的上限値。
- NUM_PARTITIONS: パーティション数。

エンティティごとのパーティション分割フィールドのサポートの詳細は、次の表にまとめられています。

エンティティ名	パーティション分割フィールド	データ型
Companies	modifiedAt	DateTime
会話	createdAt、updatedAt、modifiedAt、endedAt、lastMessageAt	DateTime
	messageCount、noteCount	BigInteger
	priority	整数
顧客	createdAt、updatedAt、modifiedAt	DateTime
メッセージ	errorAt、sentAt、createdAt	DateTime
	size	BigInteger
メモ	createdAt、updatedAt、modifiedAt	DateTime

例:

```
Kustomer_read = glueContext.create_dynamic_frame.from_options(
    connection_type="kustomer",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "conversation",
        "API_VERSION": "v1",
        "PARTITION_FIELD": "createdAt"
        "LOWER_BOUND": "2023-01-15T11:18:39.205Z"
```

```
        "UPPER_BOUND": "2023-02-15T11:18:39.205Z"
        "NUM_PARTITIONS": "2"
    }
```

Kustomer 接続のオプション

Kustomer の接続オプションは次のとおりです。

- ENTITY_NAME(文字列) – (必須) 読み取りに使用されます。Kustomer のオブジェクトの名前。
- API_VERSION(文字列) – (必須) 読み取りに使用されます。使用する Kustomer Rest API バージョン。
- SELECTED_FIELDS(List<String>) – Default: empty(SELECT *). 読み込みに使用されます。オブジェクトに選択する列。
- FILTER_PREDICATE(文字列) – デフォルト: 空 読み込みに使用されます。Spark SQL 形式である必要があります。
- QUERY(文字列) - デフォルト: 空 読み込みに使用されます。完全な Spark SQL クエリ。
- PARTITION_FIELD (文字列) – 読み取りに使用されます。クエリをパーティション化するために使用するフィールド。
- LOWER_BOUND (文字列)– 読み取りに使用されます。選択したパーティションフィールドの包括的な下限値。
- UPPER_BOUND (文字列) – 読み取りに使用されます。選択したパーティションフィールドの排他的上限値。
- NUM_PARTITIONS(整数) – デフォルト: 1。読み込みに使用されます。読み取り用のパーティションの数。
- INSTANCE_URL (文字列) – (必須) 読み取りに使用されます。Kustomer インスタンス URL。

Kustomer の制限事項

Kustomer の制限事項または注意事項は次のとおりです。

- Kustomer API ドキュメントではエンドポイントが宣言されていないため、Customer Searches エンティティはサポートされていません。
- Klasses エンティティでのフィルタリングと増分転送のサポートはありません。
- 並べ替えは、1 回のリクエストで複数の適用可能なフィールドで使用できます。

ただし、複数のフィールドでの並べ替え機能は、一部の組み合わせで SaaS エンドからの一貫性ない動作が確認されています。誤ったソート結果を示す「n」個の組み合わせが存在する可能性があるため、予測できません。例：

`Customers` エンティティの場合、`progressiveStatus desc, name asc` による並べ替えでは正しいソート結果が得られません。`progressiveStatus` の順序に基づいてのみソートされます。このような動作が確認された場合は、単一のフィールドを使用して並べ替えができます。

- 「id」フィールドで並べ替えを行う場合は、クエリパラメータとして `Conversations` および `Messages` エンティティのみがサポートされます。例: https://api.kustomerapp.com/v1/conversations?sort=desc (これにより、結果が「id」で降順にソートされます)。

  また、他のフィールドに対する他のフィルタリングまたは並べ替えは、API エンドポイントが POST https://api.kustomerapp.com/v1/customers/search として設定された POST リクエスト本文に変換されます。`Conversations` および `Messages` で「id」による並べ替えをサポートできるようにするには、id による並べ替えのみが存在するか、他のフィルタリングや並べ替えが他の適用可能なフィールドに存在する必要があります。
- Kustomer では、リクエストがフィルタリングされたかどうかに関係なく、最大 10,000 件のレコードを取得できます。この制限により、10,000 件を超えるレコードを保持するエンティティのデータが失われます。これを部分的に軽減するための回避策は 2 つあります。
  - フィルターを適用して、特定のレコードセットを取得します。
  - フィルター適用後のレコードが 10,000 件を超える場合は、新しい後続のリクエストに続きのフィルター値を適用するか、フィルターに範囲を適用します。例：

    最初のリクエストの filterExpression: `modifiedAt >= 2022-03-15T05:26:23.000Z and modifiedAt < 2023-03-15T05:26:23.000Z`

    これにより、10,000 件のレコードの制限が使い果たされると仮定します。

    filterExpression を使用して別のリクエストをトリガーできます: `modifiedAt >= 2023-03-15T05:26:23.000Z`
- SaaS の動作として、Kustomer の `CONTAINS` 演算子は完全な単語の一致のみをサポートし、単語内の部分一致はサポートしません。例えば、「body CONTAINS 'test record'」は「body」フィールドに「test」があるレコードと一致します。ただし、「body CONTAINS 'test'」は、「body」フィールドに「testAnotherRecord」があるレコードと一致しません。

LinkedIn に対する接続

LinkedIn は、さまざまなスポンサー付き投稿やその他の方法を通じて LinkedIn のソーシャルネットワークへのアクセスを提供する有料マーケティングツールです。LinkedIn は、B2B 企業がリードの獲得、オンライン認知度の向上、コンテンツの共有などを行うための強力なマーケティングツールです。

トピック

- LinkedIn の AWS Glue サポート
- 接続を作成および使用するための API オペレーションを含むポリシー
- LinkedIn の設定
- LinkedIn 接続の設定
- LinkedIn エンティティからの読み取り
- LinkedIn 接続オプション
- LinkedIn アカウントの作成
- 制限

LinkedIn の AWS Glue サポート

AWS Glue は、次のように LinkedIn をサポートします。

ソースとしてサポートされていますか?

はい。AWS Glue ETL ジョブを使用して、LinkedIn からデータをクエリできます。

ターゲットとしてサポートされていますか?

いいえ。

サポートされている LinkedIn API バージョン

[202406 (2024 年 6 月)]

接続を作成および使用するための API オペレーションを含むポリシー

次のサンプル ポリシーで、接続の作成と使用に必要な AWS の権限について説明します。新しいロールを作成する場合は、以下を含むポリシーを作成します。

```
{
```

```
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:ListConnectionTypes",
                "glue:DescribeConnectionType",
                "glue:RefreshOAuth2Tokens",
                "glue:ListEntities",
                "glue:DescribeEntity"
            ],
            "Resource": "*"
        }
    ]
}
```

上記の方法を使用しない場合は、代わりに次のマネージド IAM ポリシーを使用します。

- AWSGlueServiceRole – さまざまな AWS Glue プロセスを実行するために必要なリソースへのアクセス権をユーザーに代わって付与します。これらのリソースには、AWS Glue、Amazon S3、IAM、CloudWatch Logs、および Amazon EC2 が含まれます。このポリシーで指定されたリソースの命名規則に従った場合、AWS Glue プロセスは必要なアクセス権限を使用できます。このポリシーは、通常、クローラ、ジョブ、開発エンドポイントを定義するときに指定されたロールにアタッチされます。
- AWSGlueConsoleFullAccess — ポリシーがアタッチされているアイデンティティが AWS マネージメントコンソールを使用するときは、AWS Glue リソースへのフルアクセスを許可します。このポリシーで指定されたリソースの命名規則に従った場合、ユーザーは完全なコンソール機能を使用できます。このポリシーは、通常 AWS Glue コンソールのユーザーにアタッチされています。

LinkedIn の設定

AWS Glue を使用して LinkedIn から転送できるようにするには、次の要件を満たす必要があります。

最小要件

- お持ちのアカウントは LinkedIn である。アカウントの作成の詳細については、「LinkedIn アカウントの作成」を参照してください。
- LinkedIn アカウントで API アクセスが有効になっている。

- OAuth2 API 統合が LinkedIn アカウントで作成された。この統合により、AWS Glue がお客様のアカウントに対して認証呼び出しを行うときに、お客様のデータに安全にアクセスするために使用するクライアントの認証情報が提供されます。詳細については、「the section called “LinkedIn アカウントの作成”」を参照してください。

これらの要件を満たすと、LinkedIn アカウントに AWS Glue を接続する準備が整います。一般的な接続では、LinkedIn で他に何もする必要はありません。

LinkedIn 接続の設定

LinkedIn は OAuth2 の AUTHORIZATION_CODE グラントタイプをサポートしています。

このグラントタイプは、ユーザーを認証するためにサードパーティーの認証サーバーにユーザーをリダイレクトすることから、「three-legged」の OAuth と見なされます。ユーザーは、LinkedIn で独自の接続アプリを作成し、AWS Glue コンソールを介して接続を作成するときに独自のクライアント ID とクライアントシークレットを指定することを選択できます。このシナリオでは、引き続き LinkedIn にリダイレクトされてログインし、リソースへアクセスするために AWS Glue を承認します。

このグラントタイプは、更新トークンとアクセストークンの両方になります。このアクセストークンは、作成から 60 日後に期限切れになります。新しいアクセストークンは、更新トークンを使用して取得できます。

Authorization Code OAuth フロー用の接続アプリケーションの作成に関する LinkedIn の公開ドキュメントについては、「認証コードフロー (3-legged OAuth)」を参照してください。

LinkedIn 接続の設定

1. AWS Secrets Manager で、次の詳細を含むシークレットを作成します。

   - カスタマーマネージド接続アプリケーションの場合 – シークレットには、USER_MANAGED_CLIENT_APPLICATION_CLIENT_SECRET を使用して接続されたアプリケーションのコンシューマーシークレットをキーとして含める必要があります。
   - AWS マネージド接続アプリの場合 – 空のシークレット、または一時的な値を持つシークレット。

     > **Note**
     >
     > AWS Glue で接続ごとにシークレットを作成する必要があります。

2. AWS Glue Studio で、以下の手順に従って [データ接続] の下に接続を作成します。

   1. [接続タイプ] を選択する際に、[LinkedIn] を選択します。
   2. LinkedIn 環境を指定します。
   3. AWS Glue が次のアクションを担い、その権限を持つことができる IAM ロールを選択します。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:DescribeSecret",
                "secretsmanager:GetSecretValue",
                "secretsmanager:PutSecretValue",
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterface",
                "ec2:DeleteNetworkInterface",
            ],
            "Resource": "*"
        }
    ]
}
```

   4. AWS Glue でこの接続に使用する `secretName` を選択して、トークンを配置します。
   5. ネットワークを使用する場合は、[ネットワークオプション] を選択します。

3. AWS Glue ジョブに関連付けられている IAM ロールに `secretName` を読み取るアクセス許可を付与します。

LinkedIn エンティティからの読み取り

前提条件

読み取り元の LinkedIn オブジェクト。使用可能なエンティティを確認するには、以下のサポートされているエンティティの表を参照してください。

サポートされているエンティティ

エンティティ	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
広告アカウント	はい	はい	はい	はい	いいえ
キャンペーン	はい	はい	はい	はい	いいえ
キャンペーングループ	はい	はい	はい	はい	いいえ
クリエイティブ	はい	はい	はい	はい	いいえ
広告分析	はい	いいえ	いいえ	はい	いいえ
広告分析 すべての AdAcocunts	はい	いいえ	いいえ	はい	いいえ
広告分析 すべてのキャンペーン	はい	いいえ	いいえ	はい	いいえ
広告分析 すべての CampaignGroups	はい	いいえ	いいえ	はい	いいえ
広告分析 すべての AdCreatives	はい	いいえ	いいえ	はい	いいえ
統計の共有	はい	いいえ	いいえ	はい	いいえ
ページ統計	はい	いいえ	いいえ	はい	いいえ

エンティティ	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
フォロワー統計	はい	いいえ	いいえ	はい	いいえ

例

```
netsuiteerp_read = glueContext.create_dynamic_frame.from_options(
    connection_type="linkedin",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "adaccounts",
        "API_VERSION": "202406"
    }
)
```

LinkedIn エンティティとフィールドの詳細

フィールドデータタイプ	サポートされているフィルター演算子
String	=
DateTime	BETWEEN, =
数値	=
ブール値	=

LinkedIn 接続オプション

LinkedIn の接続オプションは次のとおりです。

- ENTITY_NAME(文字列) – (必須) 読み取り / 書き込みに使用されます。LinkedIn のオブジェクトの名前。例えば、adAccounts などです。
- API_VERSION(文字列) – (必須) 読み取り / 書き込みに使用されます。使用する LinkedIn M Rest API バージョン。LinkedIn は現在バージョン 202406 のみをサポートしているため、この値は 202406 になります。

- SELECTED_FIELDS(List<String>) – Default: empty(SELECT *). 読み込みに使用されます。選択したエンティティに対して選択する列。
- FILTER_PREDICATE(文字列) – デフォルト: 空 読み込みに使用されます。Spark SQL 形式である必要があります。
- QUERY(文字列) – デフォルト: 空 読み込みに使用されます。完全な Spark SQL クエリ。

LinkedIn アカウントの作成

LinkedIn アプリと OAuth 認証情報の作成

1. [LinkedIn 開発者ネットワーク] ページに移動し、LinkedIn アカウントの認証情報を使用してログインします。
2. [マイアプリ] ページに移動し、[アプリケーションを作成] を選択して新しい LinkedIn アプリを作成します。
3. アプリ登録フォームに次の詳細を入力します。
   - [会社名] – 既存の会社を選択するか、新しい会社を作成します。
   - [名前] – アプリケーション名を入力します。
   - [説明] – アプリケーションの説明を入力します。
   - [アプリケーションロゴ] – アプリケーションロゴとしてイメージファイルを選択します。
   - [アプリケーションの使用] – アプリケーションの使用を選択します。
   - [ウェブサイト URL] – アプリケーションに関する詳細情報を含むウェブサイトの URL を入力します。
   - [勤務先の E メール] – 勤務先の E メールアドレスを入力します。
   - [勤務先電話番号] – 勤務先の電話番号を入力します。
   - [LinkedIn API 利用規約] – 読んで同意します。
4. アプリ登録フォームの入力が完了したら、[送信] を選択します。

   [認証] ページにリダイレクトされ、認証キー (クライアント ID とクライアントシークレット) およびその他の関連する詳細が表示されます。
5. ウェブアプリケーションが LinkedIn アカウントからユーザーの E メールアドレスにアクセスする必要がある場合は、r_emailaddress アクセス許可を選択します。さらに、LinkedIn アプリケーションの承認済みリダイレクト URL を指定できます。

LinkedIn アカウントでのページの作成

1. LinkedIn 開発者製品 に移動します。
2. [LinkedIn 開発者製品] ページの右上隅で、[マイアプリ] を選択します。
3. [マイアプリ] ページの右上隅で、[アプリを作成] を選択します。
4. [アプリを作成] ページの [アプリ名] フィールドに、アプリの名前を入力します。
5. [LinkedIn ページ] フィールドに、会社のページ名または URL を入力します。

   > **Note**
   >
   > LinkedIn ページがない場合は、[新しい LinkedIn を作成] を選択してページを作成できます。

6. [プライバシーポリシー URL] フィールドに、プライバシーポリシー URL を入力します。
7. [ロゴをアップロード] を選択して、ユーザーがアプリで認可するときに表示するイメージをアップロードします。
8. [法的契約] セクションで、[これらの条項を読み、同意しました] を選択します。
9. [アプリの作成] を選択します。

   新しいアプリが作成され、[マイアプリ] タブに表示されます。

LinkedIn でのキャンペーン広告の公開

1. [キャンペーンマネージャー] にログインします。
2. 既存の [キャンペーングループ] を選択するか、[作成] を選択して新しいキャンペーングループを作成します。
3. 目標を選択します。
4. グループ、予算、スケジュールを選択します。
5. ターゲットオーディエンスを構築します。
6. 広告形式を選択します。
7. 予算とスケジュールを選択します。
8. 広告をセットアップします (複数可)。
9. 確認して起動します。

制限

Analytics フィールド `ad_analytics_all_adAccounts`、`ad_analytics_all_campaigns`、`ad_analytics_all_campaig` および `ad_analytics_all_adCreatives` では、レコードを取得するためにフィルターが必要です。

Mailchimp への接続

Mailchimp は、クライアント、顧客、その他の利害関係者の管理と対話に役立つオールインワンのマーケティングプラットフォームです。マーケティングへのアプローチは、健全な問い合わせ管理プラクティス、美しく設計された E メール、独自の自動ワークフロー、強力なデータ分析に重点を置いています。Mailchimp ユーザーの場合は、Mailchimp アカウントに AWS Glue を接続できます。その後、Mailchimp を ETL ジョブのデータソースとして使用できます。これらのジョブを実行して、Mailchimp と AWS サービス、またはその他のサポートされているアプリケーション間でデータを転送します。

トピック

- Mailchimp の AWS Glue サポート
- 接続を作成および使用するための API オペレーションを含むポリシー
- Mailchimp の設定
- Mailchimp 接続の設定
- Mailchimp エンティティからの読み取り
- Mailchimp 接続のオプション
- Mailchimp アカウントの作成
- 制限

Mailchimp の AWS Glue サポート

AWS Glue は、次のように Mailchimp をサポートします。

ソースとしてサポートされていますか?

はい。AWS Glue ETL ジョブを使用して、Mailchimp からデータをクエリできます。

ターゲットとしてサポートされていますか?

いいえ。

サポートされている Mailchimp API バージョン

3.0

接続を作成および使用するための API オペレーションを含むポリシー

次のサンプル ポリシーで、接続の作成と使用に必要な AWS の権限について説明します。新しいロールを作成する場合は、以下を含むポリシーを作成します。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:ListConnectionTypes",
                "glue:DescribeConnectionType",
                "glue:RefreshOAuth2Tokens",
                "glue:ListEntities",
                "glue:DescribeEntity"
            ],
            "Resource": "*"
        }
    ]
}
```

上記の方法を使用しない場合は、代わりに次のマネージド IAM ポリシーを使用します。

- AWSGlueServiceRole – さまざまな AWS Glue プロセスを実行するために必要なリソースへのアクセス権をユーザーに代わって付与します。これらのリソースには、AWS Glue、Amazon S3、IAM、CloudWatch Logs、および Amazon EC2 が含まれます。このポリシーで指定されたリソースの命名規則に従った場合、AWS Glue プロセスは必要なアクセス権限を使用できます。このポリシーは、通常、クローラ、ジョブ、開発エンドポイントを定義するときに指定されたロールにアタッチされます。
- AWSGlueConsoleFullAccess — ポリシーがアタッチされているアイデンティティが AWS マネージメントコンソールを使用するときは、AWS Glue リソースへのフルアクセスを許可します。このポリシーで指定されたリソースの命名規則に従った場合、ユーザーは完全なコンソール機能を使用できます。このポリシーは、通常 AWS Glue コンソールのユーザーにアタッチされています。

Mailchimp の設定

AWS Glue を使用して Mailchimp から転送できるようにするには、次の要件を満たす必要があります。

最小要件

- メールアドレスとパスワードが設定された Mailchimp アカウントがある。アカウントの作成の詳細については、「Creating a Mailchimp account」を参照してください。
- AWS Glue へのサービスアクセスで AWS アカウントを作成する必要があります。
- 次のいずれかのリソースが作成されていることを確認します。これらのリソースにより、AWS Glue がお客様のアカウントに対して認可呼び出しを行うときに、お客様のデータに安全にアクセスするために使用する認証情報が提供されます。
  - OAuth 2.0 認可をサポートするデベロッパーアプリケーション。デベロッパーアプリケーションの作成の詳細については、「Creating a Mailchimp account」を参照してください。

これらの要件を満たすと、Mailchimp アカウントに AWS Glue を接続する準備が整います。一般的な接続では、Mailchimp で他に何もする必要はありません。

Mailchimp 接続の設定

Mailchimp は、認可メカニズムとして次の 2 つのタイプをサポートしています。

- Mailchimp は `AUTHORIZATION_CODE` グラントタイプをサポートしています。
  - このグラントタイプは、ユーザーを認可するためにサードパーティーの認可サーバーにユーザーをリダイレクトすることから、「three-legged」の `OAuth` と見なされます。AWS Glue コンソール経由で接続を作成するときに使用されます。接続を作成するユーザーは、デフォルトで AWS Glue 所有の接続アプリケーションを使用できます。この場合、Mailchimp クライアント ID とクライアントシークレット以外の `OAuth` 関連情報を指定する必要はありません。AWS Glue コンソールでは、ユーザーが Mailchimp にリダイレクトされます。ここではユーザーがログインし、Mailchimp インスタンスにアクセスするためのリクエストされた権限を AWS Glue に許可する必要があります。
  - ユーザーは、Mailchimp で独自の接続アプリケーションを作成し、AWS Glue コンソールを介して接続を作成するときに独自のクライアント ID とクライアントシークレットを指定することを選択できます。このシナリオでも Mailchimp にリダイレクトされてログインし、リソースへアクセスするために AWS Glue を認可します。

- AUTHORIZATION_CODE OAuth フロー用の接続アプリケーションの作成に関する Mailchimp 公開ドキュメントについては、「Access Data on Behalf of Other Users with OAuth 2」を参照してください。
- カスタム認可 – カスタム認可に必要な API キーの生成に関する Mailchimp 公開ドキュメントについては、「About API Keys」を参照してください。

Mailchimp 接続を設定するには

1. AWS Secrets Manager で、次の詳細を含むシークレットを作成します。
   - OAuth 認可 - カスタマーマネージド接続アプリケーションの場合: シークレットには、USER_MANAGED_CLIENT_APPLICATION_CLIENT_SECRET を使用して接続されたアプリケーションのコンシューマーシークレットをキーとして含める必要があります。
   - カスタム認可 – カスタマーマネージド接続アプリケーションの場合: シークレットには、「api_key」を使用して接続されたアプリケーションのコンシューマーシークレットをキーとして含める必要があります。

   > **Note**
   >
   > AWS Glue で接続ごとにシークレットを作成する必要があります。

2. AWS Glue Studio で、以下の手順に従って [データ接続] の下に接続を作成します。
   a. [接続] で、[接続を作成] を選択します。
   b. [データソース] を選択するときは、[Mailchimp] を選択します。
   c. Mailchimp の instanceUrl を指定します。
   d. AWS Glue が次のアクションを担い、その権限を持つことができる IAM ロールを選択します。

```
{
    "Version": "2012-10-17",
        "Statement": [
            {
                "Effect": "Allow",
                "Action": [
                    "secretsmanager:DescribeSecret",
                    "secretsmanager:GetSecretValue",
                    "secretsmanager:PutSecretValue",
                    "ec2:CreateNetworkInterface",
                    "ec2:DescribeNetworkInterface",
```

```
                "ec2:DeleteNetworkInterface",
            ],
            "Resource": "*"
        }
    ]
}
```

e. Mailchimp に接続する認可タイプを選択します。

- 0Auth 認可の場合 – 接続先の Mailchimp のトークン URL、ユーザーマネージドクライアントアプリケーション ClientId を指定します。
- カスタム認可の場合 –認可タイプ CUSTOM を選択して Mailchimp に接続します。

f. AWS Glue でこの接続に使用する `secretName` を選択して、トークンを配置します。

g. ネットワークを使用する場合は、ネットワークオプションを選択します。

3. AWS Glue ジョブに関連付けられている IAM ロールに `secretName` を読み取るアクセス許可を付与します。
4. AWS Glue ジョブ設定で、[追加のネットワーク接続] として `connectionName` を指定します。

Mailchimp エンティティからの読み取り

前提条件

読み取り元の Mailchimp オブジェクト。使用可能なエンティティを確認するには、以下のサポートされているエンティティの表を参照してください。

サポートされているエンティティ

- Abuse-reports
- オートメーション
- キャンペーン
- Click-details
- Lists
- メンバー
- Open-details
- セグメント
- Stores
- Unsubscribed

エンティティ	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
Automation	あり	あり	あり	あり	あり
キャンペーン	いいえ	いいえ	いいえ	いいえ	いいえ
リスト	あり	はい	なし	あり	あり
Reports Abuse	いいえ	あり	なし	あり	あり
Reports Open	いいえ	あり	なし	あり	あり
Reports Click	あり	はい	なし	あり	あり
Reports Unsubscribe	いいえ	あり	なし	あり	あり
Segment	いいえ	あり	なし	あり	あり
Segment Members	あり	はい	なし	あり	不可
Stores	あり	あり	あり	はい	不可

例

```
mailchimp_read = glueContext.create_dynamic_frame.from_options(
            connection_type="mailchimp",
            connection_options={

                  "connectionName": "connectionName",

                  "ENTITY_NAME": "stores",
"INSTANCE_URL": "https://us14.api.mailchimp.com",
                  "API_VERSION": "3.0"
               })
```

Mailchimp エンティティとフィールドの詳細

- Abuse-reports
- オートメーション
- キャンペーン
- Click-details
- Lists
- メンバー
- Open-details
- セグメント
- Stores
- Unsubscribed

パーティショニングクエリ

Spark で同時実行を使用する場合は、追加の Spark オプション PARTITION_FIELD、LOWER_BOUND、UPPER_BOUND、および NUM_PARTITIONS を指定できます。これらのパラメータを使用すると、元のクエリは Spark タスクで同時に実行できるサブクエリの NUM_PARTITIONS の数に分割されます。

- PARTITION_FIELD: クエリのパーティション化に使用するフィールドの名前。
- LOWER_BOUND: 選択したパーティションフィールドの包括的な下限値。

  DateTime フィールドでは、ISO 形式の値を受け入れます。

  有効な値の例:

  ```
  "2024-07-01T00:00:00.000Z"
  ```

- UPPER_BOUND: 選択したパーティションフィールドの排他的上限値。
- NUM_PARTITIONS: パーティション数。

次の表に、エンティティパーティション分割フィールドのサポートの詳細を示します:

エンティティ名	パーティション分割フィールド	データ型

例:

```
read_read = glueContext.create_dynamic_frame.from_options(
    connection_type="mailchimp",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "automations",
        "API_VERSION": "3.0",
        "INSTANCE_URL": "https://us14.api.mailchimp.com",
        "PARTITION_FIELD": "create_time",
        "LOWER_BOUND": "2024-02-05T14:09:30.115Z",
        "UPPER_BOUND": "2024-06-07T13:30:00.134Z",
        "NUM_PARTITIONS": "3"
    }
```

Mailchimp 接続のオプション

Mailchimp の接続オプションは次のとおりです。

- ENTITY_NAME(文字列) – (必須) 読み取り / 書き込みに使用されます。Mailchimp のオブジェクトの名前。
- INSTANCE_URL(文字列) – (必須) 有効な Mailchimp インスタンス URL。
- API_VERSION(文字列) – (必須) 読み取りに使用されます。使用する Mailchimp Engage Rest API バージョン。例: 3.0。
- SELECTED_FIELDS(List<String>) – Default: empty(SELECT *). 読み込みに使用されます。オブジェクトに選択する列。
- FILTER_PREDICATE(文字列) – デフォルト: 空 読み込みに使用されます。Spark SQL 形式である必要があります。

- QUERY(文字列) - デフォルト: 空 読み込みに使用されます。完全な Spark SQL クエリ。
- PARTITION_FIELD(文字列) – 読み取りに使用されます。クエリをパーティション化するために使用するフィールド。
- LOWER_BOUND(文字列) - 読み取りに使用されます。選択したパーティションフィールドの包括的な下限値。
- UPPER_BOUND (文字列) - 読み取りに使用されます。選択したパーティションフィールドの排他的上限値。
- NUM_PARTITIONS(整数) – デフォルト: 1。読み込みに使用されます。読み取り用のパーティションの数。

Mailchimp アカウントの作成

1. Mailchimp ログインページに移動し、E メール ID とパスワードを入力して、[Sign up] を選択します。
2. Mailchimp からの確認 E メールを開き、確認リンクを選択してアカウントを確認します。

> **Note**
>
> アクティベーション E メールの受信にかかる時間は異なる場合があります。アクティベーション E メールを受信していない場合は、スパムフォルダをチェックし、アクティベーション E メールのトラブルシューティングのヒントをお読みください。Mailchimp は、admin@pottedplanter.com や security@example.com などのロールベースの E メールアドレスからのサインアップをブロックします。

アカウントに初めてログインすると、Mailchimp から必要な情報の入力を求められます。Mailchimp は、この情報を使用して、アカウントが利用規約に準拠していることを確認し、ユーザーおよびその会社のニーズに関連するガイダンスを提供します。

3. 情報を入力し、プロンプトに従ってアクティベーションプロセスを完了し、新しい Mailchimp アカウントで開始します。

OAuth2.0 アプリケーションの登録

1. Mailchimp ログインページに移動し、E メール ID とパスワードを入力し、[Log in] を選択します。

2. 右上隅の [User] アイコンを選択し、ドロップダウンメニューから [Account and billing] を選択します。
3. [Extras] を選択し、ドロップダウンメニューから [Registered apps] を選択します。
4. [Register An App] を選択します。
5. 次の詳細情報を入力します。

   - App name – アプリケーションの名前。
   - Company / Organization – 会社または組織の名前。
   - App website – アプリケーションのウェブサイト。
   - Redirect URI – リダイレクト URI パターンは、ログインフローが完了したときに Mailchimp がリダイレクト (リクエストされた場合) できる URI パス (またはカンマ区切りのパスのリスト) です。例えば、`https://ap-southeast-2\\.console\\.aws\\.amazon\\.com`
6. [Create] (作成) を選択します。
7. これで、[Client ID] と [Client Secret] が表示されます。これらをコピーして安全な場所に保存します。次に [完了] を選択します。

> **Note**
>
> クライアント ID とクライアントシークレット文字列は、AppFlow または AWS Glue を使用するときにこのコネクタとの接続を確立するために使用される認証情報です。

API キーの生成

1. Mailchimp ログインページに移動し、E メール ID とパスワードを入力し、[Log in] を選択します。
2. 右上隅の [User] アイコンを選択し、ドロップダウンメニューから [Account and billing] を選択します。
3. [Extras] を選択し、ドロップダウンメニューから [API keys] を選択します。
4. [Create A Key] を選択します。
5. キーの名前を入力し、[Generate Key] を選択します。

   次のページには、生成された API キーが表示されます。
6. キーをコピーして安全に保存し、[Done] を選択します。

制限

Mailchimp コネクタの制限事項は次のとおりです。

- フィルタリングは、Campaigns、Automations、Lists、Open Details、Members、および Segments エンティティでのみサポートされます。
- DateTime データ型フィールドでフィルターを使用する場合は、yyyy-mm-ddThh:MM:ssZ の形式で値を渡す必要があります。

Microsoft Teams への接続

Microsoft Teams は、Microsoft 365 内の共同ワークスペースであり、職場の会話、共同チームワーク、ビデオチャット、ドキュメント共有の中心的なハブとして機能し、統合されたツールスイートでワーカーの生産性を高めるように設計されています。

トピック

- Microsoft Teams の AWS Glue サポート
- 接続を作成および使用するための API オペレーションを含むポリシー
- Microsoft Teams の設定
- Microsoft Teams 接続の設定
- Microsoft Teams エンティティからの読み取り
- Microsoft Teams 接続オプションのリファレンス
- 制限
- 新しい Microsoft Teams アカウントを作成します。

Microsoft Teams の AWS Glue サポート

AWS Glue は、次のように Microsoft Teams をサポートします。

ソースとしてサポートされていますか?

はい。AWS Glue ETL ジョブを使用して、Microsoft Teams からデータをクエリできます。

ターゲットとしてサポートされていますか?

いいえ。

サポートされている Microsoft Teams API バージョン

v1。バージョン固有のエンティティサポートについては、「ソースに対応するエンティティ」を参照してください。

接続を作成および使用するための API オペレーションを含むポリシー

次のサンプル ポリシーで、接続の作成と使用に必要な AWS の権限について説明します。新しいロールを作成する場合は、以下を含むポリシーを作成します。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:ListConnectionTypes",
                "glue:DescribeConnectionType",
                "glue:RefreshOAuth2Tokens"
                "glue:ListEntities",
                "glue:DescribeEntity"
            ],
            "Resource": "*"
        }
    ]
}
```

また、次のマネージド IAM ポリシーを使用して、アクセスを許可できます。

- AWSGlueServiceRole – さまざまな AWS Glue プロセスを実行するために必要なリソースへのアクセス権をユーザーに代わって付与します。これらのリソースには、AWS Glue、Amazon S3、IAM、CloudWatch Logs、および Amazon EC2 が含まれます。このポリシーで指定されたリソースの命名規則に従った場合、AWS Glue プロセスは必要なアクセス権限を使用できます。このポリシーは、通常、クローラ、ジョブ、開発エンドポイントを定義するときに指定されたロールにアタッチされます。
- AWSGlueConsoleFullAccess — ポリシーがアタッチされているアイデンティティが AWS マネージメントコンソールを使用するときは、AWS Glue リソースへのフルアクセスを許可します。このポリシーで指定されたリソースの命名規則に従った場合、ユーザーは完全なコンソール機能を使用できます。このポリシーは、通常 AWS Glue コンソールのユーザーにアタッチされています。

Microsoft Teams の設定

AWS Glue を使用して Microsoft Teams からデータを転送できるようにするには、次の要件を満たす必要があります。

最小要件

- メールアドレスとパスワードが設定された Microsoft Teams デベロッパーアカウントがある。詳細については、「新しい Microsoft Teams アカウントを作成します。」を参照してください。
- Microsoft アカウントで OAuth2 アプリケーションのセットアップを完了している必要があります。このアプリケーションによって、お客様のアカウントに対して認可呼び出しを行う際にデータへの安全なアクセスのために AWS Glue が使用するクライアント ID とシークレット認証情報を提供します。詳細については、「新しい Microsoft Teams アカウントを作成します。」を参照してください。

これらの要件を満たすと、Microsoft Teams アカウントに AWS Glue を接続する準備が整います。一般的な接続では、Microsoft Teams で他に何もする必要はありません。

Microsoft Teams 接続の設定

Microsoft Teams では、認可メカニズムとして次の 2 つのタイプがサポートされています。

1. OAuth 認証: Microsoft Teams は、OAuth2 の AUTHORIZATION_CODE グラントタイプをサポートしています。
   - このグラントタイプは、ユーザーを認証するためにサードパーティーの認証サーバーにユーザーをリダイレクトすることから、「three-legged」の OAuth と見なされます。AWS Glue コンソール経由で接続を作成するときに使用されます。接続を作成するユーザーは、デフォルトで AWS Glue 所有の接続アプリケーションを使用できます。この場合、Microsoft Teams インスタンス URL 以外の OAuth 関連情報を指定する必要はありません。AWS Glue コンソールでは、ユーザーは Microsoft Teams にリダイレクトされます。ここではユーザーがログインし、Microsoft Teams インスタンスにアクセスするためのリクエストされた権限を AWS Glue に許可する必要があります。
   - ユーザーは、Microsoft Teams で独自の接続アプリケーションを作成し、AWS Glue コンソールを介して接続を作成するときに独自のクライアント ID とクライアントシークレットを指定することを選択できます。このシナリオでも Microsoft Teams にリダイレクトされてログインし、リソースへアクセスするために AWS Glue を認可します。

- このグラントタイプは、更新トークンとアクセストークンになります。アクセストークンは 1 時間有効で、更新トークンを使用することでユーザーとやり取りすることなく自動的に更新できます。
- 認証コード OAuth フロー用の接続アプリケーションの作成に関する Microsoft Teams の公開ドキュメントについては、「| Microsoft Learn」の 「Register an application with the Microsoft identity platform」を参照してください。

Microsoft Teams 接続を設定するには

1. AWS Secrets Manager で、次の詳細を含むシークレットを作成します。AWS Glue で接続ごとにシークレットを作成する必要があります。
   a. OAuth 認可の場合:
      - カスタマーマネージド接続アプリケーションの場合 – シークレットには、USER_MANAGED_CLIENT_APPLICATION_CLIENT_SECRET を使用して接続されたアプリケーションのコンシューマーシークレットをキーとして含める必要があります。
2. AWS Glue Studio で、以下の手順に従って [データ接続] の下に接続を作成します。
   a. [データ接続] の [接続を作成] を選択します。
   b. [データソース] を選択するときは、[Microsoft Teams] を選択します。
   c. Microsoft Teams の [テナント ID] を入力します。
   d. AWS Glue が次のアクションを担い、その権限を持つことができる IAM ロールを選択します。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:DescribeSecret",
                "secretsmanager:GetSecretValue",
                "secretsmanager:PutSecretValue",
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterface",
                "ec2:DeleteNetworkInterface",
            ],
            "Resource": "*"
        }
    ]
```

```
}
```

e. Microsoft Teams アプリケーションのユーザーマネージドクライアントアプリケーション ClientId を入力します。

f. AWS Glue でこの接続に使用する `secretName` を選択して、トークンを配置します。

g. ネットワークを使用する場合は、ネットワークオプションを選択します。

3. AWS Glue ジョブに関連付けられている IAM ロールに `secretName` を読み取るアクセス許可を付与します。[Next] を選択します。

4. AWS Glue ジョブ設定で、[追加のネットワーク接続] として `connectionName` を指定します。

Microsoft Teams エンティティからの読み取り

前提条件

- 読み取り元の Microsoft Teams オブジェクト。team または channel-message などのオブジェクト名が必要です。次の表に、サポートされているエンティティを示します。

ソースに対応するエンティティ

すべてのエンティティは API バージョン 1.0 でサポートされています。

エンティティ	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
チーム	いいえ	いいえ	なし	あり	不可
Team Members	あり	はい	なし	あり	あり
グループ	あり	あり	あり	あり	あり
Group Members	あり	はい	なし	あり	不可
チャンネル	あり	いいえ	なし	あり	あり

エンティティ	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
Channel Messages	いいえ	あり	なし	あり	不可
Channel Message Replies	いいえ	あり	なし	あり	不可
Channel Tabs	あり	いいえ	なし	あり	不可
Chats	あり	あり	あり	あり	あり
Calendar Events	あり	あり	あり	あり	あり

例

```
MicrosoftTeams_read = glueContext.create_dynamic_frame.from_options(
    connection_type="MicrosoftTeams",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "company",
        "API_VERSION": "v1.0"
    }
```

Microsoft Teams エンティティとフィールドの詳細

エンティティリスト:

- Team: https://docs.microsoft.com/en-us/graph/api/user-list-joinedteams?view=graph-rest-1.0
- Team-Member: https://docs.microsoft.com/en-us/graph/api/team-list-members?view=graph-rest-1.0
- Group: https://docs.microsoft.com/en-us/graph/api/group-list?view=graph-rest-1.0

- Group-Member: https://docs.microsoft.com/en-us/graph/api/group-list-members?view=graph-rest-1.0
- Channel: https://docs.microsoft.com/en-us/graph/api/channel-list?view=graph-rest-1.0
- Channel-Message: https://docs.microsoft.com/en-us/graph/api/channel-list-messages?view=graph-rest-1.0
- Channel-Message-Reply: https://docs.microsoft.com/en-us/graph/api/chatmessage-list-replies?view=graph-rest-1.0
- Channel-Tab: https://docs.microsoft.com/en-us/graph/api/channel-list-tabs?view=graph-rest-1.0
- Chat: https://docs.microsoft.com/en-us/graph/api/chat-list?view=graph-rest-1.0
- Calendar-Event: https://docs.microsoft.com/en-us/graph/api/group-list-events?view=graph-rest-1.0

パーティショニングクエリ

Spark で同時実行を使用する場合は、追加の Spark オプション PARTITION_FIELD、LOWER_BOUND、UPPER_BOUND、および NUM_PARTITIONS を指定できます。これらのパラメータを使用すると、元のクエリは Spark タスクで同時に実行できるサブクエリの NUM_PARTITIONS の数に分割されます。

- PARTITION_FIELD: クエリのパーティション化に使用するフィールドの名前。
- LOWER_BOUND: 選択したパーティションフィールドの包括的な下限値。

  日付については、Spark SQL クエリで使用される Spark の日付形式を受け入れます。有効な値の例: "2024-02-06"。
- UPPER_BOUND: 選択したパーティションフィールドの排他的上限値。
- NUM_PARTITIONS: パーティション数。

エンティティごとのパーティション分割フィールドのサポートの詳細は、以下の表にまとめられています。

エンティティ名	パーティション分割フィールド	データタイプ
Team Members	visibleHistoryStartDateTime	DateTime
グループ	createdDateTime	DateTime

エンティティ名	パーティション分割フィールド	データタイプ
チャンネル	createdDateTime	DateTime
Chats	createdDateTime、lastModifiedDateTime	DateTime
Calendar Events	createdDateTime、lastModifiedDateTime、originalStart	DateTime

例

```
microsoftteams_read = glueContext.create_dynamic_frame.from_options(
    connection_type="MicrosoftTeams",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "group",
        "API_VERSION": "v1.0",
        "PARTITION_FIELD": "createdDateTime"
        "LOWER_BOUND": "2022-07-13T07:55:27.065Z"
        "UPPER_BOUND": "2022-08-12T07:55:27.065Z"
        "NUM_PARTITIONS": "2"
    }
```

Microsoft Teams 接続オプションのリファレンス

Microsoft Teams の接続オプションは次のとおりです。

- ENTITY_NAME(文字列) – (必須) 読み取りに使用されます。Microsoft Teams のオブジェクトの名前。
- API_VERSION(文字列) – (必須) 読み取りに使用されます。使用する Microsoft Teams Rest API バージョン。例: v1.0。
- SELECTED_FIELDS(List<String>) – Default: empty(SELECT *). 読み込みに使用されます。オブジェクトに選択する列。
- FILTER_PREDICATE(文字列) – デフォルト: 空 読み込みに使用されます。Spark SQL 形式である必要があります。
- QUERY(文字列) - デフォルト: 空 読み込みに使用されます。完全な Spark SQL クエリ。

- PARTITION_FIELD(文字列) – 読み取りに使用されます。クエリをパーティション化するために使用するフィールド。
- LOWER_BOUND(文字列) - 読み取りに使用されます。選択したパーティションフィールドの包括的な下限値。
- UPPER_BOUND (文字列) - 読み取りに使用されます。選択したパーティションフィールドの排他的上限値。
- NUM_PARTITIONS(整数) – デフォルト: 1。読み込みに使用されます。読み取り用のパーティションの数。

制限

Microsoft Teams コネクタの制限事項は次のとおりです。

- Microsoft Teams API は、Chat および Team Member エンティティに指定された数よりも少ない数のレコードを返します。この問題は Microsoft Teams サポートに報告されており、調査中です。

新しい Microsoft Teams アカウントを作成します。

1. Microsoft Teams のホームページ https://account.microsoft.com/account/ に移動し、[サインイン] を選択します。
2. [アカウントを作成] を選択します。
3. アカウントの作成に必要な情報を入力し、新しいアカウントを作成します。
4. Microsoft Teams ウェブサイト https://www.microsoft.com/en-in/microsoft-teams/log-in に移動します。
5. 先ほど作成した Microsoft アカウントを使用してサインアップします。
6. Teams へのサインアップが成功したら、https://account.microsft.com/services に移動します。
7. [Microsoft 365 を試す] を選択します。
8. Microsoft Teams コネクタのすべての必要な機能にアクセスするには、以下の Microsoft 365 または Microsoft Teams サブスクリプションのいずれかを有効にします。
    - Microsoft Teams Essentials
    - Microsoft 365 Business
    - Microsoft 365 Business Basic
    - Microsoft 365 Business Standard

- Microsoft 365 Business Premium

マネージドクライアントアプリを作成します。

1. マネージドアプリケーションを作成するには、Microsoft Entra (以前の Azure Active Directory) に新しい OAuth アプリを登録する必要があります。
2. Microsoft Entra 管理センターにサインインします。
3. 複数のテナントにアクセスできる場合は、トップメニューの [設定] アイコンを使用して、[ディレクトリ + サブスクリプション] メニューからアプリケーションを登録するテナントに切り替えます。
4. [ID] > [アプリケーション] > [アプリ登録] に移動し、[新規登録] を選択します。
5. アプリケーションの表示名を入力します。
6. [サポートされているアカウントタイプ] セクションで、アプリケーションを使用できるユーザーを指定します。このアプリをグローバルにするには、[任意の組織ディレクトリ内のアカウント] または [任意の組織ディレクトリ内のアカウントと個人用 Microsoft アカウント] を選択します。
7. リダイレクト URI `https://{region}.console.aws.amazon.com/appflow/oauth` を入力します。例えば、`us-west-2 region` に `https://us-west-2.console.aws.amazon.com/appflow/oauth` を追加します。使用するリージョンごとに複数の URL を追加できます。
8. アプリを登録します。
9. 今後の使用のために、クライアント ID を書き留めます。
10. [Essentials] セクションで [証明書またはシークレットの追加] を選択します。
11. [新しいクライアントシークレット] を選択します。
12. 説明と有効期限を入力します。
13. 今後の使用のために、クライアントシークレットをコピーして保存します。
14. 左側のメニューリストで、[API アクセス許可] を選択します。
15. [アクセス許可の追加] を選択します。
16. [Microsoft Graph] を選択します。
17. [委任されたアクセス許可] を選択します。
18. 次のアクセス許可をすべて確認します。

    - User.Read
    - Offline_access

- User.Read.All
- User.ReadWrite.All
- TeamsTab.ReadWriteForTeam
- TeamsTab.ReadWriteForChat
- TeamsTab.ReadWrite.All
- TeamsTab.Read.All
- TeamSettings.ReadWrite.All
- TeamSettings.Read.All
- TeamMember.ReadWrite.All
- TeamMember.Read.All
- Team.ReadBasic.All
- GroupMember.ReadWrite.All
- GroupMember.Read.All
- Group.ReadWrite.All
- Group.Read.All
- Directory.ReadWrite.All
- Directory.Read.All
- Directory.AccessAsUser.All
- Chat.ReadWrite
- Chat.ReadBasic
- Chat.Read
- ChannelSettings.ReadWrite.All
- ChannelSettings.Read.All
- ChannelMessage.Read.All
- Channel.ReadBasic.All

19. [Add permissions (許可の追加)] を選択します。これで、アプリは正常にセットアップされました。クライアント ID とクライアントシークレットを使用して、新しい接続を作成できます。詳細については、https://learn.microsoft.com/en-us/graph/auth-register-app-v2 を参照してください。

Mixpanel に対する接続

Mixpanel は、企業がユーザーエンゲージメントを測定および最適化するのに役立つ強力なリアルタイム分析プラットフォームです。Mixpanel は、顧客の行動を追跡するために使用されるアプリです。これにより、ユーザーが製品にどのように関与しているかを追跡し、数回クリックするだけで結果をクエリおよび視覚化できるインタラクティブなレポートでこのデータを分析できます。Mixpanel ユーザーとして、AWS Glue を Mixpanel アカウントに接続できます。その後、Mixpanel を ETL ジョブのデータソースとして使用できます。これらのジョブを実行して、Mixpanel と AWS サービス、またはその他のサポートされているアプリケーション間でデータを転送します。

トピック

- Mixpanel 向けの AWS Glue サポート
- 接続を作成および使用するための API オペレーションを含むポリシー
- Mixpanel の設定
- Mixpanel 接続の設定
- Mixpanel エンティティからの読み取り
- Mixpanel 接続オプション
- Mixpanel アカウントの作成とクライアントアプリの設定
- 制限

Mixpanel 向けの AWS Glue サポート

AWS Glue は、次のように Mixpanel をサポートしています。

ソースとしてサポートされていますか?

はい。AWS Glue ETL ジョブを使用して、Mixpanel からデータをクエリできます。

ターゲットとしてサポートされていますか?

いいえ。

サポートされている Mixpanel API バージョン

2.0

接続を作成および使用するための API オペレーションを含むポリシー

次のサンプル ポリシーで、接続の作成と使用に必要な AWS の権限について説明します。新しいロールを作成する場合は、以下を含むポリシーを作成します。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:ListConnectionTypes",
                "glue:DescribeConnectionType",
                "glue:RefreshOAuth2Tokens",
                "glue:ListEntities",
                "glue:DescribeEntity"
            ],
            "Resource": "*"
        }
    ]
}
```

上記の方法を使用しない場合は、代わりに次のマネージド IAM ポリシーを使用します。

- AWSGlueServiceRole – さまざまな AWS Glue プロセスを実行するために必要なリソースへのアクセス権をユーザーに代わって付与します。これらのリソースには、AWS Glue、Amazon S3、IAM、CloudWatch Logs、および Amazon EC2 が含まれます。このポリシーで指定されたリソースの命名規則に従った場合、AWS Glue プロセスは必要なアクセス権限を使用できます。このポリシーは、通常、クローラ、ジョブ、開発エンドポイントを定義するときに指定されたロールにアタッチされます。
- AWSGlueConsoleFullAccess — ポリシーがアタッチされているアイデンティティが AWS マネージメントコンソールを使用するときは、AWS Glue リソースへのフルアクセスを許可します。このポリシーで指定されたリソースの命名規則に従った場合、ユーザーは完全なコンソール機能を使用できます。このポリシーは、通常 AWS Glue コンソールのユーザーにアタッチされています。

Mixpanel の設定

AWS Glue を使用して Mixpanel から転送できるようにするには、次の要件を満たす必要があります。

最小要件

- Mixpanel アカウントがある。アカウントの作成の詳細については、「Mixpanel アカウントの作成」を参照してください。
- Mixpanel アカウントで API アクセスが有効になっていること。Enterprise、Unlimited、Developer、Performance の各エディションで API アクセスがデフォルトで有効になっている。

これらの要件を満たすと、Mixpanel アカウントに AWS Glue を接続する準備が整います。一般的な接続では、Mixpanel で他に何もする必要はありません。

Mixpanel 接続の設定

Mixpanel は、`BasicAuth` のユーザー名とパスワードをサポートしています。基本認証は、クライアントが保護されたリソースにアクセスするための認証情報を直接提供する簡単な認証方法です。AWS Glue はユーザー名とパスワードを使用して Mixpanel API を認証できます。

`BasicAuth` フローに関する公開 Mixpanel ドキュメントについては、「Mixpanel サービスアカウント」を参照してください。

Mixpanel 接続を設定するには：

1. AWS Secrets Manager で、次の詳細を含むシークレットを作成します。
   - 基本的な認証の場合、シークレットには、`USERNAME` と `PASSWORD` を使用して接続されたアプリケーションのコンシューマーシークレットをキーとして含める必要があります。

   > **Note**
   >
   > AWS Glue で接続ごとにシークレットを作成する必要があります。

2. AWS Glue Studio で、以下の手順に従って [データ接続] の下に接続を作成します。
   a. [接続タイプ] を選択する際に、[Mixpanel] を選択します。
   b. 接続先の Mixpanel の `INSTANCE_URL` を指定します。
   c. AWS Glue が次のアクションを引き受けて、その権限を持つことができる IAM ロールを選択します。

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

```
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:DescribeSecret",
                "secretsmanager:GetSecretValue",
                "secretsmanager:PutSecretValue",
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterface",
                "ec2:DeleteNetworkInterface",
            ],
            "Resource": "*"
        }
    ]
}
```

d. AWS Glue でこの接続に使用する `secretName` を選択して、トークンを配置します。

e. ネットワークを使用する場合は、[ネットワークオプション] を選択します。

3. AWS Glue ジョブに関連付けられている IAM ロールに `secretName` を読み取るアクセス許可を付与します。

Mixpanel エンティティからの読み取り

前提条件

データを読み取る `Funnels`、`Retention`、`Retention Funnels` などの Mixpanel オブジェクトが必要です。さらに、オブジェクト名を知る必要があります。

サポートされているエンティティ

エンティティ	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
ファネル	はい	いいえ	いいえ	はい	いいえ
Retention	はい	いいえ	いいえ	はい	いいえ
セグメンテーション	はい	いいえ	いいえ	はい	いいえ

エンティティ	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
セグメンテーション合計	はい	いいえ	いいえ	はい	いいえ
セグメンテーション平均	はい	いいえ	いいえ	はい	いいえ
コホート	はい	いいえ	いいえ	はい	いいえ
Engage	いいえ	はい	いいえ	はい	いいえ
イベント	はい	いいえ	いいえ	はい	いいえ
イベントの上位	はい	いいえ	いいえ	はい	いいえ
イベント名	はい	いいえ	いいえ	はい	いいえ
イベントのプロパティ	はい	いいえ	いいえ	はい	いいえ
イベントのプロパティの上位	はい	いいえ	いいえ	はい	いいえ
イベントのプロパティ値	はい	いいえ	いいえ	はい	いいえ
注釈	はい	いいえ	いいえ	はい	いいえ
イベントアクティビティのプロファイル	はい	いいえ	いいえ	はい	いいえ

例

```
mixpanel_read = glueContext.create_dynamic_frame.from_options(
    connection_type="mixpanel",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "/cohorts/list?project_id=2603353",
        "API_VERSION": "2.0",
        "INSTANCE_URL": "https://www.mixpanel.com/api/app/me"
    }
```

Mixpanel エンティティとフィールドの詳細

エンティティ	フィールド	データタイプ	サポートされている演算子
ファネルグラフ	funnel_id	整数	'='
	workspace_id	整数	'='
	from_date	日付	'='
	to_date	日付	'=’
	length	整数	'='
	length_unit	String	'='
	interval	整数	'='
	単位	String	'='
	制限	整数	'='
	データ	Struct	
	meta	Struct	
Retention	workspace_id	整数	'='
	単位	String	'='
	addiction_unit	String	'='

エンティティ	フィールド	データタイプ	サポートされている演算子
	from_date	日付	'='
	to_date	日付	'='
	イベント	String	'='
	制限	整数	'='
	データ	Struct	
セグメンテーション	workspace_id	整数	'='
	イベント	String	'='
	from_date	日付	'='
	to_date	日付	'='
	単位	String	'='
	interval	整数	'='
	制限	整数	'='
	type	String	'='
	列	リスト	
	値	Struct	
	データ	Struct	
セグメンテーション数値	workspace_id	整数	'='
	イベント	String	'='
	オン	String	'='
	from_date	日付	'='

エンティティ	フィールド	データタイプ	サポートされている演算子
	to_date	日付	'='
	単位	String	'='
	type	String	'='
	列	リスト	
	値	Struct	
セグメンテーション合計	workspace_id	整数	'='
	イベント	String	'='
	オン	String	'='
	from_date	日付	'='
	to_date	日付	'='
	単位	String	'='
	metadata	Struct	
	結果	Struct	
セグメンテーション平均	workspace_id	整数	'='
	イベント	String	'='
	オン	String	'='
	from_date	日付	'='
	to_date	日付	'='
	単位	String	'='
	metadata	Struct	

エンティティ	フィールド	データタイプ	サポートされている演算子
	結果	Struct	
コホート	count	整数	
	is_visible	整数	
	description	String	
	作成済み	DateTime	
	project_id	整数	
	id	BigInteger	
	名前	String	
	data_group_id	String	
Engage	distinct_id	String	
		プロパティ	Struct
イベント	ワークスペース	整数	'='
	イベント	String	'='
	type	String	'='
	単位	String	'='
	interval	整数	'='
	from_date	日付	'='
	to_date	日付	'='
	列	リスト	
	値	Struct	

エンティティ	フィールド	データタイプ	サポートされている演算子
イベントの上位	type	String	'='
	workspace_id	整数	'='
	制限	整数	'='
	amount	整数	
	イベント	String	
	percent_change	浮動小数点数	
イベント名	データ	リスト	
	workspace_id	整数	'='
	type	String	'='
	制限	整数	'='
イベントのプロパティ	workspace_id	整数	'='
	イベント	String	'='
	名前	String	'='
	type	String	'='
	単位	String	'='
	interval	整数	'='
	from_date	日付	'='
	to_date	日付	'='
	制限	整数	'='
	データ	Struct	

エンティティ	フィールド	データタイプ	サポートされている演算子
	列	リスト	
	値	Struct	
イベントプロパティの上位	workspace_id	整数	'='
	イベント	String	'='
	制限	整数	'='
	データ	Struct	
イベントプロパティ値	workspace_id	整数	'='
	イベント	String	'='
	制限	整数	'='
	名前	String	'='
	データ	リスト	
注釈	workspace_id	整数	
	date	DateTime	
	project_id	整数	
	id	BigInteger	
	description	String	
	from_date	日付	BETWEEN
イベントアクティビティのプロファイル	workspace_id	整数	'='
	distinct_ids	String	'='
	from_date	日付	'='

エンティティ	フィールド	データタイプ	サポートされている演算子
	to_date	日付	'='
	イベント	String	
	プロパティ	Struct	

Mixpanel 接続オプション

Mixpanel の接続オプションは次のとおりです。

- ENTITY_NAME (文字列) – (必須) 読み取り / 書き込みに使用されます。Mixpanel のオブジェクトの名前。
- API_VERSION (文字列) – (必須) 読み取り / 書き込みに使用されます。使用する Mixpanel Rest API バージョン。例: v2.0。
- SELECTED_FIELDS(List<String>) – Default: empty (SELECT *). 読み込みに使用されます。オブジェクトに選択する列。
- FILTER_PREDICATE(文字列) – デフォルト: 空 読み込みに使用されます。Spark SQL 形式である必要があります。
- QUERY(文字列) – デフォルト: 空 読み込みに使用されます。完全な Spark SQL クエリ。

Mixpanel アカウントの作成とクライアントアプリの設定

Mixpanel アカウントの作成

1. Mixpanel ホームページ/に移動します。
2. [Mixpanel] のホームページで、ページの右上隅にある [サインアップ] を選択します。
3. [Let's get started] ページで、次のアクションを実行します。
   - 指定されたフィールドに E メールアドレスを入力します。
   - 必要なチェックボックスを選択して、条件に同意します。
   - [開始方法] を選択して続行します。

     正常に完了すると、検証 E メールが送信されます。

4. E メールの受信トレイで検証メッセージを確認し、E メールを開き、手順に従って E メールアドレスを確認します。
5. 検証ページで、[E メールの検証] を選択して E メールの検証を完了します。
6. [組織の名前付け] ページで、組織名を入力し、[次へ] を選択します。
7. [最初のプロジェクト] ページで、プロジェクトの詳細を入力し、[作成] を選択します。
8. 次のページで、[始めましょう] を選択してアカウントの作成を完了します。

Mixpanel アカウントへのログイン

1. Mixpanel ログインページ/に移動します。
2. E メールアドレスを入力し、[続行] を選択します。
3. E メールの受信トレイで検証メッセージを確認し、E メールを開き、手順に従って E メールアドレスを確認します。
4. 次のページで、[ログインボタン] を選択してアカウントにログインします。

Mixpanel プランの購入

1. Mixpanel ページで、ページの右上隅にある [設定] アイコンを選択します。
2. オプションのリストから、[プランの詳細と請求] を選択します。
3. [プランの詳細と請求] ページで、[アップグレードまたは変更] を選択します。
4. 次のページで、購入するプランを選択します。

   これで、アカウントの作成とプランの購入プロセスが完了します。

ユーザー名とクライアントシークレットの作成 (アプリを登録するには)

1. Mixpanel ページで、ページの右上隅にある [設定] アイコンを選択します。
2. オプションのリストから、[プロジェクト設定] を選択します。
3. [プロジェクト設定] ページで、[サービスアカウント] を選択し、[サービスアカウントの追加] を選択します。
4. [サービスアカウント] のドロップダウンリストから、[サービスアカウントを選択するか、作成する名前を入力し]、[プロジェクトロール] を追加し、[有効期限] を指定して、[追加] を選択します。

> **Important**
>
> 前のステップを完了すると、次のページにサービスアカウントのシークレットキーが表示されます。サービスアカウントのシークレットキーを必ず保存してください。この時点以降は、再度アクセスすることはできません。

制限

Mixpanel コネクタの制限は次のとおりです:

- `Segmentation Numeric` エンティティの場合、必須フィルターの数値データが見つからなかった場合、Mixpanel API は `400 (Bad Request)` エラーをスローします。フロー障害を防ぐための OK 対応として扱います。
- 次の理由により、クエリ可能なフィールド `limit` はサポートされているエンティティから削除されました。
  - SDK の制限機能として解釈されたため、エラーが発生していた
  - フィルターが実用的ではなかった
  - 同等の機能が制限機能の実装の対象となった
- SaaS プラットフォームからのパーティショニングに必要な演算子 (>=、<=、<、>、between) がないため、フィールドベースのパーティション分割をサポートできません。between 演算子をサポートしていますが、この演算子をサポートするフィールドは取得できません。したがって、フィールドベースのパーティショニングの基準は満たされません。
- ページ分割をサポートするエンティティの「オフセット」値を取得するプロビジョニングがないため、Mixpanel のレコードベースのパーティション分割をサポートすることはできません。
- `Cohorts` エンティティは `CreatedDate/Time` フィールドのみをサポートしており、結果 `DML_Status` を識別できないため、`UpdatedDate/Time` を識別するフィールドはありません。また、削除されたレコードを識別するエンドポイントはありません。したがって、CDC はサポートされません。
- 以下に示すエンティティの AWS Glue ジョブを実行するには、必須のフィルターが必要です。エンティティ名とその必要なフィルターについては、以下の表を参照してください。

エンティティ名と必要なフィルター

エンティティ名	必須フィルター
注釈	from_date、to_date
コホート	[なし]
Engage	[なし]
イベント	event、type、 unit、from_date、to_date、on
イベント名	type
イベントのプロパティ	event、name、type、unit、from_date、to_date
イベントのプロパティの上位	イベント
イベントのプロパティ値	event、name
イベントの上位	type
ファネル	funnel_id、from_date、to_date
イベントアクティビティのプロファイル	distinct_ids、from_date、to_date
Retention	from_date、to_date、unit、addiction_unit
セグメンテーション	event、from_date、to_date
セグメンテーション平均	event、from_date、to_date、on
セグメンテーション数値	event、from_date、to_date、on
セグメンテーション合計	event、from_date、to_date、on

Monday への接続

Monday.com は、プロジェクト管理とチームコラボレーションを合理化する汎用性の高い作業オペレーティングシステムです。カスタマイズ可能なワークフロー、ビジュアルダッシュボード、自動化ツールが用意されており、生産性が向上します。ユーザーは、1 つの統合プラットフォームでタスクを追跡し、リソースを管理し、効果的に通信できます。

トピック

- AWS Glue の Monday のサポート
- 接続を作成および使用するための API オペレーションを含むポリシー
- Monday の設定
- Monday 接続の設定
- Monday エンティティからの読み取り
- Monday 接続オプションのリファレンス
- 制限
- 新しい Monday アカウントを作成します。

AWS Glue の Monday のサポート

AWS Glue は、次のように Monday をサポートします。

ソースとしてサポートされていますか?

はい。AWS Glue ETL ジョブを使用して、Monday からデータをクエリできます。

ターゲットとしてサポートされていますか?

いいえ。

サポートされている Monday の API バージョン

v2。

接続を作成および使用するための API オペレーションを含むポリシー

次のサンプル ポリシーで、接続の作成と使用に必要な AWS の権限について説明します。新しいロールを作成する場合は、以下を含むポリシーを作成します。

```
{
```

```
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:ListConnectionTypes",
                "glue:DescribeConnectionType",
                "glue:RefreshOAuth2Tokens"
                "glue:ListEntities",
                "glue:DescribeEntity"
            ],
            "Resource": "*"
        }
    ]
}
```

また、次のマネージド IAM ポリシーを使用して、アクセスを許可できます。

- AWSGlueServiceRole – さまざまな AWS Glue プロセスを実行するために必要なリソースへのアクセス権をユーザーに代わって付与します。これらのリソースには、AWS Glue、Amazon S3、IAM、CloudWatch Logs、および Amazon EC2 が含まれます。このポリシーで指定されたリソースの命名規則に従った場合、AWS Glue プロセスは必要なアクセス権限を使用できます。このポリシーは、通常、クローラ、ジョブ、開発エンドポイントを定義するときに指定されたロールにアタッチされます。
- AWSGlueConsoleFullAccess — ポリシーがアタッチされているアイデンティティが AWS マネージメントコンソールを使用するときは、AWS Glue リソースへのフルアクセスを許可します。このポリシーで指定されたリソースの命名規則に従った場合、ユーザーは完全なコンソール機能を使用できます。このポリシーは、通常 AWS Glue コンソールのユーザーにアタッチされています。

Monday の設定

AWS Glue を使用して Monday からデータを転送できるようにするには、次の要件を満たす必要があります。

最小要件

- メールアドレスとパスワードを設定した Monday のデベロッパーアカウントがある。詳細については、「新しい Monday アカウントを作成します。」を参照してください。
- Monday デベロッパーアカウントで API アクセスが有効になっている。トライアル期間中は、追加料金なしで Monday API をすべて使用できます。トライアル期間終了後にデータを作成してアクセ

スするには、サブスクリプションを購入する必要があります。詳細については、Monday のライセンスページを参照してください。

これらの要件を満たすと、Monday アカウントに AWS Glue を接続する準備が整います。一般的な接続では、Monday で他に何もする必要はありません。

Monday 接続の設定

Monday は、認可メカニズムに次の 2 つのタイプをサポートしています。

1. OAuth 認可: Monday は、OAuth2 の AUTHORIZATION_CODE グラントタイプをサポートしています。
   - このグラントタイプは、ユーザーを認証するためにサードパーティーの認証サーバーにユーザーをリダイレクトすることから、「three-legged」の OAuth と見なされます。AWS Glue コンソール経由で接続を作成するときに使用されます。接続を作成するユーザーは、デフォルトで AWS Glue 所有の接続アプリケーションを使用できます。この場合、Monday インスタンス URL 以外の OAuth 関連情報を指定する必要はありません。AWS Glue コンソールでは、ユーザーが Monday にリダイレクトされます。ここではユーザーがログインし、Monday インスタンスにアクセスするためのリクエストされた権限を AWS Glue に許可する必要があります。
   - ユーザーは、Monday で独自の接続アプリケーションを作成し、AWS Glue コンソールを介して接続を作成するときに独自のクライアント ID とクライアントシークレットを指定することを選択することもできます。このシナリオでも Monday にリダイレクトされてログインし、リソースへアクセスするために AWS Glue を認可します。
   - このグラントタイプは、更新トークンとアクセストークンになります。アクセストークンは 1 時間有効で、更新トークンを使用することでユーザーとやり取りすることなく自動的に更新できます。
   - 詳細については、AUTHORIZATION_CODE OAuth フローでの接続アプリケーションの作成に関するドキュメントを参照してください。
2. カスタム認可:
   - カスタム認可に必要な API キーの生成に関する Monday の公開ドキュメントについては、https://developer.monday.com/api-reference/docs/authentication#api-token-permissions を参照してください。

Monday 接続を設定するには

1. AWS Secrets Manager で、次の詳細を含むシークレットを作成します。AWS Glue で接続ごとにシークレットを作成する必要があります。

   a. OAuth 認可の場合:

   - カスタマーマネージド接続アプリケーションの場合 – シークレットには、USER_MANAGED_CLIENT_APPLICATION_CLIENT_SECRET を使用して接続されたアプリケーションのコンシューマーシークレットをキーとして含める必要があります。

   b. カスタム認可の場合:

   - カスタマーマネージド接続アプリケーションの場合 – シークレットには、`personalAccessToken` を使用して接続されたアプリケーションのコンシューマーシークレットをキーとして含める必要があります。

2. AWS Glue Studio で、以下の手順に従って [データ接続] の下に接続を作成します。

   a. [データ接続] の [接続を作成] を選択します。

   b. [データソース] を選択する際に、[Monday] を選択します。

   c. Monday の [instanceURL] を指定します。

   d. AWS Glue が次のアクションを担い、その権限を持つことができる IAM ロールを選択します。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:DescribeSecret",
                "secretsmanager:GetSecretValue",
                "secretsmanager:PutSecretValue",
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterface",
                "ec2:DeleteNetworkInterface",
            ],
            "Resource": "*"
        }
    ]
}
```

   e. Monday に接続するための認可タイプを選択する

   - OAuth 認可の場合: 接続する Monday の [トークン URL] と [ユーザーマネージドクライアントアプリケーション ClientId] を指定します。

- カスタム認可の場合: 認証タイプ [CUSTOM] を選択して Monday に接続します。

f. AWS Glue でこの接続に使用する `secretName` を選択して、トークンを配置します。

g. ネットワークを使用する場合は、ネットワークオプションを選択します。

3. AWS Glue ジョブに関連付けられている IAM ロールに `secretName` を読み取るアクセス許可を付与します。[Next] を選択します。
4. AWS Glue ジョブ設定で、[追加のネットワーク接続] として `connectionName` を指定します。

Monday エンティティからの読み取り

前提条件

- 読み取り元の Monday オブジェクト。使用可能なエンティティを確認するには、以下のサポートされているエンティティの表を参照してください。

ソースに対応するエンティティ

エンティティリスト:

- アカウント: https://developer.monday.com/api-reference/docs/account#queries
- ボード: https://developer.monday.com/api-reference/docs/boards#queries
- カラム: https://developer.monday.com/api-reference/docs/columns#queries
- ドキュメント: https://developer.monday.com/api-reference/docs/docs#queries
- ドキュメントブロック: https://developer.monday.com/api-reference/docs/blocks#queries
- ファイル: https://developer.monday.com/api-reference/docs/files#queries
- フォルダ: https://developer.monday.com/api-reference/docs/folders#queries
- グループ: https://developer.monday.com/api-reference/docs/groups#queries
- アイテム: https://developer.monday.com/api-reference/docs/items#queries
- サブアイテム: https://developer.monday.com/api-reference/docs/subitems#queries
- タグ: https://developer.monday.com/api-reference/docs/tags-queries#queries
- チーム: https://developer.monday.com/api-reference/docs/teams#queries
- アップデート: https://developer.monday.com/api-reference/docs/updates#queries
- ユーザー: https://developer.monday.com/api-reference/docs/users#queries
- ワークスペース: https://developer.monday.com/api-reference/docs/workspaces#queries

エンティティ	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
アカウント	いいえ	いいえ	なし	あり	不可
ボード	あり	はい	なし	あり	不可
列	いいえ	いいえ	なし	あり	不可
ドキュメント	あり	はい	なし	あり	不可
ドキュメントブロック	いいえ	あり	なし	あり	不可
ファイル	あり	いいえ	なし	あり	不可
グループ	いいえ	いいえ	なし	あり	不可
項目	あり	はい	なし	あり	不可
サブアイテム	いいえ	いいえ	なし	あり	不可
[タグ]	あり	いいえ	なし	あり	あり
チーム	あり	いいえ	なし	あり	不可
更新	いいえ	あり	なし	あり	不可
[ユーザー]	あり	はい	なし	あり	不可
ワークスペース	あり	はい	なし	あり	不可
フォルダ	あり	はい	なし	あり	不可

例

```
monday_read = glueContext.create_dynamic_frame.from_options(
    connection_type="monday",
```

```
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "account",
        "API_VERSION": "v2"
    }
```

Monday 接続オプションのリファレンス

Monday の接続オプションは次のとおりです。

- ENTITY_NAME(文字列) – (必須) 読み取り / 書き込みに使用されます。Monday のオブジェクトの名前。
- API_VERSION(文字列) – (必須) 読み取り / 書き込みに使用されます。使用する Monday Rest API バージョン。例: v2。
- SELECTED_FIELDS(List<String>) – Default: empty(SELECT *). 読み込みに使用されます。オブジェクトに選択する列。
- FILTER_PREDICATE(文字列) – デフォルト: 空 読み込みに使用されます。Spark SQL 形式である必要があります。
- QUERY(文字列) - デフォルト: 空 読み込みに使用されます。完全な Spark SQL クエリ。

制限

Monday コネクタの制限は次のとおりです。

- 動的メタデータレスポンスには、以下で説明するように、ドキュメントと特定の競合があります。
  - グループ、列エンティティはフィルターオペレーションをサポートしますが、動的メタデータエンドポイントには存在しないため、フィルター可能でないエンティティとして保持されます。
  - 動的エンドポイントは約 15000 超の行で構成され、1 回のレスポンスですべてのエンティティのメタデータを返します。このため、フィールドのロードには平均 10 秒かかるので、ジョブの実行中にさらに時間がかかります。
  - Monday のレート制限については、以下の表を参照してください。動的エンティティのレスポンスデータのサイズが大きいと、顕著な遅延の原因となり、フィールドのロードには平均 10 秒かかります。

複雑さの制限	5,000,000 (5M) の複雑さポイント
1 日のコールの制限	Pro プランの場合は 10,000
分の制限	1 分あたり 500 クエリ
同時実行数の制限	Pro プランの場合は最大同時リクエスト数 100

新しい Monday アカウントを作成します。

1. Monday のホームページ https://monday.com/ に移動し、[ログイン] を選択します。
2. ログインページにリダイレクトされます。ページの最下部にある [サインアップ] を選択します。
3. E メールアドレスを入力し、[続行] を選択します。または、Google でサインインすることもできます。
4. 必要な情報を入力して、[続行] を選択します。
5. アンケートの質問に回答し、手順に従ってアカウント作成プロセスを完了します。

OAuth アプリケーションを登録します。

1. monday.com アカウントにログインします。画面の左下隅にあるアバター (ピクチャアイコン) をクリックします。
2. [デベロッパー] を選択します。
3. [アプリの作成] を選択します。
4. 名前と説明の必須フィールドに入力します。
5. 右側の [OAuth] セクションに移動し、スコープを追加し、[機能の保存] を選択します。
6. スコープの横にある [リダイレクト URL] タブに移動し、リダイレクト URL を追加して [機能の保存] を選択します。
7. [リダイレクト URL] タブで、アプリの URL を指定します。これは https://{region-code}.console.aws.amazon.com/appflow/oauth である必要があります。例えば、`us-east-1` を使用している場合は、`https://us-east-1.console.aws.amazon.com/appflow/oauth` を追加できます。

8. これで、アプリケーションを使用する準備ができました。認証情報は [基本情報] セクションにあります。クライアント ID とクライアントシークレット文字列を書き留めます。これらの文字列は、AppFlow コネクタを使用してこのアプリと接続するために使用されます。

個人用アクセストークンを生成する:

現在、monday.com は V2 API トークンのみを提供しており、それらはすべてパーソナルトークンです。API トークンにアクセスするには、ユーザーレベルに応じて 2 つの方法のいずれかを使用できます。管理者ユーザーは、両方の方法を使用して API トークンを取得できます。メンバーユーザーは、[デベロッパー] タブから API トークンにアクセスできます。

管理者 - monday.com アカウントの管理者ユーザーは、次のステップで [管理者] タブから API トークンにアクセスできます。

1. monday.com アカウントにログインします。画面の左下隅にあるアバター (ピクチャアイコン) をクリックします。
2. 表示されたメニューから [管理] を選択します (管理者アクセス許可が必要です)。
3. [API] セクションに移動し、[API V2 トークン] を生成します。トークンをコピーして使用できます。

デベロッパー - monday.com アカウントのメンバーユーザーは、次のステップで [デベロッパー] タブから API トークンにアクセスできます。

1. monday.com アカウントにログインします。画面の左下隅にあるアバター (ピクチャアイコン) をクリックします。
2. 表示されたメニューから [デベロッパー] を選択します。
3. 上部のメニューで、[デベロッパー] ドロップダウンメニューを選択します。[マイアクセストークン] というタイトルのドロップダウンメニューの最初のオプションを選択します。

AWS Glue Studio での MongoDB に対する接続

AWS Glue は、MongoDB のための組み込みサポートを提供します。AWS Glue Studio は、MongoDB に接続してデータ統合ジョブをオーサリングし、AWS Glue Studio サーバーレス Spark ランタイム上でそれらのジョブを実行するためのビジュアルインターフェイスを提供します。

トピック

- MongoDB 接続の作成
- MongoDB ソースノードの作成
- MongoDB ターゲットノードの作成
- 詳細オプション

MongoDB 接続の作成

前提条件:

- MongoDB インスタンスが Amazon VPC 内にある場合は、トラフィックがパブリックインターネットを経由することなく、AWS Glue ジョブが MongoDB インスタンスと通信できるように Amazon VPC を設定します。

  Amazon VPC で、AWS Glue がジョブの実行中に使用する [VPC]、[サブネット]、および [セキュリティグループ] を特定または作成します。さらに、MongoDB インスタンスとこの場所の間のネットワークトラフィックを許可するように Amazon VPC が設定されているようにする必要があります。ネットワークレイアウトに基づいて、セキュリティグループルール、ネットワーク ACL、NAT ゲートウェイ、およびピアリング接続の変更が必要になる場合があります。

MongoDB に対する接続を設定するには:

1. 必要に応じて、AWS Secrets Manager で、MongoDB 認証情報を使用してシークレットを作成します。Secrets Manager でシークレットを作成するには、AWS Secrets Manager ドキュメントの「AWS Secrets Manager シークレットを作成する」にあるチュートリアルに従ってください。シークレットを作成したら、次のステップのためにシークレット名 *secretName* を保存しておきます。

   - [key/value ペア] を選択する際に、*mongodbUser* という値を持つキー `username` のペアを作成します。

     [key/value ペア] を選択する際に、*mongodbPass* という値を持つキー `password` のペアを作成します。

2. AWS Glue コンソールで、「the section called “AWS Glue 接続の追加”」にあるステップに従って接続を作成します。接続を作成したら、将来的に AWS Glue で使用するために、接続名 *connectionName* を維持します。

   - [接続タイプ] を選択する際には、[MongoDB] または [MongoDB Atlas] を選択します。

- [MongoDB URL] または [MongoDB Atlas URL] を選択する場合は、MongoDB インスタンスのホスト名を入力します。

  MongoDB URL は、`mongodb://`*`mongoHost`*`:`*`mongoPort`*`/`*`mongoDBname`* の形式で指定されます。

  MongoDB Atlas URL は、`mongodb+srv://`*`mongoHost`*`:`*`mongoPort`*`/`*`mongoDBname`* の形式で指定されます。

  接続にデフォルトのデータベースを指定する場合、*`mongoDBname`* はオプションです。

- Secrets Manager シークレットを作成することを選択した場合は、AWS Secrets Manager の [認証情報タイプ] を選択します。

  その後、[AWS シークレット] で *`secretName`* を入力します。

- [ユーザー名とパスワード] を入力することを選択した場合は、*`mongodbUser`* および *`mongodbPass`* を入力します。

3. 次の状況では、追加の設定が必要になる場合があります。

   - Amazon VPC の AWS でホストされている MongoDB インスタンスの場合
     - MongoDB セキュリティ認証情報を定義する AWS Glue 接続に、Amazon VPC 接続に関する情報を提供する必要があります。接続を作成または更新する際に、[ネットワークオプション] で [VPC]、[サブネット]、および [セキュリティグループ] を設定します。

AWS Glue MongoDB 接続を作成した後、AWS Glue ジョブを実行する前に次のステップを実行する必要があります。

- ビジュアルエディタで AWS Glue ジョブを使用する場合、ジョブが MongoDB に接続できるように Amazon VPC の接続に関する情報を入力する必要があります。Amazon VPC 内の適切な場所を特定し、それを AWS Glue MongoDB 接続に提供します。
- Secrets Manager シークレットを作成することを選択した場合は、AWS Glue ジョブに関連付けられた IAM ロールに *`secretName`* を読み取るための許可を付与します。

MongoDB ソースノードの作成

必要な前提条件

- 前のセクション「the section called “MongoDB 接続の作成”」で説明した AWS Glue MongoDB 接続。
- Secrets Manager シークレットを作成することを選択した場合は、接続によって使用されるシークレットを読み取るためのジョブに対する適切な許可が必要です。
- 読み取り元とする MongoDB コレクション。コレクションの識別情報が必要になります。

  MongoDB コレクションは、データベース名とコレクション名である *mongodbName* および *mongodbCollection* によって識別されます。

MongoDB データソースの追加

[データソース — MongoDB] ノードを追加するには:

1. MongoDB データソース用の接続を選択します。すでに作成したので、ドロップダウンに表示されているはずです。接続を作成する必要がある場合は、[MongoDB 接続を作成] を選択します。詳細については、前の「the section called “MongoDB 接続の作成”」セクションを参照してください。

   接続を選択したあとは、[プロパティを表示] をクリックすると、接続のプロパティを表示できます。
2. [データベース] を選択します。*mongodbName* を入力します。
3. [コレクション] を選択します。*mongodbCollection* を入力します。
4. [パーティショナー]、[パーティションサイズ (MB)]、および [パーティションキー] を選択します。パーティションパラメータについての詳細は、「the section called “"connectionType": "mongodb" ソースとする”」を参照してください。
5. [MongoDB のカスタムプロパティ] で、必要に応じてパラメータと値を入力します。

MongoDB ターゲットノードの作成

必要な前提条件

- 前のセクション the section called “MongoDB 接続の作成” で説明したように、AWS Secrets Manager シークレットを使用して設定された AWS Glue MongoDB 接続。

- 接続で使用されるシークレットを読み取るためのジョブに対する適切なアクセス許可。
- 書き込み先とする MongoDB テーブルである *tableName*。

MongoDB データターゲットの追加

[データターゲット — MongoDB] ノードを追加するには:

1. MongoDB データソース用の接続を選択します。すでに作成したので、ドロップダウンに表示されているはずです。接続を作成する必要がある場合は、[MongoDB 接続を作成] を選択します。詳細については、前の「the section called “MongoDB 接続の作成”」セクションを参照してください。

   接続を選択したあとは、[プロパティを表示] をクリックすると、接続のプロパティを表示できます。
2. [データベース] を選択します。*mongodbName* を入力します。
3. [コレクション] を選択します。*mongodbCollection* を入力します。
4. [パーティショナー]、[パーティションサイズ (MB)]、および [パーティションキー] を選択します。パーティションパラメータについての詳細は、「the section called “"connectionType": "mongodb" ソースとする”」を参照してください。
5. 必要に応じて、[書き込みを再試行] を選択します。
6. [MongoDB のカスタムプロパティ] で、必要に応じてパラメータと値を入力します。

詳細オプション

MongoDB ノードを作成する際に、高度なオプションを指定できます。これらのオプションは Spark AWS Glue スクリプトのプログラミング時に使用できるオプションと同じです。

「the section called “MongoDB 接続”」を参照してください。

Oracle への接続 NetSuite

Oracle NetSuite は、 all-in-oneコアプロセスを自動化し、運用パフォーマンスと財務パフォーマンスをリアルタイムで可視化することで、組織の運用をより効果的にするのに役立つクラウドビジネス管理ソリューションです。Oracle は、会計、注文処理、在庫管理、生産、サプライチェーン、倉庫オペレーションを管理するための単一の統合されたアプリケーションスイートを使用して、企業にデータに対する明確な可視性とビジネスに対するより厳格なコントロール NetSuite を提供します。

トピック

- AWS Glue Oracle のサポート NetSuite
- 接続を作成および使用するためのAPIオペレーションを含むポリシー
- Oracle の設定 NetSuite
- Oracle NetSuite 接続の設定
- Oracle NetSuite エンティティからの読み取り
- Oracle NetSuite 接続オプション
- Oracle NetSuite コネクタの制限と注意事項

AWS Glue Oracle のサポート NetSuite

AWS Glue は Oracle NetSuite を次のようにサポートします。

ソースとしてサポートされていますか?

はい。ジョブを使用して AWS Glue ETL、Oracle からデータをクエリできます NetSuite。

ターゲットとしてサポートされているかどうか

いいえ。

サポートされている Oracle NetSuite API バージョン

次の Oracle NetSuite API バージョンがサポートされています。

- v1

バージョン固有のエンティティサポートについては、「Source でサポートされているエンティティ」を参照してください。

接続を作成および使用するためのAPIオペレーションを含むポリシー

次のサンプルポリシーでは、接続の作成と使用に必要な AWS IAMアクセス許可について説明します。新しいロールを作成する場合は、以下を含むポリシーを作成します。

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

```
            "Action": [
                "glue:ListConnectionTypes",
                "glue:DescribeConnectionType",
                "glue:RefreshOAuth2Tokens"
                "glue:ListEntities",
                "glue:DescribeEntity"
            ],
            "Resource": "*"
        }
    ]
}
```

上記の方法を使用しない場合は、次の管理IAMポリシーを使用します。

- AWSGlueServiceRole – さまざまな AWS Glue プロセスがユーザーに代わって実行するために必要なリソースへのアクセスを許可します。これらのリソースには AWS Glue、、Amazon S3、IAM、 CloudWatch Logs、Amazon が含まれますEC2。このポリシーで指定されたリソースの命名規則に従うと、 AWS Glue プロセスには必要なアクセス許可があります。このポリシーは、通常、クローラ、ジョブ、開発エンドポイントを定義するときに指定されたロールにアタッチされます。
- AWSGlueConsoleFullAccess — ポリシーがアタッチされている ID が AWS マネジメントコンソールを使用する場合、 AWS Glue リソースへのフルアクセスを許可します。このポリシーで指定されたリソースの命名規則に従った場合、ユーザーは完全なコンソール機能を使用できます。このポリシーは通常、 AWS Glue コンソールのユーザーにアタッチされます。

Oracle の設定 NetSuite

AWS Glue を使用して Oracle からデータを転送する前に NetSuite、次の要件を満たす必要があります。

最小要件

以下に、最小要件を示します。

- Oracle NetSuite アカウントがあります。詳細については、「Oracle NetSuite アカウントの作成」を参照してください。
- Oracle NetSuite アカウントはAPIアクセス可能です。
- Oracle デ NetSuite ベロッパーアカウントに OAuth 2.0 API統合を作成しました。この統合により、 がアカウントに対して認証された呼び出しを行うときに、 AWS Glue がデータに安全にアク

セスするためのクライアント認証情報が提供されます。詳細については、「Oracle NetSuite クライアントアプリケーションと 2.0 OAuth 認証情報の作成」を参照してください。

これらの要件を満たすと、Oracle NetSuite アカウント AWS Glue に接続できます。

Oracle NetSuite アカウントの作成

Oracle NetSuite に移動し、Free Product Tour を選択します。必要な詳細を入力して、ベンダーに連絡できる無料の製品ツアーを取得します。アカウントを調達するプロセスは次のとおりです。

- NetSuite アカウントの調達は、法的にレビューする必要があるフォーム/クォートを提供するベンダーを介して行われます。
- Oracle NetSuite コネクタ用に調達するアカウントは、Standard Cloud Service です。
- このアカウントはベンダーによって作成され、一時的な認証情報はベンダーによって共有されます。 NetSuite <billing@notification.netsuite.com> <system@sent-via.netsuite.com> から、ユーザー名やパスワードを設定するためのリンクなどの詳細が記載されたウェルカムメールが届きます。
- パスワードを設定するリンクを使用して、ベンダーから提供されたユーザー名のパスワードを設定します。

Oracle NetSuite クライアントアプリケーションと 2.0 OAuth 認証情報の作成

クライアント ID とクライアントシークレットを取得するには、Oracle NetSuite クライアントアプリケーションを作成します。

1. NetSuite カスタマーログイン を使用して NetSuite アカウントにログインします。
2. Setup > Company > Enable features を選択します。
3. SuiteCloud セクションに移動し、 SuiteTalk （ ウェブサービス ） のRESTWEBSERVICESチェックボックスを選択します。
4. 認証の管理 の OAUTH2.0 チェックボックスを選択します。 [Save] (保存) をクリックします。
5. Setup > Integration > Manage Integrations に移動し、New を選択して OAuth2.0 アプリケーションを作成します。
6. 選択した名前を入力し、 を有効STATEのままにします。
7. オンにした場合、トークンベースの認証 に表示される TBA: AUTHORIZATIONFLOWおよび TOKEN-BASED AUTHENTICATION チェックボックスのチェックを外します。

8. OAuth 2.0 の下にある AUTHORIZATIONCODEGRANTおよび PUBLICCLIENTチェックボックスを選択します。
9. 認証で、クライアント ID とクライアントシークレットを書き留めます。
10. を入力しますREDIRECTURI。例えば、 https://us-east-1.console.aws.amazon.com/gluestudio/oauth
11. のRESTWEBSERVICESチェックボックスをオンにしますSCOPE。
12. ユーザー認証情報 のUSERCREDENTIALSチェックボックスをオンにします。 [Save] を選択します。
13. クライアント認証情報 の CONSUMERKEY/CLIENT ID and CONSUMER SECRET/CLIENTSECRETに注意してください。これらの値は 1 回のみ表示されます。
14. 必要に応じてADMINISTRATOR、ユーザー/ロール > ロールの管理 > 新規 に移動してロールを作成します。
15. カスタムロールの作成中に、次のエンティティ/機能のアクセス許可タブにフルアクセスを追加します。
    - 「Deposit」、「Items」、「Items Fulfillment」、「Make Journal Entry」、「購入注文」、「子会社」、「ベンダー」、「Bills」、「ベンダー返品承認」、「時間の追跡」、「顧客支払い」、「カスタムレコードエントリ」、」、「カスタムレコードタイプ」、REST「ウェブサービス」、OAuth「2.0 認定アプリケーション管理」、「カスタムエンティティフィールド」、」、OAuth「2.0 アクセストークンを使用してログイン」」。

詳細については、 NetSuite Applications Suite ドキュメントのOAuth「2.0」を参照してください。

Oracle NetSuite 接続の設定

Oracle NetSuite は、OAuth2 の AUTHORIZATION_CODE のグラントタイプをサポートしています。グラントタイプは、AWS Glue が Oracle NetSuite と通信してデータへのアクセスをリクエストする方法を決定します。

- このグラントタイプは、ユーザーを認証するためにサードパーティーの認証サーバーにユーザーをリダイレクトすることから、「three-legged」の OAuth と見なされます。AWS Glue コンソール経由で接続を作成するときに使用されます。接続を作成するユーザーは、デフォルトで AWS Glue 所有の接続アプリ (AWS Glue マネージドクライアントアプリケーション) に依存する場合があります。この場合、Oracle NetSuite インスタンス URL 以外の OAuth 関連情報を提供する必要はありません。AWS Glue コンソールでは、ユーザーが Oracle NetSuite にリダイレクトされます。ここではユーザーがログインし、Oracle NetSuite インスタンスにアクセスするためのリクエストされた権限を AWS Glue に許可する必要があります。

- ユーザーは、Oracle NetSuite で独自の接続アプリを作成し、AWS Glue コンソールを介して接続を作成するときに独自のクライアント ID とクライアントシークレットを指定することを選択できます。このシナリオでは、引き続き Oracle NetSuite にリダイレクトされてログインし、リソースへアクセスするために AWS Glue を承認します。
- このグラントタイプは、更新トークンとアクセストークンになります。アクセストークンの有効期間は短く、更新トークンを使用してユーザーとやり取りすることなく自動的に更新される場合があります。
- 認可コード OAuth フロー用の接続アプリケーションの作成に関する Oracle NetSuite の公開ドキュメントについては、「パブリックアプリケーション」を参照してください。

Oracle NetSuite 接続を設定するには：

1. AWS Secrets Manager で、次の詳細を含むシークレットを作成します。
   a. カスタマーマネージド接続アプリケーションの場合、シークレットには、USER_MANAGED_CLIENT_APPLICATION_CLIENT_SECRET を使用して接続されたアプリケーションのコンシューマーシークレットをキーとして含める必要があります。
   b. 注: AWS Glue で接続ごとにシークレットを作成する必要があります。
2. AWS Glue Glue Studio で、以下の手順に従って [データ接続] の下に接続を作成します。
   a. [接続タイプ] を選択する際に、[Oracle NetSuite] を選択します。
   b. Oracle NetSuite 環境を指定します。
   c. 次のアクションを実行でき、AWS Glue がその権限を持つ AWS IAM ロールを選択します。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:DescribeSecret",
                "secretsmanager:GetSecretValue",
                "secretsmanager:PutSecretValue",
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterface",
                "ec2:DeleteNetworkInterface",
            ],
            "Resource": "*"
        }
    ]
```

```
}
```

d. AWS Glue でこの接続に使用する `secretName` を選択して、トークンを配置します。

e. ネットワークを使用する場合は、ネットワークオプションを選択します。

3. AWS Glue ジョブに関連付けられている IAM ロールに `secretName` を読み取るアクセス許可を付与します。

Oracle NetSuite エンティティからの読み取り

前提条件

読み取り元の Oracle NetSuite オブジェクト。`deposit` または `timebill` などのオブジェクト名が必要です。次の表に、サポートされているエンティティを示します。

ソースに対応するエンティティ:

エンティティ	フィルタリング可能	Order By をサポートする	制限をサポートする	SELECT * をサポートする	パーティション分割をサポートする
Deposit	はい	いいえ	はい	はい	はい
Description Item	はい	いいえ	はい	はい	はい
Inventory Item	はい	いいえ	はい	はい	はい
Item Fulfillment	はい	いいえ	はい	はい	はい
Item Group	はい	いいえ	はい	はい	はい
Journal Entry	はい	いいえ	はい	はい	はい
Non-Inventory Purchase Item	はい	いいえ	はい	はい	はい

エンティティ	フィルタリング可能	Order By をサポートする	制限をサポートする	SELECT * をサポートする	パーティション分割をサポートする
Non-Inventory Resale Item	はい	いいえ	はい	はい	はい
Non-Inventory Sale Item	はい	いいえ	はい	はい	はい
発注書	はい	いいえ	はい	はい	はい
Subsidiary	はい	いいえ	はい	はい	はい
Vendor	はい	いいえ	はい	はい	はい
Vendor Bill	はい	いいえ	はい	はい	はい
Vendor Return Authorization	はい	いいえ	はい	はい	はい
Time Bill	はい	いいえ	はい	はい	はい
Customer Payment	はい	いいえ	はい	はい	はい
Fulfillment Request	はい	いいえ	はい	はい	はい

例:

```
netsuiteerp_read = glueContext.create_dynamic_frame.from_options(
    connection_type="netsuiteerp",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "deposit",
        "API_VERSION": "v1"
    }
```

```
)
```

Oracle NetSuite エンティティとフィールドの詳細:

Oracle NetSuite は、選択したエンティティで使用可能なフィールドを動的にロードします。フィールドのデータ型に応じて、次のフィルター演算子がサポートされます。

フィールドデータタイプ	サポートされているフィルター演算子
文字列	LIKE, =, !=
日付	BETWEEN, =, <, <=, >, >=
DateTime	BETWEEN, <, <=, >, >=
数値	=, !=, <, <=, >, >=
ブール値	=, !=

パーティショニングクエリ

フィールドベースのパーティション分割

Oracle NetSuite コネクタには動的メタデータがあるため、フィールドベースのパーティショニングでサポートされているフィールドが動的に選択されます。フィールドベースのパーティショニングは、データ型が Integer、BigInteger、Date、または DateTime のフィールドでサポートされています。

Spark で同時実行を使用する場合は、追加の Spark オプション `PARTITION_FIELD`、`LOWER_BOUND`、`UPPER_BOUND`、および `NUM_PARTITIONS` を指定できます。これらのパラメータを使用すると、元のクエリは Spark タスクで同時に実行できるサブクエリの `NUM_PARTITIONS` の数に分割されます。

- `PARTITION_FIELD`: クエリのパーティション化に使用するフィールドの名前。
- `LOWER_BOUND`: 選択したパーティションフィールドの包括的な下限値。

  タイムスタンプフィールドでは、Spark SQL クエリで使用される Spark タイムスタンプ形式を受け入れます。

  有効な値の例は次のとおりです。

```
"TIMESTAMP \"1707256978123\""
"TIMESTAMP \"1702600882\""
"TIMESTAMP '2024-02-06T22:00:00:00.000Z'"
"TIMESTAMP '2024-02-06T22:00:00:00Z'"
"TIMESTAMP '2024-02-06'"
```

- UPPER_BOUND: 選択したパーティションフィールドの排他的上限値。
- NUM_PARTITIONS: パーティション数。

例：

```
oracle-netsuite_read = glueContext.create_dynamic_frame.from_options(
    connection_type="oracle-netsuite",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "company",
        "API_VERSION": "v3",
        "PARTITION_FIELD": "hs_object_id"
        "LOWER_BOUND": "50"
        "UPPER_BOUND": "16726619290"
        "NUM_PARTITIONS": "10"
    }
```

レコードベースのパーティション分割

Spark で並行処理を使用する場合は、追加の Spark オプション NUM_PARTITIONS を指定できます。これらのパラメータを使用すると、元のクエリは Spark タスクで同時に実行できるサブクエリの NUM_PARTITIONS の数に分割されます。

レコードベースのパーティショニングでは、存在するレコードの総数が Oracle NetSuite API からクエリされ、指定された NUM_PARTITIONS 数で除算され、結果のレコードの数が各サブクエリによって同時に取得されます。

- NUM_PARTITIONS: パーティション数。

例：

```
netsuiteerp_read = glueContext.create_dynamic_frame.from_options(
    connection_type="netsuiteerp",
```

```
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "deposit",
        "API_VERSION": "v1",
        "NUM_PARTITIONS": "3"
    }
```

Oracle NetSuite 接続オプション

Oracle の接続オプションは次のとおりです NetSuite。

- ENTITY_NAME (文字列) - (必須) 読み取りに使用されます。Oracle NetSuite エンティティの名前。例: 預金。
- API_VERSION (文字列) - (必須) 読み取りに使用されます。使用する Oracle NetSuite Rest API バージョン。Oracle NetSuite は現在バージョン v1 のみをサポートしているため、値は v1 になります。
- SELECTED_FIELDS (リスト < 文字列>) - デフォルト: empty(SELECT *)。読み込みに使用されます。選択したエンティティに選択する列のカンマ区切りリスト。
- FILTER_PREDICATE(文字列) - デフォルト: 空 読み込みに使用されます。Spark SQL形式である必要があります。
- QUERY(文字列) - デフォルト: 空 読み込みに使用されます。フル Spark SQLクエリ。
- PARTITION_FIELD (文字列) - 読み取りに使用されます。クエリをパーティション化するために使用するフィールド (フィールドベースのパーティション化) 。
- LOWER_BOUND(文字列) - 読み取りに使用されます。選択したパーティションフィールドの包括的な下限値。フィールドベースのパーティショニングで使用されます。
- UPPER_BOUND (文字列) - 読み取りに使用されます。選択したパーティションフィールドの排他的な上限値。フィールドベースのパーティショニングで使用されます。
- NUM_PARTITIONS(整数) - デフォルト: 1。読み込みに使用されます。読み取り用のパーティションの数。フィールドベースのパーティショニングとレコードベースのパーティショニングの両方で使用されます。
- INSTANCEE_URL (文字列) - URL形式の有効な NetSuite インスタンス https://{account-id}.suitetalk.api.netsuite.com。

Oracle NetSuite コネクタの制限と注意事項

Oracle NetSuite コネクタの制限事項または注意事項を次に示します。

- access_token パラメータと refresh_token パラメータの値は、JSONウェブトークン (JWT) 形式です。アクセストークンは 60 分間有効ですが、 refresh_token は 7 日間有効です。
- クライアント ID とクライアントシークレットの生成中に、PUBLICCLIENT「」と CODEGRANTAUTHORIZATION「」を選択した場合、更新トークンは 3 時間のみ有効で、1 回限り使用できます。
- コネクタを使用して、最大 1,000,000 件のレコードを取得できます。
- パーティションは、各パーティションが 1000 の倍数でレコードを取得するように作成されます。ただし、残りのレコードを取得する最後のパーティションは例外です。

AWS Glue Studio での OpenSearch Service に対する接続

AWS Glue は、Amazon OpenSearch Service のための組み込みサポートを提供します。AWS Glue Studio は、Amazon OpenSearch Service に接続してデータ統合ジョブをオーサリングし、AWS Glue Studio サーバーレス Spark ランタイム上でそれらのジョブを実行するためのビジュアルインターフェイスを提供します。この機能は OpenSearch サービスのサーバーレスとは互換性がありません。

AWS Glue Studio は Amazon OpenSearch Service の統合接続を作成します。詳細については、「考慮事項」を参照してください。

トピック

- OpenSearch Service 接続の作成
- OpenSearch Service のソースノードの作成
- OpenSearch Service ターゲットノードの作成
- 詳細オプション

OpenSearch Service 接続の作成

前提条件:

- 読み取り元とするドメインエンドポイント *aosEndpoint* とポート *aosPort* を特定するか、または Amazon OpenSearch Service ドキュメントの手順に従ってリソースを作成します。ドメインの作成の詳細については、Amazon OpenSearch Service ドキュメントの「Amazon OpenSearch Service ドメインの作成と管理」を参照してください。

Amazon OpenSearch Service ドメインエンドポイントのデフォルト形式は、https://search-*domainName-unstructuredIdContent*.*region*.es.amazonaws.com です。ドメインエンドポイントの識別の詳細については、Amazon OpenSearch Service ドキュメントの「Amazon OpenSearch Service ドメインの作成と管理」を参照してください。

ドメインの HTTP 基本認証情報、*aosUser*、および *aosPassword* を特定または生成します。

OpenSearch Service に対する接続を設定するには:

1. AWS Secrets Manager で、OpenSearch Service 認証情報を使用してシークレットを作成します。Secrets Manager でシークレットを作成するには、AWS Secrets Manager ドキュメントの「AWS Secrets Manager シークレットを作成する」にあるチュートリアルに従ってください。シークレットを作成したら、次のステップのためにシークレット名 *secretName* を保存しておきます。
   - [key/value ペア] を選択する際に、*aosUser* という値を持つキー USERNAME のペアを作成します。
   - [key/value ペア] を選択する際に、*aosPassword* という値を持つキー PASSWORD のペアを作成します。
2. AWS Glue コンソールで、「the section called “AWS Glue 接続の追加”」にあるステップに従って接続を作成します。接続を作成したら、将来的に AWS Glue で使用するために、接続名 *connectionName* を維持します。
   - [接続タイプ] を選択する場合は、[OpenSearch Service] を選択します。
   - ドメインエンドポイントを選択する場合は、*aosEndpoint* を入力します。
   - ポートを選択する場合は、*aosPort* を入力します。
   - [AWS Secret] をクリックして、*secretName* を入力します。

OpenSearch Service のソースノードの作成

必要な前提条件

- 前のセクション the section called “OpenSearch Service 接続の作成” で説明したような、AWS Secrets Manager シークレットを使用して設定された AWS Glue OpenSearch Service 接続。
- 接続で使用されるシークレットを読み取るためのジョブに対する適切なアクセス許可。
- 読み取り元とする OpenSearch Service インデックスである *aosIndex*。

OpenSearch Service データソースの追加

[データソース – OpenSearch Service] ノードを追加するには:

1. OpenSearch Service データソース用の接続を選択します。すでに作成したので、ドロップダウンに表示されているはずです。接続を作成する必要がある場合は、[OpenSearch Service 接続を作成] を選択します。詳細については、前の「the section called “OpenSearch Service 接続の作成”」セクションを参照してください。

   接続を選択したあとは、[プロパティを表示] をクリックすると、接続のプロパティを表示できます。
2. [インデックス] には、読み取り元とするインデックスを指定します。
3. オプションで、より具体的な結果を提供する OpenSearch クエリである [クエリ] を入力します。OpenSearch クエリの作成の詳細については、「the section called “OpenSearch Service から読み取る”」を参照してください。
4. [OpenSearch Service のカスタムプロパティ] で、必要に応じてパラメータと値を入力します。

OpenSearch Service ターゲットノードの作成

必要な前提条件

- 前のセクション the section called “OpenSearch Service 接続の作成” で説明したような、AWS Secrets Manager シークレットを使用して設定された AWS Glue OpenSearch Service 接続。
- 接続で使用されるシークレットを読み取るためのジョブに対する適切なアクセス許可。
- 書き込み先とする OpenSearch Service インデックスである *aosIndex*。

OpenSearch Service データターゲットの追加

[データターゲット – OpenSearch Service] ノードを追加するには:

1. OpenSearch Service データソース用の接続を選択します。すでに作成したので、ドロップダウンに表示されているはずです。接続を作成する必要がある場合は、[OpenSearch Service 接続を作成] を選択します。詳細については、前の「the section called “OpenSearch Service 接続の作成”」セクションを参照してください。

   接続を選択したあとは、[プロパティを表示] をクリックすると、接続のプロパティを表示できます。

2. [インデックス] には、読み取り元とするインデックスを指定します。
3. [OpenSearch Service のカスタムプロパティ] で、必要に応じてパラメータと値を入力します。

詳細オプション

OpenSearch Service ノードを作成する際に、高度なオプションを指定できます。これらのオプションは Spark AWS Glue スクリプトのプログラミング時に使用できるオプションと同じです。

「the section called “OpenSearch Service 接続”」を参照してください。

Okta への接続

Okta API は、Okta へのプログラムによるインターフェイスであり、大規模または複雑な Okta アカウントやキャンペーンの管理に使用されます。Okta ユーザーの場合は、Okta アカウントに AWS Glue を接続できます。その後、Okta を ETL ジョブのデータソースとして使用できます。これらのジョブを実行して、Okta と AWS サービス、またはその他のサポートされているアプリケーション間でデータを転送します。

トピック

- AWS Glue の Okta のサポート
- 接続を作成および使用するための API オペレーションを含むポリシー
- Okta の設定
- Okta 接続の設定
- Okta エンティティからの読み取り
- Okta 接続オプションのリファレンス
- Okta の新しいアカウントとデベロッパーアプリの作成ステップ
- 制限

AWS Glue の Okta のサポート

AWS Glue は、次のように Okta をサポートします。

ソースとしてサポートされていますか?

はい。AWS Glue ETL ジョブを使用して、Okta からデータをクエリできます。

ターゲットとしてサポートされていますか?

いいえ。

サポートされている Okta API バージョン

v1。

接続を作成および使用するための API オペレーションを含むポリシー

次のサンプル ポリシーで、接続の作成と使用に必要な AWS の権限について説明します。新しいロールを作成する場合は、以下を含むポリシーを作成します。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:ListConnectionTypes",
                "glue:DescribeConnectionType",
                "glue:RefreshOAuth2Tokens"
                "glue:ListEntities",
                "glue:DescribeEntity"
            ],
            "Resource": "*"
        }
    ]
}
```

また、次のマネージド IAM ポリシーを使用して、アクセスを許可できます。

- AWSGlueServiceRole – さまざまな AWS Glue プロセスを実行するために必要なリソースへのアクセス権をユーザーに代わって付与します。これらのリソースには、AWS Glue、Amazon S3、IAM、CloudWatch Logs、および Amazon EC2 が含まれます。このポリシーで指定されたリソースの命名規則に従った場合、AWS Glue プロセスは必要なアクセス権限を使用できます。このポリシーは、通常、クローラ、ジョブ、開発エンドポイントを定義するときに指定されたロールにアタッチされます。
- AWSGlueConsoleFullAccess — ポリシーがアタッチされているアイデンティティが AWS マネージメントコンソールを使用するときは、AWS Glue リソースへのフルアクセスを許可します。このポリシーで指定されたリソースの命名規則に従った場合、ユーザーは完全なコンソール機能を使用できます。このポリシーは、通常 AWS Glue コンソールのユーザーにアタッチされています。

Okta の設定

AWS Glue を使用して Okta との間でデータを転送できるようにするには、次の要件を満たす必要があります。

最小要件

- Okta アカウントがある。アカウントの作成の詳細については、「Okta の新しいアカウントとデベロッパーアプリの作成ステップ」を参照してください。
- Okta アカウントで API アクセスが有効になっている。
- OAuth2 API 統合が Okta アカウントで作成された。この統合により、AWS Glue がお客様のアカウントに対して認証呼び出しを行うときに、お客様のデータに安全にアクセスするために使用するクライアントの認証情報が提供されます。詳細については、クライアントアプリケーションと OAuth2.0 認証情報を作成するステップ:「Okta の新しいアカウントとデベロッパーアプリの作成ステップ」を参照してください。
- OktaApiToken を持つ Okta アカウントがある。Okta のドキュメントを参照してください。

これらの要件を満たすと、Okta アカウントに AWS Glue を接続する準備が整います。一般的な接続では、Okta で他に何もする必要はありません。

Okta 接続の設定

Okta は 2 種類の認可メカニズムをサポートしています。

- OAuth 認可: Okta は `AUTHORIZATION_CODE` グラントタイプをサポートしています。
  - このグラントタイプは、ユーザーを認証するためにサードパーティーの認証サーバーにユーザーをリダイレクトすることから、「three-legged」の OAuth と見なされます。AWS Glue コンソール経由で接続を作成するときに使用されます。AWS Glue コンソールでは、ユーザーが Okta にリダイレクトされます。ここではユーザーがログインし、Okta インスタンスにアクセスするためのリクエストされた権限を AWS Glue に許可する必要があります。
  - ユーザーは、Okta で独自の接続アプリケーションを作成し、AWS Glue コンソールを介して接続を作成するときに独自のクライアント ID とクライアントシークレットを指定することを選択できます。このシナリオでは、引き続き Okta にリダイレクトされてログインし、リソースへアクセスするために AWS Glue を認可します。
  - このグラントタイプは、更新トークンとアクセストークンになります。アクセストークンの有効期間は短く、更新トークンを使用してユーザーとやり取りすることなく自動的に更新される場合があります。

  - 詳細については、OAuth 認可コードフローでの接続アプリケーションの作成に関する Okta の公開ドキュメントを参照してください。
- カスタム認可:
  - カスタム認可に必要な API キーの生成に関する Okta の公開ドキュメントについては、Okta のドキュメントを参照してください。

Okta 接続を設定するには:

1. AWS Secrets Manager で、次の詳細を含むシークレットを作成します。AWS Glue で接続ごとにシークレットを作成する必要があります。
   a. OAuth 認可の場合:
      - カスタマーマネージド接続アプリケーションの場合 – シークレットには、USER_MANAGED_CLIENT_APPLICATION_CLIENT_SECRET を使用して接続されたアプリケーションのコンシューマーシークレットをキーとして含める必要があります。
   b. カスタム認可の場合:
      - カスタマーマネージド接続アプリケーションの場合 – シークレットには、OktaApiToken を使用して接続されたアプリケーションのコンシューマーシークレットをキーとして含める必要があります。
2. AWS Glue Studio で、以下の手順に従って [データ接続] の下に接続を作成します。
   a. [接続] で [接続を作成] を選択します。
   b. [データソース] を選択する際に、[Okta] を選択します。
   c. Okta サブドメインを指定します。
   d. Okta アカウントの Okta ドメイン URL を選択します。
   e. AWS Glue が次のアクションを担い、その権限を持つことができる IAM ロールを選択します。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:DescribeSecret",
                "secretsmanager:GetSecretValue",
                "secretsmanager:PutSecretValue",
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterface",
```

```
                "ec2:DeleteNetworkInterface",
            ],
            "Resource": "*"
        }
    ]
}
```

f. データソースに接続するための認可タイプを選択します。

g. OAuth2 認可タイプの場合は、Okta アプリケーションの [ユーザーマネージドクライアントアプリケーション ClientId] を指定します。

h. AWS Glue でこの接続に使用する `secretName` を選択して、トークンを配置します。

i. ネットワークを使用する場合は、ネットワークオプションを選択します。

3. AWS Glue ジョブに関連付けられている IAM ロールに `secretName` を読み取るアクセス許可を付与します。

4. AWS Glue ジョブ設定で、[追加のネットワーク接続] として `connectionName` を指定します。

Okta エンティティからの読み取り

前提条件

- 読み取り元の Okta オブジェクト。使用可能なエンティティを確認するには、以下のサポートされているエンティティの表を参照してください。

サポートされているエンティティ

エンティティ	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
アプリケーション	あり	はい	なし	あり	不可
デバイス	あり	はい	なし	あり	あり
グループ	あり	あり	あり	あり	あり
[ユーザー]	あり	あり	あり	あり	あり

エンティティ	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
ユーザータイプ	いいえ	いいえ	なし	あり	不可

例

```
okta_read = glueContext.create_dynamic_frame.from_options(
    connection_type="Okta",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "applications",
        "API_VERSION": "v1"
    }
```

Okta エンティティとフィールドの詳細

エンティティリスト:

- アプリケーション: https://developer.okta.com/docs/api/openapi/okta-management/management/tag/Application/
- デバイス: https://developer.okta.com/docs/api/openapi/okta-management/management/tag/Device/
- グループ: https://developer.okta.com/docs/api/openapi/okta-management/management/tag/Group/
- ユーザー: https://developer.okta.com/docs/api/openapi/okta-management/management/tag/User/
- ユーザータイプ: https://developer.okta.com/docs/api/openapi/okta-management/management/tag/UserType/

パーティショニングクエリ

Spark で同時実行を使用する場合は、追加の Spark オプション PARTITION_FIELD、LOWER_BOUND、UPPER_BOUND、および NUM_PARTITIONS を指定できます。これらのパラメータを使用すると、元のクエリは Spark タスクで同時に実行できるサブクエリの NUM_PARTITIONS の数に分割されます。

- PARTITION_FIELD: クエリのパーティション化に使用するフィールドの名前。

- LOWER_BOUND: 選択したパーティションフィールドの包括的な下限値。

  日付については、Spark SQL クエリで使用される Spark の日付形式を受け入れます。有効な値の例: "2024-02-06"。
- UPPER_BOUND: 選択したパーティションフィールドの排他的上限値。
- NUM_PARTITIONS: パーティション数。

例

```
okta_read = glueContext.create_dynamic_frame.from_options(
    connection_type="okta",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "lastUpdated",
        "API_VERSION": "v1",
        "PARTITION_FIELD": "lastMembershipUpdated"
        "LOWER_BOUND": "2022-08-10T10:28:46.000Z"
        "UPPER_BOUND": "2024-08-10T10:28:46.000Z"
        "NUM_PARTITIONS": "10"
    }
```

Okta 接続オプションのリファレンス

Okta の接続オプションは次のとおりです。

- ENTITY_NAME(文字列) – (必須) 読み取り / 書き込みに使用されます。Okta のオブジェクトの名前。
- API_VERSION(文字列) – (必須) 読み取り / 書き込みに使用されます。使用する Okta Rest API バージョン。例: v1。
- SELECTED_FIELDS(List<String>) – Default: empty(SELECT *). 読み込みに使用されます。オブジェクトに選択する列。
- FILTER_PREDICATE(文字列) – デフォルト: 空 読み込みに使用されます。Spark SQL 形式である必要があります。
- QUERY(文字列) - デフォルト: 空 読み込みに使用されます。完全な Spark SQL クエリ。
- PARTITION_FIELD(文字列) – 読み取りに使用されます。クエリをパーティション化するために使用するフィールド。

- LOWER_BOUND(文字列) - 読み取りに使用されます。選択したパーティションフィールドの包括的な下限値。
- UPPER_BOUND (文字列) - 読み取りに使用されます。選択したパーティションフィールドの排他的上限値。
- NUM_PARTITIONS(整数) – デフォルト: 1。読み込みに使用されます。読み取り用のパーティションの数。

Okta の新しいアカウントとデベロッパーアプリの作成ステップ

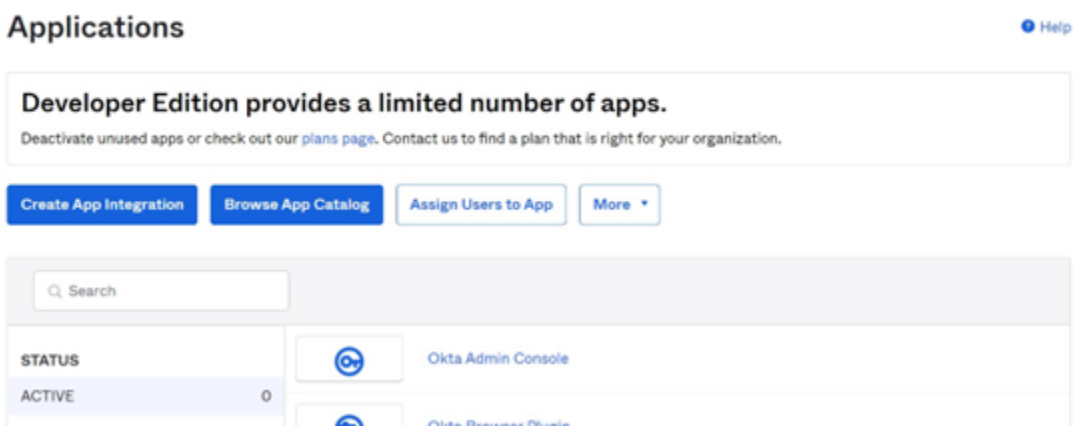
Okta API にアクセスするためのデベロッパーアカウントを Okta に作成します。無料の Okta デベロッパーアカウントでは、Okt API へのアクセスに必要なほとんどの主要なデベロッパー機能にアクセスできます。

Okta でデベロッパーアカウントを作成するには

1. https://developer.okta.com/signup/ に移動します。
2. アカウント情報の E メール、名、姓、国/地域を入力します。[私はロボットではありません] を選択し、[サインアップ] を選択します。
3. 登録したメール ID に確認メールが送信されます。Okta デベロッパーアカウントをアクティブ化するためのリンクを E メールで受信します。[有効化] を選択します。
4. パスワードのリセットのページにリダイレクトされます。新しいパスワードを 2 回入力し、[パスワードのリセット] を選択します。
5. Okta デベロッパーアカウントダッシュボードにリダイレクトされます。

クライアントアプリケーションと OAuth 2.0 認証情報を作成するには

1. デベロッパーダッシュボードで、[アプリケーションの統合の作成] を選択します。



2. [新しいアプリケーション統合の作成] ウィンドウが表示され、さまざまなサインイン方法が表示されます。[OIDC –OpenID Connect] を選択します。
3. [アプリケーションタイプ] セクションまでスクロールダウンします。[ウェブアプリケーション] として選択し、[次へ] を選択します。
4. [新規ウェブアプリケーション統合] 画面で、次の情報を入力します。
   - [アプリケーション統合名] - アプリケーションの名前を入力します。
   - [グラントタイプ] - リストから [認可コード] と [更新トークン] () を選択します。
   - [サインインリダイレクト URI] - [URI を追加] を選択し、`https://{regioncode}.console.aws.amazon.com/appflow/oauth` を追加します。例えば、`us-west-2 (Oregon)` を使用している場合は、`https://us-east-1.console.aws.amazon.com/appflow/oauth` を追加できます。
   - [制御されたアクセス] - 必要に応じてユーザーグループにアプリケーションを割り当て、[保存] を選択します。
5. クライアント ID とクライアントシークレットが生成されます。

制限

Okta コネクタの制限は次のとおりです。

- 「アプリケーション」エンティティの場合、適用できるフィルターは 1 つのみです。複数のフィルターが適用されると、400 Bad Request が返され、エラーの概要は「検索条件が無効です」となります。

- 並べ替え条件は、検索クエリでのみサポートできます。例えば、 `http://dev-15940405.okta.com/api/v1/groups?search=type e.q. "OKTA_GROUP"&sortBy=lastUpdated&sortOrder=asc`

PayPal に対する接続

PayPal は、顧客とオンラインベンダー間の送金など、当事者間のオンライン送金を容易にする支払いシステムです。PayPal ユーザーの場合、アカウントには支払者、日付、ステータスなど、トランザクションに関するデータが含まれます。AWS Glue を使用して、PayPal から特定の AWS サービスや他のサポートされているアプリケーションにデータを転送できます。

トピック

- PayPal 向けの AWS Glue サポート
- 接続を作成および使用するための API オペレーションを含むポリシー
- PayPal の設定
- PayPal 接続の設定
- PayPal エンティティからの読み取り
- PayPal の接続オプション
- PayPal コネクタの制限事項と注意事項

PayPal 向けの AWS Glue サポート

AWS Glue は、次のように PayPal をにサポートします。

ソースとしてサポートされていますか?

はい。AWS Glue ETL ジョブを使用して、PayPal からデータをクエリできます。

ターゲットとしてサポートされていますか?

いいえ。

サポートされている PayPal API バージョン

次の PayPal API バージョンがサポートされています。

- v1

接続を作成および使用するための API オペレーションを含むポリシー

次のサンプル ポリシーで、接続の作成と使用に必要な AWS IAM 権限について説明します。新しいロールを作成する場合は、以下を含むポリシーを作成します。

```
{
 "Version": "2012-10-17",
    "Statement": [
     {
            "Effect": "Allow",
            "Action": [
                "glue:ListConnectionTypes",
                "glue:DescribeConnectionType",
                "glue:RefreshOAuth2Tokens",
                "glue:ListEntities",
                "glue:DescribeEntity"
            ],
            "Resource": "*"
        }
    ]
 }
```

上記の方法を使用しない場合は、代わりに次のマネージド IAM ポリシーを使用します:

- AWSGlueServiceRole – さまざまな AWS Glue プロセスを実行するために必要なリソースへのアクセス権をユーザーに代わって付与します。これらのリソースには AWS Glue 、Amazon S3、IAM、CloudWatch Logs、および Amazon EC2 が含まれます。このポリシーで指定されたリソースの命名規則に従った場合、AWS Glue プロセスは必要なアクセス権限を使用できます。このポリシーは、通常、クローラ、ジョブ、開発エンドポイントを定義するときに指定されたロールにアタッチされます。
- AWSGlueConsoleFullAccess — ポリシーがアタッチされているアイデンティティが AWS マネージメントコンソールを使用するときは、AWS Glue リソースへのフルアクセスを許可します。このポリシーで指定されたリソースの命名規則に従った場合、ユーザーは完全なコンソール機能を使用できます。このポリシーは、通常 AWS Glue コンソールのユーザーにアタッチされています。

PayPal の設定

AWS Glue を使用して PayPal からデータを転送できるようにするには、次の要件を満たす必要があります。

最小要件

以下に、最小要件を示します。

- クライアント認証情報を持つ PayPal アカウントがある。
- PayPal アカウントに、有効なライセンスを持つ API アクセスがある。

これらの要件を満たすと、PayPal アカウントに AWS Glue を接続する準備が整います。一般的な接続では、PayPal で他に何もする必要はありません。

PayPal 接続の設定

PayPal は、OAuth2 の CLIENT CREDENTIALS 許可タイプをサポートしています。

- このグラントタイプは、クライアントがユーザーのコンテキスト外でアクセストークンを取得するために使用するため、2-legged の OAuth 2.0 と見なされます。AWS Glue は、クライアント ID とクライアントシークレットを使用して、定義したカスタムサービスによって提供される PayPal API を認証できます。
- 各カスタムサービスは、特定のアクションを実行することをサービスに許可する一連のロールとアクセス許可を持つ API 専用ユーザーによって所有されます。アクセストークンは 1 つのカスタムサービスに関連付けられています。
- このグラントタイプは、有効期間が短いアクセストークンになり、`/v2/oauth2/token` エンドポイントを再び呼び出すことで更新される場合があります。
- クライアント認証情報を使用した OAuth 2.0 の PayPal の公開ドキュメントについては、「認証」を参照してください。

PayPal 接続を設定するには:

1. AWS Secrets Manager で、次の詳細を含むシークレットを作成します。
   a. カスタマーマネージド接続アプリケーションの場合、シークレットには、`USER_MANAGED_CLIENT_APPLICATION_CLIENT_SECRET` を使用して接続されたアプリケーションのコンシューマーシークレットをキーとして含める必要があります。
   b. 注: AWS Glue で接続にシークレットを作成する必要があります。

1. AWS Glue Glue Studio で、以下の手順に従って [データ接続] の下に接続を作成します。
   a. [接続タイプ] を選択する際に、[PayPal] を選択します。

b. 接続先の PayPal インスタンスの INSTANCE_URL を指定します。

c. 次のアクションを実行でき、AWS Glue がその権限を持つ AWS IAM ロールを選択します。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:DescribeSecret",
                "secretsmanager:GetSecretValue",
                "secretsmanager:PutSecretValue",
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterface",
                "ec2:DeleteNetworkInterface",
            ],
            "Resource": "*"
        }
    ]
}
```

d. AWS Glue でこの接続に使用する secretName を選択して、トークンを配置します。

e. ネットワークを使用する場合は、ネットワークオプションを選択します。

2. AWS Glue ジョブに関連付けられている IAM ロールに secretName を読み取るアクセス許可を付与します。

OAuth 2.0 認証情報の取得

Rest API を呼び出すには、クライアント ID とクライアントシークレットをアクセストークンと交換する必要があります。詳細については、「PayPal REST API の使用を開始する」を参照してください。

PayPal エンティティからの読み取り

前提条件

読み取り元の PayPal オブジェクト。オブジェクト名 transaction が必要になります。

ソースに対応するエンティティ:

エンティティ	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
トランザクション	はい	はい	いいえ	はい	はい

例:

```
paypal_read = glueContext.create_dynamic_frame.from_options(
    connection_type="paypal",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "transaction",
        "API_VERSION": "v1",
        "INSTANCE_URL": "https://api-m.paypal.com"
    }
```

PayPal エンティティとフィールドの詳細:

静的メタデータを持つエンティティ:

エンティティ	フィールド	データ型	サポートされている演算子
トランザクション	transaction_initiation_date	DateTime	次の間
	last_refreshed_datetime	String	該当なし
	payment_instrument_type	String	=
	balance_affecting_records_only	String	=
	store_id	String	=

エンティティ	フィールド	データ型	サポートされている演算子
	terminal_id	String	=
	transaction_currency	String	=
	transaction_id	String	該当なし
	transaction_status	String	該当なし
	transaction_type	String	該当なし
	transaction_info	Struct	該当なし
	payer_info	Struct	該当なし
	shipping_info	Struct	該当なし
	cart_info	Struct	該当なし
	store_info	Struct	該当なし
	auction_info	Struct	該当なし
	incentive_info	Struct	該当なし

パーティショニングクエリ

Spark で同時実行を使用する場合は、追加の Spark オプション PARTITION_FIELD、LOWER_BOUND、UPPER_BOUND、および NUM_PARTITIONS を指定できます。これらのパラメータを使用すると、元のクエリは Spark タスクで同時に実行できるサブクエリの NUM_PARTITIONS の数に分割されます。

- PARTITION_FIELD: クエリのパーティション化に使用するフィールドの名前。
- LOWER_BOUND: 選択したパーティションフィールドの包括的な下限値。

  Datetime フィールドでは、ISO 形式の値を受け入れます。

  有効な値の例は次のとおりです:

```
"2024-07-01T00:00:00.000Z"
```

- UPPER_BOUND: 選択したパーティションフィールドの排他的上限値。
- NUM_PARTITIONS: パーティション数。

次のフィールドは、エンティティ単位のパーティション分割でサポートされています。

エンティティ名	パーティション分割フィールド	データ型
トランザクション	transaction_initiation_date	DateTime

例:

```
paypal_read = glueContext.create_dynamic_frame.from_options(
    connection_type="paypal",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "transaction",
        "API_VERSION": "v1",
        "PARTITION_FIELD": "transaction_initiation_date"
        "LOWER_BOUND": "2024-07-01T00:00:00.000Z"
        "UPPER_BOUND": "2024-07-02T00:00:00.000Z"
        "NUM_PARTITIONS": "10"
    }
```

PayPal の接続オプション

PayPal の接続オプションは次のとおりです。

- ENTITY_NAME(文字列) – (必須) 読み取りに使用されます。PayPal のオブジェクトの名前。
- API_VERSION(文字列) – (必須) 読み取りに使用されます。使用する PayPal Rest API バージョン。
- SELECTED_FIELDS(List<String>) - Default: empty(SELECT *). 読み込みに使用されます。オブジェクトに選択する列。
- FILTER_PREDICATE(文字列) – デフォルト: 空 読み込みに使用されます。Spark SQL 形式である必要があります。

- QUERY(文字列) – デフォルト: 空 読み込みに使用されます。完全な Spark SQL クエリ。
- PARTITION_FIELD(文字列) – 読み取りに使用されます。クエリをパーティション化するために使用するフィールド。
- LOWER_BOUND(文字列) - 読み取りに使用されます。選択したパーティションフィールドの包括的な下限値。
- UPPER_BOUND(文字列) – 読み取りに使用されます。選択したパーティションフィールドの排他的上限値。
- NUM_PARTITIONS(整数) – デフォルト: 1。読み込みに使用されます。読み取り用のパーティションの数。

PayPal コネクタの制限事項と注意事項

PayPal コネクタの制限事項または注意事項は次のとおりです。

- PayPal トランザクションドキュメントでは、実行されたトランザクションがトランザクション一覧の呼び出しに表示されるまでに最大 3 時間かかることが示されています。ただし、last_refreshed_datetime によってはそれよりも時間がかかることが確認されています。ここで、last_refreshed_datetime は API からデータが利用可能になるまでの時間。
- last_refreshed_datetime がリクエストされた end_date よりも小さい場合、end_date は last_refreshed_datetime に等しくなります。これは、その時点までのデータしかないためです。
- transaction_initiation_date フィールドは transaction エンティティに提供する必須のフィルターであり、このフィールドでサポートされる最大日付範囲は 31 日です。
- transaction_initiation_date フィールド以外のフィルター (クエリパラメータ) を使用して transaction エンティティ API リクエストを呼び出すと、ending_balance フィールドの値がレスポンスで取得されないことが予想されます。

Pendo への接続

Pendo は、ユーザーインタラクションデータ用の豊富なデータストアを提供します。お客様は、このデータを AWS に転送し、必要に応じて、他の製品データと結合したり、追加の分析やダッシュボードを実行したり、アラートを設定したりできるようにします。

トピック

- AWS Glue の Pendo のサポート

- 接続を作成および使用するための API オペレーションを含むポリシー
- Pendo の設定
- Pendo 接続の設定
- Pendo エンティティからの読み取り
- Pendo 接続オプション
- 制限

AWS Glue の Pendo のサポート

AWS Glue は、次のように Pendo をサポートします。

ソースとしてサポートされていますか?

はい。AWS Glue ETL ジョブを使用して、Pendo からデータをクエリできます。

ターゲットとしてサポートされていますか?

いいえ。

サポートされている Pendo API バージョン

v1

接続を作成および使用するための API オペレーションを含むポリシー

次のサンプル ポリシーで、接続の作成と使用に必要な AWS の権限について説明します。新しいロールを作成する場合は、以下を含むポリシーを作成します。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:ListConnectionTypes",
                "glue:DescribeConnectionType",
                "glue:RefreshOAuth2Tokens"
                "glue:ListEntities",
```

```
                "glue:DescribeEntity"
            ],
            "Resource": "*"
        }
    ]
}
```

上記の方法を使用しない場合は、代わりに次のマネージド IAM ポリシーを使用します。

- AWSGlueServiceRole – さまざまな AWS Glue プロセスを実行するために必要なリソースへのアクセス権をユーザーに代わって付与します。これらのリソースには、AWS Glue、Amazon S3、IAM、CloudWatch Logs、および Amazon EC2 が含まれます。このポリシーで指定されたリソースの命名規則に従った場合、AWS Glue プロセスは必要なアクセス権限を使用できます。このポリシーは、通常、クローラ、ジョブ、開発エンドポイントを定義するときに指定されたロールにアタッチされます。
- AWSGlueConsoleFullAccess — ポリシーがアタッチされているアイデンティティが AWS マネージメントコンソールを使用するときは、AWS Glue リソースへのフルアクセスを許可します。このポリシーで指定されたリソースの命名規則に従った場合、ユーザーは完全なコンソール機能を使用できます。このポリシーは、通常 AWS Glue コンソールのユーザーにアタッチされています。

Pendo の設定

AWS Glue を使用して Pendo から転送できるようにするには、次の要件を満たす必要があります。

最小要件

- `write access` が有効になっている `apiKey` を持つ Pendo アカウントがある。
- Pendo アカウントに、有効なライセンスを持つ API アクセスがある。

これらの要件を満たすと、Pendo アカウントに AWS Glue を接続する準備が整います。一般的な接続では、Pendo で他に何もする必要はありません。

Pendo 接続の設定

Pendo はカスタム認可をサポートしています。

カスタム認可に必要な API キーの生成に関する Pendo の公開ドキュメントについては、Pendo REST API ドキュメントの認可に関するページを参照してください。

Pendo 接続を設定するには:

1. AWS Secrets Manager で、次の詳細を含むシークレットを作成します。
   - カスタマーマネージド接続アプリケーションの場合 – シークレットには、apiKey を使用して接続されたアプリケーションのコンシューマーシークレットをキーとして含める必要があります。

   > **Note**
   >
   > AWS Glue で接続ごとにシークレットを作成する必要があります。

2. AWS Glue Studio で、以下の手順に従って [データ接続] の下に接続を作成します。
   a. [データソース] を選択する際に、[Pendo] を選択します。
   b. 接続先の Pendo インスタンスの instanceUrl を指定します。
   c. AWS Glue が次のアクションを担い、その権限を持つことができる IAM ロールを選択します。

```
{
    "Version": "2012-10-17",
        "Statement": [
            {
                "Effect": "Allow",
                "Action": [
                    "secretsmanager:DescribeSecret",
                    "secretsmanager:GetSecretValue",
                    "secretsmanager:PutSecretValue",
                    "ec2:CreateNetworkInterface",
                    "ec2:DescribeNetworkInterface",
                    "ec2:DeleteNetworkInterface",
                ],
                "Resource": "*"
            }
        ]
    }
```

   d. AWS Glue でこの接続に使用する secretName を選択して、トークンを配置します。
   e. ネットワークを使用する場合は、ネットワークオプションを選択します。
3. AWS Glue ジョブに関連付けられている IAM ロールに secretName を読み取るアクセス許可を付与します。
4. AWS Glue ジョブ設定で、[追加のネットワーク接続] として connectionName を指定します。

Pendo エンティティからの読み取り

前提条件

読み取り元の Pendo オブジェクト。使用可能なエンティティを確認するには、以下のサポートされているエンティティの表を参照してください。

サポートされているエンティティ

- 機能
- ガイド
- ページ
- レポートを行う
- レポートデータ
- 訪問者
- アカウント
- イベント
- フィーチャーイベント
- ガイドイベント
- ページイベント
- 投票調査イベント
- トラックイベント

エンティティ	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
機能	いいえ	いいえ	なし	あり	不可
ガイド	いいえ	いいえ	なし	あり	不可
ページ	いいえ	いいえ	なし	あり	不可
レポートを行う	いいえ	いいえ	なし	あり	不可

エンティティ	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
レポートデータ	いいえ	いいえ	なし	あり	不可
訪問者 (Aggregation API)	あり	なし	あり	はい	不可
アカウント (Aggregation API)	あり	なし	あり	はい	不可
イベント (Aggregation API)	あり	なし	あり	はい	不可
フィーチャーイベント (Aggregation API)	あり	なし	あり	あり	あり
ガイドイベント (Aggregation API)	あり	なし	あり	あり	あり
アカウント (Aggregation API)	あり	なし	あり	あり	あり
ページイベント (Aggregation API)	あり	なし	あり	あり	あり

エンティティ	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
投票調査イベント (Aggregation API)	あり	なし	あり	あり	あり
トラックイベント (Aggregation API)	あり	なし	あり	あり	あり

例

```
Pendo_read = glueContext.create_dynamic_frame.from_options(
    connection_type="glue.spark.Pendo",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "feature",
        "API_VERSION": "v1",
        "INSTANCE_URL": "instanceUrl"
    }
```

パーティショニングクエリ

Spark で同時実行を使用する場合は、追加の Spark オプション PARTITION_FIELD、LOWER_BOUND、UPPER_BOUND、および NUM_PARTITIONS を指定できます。これらのパラメータを使用すると、元のクエリは Spark タスクで同時に実行できるサブクエリの NUM_PARTITIONS の数に分割されます。

- PARTITION_FIELD: クエリのパーティション化に使用するフィールドの名前。
- LOWER_BOUND: 選択したパーティションフィールドの包括的な下限値。

  DateTime フィールドでは、ISO 形式の値を受け入れます。

  有効な値の例:

```
"2024-07-01T00:00:00.000Z"
```

- UPPER_BOUND: 選択したパーティションフィールドの排他的上限値。
- NUM_PARTITIONS: パーティション数。

次の表に、エンティティパーティション分割フィールドのサポートの詳細を示します:

エンティティ名
イベント
フィーチャーイベント
ガイドイベント
ページイベント
投票調査イベント
トラックイベント

例:

```
pendo_read = glueContext.create_dynamic_frame.from_options(
    connection_type="glue.spark.pendo",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "event",
        "API_VERSION": "v1",
        "INSTANCE_URL": "instanceUrl"
        "NUM_PARTITIONS": "10",
        "PARTITION_FIELD": "appId"
        "LOWER_BOUND": "4656"
        "UPPER_BOUND": "7788"
    }
```

Pendo 接続オプション

Pendo の接続オプションは次のとおりです。

- ENTITY_NAME(文字列) – (必須) 読み取り / 書き込みに使用されます。Pendo のオブジェクトの名前。
- INSTANCE_URL(文字列) - (必須) 以下の許容値を持つ有効な Pendo インスタンス URL。
  - デフォルト
  - 欧州
  - US1
- API_VERSION(文字列) – (必須) 読み取りに使用されます。使用する Pendo Engage Rest API バージョン。例: 3.0。
- SELECTED_FIELDS(List<String>) – Default: empty(SELECT *). 読み込みに使用されます。オブジェクトに選択する列。
- FILTER_PREDICATE(文字列) – デフォルト: 空 読み込みに使用されます。Spark SQL 形式である必要があります。
- QUERY(文字列) - デフォルト: 空 読み込みに使用されます。完全な Spark SQL クエリ。
- PARTITION_FIELD(文字列) – 読み取りに使用されます。クエリをパーティション化するために使用するフィールド。
- LOWER_BOUND(文字列) - 読み取りに使用されます。選択したパーティションフィールドの包括的な下限値。
- UPPER_BOUND (文字列) - 読み取りに使用されます。選択したパーティションフィールドの排他的上限値。
- NUM_PARTITIONS(整数) – デフォルト: 1。読み込みに使用されます。読み取り用のパーティションの数。

制限

Pendo コネクタの制限は次のとおりです。

- ページ分割は Pendo でサポートされません。
- フィルタリングは Aggregate API オブジェクト (Account、Event、Feature Event、Guide Events、Page Event、Poll Event、Track Event、Visitor) でのみサポートされます
- DateTimeRange は Aggregate API オブジェクト (Event、Feature Event、Guide Events、Page Event、Poll Event, Track Event) の必須フィルターパラメータです
- dayRange 期間は、タイムゾーンの期間の開始時に切り下げられます。例えば、指定されたフィルターが 2023-01-12T07:55:27.065Z の場合、この期間は期間の開始時、つまり 2023-01-12T00:00:00Z に丸められます。

Pipedrive への接続

Pipedrive は、小規模ビジネスがリードを管理し、営業活動を追跡し、より多くの取引を成立させるのに役立つように設計された販売パイプライン CRM です。Pipedrive を使用すると、小規模ビジネスの販売チームは、プロセスを合理化し、販売データを 1 つの統合された CRM 販売ツールに統合できます。Pipedrive ユーザーの場合は、Pipedrive アカウントに AWS Glue を接続できます。その後、Pipedrive を ETL ジョブのデータソースとして使用できます。これらのジョブを実行して、Pipedrive と AWS サービス、またはその他のサポートされているアプリケーション間でデータを転送します。

トピック

- AWS Glue の Pipedrive のサポート
- 接続を作成および使用するための API オペレーションを含むポリシー
- Pipedrive の設定
- Pipedrive 接続の設定
- Pipedrive エンティティからの読み取り
- Pipedrive 接続オプションのリファレンス
- 制限

AWS Glue の Pipedrive のサポート

AWS Glue は、次のように Pipedrive をサポートします。

ソースとしてサポートされていますか?

はい。AWS Glue ETL ジョブを使用して、Pipedrive からデータをクエリできます。

ターゲットとしてサポートされていますか?

いいえ。

サポートされている Pipedrive API バージョン

v1。

接続を作成および使用するための API オペレーションを含むポリシー

次のサンプル ポリシーで、接続の作成と使用に必要な AWS の権限について説明します。新しいロールを作成する場合は、以下を含むポリシーを作成します。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:ListConnectionTypes",
                "glue:DescribeConnectionType",
                "glue:RefreshOAuth2Tokens"
                "glue:ListEntities",
                "glue:DescribeEntity"
            ],
            "Resource": "*"
        }
    ]
}
```

また、次のマネージド IAM ポリシーを使用して、アクセスを許可できます。

- AWSGlueServiceRole – さまざまな AWS Glue プロセスを実行するために必要なリソースへのアクセス権をユーザーに代わって付与します。これらのリソースには、AWS Glue、Amazon S3、IAM、CloudWatch Logs、および Amazon EC2 が含まれます。このポリシーで指定されたリソースの命名規則に従った場合、AWS Glue プロセスは必要なアクセス権限を使用できます。このポリシーは、通常、クローラ、ジョブ、開発エンドポイントを定義するときに指定されたロールにアタッチされます。
- AWSGlueConsoleFullAccess — ポリシーがアタッチされているアイデンティティが AWS マネージメントコンソールを使用するときは、AWS Glue リソースへのフルアクセスを許可します。このポリシーで指定されたリソースの命名規則に従った場合、ユーザーは完全なコンソール機能を使用できます。このポリシーは、通常 AWS Glue コンソールのユーザーにアタッチされています。

Pipedrive の設定

AWS Glue を使用して Pipedrive からデータを転送できるようにするには、次の要件を満たす必要があります。

最小要件

- Pipedrive アカウントがある。
- Pipedrive アカウントで API アクセスが有効になっている。

これらの要件を満たすと、Pipedrive アカウントに AWS Glue を接続する準備が整います。一般的な接続では、Pipedrive で他に何もする必要はありません。

Pipedrive 接続の設定

Pipedrive は、OAuth2 の AUTHORIZATION_CODE のグラントタイプをサポートしています。

- このグラントタイプは、ユーザーを認証するためにサードパーティーの認証サーバーにユーザーをリダイレクトすることから、「three-legged」の OAuth と見なされます。AWS Glue コンソール経由で接続を作成するときに使用されます。接続を作成するユーザーは、デフォルトで AWS Glue 所有の接続アプリケーションを使用できます。この場合、Pipedrive インスタンス URL 以外の OAuth 関連情報を指定する必要はありません。AWS Glue コンソールでは、ユーザーが Pipedrive にリダイレクトされます。ここではユーザーがログインし、Pipedrive インスタンスにアクセスするためのリクエストされた権限を AWS Glue に許可する必要があります。
- ユーザーは、Pipedrive で独自の接続アプリケーションを作成し、AWS Glue コンソールを介して接続を作成するときに独自のクライアント ID とクライアントシークレットを指定することを選択する必要があります。このシナリオでは、引き続き Pipedrive にリダイレクトされてログインし、リソースへアクセスするために AWS Glue を承認します。
- このグラントタイプは、更新トークンとアクセストークンになります。アクセストークンは 1 時間有効で、更新トークンを使用することでユーザーとやり取りすることなく自動的に更新できます。
- 詳細については、AUTHORIZATION_CODE OAuth フローでの接続アプリケーションの作成に関するドキュメントを参照してください。

Pipedrive 接続を設定するには:

1. AWS Secrets Manager で、次の詳細を含むシークレットを作成します。AWS Glue で接続ごとにシークレットを作成する必要があります。
   a. カスタマーマネージド接続アプリケーションの場合 – シークレットには、USER_MANAGED_CLIENT_APPLICATION_CLIENT_SECRET を使用して接続されたアプリケーションのコンシューマーシークレットをキーとして含める必要があります。
2. AWS Glue Studio で、以下の手順に従って [データ接続] の下に接続を作成します。
   a. [データ接続] の [接続を作成] を選択します。
   b. [データソース] を選択する際に、[Pipedrive] を選択します。
   c. Pipedrive instanceURL を指定します。
   d. AWS Glue が次のアクションを担い、その権限を持つことができる IAM ロールを選択します。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:DescribeSecret",
                "secretsmanager:GetSecretValue",
                "secretsmanager:PutSecretValue",
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterface",
                "ec2:DeleteNetworkInterface",
            ],
            "Resource": "*"
        }
    ]
}
```

e. 接続する Pipedrive の [ユーザーマネージドクライアントアプリケーション ClientId] を指定します。

f. AWS Glue でこの接続に使用する `secretName` を選択して、トークンを配置します。

g. ネットワークを使用する場合は、ネットワークオプションを選択します。

3. AWS Glue ジョブに関連付けられている IAM ロールに `secretName` を読み取るアクセス許可を付与します。[Next] を選択します。
4. connectionName を指定し、[次] を選択します。
5. 次のページで、[接続を作成] を選択します。Pipedrive にログインするように求められます。ユーザー名とパスワードを入力して、[ログイン] を選択します。
6. ログインしたら、[アプリケーションに進む] を選択します。これで、接続を使用する準備ができました。
7. AWS Glue ジョブ設定で、[追加のネットワーク接続] として `connectionName` を指定します。

Pipedrive エンティティからの読み取り

前提条件

- 読み取り元の Pipedrive オブジェクト。使用可能なエンティティを確認するには、以下のサポートされているエンティティの表を参照してください。

サポートされているエンティティ

エンティティ	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
アクティビティ	あり	はい	なし	あり	あり
アクティビティタイプ	いいえ	いいえ	なし	あり	不可
通話ログ	いいえ	いいえ	なし	あり	不可
通貨	あり	はい	なし	あり	不可
Deals	あり	あり	あり	あり	あり
リード	あり	あり	あり	はい	不可
リードソース	いいえ	あり	なし	あり	不可
リードラベル	いいえ	いいえ	いいえ	いいえ	いいえ
コメント	あり	あり	あり	あり	あり
組織	あり	はい	なし	あり	あり
許可セット	あり	いいえ	なし	あり	不可
人物	あり	あり	あり	あり	あり
Pipelines	いいえ	あり	なし	あり	不可
製品	あり	はい	なし	あり	あり
ロール	いいえ	あり	なし	あり	不可
ステージ	あり	はい	なし	あり	不可
[ユーザー]	いいえ	いいえ	なし	あり	不可

例

```
pipedrive_read= glueContext.create_dynamic_frame.from_options(
    connection_type="PIPEDRIVE",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "activites",
        "API_VERSION": "v1"
    }
```

Pipedrive エンティティとフィールドの詳細

エンティティリスト:

- アクティビティ: https://developers.pipedrive.com/docs/api/v1/Activities
- アクティビティタイプ: https://developers.pipedrive.com/docs/api/v1/ActivityTypes
- 通話ログ: https://developers.pipedrive.com/docs/api/v1/CallLogs
- 通貨: https://developers.pipedrive.com/docs/api/v1/Currencies
- ディール: https://developers.pipedrive.com/docs/api/v1/Deals
- リード: https://developers.pipedrive.com/docs/api/v1/Leads
- リードソース: https://developers.pipedrive.com/docs/api/v1/LeadSources
- リードラベル: https://developers.pipedrive.com/docs/api/v1/LeadLabels
- メモ: https://developers.pipedrive.com/docs/api/v1/Notes
- 組織: https://developers.pipedrive.com/docs/api/v1/Organizations
- 許可セット: https://developers.pipedrive.com/docs/api/v1/PermissionSets
- 個人: https://developers.pipedrive.com/docs/api/v1/Persons
- パイプライン: https://developers.pipedrive.com/docs/api/v1/Pipelines
- 製品: https://developers.pipedrive.com/docs/api/v1/Products
- ロール: https://developers.pipedrive.com/docs/api/v1/Roles
- ステージ: https://developers.pipedrive.com/docs/api/v1/Stages
- ユーザー: https://developers.pipedrive.com/docs/api/v1/Users

エンティティ	データタイプ	サポートされている演算子
アクティビティ、ディール、メモ、組織、個人、製品。	日付	'='
	整数	'='
	String	'='
	ブール値	'='

パーティショニングクエリ

Pipedrive では、アクティビティエンティティの 1 つのフィールド (due_date) のみでフィールドベースのパーティショニングがサポートされています。それは日付フィールドです。

Spark で同時実行を使用する場合は、追加の Spark オプション PARTITION_FIELD、LOWER_BOUND、UPPER_BOUND、および NUM_PARTITIONS を指定できます。これらのパラメータを使用すると、元のクエリは Spark タスクで同時に実行できるサブクエリの NUM_PARTITIONS の数に分割されます。

- PARTITION_FIELD: クエリのパーティション化に使用するフィールドの名前。
- LOWER_BOUND: 選択したパーティションフィールドの包括的な下限値。

  日付については、Spark SQL クエリで使用される Spark の日付形式を受け入れます。有効な値の例: "2024-02-06"。
- UPPER_BOUND: 選択したパーティションフィールドの排他的上限値。
- NUM_PARTITIONS: パーティション数。

例

```
pipedrive_read = glueContext.create_dynamic_frame.from_options(
    connection_type="PIPEDRIVE",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "activites",
        "API_VERSION": "v1",
        "PARTITION_FIELD": "due_date"
```

```
        "LOWER_BOUND": "2023-09-07T02:03:00.000Z"
        "UPPER_BOUND": "2024-05-07T02:03:00.000Z"
        "NUM_PARTITIONS": "10"
    }
```

Pipedrive 接続オプションのリファレンス

Pipedrive の接続オプションは次のとおりです。

- ENTITY_NAME(文字列) – (必須) 読み取り / 書き込みに使用されます。Pipedrive のオブジェクトの名前。
- API_VERSION(文字列) – (必須) 読み取り / 書き込みに使用されます。使用する Pipedrive Rest API バージョン。例: v1。
- INSTANCE_URL(文字列) – (必須) ユーザーがオペレーションを実行するインスタンスの URL。例: v1。
- SELECTED_FIELDS(List<String>) – Default: empty(SELECT *). 読み込みに使用されます。オブジェクトに選択する列。
- FILTER_PREDICATE(文字列) – デフォルト: 空 読み込みに使用されます。Spark SQL 形式である必要があります。
- QUERY(文字列) - デフォルト: 空 読み込みに使用されます。完全な Spark SQL クエリ。
- PARTITION_FIELD(文字列) – 読み取りに使用されます。クエリをパーティション化するために使用するフィールド。
- LOWER_BOUND(文字列) - 読み取りに使用されます。選択したパーティションフィールドの包括的な下限値。
- UPPER_BOUND (文字列) - 読み取りに使用されます。選択したパーティションフィールドの排他的上限値。
- NUM_PARTITIONS(整数) – デフォルト: 1。読み込みに使用されます。読み取り用のパーティションの数。

制限

Pipedrive コネクタの制限は次のとおりです。

- Pipedrive は、1 つのエンティティ (アクティビティ) に対してのみフィールドベースのパーティショニングをサポートしています。

- Pipedrive は、アクティビティ、ディール、ノート、個人、組織、製品エンティティに対してレコードベースのパーティショニングをサポートしています。
- ディールエンティティで、無効な値フィルター値が使用されている場合、フィルターとしてのステータスフィールドはすべてのレコードを返します。
- ディールエンティティで、複数のフィールドを使用した順序付けはサポートされていません。
- パフォーマンスデータを取得するために、ローカル AWS アカウントを使用しています。しかし、アクセストークンをローカルで更新する制限により、1 GB のデータを処理する AWS Glue ジョブは失敗します。したがって、179 MB のデータでパフォーマンステストを最適化しており、上記の結果はこの最適化に基づいています。その場合でも、パーティションの数が増えると、SaaS エンドポイントは 1 つのパーティションと比較して時間がかかることがわかりました。この動作について Pipedrive サポートチームに相談したところ、Pipedrive はリクエストをサイレントにスロットリングし、レスポンスを遅延させているとのことでした。したがって、大規模なデータセットで AWS Glue ジョブを実行したり、同じ API エンドポイントを複数回呼び出したりすると、Pipedrive API の実装が原因でタイムアウトの問題が発生する可能性があります。ただし、コネクタとシムの応答時間は、パーティション数が増加すると期待どおりに減少します。

Productboard への接続

Productboard は、製品チームが適切な製品をより迅速に市場投入できるようにする製品管理システムです。Zendesk、UiPath、Microsoft などの 3,000 を超える最新の製品主導の企業が Productboard を使用して、ユーザーが本当に必要なものを理解し、次に何を構築するか優先順位を付け、ロードマップを中心にチーム全体を結集させます。

トピック

- AWS Glue の Productboard のサポート
- 接続を作成および使用するための API オペレーションを含むポリシー
- Productboard の設定
- Productboard 接続の設定
- Productboard エンティティからの読み取り
- Productboard 接続オプション
- Productboard アカウントの作成
- 制限

AWS Glue の Productboard のサポート

AWS Glue は、次のように Productboard をサポートします。

ソースとしてサポートされていますか?

はい。AWS Glue ETL ジョブを使用して、Productboard からデータをクエリできます。

ターゲットとしてサポートされていますか?

いいえ。

サポートされている Productboard API バージョン

v1

接続を作成および使用するための API オペレーションを含むポリシー

次のサンプル ポリシーで、接続の作成と使用に必要な AWS の権限について説明します。新しいロールを作成する場合は、以下を含むポリシーを作成します。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:ListConnectionTypes",
                "glue:DescribeConnectionType",
                "glue:RefreshOAuth2Tokens"
                "glue:ListEntities",
                "glue:DescribeEntity"
            ],
            "Resource": "*"
        }
    ]
}
```

上記の方法を使用しない場合は、代わりに次のマネージド IAM ポリシーを使用します。

- AWSGlueServiceRole – さまざまな AWS Glue プロセスを実行するために必要なリソースへのアクセス権をユーザーに代わって付与します。これらのリソースには、AWS Glue、Amazon

S3、IAM、CloudWatch Logs、および Amazon EC2 が含まれます。このポリシーで指定されたリソースの命名規則に従った場合、AWS Glue プロセスは必要なアクセス権限を使用できます。このポリシーは、通常、クローラ、ジョブ、開発エンドポイントを定義するときに指定されたロールにアタッチされます。

- AWSGlueConsoleFullAccess — ポリシーがアタッチされているアイデンティティが AWS マネージメントコンソールを使用するときは、AWS Glue リソースへのフルアクセスを許可します。このポリシーで指定されたリソースの命名規則に従った場合、ユーザーは完全なコンソール機能を使用できます。このポリシーは、通常 AWS Glue コンソールのユーザーにアタッチされています。

Productboard の設定

AWS Glue を使用して Productboard から転送できるようにするには、次の要件を満たす必要があります。

最小要件

- E メールとパスワードが設定された Productboard アカウントがある。アカウントの作成の詳細については、「Productboard アカウントの作成」を参照してください。
- AWS Glue へのサービスアクセスで AWS アカウントを作成する必要があります。
- Productboard アカウントの認証の詳細 (カスタム認証を使用する場合は JWT トークン、OAuth2.0 を使用する場合はクライアント ID とシークレット) がある。
- ユーザーが `OAuth2.0` を使用する場合は、アプリケーションを Productboard に登録し、「How to integrate with Productboard via OAuth2 - developer documentation」の手順に従ってアプリケーションをセットアップします。

これらの要件を満たすと、Productboard アカウントに AWS Glue を接続する準備が整います。一般的な接続では、Productboard で他に何もする必要はありません。

Productboard 接続の設定

Productboard は、カスタム認可と `OAuth2.0` をサポートしています。`OAuth2.0` の場合、Productboard は、`AUTHORIZATION_CODE` グラントタイプをサポートしています。

- このグラントタイプは、ユーザーを認可するためにサードパーティーの認可サーバーにユーザーをリダイレクトすることから、「three-legged」の `OAuth` と見なされます。AWS Glue コンソール経由で接続を作成するときに使用されます。接続を作成するユーザーは、デフォルトで AWS

Glue 所有の接続アプリケーションを使用できます。この場合、Productboard クライアント ID とクライアントシークレット以外の OAuth 関連情報を指定する必要はありません。AWS Glue コンソールでは、ユーザーが Productboard にリダイレクトされます。ここではユーザーがログインし、Productboard インスタンスにアクセスするためのリクエストされた権限を AWS Glue に許可する必要があります。

- ユーザーは、Productboard で独自の接続アプリケーションを作成し、AWS Glue コンソールを介して接続を作成するときに独自のクライアント ID とクライアントシークレットを指定することを選択できます。このシナリオでは、引き続き Productboard にリダイレクトされてログインし、リソースへアクセスするために AWS Glue を認可します。
- このグラントタイプは、更新トークンとアクセストークンになります。アクセストークンの有効期間は短く、更新トークンを使用してユーザーとやり取りすることなく自動的に更新される場合があります。
- AUTHORIZATION_CODE OAuth フロー用の接続アプリケーションの作成に関する Productboard の公開ドキュメントについては、「How to integrate with Productboard via OAuth2 - developer documentation」を参照してください。

Productboard 接続を設定するには:

1. AWS Secrets Manager で、次の詳細を含むシークレットを作成します。
   - OAuth 認可 – カスタマーマネージド接続アプリケーションの場合: シークレットには、USER_MANAGED_CLIENT_APPLICATION_CLIENT_SECRET を使用して接続されたアプリケーションのコンシューマーシークレットをキーとして含める必要があります。
   - Custom auth – カスタマーマネージド接続アプリケーションの場合: シークレットには、access_token を使用して接続されたアプリケーションの JWT token をキーとして含める必要があります。

   > **Note**
   > AWS Glue で接続ごとにシークレットを作成する必要があります。

2. AWS Glue Studio で、以下の手順に従って [データ接続] の下に接続を作成します。
   a. [データソース] を選択する際に、[Productboard] を選択します。
   b. AWS Glue が次のアクションを担い、その権限を持つことができる IAM ロールを選択します。

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

```
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:DescribeSecret",
                "secretsmanager:GetSecretValue",
                "secretsmanager:PutSecretValue",
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterface",
                "ec2:DeleteNetworkInterface",
            ],
            "Resource": "*"
        }
    ]
}
```

c. データソースに接続するための認可タイプを選択します。

- OAuth 認可の場合 – Productboard アプリケーションの `Token URL`、および `User Managed Client Application ClientId` を指定します。

d. AWS Glue でこの接続に使用する `secretName` を選択して、トークンを配置します。

e. ネットワークを使用する場合は、ネットワークオプションを選択します。

3. AWS Glue ジョブに関連付けられている IAM ロールに `secretName` を読み取るアクセス許可を付与します。
4. AWS Glue ジョブ設定で、[追加のネットワーク接続] として `connectionName` を指定します。

Productboard エンティティからの読み取り

前提条件

読み取り元の Productboard オブジェクト。使用可能なエンティティを確認するには、以下のサポートされているエンティティの表を参照してください。

サポートされているエンティティ

- Abuse-reports
- オートメーション
- キャンペーン
- Click-details

- Lists
- メンバー
- Open-details
- セグメント
- Stores
- Unsubscribed

エンティティ	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
機能	あり	はい	なし	あり	あり
コンポーネント	いいえ	あり	なし	あり	不可
製品	いいえ	あり	なし	あり	不可
フィーチャーステータス	いいえ	あり	なし	あり	あり
カスタムフィールド定義	いいえ	あり	なし	あり	不可
カスタムフィールド値	あり	はい	なし	あり	不可

例

```
Productboard_read = glueContext.create_dynamic_frame.from_options(
    connection_type="Productboard",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "feature",
        "API_VERSION": "1"
```

```
    }
```

Productboard エンティティとフィールドの詳細

- 特徴
- コンポーネント
- フィーチャーステータス
- 製品
- カスタムフィールド定義
- カスタムフィールド値

Productboard 接続オプション

Productboard の接続オプションは次のとおりです。

- `ENTITY_NAME`(文字列) – (必須) 読み取り / 書き込みに使用されます。Productboard のオブジェクトの名前。
- `API_VERSION`(文字列) – (必須) 読み取りに使用されます。使用する Productboard Engage Rest API バージョン。例: 3.0。
- `SELECTED_FIELDS`(List<String>) – Default: empty(SELECT *). 読み込みに使用されます。オブジェクトに選択する列。
- `FILTER_PREDICATE`(文字列) – デフォルト: 空 読み込みに使用されます。Spark SQL 形式である必要があります。
- `QUERY`(文字列) - デフォルト: 空 読み込みに使用されます。完全な Spark SQL クエリ。

Productboard アカウントの作成

1. Productboard のサインアップページに移動し、E メール ID とパスワードを入力して、[ログイン] を選択します。
2. [アカウント名] フィールドに Productboard アカウントの名前を入力し、[プライバシーポリシーに同意する] チェックボックスをオンにします。
3. [ワークスペースを作成する] ページの [ワークスペース URL] フィールドに、新しいワークスペースの URL を入力します。次に、[続行] を選択して次のページに進み、残りの詳細を入力します。

これでトライアルアカウントが作成されます。トライアルアカウントは、15 日間無料です。トライアル期間の終了後は、有料プランを購入できます。E メールアドレス、パスワード、ワークスペース URL を書き留めます。この情報は、今後アカウントにアクセスするために必要になります。

OAuth2.0 アプリケーションの登録

1. Productboard のログインページに移動し、E メール ID とパスワードを入力して、[ログイン] を選択します。
2. 右上隅の [User] アイコンを選択し、ドロップダウンメニューから [Account and billing] を選択します。
3. [Extras] を選択し、ドロップダウンメニューから [Registered apps] を選択します。
4. [Register An App] を選択します。
5. 次の詳細情報を入力します。
   - App name – アプリケーションの名前。
   - Company / Organization – 会社または組織の名前。
   - App website – アプリケーションのウェブサイト。
   - [リダイレクト URI] – リダイレクト URI パターンは、ログインフローが完了したときに Productboard がリダイレクト (リクエストされた場合) できる URI パス (またはカンマ区切りのパスのリスト) です。例えば、`https://ap-southeast-2\\.console\\.aws\\.amazon\\.com`
6. [Create] (作成) を選択します。
7. これで、[Client ID] と [Client Secret] が表示されます。これらをコピーして安全な場所に保存します。次に [完了] を選択します。

> **Note**
>
> クライアント ID とクライアントシークレット文字列は、AppFlow または AWS Glue を使用するときにこのコネクタとの接続を確立するために使用される認証情報です。

CustomAuth 認証情報の取得

1. Productboard のログインページに移動し、E メール ID とパスワードを入力して、[ログイン] を選択します。

   ホームページにリダイレクトされます。

2. ホームページで、[ワークスペース設定] > [統合] > [パブリック API] > [アクセストークン] に移動します。

   > **Note**
   >
   > [パブリック API] セクションが表示されない場合は、アカウントが Essentials プランになっている可能性があります。API トークンにアクセスするには、少なくとも Pro プランである必要があります。プランの機能と名前は変更される可能性があります。パッケージの詳細については、Productboard の料金を参照してください。

3. [+] を選択して新しいトークンを生成し、後で参照できるように安全に保存してください。

OAuth2.0 認証情報の作成

Productboard コネクタで OAuth2.0 認可を利用するには、アプリケーションを Productboard プラットフォームに登録し、Client ID と Client Secret を生成する必要があります。

1. Productboard のログインページに移動し、E メール ID とパスワードを入力して、[ログイン] を選択します。
2. 新しい OAuth2 アプリケーションを Productboard アカウントに登録するには、Producboard ページに移動します。
3. 必須フィールドを入力し、アクセスする各エンティティに必要なスコープを選択します。

   > **Note**
   >
   > サポートされている 6 つのエンティティに必要な次の 4 つのスコープを選択しています。

4. [リダイレクト URL] は次の形式である必要があります: `https://ap-southeast-2\\.console\\.aws\\.amazon\\.com`

> **Note**
>
> Appflow リダイレクト URL は変更される可能性があります。利用可能になったら、AWS Glue プラットフォームのリダイレクト URL を更新してください。

5. これで、[Client ID] と [Client Secret] が表示されます。これらをコピーして安全な場所に保存します。
6. 「How to Integrate with Productboard via OAuth2 developer」の手順に従って、OAuth2 を設定して検証できます。

制限

Productboard コネクタの制限は次のとおりです。

- Productboard は、フィールドベースまたはレコードベースのパーティショニングをサポートしていません。

QuickBooks に対する接続

QuickBooks は、中小企業向けの主要な会計アプリケーションです。QuickBooks の会計アプリケーションは、Intuit の最初の製品の 1 つとして 1980 年代にまでさかのぼり、それゆえにもともとデスクトップソフトウェアでした。現在、QuickBooks は、インストール可能なソフトウェアとクラウドベースの SaaS ソフトウェアの両方として、いくつかの会計およびビジネス財務アプリケーションを提供しています。QuickBooks ユーザーのお客様は、AWS Glue を QuickBooks アカウントに接続できます。その後、QuickBooks を ETL ジョブのデータソースとして使用できます。これらのジョブを実行して、QuickBooks と AWS サービス、またはその他のサポートされているアプリケーション間でデータを転送します。

トピック

- QuickBooks 向けの AWS Glue サポート
- 接続を作成および使用するための API オペレーションを含むポリシー
- QuickBooks の設定
- QuickBooks 接続の設定
- QuickBooks エンティティからの読み取り
- QuickBooks 接続オプション

- QuickBooks コネクタの制限事項と注意事項

QuickBooks 向けの AWS Glue サポート

AWS Glue は、次のように QuickBooks をサポートします。

ソースとしてサポートされていますか?

はい。AWS Glue ETL ジョブを使用して、QuickBooks からデータをクエリできます。

ターゲットとしてサポートされていますか?

いいえ。

サポートされている QuickBooks API バージョン

次の QuickBooks API バージョンがサポートされています。

- v3

接続を作成および使用するための API オペレーションを含むポリシー

次のサンプル ポリシーで、接続の作成と使用に必要な AWS IAM 権限について説明します。新しいロールを作成する場合は、以下を含むポリシーを作成します。

```
{
 "Version": "2012-10-17",
     "Statement": [
      {
             "Effect": "Allow",
             "Action": [
                 "glue:ListConnectionTypes",
                 "glue:DescribeConnectionType",
                 "glue:RefreshOAuth2Tokens",
                 "glue:ListEntities",
                 "glue:DescribeEntity"
             ],
             "Resource": "*"
         }
     ]
 }
```

上記の方法を使用しない場合は、代わりに次のマネージド IAM ポリシーを使用します:

- AWSGlueServiceRole – さまざまな AWS Glue プロセスを実行するために必要なリソースへのアクセス権をユーザーに代わって付与します。これらのリソースには AWS Glue 、Amazon S3、IAM、CloudWatch Logs、および Amazon EC2 が含まれます。このポリシーで指定されたリソースの命名規則に従った場合、AWS Glue プロセスは必要なアクセス権限を使用できます。このポリシーは、通常、クローラ、ジョブ、開発エンドポイントを定義するときに指定されたロールにアタッチされます。
- AWSGlueConsoleFullAccess — ポリシーがアタッチされているアイデンティティが AWS マネージメントコンソールを使用するときは、AWS Glue リソースへのフルアクセスを許可します。このポリシーで指定されたリソースの命名規則に従った場合、ユーザーは完全なコンソール機能を使用できます。このポリシーは、通常 AWS Glue コンソールのユーザーにアタッチされています。

QuickBooks の設定

AWS Glue を使用して QuickBooks からデータを転送できるようにするには、次の要件を満たす必要があります。

最小要件

以下に、最小要件を示します。

- QuickBooks アカウントがある。
- QuickBooks アカウントで API アクセスが有効になっている。

詳細については、QuickBooks ドキュメントの次のトピックを参照してください。

- Intuit アカウントを作成する
- アプリを作成して開発を開始する

これらの要件を満たすと、QuickBooks アカウントに AWS Glue を接続する準備が整います。一般的な接続では、QuickBooks で他に何もする必要はありません。

QuickBooks 接続の設定

QuickBooks は、OAuth2 の AUTHORIZATION_CODE のグラントタイプをサポートしています。グラントタイプは、AWS Glue が QuickBooks と通信してデータへのアクセスをリクエストする方法を決定します。

- このグラントタイプは、ユーザーを認証するためにサードパーティーの認証サーバーにユーザーをリダイレクトすることから、「three-legged」の OAuth と見なされます。AWS Glue コンソール経由で接続を作成するときに使用されます。
- ユーザーは、QuickBooks で独自の接続アプリを作成し、AWS Glue コンソールを介して接続を作成するときに独自のクライアント ID とクライアントシークレットを指定することを選択できます。このシナリオでは、引き続き QuickBooks にリダイレクトされてログインし、リソースへアクセスするために AWS Glue を承認します。
- このグラントタイプは、更新トークンとアクセストークンになります。アクセストークンの有効期間は短く、更新トークンを使用してユーザーとやり取りすることなく自動的に更新される場合があります。
- 認可コード OAuth フロー用の接続アプリケーションの作成に関する QuickBooks の公開ドキュメントについては、「OAuth 2.0 を設定する」を参照してください。

QuickBooks 接続を設定するには：

1. AWS Glue Glue Studio で、以下の手順に従って [データ接続] の下に接続を作成します。
   a. [接続タイプ] を選択する際に、[QuickBooks] を選択します。
   b. 接続する QuickBooks インスタンスのインスタンス URL と会社 ID を指定します。
   c. 次のアクションを実行でき、AWS Glue がその権限を持つ AWS IAM ロールを選択します。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:DescribeSecret",
                "secretsmanager:GetSecretValue",
                "secretsmanager:PutSecretValue",
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterface",
                "ec2:DeleteNetworkInterface",
            ],
            "Resource": "*"
        }
    ]
}
```

d. AWS Glue でこの接続に使用する `secretName` を選択して、トークンを配置します。

e. ネットワークを使用する場合は、ネットワークオプションを選択します。

2. AWS Glue ジョブに関連付けられている IAM ロールに `secretName` を読み取るアクセス許可を付与します。

QuickBooks エンティティからの読み取り

前提条件

読み取り元の QuickBooks オブジェクト。

ソースに対応するエンティティ:

エンティティ	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
アカウント	はい	はい	はい	はい	はい
請求	はい	はい	はい	はい	はい
会社情報	いいえ	いいえ	いいえ	はい	いいえ
お客様	はい	はい	はい	はい	はい
従業員	はい	はい	はい	はい	はい
見積り	はい	はい	はい	はい	はい
Invoice	はい	はい	はい	はい	はい
項目	はい	はい	はい	はい	はい
支払い	はい	はい	はい	はい	はい
詳細設定	いいえ	いいえ	いいえ	はい	いいえ
利益と損失	はい	いいえ	いいえ	はい	いいえ
税務機関	はい	はい	はい	はい	はい

エンティティ	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
ベンダー	はい	はい	はい	はい	はい

例:

```
QuickBooks_read = glueContext.create_dynamic_frame.from_options(
    connection_type="quickbooks",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "Account",
        "API_VERSION": "v3"
    }
```

QuickBooks エンティティとフィールドの詳細:

エンティティとフィールドの詳細については、以下を参照してください:

- アカウント
- 請求
- CompanyInfo
- お客様
- 従業員
- Estimate
- Invoice
- 項目
- Payment
- 詳細設定
- ProfitAndLoss
- TaxAgency
- ベンダー

パーティショニングクエリ

フィールドベースのパーティション分割:

QuickBooks では、整数および DateTime データ型フィールドは、フィールドベースのパーティション分割をサポートします。

Spark で同時実行を使用する場合は、追加の Spark オプション PARTITION_FIELD、LOWER_BOUND、UPPER_BOUND、および NUM_PARTITIONS を指定できます。これらのパラメータを使用すると、元のクエリは Spark タスクで同時に実行できるサブクエリの NUM_PARTITIONS の数に分割されます。

- PARTITION_FIELD: クエリのパーティション化に使用するフィールドの名前。
- LOWER_BOUND: 選択したパーティションフィールドの包括的な下限値。

  Datetime フィールドでは、Spark SQL クエリで使用される Spark タイムスタンプ形式を受け入れます。

  有効な値の例は次のとおりです:

  ```
  "2024-05-07T02:03:00.00Z"
  ```
- UPPER_BOUND: 選択したパーティションフィールドの排他的上限値。
- NUM_PARTITIONS: パーティション数。

例:

```
QuickBooks_read = glueContext.create_dynamic_frame.from_options(
    connection_type="quickbooks",
    connection_options={
        "connectionName": "connectionName",
        "REALMID": "12345678690123456789",
        "ENTITY_NAME": "Account",
        "API_VERSION": "v3",
        "PARTITION_FIELD": "MetaData_CreateTime"
        "LOWER_BOUND": "2023-09-07T02:03:00.000Z"
        "UPPER_BOUND": "2024-05-07T02:03:00.000Z"
        "NUM_PARTITIONS": "10"
    }
```

レコードベースのパーティション分割:

元のクエリは、元のクエリは Spark タスクで同時に実行できるサブクエリの NUM_PARTITIONS の数に分割されます。

- NUM_PARTITIONS: パーティション数。

例:

```
QuickBooks_read = glueContext.create_dynamic_frame.from_options(
    connection_type="quickbooks",
    connection_options={
        "connectionName": "connectionName",
        "REALMID": "1234567890123456789",
        "ENTITY_NAME": "Bill",
        "API_VERSION": "v3",
        "NUM_PARTITIONS": "10"
    }
```

QuickBooks 接続オプション

QuickBooks の接続オプションは次のとおりです。

- ENTITY_NAME(文字列) – (必須) 読み取りに使用されます。QuickBooks のオブジェクトの名前。
- INSTANCE_URL(文字列) – (必須) 有効な QuickBooks インスタンス URL。
- API_VERSION(文字列) – (必須) 読み取りに使用されます。使用する QuickBooks Rest API バージョン。
- REALM_ID(文字列) – リクエストを送信する個々の QuickBooks Online 会社を識別する ID。
- SELECTED_FIELDS(List<String>) – Default: empty(SELECT *). 読み込みに使用されます。オブジェクトに選択する列。
- FILTER_PREDICATE(文字列) – デフォルト: 空 読み込みに使用されます。Spark SQL 形式である必要があります。
- QUERY(文字列) – デフォルト: 空 読み込みに使用されます。完全な Spark SQL クエリ。
- PARTITION_FIELD(文字列) – 読み取りに使用されます。クエリをパーティション化するために使用するフィールド。
- LOWER_BOUND(文字列) - 読み取りに使用されます。選択したパーティションフィールドの包括的な下限値。
- UPPER_BOUND(文字列) – 読み取りに使用されます。選択したパーティションフィールドの排他的上限値。

- NUM_PARTITIONS(整数) – デフォルト: 1。読み込みに使用されます。読み取り用のパーティションの数。

QuickBooks コネクタの制限事項と注意事項

QuickBooks コネクタの制限事項または注意事項は次のとおりです。

- taxAgency API では、フィルタリングによる順序が期待どおりに機能しません。

Salesforce への接続

Salesforce は、販売、カスタマーサービス、e コマースなどに役立つカスタマーリレーションシップ管理 (CRM) ソフトウェアを提供します。Salesforce ユーザーの場合は、AWS Glue を Salesforce アカウント に接続できます。次に、ETL ジョブのデータソースまたは送信先として Salesforce を使用できます。これらのジョブを実行して、Salesforce と AWS サービス、またはその他のサポートされているアプリケーション間でデータを転送します。

トピック

- Salesforce 向けの AWS Glue サポート
- 接続を作成および使用するための API オペレーションを含むポリシー
- Salesforce の設定
- システム管理者プロファイルを適用する
- Salesforce 接続の設定
- Salesforce からの読み取り
- Salesforce への書き込み
- Salesforce 接続オプション
- Salesforce コネクタの制限事項
- Salesforce の認証コードフローの設定
- Salesforce の JWT ベアラ OAuth フローを設定する

Salesforce 向けの AWS Glue サポート

AWS Glue は次のように Salesforce をサポートします。

ソースとしてサポートされていますか?

はい。AWS Glue ETL ジョブを使用して、Salesforce からデータをクエリできます。

ターゲットとしてサポートされているかどうか

はい。AWS Glue ETL ジョブを使用して Salesforce にレコードを書き込むことができます。

サポートされている Salesforce API バージョン

次の Salesforce API バージョンがサポートされています。

- v58.0
- v59.0
- v60.0

接続を作成および使用するための API オペレーションを含むポリシー

次のサンプル IAM ポリシーで、AWS Glue ETL ジョブでの Salesforce 接続の作成、管理、使用に必要なアクセス許可について説明します。新しいロールを作成する場合は、以下を含むポリシーを作成します。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:DescribeSecret",
                "secretsmanager:GetSecretValue",
                "secretsmanager:PutSecretValue",
                "glue:ListConnectionTypes",
                "glue:DescribeConnectionType",
                "glue:RefreshOAuth2Tokens",
                "glue:ListEntities",
                "glue:DescribeEntity"
            ],
            "Resource": "*"
        }
    ]
}
```

また、次の IAM ポリシーを使用して、アクセスを許可できます。

- AWSGlueServiceRole – さまざまな AWS Glue プロセスを実行するために必要なリソースへのアクセス権をユーザーに代わって付与します。これらのリソースには AWS Glue 、Amazon S3、IAM、CloudWatch Logs、および Amazon EC2 が含まれます。このポリシーで指定されたリソースの命名規則に従った場合、AWS Glue プロセスは必要なアクセス権限を使用できます。このポリシーは、通常、クローラ、ジョブ、開発エンドポイントを定義するときに指定されたロールにアタッチされます。
- AWSGlueConsoleFullAccess — ポリシーがアタッチされているアイデンティティが AWS マネージメントコンソールを使用するときは、AWS Glue リソースへのフルアクセスを許可します。このポリシーで指定されたリソースの命名規則に従った場合、ユーザーは完全なコンソール機能を使用できます。このポリシーは、通常 AWS Glue コンソールのユーザーにアタッチされています。

Salesforce 接続の作成時にネットワークオプションを指定する場合は、IAM ロールに次のアクションも含める必要があります。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterface",
                "ec2:DeleteNetworkInterface",
            ],
            "Resource": "*"
        }
    ]
}
```

ゼロ ETL Salesforce 接続については、「ゼロ ETL の前提条件」を参照してください。

ゼロ ETL Salesforce 接続については、「ゼロ ETL の前提条件」を参照してください。

Salesforce の設定

AWS Glue を使用して Salesforce との間でデータを転送できるようにするには、次の要件を満たす必要があります。

最小要件

以下に、最小要件を示します。

- Salesforce アカウントがある。
- Salesforce アカウントで API アクセスが有効になっている。Enterprise、Unlimited、Developer、Performance の各工ディションで API アクセスがデフォルトで有効になっている。

これらの要件を満たすと、Salesforce アカウントに AWS Glue を接続する準備が整います。AWS Glue は、AWS マネージド接続アプリで残りの要件を処理します。

Salesforce 用の AWS マネージド接続アプリケーション

AWS マネージド接続アプリケーションは、より少ないステップで Salesforce 接続を作成するのに役立ちます。Salesforce では、接続アプリケーションは、AWS Glue などの外部アプリケーションが OAuth 2.0 を使用して Salesforce データにアクセスすることを許可するフレームワークです。AWS マネージド接続アプリケーションを使用するには、AWS Glue コンソールを使用して Salesforce 接続を作成します。接続を設定するときは、[OAuth グラントタイプ] を[認証コード] に設定し、[AWS マネージド接続アプリケーションの使用] のチェックボックスをオンのままにします。

接続を保存すると、Salesforce にリダイレクトされるので、Salesforce アカウントにログインして AWS Glue アクセスを承認します。

システム管理者プロファイルを適用する

Salesforce で、手順に従ってシステム管理者プロファイルを適用します。

1. Salesforce で、[設定] > [接続されたアプリ] > [接続されたアプリの OAuth 使用状況] に移動します。
2. 接続されたアプリのリストで、AWS Glue を見つけ [インストール] を選択します。必要に応じて、[ブロック解除] を選択します。
3. [設定] > [接続されたアプリを管理} に移動し、AWS Glue を選択します。OAuth ポリシーで、[管理者承認済みユーザーが事前承認されていること] を選択し、[システム管理] のプロファイルを選択します。このアクションにより、AWS Glue へのアクセスがシステム管理者プロファイルを持つユーザーのみに制限されます。

システム管理者プロファイルを適用する

Salesforce で、手順に従ってシステム管理者プロファイルを適用します。

1. Salesforce で、[設定] > [接続されたアプリ] > [接続されたアプリの OAuth 使用状況] に移動します。

2. 接続されたアプリのリストで、AWS Glue を見つけ [インストール] を選択します。必要に応じて、[ブロック解除] を選択します。
3. [設定] > [接続されたアプリを管理} に移動し、AWS Glue を選択します。OAuth ポリシーで、[管理者承認済みユーザーが事前承認されていること] を選択し、[システム管理] のプロファイルを選択します。このアクションにより、AWS Glue へのアクセスがシステム管理者プロファイルを持つユーザーのみに制限されます。

Salesforce 接続の設定

Salesforce 接続を設定するには：

1. AWS Secrets Manager で、次の詳細を含むシークレットを作成します。
   a. JWT_TOKEN グラントタイプの場合 - シークレットには、JWT_TOKEN キーとその値が含まれている必要があります。
   b. AuthorizationCode グラントタイプの場合:
      i. AWS マネージド接続アプリケーションの場合、空のシークレット、または一時的な値を持つシークレットを指定する必要があります。
      ii. カスタマーマネージド接続アプリケーションの場合、シークレットには、USER_MANAGED_CLIENT_APPLICATION_CLIENT_SECRET をキーとした接続アプリ Consumer Secret を含める必要があります。
   c. 注: AWS Glue で接続ごとにシークレットを作成する必要があります。
2. AWS Glue Glue Studio で、以下の手順に従って [データ接続] の下に接続を作成します。
   a. [接続タイプ] を選択する際に、[Salesforce] を選択します。
   b. 接続する Salesforce インスタンスの INSTANCE_URL を指定します。
   c. Salesforce 環境を指定します。
   d. 次のアクションを実行でき、AWS Glue がその権限を持つ AWS IAM ロールを選択します。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:DescribeSecret",
                "secretsmanager:GetSecretValue",
                "secretsmanager:PutSecretValue",
```

```
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterface",
                "ec2:DeleteNetworkInterface",
            ],
            "Resource": "*"
        }
    ]
}
```

e. 接続に使用する OAuth2 グラントタイプを選択します。グラントタイプは、AWS Glue が Salesforce と通信してデータへのアクセスをリクエストする方法を決定します。選択すると、接続を作成する前に満たす必要がある要件に影響します。次のいずれかのタイプを選択できます。

- JWT_BEARER グラントタイプ: この付与タイプは、Salesforce インスタンス内の特定のユーザーのアクセス許可を使用して JSON Web トークン (JWT) を事前に作成できるため、オートメーションシナリオに適しています。作成者は、JWT の有効期間を制御できます。AWS Glue は JWT を使って、Salesforce API の呼び出しに使用されるアクセストークンを取得できます。

  このフローでは、ユーザーが Salesforce インスタンスで接続されたアプリケーションを作成し、ユーザーに JWT ベースのアクセストークンを発行できるようにする必要があります。

  JWT ベアラ OAuth フロー用の接続されたアプリケーションの作成については、「OAuth 2.0 JWT bearer flow for server-to-server integration」を参照してください。Salesforce 接続アプリケーションで JWT ベアラフローを設定するには、「Salesforce の JWT ベアラ OAuth フローを設定する」を参照してください。

- AUTHORIZATION_CODE グラントタイプ: この付与タイプは、ユーザーを認証するためにサードパーティーの認証サーバーにユーザーをリダイレクトすることに依存するため、「three-legged」の OAuth と見なされます。AWS Glue コンソール経由で接続を作成するときに使用されます。接続を作成するユーザーは、デフォルトで AWS Glue 接続されたアプリケーション (AWS Glue マネージドクライアントアプリケーション) に依存する場合があります。この場合、Salesforce インスタンス URL 以外の OAuth 関連情報を提供する必要はありません。AWS Glue コンソールでは、ユーザーが Salesforce にリダイレクトされます。ここではユーザーがログインし、Salesforce インスタンスにアクセスするためのリクエストされた権限を AWS Glue に許可する必要があります。

  ユーザーは、Salesforce で独自の接続アプリケーションを作成し、AWS Glue コンソールを介して接続を作成するときに独自のクライアント ID とクライアントシークレットを提供する

ことを選択できます。このシナリオでは、引き続き Salesforce にリダイレクトされてログインし、リソースへアクセスするために AWS Glue を承認します。

このグラントタイプは、更新トークンとアクセストークンになります。アクセストークンの有効期間は短く、更新トークンを使用してユーザーとやり取りすることなく自動的に更新される場合があります。

OAuth 認証コードフローでの接続アプリケーションの作成の詳細については、「the section called “Salesforce の認証コードフローの設定”」を参照してください。

f. AWS Glue でこの接続に使用する `secretName` を選択して、OAuth 2.0 トークンを配置します。

g. ネットワークを使用する場合は、ネットワークオプションを選択します。

3. AWS Glue ジョブに関連付けられている IAM ロールに `secretName` を読み取るアクセス許可を付与します。

4. ネットワークオプションを提供する場合は、IAM ロールに次のアクセス許可も付与します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterface",
                "ec2:DeleteNetworkInterface",
            ],
            "Resource": "*"
        }
    ]
}
```

AWS CLI による Salesforce 接続の設定

CLI を使用して Salesforce AWS 接続を作成できます。

```
aws glue create-connection --connection-input \
"{\"Name\": \"salesforce-conn1\",\"ConnectionType\": \"SALESFORCE\",
\"ConnectionProperties\": {\"ROLE_ARN\": \"arn:aws:iam::123456789012:role/glue-role\",
\"INSTANCE_URL\": \"https://example.my.salesforce.com\"},\"ValidateCredentials\": true,
```

```
\"AuthenticationConfiguration\": {\"AuthenticationType\": \"OAUTH2\",\"SecretArn\":
 \"arn:aws:secretsmanager:us-east-1:123456789012:secret:salesforce-conn1-secret-IAmcdk
\",\"OAuth2Properties\": {\"OAuth2GrantType\": \"JWT_BEARER\",\"TokenUrl\": \"https://
login.salesforce.com/services/oauth2/token\"}}}" \
--endpoint-url https://glue.us-east-1.amazonaws.com \
--region us-east-1
```

Salesforce からの読み取り

前提条件

読み取る Salesforce sObject。`Account`、`Case`、または `Opportunity` などのオブジェクト名が必要です。

例:

```
salesforce_read = glueContext.create_dynamic_frame.from_options(
    connection_type="salesforce",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "Account",
        "API_VERSION": "v60.0"
    }
```

パーティショニングクエリ

Spark で同時実行を使用する場合は、追加の Spark オプション `PARTITION_FIELD`、`LOWER_BOUND`、`UPPER_BOUND`、および `NUM_PARTITIONS` を指定できます。これらのパラメータを使用すると、元のクエリは Spark タスクで同時に実行できるサブクエリの `NUM_PARTITIONS` の数に分割されます。

- `PARTITION_FIELD`: クエリのパーティション化に使用するフィールドの名前。
- `LOWER_BOUND`: 選択したパーティションフィールドの包括的な下限値。

  日付またはタイムスタンプフィールドについて、コネクタは Spark SQL クエリで使用される Spark タイムスタンプ形式を受け入れます。

  有効な値の例は次のとおりです。

  ```
  "TIMESTAMP \"1707256978123\""
  ```

```
"TIMESTAMP '2018-01-01 00:00:00.000 UTC'"
"TIMESTAMP \"2018-01-01 00:00:00 Pacific/Tahiti\""
"TIMESTAMP \"2018-01-01 00:00:00\""
"TIMESTAMP \"-123456789\" Pacific/Tahiti"
"TIMESTAMP \"1702600882\""
```

- UPPER_BOUND: 選択したパーティションフィールドの排他的上限値。
- NUM_PARTITIONS: パーティション数。
- TRANSFER_MODE: SYNC と ASYNC の 2 つのモードをサポートしています。デフォルトは SYNC です。ASYNC に設定すると、Bulk API 2.0 クエリが処理に使用されます。

例:

```
salesforce_read = glueContext.create_dynamic_frame.from_options(
    connection_type="salesforce",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "Account",
        "API_VERSION": "v60.0",
        "PARTITION_FIELD": "SystemModstamp",
        "LOWER_BOUND": "TIMESTAMP '2021-01-01 00:00:00 Pacific/Tahiti'",
        "UPPER_BOUND": "TIMESTAMP '2023-01-10 00:00:00 Pacific/Tahiti'",
        "NUM_PARTITIONS": "10",
        "TRANSFER_MODE": "ASYNC"
    }
```

Salesforce への書き込み

前提条件

書き込み先の Salesforce sObject。Account、Case、または Opportunity などのオブジェクト名が必要です。

Salesforce コネクタは、次の 4 つの書き込みオペレーションをサポートしています。

- INSERT
- UPSERT
- UPDATE
- DELETE

UPSERT 書き込みオペレーションを使用する場合は、ID_FIELD_NAMES オプションを指定してレコードの外部 ID フィールドを指定する必要があります。

次に示す接続オプションを追加することもできます。

- TRANSFER_MODE: SYNC と ASYNC の 2 つのモードをサポートしています。デフォルトは SYNC です。ASYNC に設定すると、一Bulk API 2.0 の取り込みが処理に使用されます。
- FAIL_ON_FIRST_ERROR: デフォルト値は FALSE です。つまり、レコードの書き込みに何度か失敗した場合でも、AWS Glue ジョブはすべてのデータの処理を続行します。TRUE に設定すると、レコードの書き込みに何度か失敗した場合、AWS Glue ジョブは失敗し、処理は続行されません。

例

```
salesforce_write = glueContext.write_dynamic_frame.from_options(
    frame=frameToWrite,
    connection_type="salesforce",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "Account",
        "API_VERSION": "v60.0",
        "WRITE_OPERATION": "INSERT",
        "TRANSFER_MODE": "ASYNC",
        "FAIL_ON_FIRST_ERROR":""true"
    }
```

Salesforce 接続オプション

Salesforce コネクタでサポートされている接続オプションは次のとおりです。

- ENTITY_NAME (文字列) - 読み取り / 書き込みに使用されます。Salesforce のオブジェクトの名前。
- API_VERSION(文字列) – (必須) 読み取り / 書き込みに使用されます。使用する Salesforce Rest API バージョン。
- SELECTED_FIELDS(List<String>) - Default: empty(SELECT *). 読み込みに使用されます。オブジェクトに選択する列。
- FILTER_PREDICATE(文字列) – デフォルト: 空 読み込みに使用されます。Spark SQL 形式である必要があります。

フィルター述語を指定する場合、 AND 演算子のみがサポートされます。OR や IN などの他の演算子は現在サポートされていません。

- QUERY(文字列) – デフォルト: 空 読み込みに使用されます。完全な Spark SQL クエリ。
- PARTITION_FIELD(文字列) – 読み取りに使用されます。クエリをパーティション化するために使用するフィールド。
- LOWER_BOUND(文字列) - 読み取りに使用されます。選択したパーティションフィールドの包括的な下限値。
- UPPER_BOUND (文字列) - 読み取りに使用されます。選択したパーティションフィールドの排他的上限値。
- NUM_PARTITIONS(整数) – デフォルト: 1。読み込みに使用されます。読み取り用のパーティションの数。
- IMPORT_DELETED_RECORDS (文字列) - デフォルト: FALSE。読み込みに使用されます。クエリ中に削除レコードを取得するには。
- WRITE_OPERATION(文字列) - デフォルト: INSERT。書き込みに使用されます。値は INSERT、UPDATE、UPSERT、DELETE である必要があります。
- ID_FIELD_NAMES(文字列) - デフォルト: null。UPDATE および UPSERT に必要です。

Salesforce コネクタの制限事項

Salesforce コネクタの制限は次のとおりです。

- サポートされるのは Spark SQL のみで、Salesforce SOQL はサポートされていません。
- ジョブのブックマークはサポートされません。
- Salesforce フィールド名では大文字と小文字が区別されます。Salesforce に書き込む場合、データは Salesforce 内で定義されたフィールドの大文字と小文字を一致させる必要があります。

Salesforce の認証コードフローの設定

OAuth 2.0 認証コードフローを有効にする方法については、Salesforce の公開ドキュメントを参照してください。

接続アプリケーションを設定するには:

1. [OAuth 設定の有効化] チェックボックスをオンにします。

2. [コールバック URL] テキストフィールドに、AWS Glue の 1 つ以上のリダイレクト URL を入力します。

   リダイレクト URL の形式は次のとおりです:

   https://*region*.console.aws.amazon.com/gluestudio/oauth

   この URL では、region は Salesforce からのデータ転送に AWS Glue を使用するときの AWS リージョンのコードです。例えば、米国東部 (バージニア北部) リージョンのコードは `us-east-1` です。そのリージョンの URL は次のとおりです。

   https://us-east-1.console.aws.amazon.com/gluestudio/oauth

   AWS Glue がサポートする AWS リージョンのリストについては、「AWS 全般のリファレンス」の「AWS Glue エンドポイントとクォータ」を参照してください。
3. [ウェブサーバーフローにシークレットをリクエストする] チェックボックスをオンにします。
4. [利用可能な OAuth スコープ] リストに、次のスコープを追加します。
   - API (api) でユーザーデータを管理する
   - カスタムアクセス許可 (custom_permissions) にアクセスする
   - ID URL サービス (ID、プロファイル、E メール、アドレス、電話) にアクセスする
   - 一意のユーザー識別子 (openid) にアクセスする
   - リクエストを随時実行する (refresh_token、offline_access)
5. 接続アプリの更新トークンポリシーを [更新トークンは取り消されるまで有効] に設定します。そうしないと、更新トークンの有効期限が切れるとジョブが失敗します。更新トークンポリシーを確認および編集する方法の詳細については、Salesforce ドキュメントの「Manage OAuth Access Policies for a Connected App」を参照してください。

Salesforce の JWT ベアラ OAuth フローを設定する

OAuth 2.0 JSON Web トークン とのサーバー間統合を有効にするには、Salesforce パブリックドキュメントを参照してください。

JWT を作成し、Salesforce で接続アプリを適切に設定したら、Secrets Manager シークレットの `JWT_TOKEN` キーセットを使用して新しい Salesforce 接続を作成できます。接続の作成時に OAuth 付与タイプを [JWT ベアラートークン] に設定します。

Salesforce Marketing Cloud への接続

Salesforce Marketing Cloud は、E メール、モバイル、ソーシャル、オンラインマーケティング用のマーケティングオートメーションおよび分析ソフトウェアのプロバイダーです。また、コンサルティングおよび実装サービスも提供します。Salesforce Marketing Cloud ユーザーとして、Salesforce Marketing Cloud アカウントに AWS Glue を接続できます。次に、ETL ジョブのデータソースまたは送信先として Salesforce Marketing Cloud を使用できます。これらのジョブを実行して、Salesforce Marketing Cloud と AWS サービス、またはその他のサポートされているアプリケーション間でデータを転送します。

トピック

- AWS Glue Salesforce Marketing Cloud のサポート
- 接続を作成および使用するためのAPIオペレーションを含むポリシー
- Salesforce Marketing Cloud の設定
- Salesforce Marketing Cloud 接続の設定
- Salesforce Marketing Cloud エンティティからの読み取り
- Salesforce Marketing Cloud エンティティへの書き込み
- Salesforce Marketing Cloud 接続オプション
- Salesforce Marketing Cloud コネクタの制限と注意事項

AWS Glue Salesforce Marketing Cloud のサポート

AWS Glue は、次のように Salesforce Marketing Cloud をサポートします。

ソースとしてサポートされていますか?

はい。ジョブを使用して AWS Glue ETL、Salesforce Marketing Cloud からデータをクエリできます。

ターゲットとしてサポートされているかどうか

いいえ。

サポートされている Salesforce Marketing Cloud APIバージョン

次の Salesforce Marketing Cloud APIバージョンがサポートされています。

- v1

接続を作成および使用するためのAPIオペレーションを含むポリシー

次のサンプルポリシーでは、接続の作成と使用に必要な AWS IAMアクセス許可について説明します。新しいロールを作成する場合は、以下を含むポリシーを作成します。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:ListConnectionTypes",
                "glue:DescribeConnectionType",
                "glue:RefreshOAuth2Tokens"
                "glue:ListEntities",
                "glue:DescribeEntity"
            ],
            "Resource": "*"
        }
    ]
}
```

上記の方法を使用しない場合は、次の管理IAMポリシーを使用します。

- AWSGlueServiceRole – さまざまな AWS Glue プロセスがユーザーに代わって実行するために必要なリソースへのアクセスを許可します。これらのリソースには AWS Glue、、Amazon S3、IAM、 CloudWatch Logs、Amazon が含まれますEC2。このポリシーで指定されたリソースの命名規則に従うと、 AWS Glue プロセスには必要なアクセス許可があります。このポリシーは、通常、クローラ、ジョブ、開発エンドポイントを定義するときに指定されたロールにアタッチされます。
- AWSGlueConsoleFullAccess — ポリシーがアタッチされている ID が AWS マネジメントコンソールを使用する場合、 AWS Glue リソースへのフルアクセスを許可します。このポリシーで指定されたリソースの命名規則に従った場合、ユーザーは完全なコンソール機能を使用できます。このポリシーは通常、 AWS Glue コンソールのユーザーにアタッチされます。

Salesforce Marketing Cloud の設定

AWS Glue を使用して Salesforce Marketing Cloud からデータを転送できるようにするには、次の要件を満たす必要があります:

最小要件

以下に、最小要件を示します。

- Salesforce Marketing Cloud アカウントを持っている。詳細については、「Salesforce Marketing Cloud アカウントの作成」を参照してください。
- Salesforce Marketing Cloud アカウントで API アクセスが有効になっている。Enterprise、Unlimited、Developer、Performance の各エディションで API アクセスがデフォルトで有効になっている。

これらの要件を満たすと、Salesforce Marketing Cloud アカウントに AWS Glue を接続する準備が整います。一般的な接続では、Salesforce Marketing Cloud で他に何もする必要はありません。

Salesforce Marketing Cloud アカウントの作成

Salesforce Marketing クラウドの場合は、アカウント作成のためにベンダーに連絡する必要があります。お客様またはお客様の会社に Salesforce との結びつきがある場合は、Salesforce アカウントマネージャーに連絡して Salesforce Marketing Cloud ライセンスをリクエストしてください。それ以外の場合は、次のように Salesforce 担当者からの問い合わせをリクエストできます:

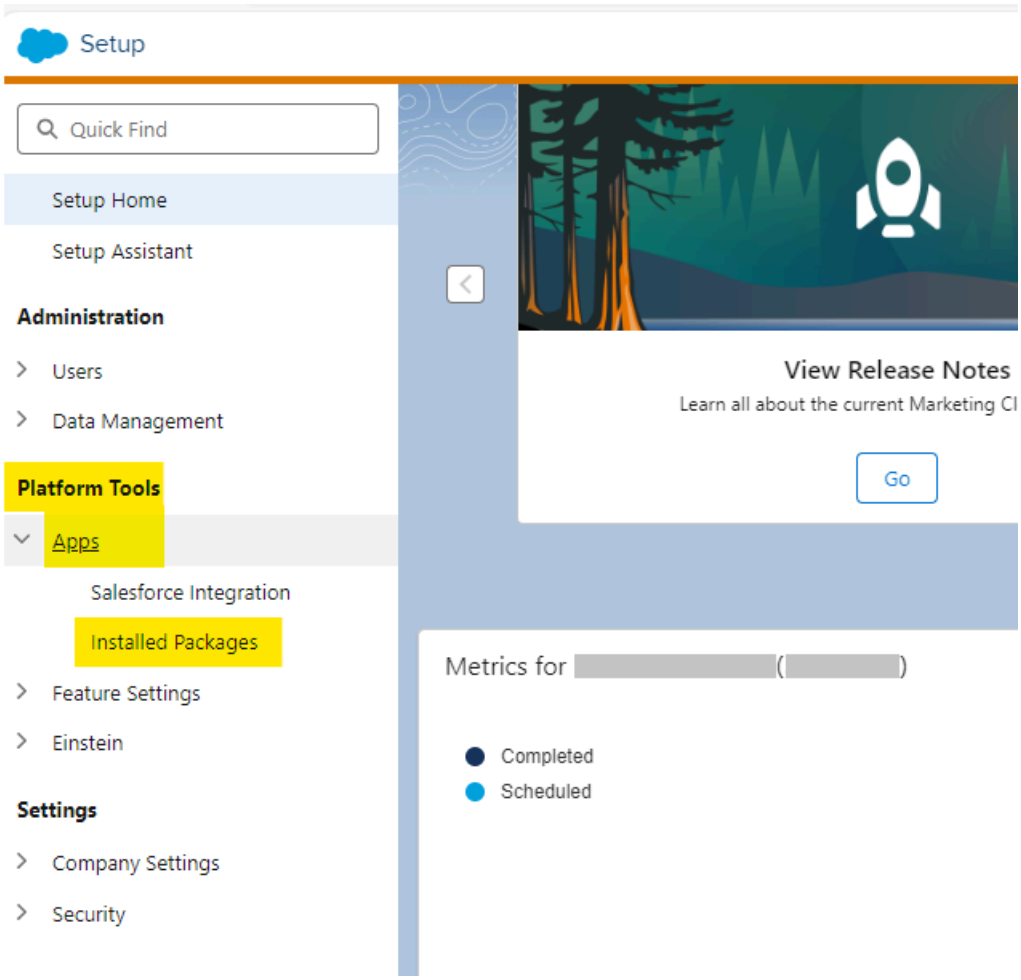
1. https://www.salesforce.com/in/products/marketing-cloud/overview/ に移動し、[サインアップ] を選択します。
2. ページの右上にある [お問い合わせ] リンクを選択します。
3. 必要な情報をフォームに入力して、[お問い合わせ] を選択します。

Salesforce の担当者から要件について連絡があります。

プロジェクトと OAuth 2.0 認証情報の作成

プロジェクトと OAuth 2.0 認証情報を取得するには:

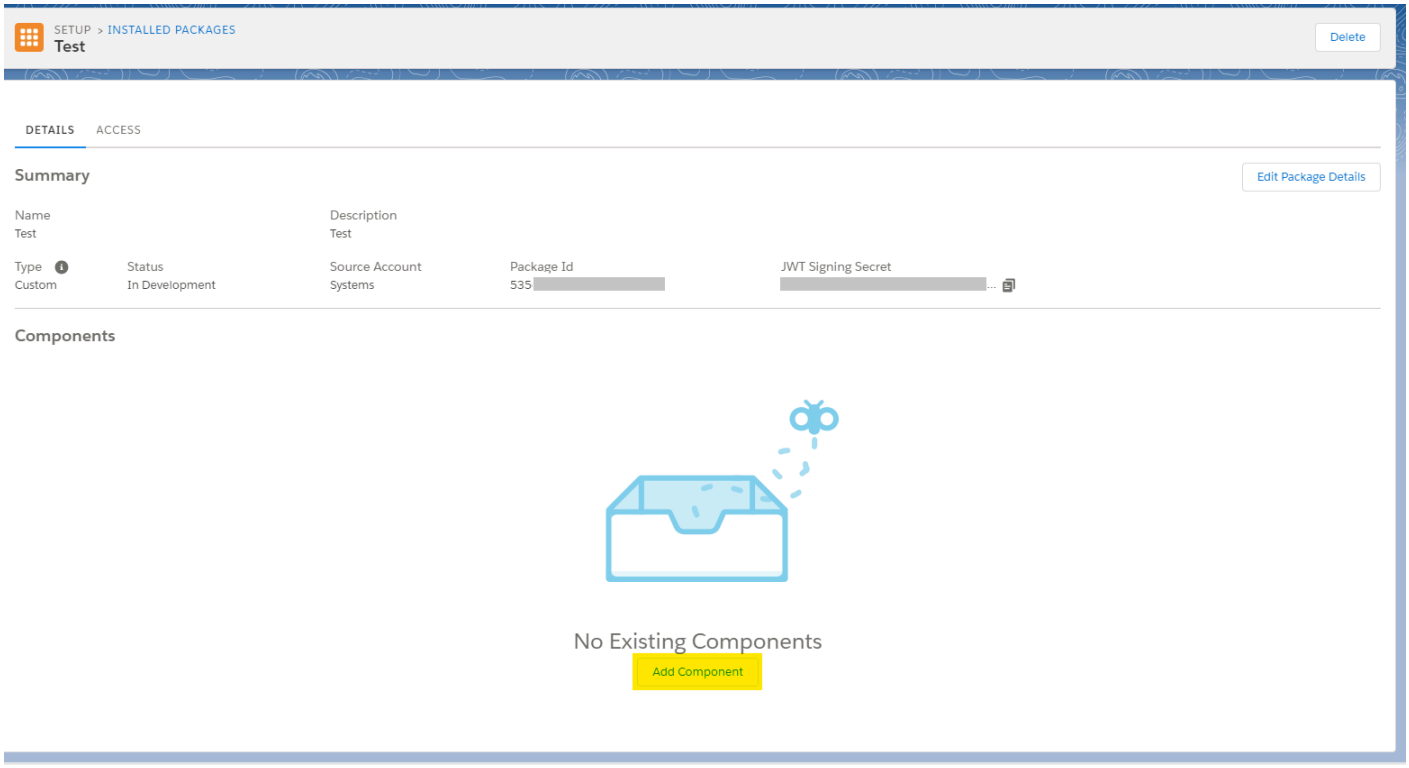
1. ユーザー名とパスワードを使用して Salesforce Marketing Cloud インスタンスにログインし、登録済みの携帯電話番号を使用して認証します。
2. 右上隅にあるプロファイルをクリックし、[セットアップ] に移動します。
3. [プラットフォームツール] で [アプリ] を選択し、[インストール済みパッケージ] を選択します。



4. [インストール済みパッケージ] ページで、右上隅にある [新規] をクリックします。パッケージの名前と説明を入力します。

   パッケージを保存します。パッケージが保存されると、パッケージの詳細を表示できます。

5. パッケージの [詳細] ページの [コンポーネント] セクションで、[コンポーネントの追加] を選択します。



6. [コンポーネントタイプ] を「API 統合」として選択し、[次へ] をクリックします。
7. [統合タイプ] を「Server-to-Server」(クライアント認証情報 OAuth グラントタイプを持つ) として選択し、[次へ] をクリックします。
8. 要件に基づいてスコープを追加し、[保存] をクリックします。

Salesforce Marketing Cloud 接続の設定

Salesforce Marketing Cloud は、OAuth2 の CLIENT CREDENTIALS 付与タイプをサポートしています。

- このグラントタイプは、クライアントがユーザーのコンテキスト外でアクセストークンを取得するために使用するため、2-legged の OAuth 2.0 と見なされます。AWS Glue は、クライアント ID とクライアントシークレットを使用して、定義したカスタムサービスによって提供される Salesforce Marketing Cloud API を認証できます。
- 各カスタムサービスは、特定のアクションを実行することをサービスに許可する一連のロールとアクセス許可を持つ API 専用ユーザーによって所有されます。アクセストークンは 1 つのカスタムサービスに関連付けられています。
- このグラントタイプは、有効期間が短いアクセストークンになり、ID エンドポイントを呼び出すことで更新される場合があります。

- クライアント認証情報を使用した OAuth 2.0 の Salesforce Marketing Cloud の公開ドキュメントについては、「拡張パッケージの開発環境を設定する」を参照してください。

Salesforce Marketing Cloud 接続を設定するには:

1. AWS Secrets Manager で、次の詳細を含むシークレットを作成します。
   a. カスタマーマネージド接続アプリケーションの場合、シークレットには、USER_MANAGED_CLIENT_APPLICATION_CLIENT_SECRET を使用して接続されたアプリケーションのコンシューマーシークレットをキーとして含める必要があります。
   b. 注: AWS Glue で接続にシークレットを作成する必要があります。
2. AWS Glue Glue Studio で、以下の手順に従って [データ接続] の下に接続を作成します。
   a. [接続タイプ] を選択する際に、[Salesforce Marketing Cloud] を選択します。
   b. 接続先の Salesforce Marketing Cloud の Subdomain Endpoint を指定します。
   c. 次のアクションを実行でき、AWS Glue がその権限を持つ AWS IAM ロールを選択します。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:DescribeSecret",
                "secretsmanager:GetSecretValue",
                "secretsmanager:PutSecretValue",
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterface",
                "ec2:DeleteNetworkInterface",
            ],
            "Resource": "*"
        }
    ]
}
```

   d. AWS Glue でこの接続に使用する secretName を選択して、トークンを配置します。
   e. ネットワークを使用する場合は、ネットワークオプションを選択します。
3. AWS Glue ジョブに関連付けられている IAM ロールに secretName を読み取るアクセス許可を付与します。

Salesforce Marketing Cloud エンティティからの読み取り

前提条件

読み取り元の Salesforce Marketing Cloud オブジェクト。`Activity` または `Campaigns` などのオブジェクト名が必要です。次の表に、サポートされているエンティティを示します。

ソースに対応するエンティティ:

エンティティ	インターフェイス	フィルタリング可能	制限をサポートする	Order By をサポートする	SELECT * をサポートする	パーティション分割をサポートする
Event Notification Callback	REST	いいえ	いいえ	いいえ	あり	いいえ
Seed-List	REST	いいえ	あり	なし	あり	いいえ
セットアップ	REST	はい	はい	なし	あり	いいえ
ドメインの検証	REST	はい	あり	あり	はい	いいえ
Objects Nested Tags	REST	はい	いいえ	いいえ	あり	いいえ
問い合わせ	REST	いいえ	あり	なし	あり	いいえ
Event Notification Subscription	REST	いいえ	いいえ	いいえ	あり	いいえ
メッセージング	REST	いいえ	あり	なし	あり	いいえ

エンティティ	インターフェイス	フィルタリング可能	制限をサポートする	Order By をサポートする	SELECT * をサポートする	パーティション分割をサポートする
アクティビティ	SOAP	いいえ	いいえ	いいえ	はい	はい
Bounce Event	SOAP	いいえ	いいえ	いいえ	はい	はい
Click Event	SOAP	いいえ	いいえ	いいえ	はい	はい
Content Area	SOAP	いいえ	いいえ	いいえ	はい	はい
Data Extension	SOAP	いいえ	あり	なし	はい	はい
E メール	SOAP	いいえ	あり	なし	はい	はい
Forwarded Email Event	SOAP	いいえ	あり	なし	はい	はい
Forward Email OptInEvent	SOAP	いいえ	あり	なし	はい	はい
リンク	SOAP	いいえ	あり	なし	はい	はい
Link Send	SOAP	いいえ	あり	なし	はい	はい
リスト	SOAP	いいえ	あり	なし	はい	はい
List Subscriber	SOAP	いいえ	あり	なし	はい	はい
Not Sent Event	SOAP	いいえ	あり	なし	はい	はい

エンティティ	インターフェイス	フィルタリング可能	制限をサポートする	Order By をサポートする	SELECT * をサポートする	パーティション分割をサポートする
Open Event	SOAP	いいえ	あり	なし	はい	はい
送信	SOAP	いいえ	あり	なし	はい	はい
Sent Event	SOAP	いいえ	あり	なし	はい	はい
サブスクライバー	SOAP	いいえ	あり	なし	はい	はい
Survey Event	SOAP	いいえ	あり	なし	はい	はい
Unsub Event	SOAP	いいえ	あり	なし	はい	はい
Audit Events	REST	いいえ	はい	あり	はい	いいえ
キャンペーン	REST	いいえ	はい	あり	はい	いいえ
インタラクション	REST	いいえ	はい	あり	はい	いいえ
Content Assets	REST	いいえ	はい	あり	はい	いいえ

REST の例:

```
salesforcemarketingcloud _read = glueContext.create_dynamic_frame.from_options(
    connection_type="salesforcemarketingcloud",
    connection_options={
        "connectionName": "connectionName",
```

```
        "ENTITY_NAME": "Campaigns",
        "API_VERSION": "v1",
        "INSTANCE_URL": "https://**********.rest.marketingcloudapis.com"
    }
```

SOAP の例:

```
salesforcemarketingcloud _read = glueContext.create_dynamic_frame.from_options(
    connection_type="salesforcemarketingcloud",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "Activity",
        "API_VERSION": "v1",
        "INSTANCE_URL": "https://**********.soap.marketingcloudapis.com"
    }
```

Salesforce Marketing Cloud エンティティとフィールドの詳細:

次の表に、Salesforce Marketing Cloud エンティティを示します。静的メタデータを持つ REST エンティティと動的メタデータを持つ SOAP エンティティがあります。

静的メタデータを持つ REST エンティティ:

エンティティ	フィールド	データ型	サポートされている演算子
Event Notification Callback	callbackId	String	
	callbackName	String	
	url	String	
	maxBatchSize	整数	
	ステータス	String	
	statusReason	String	
Seed-List	id	String	
	名前	String	

エンティティ	フィールド	データ型	サポートされている演算子
	description	文字列	
	activeSeedCount	整数	
セットアップ	customerKey	String	
	名前	String	
	description	文字列	
	locationType	String	'='
	awsFileTransferLocation	Struct	
Domain verification	enterpriseId	整数	
	ステータス	String	'='
	domainType	String	'='
	memberId	整数	
	emailSendTime	DateTime	
	ドメイン	String	
	isSendable	ブール値	
Objects Nested Tags	id	整数	
	modifiedDate	DateTime	
	tags	リスト	
	名前	String	
	description	文字列	

エンティティ	フィールド	データ型	サポートされている演算子
	parentId	整数	
連絡先	値	リスト	
Event Notification Subscription	subscriptionName	String	
	callbackId	String	
	callbackName	String	
	eventCategoryTypes	リスト	
	フィルター	リスト	
	url	String	
	maxBatchSize	整数	
	subscriptionId	String	
	ステータス	String	
	statusReason	String	
メッセージング	deliveryTime	DateTime	
	id	String	
	messageId	String	
	ステータス	String	
	次のように変更します。	Struct	
インタラクション	ステータス	String	'='
	id	String	

エンティティ	フィールド	データ型	サポートされている演算子
	キー	String	
	名前	String	
	lastPublishedDate	DateTime	
	description	文字列	
	version	整数	
	workflowApiVersion	整数	
	createdDate	DateTime	
	modifiedDate	DateTime	
	goals	Struct	
	stats	Struct	
	entryMode	String	
	defaults	Struct	
	executionMode	Struct	
	definitionId	String	
Content Assets	id	整数	
	customerKey	String	
	objectId	String	
	contentType	String	
	assetType	Struct	
	名前	String	

エンティティ	フィールド	データ型	サポートされている演算子
	description	文字列	
	owner (オーナー)	Struct	
	createdDate	DateTime	
	CreatedBy	Struct	
	modifiedDate	DateTime	
	modifiedBy	Struct	
	thumbnail	Struct	
	category	Struct	
	meta	Struct	
	ビュー	Struct	
	availableViews	Struct	
	データ	Struct	
	legacyData	Struct	
	modelVersion	整数	
	バージョン	整数	
	ロック	ブール値	
	FileProperties	Struct	
	[タグ]	リスト	
	コンテンツ	String	
	設計	String	

エンティティ	フィールド	データ型	サポートされている演算子
	SuperContent	String	
	CustomFields	Struct	
	ブロック	Struct	
	MinBlocks	整数	
	MaxBlocks	整数	
	チャンネル	Struct	
	AllowedBlocks	リスト	
	スロット	Struct	
	BusinessUnitAvailability	Struct	
	sharingProperties	Struct	
	sharingProperties.sharedWith	Struct	
	sharingProperties.sharingType	String	
	テンプレート	Struct	
	ファイル	String	
	GenerateFrom	String	
Audit Events	id	整数	
	createdDate	DateTime	
	memberId	整数	

エンティティ	フィールド	データ型	サポートされている演算子
	enterpriseId	整数	
	employee	Struct	
	objectType	Struct	
	オペレーション	Struct	
	オブジェクト	Struct	
	transactionId	String	
キャンペーン	id	整数	
	createdDate	DateTime	
	modifiedDate	DateTime	
	名前	String	
	description	文字列	
	campaignCode	[String] (文字列)	
	color	String	
	favorite	ブール値	

動的メタデータを持つ SOAP エンティティ:

エンティティ	データ型	サポートされている演算子
アクティビティ	String	LIKE,!=,=
	Struct	
	整数	!=,=,>=,<=,<,>

エンティティ	データ型	サポートされている演算子
	倍精度	!=,=,>=,<=,<,>
	ブール値	!=,=
	DateTime	>=,<=,<,>,=,BETWEEN
Bounce Event	整数	!=,=,>=,<=,<,>
	DateTime	>=,<=,<,>,=,BETWEEN
	String	LIKE,!=,=
	Struct	
Click Event	整数	!=,=,>=,<=,<,>
	DateTime	>=,<=,<,>,=,BETWEEN
	String	LIKE,!=,=
	Struct	
Content Area	Struct	
	String	LIKE,!=,=
	整数	!=,=,>=,<=,<,>
	DateTime	>=,<=,<,>,=,BETWEEN
	ブール値	!=,=
Data Extension	DateTime	>=,<=,<,>,=,BETWEEN
	String	LIKE,!=,=
E メール	整数	!=,=,>=,<=,<,>
	String	LIKE,!=,=

エンティティ	データ型	サポートされている演算子
	DateTime	>=,<=,<,>,=,BETWEEN
	ブール値	!=,=
	Struct	
Forwarded Email Event	整数	!=,=,>=,<=,<,>
	String	LIKE,!=,=
	DateTime	>=,<=,<,>,=,BETWEEN
	Struct	
Forwarded Email OptInEvent	整数	!=,=,>=,<=,<,>
	String	LIKE,!=,=
	DateTime	>=,<=,<,>,=,BETWEEN
	Struct	
リンク	整数	!=,=,>=,<=,<,>
Link Send	整数	!=,=,>=,<=,<,>
	String	LIKE,!=,=
	倍精度	!=,=,>=,<=,<,>
リスト	整数	!=,=,>=,<=,<,>
	String	LIKE,!=,=
	DateTime	>=,<=,<,>,=,BETWEEN
	Struct	
List Subscriber	整数	!=,=,>=,<=,<,>

エンティティ	データ型	サポートされている演算子
	String	LIKE,!=,=
	DateTime	>=,<=,<,>,=,BETWEEN
	Struct	
Not Sent Event	整数	!=,=,>=,<=,<,>
	String	LIKE,!=,=
	DateTime	>=,<=,<,>,=,BETWEEN
	Struct	
Open Event	整数	!=,=,>=,<=,<,>
	String	LIKE,!=,=
	DateTime	>=,<=,<,>,=,BETWEEN
	Struct	
送信	整数	!=,=,>=,<=,<,>
	String	LIKE,!=,=
	DateTime	>=,<=,<,>,=,BETWEEN
	ブール値	!=,=
	Struct	
Sent Event	整数	!=,=,>=,<=,<,>
	String	LIKE,!=,=
	DateTime	>=,<=,<,>,=,BETWEEN
	Struct	

エンティティ	データ型	サポートされている演算子
サブスクライバー	整数	!=,=,>=,<=,<,>
	String	LIKE,!=,=
	DateTime	>=,<=,<,>,=,BETWEEN
	Struct	
Survey Event	整数	!=,=,>=,<=,<,>
	String	LIKE,!=,=
	DateTime	>=,<=,<,>,=,BETWEEN
	Struct	
Unsub Event	整数	!=,=,>=,<=,<,>
	String	LIKE,!=,=
	DateTime	>=,<=,<,>,=,BETWEEN
	ブール値	!=,=
	Struct	

パーティショニングクエリ

Salesforce Marketing Cloud では、整数および DateTime データ型フィールドは、フィールドベースのパーティション分割をサポートします。

Spark で同時実行を使用する場合は、追加の Spark オプション `PARTITION_FIELD`、`LOWER_BOUND`、`UPPER_BOUND`、および `NUM_PARTITIONS` を指定できます。これらのパラメータを使用すると、元のクエリは Spark タスクで同時に実行できるサブクエリの `NUM_PARTITIONS` の数に分割されます。

- `PARTITION_FIELD`: クエリのパーティション化に使用するフィールドの名前。
- `LOWER_BOUND`: 選択したパーティションフィールドの包括的な下限値。

タイムスタンプフィールドでは、Spark SQL クエリで使用される Spark タイムスタンプ形式を受け入れます。

有効な値の例は次のとおりです:

```
“2024-05-07T02:03:00.00Z"
```

- UPPER_BOUND: 選択したパーティションフィールドの排他的上限値。
- NUM_PARTITIONS: パーティション数。

例:

```
salesforcemarketingcloud_read = glueContext.create_dynamic_frame.from_options(
    connection_type="salesforcemarketingcloud",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "ListSubscriber",
        "API_VERSION": "v1",
        "PARTITION_FIELD": "CreatedDate"
        "LOWER_BOUND": "2023-09-07T02:03:00.000Z"
        "UPPER_BOUND": "2024-05-07T02:03:00.000Z"
        "NUM_PARTITIONS": "10"
    }
```

Salesforce Marketing Cloud エンティティへの書き込み

前提条件

- 書き込み先の Salesforce Marketing オブジェクト。List または Campaigns などのオブジェクト名、または以下の表に概説されている他のエンティティのいずれかを指定する必要があります。
- Salesforce Marketing Cloud コネクタは、次の 3 つの書き込みオペレーションをサポートしています。
  - INSERT
  - UPSERT
  - UPDATE

  UPDATE および UPSERT 書き込みオペレーションを使用する場合は、ID_FIELD_NAMES オプションを指定してレコードの外部 ID フィールドを指定する必要があります。

送信先でサポートされているエンティティ:

エンティティ	優先度	インターフェイス (REST、SOAP など)	挿入可能	更新可能	アップサート可能
キャンペーン	P0	REST	Y- シングル	Y- シングル	N
Content Assets	P0	REST	Y- シングル、バルク	Y- シングル	N
連絡先	P1	REST	Y- シングル	Y- シングル	N
ドメインの検証	P1	REST	Y- シングル	Y- シングル、バルク	N
Event Notification Callback	P1	REST	Y- シングル	Y- シングル	N
Event Notification Subscription	P1	REST	Y- シングル	Y- シングル	N
メッセージング	P1	REST	Y- シングル	N	N
Object Nested Tag	P2	REST	Y- シングル	Y- シングル	N
Seed-List	P1	REST	Y- シングル	Y- シングル	N
セットアップ	P1	REST	Y- シングル	Y- シングル	N
Data Extension	P0	SOAP	Y- シングル	Y- シングル	Y- シングル
E メール	P0	SOAP	Y- シングル	Y- シングル	N

エンティティ	優先度	インターフェイス (REST、SOAP など)	挿入可能	更新可能	アップサート可能
リスト	P0	SOAP	Y- シングル	Y- シングル	N
送信	P0	SOAP	Y- シングル	N	N
サブスクライバー	P0	SOAP	Y- シングル	Y- シングル	N

REST の INSERT オペレーションの例:

```
salesforcemarketingcloud _write = glueContext.write_dynamic_frame.from_options(
    connection_type="salesforcemarketingcloud",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "Campaigns",
        "API_VERSION": "v1",
        "writeOperation" : "INSERT",
        "INSTANCE_URL": "https://**********************.rest.marketingcloudapis.com"
    }
```

SOAP の INSERT オペレーションの例:

```
salesforcemarketingcloud _write = glueContext.write_dynamic_frame.from_options(
    connection_type="salesforcemarketingcloud",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "List",
        "API_VERSION": "v1",
        "writeOperation" : "INSERT",
        "INSTANCE_URL": "https://**********************.rest.marketingcloudapis.com"
    }
```

REST の UPDATE オペレーションの例:

```
salesforcemarketingcloud _write = glueContext.write_dynamic_frame.from_options(
    connection_type="salesforcemarketingcloud",
```

```
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "Campaigns",
        "API_VERSION": "v1",
        "writeOperation" : "UPDATE",
         "ID_FIELD_NAMES": "id",
        "INSTANCE_URL": "https://**********.rest.marketingcloudapis.com"
    }
```

SOAP の UPDATE オペレーションの例:

```
salesforcemarketingcloud _write = glueContext.write_dynamic_frame.from_options(
    connection_type="salesforcemarketingcloud",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "List",
        "API_VERSION": "v1",
        "writeOperation" : "UPDATE",
         "ID_FIELD_NAMES": "id",
        "INSTANCE_URL": "https://**********.rest.marketingcloudapis.com"
    }
```

SOAP の UPSERT オペレーションの例:

```
salesforcemarketingcloud _write = glueContext.write_dynamic_frame.from_options(
    connection_type="salesforcemarketingcloud",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "DataExtension/Insert-***E/6*******3",
        "API_VERSION": "v1",
        "writeOperation" : "UPSERT",
        "INSTANCE_URL": "https://**********.rest.marketingcloudapis.com"
    }
```

Salesforce Marketing Cloud 接続オプション

Salesforce Marketing Cloud の接続オプションは次のとおりです。

- ENTITY_NAME(文字列) – (必須) 読み取りに使用されます。Salesforce Marketing Cloud のオブジェクトの名前。
- API_VERSION(文字列) – (必須) 読み取りに使用されます。使用する Salesforce Marketing Cloud Rest および SOAP API バージョン。

- SELECTED_FIELDS(List<String>) – Default: empty(SELECT *). 読み込みに使用されます。オブジェクトに選択する列。
- FILTER_PREDICATE(文字列) – デフォルト: 空 読み込みに使用されます。Spark SQL 形式である必要があります。
- QUERY(文字列) - デフォルト: 空 読み込みに使用されます。完全な Spark SQL クエリ。
- PARTITION_FIELD(文字列) – 読み取りに使用されます。クエリをパーティション化するために使用するフィールド。
- LOWER_BOUND(文字列) - 読み取りに使用されます。選択したパーティションフィールドの包括的な下限値。
- UPPER_BOUND (文字列) - 読み取りに使用されます。選択したパーティションフィールドの排他的上限値。
- NUM_PARTITIONS(整数) – デフォルト: 1。読み込みに使用されます。読み取り用のパーティションの数。
- WRITE_OPERATION(文字列) - デフォルト: INSERT。書き込みに使用されます。値は INSERT、UPDATE、UPSERT である必要があります。
- ID_FIELD_NAMES(文字列) - デフォルト: null。UPDATE/UPSERT に必要です。

Salesforce Marketing Cloud コネクタの制限と注意事項

Salesforce Marketing Cloud コネクタの制限または注意事項は次のとおりです。

- DateTime データ型フィールドでフィルターを使用する場合は、「yyyy-mm-ddThh:MM:ssZ」の形式で値を渡す必要があります。
- Data Preview では、Boolean Datatype 値は Blank になります。
- SOAP エンティティでは最大 2 つのフィルターを定義でき、REST エンティティでは 1 つのフィルターのみを定義できます。これにより、フィルターによるパーティショニングのテストが制限されます。
- SaaS 側から予期しない動作がいくつか確認されています。「linksend」エンティティの Link.Alias フィールドは CONTAINS 演算子をサポートしておらず (例: Link.Alias CONTAINS "ViewPrivacyPolicy")、Data Extension エンティティのフィルター演算子 (EQUALS や GREATER THAN など) は期待した結果を返しません。
- SFMC ClickEvent SOAP API では、新しく作成されたレコードの反映に遅延があるため、最近作成されたレコードは API レスポンスですぐには利用できない場合があります。

例: [2025-01-10T14:30:00] に 5 つの新しい ClickEvent レコードを作成し、SOAP API を使用してすぐに取得する場合、レスポンスには 5 つのレコードがすべて含まれていない可能性があります。新しく作成されたレコードが API レスポンスに表示されるまでに、最大 5 分かかる場合があります。この遅延は、データの取得とスケジュールされた実行の両方にも影響します。

- 2 つの異なる DateTime 形式: [2025-03-11T04:46:00] (ミリ秒なし) と [2025-03-11T04:46:00.000Z] は、AWS Glue (ミリ秒単位) で書き込みオペレーションを実行するときにサポートされます。
- Event Notification Subscription エンティティの場合、サブスクリプションは検証済みのコールバック URL に対してのみ作成でき、コールバックごとに最大 200 個のサブスクリプションを持つことができます。
- Event Notification Callback エンティティの場合、アカウントごとに最大 50 個のレコードを作成できます。

Salesforce Commerce Cloud への接続

B2C Commerce API は、B2C Commerce インスタンスとやり取りするための RESTful API のコレクションです。これは、Salesforce Commerce API、頭字語の SCAPI、または単に Commerce API など、いくつかの異なる名前で呼ばれています。

API を使用すると、デベロッパーはフルストアフロントからカスタムマーチャントツールまで、幅広いアプリケーションを構築して Business Manager を強化できます。B2C Commerce のすべてのお客様が API を追加料金なしで利用することできます。

API は、Shopper API と Admin API という 2 つの主要な API グループに分かれています。また、各グループ内では、API ファミリーと、関連する機能に重点を置いた小さなグループに分割されます。

トピック

- AWS Glue の Salesforce Commerce Cloud のサポート
- 接続を作成および使用するための API オペレーションを含むポリシー
- Salesforce Commerce Cloud の設定
- Salesforce Commerce Cloud 接続の設定
- Salesforce Commerce Cloud エンティティからの読み取り
- Salesforce Commerce Cloud 接続オプションのリファレンス
- 制限

AWS Glue の Salesforce Commerce Cloud のサポート

AWS Glue は、次のように Salesforce Commerce Cloud をサポートします。

ソースとしてサポートされていますか?

はい。AWS Glue ETL ジョブを使用して、Salesforce Commerce Cloud からデータをクエリできます。

ターゲットとしてサポートされていますか?

いいえ。

サポートされている Salesforce Commerce Cloud API バージョン

v1。

接続を作成および使用するための API オペレーションを含むポリシー

次のサンプル ポリシーで、接続の作成と使用に必要な AWS の権限について説明します。新しいロールを作成する場合は、以下を含むポリシーを作成します。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:ListConnectionTypes",
                "glue:DescribeConnectionType",
                "glue:RefreshOAuth2Tokens"
                "glue:ListEntities",
                "glue:DescribeEntity"
            ],
            "Resource": "*"
        }
    ]
}
```

また、次のマネージド IAM ポリシーを使用して、アクセスを許可できます。

- AWSGlueServiceRole – さまざまな AWS Glue プロセスを実行するために必要なリソースへのアクセス権をユーザーに代わって付与します。これらのリソースには、AWS Glue、Amazon

S3、IAM、CloudWatch Logs、および Amazon EC2 が含まれます。このポリシーで指定されたリソースの命名規則に従った場合、AWS Glue プロセスは必要なアクセス権限を使用できます。このポリシーは、通常、クローラ、ジョブ、開発エンドポイントを定義するときに指定されたロールにアタッチされます。

- AWSGlueConsoleFullAccess — ポリシーがアタッチされているアイデンティティが AWS マネージメントコンソールを使用するときは、AWS Glue リソースへのフルアクセスを許可します。このポリシーで指定されたリソースの命名規則に従った場合、ユーザーは完全なコンソール機能を使用できます。このポリシーは、通常 AWS Glue コンソールのユーザーにアタッチされています。

Salesforce Commerce Cloud の設定

AWS Glue を使用して Salesforce Commerce Cloud からデータを転送できるようにするには、次の要件を満たす必要があります。

最小要件

- clientId と clientSecret を持つ Salesforce Commerce Cloud クライアントアプリケーションがある。
- Salesforce Commerce Cloud アカウントで API アクセスが有効になっている。

これらの要件を満たすと、Salesforce Commerce Cloud アカウントに AWS Glue を接続する準備が整います。一般的な接続では、Salesforce Commerce Cloud で他に何もする必要はありません。

Salesforce Commerce Cloud 接続の設定

Salesforce Commerce Cloud は、OAuth2 の CLIENT CREDENTIALS グラントタイプをサポートしています。

- このグラントタイプは、クライアントがユーザーのコンテキスト外でアクセストークンを取得するために使用するため、2-legged の OAuth 2.0 とみなされます。AWS Glue は、クライアント ID とクライアントシークレットを使用して、定義したカスタムサービスによって提供される Salesforce Commerce Cloud API を認可できます。
- 各カスタムサービスは、特定のアクションを実行することをサービスに許可する一連のロールとアクセス許可を持つ API 専用ユーザーによって所有されます。アクセストークンは 1 つのカスタムサービスに関連付けられています。
- このグラントタイプは、有効期間が短いアクセストークンになり、ID エンドポイントを呼び出すことで更新される場合があります。

- クライアント認証情報の生成に関する Salesforce Commerce Cloud ドキュメントの詳細については、Salesforce のドキュメントを参照してください。

Salesforce Commerce Cloud 接続を設定するには:

1. AWS Secrets Manager で、次の詳細を含むシークレットを作成します。AWS Glue で接続ごとにシークレットを作成する必要があります。
   a. カスタマーマネージド接続アプリケーションの場合 – シークレットには、USER_MANAGED_CLIENT_APPLICATION_CLIENT_SECRET を使用して接続されたアプリケーションのコンシューマーシークレットをキーとして含める必要があります。
2. AWS Glue Studio で、以下の手順に従って [データ接続] の下に接続を作成します。
   a. [データ接続] の [接続を作成] を選択します。
   b. [データソース] を選択する際に、[Salesforce Commerce Cloud] を選択します。
   c. Salesforce Commerce Cloud のショートコード、組織 ID、サイト ID を指定します。
   d. Salesforce Commerce Cloud アカウントの Salesforce Commerce Cloud ドメイン URL を選択します。
   e. AWS Glue が次のアクションを担い、その権限を持つことができる IAM ロールを選択します。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:DescribeSecret",
                "secretsmanager:GetSecretValue",
                "secretsmanager:PutSecretValue",
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterface",
                "ec2:DeleteNetworkInterface",
            ],
            "Resource": "*"
        }
    ]
}
```

   f. OAuth スコープを指定します。オプションで、接続する Salesforce Commerce Cloud の [ユーザーマネージドクライアントアプリケーション ClientId] を入力します。

g. AWS Glue でこの接続に使用する `secretName` を選択して、トークンを配置します。

h. ネットワークを使用する場合は、ネットワークオプションを選択します。

3. AWS Glue ジョブに関連付けられている IAM ロールに `secretName` を読み取るアクセス許可を付与します。
4. AWS Glue ジョブ設定で、[追加のネットワーク接続] として `connectionName` を指定します。

Salesforce Commerce Cloud エンティティからの読み取り

前提条件

- 読み取り元の Salesforce Commerce Cloud オブジェクト。使用可能なエンティティを確認するには、以下のサポートされているエンティティの表を参照してください。

サポートされているエンティティ

エンティティ	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
割り当て	あり	あり	あり	あり	あり
キャンペーン	あり	あり	あり	あり	あり
カタログ	あり	あり	あり	あり	あり
カテゴリ	あり	あり	あり	あり	あり
Coupons	あり	あり	あり	あり	あり
ギフト券	あり	あり	あり	あり	あり
製品	あり	あり	あり	あり	あり
プロモーション	あり	あり	あり	あり	あり
ソースコードグループ	あり	あり	あり	あり	あり

例

```
salesforce_commerce_cloud_read = glueContext.create_dynamic_frame.from_options(
     connection_type="SalesforceCommerceCloud",
     connection_options={
         "connectionName": "connectionName",
         "ENTITY_NAME": "campaign",
         "API_VERSION": "v1"
     }
```

Salesforce Commerce Cloud エンティティとフィールドの詳細

エンティティリスト:

- 割り当て: https://developer.salesforce.com/docs/commerce/commerce-api/references/assignments
- キャンペーン: https://developer.salesforce.com/docs/commerce/commerce-api/references/campaigns
- カタログ: https://developer.salesforce.com/docs/commerce/commerce-api/references/catalogs
- カテゴリ: https://developer.salesforce.com/docs/commerce/commerce-api/references/catalogs?meta=searchCategories
- ギフト券: https://developer.salesforce.com/docs/commerce/commerce-api/references/gift-certificates
- 製品: https://developer.salesforce.com/docs/commerce/commerce-api/references/products
- プロモーション: https://developer.salesforce.com/docs/commerce/commerce-api/references/promotions
- ソースコードグループ: https://developer.salesforce.com/docs/commerce/commerce-api/references/source-code-groups

パーティショニングクエリ

Spark で同時実行を使用する場合は、追加の Spark オプション PARTITION_FIELD、LOWER_BOUND、UPPER_BOUND、および NUM_PARTITIONS を指定できます。これらのパラメータを使用すると、元のクエリは Spark タスクで同時に実行できるサブクエリの NUM_PARTITIONS の数に分割されます。

- PARTITION_FIELD: クエリのパーティション化に使用するフィールドの名前。
- LOWER_BOUND: 選択したパーティションフィールドの包括的な下限値。

日付については、Spark SQL クエリで使用される Spark の日付形式を受け入れます。有効な値の例: "2024-02-06"。

- UPPER_BOUND: 選択したパーティションフィールドの排他的上限値。
- NUM_PARTITIONS: パーティション数。

エンティティごとのパーティション分割フィールドのサポートの詳細は、以下の表にまとめられています。

エンティティ	Partitioning Field	DataType
キャンペーン	lastModified	DateTime
キャンペーン	startDate	DateTime
キャンペーン	endDate	DateTime
カタログ	creationDate	DateTime
カテゴリ	creationDate	DateTime
ギフト券	merchantId	String
ギフト券	creationDate	DateTime
製品	creationDate	DateTime
製品	lastModified	DateTime
ソースコードグループ	creationDate	DateTime
ソースコードグループ	startTime	DateTime
ソースコードグループ	endTime	DateTime

例

```
salesforceCommerceCloud_read = glueContext.create_dynamic_frame.from_options(
```

```
    connection_type="SalesforceCommerceCloud",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "coupons",
        "API_VERSION": "v1",
        "PARTITION_FIELD": "creationDate"
        "LOWER_BOUND": "2020-05-01T20:55:02.000Z"
        "UPPER_BOUND": "2024-07-11T20:55:02.000Z"
        "NUM_PARTITIONS": "10"
    }
```

Salesforce Commerce Cloud 接続オプションのリファレンス

Salesforce Commerce Cloud の接続オプションは次のとおりです。

- ENTITY_NAME(文字列) – (必須) 読み取りに使用されます。Salesforce Commerce Cloud のオブジェクトの名前。
- API_VERSION(文字列) – (必須) 読み取り / 書き込みに使用されます。使用する Salesforce Commerce Cloud Rest API バージョン。例: v1。
- SELECTED_FIELDS(List<String>) – Default: empty(SELECT *). 読み込みに使用されます。オブジェクトに選択する列。
- FILTER_PREDICATE(文字列) – デフォルト: 空 読み込みに使用されます。Spark SQL 形式である必要があります。
- QUERY(文字列) - デフォルト: 空 読み込みに使用されます。完全な Spark SQL クエリ。
- PARTITION_FIELD(文字列) – 読み取りに使用されます。クエリをパーティション化するために使用するフィールド。
- LOWER_BOUND(文字列) - 読み取りに使用されます。選択したパーティションフィールドの包括的な下限値。
- UPPER_BOUND (文字列) - 読み取りに使用されます。選択したパーティションフィールドの排他的上限値。
- NUM_PARTITIONS(整数) – デフォルト: 1。読み込みに使用されます。読み取り用のパーティションの数。

制限

Salesforce Commerce Cloud コネクタの制限は次のとおりです。

- パーティショニング時、[次を含む] フィルターが期待どおりに機能しません。
- CDN ゾーンのエンティティはサンドボックスインスタンスをサポートしておらず、開発インスタンスタイプと本番インスタンスタイプのみをサポートしています。詳細については、https://help.salesforce.com/s/articleView?id=cc.b2c_embedded_cdn_overview.htm を参照してください。
- Salesforce Commerce Cloud には、動的メタデータを取得するための API エンドポイントがありません。そのため、製品とカテゴリエンティティのカスタムフィールドをサポートするプロビジョニングはありません。
- サイト ID は必須のクエリパラメータです。カスタムコネクタ設定を通じてサイト ID の値を渡す必要があります。詳細については、「Base URL and Request Formation 」を参照してください。
- 次の表に示すように、異なる演算子の組み合わせを使用して、1 つの API リクエストで最大 2 つのフィールド (存在する場合はレベルを除く) にフィルターを適用できます。

フィルター条件	サポートの有無
1 つの API リクエストで CONTAINS 演算子を使用する 1 つのフィールド。	あり
1 つの API リクエストで Equals 演算子を使用する 1 つのフィールド。	あり
1 つの API リクエストで BETWEEN 演算子を使用する 1 つのフィールド。	あり
1 つの API リクエストで CONTAINS 演算子を使用する 2 つ以上のフィールド。	いいえ
1 つの API リクエストで Equals 演算子を使用する 2 つ以上のフィールド。	いいえ
1 つの API リクエストで BETWEEN 演算子を使用する 2 つ以上のフィールド。	いいえ
1 つの API リクエストで Equals を使用する 1 つのフィールドと CONTAINS 演算子を使用する 1 つのフィールド。	あり

フィルター条件	サポートの有無
1 つの API リクエストで BETWEEN を使用する 1 つのフィールドと CONTAINS 演算子を使用する 1 つのフィールド。	あり
1 つの API リクエストで BETWEEN を使用する 1 つのフィールドと Equals 演算子を使用する 1 つのフィールド。	あり
1 つの API リクエストで Equals を使用する 1 つのフィールド、CONTAINS を使用する 1 つのフィールド、および BETWEEN 演算子を使用する 1 つのフィールド。	いいえ
1 つの API リクエストで INCREMENTAL PULL が適用される場合、Equals 演算子を使用する 1 つのフィールド。	あり
1 つの API リクエストで INCREMENTAL PULL が適用される場合、CONTAINS 演算子を使用する 1 つのフィールド。	あり
1 つの API リクエストで INCREMENTAL PULL が適用される場合、BETWEEN 演算子を使用する 1 つのフィールド。	いいえ
1 つの API リクエストで INCREMENTAL PULL が適用される場合、1 つの Equals と 1 つの CONTAINS 演算子。	いいえ

- 一部のエンティティでは、取得中のフィールドのデータ型が、検索可能なフィールドとして使用される場合のデータ型と異なります。そのため、これらのフィールドにはフィルター機能のプロビジョニングがありません。このようなフィールドの詳細を次の表に示します。

シリアル番号	エンティティ名	検索可能なフィールド名	検索可能なデータ型	取得可能なデータ型
1	カタログ	名前	String	Struct
2	カタログ	description	文字列	Struct
3	カテゴリ	名前	String	Struct
4	カテゴリ	description	文字列	Struct
5	製品	名前	String	Struct
6	製品	searchable	ブール値	Struct
7	製品	validFrom	DateTime	Struct
8	製品	validTo	DateTime	Struct
9	製品	type	String	Struct
10	製品	onlineFlag	ブール値	Struct
11	プロモーション	名前	String	Struct

Salesforce Marketing Cloud Account Engagement への接続

Salesforce Marketing Cloud Account Engagement は、企業が有意義なつながりを創造し、より多くのパイプラインを生成し、より多くの取引を成立させるのに役立つマーケティング自動化ソリューションです。Salesforce Marketing Cloud Account Engagement ユーザーの場合は、Salesforce Marketing Cloud Account Engagement アカウントに AWS Glue を接続できます。Salesforce Marketing Cloud Account Engagement は、ETL ジョブのデータソースとして使用できます。これらのジョブを実行して、Salesforce Marketing Cloud Account Engagement から AWS サービスまたはその他のサポートされているアプリケーションにデータを転送します。

トピック

- AWS Glue による Salesforce Marketing Cloud Account Engagement のサポート
- 接続を作成および使用するための API オペレーションを含むポリシー

- Salesforce Marketing Cloud Account Engagement の設定
- Salesforce Marketing Cloud Account Engagement 接続の設定
- Salesforce Marketing Cloud Account Engagement エンティティからの読み取り
- Salesforce Marketing Cloud Account Engagement 接続オプション
- Salesforce Marketing Cloud Account Engagement コネクタの制限と注意事項

AWS Glue による Salesforce Marketing Cloud Account Engagement のサポート

AWS Glue は、次のように Salesforce Marketing Cloud Account Engagement をサポートします:

ソースとしてサポートされていますか?

はい。AWS Glue ETL ジョブを使用して、Async モードまたは Sync モードのいずれかで Salesforce Marketing Cloud Account Engagement からデータをクエリできます。

ターゲットとしてサポートされていますか?

いいえ。

サポートされている Salesforce Marketing Cloud Account Engagement API のバージョン

次の Salesforce Marketing Cloud Account Engagement API バージョンがサポートされています:

- v5

接続を作成および使用するための API オペレーションを含むポリシー

次のサンプル ポリシーで、接続の作成と使用に必要な AWS IAM 権限について説明します。新しいロールを作成する場合は、以下を含むポリシーを作成します。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:ListConnectionTypes",
                "glue:DescribeConnectionType",
                "glue:RefreshOAuth2Tokens"
```

```
                "glue:ListEntities",
                "glue:DescribeEntity"
            ],
            "Resource": "*"
        }
    ]
}
```

上記の方法を使用しない場合は、代わりに次のマネージド IAM ポリシーを使用します:

- AWSGlueServiceRole – さまざまな AWS Glue プロセスを実行するために必要なリソースへのアクセス権をユーザーに代わって付与します。これらのリソースには AWS Glue 、Amazon S3、IAM、CloudWatch Logs、および Amazon EC2 が含まれます。このポリシーで指定されたリソースの命名規則に従った場合、AWS Glue プロセスは必要なアクセス権限を使用できます。このポリシーは、通常、クローラ、ジョブ、開発エンドポイントを定義するときに指定されたロールにアタッチされます。
- AWSGlueConsoleFullAccess — ポリシーがアタッチされているアイデンティティが AWS マネージメントコンソールを使用するときは、AWS Glue リソースへのフルアクセスを許可します。このポリシーで指定されたリソースの命名規則に従った場合、ユーザーは完全なコンソール機能を使用できます。このポリシーは、通常 AWS Glue コンソールのユーザーにアタッチされています。

Salesforce Marketing Cloud Account Engagement の設定

AWS Glue を使用して Salesforce Marketing Cloud Account Engagement からデータを転送できるようにするには、次の要件を満たす必要があります:

最小要件

以下に、最小要件を示します。

- Salesforce マーケティングアカウントを持っていること。
- Salesforce アカウントのライセンスされた Account Engagement プランがある。
- Salesforce ユーザーと Account Engagement ユーザーを同期している。
- OAuth 認証情報を取得するために、App Manager で新しい接続アプリケーションを作成している。

これらの要件を満たすと、Salesforce Marketing Cloud Account Engagement アカウントに AWS Glue を接続する準備が整います。

Salesforce Marketing Cloud Account Engagement 接続の設定

グラントタイプは、AWS Glue が Salesforce Marketing Cloud Account Engagement と通信してデータへのアクセスをリクエストする方法を決定します。選択すると、接続を作成する前に満たす必要がある要件に影響します。Salesforce Marketing Cloud Account Engagement は、OAuth 2.0 の AUTHORIZATION_CODE のグラントタイプのみをサポートします。

- このグラントタイプは、ユーザーを認証するためにサードパーティーの認証サーバーにユーザーをリダイレクトすることから、「three-legged」の OAuth と見なされます。AWS Glue コンソール経由で接続を作成するときに使用されます。
- ユーザーは、Salesforce Marketing Cloud Account Engagement で独自の接続アプリを作成し、AWS Glue コンソールを介して接続を作成するときに独自のクライアント ID とクライアントシークレットを提供することを選択できます。このシナリオでは、引き続き Salesforce Marketing Cloud Account Engagement にリダイレクトされてログインし、リソースへアクセスするために AWS Glue を承認します。
- このグラントタイプは、更新トークンとアクセストークンになります。アクセストークンの有効期間は短く、更新トークンを使用してユーザーとやり取りすることなく自動的に更新される場合があります。
- 認可コード OAuth フロー用の接続アプリケーションの作成に関する Salesforce Marketing Cloud Account Engagement の公開ドキュメントについては、「認証」を参照してください。

Salesforce Marketing Cloud Account Engagement 接続を設定するには:

1. AWS Glue Glue Studio で、以下の手順に従って [データ接続] の下に接続を作成します。
   a. [接続タイプ] を選択するときは、[Salesforce Marketing Cloud アカウントエンゲージメント] を選択します。
   b. 接続先の Salesforce Marketing Cloud Account Engagement インスタンス の `INSTANCE_URL` を指定します。
   c. 接続先の Salesforce Marketing Cloud Account Engagement インスタンス の `PARDOT_BUSINESS_UNIT_ID` を指定します。
   d. ドロップダウンから適切な [認可コード URL] を選択します。
   e. ドロップダウンから適切な [トークン URL] を選択します。
   f. 次のアクションを実行でき、AWS Glue がその権限を持つ AWS IAM ロールを選択します。

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

```
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:DescribeSecret",
                "secretsmanager:GetSecretValue",
                "secretsmanager:PutSecretValue",
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterface",
                "ec2:DeleteNetworkInterface",
            ],
            "Resource": "*"
        }
    ]
}
```

g. ユーザーマネージドクライアントアプリケーションクライアント ID (接続されたアプリのクライアント ID) を指定します。

h. AWS Glue でこの接続に使用する `secretName` を選択して、トークンを配置します。選択したシークレットには、接続されたアプリのクライアントシークレットである値を持つ `USER_MANAGED_CLIENT_APPLICATION_CLIENT_SECRET` キーが必要です。

i. ネットワークを使用する場合は、ネットワークオプションを選択します。

2. AWS Glue ジョブに関連付けられている IAM ロールに `secretName` を読み取るアクセス許可を付与します。

3. AWS Glue ジョブ設定で、[追加のネットワーク接続] として `connectionName` を指定します。

Salesforce Marketing Cloud Account Engagement エンティティからの読み取り

前提条件

読み取り元の Salesforce Marketing Cloud Account Engagement オブジェクト。オブジェクト名が必要になります。

Sync ソースに対応するエンティティ:

エンティティ	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
キャンペーン	はい	あり	あり	あり	はい
動的コンテンツ	はい	あり	あり	あり	はい
E メール	はい	あり	あり	あり	はい
Email Template	はい	あり	あり	あり	はい
Engagement Studio Program	はい	あり	あり	あり	はい
Folder Contents	はい	あり	あり	あり	はい
Landing Page	はい	あり	あり	あり	はい
Lifecycle History	はい	あり	あり	あり	はい
Lifecycle Stage	はい	あり	あり	あり	はい
リスト	はい	あり	あり	あり	はい
List Email	はい	あり	あり	あり	はい
List Membership	はい	あり	あり	あり	はい
機会	はい	あり	あり	あり	はい
Prospect	はい	あり	あり	あり	はい

エンティティ	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
Prospect Account	はい	あり	あり	あり	はい
ユーザー	はい	あり	あり	あり	はい

例:

```
salesforcepardot_read = glueContext.create_dynamic_frame.from_options(
    connection_type="SalesforcePardot",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "entityName",
        "API_VERSION": "v5"
    }
   )
```

Async ソースに対応するエンティティ:

エンティティ	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
キャンペーン	はい	いいえ	いいえ	あり	いいえ
動的コンテンツ	はい	いいえ	いいえ	あり	いいえ
Email Template	はい	いいえ	いいえ	あり	いいえ
Landing Page	はい	いいえ	いいえ	あり	いいえ
Lifecycle History	はい	いいえ	いいえ	あり	いいえ

エンティティ	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
Lifecycle Stage	はい	いいえ	いいえ	あり	いいえ
リスト	はい	いいえ	いいえ	あり	いいえ
List Email	はい	いいえ	いいえ	あり	いいえ
List Membership	はい	いいえ	いいえ	あり	いいえ
機会	はい	いいえ	いいえ	あり	いいえ
Prospect	はい	いいえ	いいえ	あり	いいえ
Prospect Account	はい	いいえ	いいえ	あり	いいえ
ユーザー	はい	いいえ	いいえ	あり	いいえ

例:

```
salesforcepardot_read = glueContext.create_dynamic_frame.from_options(
    connection_type="SalesforcePardot",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "entityName",
        "API_VERSION": "v5",
        "TRANSFER_MODE": "ASYNC"
    }
   )
```

Salesforce Marketing Cloud Account Engagement エンティティとフィールドの詳細:

次のエンティティのフィールドの詳細を表示するには、Salesforce Marketing Cloud Account Engagement API に移動し、[ガイド] を選択し、[オープンソース API ラッパー] まで下にスクロールし、メニューから [バージョン 5 ドキュメント] を展開してエンティティを選択します。

エンティティリスト:

- キャンペーン
- 動的コンテンツ
- E メール
- Email Template
- Engagement Studio Program
- Folder Content
- Landing Page
- Lifecycle History
- Lifecycle Stage
- リスト
- List Email
- List Membership
- 機会
- Prospect
- Prospect Account
- ユーザー

上記のフィールドに加えて、Async モードは、以下の表に示すように、各エンティティの特定のフィルタリング可能なフィールドをサポートします。

エンティティ	Async でサポートされている追加のフィルタリング可能なフィールド
キャンペーン	`createdAfter` , `createdBefore` , `deleted`, `updatedAfter` , `updatedBefore`
動的コンテンツ	`createdAfter` , `createdBefore` , `deleted`, `updatedAfter` , `updatedBefore`

エンティティ	Async でサポートされている追加のフィルタリング可能なフィールド
Email Template	`createdAfter` , `createdBefore` , `deleted`, `updatedAfter` , `updatedBefore`
Engagement Studio Program	-
Landing Page	`createdAfter` , `createdBefore` , `deleted`, `updatedAfter` , `updatedBefore`
Lifecycle History	`createdAfter` , `createdBefore`
Lifecycle Stage	`createdAfter` , `createdBefore` , `deleted`, `updatedAfter` , `updatedBefore`
リスト	`createdAfter` , `createdBefore` , `deleted`, `updatedAfter` , `updatedBefore`
List Email	`createdAfter` , `createdBefore` , `deleted`, `updatedAfter` , `updatedBefore`
List Membership	`createdAfter` , `createdBefore` , `deleted`, `updatedAfter` , `updatedBefore`
機会	`createdAfter` , `createdBefore` , `deleted`, `updatedAfter` , `updatedBefore`
Prospect	`createdAfter` , `createdBefore` , `deleted`, `updatedAfter` , `updatedBefore`

エンティティ	Async でサポートされている追加のフィルタリング可能なフィールド
Prospect Account	`createdAfter` , `createdBefore` , `deleted`
ユーザー	`createdAfter` , `createdBefore` , `deleted`, `updatedAfter` , `updatedBefore`

追加フィールドの詳細については、「Salesforce Export API」を参照してください

コネクタに関する以下の考慮事項に注意してください:

- エンティティの `delete`フィールドの値は、`false` (デフォルト)、`true`、または `all` です。

パーティショニングクエリ

フィルターベースのパーティション分割:

Spark で同時実行を使用する場合は、追加の Spark オプション `PARTITION_FIELD`、`LOWER_BOUND`、`UPPER_BOUND`、および `NUM_PARTITIONS` を指定できます。これらのパラメータを使用すると、元のクエリは Spark タスクで同時に実行できるサブクエリの `NUM_PARTITIONS` の数に分割されます。

- `PARTITION_FIELD`: クエリのパーティション化に使用するフィールドの名前。
- `LOWER_BOUND`: 選択したパーティションフィールドの包括的な下限値。

  Datetime フィールドでは、SPark SQL クエリで使用される Spark タイムスタンプ形式を受け入れます。

  有効な値の例は次のとおりです:

  ```
  "2022-01-01T01:01:01.000Z"
  ```

- `UPPER_BOUND`: 選択したパーティションフィールドの排他的上限値。
- `NUM_PARTITIONS`: パーティション数。

- PARTITION_BY: 実行されるパーティション分割のタイプ。フィールドベースのパーティショニングの場合、「FIELD」が渡されます。

例:

```
salesforcepardot_read = glueContext.create_dynamic_frame.from_options(
    connection_type="salesforcepardot",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "entityName",
        "API_VERSION": "v5",
        "PARTITION_FIELD": "createdAt"
        "LOWER_BOUND": "2022-01-01T01:01:01.000Z"
        "UPPER_BOUND": "2024-01-01T01:01:01.000Z"
        "NUM_PARTITIONS": "10",
        "PARTITION_BY": "FIELD"
    }
   )
```

Salesforce Marketing Cloud Account Engagement 接続オプション

Salesforce Marketing Cloud Account Engagement の接続オプションは次のとおりです。

- ENTITY_NAME (文字列) – (必須) 読み取りに使用されます。Salesforce Marketing Cloud Account Engagement のオブジェクトの名前。
- PARDOT_BUSINESS_UNIT_ID – (必須) 接続の作成に使用されます。接続先の Salesforce Marketing Cloud Account Engagement インスタンスのビジネスユニット ID。
- API_VERSION (文字列) – (必須) 読み取りに使用されます。使用する Salesforce Marketing Cloud Account Engagement Rest API バージョン。
- SELECTED_FIELDS(List<String>) – Default: empty(SELECT *). 読み込みに使用されます。オブジェクトに選択する列。
- FILTER_PREDICATE(文字列) -
  - Sync モード - デフォルト: 空。読み込みに使用されます。Spark SQL 形式である必要があります。
  - Async モード - デフォルト: 現在の DateTime 値 (ユーザーのタイムゾーンに基づく) - 1 年。読み込みに使用されます。
- QUERY(文字列) – デフォルト: 空 読み込みに使用されます。完全な Spark SQL クエリ。

- PARTITION_FIELD(文字列) – 読み取りに使用されます。クエリをパーティション化するために使用するフィールド。
- LOWER_BOUND(文字列) - 読み取りに使用されます。選択したパーティションフィールドの包括的な下限値。
- UPPER_BOUND (文字列) - 読み取りに使用されます。選択したパーティションフィールドの排他的上限値。
- NUM_PARTITIONS(整数) – デフォルト: 1。読み込みに使用されます。読み取り用のパーティションの数。
- INSTANCE_URL (文字列) – (必須) 読み取りに使用されます。有効な Salesforce Marketing Cloud Account Engagement インスタンス URL。
- PARTITION_BY (文字列) – (必須) 読み取りに使用されます。実行されるパーティション分割のタイプ。フィールドベースのパーティショニングの場合、「FIELD」が渡されます。
- TRANSFER_MODE(文字列) - (オプション) このオプションが指定されていないジョブが SYNC モードで実行される場合、ASYNC モードでジョブを実行するために使用される値。

Salesforce Marketing Cloud Account Engagement コネクタの制限と注意事項

次の注意事項と制限が適用されます:

- 制限とパーティション分割の両方が適用される場合、制限はパーティション分割よりも優先されます。
- API ドキュメントに従って、SalesforceMarketingCloudEngagement が日次リクエストと同時リクエストに RateLimit を適用します。詳細については、「Rate Limits」を参照してください。
- Export API には、日次 Account Engagement API コール制限と、アカウントの同時 Account Engagement API コール制限が適用されます。
- キューと同様に、Export/Async API コールはアカウントごとに順番に実行されます。古いエクスポートは、新しいエクスポートの前に処理されます。
- パーティションは Async モードではサポートされていません。
- Export/Async API コールで指定された選択したフィールドの数は 150 を超えることはできません。
- [Prospect] エンティティは 150 を超えるフィールドをサポートしていますが、一度に選択できるフィールドは 150 のみです。Select All を選択した場合、一部のフィールドは除外されます。これらの除外されたフィールドのデータを取得するには、Selected Fields オプションに含める必要があります。

除外されたフィールドのリストを次に示します SELECT_ALL - updatedBy.firstName、updatedBy.lastName、updatedBy.jobTitle、updatedBy.roleName、

- 収集フィールドを Async にエクスポートすることはできません。例えば、List Email では、senderOptions および replyToOptions フィールドはサポートされていません。
- すべてのエンティティで、フィルターは必須です。フィルターが指定されていない場合、デフォルトのフィルター述語は、現在の日時 (タイムゾーンに調整) から 1 年を引いた値で Created After フィールドに設定されます。
- Salesforce Marketing Cloud Account Engagement の制限に従って、Async では、データを取得する最大範囲は 1 年です。1 年以上の期間を対象とするクエリが提供された場合、ジョブはエラーをスローします。
- 現在、Salesforce Pardot にバグがあります。データがないフィールドがジョブに 1 つだけ含まれている場合、フィールド値は正しい結果を返さず、代わりにフィールド名が複数回返されます。Salesforce Pardot チームはこの問題を認識しており、解決に向けて積極的に取り組んでいます。

AWS Glue Studio での SAP HANA に対する接続

AWS Glue は、SAP HANA のための組み込みサポートを提供します。AWS Glue Studio は、SAP HANA に接続してデータ統合ジョブをオーサリングし、AWS Glue Studio サーバーレス Spark ランタイム上でそれらのジョブを実行するためのビジュアルインターフェイスを提供します。

AWS Glue Studio は SAP HANA の統合接続を作成します。詳細については、「考慮事項」を参照してください。

トピック

- SAP HANA 接続の作成
- SAP HANA ソースノードの作成
- SAP HANA ターゲットノードを作成する
- 詳細オプション

SAP HANA 接続の作成

AWS Glue から SAP HANA に接続するには、SAP HANA 認証情報を作成して AWS Secrets Manager シークレットに保存し、そのシークレットを SAP HANA AWS Glue 接続に関連付ける必

要があります。SAP HANA サービスと AWS Glue の間のネットワーク接続を設定する必要があります。

前提条件:

- SAP HANA サービスが Amazon VPC 内にある場合は、トラフィックがパブリックインターネットを経由することなく、AWS Glue ジョブが SAP HANA サービスと通信できるように Amazon VPC を設定します。

  Amazon VPC で、AWS Glue がジョブの実行中に使用する [VPC]、[サブネット]、および [セキュリティグループ] を特定または作成します。さらに、SAP HANA エンドポイントとこの場所の間のネットワークトラフィックを許可するように Amazon VPC が設定されているようにする必要があります。ジョブでは、SAP HANA JDBC ポートとの TCP 接続を確立する必要があります。SAP HANA ポートの詳細については、SAP HANA のドキュメントを参照してください。ネットワークレイアウトに基づいて、セキュリティグループルール、ネットワーク ACL、NAT ゲートウェイ、およびピアリング接続の変更が必要になる場合があります。

SAP HANA に対する接続を設定するには:

1. AWS Secrets Manager で、SAP HANA 認証情報を使用してシークレットを作成します。Secrets Manager でシークレットを作成するには、AWS Secrets Manager ドキュメントの「AWS Secrets Manager シークレットを作成する」にあるチュートリアルに従ってください。シークレットを作成したら、次のステップのためにシークレット名 *secretName* を保存しておきます。

   - [key/value ペア] を選択する際に、*saphanaUsername* という値を持つキー `user` のペアを作成します。
   - [key/value ペア] を選択する際に、*saphanaPassword* という値を持つキー `password` のペアを作成します。
2. AWS Glue コンソールで、「the section called “AWS Glue 接続の追加”」にあるステップに従って接続を作成します。接続を作成したら、将来的に AWS Glue で使用するために、接続名 *connectionName* を維持します。

   - [接続タイプ] を選択する際に、[SAP HANA] を選択します。
   - [SAP HANA URL] を入力する場合は、インスタンスの URL を入力します。

SAP HANA JDBC URL の形式は
`jdbc:sap://saphanaHostname:saphanaPort/?databaseName=saphanaDBname,Paramete`
です

AWS Glue には次の JDBC URL パラメータが必要です。

- `databaseName` – 接続先の SAP HANA のデフォルトデータベース。

- [AWS Secret] をクリックして、*secretName* を入力します。

AWS Glue SAP HANA 接続を作成した後、AWS Glue ジョブを実行する前に次のステップを実行する必要があります。

- AWS Glue ジョブに関連付けられている IAM ロールに *secretName* を読み取るアクセス許可を付与します。

SAP HANA ソースノードの作成

必要な前提条件

- 前のセクション the section called “SAP HANA 接続の作成” で説明したように、AWS Secrets Manager シークレットを使用して設定された AWS Glue SAP HANA 接続。
- 接続で使用されるシークレットを読み取るためのジョブに対する適切なアクセス許可。
- 読み取り元とする SAP HANA テーブル、*tableName*、またはクエリ *targetQuery*。

  テーブルは、SAP HANA テーブル名とスキーマ名 (形式: *schemaName*.*tableName*) を使用して指定できます。テーブルがデフォルトのスキーマ「public」にある場合、スキーマ名と「.」区切り文字は必要ありません。この *tableIdentifier* を呼び出します。データベースは `connectionName` の JDBC URL パラメータとして指定されることに注意してください。

SAP HANA データソースの追加

[データソース — SAP HANA] ノードを追加するには:

1. SAP HANA データソース用の接続を選択します。すでに作成したので、ドロップダウンに表示されているはずです。接続を作成する必要がある場合は、[SAP HANA 接続を作成] を選択します。詳細については、前の「the section called “SAP HANA 接続の作成”」セクションを参照してください。

接続を選択したあとは、[プロパティを表示] をクリックすると、接続のプロパティを表示できます。

2. [SAP HANA ソース] のオプションを選択します。

   - [単一のテーブルを選択] – 単一のテーブルからすべてのデータにアクセスします。
   - [カスタムクエリを入力] — カスタムクエリに基づいて、複数のテーブルからデータセットにアクセスします。

3. 単一のテーブルを選択した場合は、*tableName* を入力します。

   [カスタムクエリを入力] を選択した場合は、SQL SELECT クエリを入力します。

4. [SAP HANA のカスタムプロパティ] で、必要に応じてパラメータと値を入力します。

SAP HANA ターゲットノードを作成する

必要な前提条件

- 前のセクション the section called “SAP HANA 接続の作成” で説明したように、AWS Secrets Manager シークレットを使用して設定された AWS Glue SAP HANA 接続。
- 接続で使用されるシークレットを読み取るためのジョブに対する適切なアクセス許可。
- 書き込み先とする SAP HANA テーブルである *tableName*。

  テーブルは、SAP HANA テーブル名とスキーマ名 (形式: *schemaName*.*tableName*) を使用して指定できます。テーブルがデフォルトのスキーマ「public」にある場合、スキーマ名と「.」区切り文字は必要ありません。この *tableIdentifier* を呼び出します。データベースは `connectionName` の JDBC URL パラメータとして指定されることに注意してください。

SAP HANA データターゲットの追加

[データターゲット — SAP HANA] ノードを追加するには:

1. SAP HANA データソース用の接続を選択します。すでに作成したので、ドロップダウンに表示されているはずです。接続を作成する必要がある場合は、[SAP HANA 接続を作成] を選択します。詳細については、前の「the section called “SAP HANA 接続の作成”」セクションを参照してください。

   接続を選択したあとは、[プロパティを表示] をクリックすると、接続のプロパティを表示できます。

2. *tableName* を指定して [テーブル名] を設定します。
3. [Teradata のカスタムプロパティ] で、必要に応じてパラメータと値を入力します。

詳細オプション

SAP HANA ノードを作成する際に、高度なオプションを指定できます。これらのオプションは Spark AWS Glue スクリプトのプログラミング時に使用できるオプションと同じです。

「the section called “SAP HANA 接続”」を参照してください。

SAP OData への接続

SAP OData は、ABAP (Advanced Business Application Programming) を使用して SAP に存在するデータをクエリおよび更新するために使用される標準のウェブプロトコルです。HTTP などのウェブテクノロジーを適用して構築し、さまざまな外部アプリケーション、プラットフォーム、デバイスからの情報へのアクセスを提供します。製品を使用すると、SAP システム、アプリケーション、またはデータとシームレスに統合するために必要なすべてにアクセスできます。

トピック

- AWS Glue による SAP OData のサポート
- 接続を作成する
- SAP OData ジョブの作成
- SAP OData 状態管理スクリプトの使用
- ODP 非対応エンティティのパーティション分割
- SAP OData 接続オプション
- SAP OData エンティティとフィールドの詳細

AWS Glue による SAP OData のサポート

AWS Glue は SAP OData を次のようにサポートします:

ソースとしてサポートされていますか?

はい。AWS Glue ETL ジョブを使用して、SAP OData からデータをクエリできます。

サポートされている SAP OData API バージョン

次の SAP OData API バージョンがサポートされています:

- 2.0

サポートされているソース

以下のソースがサポートされています。

- ODP (オペレーショナルデータプロビジョニング) ソース :
  - BW Extractor (DataSources)
  - CDS ビュー
  - SLT
- ODP 非対応ソース。例 :
  - CDS ビューサービス
  - RFC ベースのサービス
  - カスタム ABAP サービス

サポートされている SAP コンポーネント

以下に、最小要件を示します。

- サービス検出のためにカタログサービスを有効にする必要があります。
  - SAP システムの SAP Gateway で、抽出するオペレーショナルデータプロビジョニング (ODP) データソースを設定します。
  - OData V2.0: トランザクション `/IWFND/MAINT_SERVICE` を介して SAP Gateway で OData V2.0 カタログサービス (複数可) を有効にします。
  - トランザクション `/IWFND/MAINT_SERVICE` を介して SAP Gateway で OData V2.0 サービスを有効にします。
  - SAP OData サービスは、`$top` や `$skip` などのクライアント側のページ分割/クエリオプションをサポートしている必要があります。また、システムクエリオプション `$count` もサポートする必要があります。
  - ユーザーが SAP OData サービスを検索し、データ抽出を行えるようにするには、ユーザーに適切な権限を付与する必要があります。SAP が提供するセキュリティドキュメントを参照してください。
- OAuth 2.0 を認証メカニズムとして使用する場合は、OData サービスに対して OAuth 2.0 を有効にし、SAP ドキュメントに従って OAuth クライアントを登録する必要があります。

- ODP データソースに基づいて OData サービスを生成するには、SAP Gateway Foundation を ERP/BW スタックまたはハブの設定にローカルにインストールする必要があります。
  - ERP/BW アプリケーションの場合、SAP NetWeaver AS ABAP スタックは 7.50 SP02 以上である必要があります。
  - ハブシステム (SAP Gateway) の場合、リモートハブのセットアップには、ハブシステムの SAP NetWeaver AS ABAP が 7.50 SP01 以上である必要があります。
- ODP 非対応ソースの場合、SAP NetWeaver スタックのバージョンは 7.40 SP02 以降である必要があります。

サポートされた認証方法

次の認証方法がサポートされています。

- 基本認証
- OAuth 2.0

前提条件

SAP OData 接続を使用して SAP OData からデータ抽出する AWS Glue ジョブを開始する前に、次の前提条件を満たしてください。

- 関連する SAP OData Service を SAP システムでアクティブ化し、データソースを使用できることを確認する必要があります。OData サービスがアクティブ化されていない場合、 Glue ジョブは SAP からデータにアクセスまたは抽出できません。
- AWS Glue ジョブが SAP OData サービスとの接続を正常に確立できるように、基本 (カスタム) 認証や OAuth 2.0 などの適切な認証メカニズムを SAP で設定する必要があります。
- IAM ポリシーを設定して、SAP、Secrets Manager、およびプロセスに関連するその他の AWS リソースにアクセスするための適切なアクセス許可を AWS Glue ジョブに付与します。
- SAP システムがプライベートネットワーク内でホストされている場合、AWS Glue ジョブがパブリックインターネット経由で機密データを公開することなく SAP と安全に通信できるように VPC 接続を設定する必要があります。

AWS Secrets Manager を使用して、AWS Glue ジョブが実行時に動的に取得できる SAP 認証情報などの機密情報を安全に保存できます。このアプローチにより、認証情報をハードコードする必要がなくなり、セキュリティと柔軟性が向上します。

以下の前提条件は、 AWS Glue と SAP OData をスムーズに統合するために各コンポーネントを設定する方法に関するステップごとのガイダンスを提供します。

トピック

- SAP OData のアクティベーション
- IAM ポリシー
- 接続 / VPC 接続
- SAP 認証
- 認証シークレットを保存する AWS Secrets Manager

SAP OData のアクティベーション

SAP OData 接続には、次の手順を実行します。

ODP ソース

ODP プロバイダーからデータを転送する前に、次の要件を満たす必要があります。

- お客様のSAP NetWeaver AS ABAP インスタンスには、
- データ転送元の ODP プロバイダーが含まれています。ODP プロバイダーには以下が含まれます。
  - SAP DataSources (トランザクションコード RSO2)
  - SAP Core Data Services ABAP CDS ビュー
  - SAP BW または SAP BW/4HANA システム (InfoObject、DataStore Object)
  - SAP Landscape Replication Server (SAP SLT) を介した SAP ソースシステムからのテーブルと DB ビューのリアルタイムレプリケーション
  - SAP ABAP ベースのソースの SAP HANA Information ビュー
- SAP NetWeaver インスタンスには SAP Gateway Foundation コンポーネントが含まれています。
- ODP プロバイダーからデータを抽出する OData サービスを作成しました。OData サービスを作成するには、SAP Gateway Service Builder を使用します。ODP データにアクセスするには、Amazon AppFlow が OData API を使用してこのサービスを呼び出します。詳細については、SAP BW/4HANA ドキュメントの「Generating a Service for Extracting ODP Data via OData」を参照してください。
- ODP データソースに基づいて OData サービスを生成するには、SAP Gateway Foundation を ERP/BW スタックまたはハブの設定にローカルにインストールする必要があります。

- ERP/BW アプリケーションの場合、SAP NetWeaver AS ABAP スタックは 7.50 SP02 以上である必要があります。
- ハブシステム (SAP Gateway) の場合、リモートハブのセットアップには、ハブシステムの SAP NetWeaver AS ABAP が 7.50 SP01 以上である必要があります。

ODP 非対応ソース

- SAP NetWeaver スタックのバージョンは 7.40 SP02 以降である必要があります。
- サービス検出のためにカタログサービスを有効にする必要があります。
  - OData V2.0: OData V2.0 カタログサービスは、トランザクション `/IWFND/MAINT_SERVICE` を介して SAP ゲートウェイで有効にすることができます。
- SAP OData サービスは、`$top` や `$skip` などのクライアント側のページ分割/クエリオプションをサポートしている必要があります。また、システムクエリオプション `$count` もサポートする必要があります。
- OAuth 2.0 の場合、OData サービスで OAuth 2.0 を有効にし、SAP ドキュメントに従って OAuth クライアントを登録し、承認されたリダイレクト URL を次のように設定する必要があります。
  - `https://<region>.console.aws.amazon.com/gluestudio/oauth`、 AWS Glue を `<region>` が実行されているリージョン (例：us-east-1) に置き換えます。
  - HTTPS 経由で接続するには、安全なセットアップを有効にする必要があります。
- ユーザーが SAP OData サービスを検索し、データ抽出を行えるようにするには、ユーザーに適切な権限を付与する必要があります。SAP が提供するセキュリティドキュメントを参照してください。

IAM ポリシー

接続を作成および使用するための API オペレーションを含むポリシー

次のサンプル ポリシーで、接続の作成と使用に必要な AWS IAM 権限について説明します。新しいロールを作成する場合は、以下を含むポリシーを作成します。

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

```
                "glue:ListConnectionTypes",
                "glue:DescribeConnectionType",
                "glue:RefreshOAuth2Tokens"
                "glue:ListEntities",
                "glue:DescribeEntity"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:DescribeSecret",
                "secretsmanager:GetSecretValue",
                "secretsmanager:PutSecretValue"
            ],
            "Resource": "*"
         }
    ]
}
```

ロールは、Amazon S3 など、ジョブで使用されるすべてのリソースへのアクセスを許可する必要があります。上記の方法を使用しない場合は、代わりに次のマネージド IAM ポリシーを使用します:

- AWSGlueServiceRole – さまざまな AWS Glue プロセスを実行するために必要なリソースへのアクセス権をユーザーに代わって付与します。これらのリソースには AWS Glue 、Amazon S3、IAM、CloudWatch Logs、および Amazon EC2 が含まれます。このポリシーで指定されたリソースの命名規則に従った場合、AWS Glue プロセスは必要なアクセス権限を使用できます。このポリシーは、通常、クローラ、ジョブ、開発エンドポイントを定義するときに指定されたロールにアタッチされます。
- AWSGlueConsoleFullAccess — ポリシーがアタッチされているアイデンティティが AWS マネージメントコンソールを使用するときは、AWS Glue リソースへのフルアクセスを許可します。このポリシーで指定されたリソースの命名規則に従った場合、ユーザーは完全なコンソール機能を使用できます。このポリシーは、通常 AWS Glue コンソールのユーザーにアタッチされています。
- SecretsManagerReadWrite — AWS マネジメントコンソール経由で AWS Secrets Manager への読み取り/書き込みアクセスを提供します。注: これには IAM アクションは含まれないため、ローテーション設定が必要な場合は `IAMFullAccess` と組み合わせてください。

VPC の設定に必要な IAM ポリシー/アクセス許可

AWS Glue 接続を作成するために VPC 接続を使用する場合は、次の IAM アクセス許可が必要です。詳細については、「AWS Glue の IAM ポリシーを作成する」を参照してください。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateNetworkInterface",
                "ec2:DeleteNetworkInterface",
                "ec2:DescribeNetworkInterfaces",
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

接続 / VPC 接続

VPC 接続の手順:

1. 既存の VPC 接続を使用するか、Amazon VPC ドキュメントに従って新しい接続を作成します。
2. トラフィックをインターネットにルーティングする NAT Gateway があることを確認します。
3. 接続を作成するには、VPC エンドポイントを Amazon S3 Gateway として選択します。
4. AWS が提供する DNS サービスを使用するには、DNS 解決と DNS ホスト名を有効にします。
5. 作成した VPC に移動し、STS、AWS Glue、シークレットマネージャーなどのさまざまなサービスに必要なエンドポイントを追加します。
   a. [Create Endpoint] (エンドポイントの作成) を選択します。
   b. [サービスカテゴリ] では、[AWS サービス] を選択する。
   c. [Service Name] で、接続先のサービスを選択します。
   d. VPC を選択し、DNS 名を有効にします。
   e. VPC 接続に必要な VCP エンドポイント:
      i. STS
      ii. AWS Glue
      iii. シークレットマネージャー

セキュリティグループの設定

AWS Glue がセキュリティ グループに接続できるようにするには、セキュリティ グループで AWS Glue VPC からのリスニング ポートへのトラフィックを許可する必要があります。ソース IP アドレスの範囲をできるだけ制限することをお勧めします。

AWS Glue には、それ自体からのすべてのインバウンドトラフィックを許可する特別なセキュリティグループが必要です。セキュリティグループから発信されるすべてのトラフィックを許可する自己参照ルールを作成できます。既存のセキュリティグループを変更し、ソースとしてセキュリティグループを指定できます。

URL エンドポイント (NLB または SAP インスタンス) の HTTPS ポートから通信を開きます。

接続オプション

- 内部および外部 NLB との HTTPS 接続、認証局 (CA) からの SSL 証明書 (自己署名 SSL 証明書ではない)
- 自己署名 SSL 証明書ではなく、認証局 (CA) からの SAP インスタンス SSL 証明書を使用した HTTPS 接続

SAP 認証

SAP コネクタは、CUSTOM (SAP BASIC 認証) と OAUTH 認証の両方の方法をサポートしています。

カスタム認証

AWS Glue は、SAP システムへの接続を確立するための方法としてカスタム (基本認証) をサポートし、ユーザー名とパスワードを使用して安全なアクセスを可能にします。この認証タイプは、SAP OData インスタンス内の特定のユーザーのアクセス許可でユーザー名とパスワードを事前に使用できるため、自動化シナリオに適しています。AWS Glue はユーザー名とパスワードを使用して SAP OData APIs を認証できます。AWS Glue では、基本認可はカスタム認可として実装されます。

基本的な認証フローに関する SAP OData のパブリックドキュメントについては、「HTTP Basic Authentication」を参照してください。

OAuth 2.0 認証

AWS Glue は、SAP システムへの接続を確立するための安全な認証メカニズムとして OAuth 2.0 もサポートしています。これにより、最新の認証標準への準拠を確保し、データアクセスのセキュリティを強化しながら、シームレスな統合が可能になります。

AUTHORIZATION_CODE のグラントタイプ

グラントタイプは、AWS Glue が SAP OData と通信してデータへのアクセスをリクエストする方法を決定します。SAP OData は、AUTHORIZATION_CODE 許可タイプのみをサポートします。このグラントタイプは、ユーザーを認証するためにサードパーティーの認証サーバーにユーザーをリダイレクトすることに依存するため、「three-legged」の OAuth と見なされます。AWS Glue コンソール経由で接続を作成するときに使用されます。

ユーザーは、SAP OData で独自の接続アプリを作成し、AWS Glue コンソールを介して接続を作成するときに独自のクライアント ID とクライアントシークレットを指定することを選択できます。このシナリオでは、引き続き SAP OData にリダイレクトされてログインし、リソースへアクセスするために AWS Glue を承認します。

このグラントタイプは、更新トークンとアクセストークンになります。アクセストークンの有効期間は短く、更新トークンを使用してユーザーとやり取りすることなく自動的に更新される場合があります。

認可コード OAuth フロー用の接続アプリの作成に関する SAP OData の公開ドキュメントについては、「Authentication Using OAuth 2.0」を参照してください。

認証シークレットを保存する AWS Secrets Manager

SAP OData 接続シークレットを AWS Secrets Manager に保存し、 IAM ポリシー セクションで指定されているように取得に必要なアクセス許可を設定し、接続の作成時に使用する必要があります。

AWS Secrets Manager の AWS マネジメントコンソールを使用して、SAP ソースのシークレットを作成します。詳細については、「AWS Secrets Manager シークレットを作成する」を参照してください。AWS Secrets Manager の詳細には、次のコードの要素を含める必要があります。

カスタム認証シークレット

<SAP ユーザー名> の代わりに SAP システムのユーザー名を入力し、<SAP ユーザー名のパスワード> の代わりにパスワードを入力して、True または False を入力する必要があります。このコンテキストでは、 basicAuthDisableSSO を true に設定すると、基本認証リクエストのシングルサインオン (SSO) が無効になり、リクエストごとに明示的なユーザー認証情報が必要になります。逆に、それを false に設定すると、利用可能な既存の SSO セッションを使用できるようになります。

```
{
   "basicAuthUsername": "<your SAP username>",
   "basicAuthPassword": "<your SAP username password>",
```

```
   "basicAuthDisableSSO": "<True/False>",
   "customAuthenticationType": "CustomBasicAuth"
}
```

OAuth 2.0 シークレット

認証メカニズムとして OAuth 2.0 を使用している場合、AWS Secrets Manager のシークレットには、次の形式の ユーザーマネージドクライアントアプリケーション ClientId が必要です。< クライアントシークレット> の代わりに SAP クライアントシークレットを入力する必要があります。

```
{"USER_MANAGED_CLIENT_APPLICATION_CLIENT_SECRET": "<your client secret>"
}
```

接続を作成する

SAP OData 接続を設定するには

1. AWS マネジメントコンソールにサインインし、AWS Glue コンソールを開きます。AWS Glue Studio で、以下の手順に従って接続を作成します。
   a. 左側のパネルでデータ接続をクリックします。
   b. [接続を作成] をクリックします。
   c. [データソースを選択] で [SAP OData] を選択します。
   d. 接続先の SAP OData インスタンスの [アプリケーションホスト URL] を指定します。このアプリケーションホスト URL は、VPC 以外の接続ではパブリックインターネット経由でアクセス可能である必要があります。
   e. 接続先の SAP OData インスタンスの [アプリケーションサービスパス] を指定します。これはカタログサービスのパスと同じです。例: `/sap/opu/odata/iwfnd/catalogservice;v=2`。 AWS Glue は特定のオブジェクトのパスを受け入れません。
   f. 接続先の SAP OData インスタンスの [クライアント番号] を指定します。指定できる値は、001 から 999 です。例 : 010
   g. 接続先の SAP OData インスタンスの [ポート番号] を指定します。例: 443
   h. 接続先の SAP OData インスタンスの [ログオン言語] を指定します。EN の例
   i. 「IAM ポリシー」セクションで説明しているように、AWS Glue が引き受けることができ、かつアクセス許可を持つ AWS IAM ロールを選択します。
   j. AWS Glue でこの接続に使用する [認証タイプ] をドロップダウンリストから選択してください。選択肢は OAUTH2 または CUSTOM です。

i. CUSTOM - 「認証シークレットを保存する AWS Secrets Manager」セクションで指定したとおりに作成したシークレットを選択します。

ii. OAUTH 2.0 - OAuth 2.0 を使用する場合のみ、以下の情報を入力します。

A. [ユーザーマネージドクライアントアプリケーション ClientId] に、クライアント ID を入力します。

B. 認証シークレットを保存する AWS Secrets Manager セクションで作成した AWS Secrets Manager の `USER_MANAGED_CLIENT_APPLICATION_CLIENT_SECRET` (クライアントシークレット)。

C. [認証コードの URL] に、認証コードの URL を入力します。

D. [認証トークン URL] に、認証トークンの URL を入力します。

E. [OAuth スコープ] で、OAuth スコープをスペースで区切って入力します。例：`/IWFND/SG_MED_CATALOG_0002 ZAPI_SALES_ORDER_SRV_0001`

k. ネットワークを使用する場合は、ネットワークオプションを選択します。詳細については、接続 / VPC 接続を参照してください。

2. AWS Glue ジョブに関連付けられている IAM ロールに `secretName` を読み取るアクセス許可を付与します。詳細については、「IAM ポリシー」を参照してください。

3. [接続のテスト] を選択し、接続をテストします。接続テストに合格したら、[次へ] をクリックし、接続名を入力して接続を保存します。ネットワークオプション (VPC) を選択した場合、テスト接続機能は使用できません。

SAP OData ジョブの作成

「AWS Glue Studio でビジュアル ETL ジョブを作成する」を参照してください。

オペレーショナルデータプロビジョニング (ODP) ソース

オペレーショナルデータプロビジョニング (ODP) は、さまざまなターゲットアプリケーションのデータ抽出とレプリケーションをサポートするために使用できる技術インフラストラクチャを提供し、これらのシナリオでデルタメカニズムをサポートします。デルタプロシージャの場合、ソース (ODP プロバイダー) からのデータは、更新プロセスを使用してデルタキュー (オペレーションデルタキュー – ODQ) に自動的に書き込まれるか、エクストラクタインターフェイスを使用してデルタキューに渡されます。ODP プロバイダーは、DataSource (エクストラクター) 、ABAP Core Data Services ビュー (ABAP CDS ビュー) 、SAP BW または SAP BW/4HANA、SAP Landscape Transformation Replication Server (SLT)、SAP HANA Information ビュー (計算ビュー) のいずれかで

す。ターゲットアプリケーション (ODQ の「サブスクライバー」またはより一般的には「ODP コンシューマー」と呼ばれる) は、デルタキューからデータを取得し、データの処理を続行します。

フルロード

SAP OData エンティティおよび ODP エンティティのコンテキストにおいて、フルロードとは、1 回のオペレーションで ODP エンティティから使用可能なすべてのデータを抽出するプロセスを指します。このオペレーションは、ソースシステムから完全なデータセットを取得し、ターゲットシステムにエンティティのデータの包括的かつ最新のコピーが確実に保持されます。通常、全ロードは、増分ロードをサポートしないソースや、ターゲットシステムの更新が必要な場合に使用されます。

例

DynamicFrame を作成するときに、明示的に ENABLE_CDC フラグを false に設定できます。注: ENABLE_CDC はデフォルトで false です。デルタキューを初期化しない場合は、このフラグを送信したり true に設定したりする必要はありません。このフラグを true に設定しないと、全ロード抽出になります。

```
sapodata_df = glueContext.create_dynamic_frame.from_options(
    connection_type="SAPOData",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "entityName",
        "ENABLE_CDC": "false"
    }, transformation_ctx=key)
```

増分ロード

ODP (オペレーショナルデータプロビジョニング) エンティティのコンテキストでの増分ロードでは、前回のデータ抽出以降にソースシステムから新規または変更されたデータ (デルタ) のみを抽出し、すでに処理されたレコードを前処理しないようにします。このアプローチにより、効率が大幅に向上し、データ転送量が削減され、パフォーマンスが向上し、システム間の効率的な同期が保証され、特に頻繁に変更される大規模なデータセットの場合、処理時間が最小限に抑えられます

デルタトークンベースの増分転送

変更データキャプチャ (CDC) を使用して、それをサポートする ODP 対応エンティティの増分転送を有効にするには、次の手順に従います。

1. スクリプトモードで増分転送ジョブを作成します。

2. DataFrame または Glue DynamicFrame を作成する際には、オプション "ENABLE_CDC": "True" を渡す必要があります。このオプションにより、SAP からデルタトークンを受信できます。このトークンは、変更後のデータの取得に使用できます。

データフレームの最後の行の DELTA_TOKEN 列に格納されます。このトークンは、後続の呼び出しでコネクタ オプションとして使用して、次のデータ セットを段階的に取得できます。

例

- DynamicFrameを作成する際に、ENABLE_CDC フラグを true に設定します。注: false はデフォルトで ENABLE_CDC です。デルタキューを初期化しない場合は、このフラグを送信したり true に設定したりする必要はありません。このフラグを true に設定しないと、全ロード抽出になります。

```
sapodata_df = glueContext.create_dynamic_frame.from_options(
    connection_type="SAPOData",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "entityName",
        "ENABLE_CDC": "true"
    }, transformation_ctx=key)

# Extract the delta token from the last row of the DELTA_TOKEN column
delta_token_1 = your_logic_to_extract_delta_token(sapodata_df) # e.g.,
 D20241029164449_000370000
```

- 抽出されたデルタトークンを新しいイベントを取得するためのオプションとして渡すことができます。

```
sapodata_df_2 = glueContext.create_dynamic_frame.from_options(
    connection_type="SAPOData",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "entityName",
        // passing the delta token retrieved in the last run
        "DELTA_TOKEN": delta_token_1
    } , transformation_ctx=key)

# Extract the new delta token for the next run
delta_token_2 = your_logic_to_extract_delta_token(sapodata_df_2)
```

DELTA_TOKEN が格納されている最後のレコードは、ソースからのトランザクションレコードではなく、デルタトークン値を渡す目的でのみ存在することに注意してください。

DELTA_TOKEN とは別に、データフレームの各行には次のフィールドが返されます。

- GLUE_FETCH_SQ: これは、レコードが受信された順序で EPOC タイムスタンプから生成されるシーケンスフィールドで、レコードごとに一意です。これは、ソースシステムの変更の順序を確認または確立する必要がある場合に使用できます。このフィールドは、ODP が有効なエンティティにのみ表示されます。
- DML_STATUS: これにより、ソースから新しく挿入および更新されたすべてのレコードに対して UPDATED が表示され、ソースから削除されたレコードに対して DELETED が表示されます。

状態を管理し、差分トークンを再利用して、変更したレコードを取得する方法の詳細については、「SAP OData 状態管理スクリプトの使用」セクションを参照してください。

デルタトークンの無効化

デルタトークンは、サービスコレクションとユーザーに関連付けられます。同じサービスコレクションとユーザーに対して “ENABLE_CDC” : “true” を有効にした新しい初期プルが開始されると、以前の初期化の結果として発行された以前のデルタトークンはすべて SAP OData サービスによって無効になります。有効期限が切れたデルタトークンを使用してコネクタを呼び出すと、例外が発生します。

```
Could not open data access via extraction API RODPS_REPL_ODP_OPEN
```

OData サービス (ODP 非対応ソース)

フルロード

ODP (オペレーショナルデータプロビジョニング) に対応していないシステムの場合、フルロードでは、ソース システムからデータセット全体を抽出し、ターゲットシステムにロードします。ODP に対応していないシステムは、差分などの高度なデータ抽出メカニズムを本質的にサポートしないため、そのプロセスは単純ですが、データのサイズによってはリソースを大量に消費する可能性があります。

増分ロード

ODP (オペレーショナルデータプロビジョニング) をサポートしていないシステムまたはエンティティの場合、変更を追跡および抽出をするためのタイムスタンプベースの仕組みを実装することで、増分データ転送を手動で管理できます。

タイムスタンプベースの増分転送

ODP 非対応のエンティティの場合 (または ENABLE_CDC フラグを使用しない ODP が有効になっているエンティティの場合)、コネクタで filteringExpression オプションを使用して、データを取得する datetime 間隔を指定できます。この方法は、各レコードが最後に作成/変更された日時を表すデータ内のタイムスタンプフィールドに依存します。

例

2024-01-01T00:00:00.000 後に変更されたレコードの取得

```
sapodata_df = glueContext.create_dynamic_frame.from_options(
    connection_type="SAPOData",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "entityName",
        "filteringExpression": "LastChangeDateTime >= 2024-01-01T00:00:00.000"
    }, transformation_ctx=key)
```

注: この例では、LastChangeDateTime は各レコードが最後に変更された日時を表すフィールドです。実際のフィールド名は、特定の SAP OData エンティティによって異なる場合があります。

後続の実行でデータの新しいサブセットを取得するには、filteringExpression を新しいタイムスタンプで更新します。通常、これは以前に取得したデータの最大タイムスタンプ値になります。

例

```
max_timestamp = get_max_timestamp(sapodata_df)  # Function to get the max timestamp
 from the previous run
next_filtering_expression = f"LastChangeDateTime > {max_timestamp}"

# Use this next_filtering_expression in your next run
```

次のセクションでは、これらのタイムスタンプベースの増分転送を管理するための自動化されたアプローチを提供し、実行間でフィルタリング式を手動で更新する必要がなくなります。

SAP OData 状態管理スクリプトの使用

AWS Glue ジョブで SAP OData 状態管理スクリプトを使用するには、次の手順に従います:

- パブリック Amazon S3 バケットから状態管理スクリプトをダウンロードします。
- AWS Glue ジョブがアクセス許可を持つ Amazon S3 バケットにスクリプトをアップロードします。
- AWS Glue ジョブ内のスクリプトを参照する: AWS Glue ジョブを作成または更新するときは、Amazon S3 バケット内のスクリプトパスを参照する `'--extra-py-files'` オプションを渡します。例: `--extra-py-files s3://your-bucket/path/to/sap_odata_state_management.py`
- AWS Glue ジョブスクリプトで状態管理ライブラリをインポートして使用します。

デルタトークンベースの増分転送の例

以下は、デルタトークンベースの増分転送に 状態管理スクリプトを使用する方法の例です。

```
from sap_odata_state_management import StateManagerFactory, StateManagerType, StateType

# Initialize the state manager
state_manager = StateManagerFactory.create_manager(
    manager_type=StateManagerType.JOB_TAG,
    state_type=StateType.DELTA_TOKEN,
    options={
        "job_name": args['JOB_NAME'],
        "logger": logger
    }
)

# Get connector options (including delta token if available)
key = "SAPODataNode"
connector_options = state_manager.get_connector_options(key)

# Use the connector options in your Glue job
df = glueContext.create_dynamic_frame.from_options(
    connection_type="SAPOData",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "entityName",
        "ENABLE_CDC": "true",
        **connector_options
    }
)

# Process your data here...
```

```
# Update the state after processing
state_manager.update_state(key, sapodata_df.toDF())
```

タイムスタンプベースの増分転送の例

以下は、デルタトークンベースの増分転送に 状態管理スクリプトを使用する方法の例です。

```
from sap_odata_state_management import StateManagerFactory, StateManagerType, StateType

# Initialize the state manager
state_manager = StateManagerFactory.create_manager(
    manager_type=StateManagerType.JOB_TAG,
    state_type=StateType.DELTA_TOKEN,
    options={
        "job_name": args['JOB_NAME'],
        "logger": logger
    }
)

# Get connector options (including delta token if available)
key = "SAPODataNode"
connector_options = state_manager.get_connector_options(key)

# Use the connector options in your Glue job
df = glueContext.create_dynamic_frame.from_options(
    connection_type="SAPOData",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "entityName",
        "ENABLE_CDC": "true",
        **connector_options
    }
)

# Process your data here...

# Update the state after processing
state_manager.update_state(key, sapodata_df.toDF())
```

どちらの例でも、状態管理スクリプトはジョブ実行間の状態 (デルタトークンまたはタイムスタンプ) の保存の複雑さを処理します。コネクタオプションを取得するときに最後の既知の状態を自動的に取

得し、処理後に状態を更新して、各ジョブ実行が新規または変更されたデータのみを処理するようにします。

ODP 非対応エンティティのパーティション分割

Apache Spark では、パーティション分割とは、並列処理のためにクラスター内のワーカーノード間でデータを分割して分散する方法を指します。各パーティションは、タスクによって個別に処理できるデータの論理的なチャンクです。パーティショニングは、Spark の基本的な概念であり、パフォーマンス、スケーラビリティ、リソース使用率に直接影響します。 AWS Glue ジョブは、Spark のパーティショニングメカニズムを使用してデータセットを小さなチャンク (パーティション) に分割し、クラスターのワーカーノード間で並列に処理できます。パーティション分割は ODP エンティティには適用されないことに注意してください。

詳細については、「AWS Glue Spark と PySpark のジョブ」を参照してください。

前提条件

読み取り元の SAP OData オブジェクト。オブジェクト/EntitySet 名 が必要です。例: `/sap/opu/odata/sap/API_SALES_ORDER_SRV/A_SalesOrder`

例

```
sapodata_read = glueContext.create_dynamic_frame.from_options(
    connection_type="SAPOData",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "/sap/opu/odata/sap/API_SALES_ORDER_SRV/A_SalesOrder"
    }, transformation_ctx=key)
```

パーティショニングクエリ

フィールドベースのパーティション分割

Spark で同時実行を使用する場合は、追加の Spark オプション `PARTITION_FIELD`、`LOWER_BOUND`、`UPPER_BOUND`、および `NUM_PARTITIONS` を指定できます。これらのパラメータを使用すると、元のクエリは Spark タスクで同時に実行できるサブクエリの `NUM_PARTITIONS` の数に分割されます。整数、日付、および DateTime フィールドは、SAP OData コネクタのフィールドベースのパーティショニングをサポートします。

- `PARTITION_FIELD`: クエリのパーティション化に使用するフィールドの名前。
- `LOWER_BOUND`: 選択したパーティションフィールドの包括的な下限値。

データ型が DateTime であるフィールドでは、Spark SQL クエリで使用される Spark タイムスタンプ形式を受け入れます。

有効な値の例: "2000-01-01T00:00:00.000Z"

- UPPER_BOUND: 選択したパーティションフィールドの排他的上限値。
- NUM_PARTITIONS: パーティション数。
- PARTITION_BY: 実行するパーティショニングのタイプ。フィールドベースのパーティショニングの場合は、FIELD を渡します。

例

```
sapodata= glueContext.create_dynamic_frame.from_options(
    connection_type="sapodata",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "/sap/opu/odata/sap/SEPM_HCM_SCENARIO_SRV/EmployeeSet",
        "PARTITION_FIELD": "validStartDate"
        "LOWER_BOUND": "2000-01-01T00:00:00.000Z"
        "UPPER_BOUND": "2020-01-01T00:00:00.000Z"
        "NUM_PARTITIONS": "10",
        "PARTITION_BY": "FIELD"
    }, transformation_ctx=key)
```

レコードベースのパーティション分割

元のクエリは、元のクエリは Spark タスクで同時に実行できるサブクエリの NUM_PARTITIONS の数に分割されます。

レコードベースのパーティショニングは、ODP 以外のエンティティでのみサポートされます。ODP エンティティのページ分割は、次のトークン/スキップトークンを通じてサポートされるためです。

- PARTITION_BY: 実行されるパーティション分割のタイプ。レコードベースのパーティション分割の場合は、COUNT が渡されます。

例

```
sapodata= glueContext.create_dynamic_frame.from_options(
    connection_type="sapodata",
```

```
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "/sap/opu/odata/sap/SEPM_HCM_SCENARIO_SRV/EmployeeSet",
        "NUM_PARTITIONS": "10",
        "PARTITION_BY": "COUNT"
    }, transformation_ctx=key)
```

制限/コールアウト

- スキップトークン/デルタトークンでページネーションを行うため、ODP エンティティはレコードベースのパーティション分割と互換性がありません。したがって、レコードベースのパーティション分割の場合、maxConcurrency のデフォルト値は、ユーザー入力に関係なく「null」に設定されます。
- 制限とパーティション分割の両方が適用された場合、制限はパーティション分割よりも優先されます。

SAP OData 接続オプション

SAP OData の接続オプションは次のとおりです:

- ENTITY_NAME (文字列) – (必須) 読み取りに使用されます。SAP OData のオブジェクトの名前。

  例: /sap/opu/odata/sap/API_SALES_ORDER_SRV/A_SalesOrder
- API_VERSION(文字列) - (オプション) 読み取りに使用。使用する SAP OData Rest API バージョン。例: 2.0。
- SELECTED_FIELDS(List<String>) – Default: empty(SELECT *). 読み込みに使用されます。オブジェクトに選択する列。

  例: SalesOrder
- FILTER_PREDICATE(文字列) – デフォルト: 空 読み込みに使用されます。Spark SQL 形式である必要があります。

  例: `SalesOrder = "10"`
- QUERY(文字列) – デフォルト: 空 読み込みに使用されます。完全な Spark SQL クエリ。

  例: `SELECT * FROM /sap/opu/odata/sap/API_SALES_ORDER_SRV/A_SalesOrder`
- PARTITION_FIELD(文字列) – 読み取りに使用されます。クエリをパーティション化するために使用するフィールド。

例: ValidStartDate

- LOWER_BOUND(文字列) - 読み取りに使用されます。選択したパーティションフィールドの包括的な下限値。

  例: "2000-01-01T00:00:00.000Z"

- UPPER_BOUND (文字列) - 読み取りに使用されます。選択したパーティションフィールドの排他的上限値。

  例: "2024-01-01T00:00:00.000Z"

- NUM_PARTITIONS(整数) – デフォルト: 1。読み込みに使用されます。読み取り用のパーティションの数。
- INSTANCE_URL(文字列) – SAP インスタンスアプリケーションのホスト URL。

  例: https://example-externaldata.sierra.aws.dev

- SERVICE_PATH(文字列) – SAP インスタンスアプリケーションサービスパス。

  例: /sap/opu/odata/iwfnd/catalogservice;v=2

- CLIENT_NUMBER(文字列) – SAP インスタンスアプリケーションのクライアント番号。

  例: 100

- PORT_NUMBER(文字列) – デフォルト: SAP インスタンスアプリケーションのポート番号。

  例: 443

- LOGON_LANGUAGE(文字列) – SAP インスタンスアプリケーションの口グオン言語。

  例: EN

- ENABLE_CDC(文字列) – CDC を有効にしてジョブを実行するかどうか、つまりトラックの変更でジョブを実行するかどうかを定義します。

  例: True/False

- DELTA_TOKEN(文字列) – 指定された有効なデルタトークンに基づいて増分データプルを実行します。

  例: D20241107043437_000463000

- PAGE_SIZE (整数) - レコードをクエリするためのページサイズを定義します。デフォルトのページサイズは 50,000 です。ページサイズを指定すると、SAP はデータセット全体ではなく、API

コールごとに定義された数のレコードのみを返します。コネクタは引き続きレコードの合計数を提供し、指定したページサイズを使用してページ分割を処理します。ページサイズを大きくする必要がある場合は、最大 500,000 までの任意の値を選択できます。指定されたページサイズが 500,000 を超えると無視されます。代わりに、システムは許容される最大ページサイズを使用します。希望する値を指定した接続オプション `PAGE_SIZE` を追加することで、AWS Glue Studio UI でページサイズを指定できます。

例: `20000`

SAP OData エンティティとフィールドの詳細

エンティティ	データ型	サポートされている演算子
表 (動的エンティティ)	String	=, !=, >, >=, <, <=, BETWEEN, LIKE
	整数	=, !=, >, >=, <, <=, BETWEEN, LIKE
	Long	=, !=, >, >=, <, <=, BETWEEN, LIKE
	倍精度	=, !=, >, >=, <, <=, BETWEEN, LIKE
	日付	=, !=, >, >=, <, <=, BETWEEN, LIKE
	DateTime	=, !=, >, >=, <, <=, BETWEEN, LIKE
	ブール値	=, !=
	Struct	=、!=、<、<=、>、>=、BETWEEN、LIKE

SendGrid に対する接続

SendGrid は、取引 E メールおよびマーケティング E メール用のカスタマーコミュニケーションプラットフォームです。

- SendGrid コネクタは、連絡先リストの作成と管理、および E メールマーケティングキャンペーンの作成に役立ちます。
- SendGrid を使用すると、オンラインビジネス、非営利団体、その他のオンラインエンティティは、マーケティング E メールを作成して多くの視聴者に送信し、それらの E メールへのエンゲージメントを監視できます。

トピック

- SendGrid 向けの AWS Glue サポート
- 接続を作成および使用するための API オペレーションを含むポリシー
- SendGrid の設定
- SendGrid 接続の設定
- SendGrid エンティティからの読み取り
- SendGrid 接続オプション
- SendGrid の制限

SendGrid 向けの AWS Glue サポート

AWS Glue は、次のように SendGrid をサポートします。

ソースとしてサポートされていますか?

はい。AWS Glue ETL ジョブを使用して、SendGrid からデータをクエリできます。

ターゲットとしてサポートされていますか?

いいえ。

サポートされている SendGrid API バージョン

次の SendGrid API バージョンがサポートされています。

- v3

接続を作成および使用するための API オペレーションを含むポリシー

次のサンプル ポリシーで、接続の作成と使用に必要な AWS IAM 権限について説明します。新しいロールを作成する場合は、以下を含むポリシーを作成します。

```
{
 "Version": "2012-10-17",
    "Statement": [
     {
            "Effect": "Allow",
            "Action": [
                "glue:ListConnectionTypes",
                "glue:DescribeConnectionType",
                "glue:RefreshOAuth2Tokens",
                "glue:ListEntities",
                "glue:DescribeEntity"
            ],
            "Resource": "*"
        }
    ]
 }
```

上記の方法を使用しない場合は、代わりに次のマネージド IAM ポリシーを使用します:

- AWSGlueServiceRole – さまざまな AWS Glue プロセスを実行するために必要なリソースへのアクセス権をユーザーに代わって付与します。これらのリソースには AWS Glue 、Amazon S3、IAM、CloudWatch Logs、および Amazon EC2 が含まれます。このポリシーで指定されたリソースの命名規則に従った場合、AWS Glue プロセスは必要なアクセス権限を使用できます。このポリシーは、通常、クローラ、ジョブ、開発エンドポイントを定義するときに指定されたロールにアタッチされます。
- AWSGlueConsoleFullAccess — ポリシーがアタッチされているアイデンティティが AWS マネージメントコンソールを使用するときは、AWS Glue リソースへのフルアクセスを許可します。このポリシーで指定されたリソースの命名規則に従った場合、ユーザーは完全なコンソール機能を使用できます。このポリシーは、通常 AWS Glue コンソールのユーザーにアタッチされています。

SendGrid の設定

AWS Glue を使用して SendGrid からデータを転送できるようにするには、次の要件を満たす必要があります。

最小要件

以下に、最小要件を示します。

- API キーを持つ SendGrid アカウントがあります。
- SendGrid アカウントには、有効なライセンスを持つ API アクセスがあります。

これらの要件を満たすと、SendGrid アカウントに AWS Glue を接続する準備が整います。一般的な接続では、SendGrid で他に何もする必要はありません。

SendGrid 接続の設定

SendGrid はカスタム認証をサポートしています。

カスタム認証に必要な API キーの生成に関する SendGrid のパブリックドキュメントについては、「認証」を参照してください。

SendGrid 接続を設定するには：

1. AWS Secrets Manager で、次の詳細を含むシークレットを作成します。
   a. カスタマーマネージド接続アプリケーションの場合、シークレットには、*api_key* を使用して接続されたアプリケーションのコンシューマーシークレットをキーとして含める必要があります。
   b. 注: AWS Glue で接続にシークレットを作成する必要があります。

1. AWS Glue Glue Studio で、以下の手順に従って [データ接続] の下に接続を作成します。
   a. [接続タイプ] を選択する際に、[SendGrid] を選択します。
   b. 接続先の SendGrid インスタンスの `INSTANCE_URL` を指定します。
   c. 次のアクションを実行でき、AWS Glue がその権限を持つ AWS IAM ロールを選択します。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:DescribeSecret",
                "secretsmanager:GetSecretValue",
```

```
                "secretsmanager:PutSecretValue",
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterface",
                "ec2:DeleteNetworkInterface",
            ],
            "Resource": "*"
        }
    ]
}
```

d. AWS Glue でこの接続に使用する `secretName` を選択して、トークンを配置します。

e. ネットワークを使用する場合は、ネットワークオプションを選択します。

2. AWS Glue ジョブに関連付けられている IAM ロールに `secretName` を読み取るアクセス許可を付与します。

SendGrid エンティティからの読み取り

前提条件

読み取り元の SendGrid オブジェクト。`lists`、`singlesends`、または `segments` などのオブジェクト名が必要です。

ソースに対応するエンティティ:

エンティティ	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
リスト	いいえ	はい	いいえ	はい	いいえ
シングル送信	はい	はい	いいえ	はい	いいえ
マーケティングキャンペーンの統計自動化	はい	はい	いいえ	はい	いいえ
マーケティングキャンペー	はい	はい	いいえ	はい	いいえ

エンティティ	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
ンの統計 – シングル送信					
セグメント	はい	いいえ	いいえ	はい	いいえ
連絡先	はい	いいえ	いいえ	はい	いいえ
カテゴリ	いいえ	いいえ	いいえ	はい	いいえ
統計	はい	いいえ	いいえ	はい	いいえ
グループのサブスクリプションを解除する	はい	いいえ	いいえ	はい	いいえ

例:

```
sendgrid_read = glueContext.create_dynamic_frame.from_options(
    connection_type="sendgrid",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "lists",
        "API_VERSION": "v3",
        "INSTANCE_URL": "instanceUrl"
    }
```

SendGrid エンティティとフィールドの詳細:

静的メタデータを持つエンティティ:

エンティティ	フィールド	データ型	サポートされている演算子	
リスト	id	String	該当なし	

エンティティ	フィールド	データ型	サポートされている演算子	
	名前	String	該当なし	
	contact_count	整数	該当なし	
	_metadata	Struct	該当なし	
シングル送信	id	String	該当なし	
	名前	String	EQUAL_TO	
	省略	Struct	該当なし	
	ステータス	String	EQUAL_TO	
	カテゴリ	リスト	EQUAL_TO	
	send_at	String	該当なし	
	is_abtest	ブール値	該当なし	
	updated_at	String	該当なし	
	created_at	String	該当なし	
	チャンネル	リスト	該当なし	
マーケティングキャンペーンの統計自動化	id	String	該当なし	
	集計	String	該当なし	
	step_id	String	該当なし	
	stats	Struct	該当なし	
	Automation_ids	リスト	EQUAL_TO	

エンティティ	フィールド	データ型	サポートされている演算子	
マーケティングキャンペーンの統計シングル送信	id	String	該当なし	
	ab_variation	String	該当なし	
	ab_phase	String	該当なし	
	集計	String	該当なし	
	stats	Struct	該当なし	
	singlesend_ids	リスト	EQUAL_TO	
セグメント	id	String	該当なし	
	名前	String	該当なし	
	query_version	String	該当なし	
	contacts_count	整数	該当なし	
	sample_updated_at	String	該当なし	
	next_sample_update	String	該当なし	
	created_at	String	該当なし	
	updated_at	String	該当なし	
	parent_list_id	String	該当なし	
	ステータス	Struct	該当なし	
	parent_list_ids	String	EQUAL_TO	
	no_parent_list_id	ブール値	EQUAL_TO	
連絡先	id	String	該当なし	

エンティティ	フィールド	データ型	サポートされている演算子	
	first_name	String	該当なし	
	last_name	String	該当なし	
	unique_name	String	該当なし	
	email	String	該当なし	
	alternate_emails	リスト	該当なし	
	address_line_1	String	該当なし	
	address_line_2	String	該当なし	
	city	String	該当なし	
	state_province_region	String	該当なし	
	country	String	該当なし	
	postal_code	String	該当なし	
	phone_number	String	該当なし	
	WhatsApp	String	該当なし	
	LINE	String	該当なし	
	facebook	String	該当なし	
	list_ids	リスト	該当なし	
	custom_fields	Struct	該当なし	
	created_at	String	該当なし	
	updated_at	String	該当なし	

エンティティ	フィールド	データ型	サポートされている演算子	
	_metadata	Struct	該当なし	
	event_timestamp	DateTime	BETWEEN	
カテゴリ	カテゴリ	リスト	該当なし	
統計	date	String	該当なし	
	stats	リスト	該当なし	
	start_date	DateTime	EQUAL_TO、BETWEEN	
	aggregated_by	String	EQUAL_TO	
グループのサブスクリプションを解除する	id	整数	EQUAL_TO	
	名前	String	該当なし	
	description	String	該当なし	
	last_email_sent_at	整数	該当なし	
	is_default	ブール値	該当なし	
	サブスクリプション解除数	整数	該当なし	

> **Note**
>
> 構造体データ型とリストデータ型は、コネクタの応答で文字列データ型に変換され、DateTime データ型は Timestamp に変換されます。

パーティショニングクエリ

SendGrid は、フィルターベースのパーティショニングまたはレコードベースのパーティション分割をサポートしていません。

SendGrid 接続オプション

SendGrid の接続オプションは次のとおりです。

- `ENTITY_NAME`(文字列) – (必須) 読み取りに使用されます。SendGrid のオブジェクトの名前。
- `API_VERSION`(文字列) – (必須) 読み取りに使用されます。使用する SendGrid Rest API バージョン。
- `INSTANCE_URL`(文字列) – (必須) 読み取りに使用されます。有効な SendGrid インスタンス URL。
- `SELECTED_FIELDS`(List<String>) - Default: empty(SELECT *). 読み込みに使用されます。オブジェクトに選択する列。
- `FILTER_PREDICATE`(文字列) – デフォルト: 空 読み込みに使用されます。Spark SQL 形式である必要があります。
- `QUERY`(文字列) – デフォルト: 空 読み込みに使用されます。完全な Spark SQL クエリ。

SendGrid の制限

SendGrid の制限事項または注意事項は次のとおりです。

- 増分プルは、`start_date` フィールドの Stats エンティティと `event_timestamp` フィールドの Contact エンティティでのみサポートされます。
- ページ分割は、Marketing Campaign Stats (Automations)、Marketing Campaign Stats (Single Sends)、Single Sends、Lists エンティティでのみサポートされています。
- Stats エンティティの場合、`start_date` は必須のフィルターパラメータです。
- アクセスが制限された API キーは、E メール API および Stats エンティティの読み取りアクセスをサポートできません。フルアクセスを持つ API キーを使用します。詳細については、「API の概要」を参照してください。

への接続 ServiceNow

ServiceNow は、IT 管理ワークフローを自動化するためのクラウドベースの SaaS プラットフォームです。この ServiceNow プラットフォームは他のツールと簡単に統合できるため、ユーザーはさまざ

まなアプリケーションやプラグインを使用してプロジェクト、チーム、顧客とのインタラクションを管理できます。 ServiceNow ユーザーとして、 ServiceNow アカウント AWS Glue に接続できます。その後、 をETLジョブのデータソース ServiceNow として使用できます。これらのジョブを実行して、 ServiceNow と AWS のサービスまたはその他のサポートされているアプリケーション間でデータを転送します。

トピック

- AWS Glue のサポート ServiceNow
- 接続を作成および使用するためのAPIオペレーションを含むポリシー
- の設定 ServiceNow
- ServiceNow 接続の設定
- ServiceNow エンティティからの読み取り
- ServiceNow 接続オプション
- ServiceNow コネクタの制限と注意事項

AWS Glue のサポート ServiceNow

AWS Glue は ServiceNow 、次のようにサポートします。

ソースとしてサポートされていますか?

はい。ジョブを使用して AWS Glue ETL、 からデータをクエリできます ServiceNow。

ターゲットとしてサポートされているかどうか

いいえ。

サポートされている ServiceNow APIバージョン

次の ServiceNow APIバージョンがサポートされています。

- v2

バージョン固有のエンティティサポートについては、「Source でサポートされているエンティティ」を参照してください。

接続を作成および使用するためのAPIオペレーションを含むポリシー

次のサンプルポリシーでは、接続の作成と使用に必要な AWS IAMアクセス許可について説明します。新しいロールを作成する場合は、以下を含むポリシーを作成します。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:ListConnectionTypes",
                "glue:DescribeConnectionType",
                "glue:RefreshOAuth2Tokens"
                "glue:ListEntities",
                "glue:DescribeEntity"
            ],
            "Resource": "*"
        }
    ]
}
```

上記の方法を使用しない場合は、次の管理IAMポリシーを使用します。

- AWSGlueServiceRole – さまざまな AWS Glue プロセスがユーザーに代わって実行するために必要なリソースへのアクセスを許可します。これらのリソースには AWS Glue、、Amazon S3、IAM、 CloudWatch Logs、Amazon が含まれますEC2。このポリシーで指定されたリソースの命名規則に従うと、 AWS Glue プロセスには必要なアクセス許可があります。このポリシーは、通常、クローラ、ジョブ、開発エンドポイントを定義するときに指定されたロールにアタッチされます。
- AWSGlueConsoleFullAccess - ポリシーがアタッチされている ID が AWS マネジメントコンソールを使用する場合、 AWS Glue リソースへのフルアクセスを許可します。このポリシーで指定されたリソースの命名規則に従った場合、ユーザーは完全なコンソール機能を使用できます。このポリシーは通常、 AWS Glue コンソールのユーザーにアタッチされます。

の設定 ServiceNow

AWS Glue を使用して からデータを転送する前に ServiceNow、次の要件を満たす必要があります。

最小要件

以下に、最小要件を示します。

- E メールとパスワードを持つ ServiceNow アカウントがあります。詳細については、「ServiceNow アカウントの作成」を参照してください。
- ServiceNow アカウントへのアクセスが有効になっていますAPI。の使用 ServiceNow APIはすべて追加料金なしで利用できます。

これらの要件を満たすと、 ServiceNow アカウント AWS Glue に接続できます。

ServiceNow アカウントの作成

ServiceNow アカウントを作成するには：

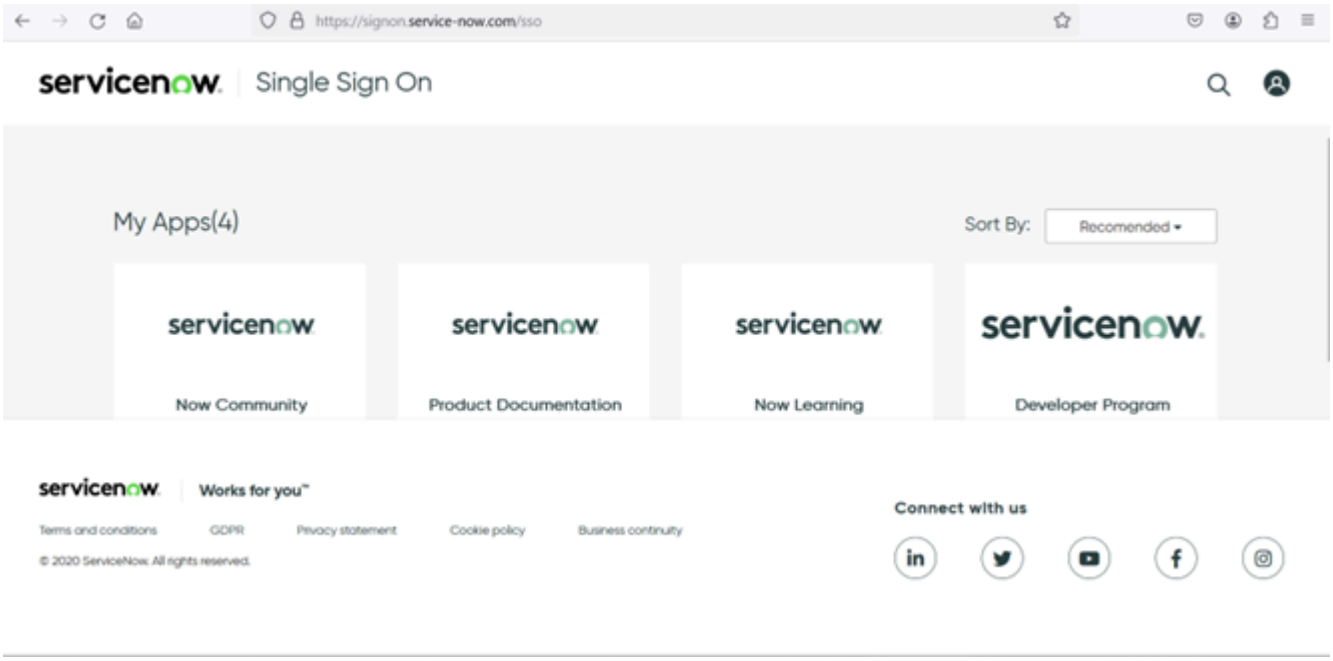
1. servicenow.com のサインアップページに移動し、詳細を入力し、続行 をクリックします。
2. 登録済みメールに検証コードを受け取ったら、そのコードを入力し、「検証」を選択します。
3. 多要素認証を設定するか、スキップします。

アカウントが作成され、プロファイル ServiceNow が表示されます。

デ ServiceNow ベロッパーインスタンスの作成

にログインした後、デベロッパーインスタンスをリクエストします ServiceNow。

1. ServiceNow ログインページで、アカウントの認証情報を入力します。
2. ServiceNow デベロッパープログラム を選択します。



3. 右上のリクエストインスタンスを選択します。
4. 職責を入力します。利用規約に同意し、セットアップの終了 を選択します。
5. インスタンスを作成したら、インスタンスURLと認証情報を書き留めます。

BasicAuth 認証情報の取得

無料アカウントの Basic Auth 認証情報を取得するには：

1. ServiceNow ログインページで、アカウントの認証情報を入力します。
2. ホームページで、プロファイルの編集セクション (右上隅) を選択し、インスタンスパスワードの管理 を選択します。
3. ユーザー名、パスワード、インスタンス などのログイン認証情報を取得しますURL。

> **Note**
>
> アカウントで が有効になっている場合MFAは、基本的な認証: <username>:<password><MFA Token> のユーザーのパスワードの末尾にMFAトークンを追加します。

詳細については、 ServiceNow ドキュメントの「アプリケーションの構築」を参照してください。

2.0 OAuth 認証情報の作成

ServiceNow コネクタで OAuth2.0 を使用するには、インバウンドクライアント) を作成してクライアント ID とクライアントシークレットを生成する必要があります。

1. ServiceNow ログインページで、アカウントの認証情報を入力します。
2. ホームページで「構築の開始」を選択します。
3. App Engine Studio ページで、Application Registry を検索します。
4. 右上で New を選択します。
5. 外部クライアントのOAuthAPIエンドポイントを作成する オプションを選択します。
6. OAuth 設定に必要な変更を加え、更新 を選択します。

   Redirect URL： https://us-east-1.console.aws.amazon.com/gluestudio/oauth の例
7. 新しく作成したOAuthクライアントアプリケーションを選択して、クライアント ID とクライアントシークレットを取得します。
8. クライアント ID とクライアントシークレットを保存して、処理を続行します。

非本番デベロッパーアカウントOAuthで を設定するには：

1. ServiceNow ドキュメントの認証プロファイルの作成トピックを使用して、認証プロファイルを作成します。
2. の認証プロファイルでOAuth、 としてタイプを選択しOAuth、上記で作成したインバウンドクライアントを選択してOAuthエンティティ を設定します。
3. 複数のクライアントがある場合は、複数の認証プロファイルを作成して、認証プロファイルに必要なOAuthエンティティを設定する必要があります。
4. 設定されていない場合は、RESTAPIアクセスポリシーを作成して、 TABLE へのアクセスを許可しますAPI。REST API アクセスポリシーの作成 を参照してください。

ServiceNow 接続の設定

グラントタイプは、AWS Glue が ServiceNow と通信してデータへのアクセスをリクエストする方法を決定します。選択すると、接続を作成する前に満たす必要がある要件に影響します。ServiceNow は、OAuth 2.0 の AUTHORIZATION_CODE のグラントタイプのみをサポートしています。

- このグラントタイプは、ユーザーを認証するためにサードパーティーの認証サーバーにユーザーをリダイレクトすることから、「three-legged」の OAuth と見なされます。AWS Glue コンソール経

由で接続を作成するときに使用されます。AWS Glue コンソールでは、ユーザーが ServiceNow にリダイレクトされます。ここではユーザーがログインし、ServiceNow インスタンスにアクセスするためのリクエストされた権限を AWS Glue に許可する必要があります。

- ユーザーは、ServiceNow で独自の接続アプリを作成し、AWS Glue コンソールを介して接続を作成するときに独自のクライアント ID とクライアントシークレットを指定することを選択できます。このシナリオでは、引き続き ServiceNow にリダイレクトされてログインし、リソースへアクセスするために AWS Glue を承認します。
- このグラントタイプは、更新トークンとアクセストークンになります。アクセストークンの有効期間は短く、更新トークンを使用してユーザーとやり取りすることなく自動的に更新される場合があります。
- 認可コード OAuth フロー用の接続アプリケーションの作成に関する ServiceNow の公開ドキュメントについては、「OAuth を設定する」を参照してください。

ServiceNow 接続を設定するには：

1. AWS Secrets Manager で、次の詳細を含むシークレットを作成します。
   a. 基本的な認証の場合、シークレットには、USERNAME と PASSWORD を使用して接続されたアプリケーションのコンシューマーシークレットをキーとして含める必要があります。
   b. 認可コード付与の場合、シークレットには、USER_MANAGED_CLIENT_APPLICATION_CLIENT_SECRET を使用して接続されたアプリケーションのコンシューマーシークレットをキーとして含める必要があります。
   c. 注: AWS Glue で接続ごとにシークレットを作成する必要があります。
2. AWS Glue Glue Studio で、以下の手順に従って [データ接続] の下に接続を作成します。
   a. [接続タイプ] を選択する際に、[ServiceNow] を選択します。
   b. 接続する ServiceNow インスタンスの INSTANCE_URL を指定します。
   c. 次のアクションを実行でき、AWS Glue がその権限を持つ AWS IAM ロールを選択します。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:DescribeSecret",
                "secretsmanager:GetSecretValue",
                "secretsmanager:PutSecretValue",
```

```
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterface",
                "ec2:DeleteNetworkInterface",
            ],
            "Resource": "*"
        }
    ]
}
```

d. AWS Glue でこの接続に使用する [認証タイプ] を選択します。

i. 基本認証: この認証タイプは、ServiceNow インスタンス内の特定のユーザーのアクセス許可でユーザー名とパスワードを事前に使用できるため、自動化シナリオに適しています。AWS Glue はユーザー名とパスワードを使用して ServiceNow API を認証できます。基本認証の場合にのみ、`Username` と `Password` を入力します。

ii. OAuth2: OAuth2 の場合にのみ、`ClientId`、`ClientSecret`、`Authorization URL`、`Authorization Token URL` を入力します。

e. AWS Glue でこの接続に使用する `secretName` を選択して、トークンを配置します。

f. ネットワークを使用する場合は、ネットワークオプションを選択します。

3. AWS Glue ジョブに関連付けられている IAM ロールに `secretName` を読み取るアクセス許可を付与します。

ServiceNow エンティティからの読み取り

前提条件

読み取り元の ServiceNow Tables オブジェクト。pa_bucket や Incident などのオブジェクト名が必要になります。

例:

```
servicenow_read = glueContext.create_dynamic_frame.from_options(
    connection_type="servicenow",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "pa_buckets",
        "API_VERSION": "v2"
        "instanceUrl": "https://<instance-name>.service-now.com"
    }
)
```

ServiceNow エンティティとフィールドの詳細:

次のエンティティについて、ServiceNow はメタデータを動的に取得するためのエンドポイントを備えているため、オペレータのサポートは各エンティティのデータ型レベルでキャプチャされます。

エンティティ	データ型	サポートされている演算子
表 (動的エンティティ)	整数	=, !=, <, <=, >, >=, BETWEEN
	BigDecimal	=, !=, <, <=, >, >=, BETWEEN
	浮動小数点数	=, !=, <, <=, >, >=, BETWEEN
	Long	=, !=, <, <=, >, >=, BETWEEN
	日付	=, !=, <, <=, >, >=, BETWEEN
	DateTime	=, !=, <, <=, >, >=, BETWEEN
	ブール値	=, !=
	文字列	=、!=、<、<=、>、>=、BETWEEN、LIKE
	Struct	該当なし

> **Note**
>
> 構造体データ型は、コネクタの応答で文字列データ型に変換されます。

> **Note**
>
> `DML_STATUS` は、CREATED/UPDATED レコードの追跡に使用される追加のユーザー定義属性です。

パーティショニングクエリ

フィールドベースのパーティショニング:

Spark で同時実行を使用する場合は、追加の Spark オプション PARTITION_FIELD、LOWER_BOUND、UPPER_BOUND、および NUM_PARTITIONS を指定できます。これらのパラメータを使用すると、元のクエリは Spark タスクで同時に実行できるサブクエリの NUM_PARTITIONS の数に分割されます。

エンティティ名	パーティション分割フィールド	データ型
動的エンティティ	sys_mod_count	整数
	sys_created_on、sys_updated_on	DateTime

- PARTITION_FIELD: クエリのパーティション化に使用するフィールドの名前。
- LOWER_BOUND: 選択したパーティションフィールドの包括的な下限値。

  Datetime フィールドでは、SPark SQL クエリで使用される Spark タイムスタンプ形式を受け入れます。

  有効な値の例は次のとおりです:

  ```
  "2024-01-30T06:47:51.000Z"
  ```
- UPPER_BOUND: 選択したパーティションフィールドの排他的上限値。
- NUM_PARTITIONS: パーティション数。

次の表に、エンティティパーティション分割フィールドのサポートの詳細を示します:

例:

```
servicenow_read = glueContext.create_dynamic_frame.from_options(
    connection_type="servicenow",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "pa_buckets",
        "API_VERSION": "v2",
        "instanceUrl": "https://<instance-name>.service-now.com"
        "PARTITION_FIELD": "sys_created_on"
        "LOWER_BOUND": "2024-01-30T06:47:51.000Z"
```

```
        "UPPER_BOUND": "2024-06-30T06:47:51.000Z"
        "NUM_PARTITIONS": "10"
    }
```

レコードベースのパーティション分割:

Spark で並行処理を使用する場合は、追加の Spark オプション NUM_PARTITIONS を指定できます。これらのパラメータを使用すると、元のクエリは Spark タスクで同時に実行できるサブクエリの NUM_PARTITIONS 数に分割されます。

レコードベースのパーティショニングでは、存在するレコードの合計数が ServiceNow API からクエリされ、指定された NUM_PARTITIONS 数で割られます。その後、結果のレコード数は、各サブクエリによって同時に取得されます。

- NUM_PARTITIONS: パーティション数。

例:

```
servicenow_read = glueContext.create_dynamic_frame.from_options(
    connection_type="servicenow",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "pa_buckets",
        "API_VERSION": "v2",
        "instanceUrl": "https://<instance-name>.service-now.com"
        "NUM_PARTITIONS": "2"
    }
```

ServiceNow 接続オプション

以下は、 の接続オプションです ServiceNow。

- ENTITY_NAME (文字列) - (必須) 読み取りに使用されます。のオブジェクトの名前 ServiceNow。
- API_VERSION (文字列) - (必須) 使用する Read. ServiceNow Rest APIバージョンに使用します。例: v1、v2、v3、v4。
- SELECTED_FIELDS (リスト < 文字列>) - デフォルト: empty(SELECT *)。読み込みに使用されます。オブジェクトに選択する列。
- FILTER_PREDICATE(文字列) - デフォルト: 空 読み込みに使用されます。Spark SQL形式である必要があります。

- QUERY(文字列) - デフォルト: 空 読み込みに使用されます。フル Spark SQLクエリ。
- PARTITION_FIELD (文字列) - 読み取りに使用されます。クエリをパーティション化するために使用するフィールド。
- LOWER_BOUND(文字列) - 読み取りに使用されます。選択したパーティションフィールドの包括的な下限値。例: 2024-01-30T06:47:51.000Z。
- UPPER_BOUND (文字列) - 読み取りに使用されます。選択したパーティションフィールドの排他的上限値。例: 2024-06-30T06:47:51.000Z。
- NUM_PARTITIONS(整数) - デフォルト: 1。読み込みに使用されます。読み取り用のパーティションの数。例: 10。
- INSTANCE_URL (文字列) - (必須) https://<instance-name>.service-now.com URLの形式の有効な ServiceNow インスタンス。

ServiceNow コネクタの制限と注意事項

ServiceNow コネクタの制限事項または注意事項は次のとおりです。

- SaaS documentationによると、sys_created_on、sys_updated_on、sys_mod_count はシステム生成フィールドです。コネクタは SaaS API に依存して、レスポンス本文にこれらのフィールドを提供します。
  - SaaS がどのエンティティに対してもこれらのフィールドを生成しない場合、フィルターベースのパーティショニングはサポートできません。
- SaaS API がレスポンスで sys_created_on フィールドと sys_updated_on フィールドを返さない場合、DML_STATUS は算出できません。

AWS Glue Studio での Slack への接続

Slack は、ユーザーがさまざまなパブリックチャネルやプライベートチャネルを通じてメッセージや添付ファイルを送信できる、エンタープライズコミュニケーションアプリです。Slack ユーザーの場合は、Slack アカウントに AWS Glue を接続できます。その後、Slack を ETL ジョブのデータソースとして使用できます。これらのジョブを実行して、Slack と AWS サービス、またはその他のサポートされているアプリケーション間でデータを転送します。

トピック

- AWS Glue Slack のサポート
- 接続を作成および使用するための API オペレーションを含むポリシー

- Slack の設定
- Slack 接続の設定
- Slack エンティティからの読み取り
- Slack 接続オプション
- 制限
- 新しい Slack アカウントの作成とクライアントアプリの設定

AWS Glue Slack のサポート

AWS Glue は、次のように Slack をサポートします:

ソースとしてサポートされていますか?

はい。AWS Glue ETL ジョブを使用して、Slack からデータをクエリできます。

ターゲットとしてサポートされているかどうか

いいえ。

サポートされている Slack API のバージョン

Slack API v2。

接続を作成および使用するための API オペレーションを含むポリシー

次のサンプル ポリシーで、接続の作成と使用に必要な IAM 権限について説明します。新しいロールを作成する場合は、以下を含むポリシーを作成します。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:ListConnectionTypes",
                "glue:DescribeConnectionType",
                "glue:RefreshOAuth2Tokens"
                "glue:ListEntities",
                "glue:DescribeEntity"
            ],
```

```
            "Resource": "*"
        }
    ]
}
```

また、次のマネージド IAM ポリシーを使用して、アクセスを許可できます。

- AWSGlueServiceRole – さまざまな AWS Glue プロセスを実行するために必要なリソースへのアクセス権をユーザーに代わって付与します。これらのリソースには、AWS Glue、Amazon S3、IAM、Amazon CloudWatch Logs および Amazon EC2 が含まれます。このポリシーで指定されたリソースの命名規則に従った場合、AWS Glue プロセスは必要なアクセス権限を使用できます。このポリシーは、通常、クローラ、ジョブ、開発エンドポイントを定義するときに指定されたロールにアタッチされます。
- AWSGlueConsoleFullAccess — ポリシーがアタッチされているアイデンティティが AWS マネージメントコンソールを使用するときは、AWS Glue リソースへのフルアクセスを許可します。このポリシーで指定されたリソースの命名規則に従った場合、ユーザーは完全なコンソール機能を使用できます。このポリシーは、通常 AWS Glue コンソールのユーザーにアタッチされています。

Slack の設定

AWS Glue を使用して Slack との間でデータを転送できるようにするには、次の要件を満たす必要があります:

最小要件

- Slack アカウントが必要です。詳細については、「新しい Slack アカウントの作成とクライアントアプリの設定」を参照してください。

これらの要件を満たすと、Slack アカウントに AWS Glue を接続する準備が整います。

Slack 接続の設定

Slack は OAuth 2 の `AUTHORIZATION_CODE` グラントタイプをサポートしています。

このグラントタイプは、ユーザーを認証するためにサードパーティーの認証サーバーにユーザーをリダイレクトすることから、「three-legged」の OAuth と見なされます。AWS Glue コンソール経由で接続を作成するときに使用されます。AWS Glue コンソールでは、ユーザーが Slack にリダイレクトされます。ここではユーザーがログインし、Slack インスタンスにアクセスするためのリクエストされた権限を AWS Glue に許可する必要があります。

ユーザーは、Slack で独自の接続アプリを作成し、AWS Glue コンソールを介して接続を作成するときに独自のクライアント ID とクライアントシークレットを指定することを選択できます。このシナリオでは、引き続き Slack にリダイレクトされてログインし、リソースへアクセスするために AWS Glue を承認します。

このグラントタイプは、更新トークンとアクセストークンになります。このアクセストークンは、作成から 1 時間後に期限切れになります。新しいアクセストークンは、更新トークンを使用してフェッチできます。

認可コード OAuth フロー用の接続アプリケーションの作成の詳細については、「Slack API」を参照してください。

Slack 接続を設定するには:

1. AWS Secrets Manager で、次の詳細を含むシークレットを作成します。AWS Glue で接続ごとにシークレットを作成する必要があります。
   a. カスタマーマネージド接続アプリケーションの場合 – シークレットには、USER_MANAGED_CLIENT_APPLICATION_CLIENT_SECRET を使用して接続されたアプリケーションのコンシューマーシークレットをキーとして含める必要があります。
2. AWS Glue Glue Studio で、以下の手順に従って [データ接続] の下に接続を作成します。
   a. [接続タイプ] を選択する際に、[Slack] を選択します。
   b. Slack 環境を指定します。
   c. AWS Glue が次のアクションを担い、その権限を持つことができる IAM ロールを選択します。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:DescribeSecret",
                "secretsmanager:GetSecretValue",
                "secretsmanager:PutSecretValue",
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterface",
                "ec2:DeleteNetworkInterface",
            ],
            "Resource": "*"
        }
    ]
```

```
}
```

d. AWS Glue でこの接続に使用する `secretName` を選択して、トークンを配置します。

e. ネットワークを使用する場合は、ネットワークオプションを選択します。

3. AWS Glue ジョブに関連付けられている IAM ロールに `secretName` を読み取るアクセス許可を付与します。

Slack エンティティからの読み取り

前提条件

- 読み取り元の Slack オブジェクト。

サポートされているエンティティ

エンティティ	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
会話	あり	はい	いいえ	はい	はい

例

```
slack_read = glueContext.create_dynamic_frame.from_options(
    connection_type="slack",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "conversations/C058W38R5J8"
    }
)
```

Slack エンティティとフィールドの詳細

エンティティ	フィールド	データタイプ	サポートされる演算子
会話	添付ファイル	リスト	該当なし

エンティティ	フィールド	データタイプ	サポートされる演算子
会話	bot_id	文字列	該当なし
会話	blocks	リスト	該当なし
会話	client_msg_id	文字列	該当なし
会話	is_starred	ブール値	該当なし
会話	last_read	文字列	該当なし
会話	latest_reply	文字列	該当なし
会話	reactions	リスト	該当なし
会話	replies	リスト	該当なし
会話	reply_count	整数	該当なし
会話	reply_users	リスト	該当なし
会話	reply_users_count	整数	該当なし
会話	subscribed	ブール値	該当なし
会話	subtype	文字列	該当なし
会話	text	文字列	該当なし
会話	team	文字列	該当なし
会話	thread_ts	文字列	該当なし

エンティティ	フィールド	データタイプ	サポートされる演算子
会話	ts	文字列	EQUAL_TO、FORE, LESS_THAN, LESS_THAN_OR_EQUAL_TO, GREATER_THAN, GREATER_THAN_OR_EQUAL_TO
会話	type	文字列	該当なし
会話	ユーザー	文字列	該当なし
会話	inviter	文字列	該当なし
会話	ルート	Struct	該当なし
会話	is_locked	ブール値	該当なし
会話	ファイル	リスト	該当なし
会話	部屋	Struct	該当なし
会話	アップロード	ブール値	該当なし
会話	display_as_bot	ブール値	該当なし
会話	チャンネル	文字列	該当なし
会話	no_notifications	ブール値	該当なし
会話	permalink	文字列	該当なし
会話	pinned_to	リスト	該当なし
会話	pinned_info	Struct	該当なし
会話	edited	Struct	該当なし

エンティティ	フィールド	データタイプ	サポートされる演算子
会話	app_id	文字列	該当なし
会話	bot_profile	Struct	該当なし
会話	metadata	Struct	該当なし

パーティショニングクエリ

Spark で同時実行を使用する場合は、追加の Spark オプション PARTITION_FIELD、LOWER_BOUND、UPPER_BOUND、および NUM_PARTITIONS を指定できます。これらのパラメータを使用すると、元のクエリは Spark タスクで同時に実行できるサブクエリの NUM_PARTITIONS の数に分割されます。

- PARTITION_FIELD: クエリのパーティション化に使用するフィールドの名前。
- LOWER_BOUND: 選択したパーティションフィールドの包括的な下限値。

  日付については、Spark SQL クエリで使用される Spark の日付形式を受け入れます。有効な値の例: "2024-07-01T00:00:00.000Z"。
- UPPER_BOUND: 選択したパーティションフィールドの排他的上限値。
- NUM_PARTITIONS: パーティション数。

エンティティごとのパーティション分割フィールドのサポートの詳細は、以下の表にまとめられています。

エンティティ名	Partitioning Field	データタイプ
会話	ts	文字列

例

```
slack_read = glueContext.create_dynamic_frame.from_options(
    connection_type="slack",
    connection_options={
```

```
        "connectionName": "connectionName",
        "ENTITY_NAME": "conversations/C058W38R5J8",
        "PARTITION_FIELD": "ts"
        "LOWER_BOUND": "2022-12-01T00:00:00.000Z"
        "UPPER_BOUND": "2024-09-23T15:00:00.000Z"
        "NUM_PARTITIONS": "2"
    }
)
```

Slack 接続オプション

Slack の接続オプションは次のとおりです:

- ENTITY_NAME (文字列) – (必須) 読み取りに使用されます。サポートされているエンティティ名。例えば、conversations/C058W38R5J8 などです。
- SELECTED_FIELDS(List<String>) - Default: empty(SELECT *). 読み込みに使用されます。オブジェクトに選択するフィールド。
- FILTER_PREDICATE(文字列) - デフォルト: 空 読み込みに使用されます。Spark SQL 形式である必要があります。
- QUERY(文字列) - デフォルト: 空 読み込みに使用されます。完全な Spark SQL クエリ。
- NUM_PARTITIONS(整数) - デフォルト: 1。読み込みに使用されます。読み取り用のパーティションの数。

制限

Slack コネクタの制限は次のとおりです:

- コネクタは、特定の会話で使用可能なレコード (メッセージ) の総数をフェッチする手段を備えていないため、レコードベースのパーティショニングはサポートされていません。

新しい Slack アカウントの作成とクライアントアプリの設定

Slack アカウントの作成

1. Slack ホームページを開いて、アカウントにサインアップします。
2. [E メールアドレスでサインアップ] を選択します。E メール ID を入力し、[続行] を選択します。

3. E メールアドレスに送信された 6 文字のコードを入力すると、ワークスペースの作成または既存のワークスペースへの参加にリダイレクトされます。
4. [ワークスペースの作成] を選択して、新しいワークスペースを作成します。セットアッププロセスの一環として、いくつかの質問に回答するようにリダイレクトされます。
   - 会社名
   - 氏名
   - E メールで同僚を追加するには
   - チームは何に取り組んでいますか? (これはチャンネル名になります)
5. これらの質問の入力フィールドに入力し、続行します。これで、アカウントを使用する準備ができました。

Slack 開発者アプリの作成

1. Slack アカウントにログインし、Slack ワークスペースにサインインします。
2. ワークスペースメニューから、[ツールと設定] を選択し、[アプリの管理] を選択します。
3. Slack アプリディレクトリのメニューから、[ビルド] を選択します。
4. [アプリ] ページで、[アプリの作成] を選択します。
5. [アプリの作成] ページで、[最初から作成] を選択します。
6. 開いた [アプリ名とワークスペースの選択] ダイアログボックスで、アプリ名を追加し、[アプリをデプロイするワークスペースを選択します]。次に、[アプリの作成] を選択します。
7. アプリ認証情報に表示されるクライアント ID とシークレットを書き留めます。
8. [OAuth とアクセス許可] サイドバーで、[スコープ] に移動し、[OAuth スコープの追加] を選択します。リダイレクト URL を設定のアプリに追加して、[Slack に追加] ボタンを自動的に生成したり、アプリを配信したりできます。[リダイレクト URL] セクションまでスクロールし、[新しいリダイレクト URL を追加] を選択して保存します。
9. 次に、[Workspace の OAuth トークン] セクションまでスクロールし、[Workspace にインストール] を選択します。
10. 作成したアプリが、接続先の Slack ワークスペースへのアクセス許可をリクエストしていることを知らせるために開くダイアログボックスで、[許可] を選択します。
11. 正常に完了すると、コンソールに [ワークスペースの OAuth トークン] 画面が表示されます。
12. [ワークスペースの OAuth トークン] 画面から、AWS Glue への接続に使用される OAuth トークンをコピーして保存します

13. 次に、Slack チーム ID を取得します。[Slack ワークスペース] メニューから、[ツールと設定] を選択し、[アプリの管理] を選択します。チーム ID は、開いたページの URL にあります。
14. アプリを公開するには、サイドバーの [ディストリビューションの管理] ボタンに進んでアクティブ化します。[アプリを他のワークスペースと共有] セクションまでスクロールし、[ハードコードされた情報の削除] を選択します。同意し、[公開配布をアクティブ化] を選択します。
15. これで、アプリが公開されました。エンティティ API にアクセスするには、ユーザーがアクセスするすべてのワークスペースチャンネルにアプリを追加する必要があります。
16. Slack アカウントにサインインし、チャンネルにアクセスする必要があるワークスペースを開きます。
17. ワークスペースで、アプリがアクセスするチャンネルを開き、チャンネルタイトルを選択します。ポップアップから [統合] タブを選択し、アプリを追加します。このようにして、アプリはチャンネルと統合され、その API にアクセスできます。

    OAuth 2.0 クライアント ID には、1 つ以上の許可されたリダイレクト URL が必要です。リダイレクト URL の形式は次のとおりです:

> **Note**
>
> Appflow リダイレクト URL は、AWS Glue プラットフォーム用のリダイレクト URL が利用可能になった後に変更される場合があります。クライアント ID とクライアントシークレットは、OAuth 2.0 クライアント ID の設定からのものです。

リダイレクト URL は次のいずれかになります。	
Gamma 環境のリダイレクト URL	
https://us-west-2.console.aws.amazon.com/appflow/oauth	https://us-east-1.awsc-integ.aws.amazon.com/appflow/oauth
https://us-east-2.console.aws.amazon.com/appflow/oauth	
https://us-west-1.console.aws.amazon.com/appflow/oauth	

リダイレクト URL は次のいずれかになります。	
Gamma 環境のリダイレクト URL	
https://ap-south-1.console.aws.amazon.com/appflow/oauth	
https://ap-southeast-1.console.aws.amazon.com/appflow/oauth	
https://ap-southeast-2.console.aws.amazon.com/appflow/oauth	
https://ap-northeast-1.console.aws.amazon.com/appflow/oauth	
https://ap-northeast-2.console.aws.amazon.com/appflow/oauth	
https://ca-central-1.console.aws.amazon.com/appflow/oauth	
https://eu-central-1.console.aws.amazon.com/appflow/oauth	
https://eu.west-1.console.aws.amazon.com/appflow/oauth	
https://eu-west-2.console.aws.amazon.com/appflow/oauth	
https://eu-west-3.console.aws.amazon.com/appflow/oauth	
https://sa-east-1.console.aws.amazon.com/appflow/oauth	
https://us-west-2.awsc-integ.aws.amazon.com/appflow/oauth	

リダイレクト URL は次のいずれかになります。	
Gamma 環境のリダイレクト URL	
https://af-south-1.console.aws.amazon.com/appflow/oauth	

Smartsheet に対する接続

Smartsheet は、ワーク管理およびコラボレーション SaaS 製品です。基本的に、Smartsheet を使用すると、ユーザーはスプレッドシートのようなオブジェクトを使用してビジネスデータを作成、保存、利用できます。

トピック

- Smartsheet 向けの AWS Glue サポート
- 接続を作成および使用するための API オペレーションを含むポリシー
- Smartsheet の設定
- Smartsheet 接続の設定
- Smartsheet エンティティからの読み取り
- Smartsheet 接続オプション
- Smartsheet アカウントの作成
- 制限

Smartsheet 向けの AWS Glue サポート

AWS Glue は、次のように Smartsheet をサポートします。

ソースとしてサポートされていますか?

はい。AWS Glue ETL ジョブを使用して、Smartsheet からデータをクエリできます。

ターゲットとしてサポートされていますか?

いいえ。

サポートされている Smartsheet API バージョン

v2.0

接続を作成および使用するための API オペレーションを含むポリシー

次のサンプル ポリシーで、接続の作成と使用に必要な AWS の権限について説明します。新しいロールを作成する場合は、以下を含むポリシーを作成します。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:ListConnectionTypes",
                "glue:DescribeConnectionType",
                "glue:RefreshOAuth2Tokens",
                "glue:ListEntities",
                "glue:DescribeEntity"
            ],
            "Resource": "*"
        }
    ]
}
```

上記の方法を使用しない場合は、代わりに次のマネージド IAM ポリシーを使用します。

- AWSGlueServiceRole – さまざまな AWS Glue プロセスを実行するために必要なリソースへのアクセス権をユーザーに代わって付与します。これらのリソースには、AWS Glue、Amazon S3、IAM、CloudWatch Logs、および Amazon EC2 が含まれます。このポリシーで指定されたリソースの命名規則に従った場合、AWS Glue プロセスは必要なアクセス権限を使用できます。このポリシーは、通常、クローラ、ジョブ、開発エンドポイントを定義するときに指定されたロールにアタッチされます。
- AWSGlueConsoleFullAccess — ポリシーがアタッチされているアイデンティティが AWS マネージメントコンソールを使用するときは、AWS Glue リソースへのフルアクセスを許可します。このポリシーで指定されたリソースの命名規則に従った場合、ユーザーは完全なコンソール機能を使用できます。このポリシーは、通常 AWS Glue コンソールのユーザーにアタッチされています。

Smartsheet の設定

AWS Glue を使用して Smartsheet から転送できるようにするには、次の要件を満たす必要があります。

最小要件

- E メールとパスワードが設定された Smartsheet アカウントがある。アカウントの作成の詳細については、「Smartsheet アカウントの作成」を参照してください。
- Smartsheet アカウントには、有効なライセンスを持つ API アクセスがあります。
- Smartsheet アカウントには、`Sheets` エンティティの [Pro 料金] プランと、`Events` エンティティの Event Reporting アドオンを使用するエンタープライズ料金プランがあります。

これらの要件を満たすと、Smartsheet アカウントに AWS Glue を接続する準備が整います。一般的な接続では、Smartsheet で他に何もする必要はありません。

Smartsheet 接続の設定

Smartsheet は OAuth2 の `AUTHORIZATION_CODE` グラントタイプをサポートしています。

このグラントタイプは、ユーザーを認証するためにサードパーティーの認証サーバーにユーザーをリダイレクトすることから、「three-legged」の `OAuth` と見なされます。ユーザーは、Smartsheet で独自の接続アプリを作成し、AWS Glue コンソールを介して接続を作成するときに独自のクライアント ID とクライアントシークレットを指定することを選択できます。このシナリオでは、引き続き Smartsheet にリダイレクトされてログインし、リソースへアクセスするために AWS Glue を承認します。

このグラントタイプは、更新トークンとアクセストークンになります。アクセストークンの有効期間は短く、更新トークンを使用してユーザーとやり取りすることなく自動的に更新される場合があります。

AUTHORIZATION_CODE OAuth フローの接続アプリケーションの作成に関する Smartsheet の公開ドキュメントについては、「Smartsheet API」を参照してください。

Smartsheet 接続を設定するには:

1. AWS Secrets Manager で、次の詳細を含むシークレットを作成します。

カスタマーマネージド接続アプリケーションの場合 – シークレットには、USER_MANAGED_CLIENT_APPLICATION_CLIENT_SECRET を使用して接続されたアプリケーションのコンシューマーシークレットをキーとして含める必要があります。

> **Note**
>
> AWS Glue で接続ごとにシークレットを作成する必要があります。

2. AWS Glue Studio で、以下の手順に従って [データ接続] の下に接続を作成します。
   a. [接続タイプ] を選択する際に、[Smartsheet] を選択します。
   b. 接続先の Smartsheet の `instanceUrl` を指定します。
   c. AWS Glue が次のアクションを担い、その権限を持つことができる IAM ロールを選択します。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:DescribeSecret",
                "secretsmanager:GetSecretValue",
                "secretsmanager:PutSecretValue",
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterface",
                "ec2:DeleteNetworkInterface",
            ],
            "Resource": "*"
        }
    ]
}
```

   d. AWS Glue でこの接続に使用する `secretName` を選択して、トークンを配置します。
   e. ネットワークを使用する場合は、ネットワークオプションを選択します。
3. AWS Glue ジョブに関連付けられている IAM ロールに `secretName` を読み取るアクセス許可を付与します。

Smartsheet エンティティからの読み取り

前提条件

読み取り元の Smartsheet オブジェクト。使用可能なエンティティを確認するには、以下のサポートされているエンティティの表を参照してください。

サポートされているエンティティ

エンティティ	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
シートを一覧表示する	はい	はい	いいえ	はい	いいえ
行メタデータ	はい	はい	いいえ	はい	いいえ
シートメタデータ	いいえ	いいえ	いいえ	はい	いいえ
シートデータ	はい	はい	はい	はい	いいえ
イベント	はい	はい	いいえ	はい	いいえ

例

```
Smartsheet_read = glueContext.create_dynamic_frame.from_options(
    connection_type="smartsheet",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "list-sheets",
        "API_VERSION": "2.0",
        "INSTANCE_URL": "https://api.smartsheet.com"
    })
```

Smartsheet エンティティとフィールドの詳細

エンティティ	フィールド	データタイプ	サポートされている演算子
シートを一覧表示する	id	Long	NA
	accessLevel	String	NA
	createdAt	DateTime	NA
	modifiedAt	DateTime	NA
	名前	String	NA
	permalink	String	NA
	modifiedSince	DateTime	>=
	version	整数	NA
	source	Struct	NA
行メタデータ	id	Long	NA
	SheetId	Long	NA
	accessLevel	String	NA
	attachments	リスト	NA
	columns	リスト	NA
	conditionalFormat	String	NA
	createdAt	DateTime	NA
	CreatedBy	Struct	NA
	説明	リスト	NA
	証明	Struct	NA
	拡張	ブール値	NA

エンティティ	フィールド	データタイプ	サポートされている演算子
	filteredOut	ブール値	NA
	format	String	NA
	inCriticalPath	ブール値	NA
	locked	ブール値	NA
	lockedForUser	ブール値	NA
	modifiedAt	DateTime	NA
	modifiedBy	Struct	NA
	permalink	String	NA
	rowNumber	整数	NA
	version	整数	NA
	totalRowCount	整数	NA
	rowsModifiedSince	DateTime	>
	filterId	Long	“="
	siblingId	Long	NA
	parentId	Long	NA
シートメタデータ	id	Long	NA
	fromId	Long	NA
	ownerId	Long	NA
	accessLevel	String	NA
	attachments	リスト	NA

エンティティ	フィールド	データタイプ	サポートされている演算子
	columns	リスト	NA
	createdAt	DateTime	NA
	crossSheetReferences	リスト	NA
	dependenciesEnabled	ブール値	NA
	説明	リスト	NA
	effectiveAttachmentOptions	リスト	NA
	favorite	ブール値	NA
	ganttEnabled	ブール値	NA
	hasSummaryFields	ブール値	NA
	modifiedAt	DateTime	NA
	名前	String	NA
	owner	String	NA
	permalink	String	NA
	projectSettings	Struct	NA
	readOnly	ブール値	NA
	resourceManagementEnabled	ブール値	NA
	showParentRowsForFilters	ブール値	NA

エンティティ	フィールド	データタイプ	サポートされている演算子
	source	Struct	NA
	概要	Struct	NA
	totalRowCount	整数	NA
	userPermissions	Struct	NA
	userSettings	Struct	NA
	version	整数	NA
	ワークスペース	Struct	NA
	フィルター	リスト	NA
	ganttConfig	Struct	NA
	resourceManagementType	String	NA
	cellImageUploadEnabled	ブール値	NA
	isMultiPicklistEnabled	ブール値	NA
イベント	eventId	String	NA
	objectType	String	NA
	action	String	NA
	objectId	Long	NA
	eventTimestamp	DateTime	NA
	userId	Long	NA
	requestUserId	Long	NA

エンティティ	フィールド	データタイプ	サポートされている演算子
	accessTokenName	String	NA
	source	String	NA
	additionalDetails	Struct	NA
	since	DateTime	>=

動的メタデータを持つエンティティ:

次のエンティティについて、Smartsheet はメタデータを動的に取得するためのエンドポイントを備えているため、オペレータのサポートはデータ型レベルでキャプチャされます。

エンティティ	データタイプ	サポートされる演算子
シートデータ	String	NA
シートデータ	Long	"="
シートデータ	整数	NA
シートデータ	DateTime	>

例

```
Smartsheet_read = glueContext.create_dynamic_frame.from_options(
    connection_type="smartsheet",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "list-sheets",
        "API_VERSION": "2.0",
        "INSTANCE_URL": "https://api.smartsheet.com"
    }
```

Smartsheet 接続オプション

Smartsheet の接続オプションは次のとおりです。

- ENTITY_NAME(文字列) – (必須) 読み取り / 書き込みに使用されます。Smartsheet 内のオブジェクトの名前。
- API_VERSION(文字列) – (必須) 読み取り / 書き込みに使用されます。使用する Smartsheet Rest API バージョン。例: v2.0。
- INSTANCE_URL(文字列) – (必須) 読み取りに使用されます。Smartsheet インスタンス URL。
- SELECTED_FIELDS(List<String>) – Default: empty(SELECT *). 読み込みに使用されます。オブジェクトに選択する列。
- FILTER_PREDICATE(文字列) – デフォルト: 空 読み込みに使用されます。Spark SQL 形式である必要があります。
- QUERY(文字列) – デフォルト: 空 読み込みに使用されます。完全な Spark SQL クエリ。

Smartsheet アカウントの作成

1. Smartsheet のサインアップページにアクセスして、スマートシートアカウントにサインアップします。
2. [1 つ作成] を選択して新しいアカウントを作成するか、登録した Google、Microsoft、または Apple アカウントを使用してサインインします。
3.
4.
5. Smartsheet からの確認 E メールを開き、確認リンクを選択してアカウントを確認します。

   デフォルトでは、トライアルプランにサブスクライブされます。
6. 左下隅の [アカウント] アイコンを選択し、[ライセンスの追加/アップグレード] を選択して料金プランをアップグレードします。

   > **Note**
   >
   > これは、[エンタープライズ] プランのアドオンである [イベントレポート] にアクセスするために必要です。

7. [エンタープライズ] プランで、[お問い合わせ] を選択して、サポートチームにアカウントのアップグレードをリクエストします。

8. サポートリクエストフォームで、プランのアップグレードに必要な詳細と要件を入力します。

   これで [エンタープライズ] プランへのアップグレードは完了です。

OAuth2.0 認証情報の作成

1. [デベロッパー用ツール] にアクセスできるようにアカウントの料金プランをアップグレードしたら、Smartsheet デベロッパーにアクセスします。

   アクティベーション E メールが届きます。

2. Smartsheet からのアクティベーション E メールを開き、アクティベーションリンクを選択してアカウントでデベロッパー用ツールをアクティブ化します。

   デベロッパーツールを使用すると、アプリケーションを作成できます。

3. Smartsheet アカウントのホームページを開き、[アカウント] を選択して、アクセスをチェックします。
4. サービスリストから [デベロッパー用ツール] を選択し、[デベロッパープロファイル] の詳細を入力します。
5. [新しいアプリの作成] を選択します。
6. アプリ登録フォームに次の詳細を入力します。

   - [名前] – アプリケーションの名前。
   - [説明] – アプリケーションの説明。
   - [URL] – アプリを起動できる URL、またはランディングページの URL。
   - [連絡先/サポート] – サポートチームの連絡先情報。
   - [リダイレクト URL] – OAuth 2.0 認証情報を受け取るアプリケーション内の URL (コールバック URL とも呼ばれます)。

7. [保存] を選択します。

   Smartsheet は、クライアント ID とクライアントシークレットをアプリに割り当てます。次のステップで、これらの値を記録します。[デベロッパー用ツール] セクションの後半で再度参照することもできます。

制限

Smartsheet は、フィールドベースまたはレコードベースのパーティショニングをサポートしていません。

AWS Glue Studio での Snapchat 広告への接続

Snapchat は、元々 Snapchat Inc. であった Snap Inc. が開発したマルチメディアインスタントメッセージングアプリおよびサービスです。Snapchat の主な機能の 1 つは、写真やメッセージは通常短い時間しか利用できず、その後受信者はアクセスできなくなるということです。Snapchat Marketing は、企業が料金を払うことで Snapchat ユーザーに出すことができる投稿です。

トピック

- AWS Glue の Snapchat 広告サポート
- 接続を作成および使用するための API オペレーションを含むポリシー
- Snapchat 広告の設定
- Snapchat 広告接続の設定
- Snapchat 広告エンティティからの読み取り
- Snapchat 広告の接続オプション
- Snapchat 広告アカウントの作成とクライアントアプリの設定
- Snapchat 広告アカウントでのアプリの作成

AWS Glue の Snapchat 広告サポート

AWS Glue は、次のように Snapchat 広告をサポートします:

ソースとしてサポートされていますか?

はい。AWS Glue ETL ジョブを使用して、Snapchat 広告からデータをクエリできます。

ターゲットとしてサポートされているかどうか

いいえ。

サポートされている Snapchat 広告 API のバージョン

v1。

接続を作成および使用するための API オペレーションを含むポリシー

次のサンプル ポリシーで、接続の作成と使用に必要な AWS の権限について説明します。新しいロールを作成する場合は、以下を含むポリシーを作成します。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:ListConnectionTypes",
                "glue:DescribeConnectionType",
                "glue:RefreshOAuth2Tokens"
                "glue:ListEntities",
                "glue:DescribeEntity"
            ],
            "Resource": "*"
        }
    ]
}
```

また、次のマネージド IAM ポリシーを使用して、アクセスを許可できます。

- AWSGlueServiceRole – さまざまな AWS Glue プロセスを実行するために必要なリソースへのアクセス権をユーザーに代わって付与します。これらのリソースには、AWS Glue、Amazon S3、IAM、Amazon CloudWatch Logs および Amazon EC2 が含まれます。このポリシーで指定されたリソースの命名規則に従った場合、AWS Glue プロセスは必要なアクセス権限を使用できます。このポリシーは、通常、クローラ、ジョブ、開発エンドポイントを定義するときに指定されたロールにアタッチされます。
- AWSGlueConsoleFullAccess — ポリシーがアタッチされているアイデンティティが AWS マネージメントコンソールを使用するときは、AWS Glue リソースへのフルアクセスを許可します。このポリシーで指定されたリソースの命名規則に従った場合、ユーザーは完全なコンソール機能を使用できます。このポリシーは、通常 AWS Glue コンソールのユーザーにアタッチされています。

Snapchat 広告の設定

AWS Glue を使用して Snapchat 広告からデータを転送できるようにするには、次の要件を満たす必要があります:

最小要件

- Snapchat 広告アカウントを持っている。アカウントの作成の詳細については、「Snapchat 広告アカウントの作成とクライアントアプリの設定」を参照してください。
- Snapchat 広告アカウントに OAuth2 アプリを作成しました。この統合により、AWS Glue がお客様のアカウントに対して認証呼び出しを行うときに、お客様のデータに安全にアクセスするために使用する認証情報が提供されます。詳細については、「Snapchat 広告アカウントでのアプリの作成」を参照してください。

これらの要件を満たすと、Snapchat 広告アカウントに AWS Glue を接続する準備が整います。Snapchat 広告では、接続されたアプリは、AWS Glue などの外部アプリが Snapchat 広告データにアクセスすることを許可するフレームワークです。

Snapchat 広告接続の設定

Snapchat 広告は、`AUTHORIZATION_CODE` グラントタイプのみをサポートします。

このグラントタイプは、ユーザーを認証するためにサードパーティーの認証サーバーにユーザーをリダイレクトすることから、「three-legged」の OAuth と見なされます。AWS Glue コンソール経由で接続を作成するときに使用されます。接続を作成するユーザーは、デフォルトで AWS Glue 所有の接続アプリ (AWS Glue マネージドクライアントアプリケーション) に依存する場合があります。この場合、Snapchat 広告のインスタンス URL 以外の OAuth 関連情報を提供する必要はありません。AWS Glue コンソールでは、ユーザーが Snapchat 広告にリダイレクトされます。ここではユーザーがログインし、Snapchat 広告インスタンスにアクセスするためのリクエストされた権限を AWS Glue に許可する必要があります。

ユーザーは、Snapchat 広告で独自の接続アプリを作成し、AWS Glue コンソールを介して接続を作成するときに独自のクライアント ID とクライアントシークレットを指定することを選択できます。このシナリオでは、引き続き Snapchat 広告にリダイレクトされてログインし、リソースにアクセスするために AWS Glue を承認します。

このグラントタイプは、更新トークンとアクセストークンになります。このアクセストークンは、作成から 1 時間後に期限切れになります。新しいアクセストークンは、更新トークンを使用してフェッチできます。

OAuth 認証コードフローでの接続されたアプリケーションの作成の詳細については、「Ads API」を参照してください。

Snapchat 広告の接続を設定するには:

1. AWS Secrets Manager で、次の詳細を含むシークレットを作成します。AWS Glue で接続ごとにシークレットを作成する必要があります。
   a. カスタマーマネージド接続アプリケーションの場合 – シークレットには、USER_MANAGED_CLIENT_APPLICATION_CLIENT_SECRET を使用して接続されたアプリケーションのコンシューマーシークレットをキーとして含める必要があります。
2. AWS Glue Glue Studio で、以下の手順に従って [データ接続] の下に接続を作成します。
   a. [接続タイプ] を選択する際に、[Snapchat 広告] を選択します。
   b. Snapchat 広告の環境を指定します。
   c. AWS Glue が次のアクションを担い、その権限を持つことができる IAM ロールを選択します。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:DescribeSecret",
                "secretsmanager:GetSecretValue",
                "secretsmanager:PutSecretValue",
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterface",
                "ec2:DeleteNetworkInterface",
            ],
            "Resource": "*"
        }
    ]
}
```

   d. AWS Glue でこの接続に使用する secretName を選択して、トークンを配置します。
   e. ネットワークを使用する場合は、ネットワークオプションを選択します。
3. AWS Glue ジョブに関連付けられている IAM ロールに secretName を読み取るアクセス許可を付与します。

Snapchat 広告エンティティからの読み取り

前提条件

- 読み取り元の Snapchat 広告のオブジェクト。使用可能なエンティティを確認するには、以下のサポートされているエンティティの表を参照してください。

サポートされているエンティティ

エンティティ	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
組織	なし	いいえ	いいえ	はい	なし
Ad Account	なし	いいえ	いいえ	はい	なし
Creative	なし	いいえ	いいえ	はい	なし
メディア	なし	いいえ	いいえ	はい	なし
Campaign	はい	いいえ	いいえ	はい	なし
Ad Under Ad Account	はい	いいえ	いいえ	はい	なし
Ad Under Campaign	なし	いいえ	いいえ	はい	なし
Ad Squad	はい	いいえ	いいえ	はい	なし
Segment	なし	いいえ	いいえ	はい	なし

例

```
snapchatads_read = glueContext.create_dynamic_frame.from_options(
    connection_type="snapchatAds",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "organization",
        "API_VERSION": "v1"
    }
)
```

Snapchat 広告エンティティとフィールドの詳細

Snapchat 広告は、選択したエンティティで使用可能なフィールドを動的にロードします。フィールドのデータ型に応じて、次のフィルター演算子がサポートされます。

フィールドデータタイプ	サポートされているフィルター演算子
ブール値	=

パーティショニングクエリ

- フィールドベースのパーティショニング: サポートされていません。
- レコードベースのパーティショニング: サポートされていません。

Snapchat 広告の接続オプション

Snapchat 広告の接続オプションは次のとおりです:

- ENTITY_NAME (文字列) – (必須) 読み取りに使用されます。Snapchat 広告エンティティの名前。例えば、 campaign などです。
- API_VERSION (文字列) – (必須) 読み取りに使用されます。使用する Snapchat 広告の Rest API バージョン。Snapchat 広告は現在バージョン v1 のみをサポートしているため、この値は v1 になります。
- SELECTED_FIELDS(List<String>) - Default: empty(SELECT *). 読み込みに使用されます。選択したエンティティに対して選択する列のカンマ区切りリスト。
- FILTER_PREDICATE(文字列) - デフォルト: 空 読み込みに使用されます。Spark SQL 形式である必要があります。
- QUERY(文字列) - デフォルト: 空 読み込みに使用されます。完全な Spark SQL クエリ。

Snapchat 広告アカウントの作成とクライアントアプリの設定

トピック

- Snapchat 広告にサインアップする
- Snapchat 広告アカウントを作成する手順

Snapchat 広告にサインアップする

Snapchat 広告にサインアップするには:

1. Snapchat 広告マネージャに移動します。[Snapchat を使用するのは初めてですか?] の横にある [登録] を選択します。
2. [アカウントの作成] 画面で、プロンプトに従ってビジネス名、E メール、パスワードなどを入力します。[Next] を選択します。
3. [プロフィールの作成] 画面で、ユーザー名、ウェブサイト (省略可) の値を入力し、[アカウントの作成] を選択します。これにより、[プロフィールの編集] 画面にプロフィール写真と自己紹介を追加するオプションが表示されます。[確認] を選択します。
4. [ビジネス情報] 画面で、国、通貨、電話番号、GSTIN などの必須フィールドを入力し、[次へ] を選択してアカウント作成プロセスを完了します。

Snapchat 広告アカウントを作成する手順

Snapchat 広告アカウントを作成するには:

1. [広告マネージャ] にログインします。次に、上部隅にあるナビゲーションをクリックし、[広告アカウント] を選択します。
2. [+ 新規広告アカウント] を選択します。広告主の詳細を入力します:
   - 広告主の代わりに広告を購入する代理業者であるかどうかを選択します。「はい」を選択すると、年齢、性別、郵便番号レベルのターゲティングなどのターゲティングパラメータを使用する場合、広告が拒否される可能性があります。最低年齢のターゲティングは、最大 21 歳まで適用できます。
   - 広告アカウントが住宅、クレジット、雇用広告を実行するかどうかを選択します。「はい」を選択すると、年齢、性別、郵便番号レベルのターゲティングなどのターゲティングパラメータを使用する場合、広告が拒否される可能性があります。最低年齢のターゲティングは、最大 21 歳まで適用できます。
   - 政治広告に広告アカウントを使用するかどうかを選択します。政治広告を運営している場合は、広告の支払いを行っているスポンサーの政治組織または支持団体を入力します。政治組織を正確に入力しないと、広告が拒否される可能性があります。また、広告を送信する前に、リンクされた必須の「Political Ad Review Form」を入力する必要があります。
3. [アカウントの詳細] を選択し、広告アカウント情報を入力します:

フィールド	説明
名前	広告アカウント名。
アカウントの種類	自動入力されていない場合は、アカウントタイプを選択します。
請求タイプ	資金源に継続的にアクセスしたい場合は、[リボルビング] を選択します。これにより、クレジットラインを補充できます。広告マネージャで広告を実行する場合はこれがよく選択されます。支出上限を設定して 1 回限りのキャンペーンを実行する場合は、[広告掲載申し込み] を選択できます。 広告掲載申し込みでアカウントを作成し、後で支出上限を設定する場合は、支出上限を 0 USD のままにしておくことができます。これにより、アカウントの支出制限は 0 USD になりますが、[後で適用] できます。
広告組織	広告を購入する組織。
請求センター	請求書を受け取る請求センターを選択します。ビジネスヘルプセンターの手順に従って、最初にビジネスを申請したときに自動的に作成された請求書を使用するか、別の請求センターを追加できます。
通貨	お使いの通貨を選択します。
Spend Cap	請求タイプとして [広告掲載申し込み] を選択した場合は、広告アカウントの予算を入力します。
[タイムゾーン]	タイムゾーンを選択します。

4. [アカウントの作成] を選択します。広告アカウントが作成され、広告マネージャの広告アカウント部分に表示されます。広告の起動を開始するには、支払い方法を入力できます。広告アカウントにメンバーを追加することもできます。
5. 既存の支払いを使用するか、新しい支払いを作成するかを選択します。次に、[支払い方法の保存] を選択します。
6. ビジネスに招待したメンバーを選択して、広告アカウントに追加します。割り当てることができるロールとアクセス許可の詳細については、「Roles and Permissions Overview」を参照してください。追加したメンバーは、広告マネージャにログインしてこの広告アカウントにアクセスできます。完了したら、メンバーを保存します。

広告アカウントの詳細については、「https://businesshelp.snapchat.com/s/article/roles-permissions?language=en_UShttps://businesshelp.snapchat.com/s/article/roles-permissions?language=en_US」を参照してください。

Snapchat 広告アカウントでのアプリの作成

Snapchat の Marketing API へのアクセスを有効にするには、ビジネスアカウントが設定されていることを確認してください。次に、下記のステップを実行します。

1. 広告マネージャにログインします。次に、左上隅のメニューを選択し、[ビジネスダッシュボード] を選択し、[ビジネスの詳細] を選択します。
2. [+OAuth アプリ] を選択します。
3. アプリ名を入力し、次の URL を Snap Redirect URI `https://<aws-region>.console.aws.amazon.com/gluestudio/oauth` として追加します。例えば、us-west-1 リージョンを使用する場合、URL は `https://us-west-1.console.aws.amazon.com/gluestudio/oauth) and choose` **`Create OAuth App`** になります。[OAuth アプリの作成] を選択します。
4. アプリの認証情報 (クライアント ID とクライアントシークレット) が表示されます。認証情報を保存します。これは接続を作成する際に必要になります。

AWS Glue Studio での Snowflake への接続

> **Note**
>
> AWS Glue for Spark を使用して、AWS Glue 4.0 以降のバージョンの Snowflake のテーブルからの読み取りとテーブルへの書き込みを行うことができます。AWS Glue ジョブで

Snowflake 接続をプログラム的に設定する方法については、「Redshift 接続」を参照してください。

AWS Glue には、Snowflake に対するサポートが組み込まれています。AWS Glue Studio には、Snowflake に接続してデータ統合ジョブをオーサリングし、AWS Glue Studio の Spark ランタイム上でジョブをサーバーレスで実行できる、ビジュアルインターフェイスが用意されています。

AWS Glue Studio は Snowflake の統合接続を作成します。詳細については、「考慮事項」を参照してください。

トピック

- Snowflake 接続の作成
- Snowflake ソースノードの作成
- Snowflake ターゲットノードの作成
- 詳細オプション

Snowflake 接続の作成

> **Note**
>
> 統合接続 (接続 v2) は、基本的な認証認証情報の USERNAME、PASSWORD キーを使用するようにすべての接続を標準化します。sfUser、sfPassword を含むシークレットを使用して、API 経由で v1 接続を作成することもできます。

AWS Glue Studio で データソース - Snowflake ノードを追加するには、既存の AWS Glue Snowflake 接続を選択するか、新しい接続を作成します。Snowflake に接続するように設定された JDBC タイプの接続ではなく、SNOWFLAKE タイプの接続を選択する必要があります。次の手順に従って、AWS Glue Snowflake 接続を作成します。

Snowflake 接続を作成するには

1. Snowflake でユーザー *snowflakeUser* とパスワード *snowflakePassword* を作成します。
2. このユーザーが操作する Snowflake ウェアハウス *snowflakeWarehouse* を決定します。これを Snowflake の *snowflakeUser* の DEFAULT_WAREHOUSE として設定するか、次のステップのために覚えておきます。

3. AWS Secrets Manager で、Snowflake の認証情報を使用してシークレットを作成します。Secrets Manager でシークレットを作成するには、AWS Secrets Manager ドキュメントの「AWS Secrets Manager シークレットを作成する」にあるチュートリアルに従ってください。シークレットを作成したら、次のステップのためにシークレット名 *secretName* を保存しておきます。

   - [キー/値のペア] をクリックして、sfUser キーを使用して *snowflakeUser* のペアを作成します。
   - [キー/値のペア] をクリックして、sfPassword キーを使用して *snowflakePassword* のペアを作成します。
   - [キー/値のペア] をクリックして、sfWarehouse キーを使用して *snowflakeWarehouse* のペアを作成します。Snowflake にデフォルト設定がある場合、これは必要ありません。

4. AWS Glue データカタログで、「AWS Glue 接続の追加」にある手順に従って接続を作成します。接続を作成したら、次のステップのために接続名 *connectionName* を保存しておきます。

   - [接続タイプ] をクリックして、[Snowflake] を選択します。
   - [Snowflake URL] をクリックして、Snowflake インスタンスのホスト名を指定します。URL では、*account_identifier*.snowflakecomputing.com という形式のホスト名を使用します。
   - [AWS Secret] をクリックして、*secretName* を入力します。

Snowflake ソースノードの作成

必要となる許可

Snowflake データソースを使用する AWS Glue Studio ジョブには追加の許可が必要です。ETL ジョブに許可を追加する方法の詳細については、「ETL ジョブに必要な IAM アクセス許可を確認する」を参照してください。

SNOWFLAKE AWS Glue 接続は、AWS Secrets Manager シークレットを使用して認証情報を提供します。AWS Glue Studio のジョブおよびデータプレビューのロールには、このシークレットを読み取る許可が必要です。

Snowflake データソースの追加

前提条件:

- Snowflake 認証情報用の AWS Secrets Manager シークレット

- Snowflake タイプの AWS Glue データカタログ接続

データソース - Snowflake ノードを追加するには

1. Snowflake データソースの接続を選択します。既に接続が存在していて、それらの接続から選択できることが前提です。接続の作成が必要な場合は、[Create Snowflake connection] をクリックします。詳細については、「コネクタと接続の使用に関する概要」を参照してください。

   接続を選択したあとは、[プロパティを表示] をクリックすると、接続のプロパティを表示できます。URL、セキュリティグループ、サブネット、アベイラビリティーゾーン、説明、作成時刻 (UTC) と最終更新時刻 (UTC) のタイムスタンプなど、接続についての情報が表示されます。
2. Snowflake ソースオプションを選択します。

   - [Choose a single table] - これは、単一の Snowflake テーブルからアクセスするデータを含むテーブルです。
   - [Enter custom query] - カスタムクエリに基づき、Snowflake の複数のテーブルからデータセットにアクセスできます。
3. 単一のテーブルを選択した場合は、Snowflake スキーマの名前を入力します。

   または、[カスタムクエリを入力] を選択します。このオプションを選択すると、Snowflake の複数のテーブルからカスタムデータセットにアクセスできます。このオプションを選択した場合は、Snowflake クエリを入力します。
4. [パフォーマンスとセキュリティ] (オプション) で、次を設定します。

   - [クエリプッシュダウンを有効化] - Snowflake インスタンスに作業をオフロードするかどうかを選択します。
5. [カスタムの Snowflake プロパティ] (オプション) で、必要に応じてパラメータと値を入力します。

Snowflake ターゲットノードの作成

必要となる許可

Snowflake データソースを使用する AWS Glue Studio ジョブには追加の許可が必要です。ETL ジョブに許可を追加する方法の詳細については、「ETL ジョブに必要な IAM アクセス許可を確認する」を参照してください。

SNOWFLAKE AWS Glue 接続は、AWS Secrets Manager シークレットを使用して認証情報を提供します。AWS Glue Studio のジョブおよびデータプレビューのロールには、このシークレットを読み取る許可が必要です。

Snowflake データターゲットの追加

Snowflake ターゲットノードを作成するには

1. 既存の Snowflake テーブルをターゲットとして選択するか、新しいテーブル名を入力します。
2. データターゲット - Snowflake ターゲットノードを使用する場合、次のオプションから選択できます。

   - APPEND — テーブルがすでに存在している場合は、新しいデータをテーブルにすべて挿入してダンプします。テーブルが存在しない場合は、新規にテーブルを作成し、そこに新しいデータをすべて挿入します。
   - MERGE — 指定した条件に基づいて、AWS Glue がターゲットとなるテーブルのデータを更新するか、またはデータを追加します。

     [オプション] を選択します。

     - キーと簡単なアクションの選択 — ソースデータとターゲットデータセットとの、マッチングキーとして使用する列を選択します。

       一致した場合、次のオプションを指定します。

       - ターゲットのデータセットにあるレコードを、ソースのデータで更新します。
       - ターゲットのデータセットにあるレコードを削除します。

       一致しない場合、次のオプションを指定します。

       - ターゲットのデータセットに、新しい行としてソースデータを挿入します。
       - 何もしない。
     - カスタム MERGE ステートメントの入力 — その後 [MERGE ステートメントの検証] を選択し、ステートメントが有効か、無効かを検証できます。
   - TRUNCATE — 既にテーブルが存在している場合は、ターゲットのテーブルの内容を削除してから、ターゲットのテーブルを削除します。削除が成功してから、すべてのデータを挿入します。テーブルが存在していない場合、テーブルを作成し、すべてのデータを挿入します。削除が成功しなかった場合、操作は失敗します。
   - DROP — 既にテーブルが存在している場合は、テーブルのメタデータとデータを削除します。削除が成功してから、すべてのデータを挿入します。テーブルが存在していない場合、

テーブルを作成し、すべてのデータを挿入します。削除が成功しなかった場合、操作は失敗します。

詳細オプション

「AWS Glue デベロッパーガイド」の「Snowflake connections」を参照してください。

AWS Glue Studio での Stripe への接続

Stripe は、ビジネス向けのオンライン支払い処理およびクレジットカード処理のプラットフォームです。Stripe プラットフォームを使用すると、企業はオンライン支払いを受け取り、電子商取引のサブスクリプション (定期請求) を作成し、支払いを受け取るようにアカウントを設定できます。Stripe は、マルチパーティー支払いにも対応しており、企業がマーケットプレイスをセットアップして支払いを回収し、「接続済み」アカウントを通じて販売者またはサービスプロバイダーに支払うことができます。

トピック

- AWS Glue による Stripe サポート
- 接続を作成および使用するための API オペレーションを含むポリシー
- Stripe の設定
- Stripe 接続の設定
- Stripe エンティティからの読み取り
- Stripe 接続オプション
- 制限
- 新しい Stripe アカウントの作成とクライアントアプリの設定

AWS Glue による Stripe サポート

AWS Glue は、Stripe を次のようにサポートします:

ソースとしてサポートされていますか?

はい。AWS Glue ETL ジョブを使用して、Stripe からデータをクエリできます。

ターゲットとしてサポートされているかどうか

いいえ。

サポートされている Slack API のバージョン

v1。

接続を作成および使用するための API オペレーションを含むポリシー

次のサンプル ポリシーで、接続の作成と使用に必要な IAM 権限について説明します。新しいロールを作成する場合は、以下を含むポリシーを作成します。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:ListConnectionTypes",
                "glue:DescribeConnectionType",
                "glue:RefreshOAuth2Tokens"
                "glue:ListEntities",
                "glue:DescribeEntity"
            ],
            "Resource": "*"
        }
    ]
}
```

また、次のマネージド IAM ポリシーを使用して、アクセスを許可できます。

- AWSGlueServiceRole – さまざまな AWS Glue プロセスを実行するために必要なリソースへのアクセス権をユーザーに代わって付与します。これらのリソースには、AWS Glue、Amazon S3、IAM、Amazon CloudWatch Logs および Amazon EC2 が含まれます。このポリシーで指定されたリソースの命名規則に従った場合、AWS Glue プロセスは必要なアクセス権限を使用できます。このポリシーは、通常、クローラ、ジョブ、開発エンドポイントを定義するときに指定されたロールにアタッチされます。
- AWSGlueConsoleFullAccess — ポリシーがアタッチされているアイデンティティが AWS マネージメントコンソールを使用するときは、AWS Glue リソースへのフルアクセスを許可します。このポリシーで指定されたリソースの命名規則に従った場合、ユーザーは完全なコンソール機能を使用できます。このポリシーは、通常 AWS Glue コンソールのユーザーにアタッチされています。

Stripe の設定

AWS Glue を使用して Stripe からデータを転送できるようにするには、次の要件を満たす必要があります:

最小要件

- E メールとパスワードが設定されている Stripe アカウントを持っていること。詳細については、「新しい Stripe アカウントの作成とクライアントアプリの設定」を参照してください。
- Stripe アカウントで API アクセスが有効になっていること。Stripe API の使用はすべて追加料金なしで利用できます。

これらの要件を満たすと、Stripe アカウントに AWS Glue を接続する準備が整います。

Stripe 接続の設定

Stripe はカスタム認証をサポートしています。カスタム認可に必要な API キーの生成の詳細については、STRIPE REST API のドキュメントを参照してください。

Stripe 接続を設定するには:

1. AWS Secrets Manager で、次の詳細を含むシークレットを作成します。AWS Glue で接続ごとにシークレットを作成する必要があります。
   a. カスタマーマネージド接続アプリケーションの場合 – シークレットには、USER_MANAGED_CLIENT_APPLICATION_CLIENT_SECRET を使用して接続されたアプリケーションのコンシューマーシークレットをキーとして含める必要があります。
2. AWS Glue Glue Studio で、以下の手順に従って [データ接続] の下に接続を作成します。
   a. [接続タイプ] を選択する際に、[Stripe] を選択します。
   b. AWS Glue が次のアクションを担い、その権限を持つことができる IAM ロールを選択します。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:DescribeSecret",
                "secretsmanager:GetSecretValue",
                "secretsmanager:PutSecretValue",
```

```
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterface",
                "ec2:DeleteNetworkInterface",
            ],
            "Resource": "*"
        }
    ]
}
```

c. AWS Glue でこの接続に使用する `secretName` を選択して、トークンを配置します。

d. ネットワークを使用する場合は、ネットワークオプションを選択します。

3. AWS Glue ジョブに関連付けられている IAM ロールに `secretName` を読み取るアクセス許可を付与します。

Stripe エンティティからの読み取り

前提条件

- 読み取り元の Stripe オブジェクト。

サポートされているエンティティ

エンティティ	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
Balance	なし	いいえ	いいえ	はい	なし
Balance Transactions	あり	はい	いいえ	はい	はい
料金	あり	はい	いいえ	はい	はい
Disputes	あり	はい	いいえ	はい	はい
File Links	あり	はい	いいえ	はい	はい
PaymentIntents	あり	はい	いいえ	はい	はい

エンティティ	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
SetupIntents	あり	はい	いいえ	はい	はい
Payouts	あり	はい	いいえ	はい	はい
Refunds	あり	はい	いいえ	はい	はい
製品	あり	はい	いいえ	はい	はい
Prices	あり	はい	いいえ	はい	はい
Coupons	あり	はい	いいえ	はい	はい
Promotion Codes	あり	はい	いいえ	はい	はい
Tax Codes	いいえ	はい	いいえ	はい	なし
Tax Rates	あり	はい	いいえ	はい	はい
Shipping Rates	あり	はい	いいえ	はい	はい
セッション	あり	はい	いいえ	はい	はい
Credit Notes	あり	はい	いいえ	はい	はい
お客様	あり	はい	いいえ	はい	はい
請求書	あり	はい	いいえ	はい	はい
Invoice Items	あり	はい	いいえ	はい	なし
プラン	あり	はい	いいえ	はい	はい
Quotes	あり	はい	いいえ	はい	なし

エンティティ	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
サブスクリプション	あり	はい	いいえ	はい	
Subscription Items	いいえ	はい	いいえ	はい	なし
Subscription Schedules	あり	はい	いいえ	はい	はい
アカウント	いいえ	はい	いいえ	はい	はい
Application Fees	あり	はい	いいえ	はい	はい
Country Specs	いいえ	はい	いいえ	はい	なし
転送	あり	はい	いいえ	はい	はい
Early Fraud Warnings	あり	はい	いいえ	はい	はい
レポートタイプ	なし	いいえ	いいえ	はい	なし

例

```
stripe_read = glueContext.create_dynamic_frame.from_options(
    connection_type="stripe",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "coupons",
        "API_VERSION": "v1"
    }
)
```

ストライプエンティティとフィールドの詳細

エンティティ	フィールド	データタイプ	サポートされる演算子
Balance			
	使用可能	リスト	
	connect_reserved	リスト	
	保留中	リスト	
	livemode	ブール値	
	オブジェクト	文字列	
	instant_available	リスト	
	issuing	Struct	
Balance Transactions			
	id	文字列	
	オブジェクト	文字列	
	amount	整数	
	available_on	DateTime	=, >=, <=,<,>
	作成済み	DateTime	=, >=, <=,<,>
	currency	文字列	
	description	文字列	
	exchange_rate	BigDecimal	
	fee	整数	
	fee_details	リスト	

エンティティ	フィールド	データタイプ	サポートされる演算子
	net	整数	
	reporting_category	文字列	
	ソース	文字列	=
	status	文字列	
	type	文字列	=
	cross_border_classification	文字列	
料金			
	id	文字列	
	オブジェクト	文字列	
	amount	整数	=, <, >
	amount_captured	整数	
	amount_refunded	整数	
	アプリケーション	文字列	
	application_fee	文字列	
	application_fee_amount	整数	
	balance_transaction	文字列	
	billing_details	Struct	
	calculated_statement_descriptor	文字列	

エンティティ	フィールド	データタイプ	サポートされる演算子
	captured	ブール値	
	作成済み	DateTime	=, >=, <=,<,>
	currency	文字列	
	カスタマー	文字列	=
	description	文字列	
	宛先	文字列	
	dispute	文字列	
	disputed	ブール値	=
	failure_balance_transaction	文字列	
	failure_code	文字列	
	failure_message	文字列	
	fraud_details	Struct	
	請求書	文字列	
	livemode	ブール値	
	metadata	Struct	
	on_behalf_of	文字列	
	order	文字列	
	outcome	Struct	
	paid	ブール値	

エンティティ	フィールド	データタイプ	サポートされる演算子
	payment_intent	文字列	=
	payment_method	文字列	
	payment_method_details	Struct	
	receipt_email	文字列	
	receipt_number	文字列	
	receipt_url	文字列	
	refunded	ブール値	=
	返金	Struct	
	確認	文字列	
	shipping	Struct	
	ソース	Struct	
	source_transfer	文字列	
	statement_descriptor	文字列	
	statement_descriptor_suffix	文字列	
	status	文字列	
	移管	文字列	
	transfer_data	Struct	
	transfer_group	文字列	=
Disputes			

エンティティ	フィールド	データタイプ	サポートされる演算子
	id	文字列	
	オブジェクト	文字列	
	amount	整数	=, <, >
	balance_transaction	文字列	
	balance_transactions	リスト	
	料金	文字列	=
	作成済み	DateTime	=, >=, <=,<,>
	currency	文字列	
	evidence	Struct	
	evidence_details	Struct	
	is_charge_refundable	ブール値	
	livemode	ブール値	
	metadata	Struct	
	payment_intent	文字列	=
	理由	文字列	=
	status	文字列	
	payment_method_details	Struct	
File Links			
	id	文字列	

エンティティ	フィールド	データタイプ	サポートされる演算子
	オブジェクト	文字列	
	作成済み	DateTime	=, >=, <=,<,>
	expired	ブール値	=
	expires_at	DateTime	
	file	文字列	=
	livemode	ブール値	
	metadata	Struct	
	url	文字列	
PaymentIntents			
	id	文字列	
	オブジェクト	文字列	
	amount	整数	
	amount_capturable	整数	
	amount_details	Struct	
	amount_received	整数	
	アプリケーション	文字列	
	application_fee_amount	整数	
	automatic_payment_methods	Struct	
	cancelled_at	DateTime	

エンティティ	フィールド	データタイプ	サポートされる演算子
	cancellation_reason	文字列	
	capture_method	文字列	
	client_secret	文字列	
	confirmation_method	文字列	
	作成済み	DateTime	=, >=, <=,<,>
	currency	文字列	
	カスタマー	文字列	=
	description	文字列	
	請求書	文字列	
	last_payment_error	Struct	
	latest_charge	文字列	
	livemode	ブール値	
	metadata	Struct	
	next_action	Struct	
	on_behalf_of	文字列	
	payment_method	文字列	
	payment_method_options	Struct	
	payment_method_types	リスト	

エンティティ	フィールド	データタイプ	サポートされる演算子
	payment_method_configuration_details	Struct	
	処理	Struct	
	receipt_email	文字列	
	確認	文字列	
	setup_future_usage	文字列	
	shipping	Struct	
	ソース	文字列	
	statement_descriptor	文字列	
	statement_descriptor_suffix	文字列	
	status	文字列	
	transfer_data	Struct	
	transfer_group	文字列	
SetupIntents			
	id	文字列	
	オブジェクト	文字列	
	アプリケーション	文字列	
	cancellation_reason	文字列	
	client_secret	文字列	
	作成済み	DateTime	=, >=, <=,<,>

エンティティ	フィールド	データタイプ	サポートされる演算子
	カスタマー	文字列	=
	description	文字列	
	flow_directions	リスト	
	last_setup_error	Struct	
	latest_attempt	文字列	
	livemode	ブール値	
	mandate	文字列	
	metadata	Struct	
	next_action	Struct	
	on_behalf_of	文字列	
	payment_method	文字列	
	payment_method_options	Struct	
	payment_method_types	リスト	
	single_use_mandate	文字列	
	status	文字列	
	使用	文字列	
	automatic_payment_methods	Struct	
Payouts			

エンティティ	フィールド	データタイプ	サポートされる演算子
	id	文字列	
	オブジェクト	文字列	
	amount	整数	=, <, >
	arrival_date	DateTime	=, >=, <=,<,>
	automatic	ブール値	
	balance_transaction	文字列	
	作成済み	DateTime	=, >=, <=,<,>
	currency	文字列	
	description	文字列	=
	宛先	文字列	
	failure_balance_transaction	文字列	
	failure_code	文字列	
	failure_message	文字列	
	livemode	ブール値	
	metadata	Struct	
	method	文字列	
	original_payout	文字列	
	reversed_by	文字列	
	reconciliation_status	文字列	

エンティティ	フィールド	データタイプ	サポートされる演算子
	source_type	文字列	
	statement_descriptor	文字列	
	status	文字列	
	type	文字列	
	application_fee	文字列	
	application_fee_amount	整数	
Refunds			
	id	文字列	
	オブジェクト	文字列	
	amount	整数	
	balance_transaction	文字列	
	料金	文字列	=
	作成済み	DateTime	=, >=, <=,<,>
	currency	文字列	
	metadata	Struct	
	destination_details	Struct	
	payment_intent	文字列	=
	理由	文字列	
	receipt_number	文字列	

エンティティ	フィールド	データタイプ	サポートされる演算子
	source_transfer_reversal	文字列	
	status	文字列	
	transfer_reversal	文字列	
製品			
	id	文字列	
	オブジェクト	文字列	
	ACTIVE	ブール値	=
	attributes	リスト	
	作成済み	DateTime	=, >=, <=,<,>
	default_price	文字列	
	description	文字列	
	images	リスト	
	livemode	ブール値	
	metadata	Struct	
	名前	文字列	
	package_dimensions	Struct	
	shippable	ブール値	
	statement_descriptor	文字列	
	tax_code	文字列	

エンティティ	フィールド	データタイプ	サポートされる演算子
	type	文字列	=
	unit_label	文字列	
	更新済み	DateTime	
	url	文字列	
	機能	リスト	
Prices			
	id	文字列	
	オブジェクト	文字列	
	ACTIVE	ブール値	=
	billing_scheme	文字列	
	作成済み	DateTime	=, >=, <=,<,>
	currency	文字列	=
	custom_unit_amount	Struct	
	livemode	ブール値	
	lookup_key	文字列	
	metadata	Struct	
	nickname	文字列	
	product	文字列	=
	recurring	Struct	
	tax_behavior	文字列	

エンティティ	フィールド	データタイプ	サポートされる演算子
	tiers_mode	文字列	
	transform_quantity	Struct	
	type	文字列	=
	unit_amount	整数	
	unit_amount_decimal	文字列	
Coupons			
	ID	文字列	
	オブジェクト	文字列	
	amount_off	整数	
	作成済み	DateTime	=, >=, <=,<,>
	currency	文字列	=
	duration	文字列	=
	duration_in_months	整数	=,<,>
	livemode	ブール値	
	max_redemptions	整数	=, <, >
	metadata	Struct	
	名前	文字列	
	percent_off	ダブル	=
	redeem_by	DateTime	=, >=, <=, <, >
	times_redeemed	整数	

エンティティ	フィールド	データタイプ	サポートされる演算子
	valid	ブール値	
Promotion Codes			
	ID	文字列	
	オブジェクト	文字列	
	ACTIVE	ブール値	=
	コード	文字列	=
	coupon	Struct	
	作成済み	DateTime	=,>=,<=,<,>
	カスタマー	文字列	
	expires_at	DateTime	
	livemode	ブール値	
	max_redemptions	整数	
	metadata	Struct	
	制限	Struct	
	times_redeemed	整数	
Tax Codes			
	ID	文字列	
	オブジェクト	文字列	
	description	文字列	
	名前	文字列	

エンティティ	フィールド	データタイプ	サポートされる演算子
Tax Rates			
	ID	文字列	
	オブジェクト	文字列	
	ACTIVE	ブール値	=
	country	文字列	
	作成済み	DateTime	=, >=, <=, <, >
	説明	文字列	
	display_name	文字列	
	inclusive	ブール値	=
	jurisdiction	文字列	
	jurisdiction_level	文字列	
	livemode	ブール値	
	metadata	Struct	
	割合(%)	ダブル	
	effective_percentage	ダブル	
	state	文字列	
	tax_type	文字列	
Shipping Rates			
	ID	文字列	
	オブジェクト	文字列	

エンティティ	フィールド	データタイプ	サポートされる演算子
	ACTIVE	ブール値	=
	作成済み	DateTime	=, >=, <=, <, >
	delivery_estimate	Struct	
	display_name	文字列	
	fixed_amount	Struct	
	livemode	ブール値	
	metadata	Struct	
	tax_behavior	文字列	
	tax_code	文字列	
	type	文字列	
セッション			
	id	文字列	
	オブジェクト	文字列	
	after_expiration	Struct	
	allow_promotion_codes	ブール値	
	amount_subtotal	整数	
	amount_total	整数	
	automatic_tax	Struct	
	billing_address_collection	文字列	

エンティティ	フィールド	データタイプ	サポートされる演算子
	cancel_url	文字列	
	client_reference_id	文字列	
	consent	Struct	
	consent_collection	Struct	
	作成済み	DateTime	=, >=, <=, <, >
	currency	文字列	
	custom_text	Struct	
	カスタマー	文字列	
	customer_creation	文字列	
	customer_details	Struct	
	customer_email	文字列	
	expires_at	DateTime	
	請求書	文字列	
	invoice_creation	Struct	
	livemode	ブール値	
	サイト	文字列	
	metadata	Struct	
	モード	文字列	
	payment_intent	文字列	=
	payment_link	文字列	

エンティティ	フィールド	データタイプ	サポートされる演算子
	payment_method_collection	文字列	
	payment_method_options	Struct	
	payment_method_types	リスト	
	payment_status	文字列	
	phone_number_collection	Struct	
	recovered_from	文字列	
	setup_intent	文字列	
	shipping_address_collection	Struct	
	shipping_cost	Struct	
	shipping_details	Struct	
	shipping_options	リスト	
	status	文字列	
	submit_type	文字列	
	サブスクリプション	文字列	
	success_url	文字列	
	tax_id_collection	Struct	
	total_details	Struct	

エンティティ	フィールド	データタイプ	サポートされる演算子
	url	文字列	
	ui_mode	文字列	
Credit Notes			
	id	文字列	
	オブジェクト	文字列	
	amount	整数	
	作成済み	DateTime	=, >=, <=, <, >
	currency	文字列	
	カスタマー	文字列	=
	customer_balance_transaction	文字列	
	discount_amount	整数	
	discount_amounts	リスト	
	請求書	文字列	=
	lines	Struct	
	livemode	ブール値	
	memo	文字列	
	metadata	Struct	
	数値	文字列	
	out_of_band_amount	整数	

エンティティ	フィールド	データタイプ	サポートされる演算子
	pdf	文字列	
	理由	文字列	
	refund	文字列	
	status	文字列	
	subtotal	整数	
	subtotal_excluding_tax	整数	
	tax_amounts	リスト	
	total	整数	
	total_excluding_tax	整数	
	type	文字列	
	voided_at	DateTime	
	amount_shipping	整数	
	effective_at	DateTime	
	shipping_cost	Struct	
お客様			
	id	文字列	
	オブジェクト	文字列	
	アドレス	Struct	
	balance	整数	

エンティティ	フィールド	データタイプ	サポートされる演算子
	作成済み	DateTime	
	currency	文字列	=, >=, <=, <, >
	default_source	文字列	
	delinquent	ブール値	=
	説明	文字列	
	discount	Struct	
	email	文字列	=
	invoice_prefix	文字列	
	invoice_settings	Struct	
	livemode	ブール値	
	metadata	Struct	
	名前	文字列	
	next_invoice_sequence	整数	
	phone	文字列	
	preferred_locales	リスト	
	shipping	Struct	
	tax_exempt	文字列	
	test_clock	文字列	
請求書			

エンティティ	フィールド	データタイプ	サポートされる演算子
	id	文字列	
	オブジェクト	文字列	
	account_country	文字列	
	account_name	文字列	
	account_tax_ids	リスト	
	amount_due	整数	
	amount_paid	整数	
	amount_remaining	整数	
	アプリケーション	文字列	
	application_fee_amount	整数	
	attempt_count	整数	
	attempted	ブール値	=
	auto_advance	ブール値	=
	automatic_tax	Struct	
	billing_reason	文字列	
	料金	文字列	
	collection_method	文字列	=
	作成済み	DateTime	=, >=, <=, <, >
	currency	文字列	

エンティティ	フィールド	データタイプ	サポートされる演算子
	custom_fields	リスト	
	カスタマー	文字列	=
	customer_address	Struct	
	customer_email	文字列	
	customer_name	文字列	
	customer_phone	文字列	
	customer_shipping	Struct	
	customer_tax_exempt	文字列	
	customer_tax_ids	リスト	
	default_payment_method	文字列	
	default_source	文字列	
	default_tax_rates	リスト	
	説明	文字列	
	discount	Struct	
	discounts	リスト	
	due_date	DateTime	=, >=, <=, <, >
	ending_balance	整数	
	footer	文字列	
	from_invoice	Struct	

エンティティ	フィールド	データタイプ	サポートされる演算子
	hosted_invoice_url	文字列	
	invoice_pdf	文字列	
	last_finalization_error	Struct	
	latest_revision	文字列	
	lines	Struct	
	livemode	ブール値	
	metadata	Struct	
	next_payment_attempt	DateTime	
	数値	文字列	
	on_behalf_of	文字列	
	paid	ブール値	=
	paid_out_of_band	ブール値	
	payment_intent	文字列	
	payment_settings	Struct	
	period_end	DateTime	=, >=, <=, <, >
	period_start	DateTime	=, >=, <=, <, >
	post_payment_credit_notes_amount	整数	
	pre_payment_credit_notes_amount	整数	

エンティティ	フィールド	データタイプ	サポートされる演算子
	quote	文字列	
	receipt_number	文字列	
	rendering	Struct	
	rendering_options	Struct	
	starting_balance	整数	
	statement_descriptor	文字列	
	status	文字列	=
	status_transitions	Struct	
	サブスクリプション	文字列	
	subscription_details	Struct	
	subtotal	整数	=, <, >
	subtotal_excluding_tax	整数	
	tax	整数	
	test_clock	文字列	
	total	整数	=, <, >
	total_discount_amounts	リスト	
	total_excluding_tax	整数	
	total_tax_amounts	リスト	
	transfer_data	Struct	

エンティティ	フィールド	データタイプ	サポートされる演算子
	webhooks_delivered_at	DateTime	
	automatically_finalizes_at	DateTime	
	effective_at	DateTime	
	発行者	Struct	
Invoice Items			
	id	文字列	
	オブジェクト	文字列	
	amount	整数	=, <, >
	currency	文字列	
	カスタマー	文字列	=
	date	DateTime	
	説明	文字列	
	discountable	ブール値	
	discounts	リスト	
	請求書	文字列	=
	livemode	ブール値	
	metadata	Struct	
	period	Struct	
	計画	Struct	

エンティティ	フィールド	データタイプ	サポートされる演算子
	price	Struct	
	proration	ブール値	=
	数量	整数	
	サブスクリプション	文字列	
	subscription_item	文字列	
	tax_rates	リスト	
	test_clock	文字列	
	unit_amount	整数	
	unit_amount_decimal	文字列	
プラン			
	id	文字列	
	オブジェクト	文字列	
	ACTIVE	ブール値	=
	aggregate_usage	文字列	
	amount	整数	
	amount_decimal	文字列	
	billing_scheme	文字列	
	作成済み	DateTime	=, >=, <=, <, >
	currency	文字列	=
	interval	文字列	=

エンティティ	フィールド	データタイプ	サポートされる演算子
	interval_count	整数	
	livemode	ブール値	
	metadata	Struct	
	nickname	文字列	
	product	文字列	=
	tiers_mode	文字列	
	transform_usage	Struct	
	trial_period_days	整数	=, <, >
	usage_type	文字列	
	測定	文字列	
Quotes			
	id	文字列	
	オブジェクト	文字列	
	amount_subtotal	整数	
	amount_total	整数	
	アプリケーション	文字列	
	application_fee_amount	整数	
	application_fee_percent	ダブル	
	automatic_tax	Struct	

エンティティ	フィールド	データタイプ	サポートされる演算子
	collection_method	文字列	
	computed	Struct	
	作成済み	DateTime	
	currency	文字列	
	カスタマー	文字列	=
	default_tax_rates	リスト	
	説明	文字列	
	discounts	リスト	
	expires_at	DateTime	
	footer	文字列	
	from_quote	Struct	
	ヘッダー	文字列	
	請求書	文字列	
	invoice_settings	Struct	
	livemode	ブール値	
	metadata	Struct	
	数値	文字列	
	on_behalf_of	文字列	
	status	文字列	=
	status_transitions	Struct	

エンティティ	フィールド	データタイプ	サポートされる演算子
	サブスクリプション	文字列	
	subscription_data	Struct	
	subscription_schedule	文字列	
	test_clock	文字列	
	total_details	Struct	
	transfer_data	Struct	
サブスクリプション			
	id	文字列	
	オブジェクト	文字列	
	アプリケーション	文字列	
	application_fee_percent	ダブル	
	automatic_tax	Struct	
	billing_cycle_anchor	DateTime	
	billing_thresholds	Struct	
	cancel_at	DateTime	
	cancel_at_period_end	ブール値	
	cancelled_at	DateTime	
	collection_method	文字列	=
	作成済み	DateTime	=, >=, <=,<,>

エンティティ	フィールド	データタイプ	サポートされる演算子
	currency	文字列	
	current_period_end	DateTime	=, >=, <=
	current_period_start	DateTime	=, >=, <=
	カスタマー	文字列	=
	days_until_due	整数	
	default_payment_method	文字列	
	default_source	文字列	
	default_tax_rates	リスト	
	説明	文字列	
	discount	Struct	
	ended_at	DateTime	
	項目	Struct	
	latest_invoice	文字列	
	livemode	ブール値	
	metadata	Struct	
	next_pending_invoice_item_invoice	DateTime	
	pause_collection	Struct	
	payment_settings	Struct	

エンティティ	フィールド	データタイプ	サポートされる演算子
	pending_invoice_item_interval	Struct	
	pending_setup_intent	文字列	
	pending_update	Struct	
	計画	Struct	
	数量	整数	
	schedule	文字列	
	start_date	DateTime	
	status	文字列	=
	test_clock	文字列	
	transfer_data	Struct	
	trial_end	DateTime	
	trial_start	DateTime	
Subscription Items			
	ID	文字列	
	オブジェクト	文字列	
	billing_thresholds	Struct	
	作成済み	DateTime	=, >=, <=, <, >
	metadata	Struct	
	計画	Struct	

エンティティ	フィールド	データタイプ	サポートされる演算子
	price	Struct	
	サブスクリプション	文字列	
	tax_rates	リスト	
	discounts	リスト	
Subscription Schedules			
	オブジェクト	文字列	
	アプリケーション	文字列	
	cancelled_at	DateTime	
	completed_at	DateTime	
	作成済み	DateTime	
	current_phase	Struct	
	カスタマー	文字列	=
	default_settings	Struct	
	end_behavior	文字列	
	livemode	ブール値	
	metadata	Struct	
	phases	リスト	
	released_at	DateTime	
	released_subscription	文字列	

エンティティ	フィールド	データタイプ	サポートされる演算子
	renewal_interval	文字列	
	status	文字列	
	サブスクリプション	文字列	
	test_clock	文字列	
アカウント			
	details_submitted	ブール値	
	tos_acceptance	Struct	
	type	文字列	
	metadata	Struct	
	id	文字列	
	オブジェクト	文字列	
	default_currency	文字列	
	機能	Struct	
	charges_enabled	ブール値	
	設定	Struct	
	要件	Struct	
	payouts_enabled	ブール値	
	future_requirements	Struct	
	external_accounts	Struct	
	コントローラ	Struct	

エンティティ	フィールド	データタイプ	サポートされる演算子
	country	文字列	
	email	文字列	
	作成済み	DateTime	=, >=, <=, <, >
	business_profile	Struct	
	business_type	文字列	
	company	Struct	
Application Fees			
	id	文字列	
	オブジェクト	文字列	
	アカウント	文字列	
	amount	整数	=, <, >
	amount_refunded	整数	=, <, >
	アプリケーション	文字列	
	balance_transaction	文字列	
	料金	文字列	=
	作成済み	DateTime	
	currency	文字列	
	livemode	ブール値	
	originating_transaction	文字列	

エンティティ	フィールド	データタイプ	サポートされる演算子
	refunded	ブール値	=
	返金	Struct	
	fee_source	Struct	
Country Specs			
	id	文字列	
	オブジェクト	文字列	
	default_currency	文字列	
	supported_bank_account_currencies	Struct	
	supported_payment_currencies	リスト	
	supported_payment_methods	リスト	
	supported_transfer_countries	リスト	
	verification_fields	Struct	
転送			
	id	文字列	
	オブジェクト	文字列	
	amount	整数	=, <, >
	amount_reversed	整数	

エンティティ	フィールド	データタイプ	サポートされる演算子
	balance_transaction	文字列	
	作成済み	DateTime	=, >=, <=, <, >
	currency	文字列	=
	description	文字列	
	宛先	文字列	=
	destination_payment	文字列	
	livemode	ブール値	
	metadata	Struct	
	reversals	Struct	
	reversed	ブール値	
	source_transaction	文字列	
	source_type	文字列	
	transfer_group	文字列	=
Early Fraud Warnings			
	id	文字列	
	オブジェクト	文字列	
	actionable	ブール値	
	料金	文字列	=
	作成済み	DateTime	=, >=, <=, <, >
	fraud_type	文字列	

エンティティ	フィールド	データタイプ	サポートされる演算子
	livemode	ブール値	
	payment_intent	文字列	=
レポートタイプ			
	id	文字列	
	オブジェクト	文字列	
	data_available_end	DateTime	
	data_available_start	DateTime	
	default_columns	リスト	
	livemode	ブール値	
	名前	文字列	
	更新済み	DateTime	
	version	整数	

パーティショニングクエリ

Spark で同時実行を使用する場合は、追加の Spark オプション PARTITION_FIELD、LOWER_BOUND、UPPER_BOUND、および NUM_PARTITIONS を指定できます。これらのパラメータを使用すると、元のクエリは Spark タスクで同時に実行できるサブクエリの NUM_PARTITIONS の数に分割されます。

- PARTITION_FIELD: クエリのパーティション化に使用するフィールドの名前。
- LOWER_BOUND: 選択したパーティションフィールドの包括的な下限値。

  日付については、Spark SQL クエリで使用される Spark の日付形式を受け入れます。有効な値の例: "2024-07-01T00:00:00.000Z"。
- UPPER_BOUND: 選択したパーティションフィールドの排他的上限値。

- NUM_PARTITIONS: パーティション数。

エンティティごとのパーティション分割フィールドのサポートの詳細は、以下の表にまとめられています。

エンティティ名	Partitioning Field	データタイプ
Balance Transactions	作成済み	DateTime
料金	作成済み	DateTime
Disputes	作成済み	DateTime
File Links	作成済み	DateTime
PaymentIntents	作成済み	DateTime
SetupIntents	作成済み	DateTime
Payouts	作成済み	DateTime
Refunds	作成済み	DateTime
製品	作成済み	DateTime
Prices	作成済み	DateTime
Coupons	作成済み	DateTime
Promotion Codes	作成済み	DateTime
Tax Rates	作成済み	DateTime
Shipping Rates	作成済み	DateTime
セッション	作成済み	DateTime
Credit Notes	作成済み	DateTime
お客様	作成済み	DateTime

エンティティ名	Partitioning Field	データタイプ
請求書	作成済み	DateTime
プラン	作成済み	DateTime
サブスクリプション	作成済み	DateTime
Subscription Schedules	作成済み	DateTime
アカウント	作成済み	DateTime
Application Fees	作成済み	DateTime
転送	作成済み	DateTime
Early Fraud Warnings	作成済み	DateTime

例

```
stripe_read = glueContext.create_dynamic_frame.from_options(
    connection_type="stripe",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "coupons",
        "API_VERSION": "v1",
        "PARTITION_FIELD": "created"
        "LOWER_BOUND": "2024-05-01T20:55:02.000Z"
        "UPPER_BOUND": "2024-07-11T20:55:02.000Z"
        "NUM_PARTITIONS": "10"
    }
)
```

Stripe 接続オプション

Stripe の接続オプションは次のとおりです:

- ENTITY_NAME (文字列) - 読み取り / 書き込みに使用されます。Stripe のオブジェクトの名前。
- API_VERSION (文字列) - 読み取り / 書き込みに使用されます。使用する Stripe Rest API バージョン。例: v1。

- SELECTED_FIELDS(List<String>) - Default: empty(SELECT *). 読み込みに使用されます。オブジェクトに選択する列。
- FILTER_PREDICATE(文字列) - デフォルト: 空 読み込みに使用されます。Spark SQL 形式である必要があります。
- QUERY(文字列) - デフォルト: 空 読み込みに使用されます。完全な Spark SQL クエリ。
- PARTITION_FIELD (文字列) - 読み取りに使用されます。クエリをパーティション化するために使用するフィールド。
- LOWER_BOUND(文字列) - 読み取りに使用されます。選択したパーティションフィールドの包括的な下限値。
- UPPER_BOUND (文字列) - 読み取りに使用されます。選択したパーティションフィールドの排他的上限値。
- NUM_PARTITIONS(整数) - デフォルト: 1。読み込みに使用されます。読み取り用のパーティションの数。

制限

Stripe コネクタの制限は次のとおりです:

- コネクタがサポートするのはフィールドベースのパーティション分割のみです。
- レコードベースのパーティション分割はコネクタではサポートされておらず、レコードの合計数を取得するためのプロビジョニングはありません。
- プライマリキーのデータ型は文字列であるため、ID ベースのパーティション分割はコネクタでサポートされません。

新しい Stripe アカウントの作成とクライアントアプリの設定

Stripe アカウントの作成

1. リンク [https://dashboard.stripe.com/register] を選択します。
2. E メール、氏名、パスワードを入力し、[アカウントの作成] を選択します。
3. アカウントでログインしたら、[Gmail を開く] を選択してアカウントを検証します。
4. E メールで受信した検証リンクをクリックして、アカウントを確認します。
5. E メールアドレスの検証をクリックすると、別のページにリダイレクトされます。

6. [支払いのアクティブ化] をクリックしてアカウントをアクティブ化すると、[支払いのアクティブ化] ([https://dashboard.stripe.com/welcome]) ページにリダイレクトされます。必要な情報すべてを正しく入力してから、[続行] ボタンを選択します。

Slack 開発者アプリの作成

1. Stripe にログインします。
2. 下の図の一番上に表示される [開発者] を選択します。
3. [開発者] で [API キー]を選択します。
4. [テストキーを表示する] を選択して API キーを取得します。

AWS Glue Studio での Teradata Vantage に対する接続

AWS Glue は、Teradata Vantage のための組み込みサポートを提供します。AWS Glue Studio は、Teradata に接続してデータ統合ジョブをオーサリングし、AWS Glue Studio サーバーレス Spark ランタイム上でそれらのジョブを実行するためのビジュアルインターフェイスを提供します。

AWS Glue Studio は Teradata Vantage の統合接続を作成します。詳細については、「考慮事項」を参照してください。

トピック

- Teradata Vantage 接続の作成
- Teradata ソースノードの作成
- Teradata ターゲットノードの作成
- 詳細オプション

Teradata Vantage 接続の作成

AWS Glue から Teradata Vantage に接続するには、Teradata 認証情報を作成して AWS Secrets Manager シークレットに保存し、そのシークレットを AWS Glue Teradata 接続に関連付ける必要があります。

前提条件:

- Amazon VPC を通じて Teradata 環境にアクセスしている場合は、AWS Glue ジョブが Teradata 環境と通信できるように Amazon VPC を設定します。パブリックインターネット経由で Teradata 環境にアクセスすることは推奨されていません。

  Amazon VPC で、AWS Glue がジョブの実行中に使用する [VPC]、[サブネット]、および [セキュリティグループ] を特定または作成します。さらに、Teradata インスタンスとこの場所の間のネットワークトラフィックを許可するように Amazon VPC が設定されているようにする必要があります。ジョブでは、Teradata クライアントポートとの TCP 接続を確立する必要があります。Teradata ポートの詳細については、Teradata のドキュメントを参照してください。

  ネットワークレイアウトに応じて、安全な VPC 接続を実現するには、Amazon VPC および他のネットワークサービスの変更が必要な場合があります。AWS 接続の詳細については、Teradata ドキュメントの「AWS 接続オプション」を参照してください。

AWS Glue Teradata 接続を設定するには:

1. Teradata 設定で、AWS Glue が接続するユーザーとパスワード (*teradataUser* および *teradataPassword*) を識別または作成します。詳細については、Teradata ドキュメントの「Vantage セキュリティの概要」を参照してください。
2. AWS Secrets Manager で、Teradata 認証情報を使用してシークレットを作成します。Secrets Manager でシークレットを作成するには、AWS Secrets Manager ドキュメントの「AWS Secrets Manager シークレットを作成する」にあるチュートリアルに従ってください。シークレットを作成したら、次のステップのためにシークレット名 *secretName* を保存しておきます。

   - [key/value ペア] を選択する際に、*teradataUsername* という値を持つキー `user` のペアを作成します。
   - [key/value ペア] を選択する際に、*teradataPassword* という値を持つキー `password` のペアを作成します。
3. AWS Glue コンソールで、「the section called “AWS Glue 接続の追加”」にあるステップに従って接続を作成します。接続を作成したら、次のステップのために接続名 *connectionName* を保存しておきます。

   - [接続タイプ] を選択する際に、[Teradata] を選択します。
   - [JDBC URL] を入力する場合は、インスタンスの URL を入力します。JDBC URL に特定のカンマ区切りの接続パラメータをハードコーディ

ングすることもできます。URL は次の形式に準拠する必要があります:
`jdbc:teradata://teradataHostname/ParameterName=ParameterValue,ParameterName`

サポートされる URL パラメータには次が含まれます。

- DATABASE – デフォルトでアクセスするホスト上のデータベースの名前。
- DBS_PORT – データベースポート。非標準ポートで実行する場合に使用されます。

- [認証情報タイプ] を選択する場合は、[AWS Secrets Manager] を選択し、[AWS シークレット] を *secretName* に設定します。

4. 次の状況では、追加の設定が必要になる場合があります。

   - Amazon VPC の AWS でホストされている Teradata インスタンスの場合
     - Teradata セキュリティ認証情報を定義する AWS Glue 接続に、Amazon VPC 接続に関する情報を提供する必要があります。接続を作成または更新する際に、[ネットワークオプション] で [VPC]、[サブネット]、および [セキュリティグループ] を設定します。

Teradata ソースノードの作成

必要な前提条件

- 前のセクション the section called “Teradata Vantage 接続の作成” で説明したように、AWS Secrets Manager シークレットを使用して設定された AWS Glue Teradata Vantage 接続。
- 接続で使用されるシークレットを読み取るためのジョブに対する適切なアクセス許可。
- 読み取り元とする Teradata テーブル、*tableName*、またはクエリ *targetQuery*。

Teradata データソースの追加

[データソース — Teradata] ノードを追加するには:

1. Teradata データソース用の接続を選択します。すでに作成したので、ドロップダウンに表示されているはずです。接続を作成する必要がある場合は、[新しい接続を作成] を選択します。詳細については、前の「the section called “Teradata Vantage 接続の作成”」セクションを参照してください。

   接続を選択したあとは、[プロパティを表示] をクリックすると、接続のプロパティを表示できます。

2. [Teradata ソース] オプションを選択します。

- [単一のテーブルを選択] – 単一のテーブルからすべてのデータにアクセスします。
- [カスタムクエリを入力] — カスタムクエリに基づいて、複数のテーブルからデータセットにアクセスします。

3. 単一のテーブルを選択した場合は、*tableName* を入力します。

   [カスタムクエリを入力] を選択した場合は、SQL SELECT クエリを入力します。

4. [Teradata のカスタムプロパティ] で、必要に応じてパラメータと値を入力します。

Teradata ターゲットノードの作成

必要な前提条件

- 前のセクション the section called “Teradata Vantage 接続の作成” で説明したように、AWS Secrets Manager シークレットを使用して設定された AWS Glue Teradata Vantage 接続。
- 接続で使用されるシークレットを読み取るためのジョブに対する適切なアクセス許可。
- 書き込み先とする Teradata テーブルである *tableName*。

Teradata データターゲットの追加

[データターゲット — Teradata] ノードを追加するには:

1. Teradata データソース用の接続を選択します。すでに作成したので、ドロップダウンに表示されているはずです。接続を作成する必要がある場合は、[Teradata 接続を作成] を選択します。詳細については、「コネクタと接続の使用に関する概要」を参照してください。

   接続を選択したあとは、[プロパティを表示] をクリックすると、接続のプロパティを表示できます。

2. *tableName* を指定して [テーブル名] を設定します。
3. [Teradata のカスタムプロパティ] で、必要に応じてパラメータと値を入力します。

詳細オプション

Teradata ノードを作成する際に、高度なオプションを指定できます。これらのオプションは Spark AWS Glue スクリプトのプログラミング時に使用できるオプションと同じです。

「the section called “Teradata Vantage 接続”」を参照してください。

Twilio に対する接続

Twilio は、ウェブサービス API を使用して、通話の送受信、テキストメッセージの送受信、その他の通信機能を実行するためのプログラム可能な通信ツールを提供します。Twilio の API は、通信用のプラットフォームを強化します。これらの API の背後にあるのは、世界中の通信ネットワークを接続し最適化するソフトウェアレイヤーです。これにより、ユーザーはグローバルに誰でも呼び出し、メッセージを送信できます。Twilio ユーザーとして、Twilio アカウントに AWS Glue を接続できます。その後、Twilio を ETL ジョブのデータソースとして使用できます。これらのジョブを実行して、Twilio と AWS サービス、またはその他のサポートされているアプリケーション間でデータを転送します。

トピック

- Twilio 向けの AWS Glue サポート
- 接続を作成および使用するための API オペレーションを含むポリシー
- Twilio の設定
- Twilio 接続の設定
- Twilio エンティティからの読み取り
- Twilio 接続オプション
- Twilio コネクタの制限と注意事項

Twilio 向けの AWS Glue サポート

AWS Glue は、次のように Twilio をにサポートします。

ソースとしてサポートされていますか?

はい。AWS Glue ETL ジョブを使用して、Twilio からデータをクエリできます。

ターゲットとしてサポートされていますか?

いいえ。

サポートされている Twilio API バージョン

次の Twilio API バージョンがサポートされています。

- v1
- 2010-04-01

接続を作成および使用するための API オペレーションを含むポリシー

次のサンプル ポリシーで、接続の作成と使用に必要な AWS IAM 権限について説明します。新しいロールを作成する場合は、以下を含むポリシーを作成します。

```
{
 "Version": "2012-10-17",
    "Statement": [
     {
            "Effect": "Allow",
            "Action": [
                "glue:ListConnectionTypes",
                "glue:DescribeConnectionType",
                "glue:RefreshOAuth2Tokens",
                "glue:ListEntities",
                "glue:DescribeEntity"
            ],
            "Resource": "*"
        }
    ]
 }
```

上記の方法を使用しない場合は、代わりに次のマネージド IAM ポリシーを使用します:

- AWSGlueServiceRole – さまざまな AWS Glue プロセスを実行するために必要なリソースへのアクセス権をユーザーに代わって付与します。これらのリソースには AWS Glue 、Amazon S3、IAM、CloudWatch Logs、および Amazon EC2 が含まれます。このポリシーで指定されたリソースの命名規則に従った場合、AWS Glue プロセスは必要なアクセス権限を使用できます。このポリシーは、通常、クローラ、ジョブ、開発エンドポイントを定義するときに指定されたロールにアタッチされます。
- AWSGlueConsoleFullAccess — ポリシーがアタッチされているアイデンティティが AWS マネージメントコンソールを使用するときは、AWS Glue リソースへのフルアクセスを許可します。このポリシーで指定されたリソースの命名規則に従った場合、ユーザーは完全なコンソール機能を使用できます。このポリシーは、通常 AWS Glue コンソールのユーザーにアタッチされています。

Twilio の設定

AWS Glue を使用して Twilio からデータを転送できるようにするには、次の要件を満たす必要があります。

最小要件

以下に、最小要件を示します。

- ユーザー名とパスワードを持つ Twilio アカウントがある。
- Twilio アカウントで API アクセスが有効になっている。

これらの要件を満たすと、Twilio アカウントに AWS Glue を接続する準備が整います。一般的な接続では、Twilio で他に何もする必要はありません。

Twilio 接続の設定

Twilio は、基本認証のユーザー名とパスワードをサポートしています。基本認証は、クライアントが保護されたリソースにアクセスするための認証情報を直接提供する簡単な認証方法です。AWS Glue はユーザー名 (アカウント SID) とパスワード (認証トークン) を使用して Twilio API を認証できます。

基本的な認証フローに関する Twilio のパブリックドキュメントについては、「基本認証 | Twilio」を参照してください。

Twilio 接続を設定するには:

1. AWS Secrets Manager で、次の詳細を含むシークレットを作成します。
   - 基本認証の場合: シークレットには、[アカウント SID] (ユーザー名) と [認証トークン] (パスワード) を使用して接続されたアプリケーションのコンシューマーシークレットを含める必要があります。

     > **Note**
     >
     > AWS Glue で接続ごとにシークレットを作成する必要があります。

2. AWS Glue Glue Studio で、以下の手順に従って [データ接続] の下に接続を作成します。
   a. [接続タイプ] を選択する際に、[Twilio] を選択します。
   b. 接続先の Twilio インスタンスの `Edge_Location` を指定します。
   c. 次のアクションを実行でき、AWS Glue がその権限を持つ AWS IAM ロールを選択します。

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

```
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:DescribeSecret",
                "secretsmanager:GetSecretValue",
                "secretsmanager:PutSecretValue",
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterface",
                "ec2:DeleteNetworkInterface",
            ],
            "Resource": "*"
        }
    ]
}
```

d. AWS Glue でこの接続に使用する `secretName` を選択して、トークンを配置します。

e. ネットワークを使用する場合は、ネットワークオプションを選択します。

3. AWS Glue ジョブに関連付けられている IAM ロールに `secretName` を読み取るアクセス許可を付与します。

Twilio エンティティからの読み取り

前提条件

読み取り元の Twilio オブジェクト。`SMS-Message` または `SMS-CountryPricing` などのオブジェクト名が必要です。

ソースに対応するエンティティ:

エンティティ	インターフェイス	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
SMS-Message	REST	はい	はい	いいえ	はい	はい
SMS-CountryPricing	REST	いいえ	いいえ	いいえ	はい	いいえ

エンティティ	インターフェイス	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
Voice-Call	REST	はい	はい	いいえ	はい	いいえ
Voice-Application	REST	はい	はい	いいえ	はい	いいえ
Voice-OutgoingCallerID	REST	はい	はい	いいえ	はい	いいえ
Voice-Queue	REST	はい	はい	いいえ	はい	いいえ
Conversations-Conversation	REST	はい	はい	いいえ	はい	いいえ
Conversations-User	REST	いいえ	はい	いいえ	はい	いいえ
Conversations-Role	REST	いいえ	はい	いいえ	はい	いいえ
Conversations-Configuration	REST	いいえ	いいえ	いいえ	はい	いいえ
Conversations-AddressConfiguration	REST	はい	はい	いいえ	はい	いいえ

エンティティ	インターフェイス	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
Conversations-WebhookConfiguration	REST	いいえ	いいえ	いいえ	はい	いいえ
Conversations-ParticipantConversation	REST	いいえ	いいえ	いいえ	はい	いいえ
Conversations-Credential	REST	いいえ	はい	いいえ	はい	いいえ
Conversations-ConversationService	REST	いいえ	はい	いいえ	はい	いいえ

例:

```
twilio_read = glueContext.create_dynamic_frame.from_options(
    connection_type="twilio",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "sms-message",
        "API_VERSION": "2010-04-01",
        "Edge_Location": "sydney.us1"
    }
```

Twilio エンティティとフィールドの詳細:

エンティティ	フィールド	データ型	サポートされている演算子
SMS-Message	account_sid	String	該当なし
	api_version	String	該当なし
	本文	String	該当なし
	date_created	Datetime	該当なし
	date_sent	Datetime	>=, <=, =
	date_updated	Datetime	該当なし
	direction	String	該当なし
	error_code	整数	該当なし
	error_message	String	該当なし
	送信元	整数	=
	messaging_service_sid	String	該当なし
	num_media	String	該当なし
	num_segments	String	該当なし
	料金	String	該当なし
	price_unit	Struct	該当なし
	sid	整数	該当なし
	ステータス	String	該当なし
	subresource_uris	マッピング	該当なし
	から	整数	=

エンティティ	フィールド	データ型	サポートされている演算子
	uri	Datetime	該当なし
SMS-CountryPricing	country	String	該当なし
	iso_country	String	該当なし
	url	String	該当なし
	outbound_sms_prices	リスト	該当なし
	inbound_sms_prices	リスト	該当なし
	price_unit	String	該当なし
Voice-Call	account_sid	String	該当なし
	注釈	String	該当なし
	answered_by	String	該当なし
	api_version	String	該当なし
	caller_name	String	該当なし
	date_created	Datetime	該当なし
	date_updated	Datetime	該当なし
	direction	String	該当なし
	duration	String	該当なし
	end_time	Datetime	>=, <=, =
	forwarded_from	String	該当なし
	送信元	String	=
	from_formatted	String	該当なし

エンティティ	フィールド	データ型	サポートされている演算子
	group_sid	String	該当なし
	parent_call_sid	String	該当なし
	phone_number_sid	String	該当なし
	料金	String	該当なし
	price_unit	String	該当なし
	sid	String	該当なし
	start_time	Datetime	>=, <=, =
	ステータス	String	=
	subresource_uris	String	該当なし
	から	String	=
	to_formatted	String	該当なし
	trunk_sid	String	該当なし
	uri	String	該当なし
	queue_time	String	該当なし
Voice-Application	account_sid	String	該当なし
	api_version	String	該当なし
	date_created	Datetime	該当なし
	date_updated	Datetime	該当なし
	friendly_name	String	=

エンティティ	フィールド	データ型	サポートされている演算子
	message_status_callback	String	該当なし
	sid	String	該当なし
	sms_fallback_method	String	該当なし
	sms_fallback_url	String	該当なし
	sms_method	String	該当なし
	sms_status_callback	String	該当なし
	sms_url	String	該当なし
	status_callback	String	該当なし
	status_callback_method	String	該当なし
	uri	String	該当なし
	voice_caller_id_lookup	ブール値	該当なし
	voice_fallback_method	String	該当なし
	voice_fallback_url	String	該当なし
	voice_method	String	該当なし
	voice_url	String	該当なし
public_application_connect_enabled	ブール値	該当なし	
Voice-OutgoingCallerID	sid	String	該当なし

エンティティ	フィールド	データ型	サポートされている演算子
	date_created	Datetime	該当なし
	date_updated	Datetime	該当なし
	account_sid	String	該当なし
	friendly_name	String	=
	phone_number	String	=
	uri	String	該当なし
Voice-Queue	date_created	Datetime	該当なし
	date_updated	Datetime	該当なし
	current_size	整数	該当なし
	friendly_name	String	該当なし
	uri	String	該当なし
	account_sid	String	該当なし
	average_wait_time	整数	該当なし
	sid	String	該当なし
	max_size	整数	該当なし
Conversations-Conversation	account_sid	String	該当なし
	chat_service_sid	String	該当なし
	messaging_service_sid	String	該当なし
	sid	String	該当なし

エンティティ	フィールド	データ型	サポートされている演算子
	friendly_name	String	該当なし
	unique_name	String	該当なし
	属性	String	該当なし
	state	String	=
	date_created	Datetime	該当なし
	date_updated	Datetime	該当なし
	タイマー	Struct	該当なし
	url	String	該当なし
	links	Struct	該当なし
	バインディング	Struct	該当なし
	start_date	Datetime	=
	end_date	Datetime	=
	Timers.DateInactive	String	該当なし
	Timers.DateClosed	String	該当なし
Conversations-User	sid	String	該当なし
	account_sid	String	該当なし
	chat_service_sid	String	該当なし
	role_sid	String	該当なし
	identity	String	該当なし
	friendly_name	String	該当なし

エンティティ	フィールド	データ型	サポートされている演算子
	属性	String	該当なし
	is_online	ブール値	該当なし
	is_notable	ブール値	該当なし
	date_created	Datetime	該当なし
	date_updated	Datetime	該当なし
	url	String	該当なし
	links	Struct	該当なし
Conversations-Role	sid	String	該当なし
	account_sid	String	該当なし
	chat_service_sid	String	該当なし
	friendly_name	String	該当なし
	type	String	該当なし
	アクセス許可	String	該当なし
	date_created	Datetime	該当なし
	date_updated	Datetime	該当なし
	url	String	該当なし
Conversations-Configuration	account_sid	Long	該当なし
	default_chat_service_sid	String	該当なし
	default_messaging_service_sid	String	該当なし

エンティティ	フィールド	データ型	サポートされている演算子
	default_inactive_timer	String	該当なし
	default_closed_timer	String	該当なし
	url	String	該当なし
	links	マッピング	該当なし
Conversations-AddressConfiguration	sid	String	該当なし
	account_sid	String	該当なし
	type	String	該当なし
	アドレス	String	該当なし
	friendly_name	String	該当なし
	auto_creation	Struct	該当なし
	date_created	Datetime	該当なし
	date_updated	Datetime	該当なし
	url	String	該当なし
	address_country	String	該当なし
	AutoCreation.Enabled	ブール値	該当なし
	AutoCreation.Type	String	該当なし
	AutoCreation.ConversationServiceSid	String	該当なし
	AutoCreation.WebhookUrl	String	該当なし

エンティティ	フィールド	データ型	サポートされている演算子
	AutoCreation.WebhookMethod	String	該当なし
	AutoCreation.WebhookFilters	リスト	該当なし
	AutoCreation.StudioFlowSid	String	該当なし
	q AutoCreation.StudioRetryCount	整数	該当なし
Conversations-WebhookConfiguration	account_sid	String	該当なし
	method	String	該当なし
	フィルター	リスト	該当なし
	pre_webhook_url	String	該当なし
	post_webhook_url	String	該当なし
	target	String	該当なし
	url	String	該当なし
Converations-ParticipantConversation	account_sid	String	該当なし
	chat_service_sid	String	該当なし
	participant_sid	String	該当なし
	Participant_user_sid	String	該当なし
	participant_identity	String	該当なし
	Participant_messaging_binding	Struct	該当なし

エンティティ	フィールド	データ型	サポートされている演算子
	Conversation_sid	String	該当なし
	conversation_unique_name	String	該当なし
	conversation_friendly_name	String	該当なし
	conversation_attributes	String	該当なし
	conversation_date_created	Datetime	該当なし
	conversation_date_updated	Datetime	該当なし
	conversation_created_by	String	該当なし
	conversation_state	String	該当なし
	conversation_timers	Struct	該当なし
	links	マッピング	該当なし
	アドレス	String	=
	identity	String	=
Conversation-Credentials	sid	String	該当なし
	account_sid	String	該当なし
	friendly_name	String	該当なし
	type	String	該当なし

エンティティ	フィールド	データ型	サポートされている演算子
	サンドボックス	String	該当なし
	date_created	Datetime	該当なし
	dated_updated	Datetime	該当なし
	url	String	該当なし
	証明書	String	該当なし
	private_key	String	該当なし
	api_key	String	該当なし
	シークレット	String	該当なし
Conversations-ConversationService	sid	String	該当なし
	account_sid	String	該当なし
	friendly_name	String	該当なし
	date_created	Datetime	該当なし
	date_updated	Datetime	該当なし
	url	String	該当なし
	links	マッピング	該当なし

パーティショニングクエリ

パーティション分割をサポートするフィールド:

Twilio では、DateTime データ型フィールドはフィールドベースのパーティシン分割をサポートしています。

Spark で同時実行を使用する場合は、追加の Spark オプション PARTITION_FIELD、LOWER_BOUND、UPPER_BOUND、および NUM_PARTITIONS を指定できます。これらのパラメータを使用すると、元のクエリは Spark タスクで同時に実行できるサブクエリの NUM_PARTITIONS の数に分割されます。

- PARTITION_FIELD: クエリのパーティション化に使用するフィールドの名前。
- LOWER_BOUND: 選択したパーティションフィールドの包括的な下限値。

  Datetime フィールドでは、Spark SQL クエリで使用される Spark タイムスタンプ形式を受け入れます。

  有効な値の例は次のとおりです:

  ```
  "2024-05-01T20:55:02.000Z"
  ```

- UPPER_BOUND: 選択したパーティションフィールドの排他的上限値。
- NUM_PARTITIONS: パーティション数。

例:

```
twilio_read = glueContext.create_dynamic_frame.from_options(
    connection_type="twilio",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "sms-message",
        "API_VERSION": "2010-04-01",
        "PARTITION_FIELD": "date_sent"
        "LOWER_BOUND": "2024-05-01T20:55:02.000Z"
        "UPPER_BOUND": "2024-06-01T20:55:02.000Z"
        "NUM_PARTITIONS": "10"
    }
```

Twilio 接続オプション

Twilio の接続オプションは次のとおりです。

- ENTITY_NAME(文字列) – (必須) 読み取りに使用されます。Twilio のオブジェクトの名前。
- EDGE_LOCATION(文字列) – (必須) 有効な Twilio エッジロケーション。

- API_VERSION(文字列) – (必須) 読み取りに使用されます。使用する Twilio Rest API バージョン。Twilio は、「v1」と「2010-04-01」の 2 つの API バージョンをサポートしています。
- SELECTED_FIELDS(List<String>) – Default: empty(SELECT *). 読み込みに使用されます。オブジェクトに選択する列。
- FILTER_PREDICATE(文字列) – デフォルト: 空 読み込みに使用されます。Spark SQL 形式である必要があります。
- QUERY(文字列) – デフォルト: 空 読み込みに使用されます。完全な Spark SQL クエリ。
- PARTITION_FIELD(文字列) – 読み取りに使用されます。クエリをパーティション化するために使用するフィールド。
- LOWER_BOUND(文字列) - 読み取りに使用されます。選択したパーティションフィールドの包括的な下限値。
- UPPER_BOUND(文字列) – 読み取りに使用されます。選択したパーティションフィールドの排他的上限値。
- NUM_PARTITIONS(整数) – デフォルト: 1。読み込みに使用されます。読み取り用のパーティションの数。
- INSTANCE_URL(文字列) – (必須) 読み取りに使用されます。有効な Twilio インスタンス URL。

Twilio コネクタの制限と注意事項

Twilio コネクタの制限事項または注意事項は次のとおりです。

- レコードベースのパーティション分割はサポートされていないのは、Twilio からレコードの合計数を取得するためのプロビジョニングがないためです
- フィールド date_sent、start_time、end_time は Datetime データ型ですが、フィルタリング時には日付値のみをサポートします (時間コンポーネントは考慮されません)。
- 「from」フィールドまたは「to」フィールドのフィルタリングは、値にプレフィックス (プロトコルやラベルなど) が含まれていない場合にのみ機能します。プレフィックスが存在する場合、それぞれのフィールドのフィルタリングは機能しません。たとえば、「to」:「whatsapp:+14xxxxxxxxxx」をフィルターとして渡すと、Twilio はレスポンスを返しません。「to」:「+14xxxxxxxx」として渡す必要があります。レコードが存在する場合はレコードが返されます。
- conversation-participant-conversation エンティティをクエリするときは、「identity」フィールドフィルターは必須です。

AWS Glue Studio での Vertica に対する接続

AWS Glue は、Vertica のための組み込みサポートを提供します。AWS Glue Studio は、Vertica に接続してデータ統合ジョブをオーサリングし、AWS Glue Studio サーバーレス Spark ランタイム上でそれらのジョブを実行するためのビジュアルインターフェイスを提供します。

AWS Glue Studio は Vertica の統合接続を作成します。詳細については、「考慮事項」を参照してください。

トピック

- Vertica 接続の作成
- Vertica ソースノードの作成
- Vertica ターゲットノードの作成
- 詳細オプション

Vertica 接続の作成

前提条件:

- データベースの読み取りおよび書き込み時に一時ストレージとして使用する Amazon S3 バケットまたはフォルダ。*tempS3Path* によって参照されます。

> **Note**
>
> AWS Glue ジョブデータプレビューで Vertica を使用する場合、一時ファイルは *tempS3Path* から自動的に削除されない場合があります。一時ファイルを確実に削除するには、[データプレビュー] ペインで [セッションの終了] を選択して、データプレビューセッションを直接終了します。
> データプレビューセッションが直接終了することを保証できない場合は、Amazon S3 ライフサイクル構成を設定して古いデータを削除することを検討してください。ジョブの最大実行時間にマージンを加えた値に基づいて、49 時間を超える時間が経過しているデータを削除することをお勧めします。Amazon S3 ライフサイクルの設定の詳細については、Amazon S3 ドキュメントの「ストレージのライフサイクルの管理」を参照してください。

- AWS Glue ジョブロールに関連付けることができる、Amazon S3 パスに対する適切な許可を持つ IAM ポリシー。

- Vertica インスタンスが Amazon VPC 内にある場合は、トラフィックがパブリックインターネットを経由することなく、AWS Glue ジョブが Vertica インスタンスと通信できるように Amazon VPC を設定します。

  Amazon VPC で、AWS Glue がジョブの実行中に使用する [VPC]、[サブネット]、および [セキュリティグループ] を特定または作成します。さらに、Vertica インスタンスとこの場所の間のネットワークトラフィックを許可するように Amazon VPC が設定されているようにする必要があります。ジョブは、Vertica クライアントポート (デフォルトは 5433) との TCP 接続を確立する必要があります。ネットワークレイアウトに基づいて、セキュリティグループルール、ネットワーク ACL、NAT ゲートウェイ、およびピアリング接続の変更が必要になる場合があります。

Vertica に対する接続を設定するには:

1. AWS Secrets Manager で、Vertica 認証情報、*verticaUsername* および *verticaPassword* を使用してシークレットを作成します。Secrets Manager でシークレットを作成するには、AWS Secrets Manager ドキュメントの「AWS Secrets Manager シークレットを作成する」にあるチュートリアルに従ってください。シークレットを作成したら、次のステップのためにシークレット名 *secretName* を保存しておきます。

   - [key/value ペア] を選択する際に、*verticaUsername* という値を持つキー `user` のペアを作成します。
   - [key/value ペア] を選択する際に、*verticaPassword* という値を持つキー `password` のペアを作成します。

2. AWS Glue コンソールで、「the section called “AWS Glue 接続の追加”」にあるステップに従って接続を作成します。接続を作成したら、次のステップのために接続名 *connectionName* を保存しておきます。

   - [接続タイプ] を選択する際に、[Vertica] を選択します。
   - [Vertica ホスト] を選択する場合は、Vertica インストールのホスト名を入力します。
   - [Vertica ポート] を選択すると、Vertica インストールを使用できるポートが選択されます。
   - [AWS Secret] をクリックして、*secretName* を入力します。

3. 次の状況では、追加の設定が必要になる場合があります。

   - Amazon VPC の AWS でホストされている Vertica インスタンスの場合

- Vertica セキュリティ認証情報を定義する AWS Glue 接続に、Amazon VPC 接続に関する情報を提供します。接続を作成または更新する際に、[ネットワークオプション] で [VPC]、[サブネット]、および [セキュリティグループ] を設定します。

AWS Glue ジョブを実行する前に、次のステップを実行する必要があります。

- AWS Glue ジョブの許可に関連付けられた IAM ロールを *tempS3Path* に付与します。
- AWS Glue ジョブに関連付けられている IAM ロールに *secretName* を読み取るアクセス許可を付与します。

Vertica ソースノードの作成

必要な前提条件

- 前のセクション「the section called “Vertica 接続の作成”」で説明したように、Vertica タイプの AWS Glue データカタログ接続である *connectionName* と、一時的な Amazon S3 の場所である *tempS3Path*。
- 読み取り元とする Vertica テーブル、*tableName*、またはクエリ *targetQuery*。

Vertica データソースの追加

[データソース — Vertica] ノードを追加するには:

1. Vertica データソース用の接続を選択します。すでに作成したので、ドロップダウンに表示されているはずです。接続を作成する必要がある場合は、[Vertica 接続を作成] を選択します。詳細については、前の「the section called “Vertica 接続の作成”」セクションを参照してください。

   接続を選択したあとは、[プロパティを表示] をクリックすると、接続のプロパティを表示できます。
2. テーブルを含む [データベース] を選択します。
3. [Amazon S3 のステージングエリア] を選択し、S3A URI を *tempS3Path* に入力します。
4. [Vertica ソース] を選択します。

   - [単一のテーブルを選択] – 単一のテーブルからすべてのデータにアクセスします。
   - [カスタムクエリを入力] — カスタムクエリに基づいて、複数のテーブルからデータセットにアクセスします。

5. 単一のテーブルを選択した場合は、*tableName* を入力し、オプションで [スキーマ] を選択します。

   [カスタムクエリを入力] を選択した場合は、SQL SELECT クエリを入力し、オプションで [スキーマ] を選択します。

6. [Vertica のカスタムプロパティ] で、必要に応じてパラメータと値を入力します。

Vertica ターゲットノードの作成

必要な前提条件

- 前のセクション「the section called “Vertica 接続の作成”」で説明したように、Vertica タイプの AWS Glue データカタログ接続である *connectionName* と、一時的な Amazon S3 の場所である *tempS3Path*。

Vertica データターゲットの追加

[データターゲット — Vertica] ノードを追加するには:

1. Vertica データソース用の接続を選択します。すでに作成したので、ドロップダウンに表示されているはずです。接続を作成する必要がある場合は、[Vertica 接続を作成] を選択します。詳細については、前の「the section called “Vertica 接続の作成”」セクションを参照してください。

   接続を選択したあとは、[プロパティを表示] をクリックすると、接続のプロパティを表示できます。

2. テーブルを含む [データベース] を選択します。
3. [Amazon S3 のステージングエリア] を選択し、S3A URI を *tempS3Path* に入力します。
4. *tableName* を入力し、オプションで [スキーマ] を選択します。
5. [Vertica のカスタムプロパティ] で、必要に応じてパラメータと値を入力します。

詳細オプション

Vertica ノードを作成する際に、高度なオプションを指定できます。これらのオプションは Spark AWS Glue スクリプトのプログラミング時に使用できるオプションと同じです。

「the section called “Vertica 接続”」を参照してください。

WooCommerce に対する接続

WooCommerce は、WordPress ベースのウェブサイト用に構築されたオープンソースの柔軟なソフトウェアソリューションです。オンラインの e コマースショップを作成するのによく使用されます。このソフトウェアソリューションを使用すると、誰でも通常のウェブサイトを完全に機能するオンラインストアに変えることができます。

トピック

- WooCommerce 向け AWS Glue サポート
- 接続を作成および使用するための API オペレーションを含むポリシー
- WooCommerce の設定
- WooCommerce 接続の設定
- WooCommerce エンティティからの読み取り
- WooCommerce 接続オプション

WooCommerce 向け AWS Glue サポート

AWS Glue は、次のように WooCommerce をサポートします。

ソースとしてサポートされていますか?

はい。AWS Glue ETL ジョブを使用して、WooCommerce からデータをクエリできます。

ターゲットとしてサポートされていますか?

いいえ。

サポートされている WooCommerce API バージョン

次の WooCommerce API バージョンがサポートされています。

- v3

接続を作成および使用するための API オペレーションを含むポリシー

次のサンプル ポリシーで、接続の作成と使用に必要な AWS IAM 権限について説明します。新しいロールを作成する場合は、以下を含むポリシーを作成します。

```
{
 "Version": "2012-10-17",
    "Statement": [
     {
            "Effect": "Allow",
            "Action": [
                "glue:ListConnectionTypes",
                "glue:DescribeConnectionType",
                "glue:RefreshOAuth2Tokens",
                "glue:ListEntities",
                "glue:DescribeEntity"
            ],
            "Resource": "*"
        }
    ]
 }
```

上記の方法を使用しない場合は、代わりに次のマネージド IAM ポリシーを使用します:

- AWSGlueServiceRole – さまざまな AWS Glue プロセスを実行するために必要なリソースへのアクセス権をユーザーに代わって付与します。これらのリソースには AWS Glue 、Amazon S3、IAM、CloudWatch Logs、および Amazon EC2 が含まれます。このポリシーで指定されたリソースの命名規則に従った場合、AWS Glue プロセスは必要なアクセス権限を使用できます。このポリシーは、通常、クローラ、ジョブ、開発エンドポイントを定義するときに指定されたロールにアタッチされます。
- AWSGlueConsoleFullAccess — ポリシーがアタッチされているアイデンティティが AWS マネージメントコンソールを使用するときは、AWS Glue リソースへのフルアクセスを許可します。このポリシーで指定されたリソースの命名規則に従った場合、ユーザーは完全なコンソール機能を使用できます。このポリシーは、通常 AWS Glue コンソールのユーザーにアタッチされています。

WooCommerce の設定

AWS Glue を使用して WooCommerce からデータを転送できるようにするには、次の要件を満たす必要があります。

最小要件

以下に、最小要件を示します。

- `consumerKey` と `consumerSecret` を持つ WooCommerce アカウントがある。

- お客様の WooCommerce アカウントには、有効なライセンスを持つ API アクセスがある。

これらの要件を満たすと、WooCommerce アカウントに AWS Glue を接続する準備が整います。一般的な接続では、WooCommerce で他に何もする必要はありません。

WooCommerce 接続の設定

WooCommerce はカスタム認証をサポートしています。カスタム認可に必要な API キーの生成に関する WooCommerce の公開ドキュメントについては、「認証 – WooCommerce REST API ドキュメント」を参照してください。

WooCommerce 接続を設定するには：

1. AWS Secrets Manager で、次の詳細を含むシークレットを作成します。
   - カスタマーマネージド接続アプリケーションの場合、シークレットには、`consumerKey` と `consumerSecret` をキーとした接続アプリのコンシューマーシークレットを含める必要があります。注: AWS Glue で接続ごとにシークレットを作成する必要があります。

1. AWS Glue Glue Studio で、以下の手順に従って [データ接続] の下に接続を作成します。
   a. [接続タイプ] を選択する際に、[WooCommerce] を選択します。
   b. 接続先の WooCommerce インスタンスの `INSTANCE_URL` を指定します。
   c. 次のアクションを実行でき、AWS Glue がその権限を持つ AWS IAM ロールを選択します。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:DescribeSecret",
                "secretsmanager:GetSecretValue",
                "secretsmanager:PutSecretValue",
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterface",
                "ec2:DeleteNetworkInterface",
            ],
            "Resource": "*"
        }
    ]
```

```
}
```

d. AWS Glue でこの接続に使用する `secretName` を選択して、トークンを配置します。

e. ネットワークを使用する場合は、ネットワークオプションを選択します。

2. AWS Glue ジョブに関連付けられている IAM ロールに `secretName` を読み取るアクセス許可を付与します。

WooCommerce エンティティからの読み取り

前提条件

読み取り元の WooCommerce オブジェクト。クーポン、注文、製品などのオブジェクト名が必要です。

ソースに対応するエンティティ:

エンティティ	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
クーポン	はい	はい	はい	はい	はい
クーポンの合計	いいえ	いいえ	いいえ	はい	いいえ
顧客合計	いいえ	いいえ	いいえ	はい	いいえ
注文	はい	はい	はい	はい	はい
注文合計	いいえ	いいえ	いいえ	はい	いいえ
ペイメントゲートウェイ	いいえ	いいえ	いいえ	はい	いいえ
製品	はい	はい	はい	はい	はい
製品属性	はい	はい	はい	はい	はい
製品カテゴリ	はい	はい	はい	はい	はい

エンティティ	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
製品レビュー	はい	はい	はい	はい	はい
製品の配送クラス	はい	はい	はい	はい	はい
製品タグ	はい	はい	はい	はい	はい
製品のバリエーション	はい	はい	はい	はい	はい
製品合計	いいえ	いいえ	いいえ	はい	いいえ
レポート (リスト)	いいえ	いいえ	いいえ	はい	いいえ
レビュー合計	いいえ	いいえ	いいえ	はい	いいえ
販売レポート	はい	いいえ	いいえ	はい	いいえ
配送方法	いいえ	いいえ	いいえ	はい	いいえ
配送ゾーン	いいえ	いいえ	いいえ	はい	いいえ
配送ゾーンの場所	いいえ	いいえ	いいえ	はい	いいえ
配送ゾーンの方法	いいえ	いいえ	いいえ	はい	いいえ
税率	はい	はい	はい	はい	はい
税クラス	いいえ	いいえ	いいえ	はい	いいえ
上位の販売者レポート	はい	いいえ	いいえ	はい	いいえ

例:

```
woocommerce_read = glueContext.create_dynamic_frame.from_options(
    connection_type="glue.spark.woocommerce",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "coupon",
        "API_VERSION": "v3",
        "INSTANCE_URL": "instanceUrl"
    }
```

WooCommerce エンティティとフィールドの詳細:

エンティティ	フィールド	データ型	サポートされている演算子
coupon	id	整数	該当なし
	コード	String	EQUAL_TO
	amount	String	該当なし
	ステータス	String	該当なし
	date_created	DateTime	該当なし
	date_created_gmt	DateTime	該当なし
	date_modified	DateTime	該当なし
	date_modified_gmt	DateTime	該当なし
	discount_type	String	該当なし
	description	String	該当なし
	date_expires	String	該当なし
	date_expires_gmt	String	該当なし
	usage_count	整数	該当なし

エンティティ	フィールド	データ型	サポートされている演算子
	individual_use	ブール値	該当なし
	product_ids	リスト	該当なし
	excluded_product_ids	リスト	該当なし
	usage_limit	整数	該当なし
	usage_limit_per_user	整数	該当なし
	limit_usage_to_x_items	整数	該当なし
	free_shipping	ブール値	該当なし
	product_categories	リスト	該当なし
	excluded_product_categories	リスト	該当なし
	exclude_sale_items	ブール値	該当なし
	minimum_amount	String	該当なし
	maximum_amount	String	該当なし
	email_restrictions	リスト	該当なし
	used_by	リスト	該当なし
	meta_data	リスト	該当なし
	context	String	EQUAL_TO
	検索	String	EQUAL_TO
	後に	DateTime	EQUAL_TO
	前に	DateTime	EQUAL_TO

エンティティ	フィールド	データ型	サポートされている演算子
	order	String	EQUAL_TO
	orderby	String	EQUAL_TO
	modified_after	DateTime	EQUAL_TO
	modified_before	DateTime	EQUAL_TO
	dates_are_gmt	ブール値	EQUAL_TO
coupon-total	slug	String	該当なし
	名前	String	該当なし
	total	整数	該当なし
customer-total	slug	String	該当なし
	名前	String	該当なし
	total	整数	該当なし
order	id	整数	該当なし
	parent_id	整数	該当なし
	number	String	該当なし
	order_key	String	該当なし
	created_via	String	該当なし
	ステータス	String	該当なし
	currency	String	該当なし
	version	String	該当なし
	date_created	DateTime	該当なし

エンティティ	フィールド	データ型	サポートされている演算子
	date_modified	DateTime	該当なし
	discount_total	String	該当なし
	discount_tax	String	該当なし
	shipping_total	String	該当なし
	shipping_tax	String	該当なし
	cart_tax	String	該当なし
	total	String	該当なし
	total_tax	String	該当なし
	prices_include_tax	ブール値	該当なし
	customer_id	整数	該当なし
	customer_ip_address	String	該当なし
	customer_user_agent	String	該当なし
	customer_note	String	該当なし
	請求	Struct	該当なし
	shipping	Struct	該当なし
	payment_method	String	該当なし
	payment_method_title	String	該当なし
	transaction_id	String	該当なし
	date_paid	DateTime	該当なし
	date_completed	DateTime	該当なし

エンティティ	フィールド	データ型	サポートされている演算子
	cart_hash	String	該当なし
	meta_data	リスト	該当なし
	line_items	リスト	該当なし
	tax_lines	リスト	該当なし
	shipping_lines	リスト	該当なし
	fee_lines	リスト	該当なし
	coupon_lines	リスト	該当なし
	返金	リスト	該当なし
	payment_url	String	該当なし
	is_editable	ブール値	該当なし
	needs_payment	ブール値	該当なし
	needs_processing	ブール値	該当なし
	date_created_gmt	DateTime	該当なし
	date_modified_gmt	DateTime	該当なし
	date_completed_gmt	DateTime	該当なし
	date_paid_gmt	DateTime	該当なし
	currency_symbol	String	該当なし
	set_paid	ブール値	該当なし
	context	String	EQUAL_TO
	検索	String	EQUAL_TO

エンティティ	フィールド	データ型	サポートされている演算子
	後に	DateTime	EQUAL_TO
	前に	DateTime	EQUAL_TO
	order	String	EQUAL_TO
	orderby	String	EQUAL_TO
	カスタマー	整数	EQUAL_TO
	product	整数	EQUAL_TO
	dp	整数	EQUAL_TO
	modified_before	DateTime	EQUAL_TO
	modified_after	DateTime	EQUAL_TO
	dates_are_gmt	ブール値	EQUAL_TO
order-total	slug	String	該当なし
	名前	String	該当なし
	total	整数	該当なし
payment-gateway	title	String	該当なし
	description	String	該当なし
	order	String	該当なし
	有効	ブール値	該当なし
	method_title	String	該当なし
	method_description	String	該当なし
	method_supports	リスト	該当なし

エンティティ	フィールド	データ型	サポートされている演算子
	設定	String	該当なし
	needs_setup	ブール値	該当なし
	post_install_scripts	リスト	該当なし
	settings_url	String	該当なし
	connection_url	String	該当なし
	setup_help_text	String	該当なし
	required_settings_keys	リスト	該当なし
product	id	整数	該当なし
	名前	String	該当なし
	type	String	EQUAL_TO
	permalink	String	該当なし
	date_created	DateTime	該当なし
	date_created_gmt	DateTime	該当なし
	date_modified	DateTime	該当なし
	date_modified_gmt	DateTime	該当なし
	catalog_visibility	String	該当なし
	description	String	該当なし
	short_description	String	該当なし
	料金	String	該当なし

エンティティ	フィールド	データ型	サポートされている演算子
	regular_price	String	該当なし
	sale_price	String	該当なし
	date_on_sale_from	DateTime	該当なし
	date_on_sale_from_gmt	DateTime	該当なし
	date_on_sale_to	DateTime	該当なし
	date_on_sale_to_gmt	DateTime	該当なし
	price_html	String	該当なし
	購入可能	ブール値	該当なし
	total_sales	整数	該当なし
	仮想	ブール値	該当なし
	ダウンロードできます	ブール値	該当なし
	ダウンロード	リスト	該当なし
	download_limit	整数	該当なし
	download_expiry	整数	該当なし
	external_url	String	該当なし
	button_text	String	該当なし
	tax_status	String	該当なし
	manage_stock	ブール値	該当なし
	stock_quantity	整数	該当なし

エンティティ	フィールド	データ型	サポートされている演算子
	バックオーダー	String	該当なし
	backorders_allowed	ブール値	該当なし
	バックオーダー済み	ブール値	該当なし
	sold_invidually	ブール値	該当なし
	weight	String	該当なし
	ディメンション	Struct	該当なし
	shipping_required	ブール値	該当なし
	shipping_taxable	ブール値	該当なし
	shipping_class_id	整数	該当なし
	reviews_allowed	ブール値	該当なし
	average_rating	String	該当なし
	rating_count	整数	該当なし
	related_ids	リスト	該当なし
	upsell_ids	リスト	該当なし
	cross_sell_ids	リスト	該当なし
	parent_id	整数	該当なし
	purchase_note	String	該当なし
	カテゴリ	リスト	該当なし
	タグ	リスト	該当なし
	画像	リスト	該当なし

エンティティ	フィールド	データ型	サポートされている演算子
	属性	リスト	該当なし
	default_attributes	リスト	該当なし
	バリエーション	リスト	該当なし
	grouped_products	リスト	該当なし
	menu_order	整数	該当なし
	meta_data	リスト	該当なし
	low_stock_amount	整数	該当なし
	jetpack_publicize_connections	リスト	該当なし
	jetpack-related-posts	リスト	該当なし
	jetpack_likes_enabled	ブール値	該当なし
	jetpack_sharing_enabled	ブール値	該当なし
	context	String	EQUAL_TO
	検索	String	EQUAL_TO
	後に	DateTime	EQUAL_TO
	前に	DateTime	EQUAL_TO
	order	String	EQUAL_TO
	orderby	String	EQUAL_TO
	slug	String	EQUAL_TO
	ステータス	String	EQUAL_TO

エンティティ	フィールド	データ型	サポートされている演算子
	SKU	String	EQUAL_TO
	注目	ブール値	EQUAL_TO
	タグ	String	EQUAL_TO
	shipping_class	String	EQUAL_TO
	tax_class	String	EQUAL_TO
	on_sale	ブール値	EQUAL_TO
	stock_status	String	EQUAL_TO
	has_options	ブール値	該当なし
	modified_after	DateTime	EQUAL_TO
	modified_before	DateTime	EQUAL_TO
	dates_are_gmt	ブール値	EQUAL_TO
	category	String	EQUAL_TO
	属性	String	EQUAL_TO
	min_price	String	EQUAL_TO
	max_price	String	EQUAL_TO
product-attribute	id	整数	該当なし
	名前	String	該当なし
	slug	String	該当なし
	type	String	該当なし
	order_by	String	該当なし

エンティティ	フィールド	データ型	サポートされている演算子
	has_archives	ブール値	該当なし
	context	String	EQUAL_TO
product-attribute-term	id	整数	該当なし
	名前	String	該当なし
	slug	String	該当なし
	description	String	該当なし
	menu_order	整数	該当なし
	count	整数	該当なし
	context	String	EQUAL_TO
	検索	String	EQUAL_TO
	order	String	EQUAL_TO
	orderby	String	EQUAL_TO
	hide_empty	ブール値	EQUAL_TO
	parent	整数	EQUAL_TO
	product	整数	EQUAL_TO
product-category	id	整数	該当なし
	名前	String	該当なし
	slug	String	EQUAL_TO
	description	String	該当なし
	display	String	該当なし

エンティティ	フィールド	データ型	サポートされている演算子
	画像	Struct	該当なし
	menu_order	整数	該当なし
	count	整数	該当なし
	context	String	EQUAL_TO
	検索	String	EQUAL_TO
	order	String	EQUAL_TO
	orderby	String	EQUAL_TO
	hide_empty	ブール値	EQUAL_TO
	parent	整数	EQUAL_TO
	product	整数	EQUAL_TO
product-review	id	整数	該当なし
	date_created	DateTime	該当なし
	date_created_gmt	DateTime	該当なし
	product_id	整数	該当なし
	product_name	String	該当なし
	product_permalink	String	該当なし
	確認	String	該当なし
	rating	整数	該当なし
	verified	ブール値	該当なし
	reviewer	String	該当なし

エンティティ	フィールド	データ型	サポートされている演算子
	reviewer_email	String	該当なし
	reviewer_avatar_urls	Struct	該当なし
	context	String	EQUAL_TO
	検索	String	EQUAL_TO
	後に	DateTime	EQUAL_TO
	前に	DateTime	EQUAL_TO
	order	String	EQUAL_TO
	orderby	String	EQUAL_TO
	ステータス	String	EQUAL_TO
product-shipping-class	id	整数	該当なし
	名前	String	該当なし
	slug	String	EQUAL_TO
	description	String	該当なし
	count	整数	該当なし
	context	String	EQUAL_TO
	検索	String	EQUAL_TO
	order	String	EQUAL_TO
	orderby	String	EQUAL_TO
	hide_empty	String	EQUAL_TO
	product	整数	EQUAL_TO

エンティティ	フィールド	データ型	サポートされている演算子
product-tag	id	整数	該当なし
	名前	String	該当なし
	slug	String	EQUAL_TO
	description	String	該当なし
	count	整数	該当なし
	context	String	EQUAL_TO
	検索	String	EQUAL_TO
	order	String	EQUAL_TO
	orderby	String	EQUAL_TO
	hide_empty	ブール値	EQUAL_TO
	product	整数	EQUAL_TO
product-total	slug	String	該当なし
	名前	String	該当なし
	total	整数	該当なし
product-variation	id	整数	該当なし
	date_created	DateTime	該当なし
	date_created_gmt	DateTime	該当なし
	date_modified	DateTime	該当なし
	date_modified_gmt	DateTime	該当なし
	description	String	該当なし

エンティティ	フィールド	データ型	サポートされている演算子
	permalink	String	該当なし
	料金	String	該当なし
	regular_price	String	該当なし
	sale_price	String	該当なし
	date_on_sale_from	DateTime	該当なし
	date_on_sale_from_gmt	DateTime	該当なし
	date_on_sale_to	DateTime	該当なし
	date_on_sale_to_gmt	DateTime	該当なし
	購入可能	ブール値	該当なし
	仮想	ブール値	該当なし
	ダウンロードできます	ブール値	該当なし
	ダウンロード	リスト	該当なし
	download_limit	整数	該当なし
	download_expiry	整数	該当なし
	tax_status	String	該当なし
	manage_stock	ブール値	該当なし
	stock_quantity	整数	該当なし
	バックオーダー	String	該当なし
	backorders_allowed	ブール値	該当なし

エンティティ	フィールド	データ型	サポートされている演算子
	バックオーダー済み	ブール値	該当なし
	low_stock_amount	整数	該当なし
	weight	String	該当なし
	ディメンション	Struct	該当なし
	shipping_class	String	該当なし
	shipping_class_id	整数	該当なし
	画像	Struct	該当なし
	属性	リスト	該当なし
	menu_order	整数	該当なし
	meta_data	リスト	該当なし
	context	String	EQUAL_TO
	検索	String	EQUAL_TO
	後に	DateTime	EQUAL_TO
	前に	DateTime	EQUAL_TO
	order	String	EQUAL_TO
	orderby	String	EQUAL_TO
	slug	String	EQUAL_TO
	ステータス	String	EQUAL_TO
	SKU	String	EQUAL_TO
	tax_class	String	EQUAL_TO

エンティティ	フィールド	データ型	サポートされている演算子
	on_sale	ブール値	EQUAL_TO
	min_price	String	EQUAL_TO
	max_price	String	EQUAL_TO
	stock_status	String	EQUAL_TO
report	slug	String	該当なし
	description	String	該当なし
review-total	slug	String	該当なし
	名前	String	該当なし
	total	整数	該当なし
sales-report	total_sales	String	該当なし
	net_sales	String	該当なし
	average_sales	String	該当なし
	total_orders	整数	該当なし
	total_items	整数	該当なし
	total_tax	String	該当なし
	total_shipping	String	該当なし
	total_refunds	整数	該当なし
	total_discount	String	該当なし
	totals_grouped_by	String	該当なし
	totals	Struct	該当なし

エンティティ	フィールド	データ型	サポートされている演算子
	total_customers	整数	該当なし
	context	String	EQUAL_TO
	period	String	EQUAL_TO
	date_min	日付	EQUAL_TO
	date_max	日付	EQUAL_TO
shipping-method	id	String	該当なし
	title	String	該当なし
	description	String	該当なし
shipping-zone	id	整数	EQUAL_TO
	名前	String	該当なし
	order	整数	該当なし
shipping-zone-location	コード	String	該当なし
	type	String	該当なし
shipping-zone-method	instance_id	整数	該当なし
	id	整数	EQUAL_TO
	title	String	該当なし
	order	整数	該当なし
	有効	ブール値	該当なし
	method_id	String	該当なし
	method_title	String	該当なし

エンティティ	フィールド	データ型	サポートされている演算子
	method_description	String	該当なし
	設定	Struct	該当なし
tax-class	slug	String	該当なし
	名前	String	該当なし
tax-rate	id	整数	該当なし
	country	String	該当なし
	state	String	該当なし
	郵便番号	String	該当なし
	city	String	該当なし
	郵便番号	リスト	該当なし
	都市	リスト	該当なし
	rate	String	該当なし
	名前	String	該当なし
	優先度	整数	該当なし
	compound	ブール値	該当なし
	shipping	ブール値	該当なし
	context	String	EQUAL_TO
	order	String	EQUAL_TO
	orderby	String	EQUAL_TO
	class	String	EQUAL_TO

エンティティ	フィールド	データ型	サポートされている演算子
top-seller-report	名前	String	該当なし
	product_id	整数	該当なし
	数量	整数	該当なし
	context	String	EQUAL_TO
	period	String	EQUAL_TO
	date_min	日付	EQUAL_TO
	date_max	日付	EQUAL_TO

> **Note**
>
> 構造体データ型とリストデータ型は、コネクタの応答で文字列データ型に変換され、DateTime データ型は Timestamp に変換されます。

パーティショニングクエリ

レコードベースのパーティション分割:

Spark で並行処理を使用する場合は、追加の Spark オプション NUM_PARTITIONS を指定できます。これらのパラメータを使用すると、元のクエリは Spark タスクで同時に実行できるサブクエリの NUM_PARTITIONS の数に分割されます。

レコードベースのパーティショニングでは、存在するレコードの合計数が WooCommerce API からクエリされ、指定された NUM_PARTITIONS 数で割られます。その後、結果のレコード数は、各サブクエリによって同時に取得されます。

- NUM_PARTITIONS: パーティション数。

次のエンティティは、レコードベースのパーティショニングをサポートしています。

- coupon
- order
- product
- product-attribute
- product-attribute-term
- product-category
- product-review
- product-shipping-class
- product-tag
- product-variation
- tax-rate

例:

```
woocommerce_read = glueContext.create_dynamic_frame.from_options(
    connection_type="glue.spark.woocommerce",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "coupon",
        "API_VERSION": "v3",
        "INSTANCE_URL": "instanceUrl"
        "NUM_PARTITIONS": "10"
    }
```

レコードベースのパーティション分割:

元のクエリは、元のクエリは Spark タスクで同時に実行できるサブクエリの NUM_PARTITIONS の数に分割されます。

- NUM_PARTITIONS: パーティション数。

例:

```
WooCommerce_read = glueContext.create_dynamic_frame.from_options(
    connection_type="WooCommerce",
    connection_options={
        "connectionName": "connectionName",
```

```
        "REALMID": "1234567890123456789",
        "ENTITY_NAME": "Bill",
        "API_VERSION": "v3",
        "NUM_PARTITIONS": "10"
    }
```

WooCommerce 接続オプション

WooCommerce の接続オプションは次のとおりです。

- ENTITY_NAME(文字列) – (必須) 読み取りに使用されます。WooCommerce のオブジェクトの名前。
- API_VERSION(文字列) – (必須) 読み取りに使用されます。使用する WooCommerce Rest API バージョン。
- REALM_ID(文字列) – リクエストを送信する個々の WooCommerce Online 会社を識別する ID。
- SELECTED_FIELDS(List<String>) - Default: empty(SELECT *). 読み込みに使用されます。オブジェクトに選択する列。
- FILTER_PREDICATE(文字列) – デフォルト: 空 読み込みに使用されます。Spark SQL 形式である必要があります。
- QUERY(文字列) – デフォルト: 空 読み込みに使用されます。完全な Spark SQL クエリ。
- INSTANCE_URL(文字列) – (必須) https://<instance>.wpcomstaging.com 形式の有効な WooCommerce インスタンス URL
- NUM_PARTITIONS(整数) – デフォルト: 1。読み込みに使用されます。読み取り用のパーティションの数。

Zendesk への接続

Zendesk は、カスタマーサービスポータル、ナレッジベース、オンラインコミュニティを構築するためのカスタマイズ可能なツールを提供するクラウドベースのヘルプデスク管理ソリューションです。

トピック

- AWS Glue Zendesk のサポート
- 接続を作成および使用するためのAPIオペレーションを含むポリシー
- Zendesk の設定
- Zendesk 接続の設定

- Zendesk エンティティからの読み取り
- Zendesk 接続オプション
- 制限

AWS Glue Zendesk のサポート

AWS Glue は、次のように Zendesk をサポートします。

ソースとしてサポートされていますか?

はい。ジョブを使用して AWS Glue ETL、Zendesk からデータをクエリできます。

ターゲットとしてサポートされているかどうか

いいえ。

サポートされている Zendesk APIバージョン

次の Zendesk APIバージョンがサポートされています

- v2

接続を作成および使用するためのAPIオペレーションを含むポリシー

次のサンプルポリシーでは、接続の作成と使用に必要な AWS IAMアクセス許可について説明します。新しいロールを作成する場合は、以下を含むポリシーを作成します。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:ListConnectionTypes",
                "glue:DescribeConnectionType",
                "glue:RefreshOAuth2Tokens"
                "glue:ListEntities",
                "glue:DescribeEntity"
            ],
            "Resource": "*"
        }
    ]
```

```
}
```

上記の方法を使用しない場合は、次の管理IAMポリシーを使用します。

- AWSGlueServiceRole – さまざまな AWS Glue プロセスがユーザーに代わって実行するために必要なリソースへのアクセスを許可します。これらのリソースには AWS Glue、、Amazon S3、IAM、 CloudWatch Logs、Amazon が含まれますEC2。このポリシーで指定されたリソースの命名規則に従うと、 AWS Glue プロセスには必要なアクセス許可があります。このポリシーは、通常、クローラ、ジョブ、開発エンドポイントを定義するときに指定されたロールにアタッチされます。
- AWSGlueConsoleFullAccess - ポリシーがアタッチされている ID が AWS マネジメントコンソールを使用する場合、 AWS Glue リソースへのフルアクセスを許可します。このポリシーで指定されたリソースの命名規則に従った場合、ユーザーは完全なコンソール機能を使用できます。このポリシーは通常、 AWS Glue コンソールのユーザーにアタッチされます。

Zendesk の設定

AWS Glue を使用して Zendesk からデータを転送する前に、以下の要件を満たす必要があります。

最小要件

以下に、最小要件を示します。

- Zendesk アカウントがあります。詳細については、「Zendesk アカウントの作成」を参照してください。
- Zendesk アカウントはAPIアクセス可能です。
- Zendesk アカウントでは、接続されたアプリケーションをインストールできます。

これらの要件を満たすと、Zendesk アカウント AWS Glue に接続できます。

Zendesk アカウントの作成

Zendesk アカウントを作成するには：

1. register https://www.zendesk.com/in// に進む
2. 勤務先の E メール、名、姓、電話番号、役職、会社名、社内の従業員数、パスワード、優先言語などの詳細を入力します。次に、トライアルサインアップの完了 を選択します。
3. アカウントを作成したら、受信した検証リンクを完了して E メールアドレスを確認します。

4. 作業用 E メールアドレスが検証されると、Zendesk アカウントにリダイレクトされます。希望するプランの Zendesk の購入 オプションを選択します。注: Zendesk コネクタの場合は、スイートエンタープライズプランを購入することをお勧めします。

クライアントアプリケーションと 2.0 OAuth 認証情報の作成

クライアントアプリケーションと 2.0 OAuth 認証情報を作成するには：

1. 2.0 アプリを作成する Zendesk OAuth アカウントにログインします https://www.zendesk.com/in/。ログイン/
2. 歯車アイコンをクリックします。Go to admin center リンクを選択して、admin center ページを開きます。
3. 左側のサイドバーでアプリケーションと統合を選択し、 APIs > Zendesk APIを選択します。
4. Zendesk APIページで、OAuthクライアントタブを選択します。
5. 右側の Oauth クライアントの追加 を選択します。
6. クライアントを作成するには、次のフィールドに入力します。
   a. クライアント名 - アプリの名前を入力します。これは、ユーザーがアプリケーションへのアクセスを許可するように求められたとき、および Zendesk にアクセスできるサードパーティーアプリのリストをチェックするときに表示される名前です。
   b. 説明 - オプション。アプリケーションへのアクセス許可を求められたときにユーザーに表示されるアプリケーションの概要。
   c. 会社 - オプション。アプリケーションへのアクセスを許可するように求められたときにユーザーに表示される会社名。情報は、誰にアクセスを許可するかを理解するのに役立ちます。
   d. 口ゴ - オプション。これは、アプリケーションへのアクセスを許可するように求められたときにユーザーに表示される口ゴです。イメージは、JPG、GIF、または ですPNG。最良の結果を得るには、正方形の画像をアップロードします。承認ページのサイズが変更されます。
   e. 一意の識別子 - フィールドには、アプリに入力した名前のリフォーマットされたバージョンが自動的に入力されます。この名前は必要な場合、変更できます。
   f. リダイレクト URLs - Zendesk URLsがアプリケーションへのアクセスを許可するユーザーの決定を送信するために使用する URLまたは を入力します。

      例： https://us-east-1.console.aws.amazon.com/gluestudio/oauth
7. [Save]（保存）をクリックします。
8. ページが更新されると、新しい事前入力されたシークレットフィールドが下部に表示されます。これは、OAuth2仕様で指定された「client_secret」値です。Secret 値をクリップボードにコピー

し、安全な場所に保存します。注: 文字はテキストボックスの幅を超える可能性があるため、コピーする前に必ずすべてを選択してください。

9. [Save] (保存) をクリックします。

Zendesk 接続の設定

Zendesk コネクタは認可コードの付与タイプをサポートしています。

- このグラントタイプは、ユーザーを認証するためにサードパーティーの認証サーバーにユーザーをリダイレクトすることから、「three-legged」の OAuth と見なされます。AWS Glue コンソール経由で接続を作成するときに使用されます。接続を作成するユーザーは、デフォルトで AWS Glue 所有の接続されたアプリケーション (AWS Glue マネージドクライアントアプリケーション) に依存する場合があります。この場合、Zendesk インスタンス URL 以外の OAuth 関連情報を提供する必要はありません。AWS Glue コンソールでは、ユーザーが Zendesk にリダイレクトされます。ここではユーザーがログインし、Zendesk インスタンスにアクセスするためのリクエストされた権限を AWS Glue に許可する必要があります。
- Zendesk で独自の接続アプリを作成し、AWS Glue コンソールを介して接続を作成するときに独自のクライアント ID とクライアントシークレットを指定することを選択できます。このシナリオでは、引き続き Zendesk にリダイレクトされてログインし、リソースへアクセスするために AWS Glue を承認します。
- このグラントタイプは、アクセストークンになります。アクセストークンは期限切れになりません。

認可コード OAuth フロー用の接続アプリケーションの作成に関する Zendesk の公開ドキュメントについては、「グラントタイプの OAuth トークン」を参照してください。

Zendesk 接続を設定するには :

1. AWS Secrets Manager で、次の詳細を含むシークレットを作成します。
   a. AuthorizationCode グラントタイプの場合: カスタマーマネージド接続アプリケーションの場合、シークレットには `USER_MANAGED_CLIENT_APPLICATION_CLIENT_SECRET` を使用して接続されたアプリケーションのコンシューマーシークレットをキーとして含める必要があります。
   b. 注: AWS Glue で接続ごとにシークレットを作成する必要があります。
2. AWS Glue Glue Studio で、以下の手順に従って [データ接続] の下に接続を作成します。
   a. [接続タイプ] を選択する際に、[Zendesk] を選択します。

b. 接続する Zendesk の INSTANCE_URL を指定します。

c. Zendesk 環境を指定します。

d. 次のアクションを実行でき、AWS Glue がその権限を持つ AWS IAM ロールを選択します。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:DescribeSecret",
                "secretsmanager:GetSecretValue",
                "secretsmanager:PutSecretValue",
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterface",
                "ec2:DeleteNetworkInterface",
            ],
            "Resource": "*"
        }
    ]
}
```

e. AWS Glue でこの接続に使用する `secretName` を選択して、トークンを配置します。

f. ネットワークを使用する場合は、ネットワークオプションを選択します。

3. AWS Glue ジョブに関連付けられている IAM ロールに `secretName` を読み取るアクセス許可を付与します。

Zendesk エンティティからの読み取り

前提条件

読み取り元の Zendesk オブジェクト。次の表に示すように、チケット、ユーザー、記事などのオブジェクト名が必要になります。

エンティティ	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
Ticket	Y	Y	Y	Y	N

エンティティ	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
ユーザー	Y	Y	Y	Y	N
組織	Y	Y	Y	Y	N
Article	Y	Y	N	Y	N
Ticket Event	Y	Y	N	Y	N
Ticket Metric Event	Y	Y	N	Y	N
Ticket Comment	Y	Y	Y	Y	N
Ticket Field	Y	Y	N	Y	N
Ticket Metric	Y	Y	N	Y	N
Ticket Activity	Y	Y	N	Y	N
Ticket Skip	N	Y	N	Y	N
グループ	Y	Y	Y	Y	N
Group Membership	N	Y	Y	Y	N
Satisfaction Rating	Y	Y	N	Y	N
ビュー	Y	Y	Y	Y	N
Trigger トリガー)	Y	Y	Y	Y	N
Trigger Category	N	Y	Y	Y	N

エンティティ	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
マクロ	Y	Y	Y	Y	N
Automation	N	Y	Y	Y	N

例:

```
Zendesk_read = glueContext.create_dynamic_frame.from_options(
    connection_type="Zendesk",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "Account",
        "API_VERSION": "v2"
    }
```

Zendesk エンティティとフィールドの詳細:

エンティティ	フィールド	データ型	サポートされている演算子	コメント
記事	url	文字列		
	id	Long		
	author_id	Long		
	本文	文字列		
	comments_disabled	ブール値		
	draft	ブール値		
	edited_at	DateTime		
	html_url	文字列		

エンティティ	フィールド	データ型	サポートされている演算子	コメント
	label_names	リスト		
	サイト	文字列	EQUAL_TO	
	outdated (旧式)	ブール値		
	outdated_locales	リスト		
	permissio n_group_id	Long		
	position	整数		
	promoted	ブール値		
	section_id	Long		
	source_locale	文字列		
	名前	文字列		
	title	文字列		
	user_segment_id	Long		
	content_tags_id	リスト		
	vote_count	整数		
	vote_sum	整数		
	created_at	DateTime		
	updated_at	DateTime	EQUAL_TO	
	label_name	文字列	EQUAL_TO	
グループ	url	文字列		

エンティティ	フィールド	データ型	サポートされている演算子	コメント
	id	Long		
	is_public	ブール値		
	名前	文字列		
	description	文字列		
	デフォルト	ブール値		
	削除済み	ブール値		
	created_at	DateTime		
	updated_at	DateTime		
	exclude_deleted	ブール値	EQUAL_TO	
Automation	url	文字列		
	id	Long		
	title	文字列		
	ACTIVE	ブール値		
	created_at	DateTime		
	updated_at	DateTime		
	デフォルト	ブール値		
	actions	リスト		
	positions	整数		
	条件	Struct		
	raw_title	文字列		

エンティティ	フィールド	データ型	サポートされている演算子	コメント
group-membership	url	文字列		
	id	Long		
	user_id	Long		
	group_id	Long		
	デフォルト	ブール値		
	created_at	DateTime		
	updated_at	DateTime		
マクロ	url	文字列		
	id	Long		
	title	文字列		
	ACTIVE	ブール値	EQUAL_TO	
	created_at	DateTime		
	updated_at	DateTime		
	デフォルト	ブール値		
	actions	リスト		
	position	整数		
	説明	文字列		
	raw_title	文字列		
	制限	Struct		
	アクセス	文字列	EQUAL_TO	

エンティティ	フィールド	データ型	サポートされている演算子	コメント
	category	整数	EQUAL_TO	
	group_id	Long	EQUAL_TO	
	only_viewable	ブール値	EQUAL_TO	
組織	url	文字列		
	id	Long		
	external_id	文字列		
	名前	文字列		
	domain_names	リスト		
	details	文字列		
	注意事項	文字列		
	group_id	Long		
	shared_tickets	ブール値		
	shared_comments	ブール値		
	タグ	リスト		
	organization_fields	Struct		
	created_at	DateTime		
	updated_at	DateTime	EQUAL_TO	

エンティティ	フィールド	データ型	サポートされている演算子	コメント
	DML_STATUS	文字列		レコードの作成、更新、削除のステータスを追跡するために使用されるユーザー定義フィールド。
satisfaction-rating	url	文字列		
	id	Long		
	assignee_id	Long		
	コメント	文字列		
	group_id	Long		
	理由	文字列		
	reason_code	整数		
	reason_id	Long		
	requester_id	Long		
	スコア	文字列	EQUAL_TO	
	ticket_id	整数		
	created_at	DateTime		
	updated_at	DateTime	EQUAL_TO	
	start_time	DateTime	EQUAL_TO	
	end_time	DateTime	EQUAL_TO	

エンティティ	フィールド	データ型	サポートされている演算子	コメント
	DML_STATUS	文字列		レコードの作成、更新、削除のステータスを追跡するために使用されるユーザー定義フィールド。
ticket-activity	アクター	Struct		
	actor_id	Long		
	created_at	DateTime		
	id	Long		
	オブジェクト	Struct		
	target	Struct		
	title	文字列		
	updated_at	DateTime		
	url	文字列		
	ユーザー	Struct		
	user_id	Long		
	verb	文字列		
	since	DateTime	EQUAL_TO	
ticket-comment	id	Long		
	type	文字列		

エンティティ	フィールド	データ型	サポートされている演算子	コメント
	author_id	Long		
	本文	文字列		
	html_body	文字列		
	plain_body	文字列		
	パブリック	ブール値		
	添付ファイル	リスト		
	audit_id	Long		
	via	Struct		
	created_at	DateTime		
	metadata	Struct		
	ticket_id	整数	EQUAL_TO	
	include_inline_images	ブール値	EQUAL_TO	
ticket-events	id	Long		
	ticket_id	Long		
	timestamp	Long		
	created_at	DateTime		
	updater_id	Long		
	child_events	リスト		
	via	文字列		

エンティティ	フィールド	データ型	サポートされている演算子	コメント
	システム	Struct		
	event_type	文字列		
	comment_present	ブール値		
	comment_public	ブール値		
	via_reference_id	Long		
	created_at	DateTime	EQUAL_TO	
	DML_STATUS	文字列		レコードの作成、更新、削除のステータスを追跡するために使用されるユーザー定義フィールド。
ticket-field	url	文字列		
	id	Long		
	type	文字列		
	title	文字列		
	raw_title	文字列		
	description	文字列		
	raw_description	文字列		
	position	整数		
	ACTIVE	ブール値		

エンティティ	フィールド	データ型	サポートされている演算子	コメント
	必須	ブール値		
	collapsed_for_agents	ブール値		
	regexp_for_validation	文字列		
	title_in_portal	文字列		
	raw_title_in_portal	文字列		
	visible_in_portal	ブール値		
	editable_on_portal	ブール値		
	required_in_portal	ブール値		
	タグ	文字列		
	created_at	DateTime		
	updated_at	DateTime		
	removable	ブール値		
	agent_description	文字列		
	custom_field_options	リスト		
	custom_statuses	リスト		
	relationship_filter	Struct		

エンティティ	フィールド	データ型	サポートされている演算子	コメント
	relationship_target_type	文字列		
	sub_type_id	整数		
	system_field_options	リスト		
	サイト	文字列	EQUAL_TO	
ticket-metric-events	id	Long		
	時系	DateTime	EQUAL_TO	
	ticket_id	整数		
	メトリクス	文字列		
	instance_id	整数		
	type	文字列		
	DML_STATUS	文字列	EQUAL_TO	レコードの作成、更新、削除のステータスを追跡するために使用されるユーザー定義フィールド。
ticket-metric	url	文字列		
	id	Long		
	ticket_id	整数		
	created_at	DateTime		

エンティティ	フィールド	データ型	サポートされている演算子	コメント
	updated_at	DateTime		
	group_stations	整数		
	assignee_stations	整数		
	reopens	整数		
	replies	整数		
	assignee_updated_at	DateTime		
	requester_updated_at	DateTime		
	initially_assigned_at	DateTime		
	assigned_at	DateTime		
	solved_at	DateTime		
	last_comment_added_at	DateTime		
	reply_time_in_minutes	Struct		
	first_resolution_time_in_minutes	Struct		
	full_resolution_time_in_minutes	Struct		

エンティティ	フィールド	データ型	サポートされている演算子	コメント
	agent_wait_time_in_minutes	Struct		
	requester_wait_time_in_minutes	Struct		
	on_hold_time_in_seconds	Struct		
	reply_time_in_seconds	Struct		
	custom_status_updated_at	DateTime		
ticket-skip	created_at	DateTime		
	id	Long		
	理由	文字列		
	ticket	Struct		
	ticket_id	整数		
	updated_at	DateTime		
	user_id	Long		
tickets	url	文字列		
	id	Long		
	external_id	文字列	EQUAL_TO	
	type	文字列		

エンティティ	フィールド	データ型	サポートされている演算子	コメント
	subject	文字列		
	raw_subject	文字列		
	description	文字列		
	priority	文字列		
	status	文字列		
	受取人	文字列		
	リクエスタ	Struct		
	requester_id	Long		
	submitter_id	Long		
	assignee_id	Long		
	organization_id	Long		
	group_id	Long		
	collaborator_ids	リスト		
	email_cc_ids	リスト		
	follower_ids	リスト		
	forum_topic_id	Ling		
	problem_id	Long		
	has_incidents	ブール値		
	due_at	DateTime		
	タグ	リスト		

エンティティ	フィールド	データ型	サポートされている演算子	コメント
	via	Struct		
	custom_fields	リスト		
	satisfaction_rating	Struct		
	sharing_agreement_ids	リスト		
	followup_ids	リスト		
	via_followup_source_id	Long		
	ticket_form_id	Long		
	brand_id	Long		
	allow_channelback	ブール値		
	allow_attachments	ブール値		
	is_public	ブール値		
	from_messaging_channel	ブール値		
	created_at	DateTime		
	updated_at	DateTime	EQUAL_TO	
	assignee_email	文字列		
	attribute_value_ids	リスト		

エンティティ	フィールド	データ型	サポートされている演算子	コメント
	collaborators	リスト		
	コメント	Struct		
	custom_status_id	Long		
	email_ccs	Struct		
	followers	Struct		
	macro_id	Long		
	macros_ids	リスト		
	metadata	Struct		
	safe_update	ブール値		
	updated_stamp	DateTime		
	via_id	Long		
	voice_comment	Struct		
	DML_STATUS	文字列		レコードの作成、更新、削除のステータスを追跡するために使用されるユーザー定義フィールド。
trigger-category	url	文字列		
	id	文字列		
	名前	文字列		

エンティティ	フィールド	データ型	サポートされている演算子	コメント
	updated_at	DateTime		
	created_at	DateTime		
	position	整数		
トリガー	url	文字列		
	id	Long		
	title	文字列		
	ACTIVE	ブール値	EQUAL_TO	
	updated_at	DateTime		
	created_at	DateTime		
	デフォルト	ブール値		
	actions	リスト		
	条件	Struct		
	説明	文字列		
	position	整数		
	raw_title	文字列		
	category_id	文字列	EQUAL_TO	
ユーザー	url	文字列		
	id	Long		
	external_id	文字列	EQUAL_TO	
	email	文字列		

エンティティ	フィールド	データ型	サポートされている演算子	コメント
	ACTIVE	ブール値		
	alias	文字列		
	chat_only	ブール値		
	custom_roll_id	Long		
	roll_type	整数		
	details	文字列		
	last_login_at	DateTime		
	サイト	文字列		
	locale_id	整数		
	moderator	ブール値		
	注意事項	文字列		
	名前	文字列		
	only_private_comments	ブール値		
	organization_id	Long		
	default_group_id	Long		
	phone	文字列		
	photo	Struct		
	remote_photo_url	文字列		
	restricted_agent	ブール値		

エンティティ	フィールド	データ型	サポートされている演算子	コメント
	ロール	文字列	EQUAL_TO	
	共有	ブール値		
	shared_agent	ブール値		
	タグ	リスト		
	署名	文字列		
	suspended	ブール値		
	ticket_restriction	文字列		
	time_zone	文字列		
	iana_time_zone			
	two_factor_auth_enabled			
	user_fields			
	verified	ブール値		
	report_csv	ブール値		
	created_at	DateTime		
	updated_at	DateTime	EQUAL_TO	
	permission_set	Long	EQUAL_TO	
	shared_phone_number	ブール値		

エンティティ	フィールド	データ型	サポートされている演算子	コメント
	DML_STATUS	文字列		レコードの作成、更新、削除のステータスを追跡するために使用されるユーザー定義フィールド。
表示	url	文字列		
	id	Long		
	title	文字列		
	ACTIVE	ブール値	EQUAL_TO	
	updated_at	DateTime		
	created_at	DateTime		
	デフォルト	ブール値		
	position	整数		
	説明	文字列		
	実行	Struct		
	制限	Struct		
	raw_title	文字列		
	条件	Struct		
	アクセス	文字列	EQUAL_TO	
	group_id	Long	EQUAL_TO	

> **Note**
>
> 構造体データ型とリストデータ型は、コネクタの応答で文字列データ型に変換されます。

パーティショニングクエリ

パーティションは Zendesk ではサポートされていません。

Zendesk 接続オプション

Zendesk の接続オプションは次のとおりです。

- ENTITY_NAME (文字列) - (必須) 読み取りに使用されます。Zendesk のオブジェクトの名前。
- API_VERSION (文字列) - (必須) 読み取りに使用されます。使用する Zendesk Rest APIバージョン。例: v2。
- SELECTED_FIELDS (リスト < 文字列>) - デフォルト: empty(SELECT *)。読み込みに使用されます。オブジェクトに選択する列。例: id、name、url、created_at
- FILTER_PREDICATE(文字列) - デフォルト: 空 読み込みに使用されます。Spark SQL形式である必要があります。例: group_id = 100
- QUERY(文字列) - デフォルト: 空 読み込みに使用されます。フル Spark SQLクエリ。例: SELECT「id,url FROM users WHERE role=\"end-user\"」
- PARTITION_FIELD (文字列) - 読み取りに使用されます。クエリをパーティション化するために使用するフィールド。デフォルトフィールドは、増分エクスポートcreated_atをサポートするエンティティ update_at (API では ticket-events 、 では) timeですticket-metric-events。
- LOWER_BOUND(文字列) - 読み取りに使用されます。選択したパーティションフィールドの包括的な下限値。
- UPPER_BOUND (文字列) - 読み取りに使用されます。選択したパーティションフィールドの排他的上限値。オプション。このオプションは、ジョブオプションで指定されていない場合、コネクタによって処理されます。デフォルト値 -2024-05-01T20:55:02.000Z
- NUM_PARTITIONS(整数) - デフォルト: 1。読み込みに使用されます。読み取り用のパーティションの数。オプション。このオプションは、ジョブオプションで指定されていない場合、コネクタによって処理されます。デフォルト値 : 1。
- IMPORT_DELETED_RECORDS (文字列) - デフォルト: FALSE。読み込みに使用されます。クエリ中に削除レコードを取得するには。

- ACCESS_TOKEN - リクエストで使用するアクセストークン。
- INSTANCE_URL - ユーザーがオペレーションを実行するURLインスタンス。例： https://{subdomain}/.zendesk.com

制限

Zendesk コネクタの制限は次のとおりです。

- オフセットベースのページ分割では、取得できるページ数は 100 に制限されますが、取得できるレコードの合計数は 10,000 であるため、推奨されません。ただし、Zendesk コネクタに実装されているカーソルベースのページ分割は、この制限を克服します。Zendesk では、EQUAL_TO フィルター演算子のみがサポートされていますAPI。

  この制限により、Zendesk コネクタではパーティショニングはサポートされていません。
- 「チケットイベント」エンティティの場合、レート制限は 1 分あたり 10 リクエストです。ジョブの実行 AWS Glue ETL中に、429 (リクエストが多すぎる) エラーが表示される場合があります。

Zoho CRM への接続

Zoho CRM は、販売、マーケティング、カスタマーサポートのアクティビティをまとめて、プロセス、ポリシー、人材を 1 つのプラットフォームに合理化する単一のリポジトリとして機能します。Zoho CRM は、あらゆるビジネスのタイプや規模の特定のニーズに合わせて簡単にカスタマイズできます。

Zoho CRM の開発者プラットフォームは、作業を自動化し、エンタープライズスタック間でデータを統合し、ウェブやモバイル用のカスタムソリューションを作成するために、ビジネス/企業向けにローコードツールとプロコードツールの適切に組み合わせて提供します。

トピック

- AWS Glue の Zoho CRM サポート
- 接続を作成および使用するための API オペレーションを含むポリシー
- Zoho CRM の設定
- Zoho CRM 接続の設定
- Zoho CRM エンティティからの読み取り
- Zoho CRM 接続オプション

- Zoho CRM コネクタの制限と注意事項

AWS Glue の Zoho CRM サポート

AWS Glue は、Zoho CRM を次のようにサポートします。

ソースとしてサポートされていますか?

はい – Sync と Async。AWS Glue ETL ジョブを使用して、Zoho CRM からデータをクエリできます。

ターゲットとしてサポートされていますか?

いいえ。

サポートされている Zoho CRM API バージョン

次の Zoho CRM API バージョンがサポートされています:

- v7

接続を作成および使用するための API オペレーションを含むポリシー

次のサンプル ポリシーで、接続の作成と使用に必要な AWS IAM 権限について説明します。新しいロールを作成する場合は、以下を含むポリシーを作成します。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:ListConnectionTypes",
                "glue:DescribeConnectionType",
                "glue:RefreshOAuth2Tokens"
                "glue:ListEntities",
                "glue:DescribeEntity"
            ],
            "Resource": "*"
        }
    ]
```

```
}
```

上記の方法を使用しない場合は、代わりに次のマネージド IAM ポリシーを使用します:

- AWSGlueServiceRole – さまざまな AWS Glue プロセスを実行するために必要なリソースへのアクセス権をユーザーに代わって付与します。これらのリソースには AWS Glue 、Amazon S3、IAM、CloudWatch Logs、および Amazon EC2 が含まれます。このポリシーで指定されたリソースの命名規則に従った場合、AWS Glue プロセスは必要なアクセス権限を使用できます。このポリシーは、通常、クローラ、ジョブ、開発エンドポイントを定義するときに指定されたロールにアタッチされます。
- AWSGlueConsoleFullAccess — ポリシーがアタッチされているアイデンティティが AWS マネージメントコンソールを使用するときは、AWS Glue リソースへのフルアクセスを許可します。このポリシーで指定されたリソースの命名規則に従った場合、ユーザーは完全なコンソール機能を使用できます。このポリシーは、通常 AWS Glue コンソールのユーザーにアタッチされています。

Zoho CRM の設定

AWS Glue を使用して Zoho CRM からデータを転送できるようにするには、次の要件を満たす必要があります:

最小要件

以下に、最小要件を示します。

- Zoho CRM アカウントを持っている。
- Zoho CRM アカウントで API アクセスが有効になっている。
- API コンソールの下に OAuth 認証情報を取得するための登録済みの API クライアントがあります。

Zoho CRM 接続の設定

グラントタイプは、AWS Glue が Zoho CRM と通信してデータへのアクセスをリクエストする方法を決定します。選択すると、接続を作成する前に満たす必要がある要件に影響します。Zoho CRM は、OAuth 2.0 の AUTHORIZATION_CODE のグラントタイプのみをサポートしています。

- このグラントタイプは、ユーザーを認証するためにサードパーティーの認証サーバーにユーザーをリダイレクトすることから、「three-legged」の OAuth と見なされます。AWS Glue コンソール経

由で接続を作成するときに使用されます。AWS Glue コンソールでは、ユーザーが Zoho CRM にリダイレクトされます。ここではユーザーがログインし、Zoho CRM インスタンスにアクセスするためのリクエストされた権限を Glue に許可する必要があります。

- ユーザーは、Zoho CRM で独自の接続アプリを作成し、AWS Glue コンソールを介して接続を作成するときに独自のクライアント ID や認証 URL、トークン URL、インスタンス URL を指定することを選択できます。このシナリオでは、引き続き Zoho CRM にリダイレクトされてログインし、リソースへアクセスするために AWS Glue を承認します。
- このグラントタイプは、更新トークンとアクセストークンになります。アクセストークンの有効期間は 1 時間のままなので、更新トークンを使用することでユーザーとやり取りすることなく自動的に更新される場合があります。
- 認可コード OAuth フロー用の接続アプリケーションの作成に関する Zoho CRM の公開ドキュメントについては、「認証」を参照してください。

Zoho CRM 接続を設定するには:

1. AWS Glue Glue Studio で、以下の手順に従って [データ接続] の下に接続を作成します。
   a. [接続タイプ] を選択する際に、[Zoho CRM] を選択します。
   b. 接続したい Zoho CRM インスタンスの `INSTANCE_URL` を指定します。
   c. ユーザークライアントアプリケーションのクライアント ID を指定します。
   d. ドロップダウンから適切な [認証 URL] を選択します。
   e. ドロップダウンから適切な [トークン URL] を選択します。
   f. 次のアクションを実行でき、AWS Glue がその権限を持つ AWS IAM ロールを選択します。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:DescribeSecret",
                "secretsmanager:GetSecretValue",
                "secretsmanager:PutSecretValue",
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterface",
                "ec2:DeleteNetworkInterface",
            ],
            "Resource": "*"
```

```
        }
    ]
}
```

g. AWS Glue でこの接続に使用する `secretName` を選択して、トークンを配置します。

h. ネットワークを使用する場合は、ネットワークオプションを選択します。

2. AWS Glue ジョブに関連付けられている IAM ロールに `secretName` を読み取るアクセス許可を付与します。
3. AWS Glue ジョブ設定で、[追加のネットワーク接続] として `connectionName` を指定します。

Zoho CRM エンティティからの読み取り

前提条件

読み取り元の Zoho CRM オブジェクト。オブジェクト名が必要になります。

Sync ソースに対応するエンティティ:

エンティティ	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
製品	はい	あり	あり	あり	はい
Quote	はい	あり	あり	あり	はい
発注書	はい	あり	あり	あり	はい
ソリューション	はい	あり	あり	あり	はい
電話	はい	あり	あり	あり	はい
タスク	はい	あり	あり	あり	はい
イベント	はい	あり	あり	あり	はい
Invoice	はい	あり	あり	あり	はい
アカウント	はい	あり	あり	あり	はい

エンティティ	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
連絡先	はい	あり	あり	あり	はい
Vendor	はい	あり	あり	あり	はい
キャンペーン	はい	あり	あり	あり	はい
Deal	はい	あり	あり	あり	はい
リード	はい	あり	あり	あり	はい
Custom Module	はい	あり	あり	あり	はい
Sales Order	はい	あり	あり	あり	はい
Price Books	はい	あり	あり	あり	はい
ケース	はい	あり	あり	あり	はい

例:

```
zoho_read = glueContext.create_dynamic_frame.from_options(
    connection_type="ZOHO",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "entityName",
        "API_VERSION": "v7",
        "INSTANCE_URL": "https://www.zohoapis.in/"
    }
```

Async ソースに対応するエンティティ:

エンティティ	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
製品	はい	いいえ	いいえ	あり	いいえ
Quote	はい	いいえ	いいえ	あり	いいえ
発注書	はい	いいえ	いいえ	あり	いいえ
ソリューション	はい	いいえ	いいえ	あり	いいえ
電話	はい	いいえ	いいえ	あり	いいえ
タスク	はい	いいえ	いいえ	あり	いいえ
イベント	はい	いいえ	いいえ	あり	いいえ
Invoice	はい	いいえ	いいえ	あり	いいえ
アカウント	はい	いいえ	いいえ	あり	いいえ
問い合わせ	はい	いいえ	いいえ	あり	いいえ
Vendor	はい	いいえ	いいえ	あり	いいえ
キャンペーン	はい	いいえ	いいえ	あり	いいえ
Deal	はい	いいえ	いいえ	あり	いいえ
リード	はい	いいえ	いいえ	あり	いいえ
Custom Module	はい	いいえ	いいえ	あり	いいえ
Sales Order	はい	いいえ	いいえ	あり	いいえ
Price Books	はい	いいえ	いいえ	あり	いいえ
ケース	はい	いいえ	いいえ	あり	いいえ

例:

```
zoho_read = glueContext.create_dynamic_frame.from_options(
    connection_type="ZOHO",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "entityName",
        "API_VERSION": "v7",
        "INSTANCE_URL": "https://www.zohoapis.in/",
        "TRANSFER_MODE": "ASYNC"
    }
```

Zoho CRM フィールドの詳細:

Zoho CRM では、サポートされているエンティティのメタデータを動的に取得するためのエンドポイントが提供されます。したがって、オペレータのサポートはデータ型レベルでキャプチャされます。

エンティティ	データ型	サポートされている演算子
Zoho エンティティ (すべてのエンティティ)	整数	!=, =, <, <=, >, >=, BETWEEN
	String	Like, =, !=
	BigInteger	!=, =, <, <=, >, >=, BETWEEN
	ブール値	=
	倍精度	!=, =, <, <=, >, >=, BETWEEN
	BigDecimal	!=, =, <, <=, >, >=, BETWEEN
	日付	!=, =, <, <=, >, >=, BETWEEN
	DateTime	!=, =, <, <=, >, >=, BETWEEN
	Struct	該当なし
	リスト	該当なし

パーティショニングクエリ

パーティショニングは Async モードではサポートされていません。

フィルターベースのパーティショニング (Sync モード):

Spark で同時実行を使用する場合は、追加の Spark オプション PARTITION_FIELD、LOWER_BOUND、UPPER_BOUND、および NUM_PARTITIONS を指定できます。これらのパラメータを使用すると、元のクエリは Spark タスクで同時に実行できるサブクエリの NUM_PARTITIONS の数に分割されます。

- PARTITION_FIELD: クエリのパーティション化に使用するフィールドの名前。
- LOWER_BOUND: 選択したパーティションフィールドの包括的な下限値。

  Datetime フィールドでは、Spark SQL クエリで使用される Spark タイムスタンプ形式を受け入れます。

  有効な値の例は次のとおりです:

  ```
  "2024-09-30T01:01:01.000Z"
  ```
- UPPER_BOUND: 選択したパーティションフィールドの排他的上限値。
- NUM_PARTITIONS: パーティション数。

例:

```
zoho_read = glueContext.create_dynamic_frame.from_options(
    connection_type="zohocrm",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "entityName",
        "API_VERSION": "v7",
        "PARTITION_FIELD": "Created_Time"
        "LOWER_BOUND": "2022-01-01T01:01:01.000Z"
        "UPPER_BOUND": "2024-01-01T01:01:01.000Z"
        "NUM_PARTITIONS": "10"
    }
```

Zoho CRM 接続オプション

Zoho CRM の接続オプションは次のとおりです:

- ENTITY_NAME (文字列) – (必須) 読み取りに使用されます。Zoho CRM のオブジェクトの名前。
- API_VERSION (文字列) – (必須) 読み取りに使用されます。使用する Zoho CRM Rest API バージョン。
- SELECTED_FIELDS(List<String>) – Default: empty(SELECT *). 読み込みに使用されます。オブジェクトに選択する列。
- FILTER_PREDICATE(文字列) – デフォルト: 空 読み込みに使用されます。Spark SQL 形式である必要があります。
- QUERY(文字列) - デフォルト: 空 読み込みに使用されます。完全な Spark SQL クエリ。
- PARTITION_FIELD(文字列) – 読み取りに使用されます。クエリをパーティション化するために使用するフィールド。
- LOWER_BOUND(文字列) - 読み取りに使用されます。選択したパーティションフィールドの包括的な下限値。
- UPPER_BOUND (文字列) - 読み取りに使用されます。選択したパーティションフィールドの排他的上限値。
- NUM_PARTITIONS(整数) – デフォルト: 1。読み込みに使用されます。読み取り用のパーティションの数。
- INSTANCE_URL (文字列) – (必須) 読み取りに使用されます。有効な Zoho CRM インスタンス URL。
- TRANSFER_MODE(文字列) - クエリを Async モードで実行するかどうかを示すために使用されます。

Zoho CRM コネクタの制限と注意事項

Zoho CRM コネクタの制限と注意事項は次のとおりです:

- API バージョン v7 では、最大 100,000 件のレコードをフェッチできます。Zoho のドキュメントを参照してください。
- イベントエンティティの場合、「会議」というラベルがZoho のドキュメントで説明されているように表示されます。
- Select All 機能の場合:
  - GET コールと POST コールの両方で、SaaS から最大 50 個のフィールドをフェッチできます。
  - 最初の 50 個のフィールドに属さない特定のフィールドのデータがある場合は、選択したフィールドのリストを手動で指定する必要があります。

  - 50 以上のフィールドが選択されている場合、50 を超えるフィールドはすべて調整され、Amazon S3 には NULL データが入ります。
  - フィルター式の場合、ユーザーが指定した 50 個のフィールドのリストに「id」と「Created_Time」がない場合、カスタム例外が発生し、ユーザーにこれらのフィールドを含めるように求められます。
- フィルター演算子は、データ型が同じであっても、フィールドに応じて異なる場合があります。したがって、SaaS プラットフォームでエラーをトリガーするフィールドには、別の演算子を手動で指定する必要があります。
- Sort By 機能の場合:
  - データはフィルター式なしで 1 つのフィールドでのみソートできますが、フィルター式が適用されると複数のフィールドでソートできます。
  - 選択したフィールドにソート順が指定されていない場合、データはデフォルトで昇順で取得されます。
- Zoho CRM コネクタでサポートされているリージョンは、米国、欧州、インド、オーストラリア、および日本です。
- Async 読み取り機能 制限事項:
  - Async モードでは、制限順序およびパーティショニングはサポートされていません。
  - Async モードでは、最大 500 ページ、1 ページあたり 200,000 レコードのデータを転送できます。
  - 1 分間隔では、ダウンロードできるリクエストは 10 件のみです。ダウンロード制限を超えると、システムは HTTP 429 エラーを返し、すべてのダウンロードリクエストを 1 分間一時停止し、その後処理が再開されます。
  - 一括ジョブが完了したら、ダウンロード可能なファイルには 1 日間のみアクセスできます。その後、エンドポイント経由でファイルにアクセスすることはできません。
  - エンドポイントを介して指定できる選択フィールドは最大 200 個です。エンドポイントで 200 を超える選択フィールドを指定すると、そのモジュールで使用可能なすべてのフィールドが自動的にエクスポートされます。
  - モジュールで作成された外部フィールドは、Bulk Read API ではサポートされていません。
  - この API エンドポイントでは、Sorting および `Group_by` 句はサポートされていません。
  - 機密ヘルスデータを含むフィールドの値は、コンプライアンス設定の [API を介したデータアクセスの制限] オプションが [無効] になっている場合にのみ取得されます。オプションが有効になっている場合、結果は [空] になります。
  - フィルタリング/基準の制限

- クエリで使用できる基準の最大数は 25 です。
- 複数行のテキストフィールドでのフィルタリング/基準はサポートされていません。

Zoom Meetings に対する接続

Zoom Meetings は、ビデオ会議、音声会議、ウェビナー、会議録音、ライブチャットに使用できるクラウドベースのビデオ会議プラットフォームです。

トピック

- Zoom Meetings 向けの AWS Glue サポート
- 接続を作成および使用するための API オペレーションを含むポリシー
- Zoom Meetings の設定
- Zoom Meetings クライアントアプリの設定
- Zoom Meetings 接続の設定
- Zoom Meetings エンティティからの読み取り
- Zoom Meetings 接続オプション
- Zoom Meetings の制限事項

Zoom Meetings 向けの AWS Glue サポート

AWS Glue は、次のように Zoom Meetings をサポートします。

ソースとしてサポートされていますか?

はい。AWS Glue ETL ジョブを使用して、Zoom Meetings からデータをクエリできます。

ターゲットとしてサポートされていますか?

いいえ。

サポートされている Zoom Meetings API バージョン

次の Zoom Meetings API バージョンがサポートされています。

- v2

接続を作成および使用するための API オペレーションを含むポリシー

次のサンプル ポリシーで、接続の作成と使用に必要な AWS IAM 権限について説明します。新しいロールを作成する場合は、以下を含むポリシーを作成します。

```
{
 "Version": "2012-10-17",
    "Statement": [
     {
            "Effect": "Allow",
            "Action": [
                "glue:ListConnectionTypes",
                "glue:DescribeConnectionType",
                "glue:RefreshOAuth2Tokens",
                "glue:ListEntities",
                "glue:DescribeEntity"
            ],
            "Resource": "*"
        }
    ]
 }
```

上記の方法を使用しない場合は、代わりに次のマネージド IAM ポリシーを使用します:

- AWSGlueServiceRole – さまざまな AWS Glue プロセスを実行するために必要なリソースへのアクセス権をユーザーに代わって付与します。これらのリソースには AWS Glue 、Amazon S3、IAM、CloudWatch Logs、および Amazon EC2 が含まれます。このポリシーで指定されたリソースの命名規則に従った場合、AWS Glue プロセスは必要なアクセス権限を使用できます。このポリシーは、通常、クローラ、ジョブ、開発エンドポイントを定義するときに指定されたロールにアタッチされます。
- AWSGlueConsoleFullAccess — ポリシーがアタッチされているアイデンティティが AWS マネージメントコンソールを使用するときは、AWS Glue リソースへのフルアクセスを許可します。このポリシーで指定されたリソースの命名規則に従った場合、ユーザーは完全なコンソール機能を使用できます。このポリシーは、通常 AWS Glue コンソールのユーザーにアタッチされています。

Zoom Meetings の設定

AWS Glue を使用して Zoom Meetings からデータを転送できるようにするには、次の要件を満たす必要があります。

最小要件

以下に、最小要件を示します。

- Zoom Meetings アカウントを持っている。
- Zoom アカウントで API アクセスが有効になっている。
- Zoom Meetings アカウントに OAuth2 アプリを作成している。この統合により、AWS Glue がお客様のアカウントに対して認証呼び出しを行うときに、お客様のデータに安全にアクセスするために使用する認証情報が提供されます。詳細については、「the section called “Zoom Meetings クライアントアプリの設定”」を参照してください。

これらの要件を満たすと、Zoom Meetings アカウントに AWS Glue を接続する準備が整います。一般的な接続では、Zoom Meetings で他に何もする必要はありません。

Zoom Meetings クライアントアプリの設定

1. Zoom App Marketplace にログインします。
2. [開発]、[アプリの構築] の順で選択します。
3. OAuth 2.0 ベースのアプリの [一般アプリ] を選択します。
4. [基本情報] ページで、アプリの名前、アプリの管理方法、アプリの認証情報、OAuth 情報など、アプリに関する情報を追加または更新します。
5. [アプリの管理方法の選択] セクションで、アプリの管理方法を確認します。
   a. [管理者管理]: アカウント管理者はアプリを追加および管理します
   b. [ユーザー管理]: 個々のユーザーがアプリを追加および管理します。アプリは、ユーザーの承認されたデータにのみアクセスできます。
6. [アプリ認証情報]: ビルドフローは、アプリのアプリ認証情報 (クライアント ID とクライアントシークレット) を自動的に生成します。
7. OAuth 情報セクションで、アプリの OAuth を設定します。
   a. [OAuth リダイレクト URL] (必須): リダイレクト URL またはエンドポイントを入力して、アプリと Zoom の間に OAuth を設定します。
   b. [厳格モード URL を使用する] (オプション)
   c. サブドメインチェック (オプション)
   d. [OAuth 許可リスト] (必須): Zoom が OAuth フローの有効なリダイレクトとして許可する一意の URL を追加します。

8. [スコープ] ページで、アプリが呼び出すことができる Zoom API メソッドを選択します。スコープは、ユーザーが利用できる情報と機能を定義します。次の詳細なスコープを選択します。
   - user:read:list_users:admin
   - zoom_rooms:read:list_rooms:admin
   - group:read:list_members:admin
   - group:read:administrator:admin
   - group:read:list_groups:admin
   - report:read:admin
   - role:read:list_roles、role:read:list_roles:admin

   スコープが追加されたら、[続行] を選択すると、アプリを使用する準備が整います。

OAuth 2.0 のセットアップの詳細については、「統合 (OAuth アプリ)」を参照してください。

Zoom Meetings 接続の設定

Zoom Meetings は、OAuth2 の AUTHORIZATION_CODE のグラントタイプをサポートしています。グラントタイプは、AWS Glue が Zoom Meetings と通信してデータへのアクセスをリクエストする方法を決定します。

- このグラントタイプは、ユーザーを認証するためにサードパーティーの認証サーバーにユーザーをリダイレクトすることから、「three-legged」の OAuth と見なされます。AWS Glue コンソール経由で接続を作成するときに使用されます。接続を作成するユーザーは、Zoom Meetings クライアントアプリケーションのクライアント ID やクライアントシークレットなどの OAuth 関連情報を指定する必要があります。AWS Glue コンソールでは、ユーザーが Zoom にリダイレクトされます。ここではユーザーがログインし、Zoom Meetings インスタンスにアクセスするためのリクエストされた権限を AWS Glue に許可する必要があります。
- ユーザーは、Zoom Meetings で独自の接続アプリを作成し、AWS Glue コンソールを介して接続を作成するときに独自のクライアント ID とクライアントシークレットを指定することを選択できます。このシナリオでは、引き続き Zoom Meetings にリダイレクトされてログインし、リソースへアクセスするために AWS Glue を承認します。
- このグラントタイプは、更新トークンとアクセストークンになります。アクセストークンの有効期間は短く、更新トークンを使用してユーザーとやり取りすることなく自動的に更新される場合があります。
- 認可コード OAuth フロー用の接続アプリの作成に関する Zoom Meetings の公開ドキュメントについては、「OAuth 2.0 を使用する」を参照してください。

Zoom Meetings 接続を設定するには：

1. AWS Secrets Manager で、次の詳細を含むシークレットを作成します。
   a. カスタマーマネージド接続アプリケーションの場合、シークレットには、USER_MANAGED_CLIENT_APPLICATION_CLIENT_SECRET を使用して接続されたアプリケーションのコンシューマーシークレットをキーとして含める必要があります。
   b. 注: AWS Glue で接続にシークレットを作成する必要があります。

1. AWS Glue Glue Studio で、以下の手順に従って [データ接続] の下に接続を作成します。
   a. [接続タイプ] を選択する際に、[Zoom Meetings] を選択します。
   b. 接続先の Zoom Meetings 環境を指定します。
   c. 次のアクションを実行でき、AWS Glue がその権限を持つ AWS IAM ロールを選択します。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:DescribeSecret",
                "secretsmanager:GetSecretValue",
                "secretsmanager:PutSecretValue",
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterface",
                "ec2:DeleteNetworkInterface",
            ],
            "Resource": "*"
        }
    ]
}
```

   d. AWS Glue でこの接続に使用する secretName を選択して、トークンを配置します。
   e. ネットワークを使用する場合は、ネットワークオプションを選択します。
2. AWS Glue ジョブに関連付けられている IAM ロールに secretName を読み取るアクセス許可を付与します。

Zoom Meetings エンティティからの読み取り

前提条件

読み取り元の Zoom Meetings オブジェクト。Group または Zoom Rooms などのオブジェクト名が必要です。

ソースに対応するエンティティ:

エンティティ	フィルタリング可能	制限をサポートする	Order By をサポートする	Select * をサポートする	パーティション分割をサポートする
Zoom Rooms	いいえ	はい	いいえ	はい	いいえ
グループ	いいえ	いいえ	いいえ	はい	いいえ
グループメンバー	はい	はい	いいえ	はい	いいえ
グループ管理者	いいえ	はい	いいえ	はい	いいえ
レポート (毎日)	はい	いいえ	いいえ	はい	いいえ
ロール	いいえ	いいえ	いいえ	はい	いいえ
ユーザー	はい	はい	いいえ	はい	いいえ

例:

```
zoom_read = glueContext.create_dynamic_frame.from_options(
    connection_type="zoom",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "organization",
        "API_VERSION": "v2"
    }
)
```

Zoom Meetings エンティティとフィールドの詳細：

Zoom Meetings は、選択したエンティティで使用可能なフィールドを動的にロードします。フィールドのデータ型に応じて、次のフィルター演算子がサポートされます。

エンティティ	フィールド	データ型	サポートされている演算子	
Zoom Room	ステータス	String	=	
	type	String	=	
	unassigned_rooms	ブール値	=	
	location_id	String	=	
	room_id	String	該当なし	
	activation_code	String	該当なし	
	id	String	該当なし	
	名前	String	該当なし	
	tag_ids	String	該当なし	
	query_name	String	該当なし	
日次レポート	month	日付	=	
	date	日付	該当なし	
	meeting_minutes	整数	該当なし	
	会議	整数	該当なし	
	new_users	整数	該当なし	
	participants	整数	該当なし	
	group_id	String	該当なし	

エンティティ	フィールド	データ型	サポートされている演算子	
ユーザー	created_at	DateTime	該当なし	
	部門	String	該当なし	
	email	String	該当なし	
	employee_unique_id	String	該当なし	
	first_name	String	該当なし	
	group_id	リスト	該当なし	
	host_key	String	該当なし	
	id	String	該当なし	
	im_group_ids	String	該当なし	
	last_client_version	String	該当なし	
	last_login_time	DateTime	該当なし	
	last_name	String	該当なし	
	plan_united_type	String	該当なし	
	custom_attributes	リスト	該当なし	
	pmi	BigInteger	該当なし	
	role_id	String	=	
	ステータス	String	=	
	timezone	String	該当なし	

エンティティ	フィールド	データ型	サポートされている演算子	
	type	整数	該当なし	
	verified	整数	該当なし	
	user_created_at	DateTime	該当なし	
	display_name	String	該当なし	
	phone_number	String	該当なし	
	language	String	該当なし	
	ライセンス	String	=	
グループ	id	String	該当なし	
	名前	String	該当なし	
	total_members	整数	該当なし	
グループメンバー	email	String	該当なし	
	first_name	String	該当なし	
	id	String	該当なし	
	last_name	String	該当なし	
	type	整数	該当なし	
	primary_group	ブール値	該当なし	
	member_id	String	該当なし	
グループ管理者	id	String	該当なし	
	email	String	該当なし	
	名前	String	該当なし	

エンティティ	フィールド	データ型	サポートされている演算子	
ロール	description	String	該当なし	
	id	String	該当なし	
	名前	String	該当なし	
	total_members	整数	該当なし	
	type	String	=	

パーティショニングクエリ

Zoom Meetings は、フィルターベースのパーティショニングまたはレコードベースのパーティショニングをサポートしていません。

Zoom Meetings 接続オプション

Zoom Meetings の接続オプションは次のとおりです。

- `ENTITY_NAME`(文字列) – (必須) 読み取りに使用されます。Zoom Meetings エンティティの名前。例えば、`group` と指定します。
- `API_VERSION`(文字列) – (必須) 読み取りに使用されます。使用する Zoom Meetings Rest API バージョン。Zoom Meetings は現在バージョン v2 のみをサポートしているため、この値は v2 になります。
- `SELECTED_FIELDS`(List<String>) – Default: empty(SELECT *). 読み込みに使用されます。選択したエンティティに対して選択する列のカンマ区切りリスト。
- `FILTER_PREDICATE`(文字列) – デフォルト: 空 読み込みに使用されます。Spark SQL 形式である必要があります。
- `QUERY`(文字列) – デフォルト: 空 読み込みに使用されます。完全な Spark SQL クエリ。

Zoom Meetings の制限事項

以下は、Zoom Meetings の制限事項または注意事項は次のとおりです。

- Zoom Meetings は orderby をサポートしていません。

- Zoom Meetings では、必要な条件を満たすフィールドがないため、フィルターベースのパーティション分割はサポートされていません。
- Zoom Meetings では、ページ分割の制限とオフセットベースのページ分割がサポートされていないため、レコードベースのパーティション分割はサポートされていません。

独自の JDBC ドライバーを使用した JDBC 接続の追加

JDBC 接続を使用する際に、独自の JDBC ドライバーを使用できます。AWS Glue クローラーが使用するデフォルトドライバーがデータベースに接続できない場合、独自の JDBC ドライバーを使用できます。例えば、Postgres データベースで SHA-256 を使用したいが、古い Postgres ドライバーがこれをサポートしていない場合は、独自の JDBC ドライバーを使用できます。

サポート対象データソース

サポート対象データソース	サポートされていないデータソース
MySQL	Snowflake
Postgres	
Oracle	
Redshift	
SQL Server	
Aurora*	

* ネイティブの JDBC ドライバーが使用されている場合にサポートされます。すべてのドライバー機能を使用できるわけではありません。

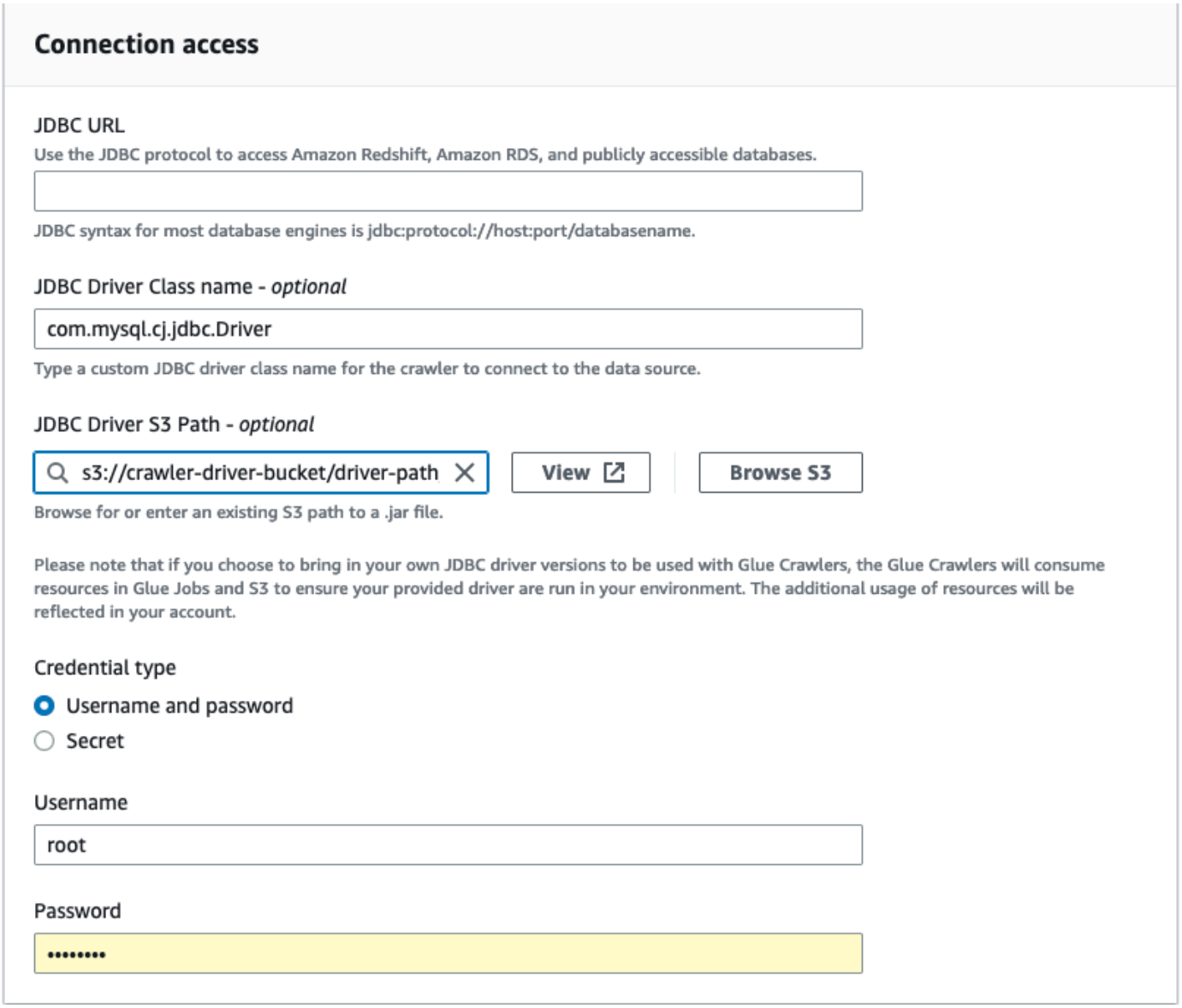
JDBC 接続への JDBC ドライバーの追加

> **Note**
>
> 独自の JDBC ドライバーバージョンを導入する場合、AWS Glue クローラーは AWS Glue ジョブと Amazon S3 バケットのリソースを使用して、用意したドライバーが自分の環境で実行されるようにします。リソースの追加使用量はアカウントに反映されます。AWS Glue

クローラーとジョブのコストは、請求の AWS Glue カテゴリに含まれます。さらに、独自の JDBC ドライバーを用意しても、クローラーがドライバーの機能をすべて活用できるわけではありません。

独自の JDBC ドライバーを JDBC 接続に追加するには:

1. JDBC ドライバーファイルを Amazon S3 の場所に追加します。バケットおよび/またはフォルダを作成または使用できます。
2. AWS Glue コンソールで、左側のメニューの [データカタログ] の [接続] を選択し、新しい接続を作成します。
3. [接続プロパティ] のフィールドに入力し、[接続タイプ] で JDBC を選択します。
4. [接続アクセス] に、[JDBC URL] と [JDBC ドライバーのクラス名] を入力します (オプション)。ドライバークラス名は、AWS Glue クローラーがサポートするデータソースのものである必要があります。

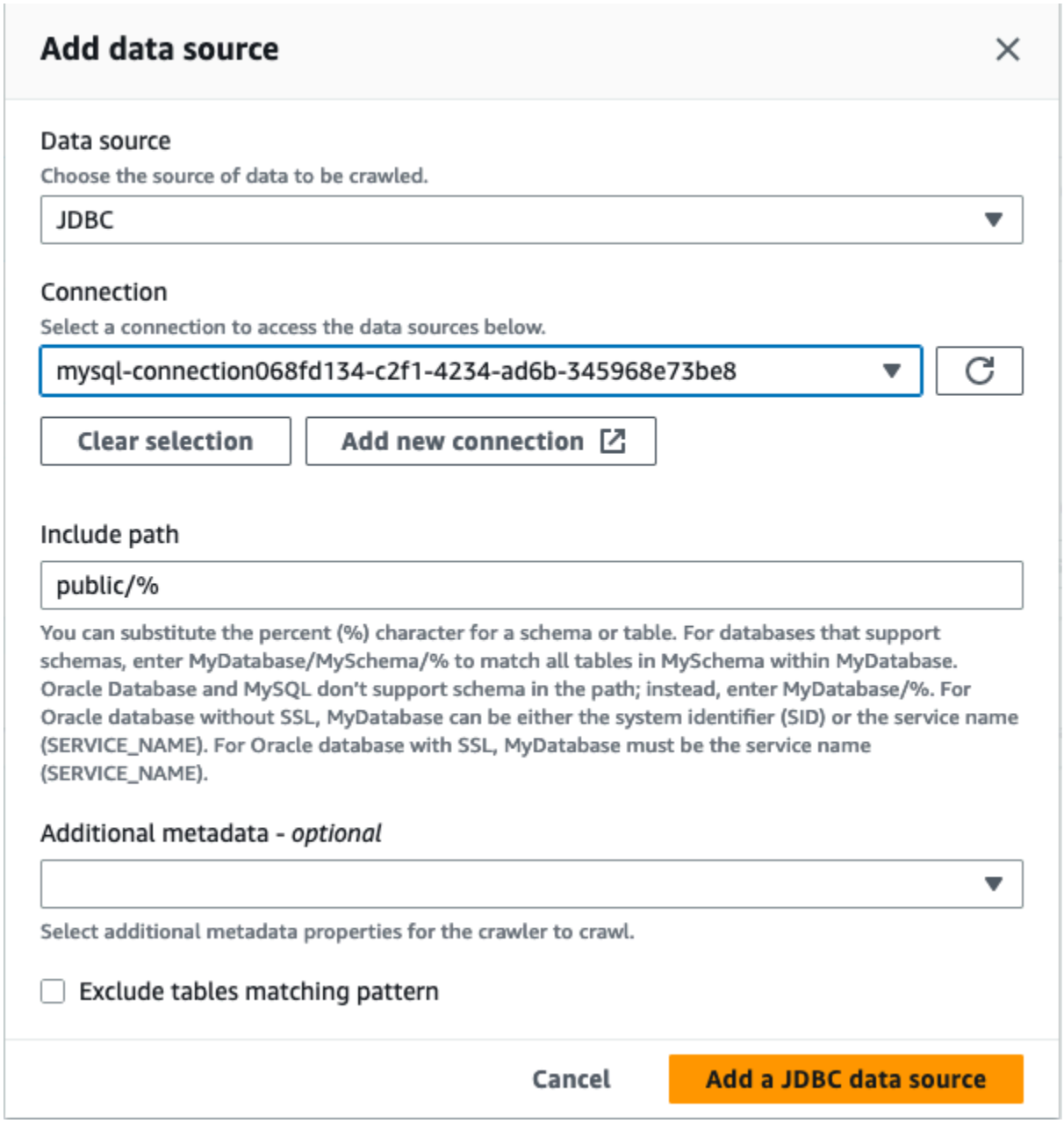


5. JDBC ドライバーが置かれている Amazon S3 パスを [JDBC ドライバーの Amazon S3 パス] で選択します (オプションフィールド)。
6. ユーザー名とパスワード、またはシークレットを入力する場合は、認証情報タイプのフィールドに入力します。完了したら、[接続を作成] を選択します。

> **Note**
>
> テスト接続は現在サポートされていません。用意した JDBC ドライバーを使用してデータソースをクローリングする場合、クローラーはこのステップをスキップします。

7. 新しく作成した接続をクローラーに追加します。AWS Glue コンソールで、左側のメニューの [データカタログ] で [クローラー] を選択し、新しいクローラーを作成します。

8. [クローラーの追加] ウィザードの [ステップ 2] で [データソースの追加] を選択します。



9. データソースとして [JDBC] を選択し、これまでの手順で作成した接続を選択します。完了
10. AWS Glue クローラーで独自の JDBC ドライバーを使用するには、クローラーで使用されるロールに次の権限を追加します。

    - ジョブアクション `CreateJob`、`DeleteJob`、`GetJob`、`GetJobRun`、`StartJobRun` にアクセス権限を付与します。
    - IAM アクション `iam:PassRole` にアクセス権限を付与します。

- Amazon S3 アクション `s3:DeleteObjects`、`s3:GetObject`、`s3:ListBucket`、`s3:PutObject` にアクセス権限を付与します。
- サービスプリンシパルに IAM ポリシーのバケット/フォルダへのアクセス権限を付与します。

IAM ポリシーの例:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject",
                "s3:ListBucket",
                "s3:DeleteObject"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket/driver-parent-folder/driver.jar",
                "arn:aws:s3:::amzn-s3-demo-bucket"
            ]
        }
    ]
}
```

AWS Glue クローラーは、_glue_job_crawler と _crawler の 2 つのフォルダを作成します。

ドライバー jar が `s3://amzn-s3-demo-bucket/driver.jar"` フォルダにある場合は、次のリソースを追加します:

```
"Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket/_glue_job_crawler/*",
     "arn:aws:s3:::amzn-s3-demo-bucket/_crawler/*"
            ]
```

ドライバー jar が `s3://amzn-s3-demo-bucket/tmp/driver/subfolder/driver.jar"`フォルダにある場合は、次のリソースを追加します:

```
"Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket/tmp/_glue_job_crawler/*",
    "arn:aws:s3:::amzn-s3-demo-bucket/tmp/_crawler/*"
            ]
```

11. VPC を使用している場合は、インターフェイスエンドポイントを作成し、これをルートテーブルに追加して AWS Glue へのアクセスを許可する必要があります。詳細については、「AWS Glue 用のインターフェイス VPC エンドポイントの作成」を参照してください。
12. データカタログで暗号化を使用している場合は、AWS KMS インターフェイスエンドポイントを作成し、これをルートテーブルに追加します。詳細については、「AWS KMS 用の VPC エンドポイントの作成」を参照してください。

AWS Glue Studio でのカスタムコネクタと接続の使用

AWS Glue には、JDBC 接続を使用する最も一般的なデータストア (Amazon Redshift、Amazon Aurora、Microsoft SQL Server、MySQL、MongoDB、PostgreSQL など) に対するサポートが組み込まれています。また、AWS Glue では、抽出、変換、ロード (ETL) ジョブ用にカスタム JDBC ドライバも使用できます。SaaS アプリケーションなど、ネイティブにサポートされていないデータストアに対しては、コネクタを使用することができます。

コネクタ とは AWS Glue Studio 内でデータストアに対するアクセスを支援するための、オプションのコードパッケージです。AWS Marketplace で提供されている複数のコネクタをサブスクライブすることができます。

ETL ジョブの作成時には、ネイティブにサポートされているデータストアを使用するか、AWS Marketplace からのコネクタ、または独自のカスタムコネクタを使用できます。コネクタを使用するには、最初にコネクタのための接続を作成する必要があります。接続には、特定のデータストアに接続するために必要なプロパティが含まれます。ETL ジョブでは、データソースおよびデーターゲットとの接続を使用します。コネクタと接続は、データストアへのアクセスを容易にするために連携して動作します。

コネクタの接続を作成する際に、次の接続を使用できます。

- Amazon Aurora – セキュリティ、バックアップと復元、メモリ内アクセラレーションを組み込んだ、スケーラブルで高性能なリレーショナルデータベースエンジン。

- Amazon DocumentDB – MongoDB と SQL API をサポートする、スケーラブルで可用性の高いフルマネージドのドキュメントデータベースサービス。
- Amazon Redshift – MongoDB および SQL API をサポートする、スケーラブルで可用性の高いフルマネージドのドキュメントデータベースサービス。
- Azure SQL – スケーラブルで信頼性が高く安全なデータストレージと管理機能を提供する、Microsoft Azure のクラウドベースのリレーショナルデータベースサービス。
- Cosmos DB – スケーラブルで高性能なデータストレージとクエリ機能を提供する、Microsoft Azure のグローバルに分散されたクラウドデータベースサービス。
- Google BigQuery – 大規模なデータセットに対して高速 SQL クエリを実行するためのサーバーレスクラウドデータウェアハウス。
- JDBC – 接続と、データ接続とのインタラクションに Java API を使用するリレーショナルデータベース管理システム (RDBMS)。
- Kafka – リアルタイムのデータストリーミングとメッセージングに使用されるオープンソースのストリーム処理プラットフォーム。
- MariaDB – 強化されたパフォーマンス、スケーラビリティ、および機能を提供する、コミュニティによって開発された MySQL のフォーク。
- MongoDB – 高いスケーラビリティ、柔軟性、パフォーマンスを提供する、クロスプラットフォームのドキュメント指向データベース。
- MongoDB Atlas – MongoDB のデプロイの管理とスケーリングを簡素化する、MongoDB のクラウドベースの Database as a Service (DBaaS) オファリング。
- Microsoft SQL Server – 堅牢なデータストレージ、分析、レポート機能を提供する、Microsoft のリレーショナルデータベース管理システム (RDBMS)。
- [Mixpanel] – ユーザーがウェブサイト、モバイルアプリケーション、その他のデジタル製品をどのように操作するかを企業が分析できるようにする分析プラットフォーム。
- MySQL – ウェブアプリケーションで広く使用されており、その信頼性とスケーラビリティで知られる、オープンソースのリレーショナルデータベース管理システム (RDBMS)。
- ネットワーク- ネットワークデータソースは、データ統合プラットフォームによってアクセスできる、ネットワークからアクセス可能なリソースまたはサービスを表します。
- OpenSearch – OpenSearch データソースは、OpenSearch が接続してデータを取り込むことができるアプリケーションです。
- Oracle – 堅牢なデータストレージ、分析、レポート機能を提供する、Oracle Corporation のリレーショナルデータベース管理システム (RDBMS)。

- PostgreSQL – 堅牢なデータストレージ、分析、レポート機能を提供する、オープンソースのリレーショナルデータベース管理システム (RDBMS)。
- [Salesforce] – Salesforce は、販売、カスタマーサービス、e コマースなどに役立つカスタマーリレーションシップ管理 (CRM) ソフトウェアを提供します。Salesforce ユーザーの場合は、AWS Glue を Salesforce アカウント に接続できます。次に、ETL ジョブのデータソースまたは送信先として Salesforce を使用できます。これらのジョブを実行して、Salesforce と AWS サービス、またはその他のサポートされているアプリケーション間でデータを転送します。
- SAP HANA – 高速データ処理、高度な分析、リアルタイムのデータ統合を提供する、インメモリデータベースおよび分析プラットフォーム。
- Snowflake – スケーラブルで高性能なデータストレージと分析サービスを提供するクラウドベースのデータウェアハウス。
- Teradata – 高性能のデータストレージ、分析、レポート機能を提供する、リレーショナルデータベース管理システム (RDBMS)。
- Vertica – 高速なクエリパフォーマンス、高度な分析、およびスケーラビリティを提供する、ビッグデータ分析用に設計されたカラム指向の分析データウェアハウス。

カスタムコネクタを作成する

独自のコネクタを作成し、そのコネクタのコードを AWS Glue Studio にアップロードすることもできます。

カスタムコネクタは、AWS Glue Spark ランタイム API を介して AWS Glue Studio に組み込まれます。Spark、Athena、または JDBC インターフェイスと準拠している任意のコネクタを、AWS Glue Spark ランタイムを使用してプラグインすることができます。これにより、カスタムコネクタで使用できる任意の接続オプションを渡すことができます。

AWS Glue 接続でのすべての接続プロパティをカプセル化して、ETL ジョブにその接続名を指定することができます。Data Catalog 接続を統合することで、単一の Spark アプリケーションからの複数の呼び出しや、異なるアプリケーション間での同じ接続プロパティの使用ができるようにします。

接続には、他のオプションを指定することもできます。AWS Glue Studio が生成するジョブスクリプトには、指定された接続オプションを使用してコネクタをプラグインする接続を使用する `Datasource` エントリが含まれています。例:

```
Datasource = glueContext.create_dynamic_frame.from_options(connection_type =
"custom.jdbc", connection_options = {"dbTable":"Account","connectionName":"my-custom-
jdbc-
```

```
connection"}, transformation_ctx = "DataSource0")
```

AWS Glue Studio にカスタムコネクタを追加する方法

1. カスタムコネクタ用のコードを作成します。詳しくは、「カスタムコネクタの開発 」を参照してください。
2. コネクタに、AWS Glue 機能に対するサポートを追加します。以下に、これらの機能について、さらにそれらが AWS Glue Studio で生成されたジョブスクリプトでどのように使用されるかに関する例をいくつか挙げてみます。

   - データ型のマッピング – コネクタは、基盤データストアから列を読み込む際に、列をタイプキャストすることができます。例えば {"INTEGER":"STRING"} の dataTypeMapping では、レコードの解析と DynamicFrame の構築時に、Integer 型のすべての列を String 型の列に変換します。これにより、ユーザーは任意のタイプに列をキャストすることができます。

     ```
     DataSource0 = glueContext.create_dynamic_frame.from_options(connection_type
     = "custom.jdbc", connection_options = {"dataTypeMapping":{"INTEGER":"STRING"}",
     connectionName":"test-connection-jdbc"}, transformation_ctx = "DataSource0")
     ```

   - 並列読み取りのパーティション化 – AWS Glue は、列にあるデータを分割することで、データストアから並列データを読み取れるようにします。パーティション列、パーティションの下限、パーティションの上限、およびパーティション数を指定する必要があります。この機能により、データの並列処理と、Spark アプリケーションに割り当てる複数の Spark エグゼキュータの使用が可能になります。

     ```
     DataSource0 = glueContext.create_dynamic_frame.from_options(connection_type
     = "custom.jdbc", connection_options = {"upperBound":"200","numPartitions":"4",
     "partitionColumn":"id","lowerBound":"0","connectionName":"test-connection-jdbc"},
     transformation_ctx = "DataSource0")
     ```

   - AWS Secrets Manager を使用した認証情報の保存 – Data Catalog 接続には、AWS Secrets Manager に保存されているシークレットのための secretId も含めることができます。AWS シークレットにより、承認と認証に関する情報を安全に保存し、また、AWS Glue の実行時にその情報を提供できます。または、以下に示すように、Spark スクリプトから secretId を指定することもできます。

     ```
     DataSource = glueContext.create_dynamic_frame.from_options(connection_type
     = "custom.jdbc", connection_options = {"connectionName":"test-connection-jdbc",
     ```

```
 "secretId"-> "my-secret-id"}, transformation_ctx = "DataSource0")
```

- 行述語と列射影によるソースデータのフィルタリング – AWS Glue Spark ランタイムでは、行述語と列射影を使用しながら、SQL クエリをプッシュダウンすることでソースにあるデータをフィルタリングすることもできます。これにより ETL ジョブは、フィルタリングされたデータを、プッシュダウンをサポートするデータストアからより迅速にロードできます。`SELECT id, name, department FROM department WHERE id < 200.` は、JDBC データ・ソースにプッシュダウンされた SQL クエリの例です。

```
DataSource = glueContext.create_dynamic_frame.from_options(connection_type =
"custom.jdbc", connection_options = {"query":"SELECT id, name, department FROM
 department
WHERE id < 200","connectionName":"test-connection-jdbc"}, transformation_ctx =
"DataSource0")
```

- ジョブのブックマーク – AWS Glue は、JDBC ソースからのデータの増分ロードをサポートしています。AWS Glue は、最後に処理された (データストアからの) レコードを追跡し、後続の ETL ジョブ実行で新しいデータレコードを処理します。ジョブのブックマークは、その列が順番に増減しているのであれば、ブックマークキーのデフォルト列としてプライマリキーを使用します。ジョブのブックマークの詳細については、AWS Glue デベロッパーガイドの「ジョブ ブックマーク」を参照してください。

```
DataSource0 = glueContext.create_dynamic_frame.from_options(connection_type =
"custom.jdbc", connection_options = {"jobBookmarkKeys":["empno"],
 "jobBookmarkKeysSortOrder"
:"asc", "connectionName":"test-connection-jdbc"}, transformation_ctx =
 "DataSource0")
```

3. カスタムコネクタを JAR ファイルとしてパッケージ化し、そのファイルを Amazon S3 にアップロードします。
4. カスタムコネクタをテストします。詳細については、GitHub の「Glue カスタムコネクタ: ローカル検証テストガイド」で手順を参照してください。
5. AWS Glue Studio コンソールのナビゲーションペインで、[Connectors] (コネクタ) をクリックします。
6. [Connectors] (コネクタ) ページで、[Create custom connector] (カスタムコネクタを作成) をクリックします。
7. [Create custom connector] (カスタムコネクタの作成) ページで、次の情報を入力します。

   - カスタムコード JAR ファイルの Amazon S3 内ロケーションへのパス。

- AWS Glue Studio によって使用されるコネクタの名前。
- コネクタのタイプ (JDBC、Spark、または Athena)。
- コネクタを使用するために AWS Glue Studio が呼び出す、カスタムコード内のエントリポイントの名前。
  - JDBC コネクタの場合、このフィールドは JDBC ドライバのクラス名です。
  - Spark コネクタの場合、このフィールドは、データソースの完全修飾クラス名、またはそのエイリアスである必要があります。これは、`format` 演算子を使用して Spark データソースをロードする際に使用します。
- (JDBC のみ) データストアの JDBC 接続で使用されるベース URL。
- (オプション) カスタムコネクタの説明。

8. [Create connector] (コネクターを作成) をクリックします。
9. コネクタ用の接続を作成する の説明を参考に、[Connectors] (コネクタ) ページで、そのコネクタを使用する接続を作成します。

AWS Glue Studio にコネクタを追加する

コネクタとは、データストアと AWS Glue 間の通信を容易にするための、一連のコードのことです。AWS Marketplace で提供されているコネクタをサブスクライブするか、独自のカスタムコネクタを作成することが可能です。

AWS Marketplace コネクタをサブスクライブする

AWS Glue Studio を使用すると、AWS Marketplace からコネクタを簡単に導入できます。

AWS Marketplace から AWS Glue Studio にコネクタを追加する方法

1. AWS Glue Studio コンソールのナビゲーションペインで、[Connectors] (コネクタ) をクリックします。
2. [Connectors] (コネクタ) ページで、[Go to AWS Marketplace] をクリックします。
3. AWS Marketplace の [Featured products] (注目の製品) で、使用するコネクタを選択します。表示されているコネクタの 1 つを選択するか、または検索することができます。コネクタを、名前またはタイプにより検索したり、オプションを使用して検索結果を絞り込むことができます。

   表示されているコネクタのいずれかを使用するには、[View product] (製品を表示する) をクリックします。検索結果からコネクタを見つけた場合は、そのコネクタの名前を選択します。

4. コネクタの製品ページで、そのコネクタに関するタブを開いて情報を表示します。そのコネクタを購入する場合は、[Continue to Subscribe] (続行してサブスクライブする) をクリックします。
5. 支払い情報を入力し、[Continue to Configure] (設定に進む) をクリックします。
6. [Configure this software] (このソフトウェアを設定する) ページで、デプロイの方法と使用するコネクタのバージョンを選択します。[Continue to Launch] (続行して起動する) をクリックします。
7. [Launch this software] (このソフトウェアを起動する) ページでは、コネクタプロバイダから提供される [Usage Instructions] (使用手順) を確認することができます。次に進む準備ができたら、[AWS Glue Studio で接続をアクティブにする] を選択します。

   少し待機すると、コンソールに、AWS Glue Studio の [Create marketplace connection] (マーケットプレイス接続の作成) ページが表示されます。
8. コネクタ用の接続を作成する の説明を参考に、このコネクタを使用する接続を作成します。

   または、[Activate connector only] (アクティブなコネクタのみ) を選択し、この時点での接続の作成をスキップすることも可能です。後にコネクタを使用する際には、先に接続を作成する必要があります。

コネクタ用の接続を作成する

AWS Glue 接続は、特定のデータストアの接続情報を保存する Data Catalog オブジェクトです。接続は、ログイン認証情報、URI文字列、仮想プライベートクラウド (VPC) 情報などを保存します。データカタログで接続を作成することで、クローラーまたはジョブを作成するたびに、すべての接続の詳細を指定する必要がなくなります。

コネクタのために接続を作成するには

1. AWS Glue Studio コンソールで、コンソールナビゲーションペインでコネクタを選択します。[接続] セクションで、[接続を作成] を選択します。
2. [データ接続を作成] ウィザードのステップ 1 で、接続を作成するデータソースを選択します。使用可能なデータソースを表示するには、以下のような複数の方法があります。

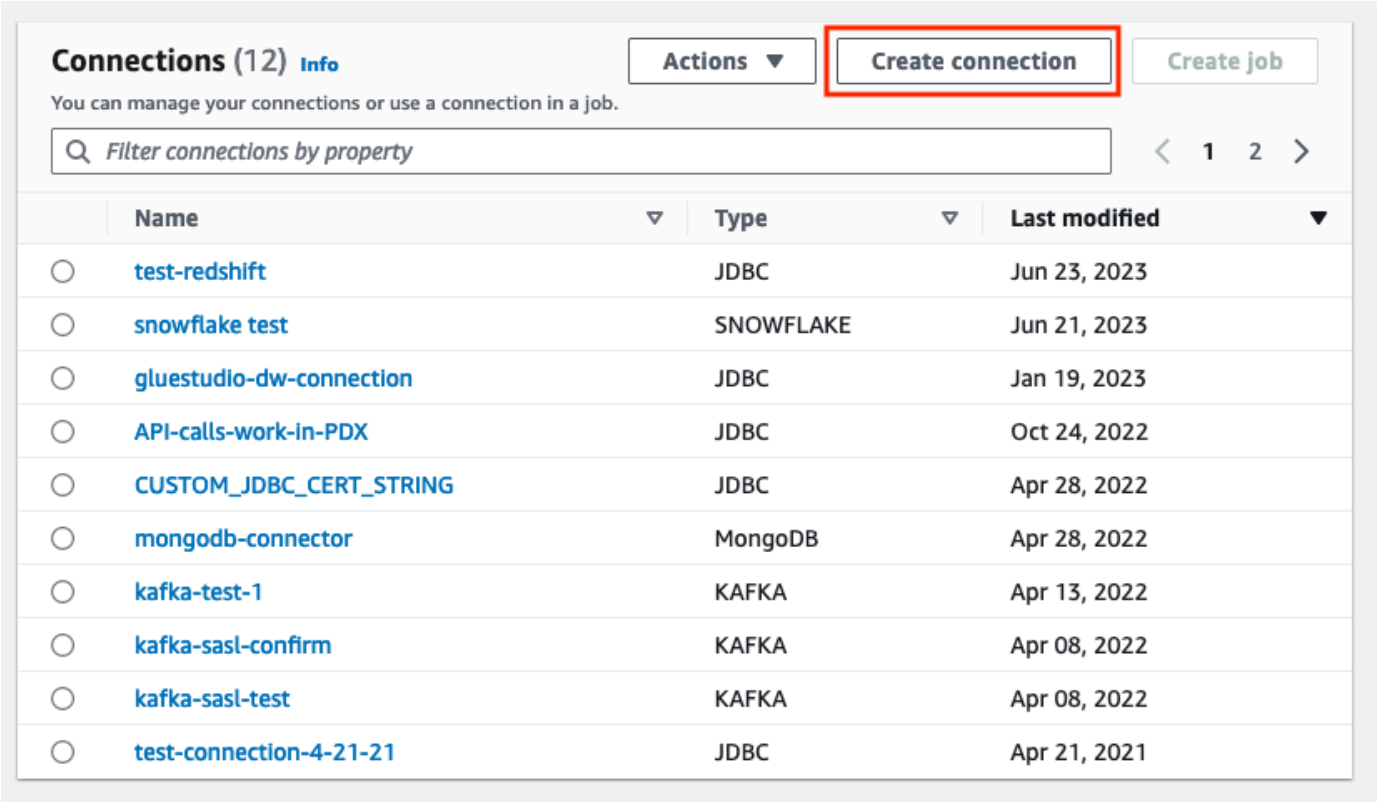
   - タブを選択して、使用可能なデータソースをフィルタリングします。デフォルトでは、[すべてのコネクタ] が選択されています。
   - [リスト] に切り替えてデータソースをリストとして表示するか、または [グリッド] に切り替えてグリッドレイアウトで使用可能なコネクタを表示します。

- データソースのリストを絞り込むには、検索バーを使用します。入力すると、検索に一致するソースが表示され、一致しないソースはビューから削除されます。

データソースを選択したら、[次へ] を選択します。

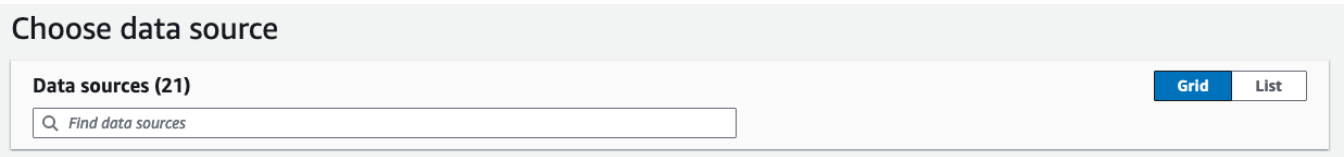
3. ウィザードのステップ 2 で接続を設定します。

接続の詳細を入力します。選択したコネクタのタイプに応じて、追加情報の入力を促されます。



4. [データ接続を作成] ウィザードのステップ 1 で、接続を作成するデータソースを選択します。使用可能なデータソースを表示するには、いくつかの方法があります。デフォルトでは、使用可能なすべてのデータソースがグリッドレイアウトで表示されます。以下の操作も可能です。

- [リスト] に切り替えてデータソースをリストとして表示するか、または [グリッド] に切り替えてグリッドレイアウトで使用可能なコネクタを表示します。
- データソースのリストを絞り込むには、検索バーを使用します。入力すると、検索に一致するソースが表示され、一致しないソースはビューから削除されます。

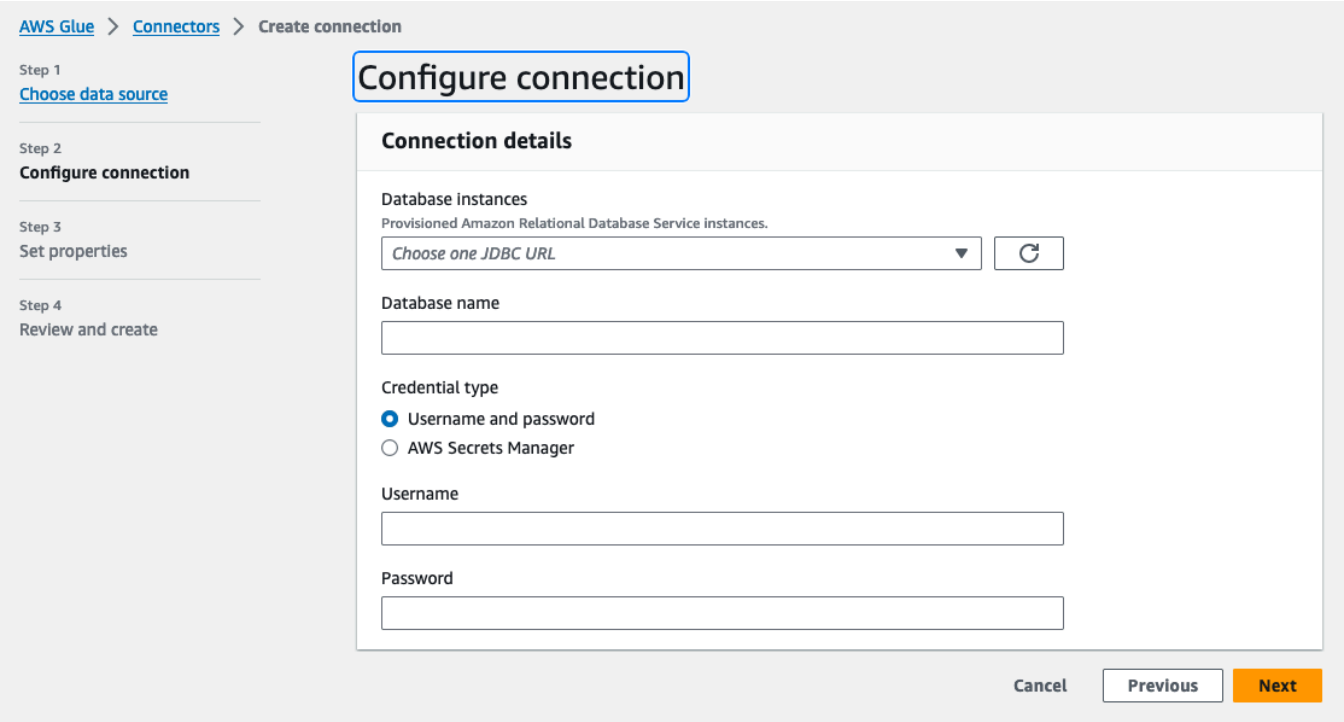


データソースを選択したら、[次へ] を選択します。

5. ウィザードのステップ 2 で接続を設定します。

   接続の詳細を入力します。選択したコネクタの種類によっては、追加の接続情報の入力が必要になる場合があります。これには次が含まれる場合があります。

   - [接続の詳細] – これらのフィールドは、接続先のデータソースに応じて変わります。例えば、Amazon DocumentDB データベースに接続する場合は、Amazon DocumentDB を入力しますURL。に接続する場合は Amazon Aurora、データベースインスタンスを選択し、データベース名を入力します。以下は、 に必要な接続の詳細です Amazon Aurora。



   - 認証情報タイプ – [ユーザー名とパスワード]、または [AWS Secrets Manager] のいずれかを選択します。要求された認証情報を入力します。
   - を使用するコネクタの場合はJDBC、データストアJDBCURLの の作成に必要な情報を入力します。
   - 仮想プライベートクラウド (VPC) を使用する場合は、 のネットワーク情報を入力しますVPC。

6. ウィザードのステップ 3 で接続のプロパティを設定します。このステップのオプション部分として、説明とタグを追加できます。名前は必須であり、デフォルト値が事前に入力されています。[Next (次へ)] を選択します。

7. 接続ソース、詳細、プロパティを確認します。変更を加える必要がある場合は、ウィザードのステップで [編集] を選択します。準備ができたら、[接続を作成] を選択します。

   [Create connection] (接続の作成) を選択します。

   再び [Connectors] (コネクタ) ページが表示され、作成された接続が情報バナーに表示されます。 AWS Glue Studio ジョブで接続を使用することができるようになりました。

Kafka 接続を作成する

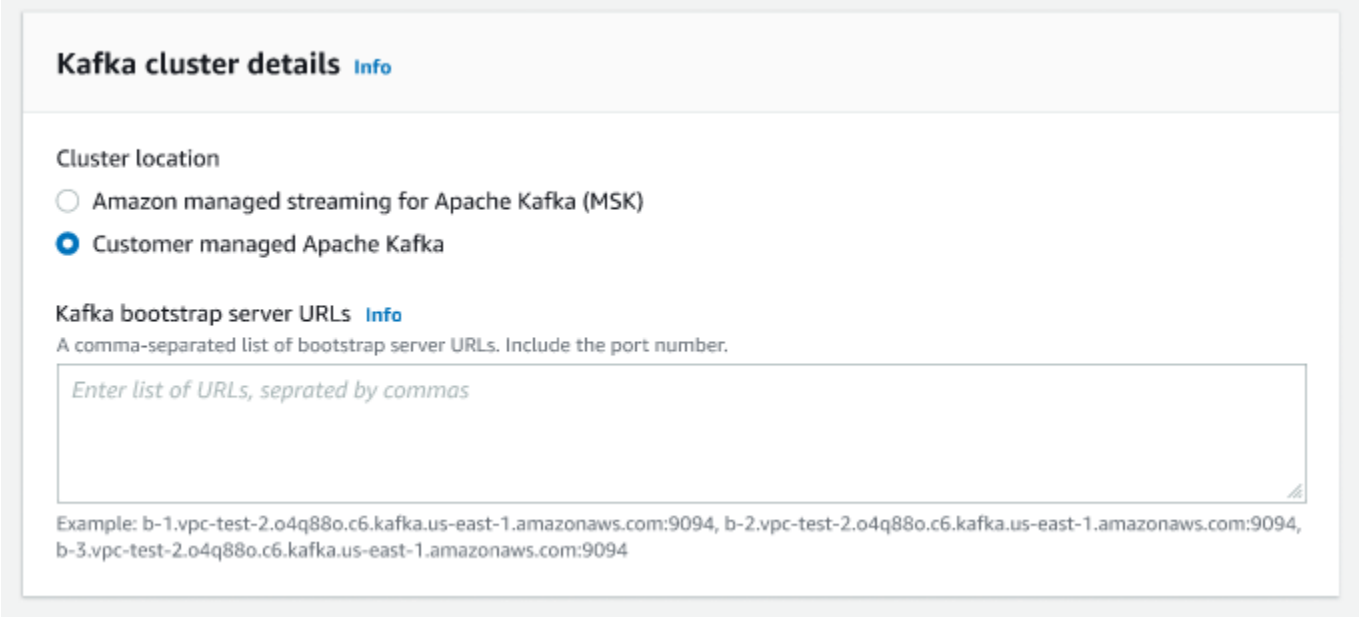
Kafka 接続を作成するときには、ドロップダウンメニューから Kafka を選択すると追加設定が表示され、構成できるようになります。

- Kafka クラスターの詳細
- 認証
- 暗号化
- ネットワークオプション

Kafka クラスターの詳細を設定する

1. クラスターの場所を選択します。Apache Kafka (MSK) クラスターの Amazon マネージドストリーミングまたはカスタマーマネージド Apache Kafka クラスターから選択できます。Apache Kafka の Amazon Managed ストリーミングの詳細については、「Apache Kafka の Amazon マネージドストリーミング (MSK)」を参照してください。

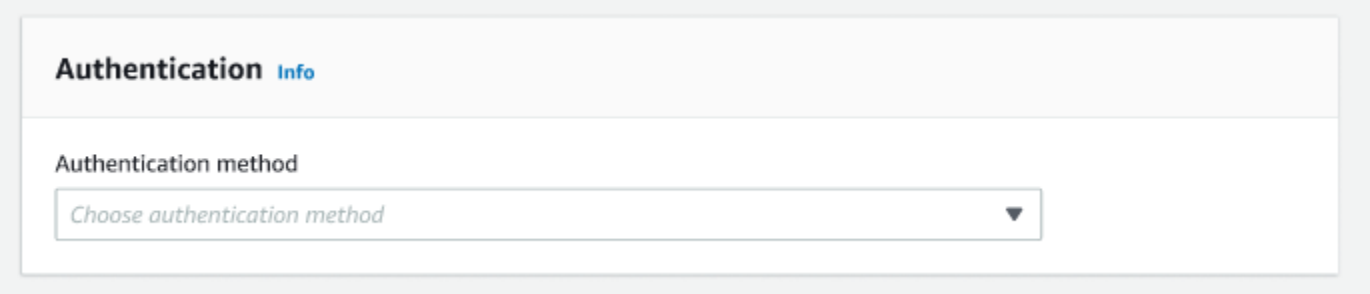
   > **Note**
   >
   > Amazon Managed Streaming for Apache Kafka は TLSおよび SASL/SCRAMSHA-512 認証方法のみをサポートしています。



2. Kafka ブートストラップサーバーの URLsを入力します。各サーバーをカンマで区切ることで、複数のサーバーを入力できます。URL を追加して、 の末尾にポート番号を含めます:<port number>。

   例: b-1.vpc-test-2.034a88o.kafka-us-east-1.amazonaws.com:9094

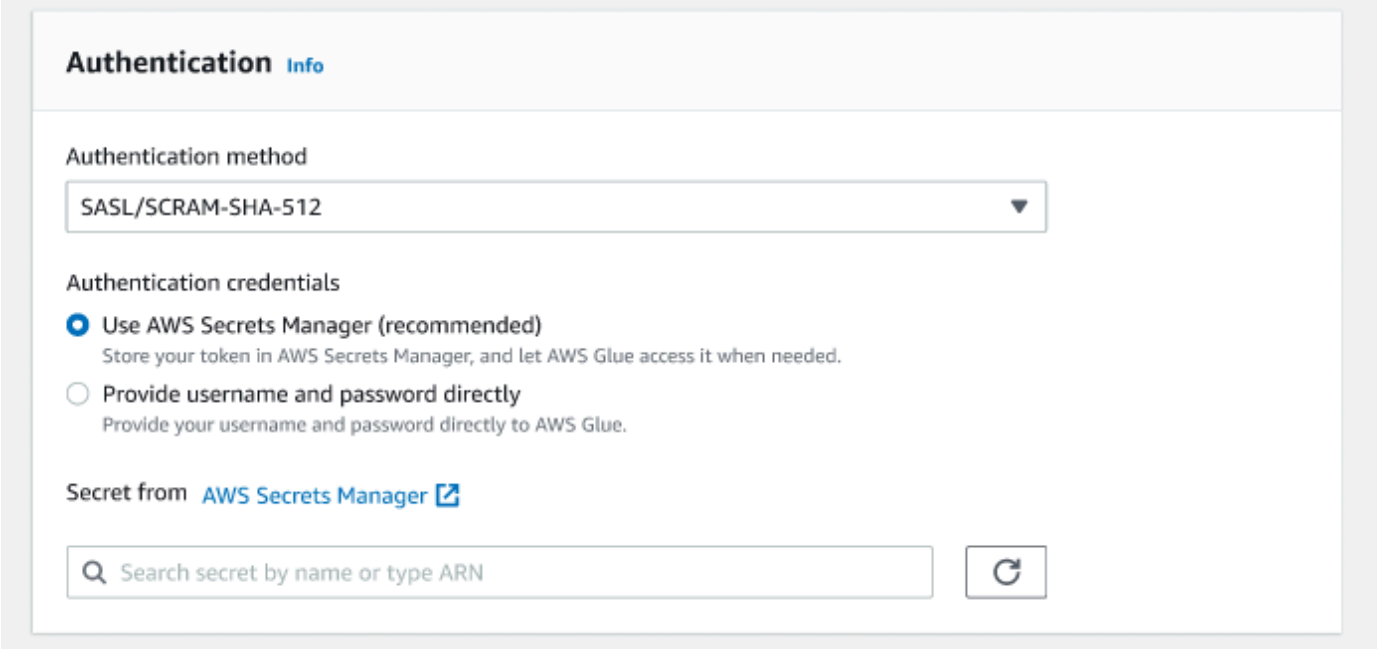
認証方法の選択



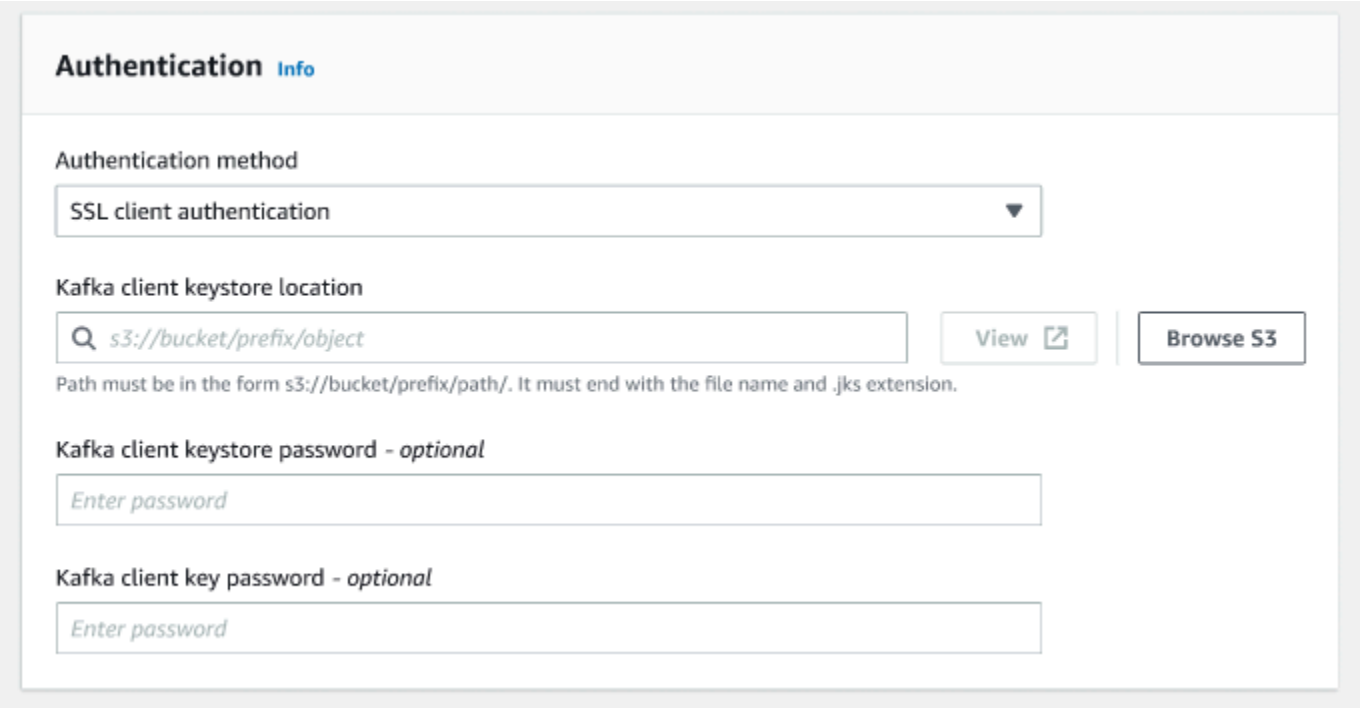
AWS Glue は、認証用の Simple Authentication and Security Layer (SASL) フレームワークをサポートしています。このSASLフレームワークは、さまざまな認証メカニズムをサポートし、 SCRAM (ユーザー名とパスワード) GSSAPI、 (Kerberos プロトコル)、 PLAIN (ユーザー名とパスワード) プロトコル AWS Glue を提供します。

ドロップダウンメニューから認証方法を選択するときには、次のクライアント認証方法を選択できます。

- None - 認証なし。これは、テスト目的で接続する場合に便利です。
- SASL/SCRAM-SHA-512 - 認証認証情報を指定するには、この認証方法を選択します。2 つのオプションがあります。
  - AWS Secrets Manager を使用する (推奨) - このオプションを選択すると、認証情報を AWS Secrets Manager に保存し、必要に応じて情報 AWS Glue にアクセスできます。SSL または SASL認証認証情報を保存するシークレットを指定します。



  - ユーザー名とパスワードを直接指定します。
- SASL/GSSAPI (Kerberos) - if you select this option, you can select the location of the keytab file, krb5.conf file and enter the Kerberos principal name and Kerberos service name. The locations for the keytab file and krb5.conf file must be in an Amazon S3 location. Since MSK does not yet support SASL/GSSAPI。このオプションを使用できるのは、カスタマー管理の Apache Kafka クラスターのみです。詳細については、MITKerberos ドキュメント: Keytab を参照してください。
- SASL/PLAIN - 認証認証情報を指定するには、この認証方法を選択します。2 つのオプションがあります。
  - AWS Secrets Manager を使用する (推奨) - このオプションを選択すると、認証情報を AWS Secrets Manager に保存し、必要に応じて情報 AWS Glue にアクセスできます。SSL または SASL認証認証情報を保存するシークレットを指定します。
  - ユーザー名とパスワードを直接指定します。
- SSL クライアント認証 - このオプションを選択すると、Amazon S3 を参照して Kafka クライアントキーストアの場所を選択できます。オプションで、Kafka クライアントキーストアのパスワードと Kafka クライアントキーのパスワードを入力できます。



暗号化設定の構成

1. Kafka 接続でSSL接続が必要な場合は、SSL接続を要求する のチェックボックスを選択します。経由で接続できない場合、接続は失敗することに注意してくださいSSL。SSL for 暗号化は、任意の認証方法 (SASL/SCRAM-SHA-512, SASL/GSSAPI, SASL/PLAIN、またはSSLクライアント認証) で使用でき、オプションです。

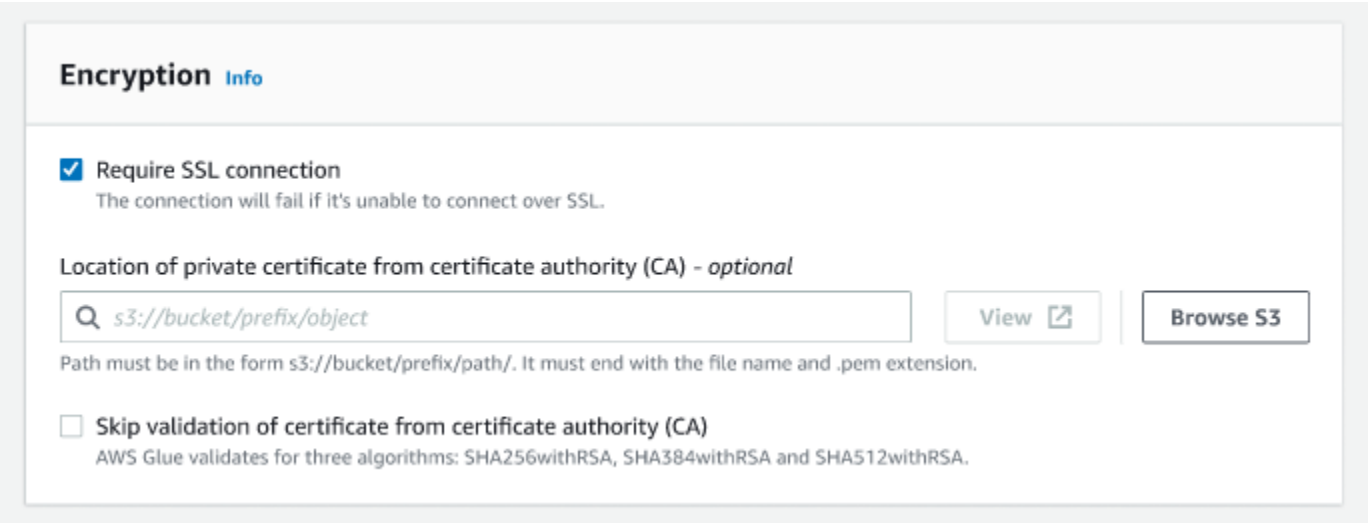
   認証方法がSSLクライアント認証 に設定されている場合、このオプションは自動的に選択され、変更を防ぐために無効になります。

2. （オプション）。認証機関 (CA) からプライベート証明書の場所を選択します。証明書の場所は S3 が存在する場所の中である必要があることに留意してください。[Browse] (参照) を選択して、接続された S3 バケットからファイルを選択します。パスは `s3://bucket/prefix/filename.pem` の形式で指定する必要があります。ファイル名と .pem 拡張子で終わる必要があります。

3. 認証機関 (CA) からの証明書の検証はスキップできます。[Skip validation of certificate from certificate authority (CA)] (認証機関 (CA) からの証明書の検証をスキップする) チェックボックスを選択してください。このチェックボックスにチェックは入っていない場合、 AWS Glue は、3 つのアルゴリズムの証明書を検証します。

   - SHA256withRSA
   - SHA384withRSA

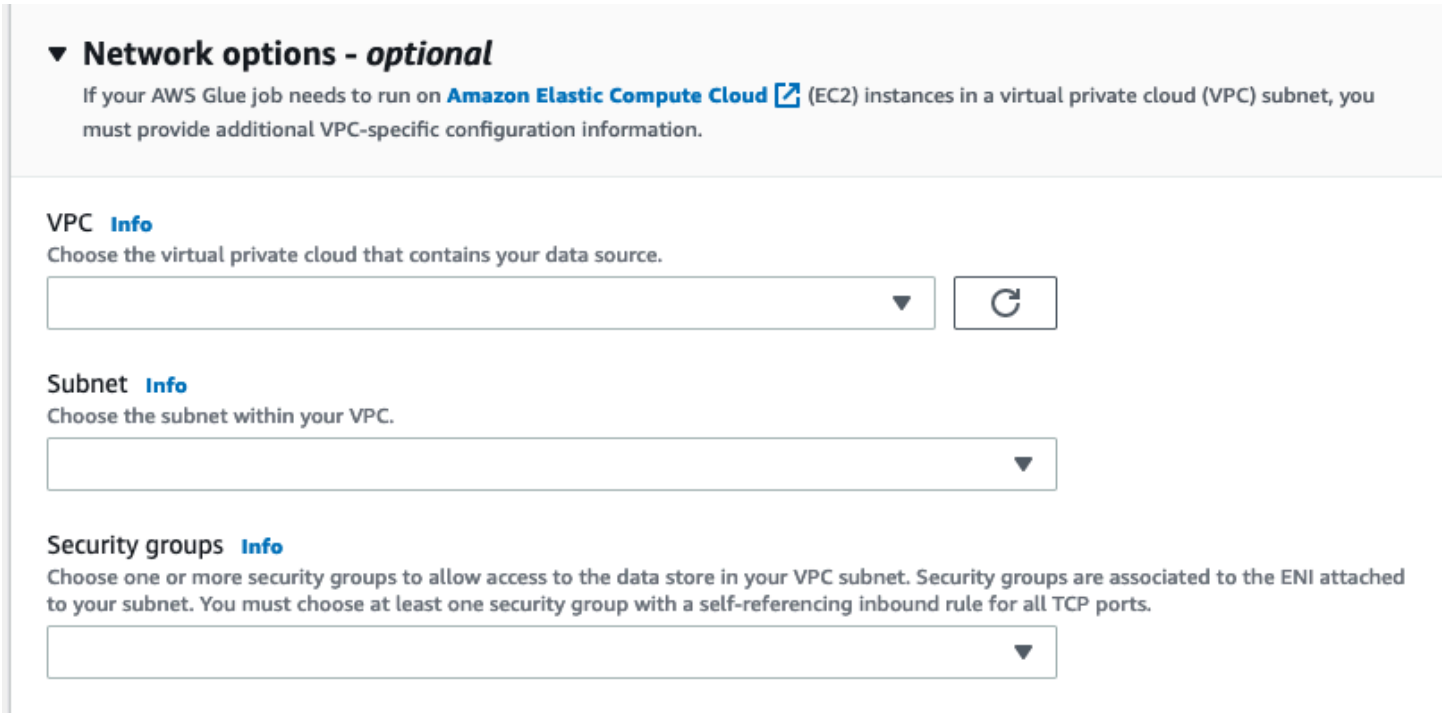
- SHA512withRSA



(オプション) ネットワークオプション

、サブネットVPC、セキュリティグループを設定するオプションの手順は次のとおりです。 AWS Glue ジョブを仮想プライベートクラウド (VPC) サブネットの Amazon EC2インスタンスで実行する必要がある場合は、追加の VPC固有の設定情報を提供する必要があります。

1. データソースを含む VPC (仮想プライベートクラウド) を選択します。
2. でサブネットを選択しますVPC。
3. VPC サブネット内のデータストアへのアクセスを許可するセキュリティグループを 1 つ以上選択します。セキュリティグループは、サブネットにENIアタッチされた に関連付けられています。すべてのTCPポートに対して自己参照インバウンドルールを持つセキュリティグループを少なくとも 1 つ選択する必要があります。



カスタムコネクタを使用したジョブのオーサリング

AWS Glue Studio では、データソースノードとデーターゲットノードの両方に対し、コネクタと接続を使用できます

トピック

- データソースに対しコネクタを使用するジョブを作成する
- コネクタを使用するノードのソースプロパティを設定する
- コネクタを使用するノードのターゲットプロパティの設定

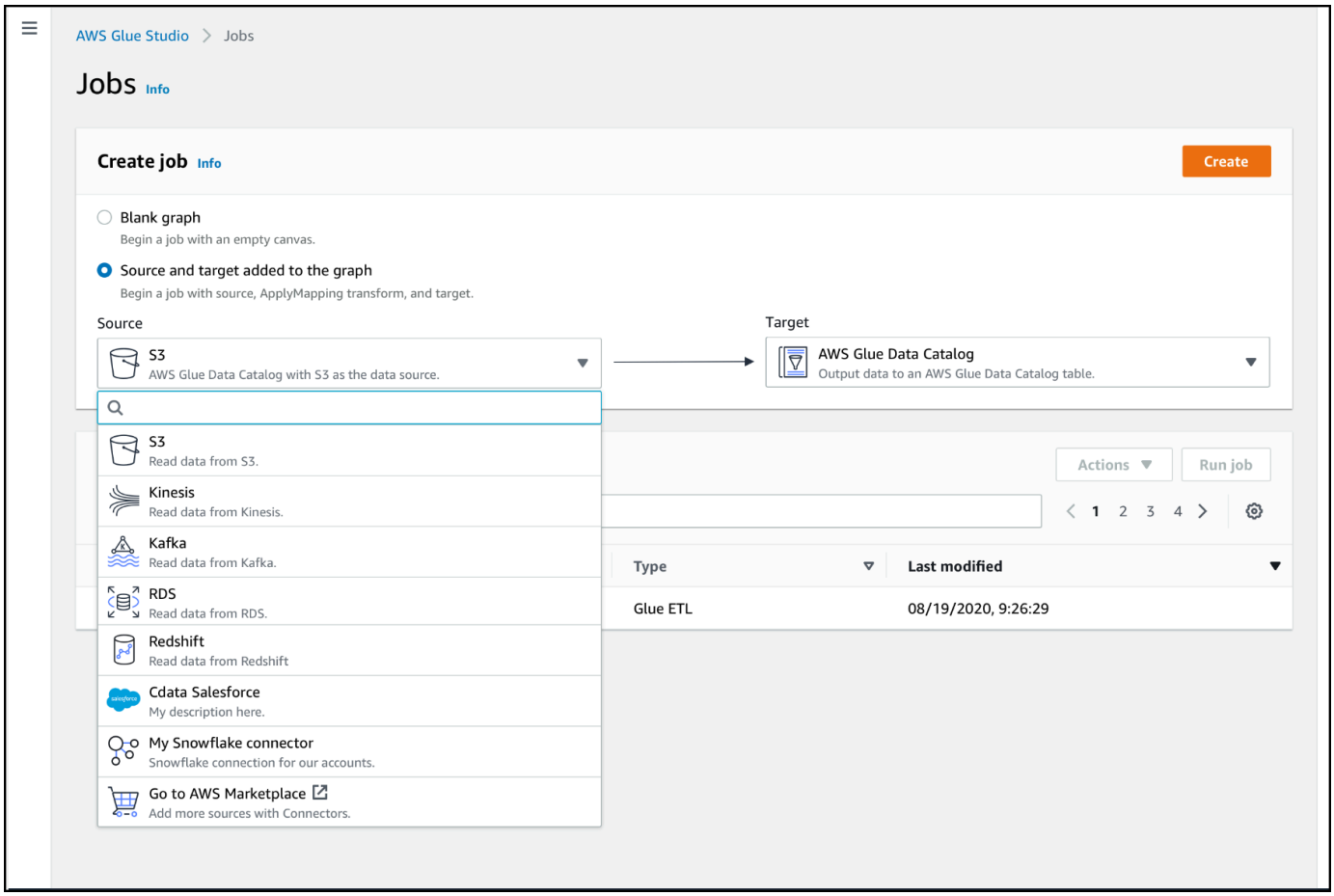
データソースに対しコネクタを使用するジョブを作成する

新しいジョブの作成時に、データソースとデーターゲットで使用するコネクタを選択できます。

データソースまたはデーターゲットに対しコネクタを使用するジョブを作成するには

1. AWS Management Console にサインインして、AWS Glue Studio コンソール (https://console.aws.amazon.com/gluestudio/) を開きます。
2. [Connectors] (コネクタ) ページの [Your connections] (使用する接続)リソースリストで、ジョブで使用する接続を選択してから、[Create job] (ジョブを作成) をクリックします。

または、AWS Glue Studio Glue Studio の [Jobs] (ジョブ) ページの [Create job] (ジョブを作成) で、[Source and target added to the graph] (グラフに追加されたソースとターゲット) を選択します。[Source] (ソース) ドロップダウンリストから、ジョブで使用するカスタムコネクタを選択します。また、[Target] (ターゲット) のためにコネクタを選択することもできます。



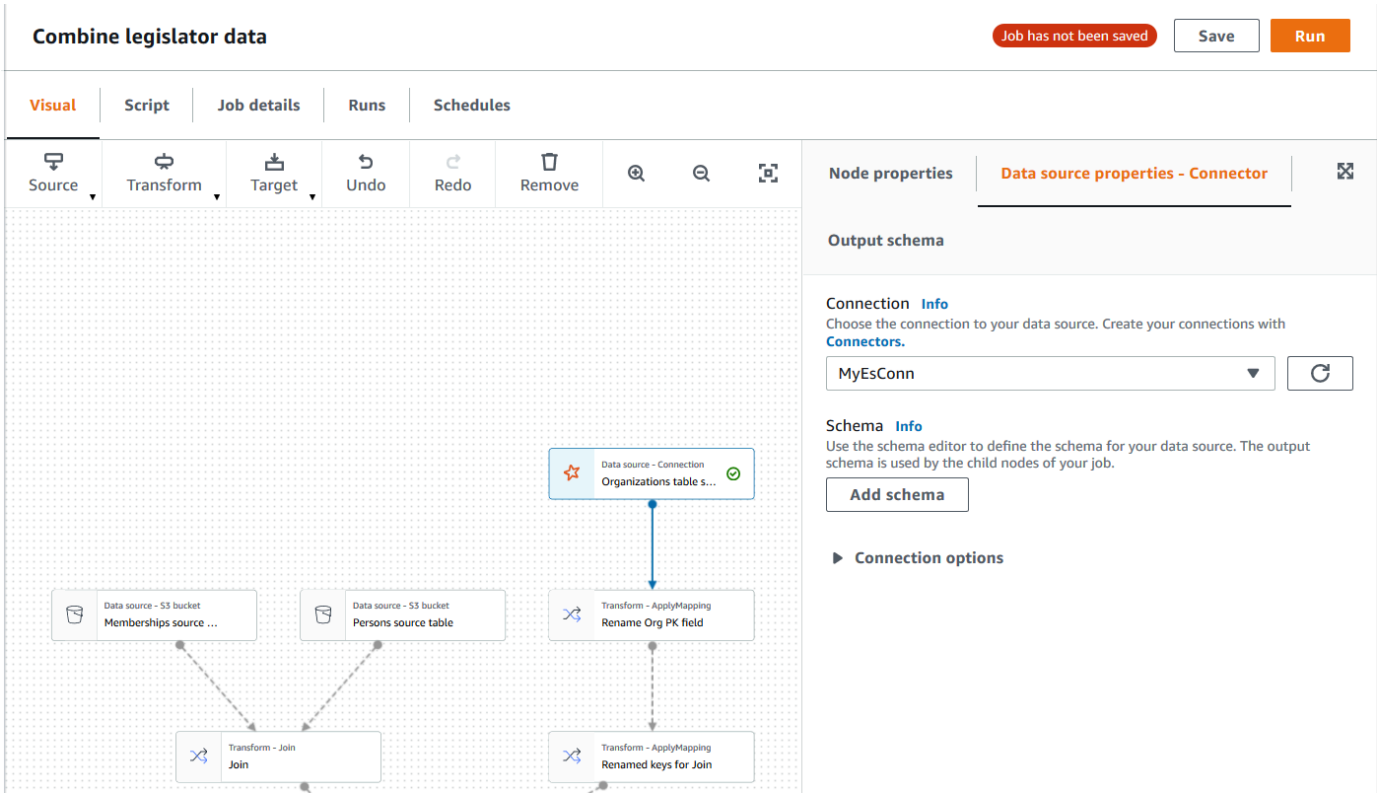
3. [Create] (作成) をクリックして、ビジュアルジョブエディタを開きます。
4. 「コネクタを使用するノードのソースプロパティを設定する」にある説明に従い、データソースノードを設定します。
5. 「AWS Glue Studio でビジュアル ETL ジョブを開始する」を参考に、変換、追加のデータストア、データターゲットを追加しながら、ETL ジョブを作成します。
6. 「ジョブのプロパティを変更する」のように、ジョブのプロパティを設定して、ジョブの実行環境をカスタマイズします。
7. このジョブを保存して、実行します。

コネクタを使用するノードのソースプロパティを設定する

データソース用にコネクタを使用するジョブを作成すると、ビジュアルジョブエディタには、コネクタ用に設定されたデータソースのノードを含むジョブグラフが表示されます。このノードには、データソースのプロパティを設定する必要があります。

コネクタを使用するデータソースノードのプロパティを設定するには

1. ジョブグラフでコネクタのデータソースノードを選択するか、新しいノードを追加して [Node type] (ノードタイプ) でコネクタを選択します。次に、右側のノードの詳細パネルで、[Data source properties] (データソースのプロパティ) タブを選択します (まだ選択されていない場合)。



2. [Data source properties] (データソースのプロパティ) タブで、このジョブに使用する接続を選択します。

   各接続タイプに必要な追加情報を入力します。

JDBC

- [Data source input type] (データソースの入力タイプ): テーブル名または SQL クエリのどちらを、データソースとして指定するか選択します。選択した選択に応じて、以下の追加情報を設定する必要があります。
  - [Table name] (テーブル名): データソース内のテーブルの名前。データソースで、テーブルという用語を使用しない場合、カスタムコネクタの使用状況情報で示されている、適切なデータ構造の名前を指定します (これは、AWS Marketplace で見つかります)。
  - [Filter predicate] (フィルター述語): データソースを読み取る際に使用する条件句。これは、データのサブセットを取得するために使用される、WHERE 句と類似しています。
  - [Query code] (クエリコード): データソースから特定のデータセットを取得するために使用する SQL クエリを入力します。基本 SQL クエリの例は以下のとおりです。

```
SELECT column_list FROM
                          table_name WHERE where_clause
```

- [Schema] (スキーマ): AWS Glue Studio がデータソースにアクセスする際には、Data Catalog テーブルからメタデータ情報を取得する代わりに、接続内に格納された情報を使用します。このため、データソースのスキーマメタデータを指定する必要があります。[Add schema] (スキーマを追加) をクリックして、スキーマエディタを開きます。

  スキーマエディタの使用方法については、「カスタム変換ノードでスキーマを編集する」を参照してください。

- [Partition column] (パーティション列): (オプション) データの読み取りをパーティション化するには、[Partition column] (パーティション列)、[Lower bound] (下限)、[Upper bound] (上限)、および [Number of partitions] (パーティションの数) を、それぞれ指定します。

  lowerBound および upperBound 値は、パーティションのストライドを決定するために使用されます (テーブル内の行のフィルタリングには使用しません)。返されるテーブル内のすべての行は、パーティション化されています。

  > **Note**
  >
  > 列のパーティショニングは、データの読み取りに使用されるクエリに対し、さらにパーティショニング条件を追加します。テーブル名の代わりにクエリを使用する場

合は、指定されたパーティショニング条件でクエリが動作することを確認する必要があります。例:

- "SELECT col1 FROM table1" の形式のクエリでパーティション列を使用する場合、末尾に WHERE 句を追加してそのクエリをテストします。
- クエリを "SELECT col1 FROM table1 WHERE col2=val" 形式で行っている場合、AND とパーティション列を使用する式を使用して、WHERE 句を拡張することでそのクエリをテストします。

- データ型のキャスティング: データソースが JDBC で使用できないデータ型を使用している場合、このセクションを使用して、データソースのデータ型を JDBC データ型に変換する方法を指定します。データ型の変換方法には、最大 50 種類までを指定できます。データソース内で同じデータ型を使用しているすべての列は、同じ方法で変換されます。

  例えば、データソース内に Float データ型を使用する 3 つの列があり、Float データ型に対し JDBC の String データ型への変換を指定している場合には、Float データ型を使用する 3 つの列がすべて String データ型に変換されます。

- ジョブのブックマークキー: ジョブのブックマークは、AWS Glue が状態情報を保持することと、古いデータの再処理を防ぐことを助けます。ブックマークキーとして 1 つ以上の列を指定します。AWS Glue Studio はブックマークキーを使用して ETL ジョブの以前の実行中にすでに処理されたデータを追跡します。カスタムブックマークキーに使用する列は、厳密に単調に増加または減少する必要がありますが、そこにギャップを含むことは可能です。

  ブックマークキーを複数入力した場合は、それらは結合され単一の複合キーを形成します。複合ジョブブックマークキーには、重複する列を含めることはできません。ブックマークキーを指定しない場合、AWS Glue Studio はデフォルトでプライマリキーをブックマークキーとして使用します。ただし、そのプライマリキーが連続して (ギャップなく) 増減することが条件です。ジョブブックマークのプロパティでは有効になっているものの、テーブルにプライマリキーがない場合には、カスタムジョブブックマークキーを指定する必要があります。この指定を行わないと、デフォルトとして使用するプライマリキーの検索が失敗し、ジョブの実行も失敗します。

- [Job bookmark keys sorting order] (Job ブックマークキーの並べ替え順序): キー値を、連続的に増加させるか減少させるかを選択します。

Spark

- [Schema] (スキーマ): AWS Glue Studio がデータソースにアクセスする際には、Data Catalog テーブルからメタデータ情報を取得する代わりに、接続内に格納された情報を使用します。このため、データソースのスキーマメタデータを指定する必要があります。[Add schema] (スキーマを追加) をクリックして、スキーマエディタを開きます。

  スキーマエディタの使用方法については、「カスタム変換ノードでスキーマを編集する」を参照してください。

- [Connection options:] (接続オプション): 追加の接続情報やオプションを提供するために、必要に応じて追加のキーと値のペアを入力します。例えば、データベース名、テーブル名、ユーザー名、パスワードを入力します。

  例えば OpenSearch の場合には、the section called “ チュートリアル: Elasticsearch 向けの AWS Glue コネクタを使用する ” で説明されているように以下のキーと値のペアを入力します

  - `es.net.http.auth.user` : *`username`*
  - `es.net.http.auth.pass` : *`password`*
  - `es.nodes` : `https://`*`<Elasticsearch endpoint>`*
  - `es.port` : `443`
  - `path`: *`<Elasticsearch resource>`*
  - `es.nodes.wan.only` : `true`

最小接続オプションの使用例については、GitHub のサンプルテストスクリプト、MinimalSparkConnectorTest.scala を参照してください。ここでは、接続のために指定する一般的な接続オプションが確認できます。

Athena

- [Table name] (テーブル名): データソース内のテーブルの名前。Athena-CloudWatch のログからの読み取り用にコネクタを使用している場合は、テーブル名に `all_log_streams` を入力します。
- [Athena schema name:] (Athena のスキーマ名): テーブルを含むデータベースに対応する、Athena データソース内のスキーマを選択します。Athena-CloudWatch のログからの

読み取りのためにコネクタを使用している場合は、スキーマ名として `/aws/glue/name` のように入力します。

- [Schema] (スキーマ): AWS Glue Studio がデータソースにアクセスする際には、Data Catalog テーブルからメタデータ情報を取得する代わりに、接続内に格納された情報を使用します。このため、データソースのスキーマメタデータを指定する必要があります。[Add schema] (スキーマを追加) をクリックして、スキーマエディタを開きます。

  スキーマエディタの使用方法については、「カスタム変換ノードでスキーマを編集する」を参照してください。

- [Additional connection options] (その他の接続オプション): 他の接続情報やオプションを提供する際には、必要に応じて追加のキーと値のペアを入力します。

この例については、https://github.com/aws-samples/aws-glue-samples/tree/master/GlueCustomConnectors/development/Athena にある `README.md` ファイルでご確認ください。このドキュメント内に示す手順では、サンプルコードにより、必要最小限の接続オプション (`tableName`、`schemaName`、および `className`) を使用しています。コード例では、これらのオプションを `optionsMap` 変数の一部として指定しており、それらを実際に指定することで接続を使用できるようになります。

3. (オプション) 必要な情報を指定した後は、[Output schema] (出力スキーマ) タブを選択することで、出力されたデータソース用のデータスキーマを、ノードの詳細パネルに表示できるようになります。このタブに表示されるスキーマは、ジョブグラフに追加される任意の子ノードによって使用されます。
4. (オプション) ノードおよびデータソースのプロパティを設定した後、ノードの詳細パネルの [Data preview] (データのプレビュー) タブを選択して、データソースからのデータセットをプレビューできます。ジョブ内の任意のノードに対してこのタブを初めて選択すると、データにアクセスする IAM ロールを指定するよう求められます。この機能を使用するには費用がかかり、IAM ロールを指定するとすぐに請求が開始します。

コネクタを使用するノードのターゲットプロパティの設定

データターゲットタイプにコネクタを使用する場合は、データターゲットノードのプロパティを設定する必要があります。

コネクタを使用するデータターゲットノードでプロパティを設定するには

1. ジョブグラフで、コネクタのデータターゲットノードを選択します。次に、右側のノードの詳細パネルで、[Data target properties] (データターゲットのプロパティ) タブを選択します (選択されていない場合)。
2. [Data target properties] (データターゲットのプロパティ) タブで、ターゲットへの書き込みに使用する接続を選択します。

   各接続タイプに必要な追加情報を入力します。

   JDBC

   - [Connection] (接続): コネクタで使用する接続を選択します。接続の作成方法については、「コネクタ用の接続を作成する」を参照してください。
   - [Table name] (テーブル名): データターゲット内のテーブルの名前。データターゲットで、テーブルという用語を使用しない場合、カスタムコネクタの使用状況情報で示されている、適切なデータ構造の名前を指定します (これは、AWS Marketplace で見つかります)。
   - [Batch size] (バッチサイズ) (オプション): 1 回のオペレーションでターゲットテーブルに挿入する、行数またはレコード数を入力します。デフォルト値は 1000 行です。

   Spark

   - [Connection] (接続): コネクタで使用する接続を選択します。以前に接続を作成していない場合は、[Create connection] (接続を作成する) をクリックして作成します。接続の作成方法については、「コネクタ用の接続を作成する」を参照してください。
   - [Connection options:] (接続オプション): 追加の接続情報やオプションを提供するために、必要に応じて追加のキーと値のペアを入力します。データベース名、テーブル名、ユーザー名、パスワードを入力することもできます。

     例えば OpenSearch の場合には、the section called “ チュートリアル: Elasticsearch 向けの AWS Glue コネクタを使用する ” で説明されているように以下のキーと値のペアを入力します

     - `es.net.http.auth.user` : *`username`*
     - `es.net.http.auth.pass` : *`password`*
     - `es.nodes` : `https://`*`<Elasticsearch endpoint>`*
     - `es.port` : `443`

- path: *<Elasticsearch resource>*
- es.nodes.wan.only : true

最小接続オプションの使用例については、GitHub のサンプルテストスクリプト、MinimalSparkConnectorTest.scala を参照してください。ここでは、接続のために指定する一般的な接続オプションが確認できます。

3. 必要な情報を指定した後は、[Output schema] (出力スキーマ) タブを選択することで、出力されたデータソース用のデータスキーマを、ノードの詳細パネルに表示できるようになります。

コネクタと接続を管理する

コネクタと接続を管理するには、AWS Glue の [コネクタ] ページを使用します。

トピック

- コネクタと接続の詳細を表示する
- コネクタと接続を編集する
- コネクタおよび接続を削除する
- コネクタのサブスクリプションをキャンセルする

コネクタと接続の詳細を表示する

[Connectors] (コネクタ) ページの [Your connectors] (使用中のコネクタ)、および [Your connections] (使用中の接続) リソーステーブルで、コネクタと接続に関する概要情報を表示できます。詳細な情報を表示するには、以下の手順を実行します。

コネクタまたは接続の詳細を表示するには

1. AWS Glue Studio コンソールのナビゲーションペインで、[Connectors] (コネクタ) をクリックします。
2. 詳細情報を表示するコネクタまたは接続を選択します。
3. [Actions] (アクション)、[View details] (詳細を表示) の順にクリックして、選択したコネクタまたは接続の詳細ページを開きます。
4. 詳細ページでは、コネクタまたは接続に対し、[Edit] (編集) または [Delete] (削除) を選択できます。

- コネクタの場合は、[Create connection] (接続を作成する) をクリックすることで、そのコネクタを使用する新しい接続を作成できます。
- 接続の場合は、[Create job] (ジョブを作成する) をクリックすると、その接続を使用するジョブを作成できます。

コネクタと接続を編集する

[Connectors] (コネクタ) ページを使用して、コネクタと接続に保存されている情報を変更します。

コネクタまたは接続を変更するには

1. AWS Glue Studio コンソールのナビゲーションペインで、[Connectors] (コネクタ) をクリックします。
2. 変更するコネクタまたは接続を選択します。
3. [Actions] (アクション) を選択して、[Edit] (編集) を選択します。

   また、[View details] (詳細を表示する) をクリックし、コネクタまたは接続の詳細ページを開いて、[Edit] (編集) を実行することもできます。
4. [Edit connector] (コネクタを編集する) または [Edit connection] (接続を編集する) ページで、情報を更新した上で、[Save] (保存) をクリックします。

コネクタおよび接続を削除する

コネクタと接続を削除するには、[Connectors] (コネクタ) ページを使用します。コネクタを削除すると、そのコネクタのために作成された接続もすべて削除されます。

AWS Glue Studio からコネクタを削除する方法

1. AWS Glue Studio コンソールのナビゲーションペインで、[Connectors] (コネクタ) をクリックします。
2. 削除するコネクタまたは接続を選択します。
3. [Actions] (アクション) を選択してから、[Delete] (削除) をクリックします。

   または、[View details] (詳細を表示する) をクリックし、コネクタまたは接続の詳細ページで、[Delete] (削除) を実行することもできます。

4. 「**Delete**」と入力し、コネクタまたは接続を削除することを確認した上で、[Delete] (削除) をクリックします。

   コネクタを削除すると、そのコネクタのために作成された接続もすべて削除されます。

ここで削除される接続を使用していたジョブも、すべて機能しなくなります。これらのジョブは、別のデータストアを使用するように編集することも、あるいは削除することもできます。ジョブを削除する方法については、「ジョブの削除」を参照してください。

コネクタを削除しても、そのコネクタの AWS Marketplace に対するサブスクリプションはキャンセルされません。削除されたコネクタのサブスクリプションを削除するには、「コネクタのサブスクリプションをキャンセルする」の手順に従います。

コネクタのサブスクリプションをキャンセルする

AWS Glue Studio から接続とコネクタをキャンセルした後、不要になったコネクタのサブスクリプションを AWS Marketplace でキャンセルできます。

> **Note**
>
> コネクタのサブスクリプションをキャンセルしても、コネクタや接続はアカウントから削除されません。コネクタと、それに関連する接続を使用するジョブは、以後コネクタを使用できなくなり失敗します。
> AWS Marketplace から、コネクタのサブスクリプション解除または再サブスクライブを行う際には、その前に、対象の AWS Marketplace 製品に関連付けられた既存の接続とコネクタを削除する必要があります。

AWS Marketplace でコネクタのサブスクリプションを解除するには

1. AWS Marketplace コンソール (https://console.aws.amazon.com/marketplace) にサインインします。
2. [Manage subscriptions] (サブスクリプションを管理する) をクリックします。
3. [Manage subscriptions] (サブスクリプションの管理) ページで、キャンセルするコネクタのサブスクリプションの横にある [Manage] (管理) をクリックします。
4. [Actions] (アクション)、[Delete Application] (アプリケーションの削除) の順にクリックします。

5. 実行中のインスタンスがアカウントで課金されることを受け入れるチェックボックスをオンにして、[Yes, cancel subscription] (はい、サブスクリプションをキャンセルします) をクリックします。

カスタムコネクタの開発

コードを記述すると、データストアとの間でデータの読み取りや書き込みを実行したり、AWS Glue Studio ジョブで使用するためにデータのフォーマットを行ったりできます。Spark、Athena、JDBC データストアのためのコネクタを作成できます。実装する必要がある基本的なインターフェイスの概要は、GitHub に掲載されているサンプルコードで提供しています。

コネクタコードを作成するには、ローカルの開発環境が必要です。コネクタを書き込むには、任意の IDE やコマンドラインエディタを使用することもできます。開発環境の例には以下があります。

- ローカルの AWS Glue ETL Mavenライブラリを使用したローカル Scala 環境 (AWS Glueデベロッパーガイドの「Developing Locally with Scala」を参照)。
- https://www.jetbrains.com/idea/ からダウンロードして使用する、IntelliJ IDE。

トピック

- Spark コネクタの開発
- Athena コネクタの開発
- JDBC コネクタの開発
- AWS Glue Studio でのカスタムコネクタ使用例
- AWS Marketplace 向け AWS Glue コネクターの開発

Spark コネクタの開発

Spark DataSource API V2 (Spark 2.4) を使用して、データ読み取りのために Spark コネクタを作成することができます。

カスタム Spark コネクタを作成するには

Spark コネクタ開発のための AWS Glue GitHub サンプルライブラリ (https://github.com/aws-samples/aws-glue-samples/tree/master/GlueCustomConnectors/development/Spark/README.md) にある手順に従います。

Athena コネクタの開発

Athena コネクタを作成し、AWS Glue および AWS Glue Studio でカスタムのデータソースをクエリするために使用できます。

カスタムの Athena コネクタを作成するには

Athena コネクタ開発のための AWS Glue GitHub サンプルライブラリ(https://github.com/aws-samples/aws-glue-samples/tree/master/GlueCustomConnectors/development/Athena) にある手順に従います。

JDBC コネクタの開発

データストアにアクセスするために、JDBC を使用するコネクタを作成できます。

カスタムの JDBC コネクタを作成するには

1. ローカルの開発環境に、AWS Glue Spark ランタイムライブラリをインストールします。手順については、AWS Glue GitHub のサンプルライブラリ (https://github.com/aws-samples/aws-glue-samples/tree/master/GlueCustomConnectors/development/GlueSparkRuntime/README.md) を参照してください。
2. データソースからデータを取得するための JDBC ドライバーを実装します。Java SE 8 向けの Java ドキュメントを参照してください。

   作成しているコード内に、AWS Glue Studio がコネクタの位置を特定するために使用するエントリポイントを記述します。[Class name] (クラス名) フィールドは、JDBC ドライバーへの完全なパスを指定する必要があります。
3. `GlueContext` API を使用して、コネクタによりデータを読み取ります。必要に応じてユーザーは、AWS Glue Studio コンソールを使用して、データソースへの接続を設定するための他の入力オプションを追加できます。カスタム JDBC コネクタを使用しながら、JDBC データベースとの間で読み取りおよび書き込みを行うコードの例については、「Custom and AWS Marketplace connectionType values」を参照してください。

AWS Glue Studio でのカスタムコネクタ使用例

カスタムコネクタの使用例については、次のブログを参照してください。

- AWS Glue を使用したデータストア用のカスタムコネクタの開発、テスト、およびデプロイ

- Apache Hudi: Writing to Apache Hudi tables using AWS Glue Custom Connector
- Google BigQuery: Migrating data from Google BigQuery to Amazon S3 using AWS Glue custom connectors
- Snowflake (JDBC): Performing data transformations using Snowflake and AWS Glue
- SingleStore: Building fast ETL using SingleStore and AWS Glue
- Salesforce: Ingest Salesforce data into Amazon S3 using the CData JDBC custom connector with AWS Glue
- MongoDB: Building AWS Glue Spark ETL jobs using Amazon DocumentDB (with MongoDB compatibility) and MongoDB
- Amazon Relational Database Service (Amazon RDS): Building AWS Glue Spark ETL jobs by bringing your own JDBC drivers for Amazon RDS
- MySQL (JDBC): https://github.com/aws-samples/aws-glue-samples/blob/master/GlueCustomConnectors/development/Spark/SparkConnectorMySQL.scala

AWS Marketplace 向け AWS Glue コネクターの開発

AWS パートナーであれば、作成したカスタムコネクタを AWS Marketplace にアップロードし、それを AWS Glue のお客様に販売することができます

コネクタコードの開発プロセスは、カスタムコネクタの場合と同様です。ただし、コネクタコードのアップロードと検証のプロセスには、より詳細な要素が含まれます。手順については、GitHub ウェブサイトの「 Creating Connectors for AWS Marketplace」を参照してください。

AWS Glue Studio でのコネクタおよび接続の使用に関する制約事項

カスタムコネクタ、または AWS Marketplace で提供されているコネクタを使用する際は、以下の制限事項に注意してください。

- カスタムコネクタ用に作成された接続では、TestConnection API は使用できません。
- Data Catalog 接続でのパスワードの暗号化は、カスタムコネクタではサポートされていません。
- JDBC コネクタを使用するデータソースノードでフィルター述語を指定した場合は、ジョブブックマークを使用できなくなります。
- Marketplace 接続の作成は、AWS Glue Studio ユーザーインターフェイス以外ではサポートされていません。

AWS Glue 接続のテスト

ベストプラクティスとして、ETL ジョブで AWS Glue 接続を使用する前に、AWS Glue コンソールを使用して接続をテストします。AWS Glue は、接続内のパラメータを使用してデータストアにアクセスできることを確認し、エラーを報告します。AWS Glue 接続の詳細については、「データへの接続」を参照してください。

AWS Glue 接続をテストするには

1. AWS Management Consoleにサインインし、AWS Glue コンソール (https://console.aws.amazon.com/glue/) を開きます。
2. ナビゲーションペインの [データカタログ] で [接続] を選択します。ナビゲーションウィンドウの [データ カタログ] 上で、[データ接続] を選択することもできます。
3. [接続] で、目的の接続の横にあるチェックボックスをオンにし、[アクション] を選択します。ドロップダウンメニューで、[接続のテスト] を選択します。
4. [Test connection] (テスト接続) ダイアログボックスで、ロールを選択するか [Create IAM role] (IAM ロールの作成) を選択して AWS Identity and Access Management (IAM) コンソールに移動し、新しいロールを作成します。ロールには、データストアに対する権限が必要です。
5. [確認] を選択します。

   テストが開始されます。完了までに数分かかることがあります。テストが失敗した場合、[Troubleshoot] をクリックし、問題解決のステップを表示します。
6. CloudWatch でログを表示するには、[ログ] を選択します。ログを表示するには、必要な IAM アクセス許可が付与されている必要があります。詳細については、Amazon CloudWatch Logs ユーザーガイドの「CloudWatch Logs の AWS マネージド (事前定義) ポリシー」を参照してください。

を経由するように AWS 通話を設定する VPC

特別なジョブパラメータ`disable-proxy-v2`を使用すると CloudWatch、 AWS Glue 経由で Amazon S3 などの サービスに通話をルーティングできますVPC。デフォルトでは、 はローカルプロキシ AWS Glue を使用して 経由で AWS Glue VPCトラフィックを送信し、Amazon S3 からスクリプトとライブラリをダウンロードし、ログとメトリクスの CloudWatch 公開リクエストを に送信し、データカタログ AWS Glue へのアクセスリクエストを に送信します。このプロキシは、 VPCが Amazon S3 などの他の AWS サービスへの適切なルートを設定していない場合でも、ジョブが正常に機能することを可能にし CloudWatch、 AWS Glue. AWS Glue now はこの動作をオフにするため

のパラメータを提供します。詳細については、「 で使用されるジョブパラメータ AWS Glue」を参照してください。 AWS Glue は、 AWS Glue ジョブの CloudWatch ログの公開に引き続きローカルプロキシを使用します。

> **Note**
>
> - この機能は、 AWS Glue バージョン 2.0 以降の AWS Glue ジョブでサポートされています。この機能を使用する場合は、 VPC が NATまたは サービスVPCエンドポイントを介して Amazon S3 へのルートを設定している必要があります。
> - 非推奨のジョブパラメータは、 経由でスクリプトとライブラリをダウンロードするために Amazon S3 への呼び出しdisable-proxyのみをルーティングしますVPC。代わりに、新しいパラメータ disable-proxy-v2 を使用することをお勧めします。

使用例

を使用して AWS Glue ジョブを作成しますdisable-proxy-v2。

```
aws glue create-job \
    --name no-proxy-job \
    --role GlueDefaultRole \
    --command "Name=glueetl,ScriptLocation=s3://my-bucket/glue-script.py" \
    --connections Connections="traffic-monitored-connection" \
    --default-arguments '{"--disable-proxy-v2" : "true"}'
```

のデータJDBCストアへの接続 VPC

通常、Amazon Virtual Private Cloud (Amazon VPC) 内にリソースを作成して、パブリックインターネット経由でアクセスできないようにします。デフォルトでは、 AWS Glue は 内のリソースにアクセスできませんVPC。 AWS Glue が 内のリソースにアクセスできるようにするにはVPC、VPCサブネットIDsとセキュリティグループ を含む追加の VPC固有の設定情報を指定する必要がありますIDs。 は、この情報 AWS Glue を使用して、関数がプライベート 内の他のリソースに安全に接続できるようにするElastic Network Interface を設定しますVPC。

VPC エンドポイントを使用する場合は、ルートテーブルに追加します。詳細については、「 およびのインターフェイスVPCエンドポイントの作成 AWS Glue」を参照してください前提条件。

Data Catalog で暗号化を使用する場合は、KMSインターフェイスエンドポイントを作成し、ルートテーブルに追加します。詳細については、「」を参照してください。

のVPCエンドポイントを作成する AWS KMS。

Elastic Network Interface を使用したVPCデータへのアクセス

AWS Glue が 内のJDBCデータストアに接続するとVPC、 はVPC、データにアクセスするためのElastic Network Interface (プレフィックス を含むGlue_) をアカウントに AWS Glue 作成します。このネットワークインターフェイスは、 にアタッチされている限り削除できません AWS Glue。Elastic Network Interface の作成の一環として、 は 1 つ以上のセキュリティグループを AWS Glue 関連付けます。 AWS Glue がネットワークインターフェイスを作成できるようにするには、リソースに関連付けられているセキュリティグループが、ソースルールによるインバウンドアクセスを許可する必要があります。このルールには、リソースに関連付けられたセキュリティグループが含まれています。これにより、Elastic Network Interface は同じセキュリティグループを持つデータストアにアクセスできるようになります。

がコンポーネントと通信 AWS Glue できるようにするには、すべてのTCPポートに自己参照インバウンドルールを持つセキュリティグループを指定します。自己参照ルールを作成することで、ソースを 内の同じセキュリティグループに制限VPCし、すべてのネットワークに開放することはできません。のデフォルトのセキュリティグループには、 の自己参照インバウンドルールが既にあるVPC場合がありますALL Traffic。

Amazon VPCコンソールでルールを作成できます。経由でルール設定を更新するには AWS Management Console、VPCコンソール (https://console.aws.amazon.com/vpc/) に移動し、適切なセキュリティグループを選択します。ALL TCP のインバウンドルールを指定して、同じセキュリティグループ名をソースとして指定します。セキュリティグループルールの詳細については、「 のセキュリティグループVPC」を参照してください。

それぞれの Elastic Network Interface には、指定したサブネット内の IP アドレス範囲からプライベート IP アドレスが割り当てられます。ネットワークインターフェイスにはパブリック IP アドレスが割り当てられていません。インターネットアクセス AWS Glue が必要です (VPCエンドポイントがない AWS サービスにアクセスするにはなど) 。ネットワークアドレス変換 (NAT) インスタンスを内で設定することもVPC、Amazon VPCNATゲートウェイを使用することもできます。詳細については、「Amazon VPCユーザーガイドNAT」の「ゲートウェイ」を参照してください。にアタッチされたインターネットゲートウェイをサブネットルートテーブルのルートVPCとして直接使用することはできません。これは、ネットワークインターフェイスにパブリック IP アドレスが必要であるためです。

VPC ネットワーク属性 enableDnsHostnamesと は true に設定enableDnsSupportする必要があります。詳細については、「 DNSで を使用するVPC」を参照してください。

> **Important**
>
> インターネットアクセスのないパブリックサブネットまたはプライベートサブネットにデータストアを配置しないでください。代わりに、NATインスタンスまたは Amazon VPCNAT ゲートウェイを介してインターネットにアクセスできるプライベートサブネットにのみアタッチします。

Elastic Network Interface プロパティ

Elastic Network Interface を作成するには、次のプロパティを指定する必要があります。

VPC

データストアVPCを含む の名前。

サブネット

データストアVPCを含む のサブネット。

セキュリティグループ

データストアに関連付けられているセキュリティグループ。 は、これらのセキュリティグループをVPCサブネットにアタッチされているElastic Network Interface に AWS Glue 関連付けます。AWS Glue コンポーネントが通信できるようにし、他のネットワークからのアクセスを防ぐには、少なくとも 1 つの選択したセキュリティグループで、すべてのTCPポートに自己参照インバウンドルールを指定する必要があります。

Amazon Redshift VPCでの の管理については、Amazon Virtual Private Cloud でのクラスターの管理 (VPC) 」を参照してください。

Amazon Relational Database Service (Amazon RDS) VPCでの の管理については、「 での Amazon RDS DB インスタンスの操作VPC」を参照してください。

MongoDB または MongoDB Atlas 接続を使用する

MongoDB または MongoDB Atlas の接続を作成すると、ETL ジョブで接続を使用することができます。AWS Glue Data Catalog にテーブルを作成し、テーブルの `connection` 属性に MongoDB または MongoDB Atlas 接続を指定します。

AWS Glue では、接続 `url` と MongoDB 接続の認証情報が保存されます。接続 URI の形式は次のとおりです。

- MongoDB の場合: mongodb://host:port/database ホストは、ホスト名、IP アドレス、UNIX ドメインソケットのいずれかにすることができます。接続文字列でポートが指定されていない場合は、デフォルトの MongoDB ポート、27017 が使用されます。
- MongoDB Atlas: mongodb+srv://server.example.com/database ホストは、DNS SRV レコードに対応するホスト名にすることができます。SRV 形式ではポートは不要であり、デフォルトの MongoDB ポート、27017 が使用されます。

さらに、ジョブスクリプトでオプションを指定できます。詳細については、「the section called “MongoDB 接続”」を参照してください。

VPC エンドポイントを使用した Amazon S3 データストアのクロール

セキュリティ、監査、または制御の目的で、Amazon S3 データストアまたは Amazon S3 でバックアップされたデータカタログテーブルにはAmazon Virtual Private Cloud 環境 (Amazon) を介してのみアクセスするようにすることができますVPC。このトピックでは、接続`Network`タイプを使用してVPCエンドポイント内の Amazon S3 データストアまたは Amazon S3 でバックアップされたデータカタログテーブルへの接続を作成してテストする方法について説明します。

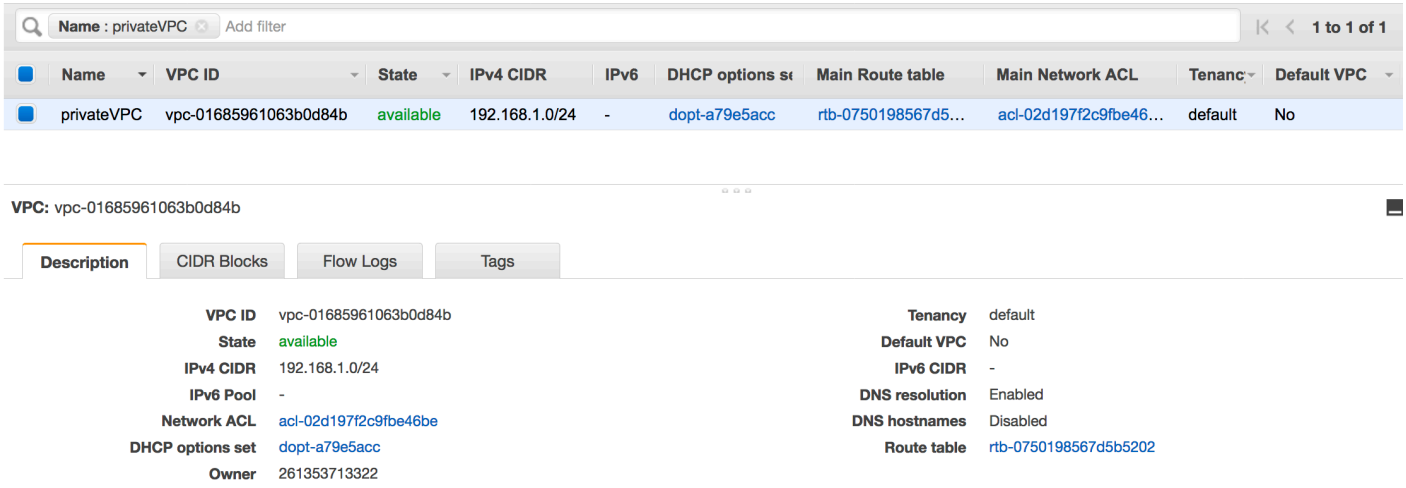
データストアでクローラーを実行するには、次のタスクを実行します。

- the section called “前提条件”
- the section called “Amazon S3 への接続を作成する”
- the section called “Amazon S3 への接続をテストする”
- the section called “Amazon S3 データストアのクローラーを作成する”
- the section called “クローラーの実行”

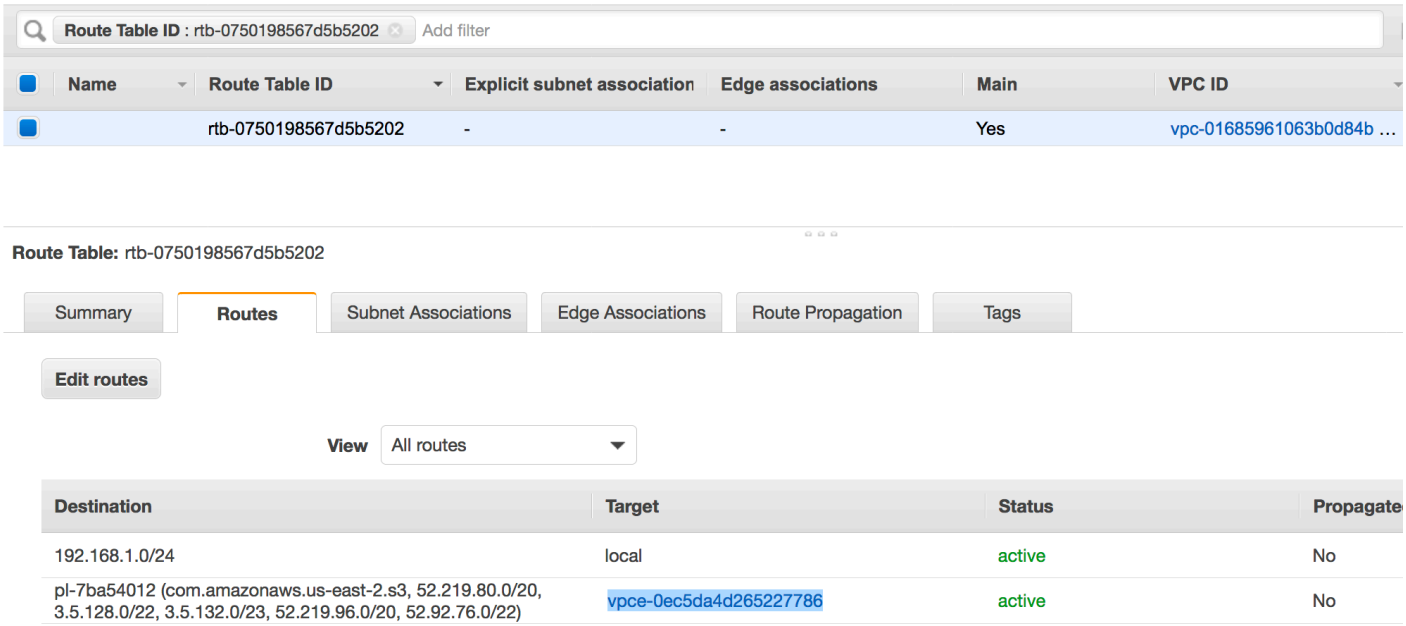
前提条件

Amazon Virtual Private Cloud 環境 (Amazon) を介してアクセスする Amazon S3 データストアまたは Amazon S3 でバックアップされたデータカタログテーブルをセットアップするためのこれらの前提条件を満たしていることを確認しますVPC。

- 設定済みの VPC。例: vpc-01685961063b0d84b。詳細については、「Amazon ユーザーガイド」の「Amazon の使用開始VPC」を参照してください。 VPC
- にアタッチされた Amazon S3 エンドポイントVPC。例: vpc-01685961063b0d84b。詳細については、Amazon S3のエンドポイント」を参照してください。 VPC



- VPC エンドポイントを指すルートエントリ。例えば、VPCエンドポイントで使用されるルートテーブルの vpce-0ec5da4d265227786 (vpce-0ec5da4d265227786) です。



- にACLアタッチされたネットワークは、トラフィックVPCを許可します。
- にアタッチされたセキュリティグループは、トラフィックVPCを許可します。

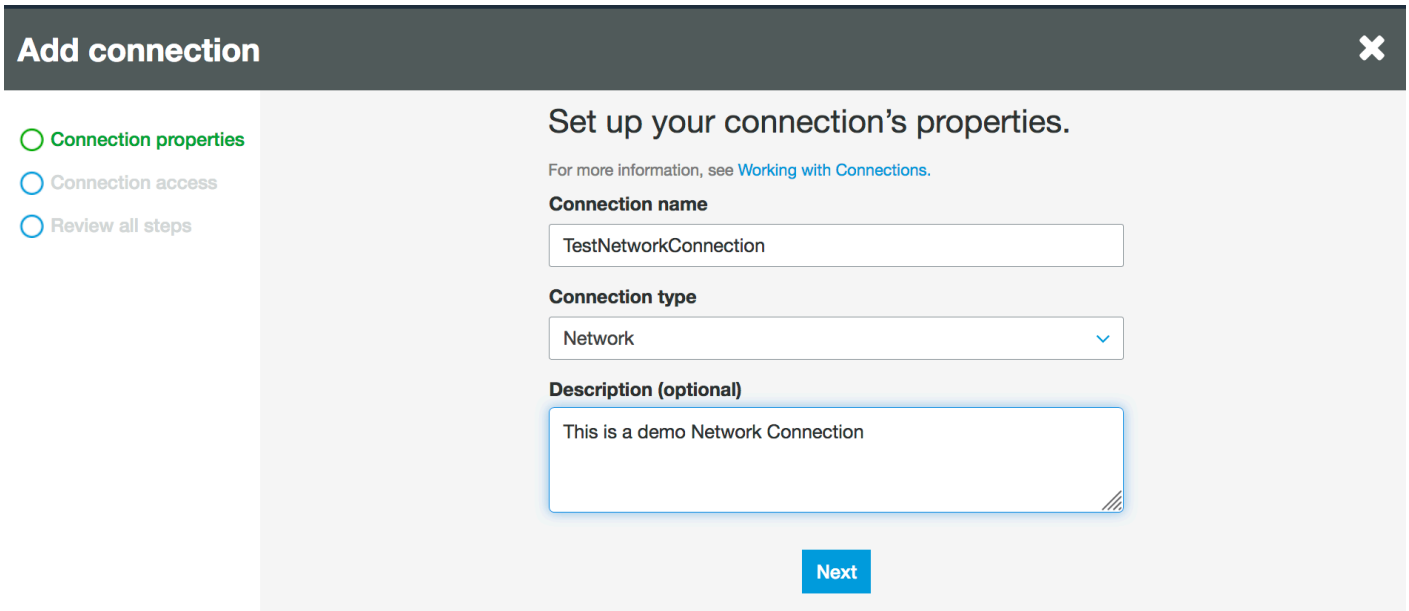
Amazon S3 への接続を作成する

通常、Amazon Virtual Private Cloud (Amazon VPC) 内にリソースを作成して、パブリックインターネット経由でアクセスできないようにします。デフォルトでは、 AWS Glue は 内のリソースにアクセスできませんVPC。 AWS Glue が 内のリソースにアクセスできるようにするにはVPC、VPCサブネットIDsとセキュリティグループ を含む追加の VPC固有の設定情報を提供する必要がありますIDs。`Network` 接続を作成するには、次の情報を指定することが必要です。

- VPC ID
- 内のサブネット VPC
- セキュリティグループ

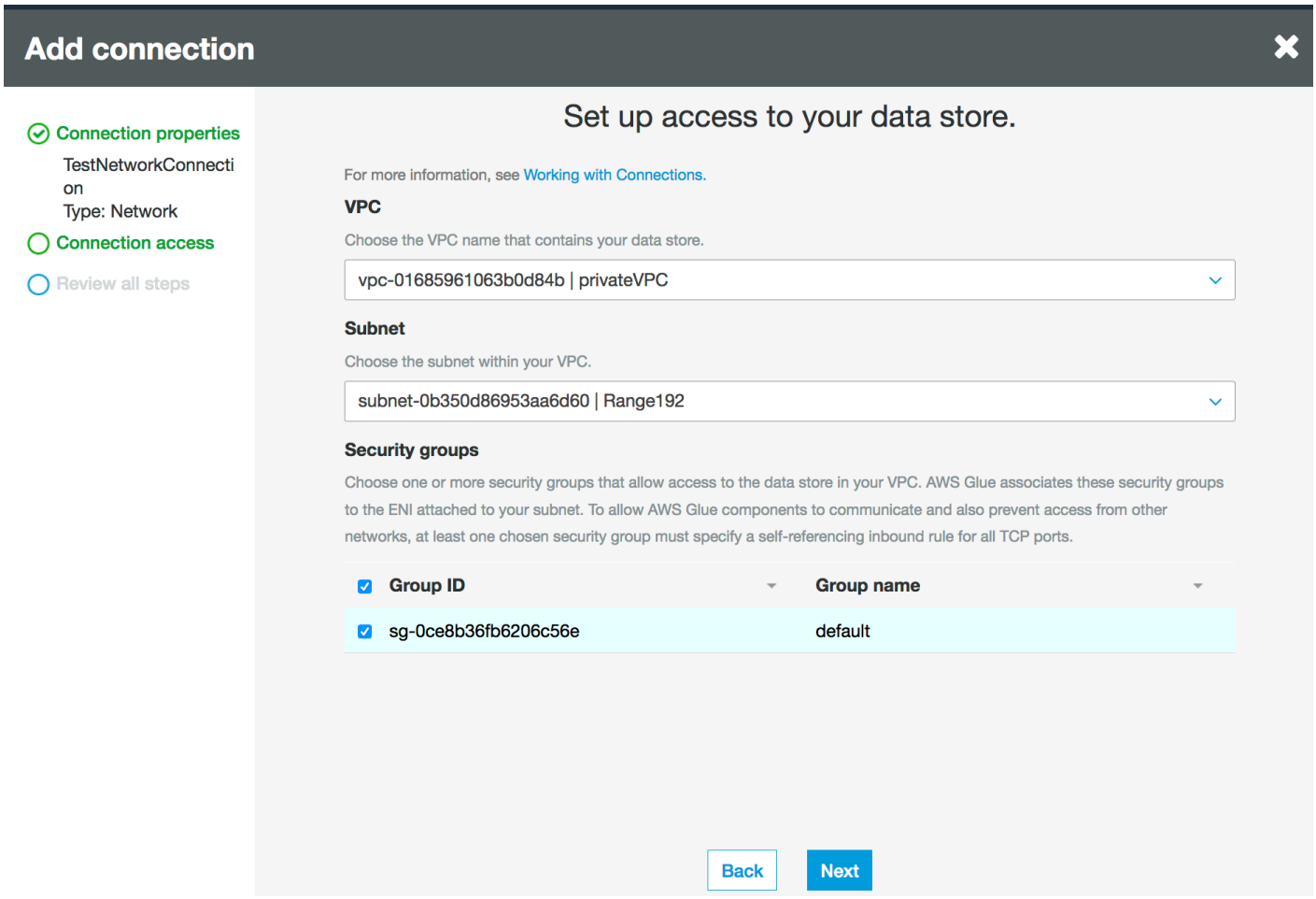
`Network` 接続をセットアップするには

1. AWS Glue コンソールのナビゲーションペインで、[Add connection] (接続の追加) をクリックします。
2. 接続名を入力し、接続タイプとして、[Network] (ネットワーク) を選択します。[Next (次へ)] を選択します。

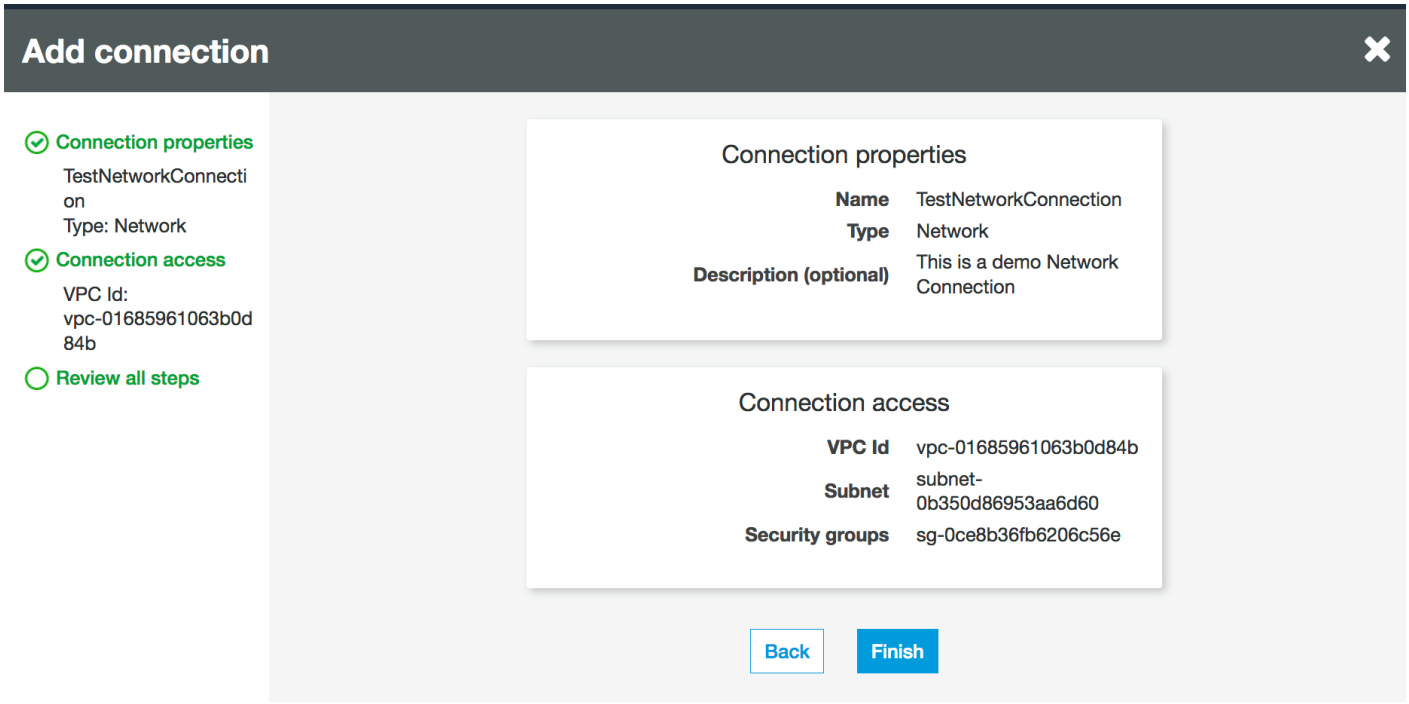


3. VPC、サブネット、セキュリティグループ情報を設定します。
   - VPC: データストアを含むVPC名前を選択します。
   - サブネット: 内のサブネットを選択しますVPC。

- セキュリティグループ: 内のデータストアへのアクセスを許可する 1 つ以上のセキュリティグループを選択しますVPC。



4. [Next (次へ)] を選択します。
5. 接続情報を確認し、[Finish] (完了) をクリックします。



Amazon S3 への接続をテストする

Network 接続を作成したら、VPCエンドポイント内の Amazon S3 データストアへの接続をテストできます。

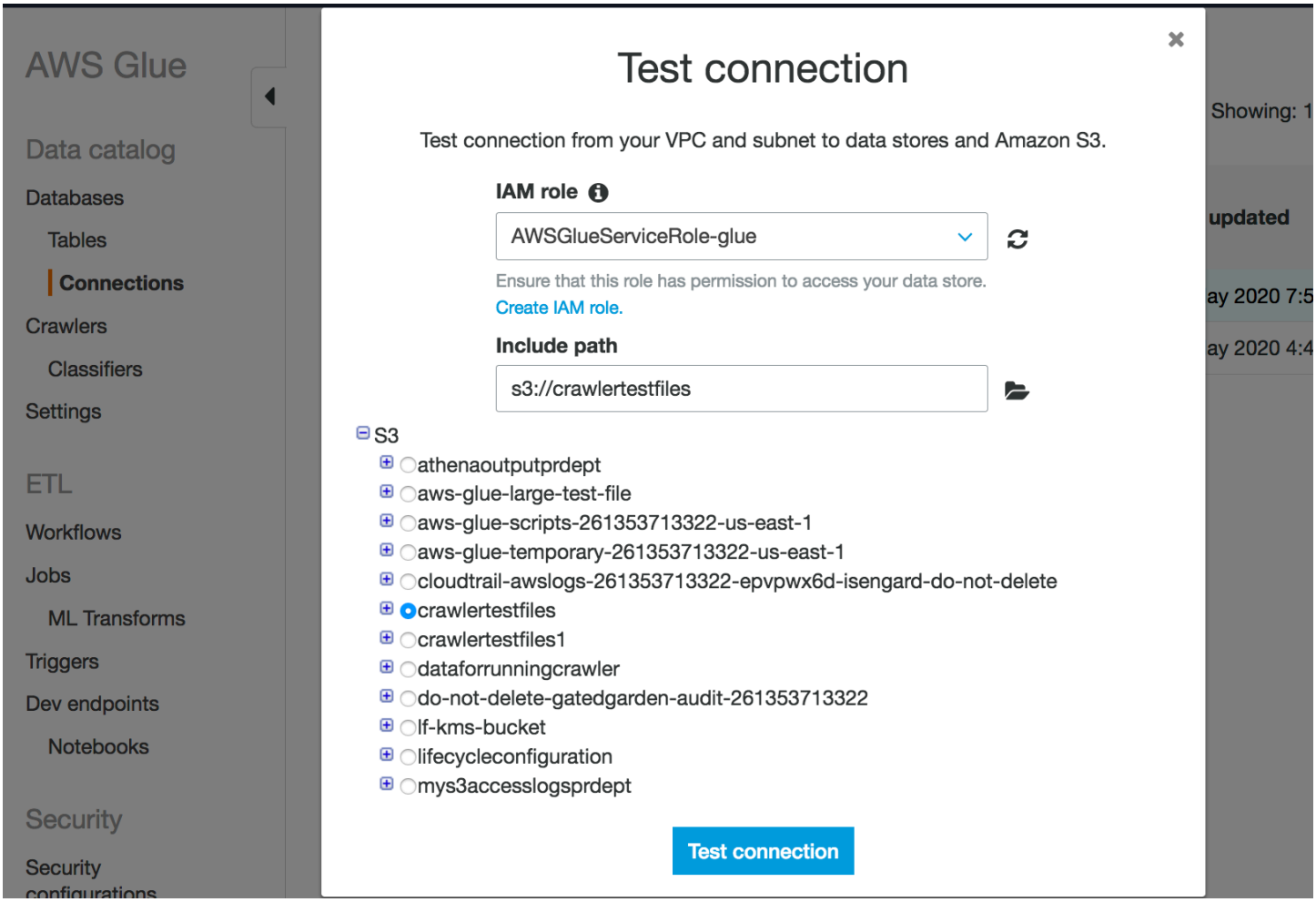
接続のテスト時に、次のエラーが発生することがあります。

- INTERNET CONNECTION ERROR: インターネット接続の問題を示します。
- INVALID BUCKET ERROR: Amazon S3 バケットに問題があることを示します。
- S3 CONNECTION ERROR: Amazon S3 への接続に失敗したことを示します。
- INVALID CONNECTION TYPE: 接続タイプに期待値がないことを示します。 NETWORK
- INVALID CONNECTION TEST TYPE: ネットワーク接続テストのタイプに問題があることを示します。
- INVALID TARGET: Amazon S3 バケットが正しく指定されていないことを示します。

Network 接続をテストするには

1. AWS Glue コンソールで [Network] (ネットワーク) 接続を選択します。
2. [Test connection] を選択します。

3. 前のステップで作成したIAMロールを選択し、Amazon S3 バケットを指定します。
4. テストを開始するには、[Test connection] (接続のテスト) をクリックします。結果を表示するには少し時間がかかることがあります。



エラーが発生した場合は、次のチェックを行います。

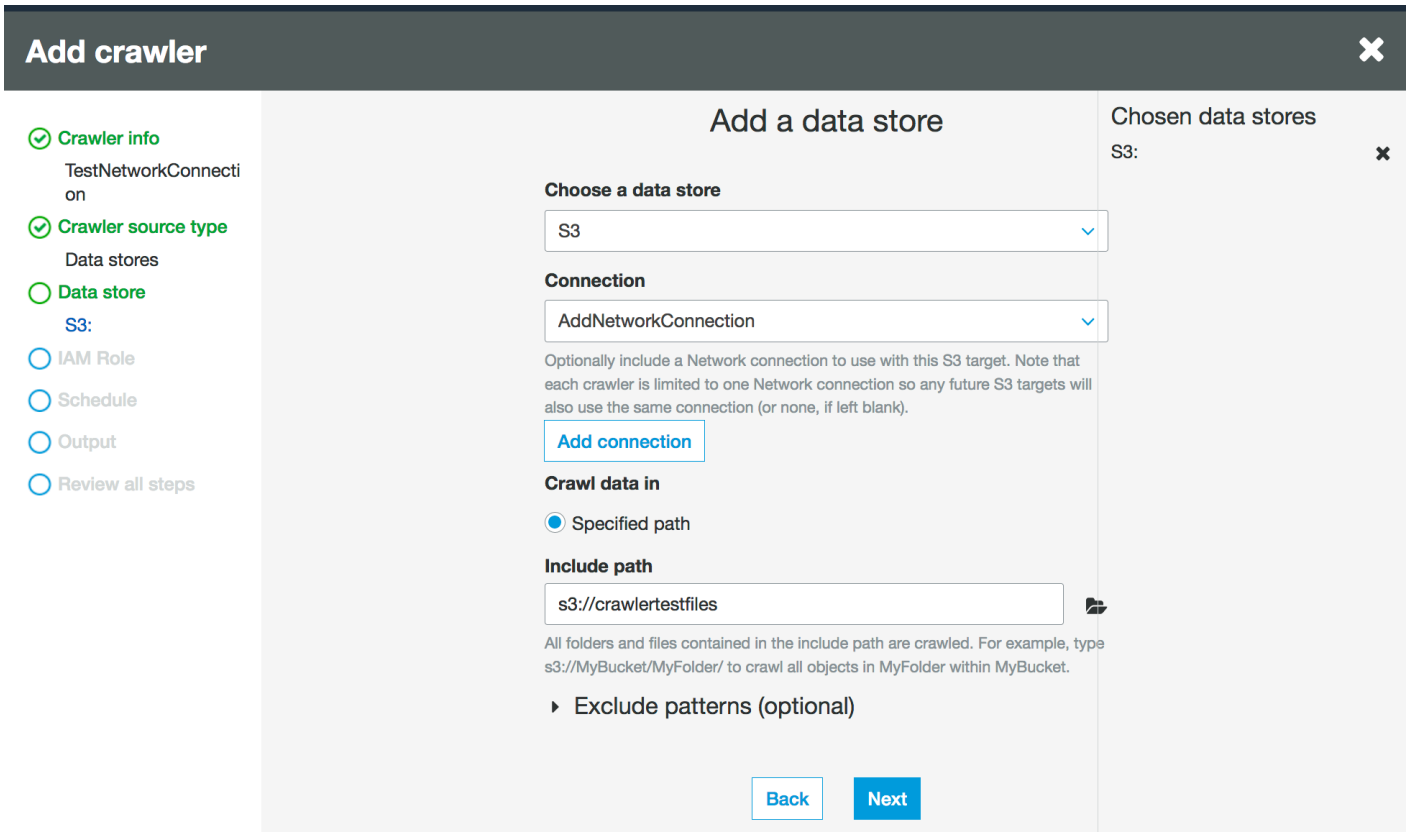
- 選択したロールに正しい権限が与えられている。
- 正しい Amazon S3 バケットが指定されている。
- セキュリティグループとネットワークは、必要な送受信トラフィックACLを許可します。
- VPC 指定した は Amazon S3 VPCエンドポイントに接続されています。

接続のテストに成功したら、クローラーを作成できます。

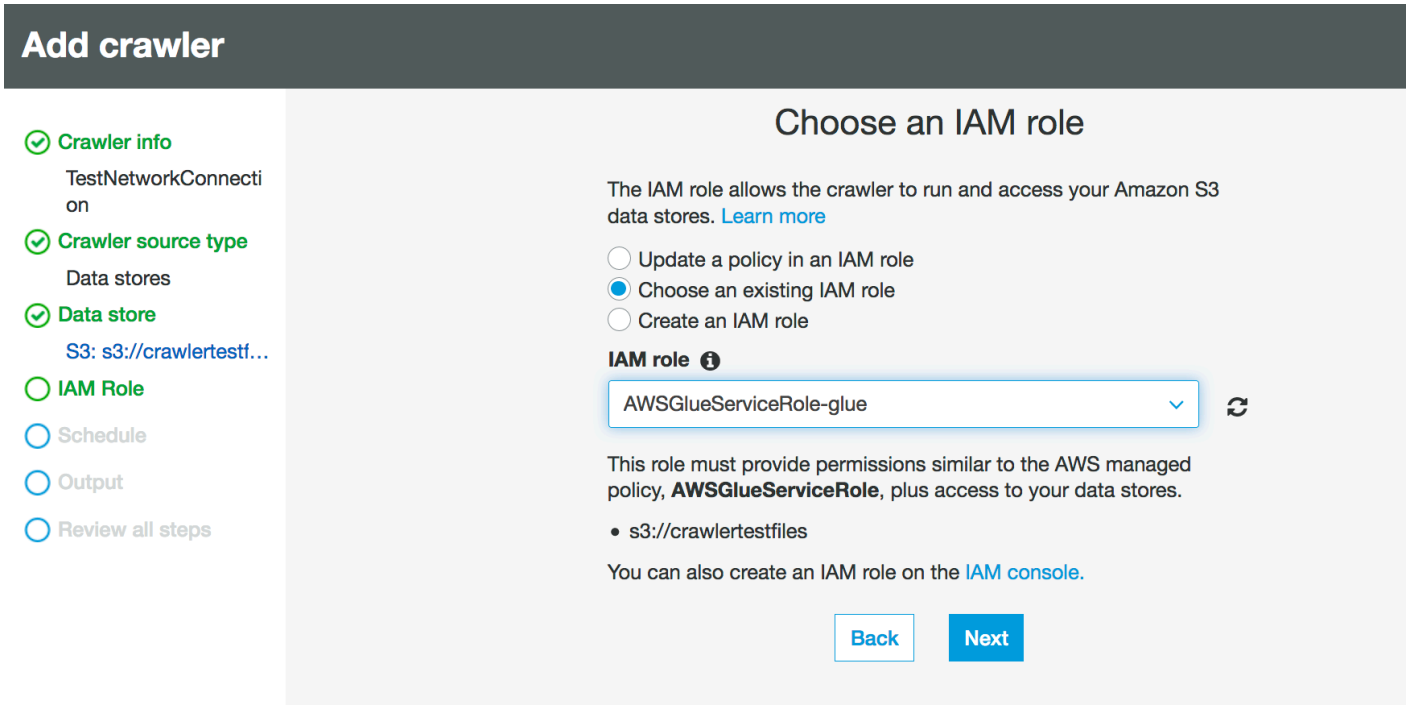
Amazon S3 データストアのクローラーを作成する

これで、作成した Network 接続を指定したクローラーを作成できます。クローラーの作成についての詳細については、「クローラーの設定」を参照してください。

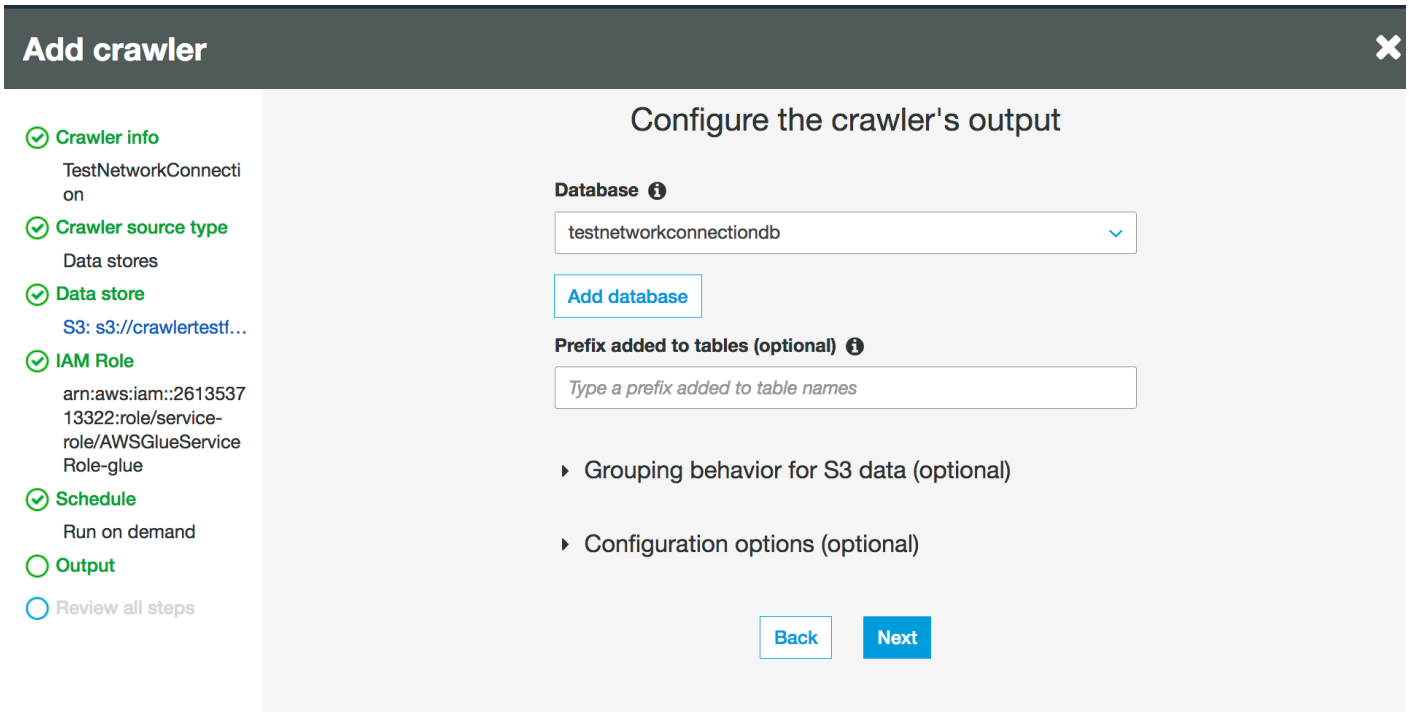
1. まず、 AWS Glue コンソールのナビゲーションペインでクローラーを選択します。
2. [Add crawler (クローラーの追加)] を選択します。
3. クローラーの名前を指定して、[Next] (次へ) をクリックします。
4. データソースを要求されたら、[S3] (S3) を選択し、Amazon S3 バケットプレフィックスと前に作成した接続を指定します。



5. 必要に応じて、同じネットワーク接続に別のデータストアを追加します。
6. IAM ロールを選択します。IAM ロールは、 AWS Glue サービスと Amazon S3 バケットへのアクセスを許可する必要があります。詳細については、「the section called “クローラーの設定”」を参照してください。



7. クローラーのスケジュールを定義します。
8. データカタログの既存のデータベースを選択するか、新しいデータベースエントリを作成します。

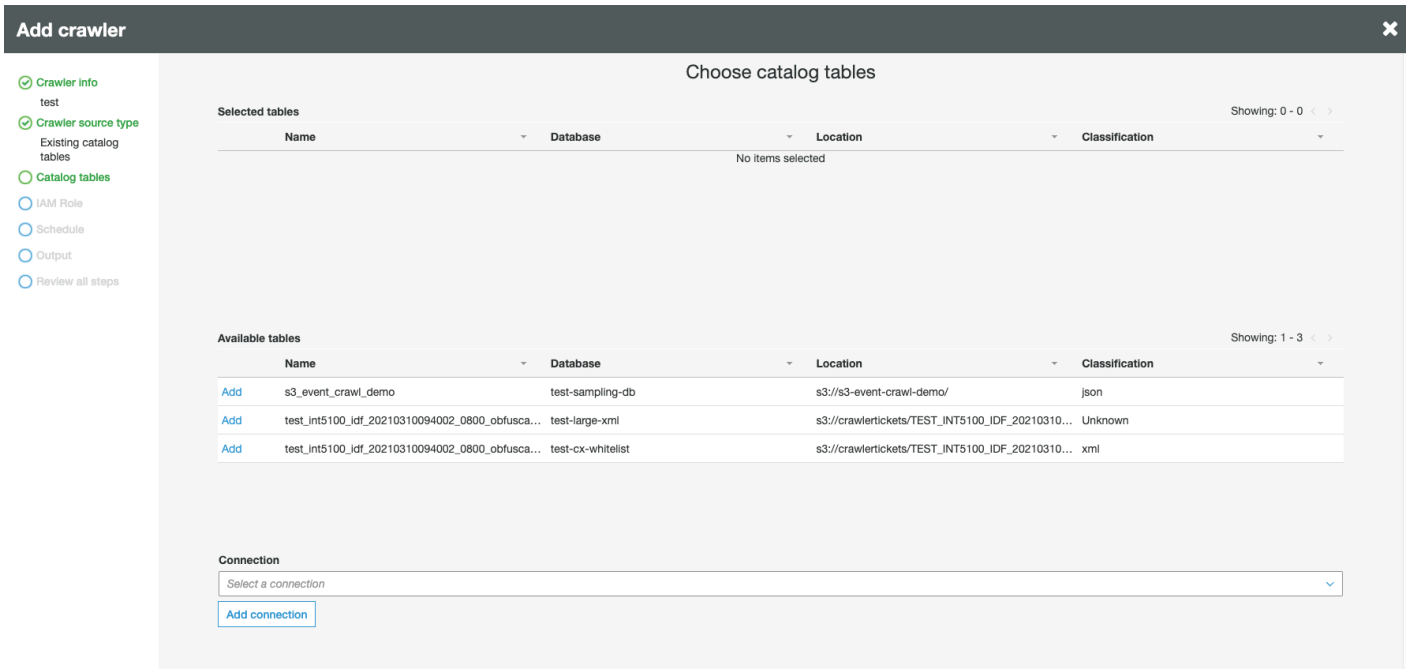


9. 残りのセットアップを完了します。

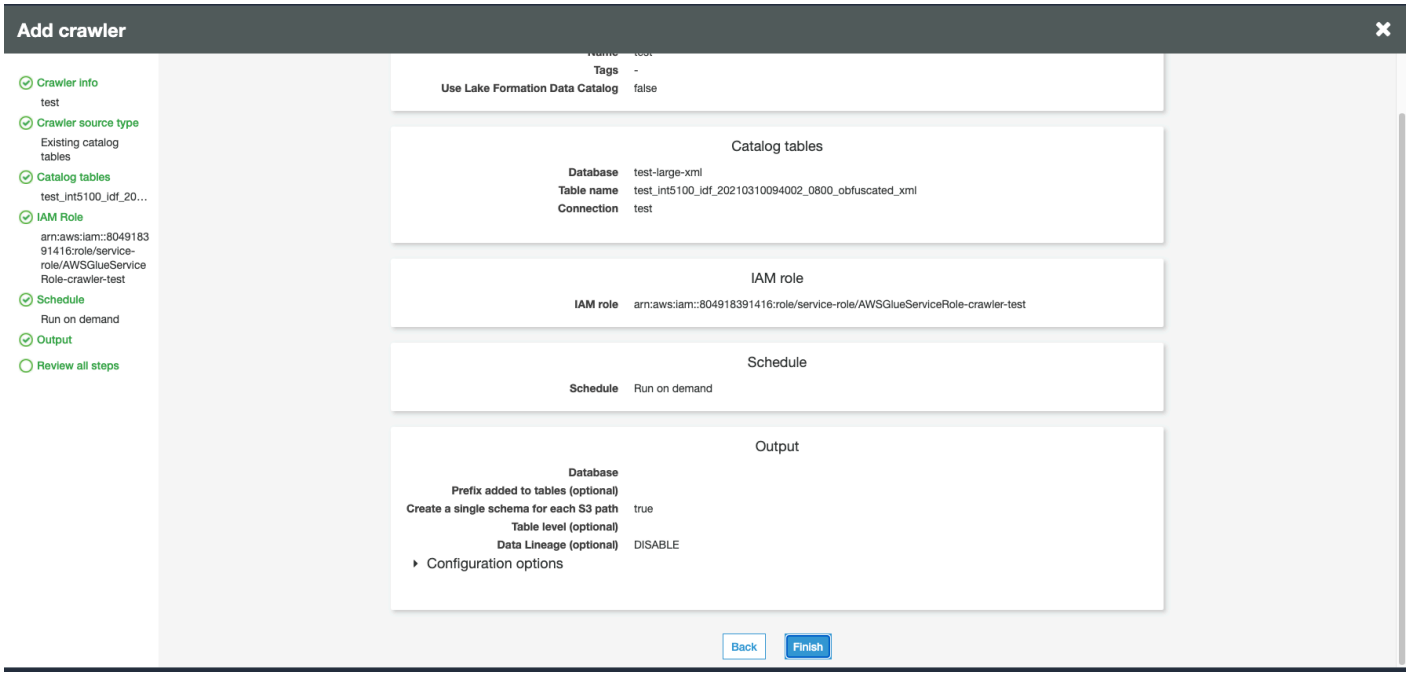
Amazon S3 backed データカタログテーブルのクローラーを作成する

これで、作成した Network 接続とカタログソースタイプを指定するクローラーを作成できます。クローラーの作成についての詳細については、「クローラーの設定」を参照してください。

1. まず、 AWS Glue コンソールのナビゲーションペインでクローラーを選択します。
2. [Add crawler (クローラーの追加)] を選択します。
3. クローラーの名前を指定して、[Next] (次へ) をクリックします。
4. クローラーソースタイプを要求されたら、[既存のカタログテーブル] を選択し、使用可能なテーブルのリストからクロールする既存のカタログテーブルを指定します。

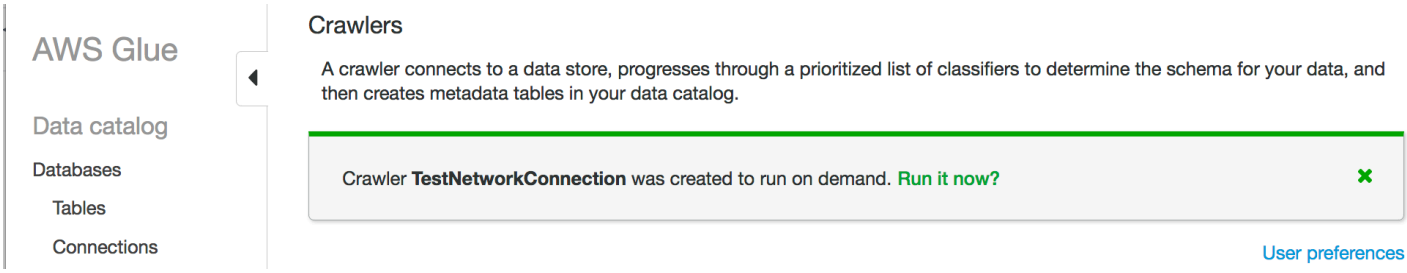


5. IAM ロールを選択します。IAM ロールは、 AWS Glue サービスと Amazon S3 バケットへのアクセスを許可する必要があります。詳細については、「the section called “クローラーの設定”」を参照してください。
6. クローラーのスケジュールを定義します。
7. データカタログの既存のデータベースを選択するか、新しいデータベースエントリを作成します。
8. 残りのセットアップを完了し、ステップを確認します。



クローラーの実行

クローラーを実行します。



トラブルシューティング

VPC ゲートウェイを使用した Amazon S3 バケットに関連するトラブルシューティングについては、「ゲートウェイVPCエンドポイントを使用して S3 バケットに接続できないのはなぜですか？」を参照してください。

AWS Lake Formation で保護されたデータの操作

フルテーブルアクセス

AWS Glue は、AWS Lake Formation で保護されたテーブルのフルテーブルアクセス (FTA) をサポートします。これにより、ETL ジョブはフルテーブルアクセス許可でデータを読み書きできます。

前提条件

- 適切な IAM ロールとアクセス許可
- データカタログ用に設定された AWS Lake Formation
- 互換性のあるテーブルタイプ (Hive または Iceberg)

主な考慮事項

必要なアクセス許可

- `lakeformation:GetDataAccess` IAM 許可
- AWS Lake Formation テーブルの許可
- Amazon S3 バケットへのアクセス許可

サポートされているテーブルタイプ

- Hive テーブル
- Iceberg テーブルの使用

制限

- Spark Streaming と互換性がありません
- きめ細かなアクセスコントロールと同時に使用することはできません
- Delta または Hudi テーブルをサポートしていません

ベストプラクティス

1. ジョブの実行前に IAM ロールが正しく設定されていることを確認する
2. 開発環境でアクセス許可をテストする
3. アクセス許可関連の問題についてジョブ実行ログをモニタリングする
4. アクセスパターンの明確なドキュメントを維持する

トラブルシューティング

一般的な問題は次のとおりです。

- IAM アクセス許可がない
- AWS Lake Formation 設定が正しくない
- テーブルタイプの互換性の問題

セットアップ手順と設定の詳細については、「フルテーブルアクセスでの AWS Lake Formation の使用」を参照してください。

AWS Glue での接続の問題のトラブルシューティング

AWS Glue クローラーまたはジョブが、データストアにアクセスするため接続プロパティを使用する場合、接続しようとするとエラーが発生する場合があります。指定したVirtual Private Cloud (VPC) とサブネットで Elastic Network Interface を作成する際に、AWS Glue はサブネット内のプライベート IP アドレスを使用しています。 接続で指定されたセキュリティグループは、各 Elastic Network Interface に適用されます。セキュリティグループがアウトバウンドアクセスを許可しているかどうか、またデータベースクラスターへの接続を許可しているかどうかを確認します。

さらに、Apache Spark ではドライバーノードとエグゼキューターノード間の双方向接続が必要です。セキュリティグループのいずれかが、すべての TCP ポートの進入ルールを許可する必要があります。自己参照のセキュリティグループでセキュリティグループのソースを自身に制限することによって、世界に向けて開かれないようにすることができます。

接続の問題をトラブルシューティングするために行える一般的なアクションをいくつか示します。

- 接続のポートアドレスを確認します。
- 接続もしくはシークレットの、ユーザー名とパスワードの文字列を確認します。
- JDBC データストアの場合、着信接続を許可していることを検証します。
- データストアが VPC 内でアクセスできることを検証します。
- AWS Secrets Manager を使用して接続の認証情報を保存する場合、AWS Glue のための IAM ロールには、対象のシークレットにアクセスするための許可が必要です。詳細については、「AWS Secrets Manager ユーザーガイド」の「例: シークレット値を取得するアクセス許可」を参照してください。ネットワークの設定によっては、VPC エンドポイントを作成して VPC と Secrets Manager 間のプライベート接続を確立する必要がある場合もあります。詳細については、「AWS Secrets Manager VPC エンドポイントの使用」を参照してください。

チュートリアル: Elasticsearch 向けの AWS Glue コネクタを使用する

Elasticsearch は、ログ分析、リアルタイムアプリケーションモニタリング、クリックストリーム分析などのユースケースで人気のあるオープンソースの検索および分析エンジンです。AWS Glue Studio で Elasticsearch 向けの AWS Glue コネクタを設定することで、OpenSearch を抽出、変換、ロード (ETL) ジョブ用のデータストアとして使用できます。このコネクタは AWS Marketplace から無料で入手できます。

> **Note**
>
> AWS Marketplace Elasticsearch Spark コネクタは廃止されました。代わりに、Elasticsearch 向けの AWS Glue コネクタを使用してください。

このチュートリアルでは、最小限の手順で Amazon OpenSearch Service ノードに接続する方法を示します。

トピック

- 前提条件
- ステップ 1: (オプション) OpenSearch クラスター情報の AWS シークレットを作成する
- ステップ 2: コネクタをサブスクライブする
- ステップ 3: AWS Glue Studio でコネクタをアクティブにして、接続を作成します。
- ステップ 4: ETL ジョブの IAM ロールを設定する
- ステップ 5: OpenSearch 接続を使用するジョブを作成する
- ステップ 6: ジョブを実行する

前提条件

このチュートリアルを使用するには、以下のものが必要です。

- AWS Glue Studio へのアクセス
- AWS Cloud の OpenSearch クラスターにアクセスする
- (オプション) AWS Secrets Manager にアクセスします。

ステップ 1: (オプション) OpenSearch クラスター情報の AWS シークレットを作成する

接続の認証情報を安全に保存して使用するには、認証情報を AWS Secrets Manager に保存します。作成するシークレットは、チュートリアルの後半で接続で使用されます。認証情報のキーと値のペアは、通常の接続オプションとして Elasticsearch 向けの AWS Glue コネクタに供給されます。

シークレットの作成についての詳細は、AWS Secrets Manager ユーザーガイドの「AWS Secrets Manager を使用したシークレットの作成と管理」を参照してください。

AWS シークレットを作成するには

1. AWS Secrets Manager コンソール にサインインします。
2. サービスの概要ページまたは [シークレット] リストページで、[新しいシークレットの保存] を選択します。
3. [新しいシークレットの保存] ページで、[他の種類のシークレット] を選択します。このオプションの意味は、シークレットの構造と詳細を指定する必要があるということです。
4. の追加キーおよび値のペアを OpenSearch クラスターユーザー名として使用します。例:

   `es.net.http.auth.user`: *`username`*
5. [+ 行を追加] を選択して、パスワードとして別のキーと値のペアを入力します。例:

   `es.net.http.auth.pass`: *`password`*
6. [Next] を選択します。
7. シークレット名を入力します 例えば、my-es-secret です。オプションで説明を含めることができます。

   このチュートリアルの後半で使用するシークレット名を記録し、[Next] (次へ) を選択します。
8. [Next] (次へ) をもう一度選択して、[Store] (保存) を選択してシークレットを作成します。

次のステップ

ステップ 2: コネクタをサブスクライブする

ステップ 2: コネクタをサブスクライブする

Elasticsearch 向けの AWS Glue コネクタは、AWS Marketplace で無料で入手できます。

AWS Marketplace で Elasticsearch 向けの AWS Glue コネクタにサブスクライブするには

1. まだ　License Manager を使用するための AWS アカウント設定をしていない場合、次の操作を行います。

   a. AWS License Manager コンソール、https://console.aws.amazon.com/license-manager を開きます 。

   b. Create customer managed license (カスタマー管理ライセンスの作成) を選択します。

   c. [IAM permissions (one-time setup)] 画面で [I grant AWS License Manager the required permissions] を選択し、次に [Grant permissions] を選択します。

      このウィンドウが表示されない場合は、必要なアクセス許可がすでに設定されています。

2. https://console.aws.amazon.com/gluestudio/ で AWS Glue Studio コンソールを開きます。
3. AWS Glue Studio コンソールのナビゲーションペインで、メニューアイコン (☰) を展開して、[Connectors (コネクタ) を選択します。
4. [Connectors] (コネクタ) ページで、[Go to AWS Marketplace] をクリックします。
5. AWS Marketplace の [AWS Glue Studio の製品を検索] セクションで、検索フィールドに「Elasticsearch 向けの AWS Glue コネクタ」と入力し、Enter キーを押します。
6. [Elasticsearch 向けの AWS Glue コネクタ] というコネクタの名前を選択します。
7. コネクタの製品ページで、そのコネクタに関するタブを開いて情報を表示します。続行する準備が整ったら、[Continue to Subscribe] (スクライブを続行する) を選択します。
8. ソフトウェアの利用規約を確認します。[Accept Terms] (規約に同意する) をクリックします。
9. サブスクリプションプロセスが完了すると、「Thank you for subscribing to this product! You can now configure your software.」(製品にサブスクライブしていただきありがとうございます。ソフトウェアの設定が可能になりました。) という通知が表示されます。バナーの上に [Continue to Configuration] (設定に進む) ボタンがあります。[Continue to Configuration (設定に進む)] を選択します。
10. [Configure this software] (このソフトウェアを設定) ページで、フルフィルメントオプションを選択します。AWS Glue 1.0/2.0 または AWS Glue 3.0 を選択できます。[Continue to Launch] (続行して起動する) を選択します。

次のステップ

ステップ 3: AWS Glue Studio でコネクタをアクティブにして、接続を作成します。

ステップ 3: AWS Glue Studio でコネクタをアクティブにして、接続を作成します。

[Continue to Launch (起動を続ける) を選択すると、AWS Marketplace の [Launch this software] (ソフトウェアの起動) ページが表示されます。AWS Glue Studio でリンクを使用してコネクタをアクティブ化した後、接続を作成します。

コネクタを展開し、AWS Glue Studio で接続を作成する方法

1. AWS Marketplace コンソールの [Launch this software] (ソフトウェアの起動) ページで、Usage Instructions (使用方法) を選択し、表示されるウィンドウでリンクを選択します。

   ブラウザ は AWS Glue Studio コンソールの 「Create marketplace connection] ページにリダイレクトされます。
2. 接続の名前を入力します。例えば、my-es-connection です。
3. [Connection access] (接続アクセス) セクションの [Connection credential type] (接続の認証情報タイプ) で、[User name and password] (ユーザー名とパスワード) を選択します。
4. [AWS シークレット] で、シークレット名を入力します。例えば、my-es-secret です。
5. [Network options] (ネットワークオプション) セクションで、VPC 情報を入力して OpenSearch クラスターに接続します。
6. [Create connection and activate connector] (接続の作成とコネクタのアクティブ化) を選択します。

次のステップ

ステップ 4: ETL ジョブの IAM ロールを設定する

ステップ 4: ETL ジョブの IAM ロールを設定する

AWS Glue ETLジョブを作成するとき、使用するジョブの AWS Identity and Access Management (IAM) ロールを指定します。ロールは、Amazon S3 (ソース、ターゲット、スクリプト、ドライバーファイル、一時ディレクトリ) および AWS Glue Data Catalog オブジェクトを含む、ジョブで使用されるすべてのリソースへのアクセスを許可する必要があります。

AWS Glue ETL ジョブで想定される IAM ロールは、前のセクションで作成したシークレットにもアクセスできる必要があります。デフォルトでは、AWS が管理するロール `AWSGlueServiceRole`

はシークレットにアクセスできません。シークレットのアクセスコントロールを設定するには、「AWS Secrets Manager の認証とアクセスコントロール」および「特定のシークレットへのアクセスの制限」を参照してください。

ETL ジョブの IAM ロールを設定するには

1. the section called “ETL ジョブに必要なIAMアクセス許可を確認する” で説明しているように、アクセス許可を設定します。
2. AWS Glue Studio で説明しているように、the section called “コネクタの使用に必要なアクセス許可” でコネクタを使用するときに必要な追加のアクセス許可を設定します。

次のステップ

ステップ 5: OpenSearch 接続を使用するジョブを作成する

ステップ 5: OpenSearch 接続を使用するジョブを作成する

ETL ジョブのロールを作成した後、AWS Glue Studio で Open Spark ElasticSearch の接続とコネクタを使用するジョブを作成できます。

ジョブが Amazon Virtual Private Cloud (Amazon VPC) で実行されている場合は、VPC が正しく設定されていることを確認してください。詳細については、「the section called “ETL ジョブの VPC を設定します”」(ETL ジョブの VPC を設定します) を参照してください。

Elasticsearch Spark コネクタを使用するジョブを作成するには

1. AWS Glue Studio で コネクタ を選択します。
2. [Your connections] (接続) リストで、作成した接続を選択し、[Create job] を選択します。
3. ビジュアルジョブエディタで、[Data source node] (データソースノード) を選択します。右側の [Data source properties - Connector] (データソースのプロパティ- コネクタ) タブで、コネクタの追加情報を設定します。

    a. [Add schema] (スキーマの追加) を選択し、データソースにデータセットのスキーマを入力します。接続には、データカタログに格納されたテーブルは使用されません。つまり、AWS Glue Studio はデータのスキーマを認識しません。このスキーマ情報は手動で指定する必要があります。スキーマエディタの使用方法については、「the section called “カスタム変換ノードでスキーマを編集する”」を参照してください。

    b. Connection options (接続オプション) を展開します。

c. [Add new option] (新しいオプションの追加) を選択し、AWS シークレットに入力されなかったコネクタに必要な情報を入力します。

- es.nodes: https://<OpenSearch ドメインエンドポイント>
- es.port: 443
- path: test
- es.nodes.wan.only.: true

これらの接続オプションの詳細については、「https://www.elastic.co/guide/en/elasticsearch/hadoop/current/configuration.html」を参照してください。

4. グラフにターゲットノードを追加します。

データターゲットは Amazon S3 にすることも、AWS Glue Data Catalog またはコネクタからの情報を使用して別の場所にデータを書き込むこともできます。例えば、データカタログテーブルを使用して Amazon RDS のデータベースに書き込むことも、コネクタをデータターゲットとして使用して、AWS Glue でネイティブにサポートされていないデータストアに書き込むこともできます。

データターゲットのコネクタを選択する場合は、そのコネクタ用に作成された接続を選択する必要があります。また、コネクタプロバイダで必要な場合は、コネクタに追加情報を提供するオプションを追加する必要があります。AWS シークレットの情報を含む接続を使用する場合は、接続オプションでユーザー名とパスワード認証を指定する必要はありません。

5. オプションで、the section called “AWS Glue マネージド変換によるデータの変換” の説明の通り、追加のデータソースと 1 つ以上の変換ノードを追加します。
6. ステップ 3 から the section called “ジョブのプロパティを変更する” の説明の通りジョブプロパティを設定し、ジョブを保存します。

次のステップ

ステップ 6: ジョブを実行する

ステップ 6: ジョブを実行する

ジョブを保存した後、ジョブを実行し ETL 操作が行えるようになります。

Elasticsearch 向けの AWS Glue コネクタ用に作成したジョブを実行するには

1. AWS Glue Studio コンソールを使用して、ビジュアルエディタページで [Run](実行) を選択します。
2. 成功バナーで、[Run Details] (実行の詳細) を選択するか、[Runs] (実行) タブをクリックして、ジョブの実行に関する情報を表示します。

インタラクティブセッションで AWS Glue ジョブを構築する

データエンジニアは AWS Glue のインタラクティブセッションを使用して、以前よりも迅速かつ簡単に AWS Glue のジョブをオーサリングできます。

トピック

- AWS Glue インタラクティブセッションの概要
- AWS Glue のインタラクティブセッションの開始方法
- AWS Glue Jupyter および AWS Glue Studio ノートブックのインタラクティブセッションの設定
- スクリプトまたはノートブックの AWS Glue ジョブジョブへの変換
- AWS Glue インタラクティブセッションでのストリーミングオペレーションの使用
- AWS Glue ジョブスクリプトのローカルでの開発およびテスト
- 開発エンドポイント

AWS Glue インタラクティブセッションの概要

AWS Glue のインタラクティブセッションでは、データ準備および分析アプリケーションの構築、テスト、実行を迅速に行うことができます。インタラクティブセッションは、データ準備のための抽出、変換、ロード (ETL) スクリプトを構築してテストするためのプログラムおよびビジュアルインターフェイスを提供します。インタラクティブセッションでは Apache Spark 分析アプリケーションが実行され、リモートの Spark ランタイム環境にオンデマンドでアクセスできます。AWS Glue はこれらのインタラクティブセッションのサーバーレス Spark を透過的に管理します。

インタラクティブセッションには柔軟性が備わっているため、お好みの環境からアプリケーションを構築し、テストできます。インタラクティブセッションは、AWS Command Line Interface とAPIを介して作成、操作することができます。Jupyter と互換性のあるノートブックを使用して、ノートブックスクリプトを視覚的に作成し、テストできます。インタラクティブセッションは、PyCharm、IntelliJ、VS Code などの IDE との統合を含め、Jupyter が統合するほぼすべてと統合できるオープンソースの Jupyter カーネルを提供します。これにより、ローカル環境でコードをオーサリングし、インタラクティブセッションのバックエンドでシームレスに実行できます。

インタラクティブセッション API を使用すると、Spark インフラストラクチャを管理する必要なく Apache Spark 分析を使用するアプリケーションをプログラムで実行できます。単一のインタラクティブセッション内で 1 つ以上の Spark ステートメントを実行できます。

したがって、インタラクティブセッションは、より速く、安価で、柔軟にデータ準備および分析アプリケーションを構築して実行することができます。インタラクティブセッションの使用方法については、このセクションのドキュメントを参照してください。AWS Glue がサポートするマジック

制限

- ジョブのブックマークは、インタラクティブセッションではサポートされていません。
- AWS Command Line Interface を使用したノートブックの作成はサポートされていません。
- AWS Glue Studio ノートブックは Scala をサポートしていません。

AWS Glue のインタラクティブセッションの開始方法

これらのセクションでは、AWS Glue の対話型セッションをローカルで実行する方法について説明します。

ローカルでインタラクティブセッションを設定するための前提条件

インタラクティブセッションをインストールするための前提条件は次のとおりです。

- サポートされている Python バージョンは 3.6 ~ 3.10 以降です。
- MacOS/Linux および Windows の手順については、以下のセクションを参照してください。

Jupyter と AWS Glue インタラクティブセッション Jupyter カーネルのインストール

次を使用してカーネルをローカルにインストールします。

`install-glue-kernels` コマンドは、pyspark カーネルと Spark カーネルの両方に jupyter kernelspec をインストールし、また適切なディレクトリに口ゴをインストールします。

```
pip3 install --upgrade jupyter boto3 aws-glue-sessions
```

```
install-glue-kernels
```

Jupyter の実行

Jupyter Notebook を実行するには、以下の手順に従います。

1. 次のコマンドを実行して Jupyter Notebook をインストールします。

```
jupyter notebook
```

2. [New] (新規) を選択して、次の AWS Glue カーネルのいずれかを選択して AWS Glue に対するコーディングを開始します。

セッション認証情報とリージョンの設定

MacOS/Linux の手順

AWS Glue インタラクティブセッションには、AWS Glue ジョブおよび開発エンドポイントと同じ IAM アクセス許可が必要です。インタラクティブセッションで使用するロールは、次のいずれかの方法で指定します。

1. `%iam_role` と `%region` マジックを使用
2. `~/.aws/config` の追加行を使用

マジックを使用したセッションロールの設定

最初のセルで、実行された最初のセルに「`%iam_role <YourGlueServiceRole>`」と入力します。

`~/.aws/config` を使用したセッションロールの設定

インタラクティブセッションの AWS Glue サービスロールは、ノートブック自体で指定することも、AWS CLI 設定と一緒に保存することもできます。AWS Glue ジョブで通常使用するロールがある場合、これがロールになります。AWS Glue ジョブに使用するロールがない場合は、「AWS Glue 用の IAM アクセス許可の設定」のガイドに従ってセットアップしてください。

このロールをインタラクティブセッションのデフォルトのロールとして設定するには、以下の手順を実施します。

1. テキストエディタで `~/.aws/config` を開きます。
2. AWS Glue に使用するプロファイルを探します。いずれのプロファイルも使わない場合は、`[Default]` プロファイルを使用します。
3. `glue_role_arn=<AWSGlueServiceRole>` に示すように、使用するロールのプロファイルに 1 行を追加します。

4. [オプション]: プロファイルにデフォルトのリージョンセットがない場合は、region=us-east-1 でリージョンを追加し、us-east-1 を対象リージョンに置き換えることをお勧めします。
5. 設定を保存します。

詳細については、「IAM を使用したインタラクティブセッション」を参照してください。

Windows の手順

AWS Glue インタラクティブセッションには、AWS Glue ジョブおよび開発エンドポイントと同じ IAM アクセス許可が必要です。インタラクティブセッションで使用するロールは、次のいずれかの方法で指定します。

1. %iam_role と %region マジックを使用
2. ~/.aws/config の追加行を使用

マジックを使用したセッションロールの設定

最初のセルで、実行された最初のセルに「%iam_role <YourGlueServiceRole>」と入力します。

~/.aws/config を使用したセッションロールの設定

インタラクティブセッションの AWS Glue サービスロールは、ノートブック自体で指定することも、AWS CLI 設定と一緒に保存することもできます。AWS Glue ジョブで通常使用するロールがある場合、これがロールになります。AWS Glue ジョブに使用するロールがない場合は、「AWS Glue での IAM アクセス許可のセットアップ」 のガイドに従って、設定してください。

このロールをインタラクティブセッションのデフォルトのロールとして設定するには、以下の手順を実施します。

1. テキストエディタで ~/.aws/config を開きます。
2. AWS Glue に使用するプロファイルを探します。いずれのプロファイルも使わない場合は、[Default] プロファイルを使用します。
3. glue_role_arn=<AWSGlueServiceRole> に示すように、使用するロールのプロファイルに 1 行を追加します。

4. [オプション]: プロファイルにデフォルトのリージョンセットがない場合は、region=us-east-1 でリージョンを追加し、us-east-1 を対象リージョンに置き換えることをお勧めします。
5. 設定を保存します。

詳細については、「IAM を使用したインタラクティブセッション」を参照してください。

インタラクティブセッションプレビューからのアップグレード

カーネルがバージョン 0.27 でリリースされたときに、新しい名前でアップグレードされました。カーネルのプレビューバージョンをクリーンアップするには、ターミナルまたは PowerShell から次のコマンドを実行します。

> **Note**
>
> カスタムサービスモデルを必要とする他の AWS Glue プレビューに参加している場合は、このカーネルを削除することで、対象のカスタムサービスモデルも削除されます。

```
# Remove Old Glue Kernels
jupyter kernelspec remove glue_python_kernel
jupyter kernelspec remove glue_scala_kernel

# Remove Custom Model
cd ~/.aws/models
rm -rf glue/
```

SageMaker AI Studio でのインタラクティブセッションの使用

AWS Glue インタラクティブセッションは、データサイエンティストおよびエンジニアがデータ準備および分析アプリケーションを迅速に構築、テスト、実行するために使用できる、オンデマンドのサーバーレス Apache Spark ランタイム環境です。Amazon SageMaker AI Studio Classic ノートブックを起動することにより、AWS Glue インタラクティブセッションを開始できます。

詳細については、「AWS Glue インタラクティブセッションを使用してデータの準備」を参照してください。

Microsoft Visual Studio Code によるインタラクティブセッションの使用

前提条件

- AWS Glue インタラクティブセッションをインストールし、Jupyter Notebook で動作することを確認します。
- Jupyter により Visual Studio Code をダウンロードし、インストールします。詳細については、「Jupyter Notebooks in VS Code」(VS コードでの Jupyter Notebook) を参照してください。

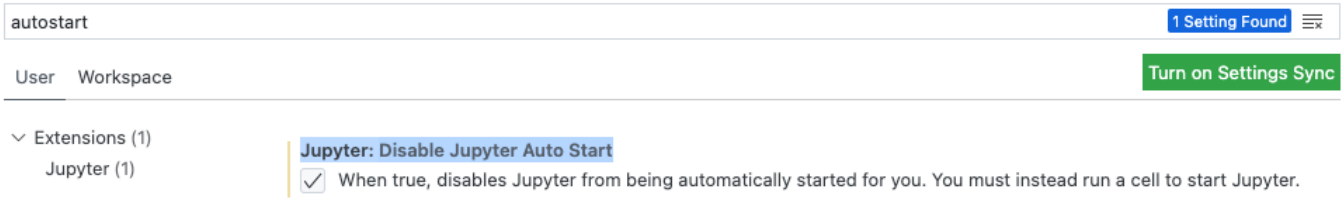
VSCode でインタラクティブセッションを始めるには

1. VS Code で Jupyter AutoStart を無効にします。

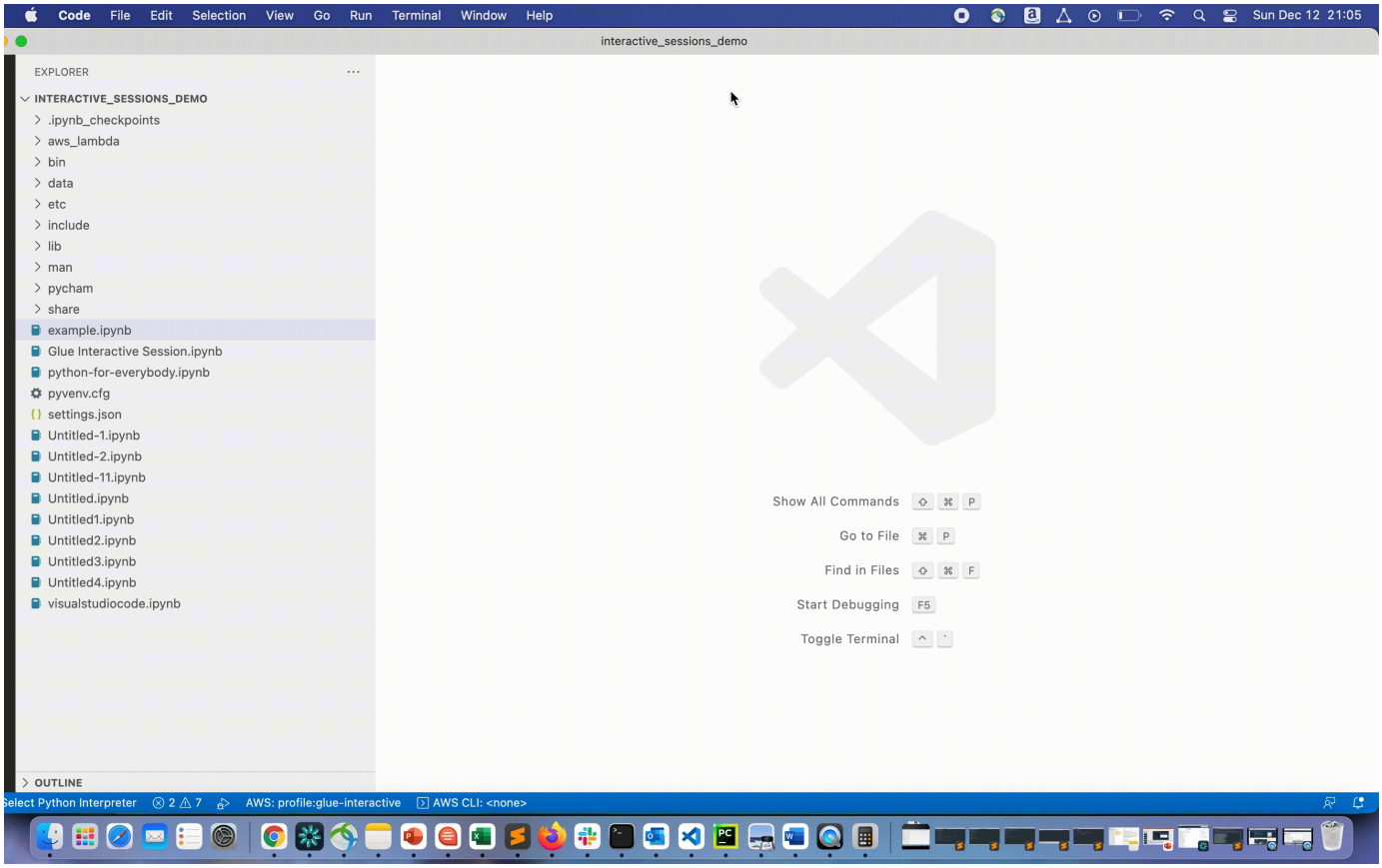
   Visual Studio Code では、Jupyter カーネルが自動起動します。セッションが開始済みであるため、マジックは有効になりません。Windows で [自動起動] を無効にするには、[ファイル] > [設定] > [拡張機能] > [Jupyter] に移動し、Jupyter を右クリックして [拡張機能の設定] を選択します。

   MacOS では、[コード] > [設定] > [拡張機能] > [Jupyter] に移動し、Jupyter を右クリックして [拡張機能の設定] を選択します。

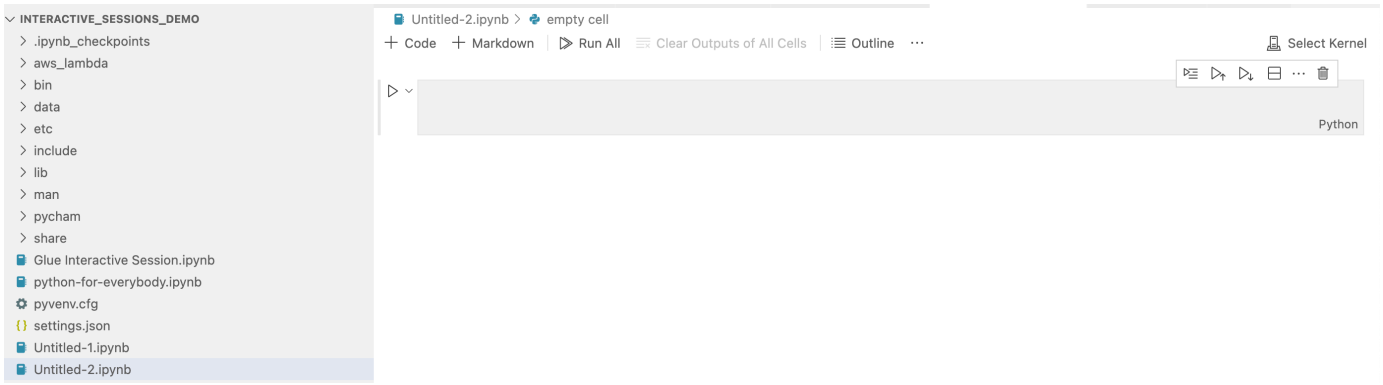
   [Jupyter: 自動起動を無効にする] が表示されるまで下にスクロールします。[true の場合、Jupyter が自動的に起動されないようにする] のチェックボックスをオンにします。You must instead run a cell to start Jupyter.] (Jupyter を開始するには、セルを実行してください)

2. [File] (ファイル) > [New File] (新規ファイル) > [Save] (保存) の順に選択して、選択した名前でこのファイルを `.ipynb` 拡張子として保存するか、または [Select a language] (言語の選択) の直下で [jupyter] を選択してファイルを保存します。



3. ファイルをダブルクリックします。Jupyter シェルが表示され、ノートブックが開きます。



4. Windows では、最初にファイルを作成するとき、デフォルトではカーネルが選択されていません。[Select Kernel] (カーネルの選択) をクリックすると、使用可能なカーネルのリストが表示されます。[Glue PySpark] を選択します。

   MacOS で [Glue PySpark] カーネルが表示されない場合は、次の手順を試してください。

   1. URL を取得するには、ローカルの Jupyter セッションを実行します。

      たとえば、次のコマンドを実行して Jupyter Notebook をインストールします。

```
jupyter notebook
```

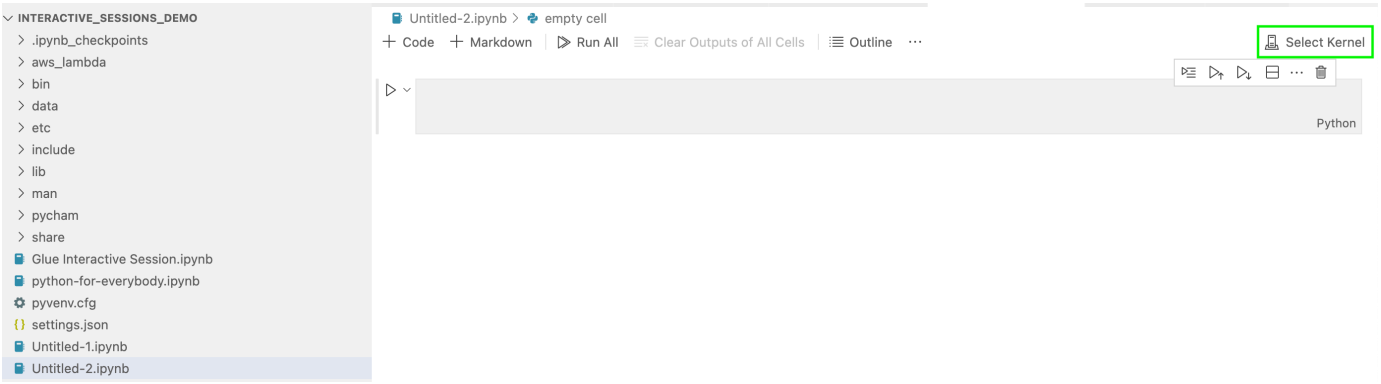
ノートブックを初めて実行すると、http://localhost:8888/?token=3398XXXXXXXXXXXXXXXX のような URL が表示されます。

URL をコピーします。

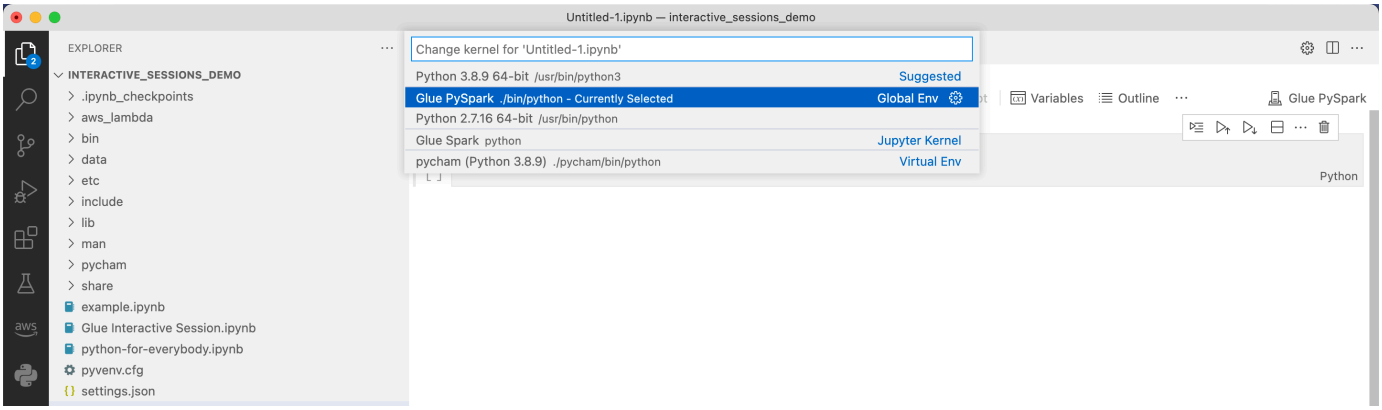
2. VS Code で、現在のカーネルをクリックし、[別のカーネルを選択...]、[既存の Jupyter Server...] を選択します。上記の手順でコピーした URL を貼り付けます。

   エラーメッセージが表示される場合は、「VS コードジュピターウィキ」を参照してください。

3. 成功すると、カーネルが [Glue PySpark] に設定されます。



「Glue PySpark」または「Glue Spark」カーネルを (それぞれ Python と Scala 用に) 選択します。



ドロップダウンリストに [AWS Glue PySpark] および [AWS Glue Spark] カーネルが表示されない場合、上記のステップで AWS Glue カーネルがインストール済みであるか、Visual Studio

Code で `python.defaultInterpreterPath` 設定が正しいことを確認してください。詳細については、「python.defaultInterpreterPath 設定の説明」を参照してください。

5. AWS Glue のインタラクティブセッションを作成します。セッションの作成に進み、Jupyter Notebook と同じ方法を使用します。最初のセルの先頭に必要なマジックを指定し、コードのステートメントを実行します。

IAM を使用したインタラクティブセッション

ここでは、AWS Glue インタラクティブセッションのセキュリティ上の考慮事項について説明します。

トピック

- インタラクティブセッションで使用する IAM プリンシパル
- クライアントプリンシパルを設定する
- ランタイムロールを設定する
- TagOnCreate を使用してセッションをプライベートにする
- IAM ポリシーに関する考慮事項

インタラクティブセッションで使用する IAM プリンシパル

AWS Glue インタラクティブセッションでは、2 つの IAM プリンシパルを使用します。

- [Client principal] (クライアントプリンシパル): クライアントプリンシパル (ユーザーまたはロール) は、プリンシパルのアイデンティティベースの認証情報で構成された AWS Glue クライアントからのインタラクティブセッションの API オペレーションを認証します。例えば、AWS Glue コンソールにアクセスするために通常使用する IAM ロールのプリンシパルがこれに該当する場合があります。また、AWS Command Line Interface に認証情報が使用される IAM ユーザー、またはインタラクティブセッション の Jupyter カーネルで使用される AWS Glue クライアントもこれに該当する場合があります。
- ランタイムロール: ランタイムロールは、クライアントプリンシパルがインタラクティブセッション API オペレーションに渡す IAM ロールです。AWS Glue がこのロールを使用して、セッションでステートメントを実行します。例えば、このロールは AWS Glue ETL ジョブの実行に使用される場合があります。

  詳しくは、「ランタイムロールを設定する 」を参照してください。

クライアントプリンシパルを設定する

インタラクティブセッション API を呼び出すには、アイデンティティポリシーをクライアントプリンシパルにアタッチする必要があります。このロールには、`CreateSession` などのインタラクティブセッション API に渡す実行ロールへの `iam:PassRole` アクセス権が必要です。例えば、AWSGlueConsoleFullAccess 管理ポリシーを IAM ロールにアタッチできるため、同一または類似のポリシーがアタッチされているアカウント内のすべての IAM ユーザーが、アカウントで作成されたすべてのセッション (ランタイムステートメントやキャンセルステートメントなど) にアクセスできます。

セッションを保護し、特定の IAM ロール (セッションを作成したユーザーに関連するものなど) のみのプライベートセッションにする場合は、AWS Glue インタラクティブセッションのタグベースの TagOnCreate という認可コントロールを使用できます。所有者タグベースのスコープダウンマネージドポリシーが TagOnCreate を使用してセッションをプライベートにする方法について詳しくは、「TagOnCreate を使用してセッションをプライベートにする」を参照してください。アイデンティティベースポリシーの詳細については、「AWS Glue のアイデンティティベースポリシー」を参照してください。

ランタイムロールを設定する

AWS Glue がインタラクティブセッションでステートメントを引き受け、実行することを許可するには、IAM ロールを CreateSession API オペレーションに渡す必要があります。ロールには、通常の AWS Glue ジョブの実行に必要なアクセス許可と同じ IAM アクセス許可が必要です。例えば、AWSGlueServiceRole ポリシーを使用して、AWS Glue がユーザーに代わって AWS サービスを呼び出すサービスロールを作成できます。AWS Glue コンソールを使用すると、ユーザーに代わってサービスロールが自動的に作成されるか、既存のものを使用します。独自の IAM ロールを作成し、独自の IAM ポリシーをアタッチして、同様の許可を許可することもできます。

セッションを保護し、作成した ユーザーのみのプライベートセッションにする場合は、AWS Glue インタラクティブセッションの TagOnCreate というタグベースの認可コントロールを使用できます。所有者タグベースのスコープダウンマネージドポリシーが TagOnCreate を使用してセッションをプライベートにする方法について詳しくは、「TagOnCreate を使用してセッションをプライベートにする」を参照してください。アイデンティティベースポリシーの詳細については、「AWS Glue のアイデンティティベースのポリシー」を参照してください。IAM コンソールから実行ロールを作成していて、TagOnCreate 機能を使用してサービスをプライベートにしたい場合は、以下の手順に従います。

1. ロールタイプを `Glue` に設定して IAM ロールを作成します。

2. AWS Glue マネージドポリシー AwsGlueSessionUserRestrictedServiceRole をアタッチします。
3. プレフィックスとしてロール名の先頭にポリシー名 AwsGlueSessionUserRestrictedServiceRole を付けます。例えば、AwsGlueSessionUserRestrictedServiceRole-myrole という名前のロールを作成するのであれば、AWS Glue マネージドポリシー AwsGlueSessionUserRestrictedServiceRole をアタッチします。
4. 次のような信頼ポリシーをアタッチして、AWS Glue がロールを引き受けることを許可します。

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

インタラクティブセッションの Jupyter カーネルでは、`iam_role` キーを AWS Command Line Interface プロファイルで指定できます。詳細については、「~/.aws/config でのセッションの設定」を参照してください。AWS Glue ノートブックを使用してインタラクティブセッションと対話している場合は、実行する最初のセルで、`%iam_role` マジックの実行ロールを渡すことができます。

TagOnCreate を使用してセッションをプライベートにする

AWS Glue インタラクティブセッションは、名前付きリソースとして、インタラクティブセッション向けにタグ付けとタグベースの認可 (TBAC) をサポートしています。TagResource と UntagResource API を使用する TBAC のほかに、AWS Glue インタラクティブセッションは、CreateSession オペレーションによるセッション作成中にのみ、特定のタグでセッションに「タグ付け」する TagOnCreate 機能をサポートしています。これは、こうしたタグが DeleteSession (つまり UntagOnDelete) から削除されることも意味します。

TagonCreate は、セッションの作成者にセッションをプライベートにするための強力なセキュリティメカニズムとなります。例えば、「owner」RequestTag と ${aws:userId} の値を持つ IAM ポリシーをクライアントプリンシパル (ユーザーなど) にアタッチして、呼び出し元の userId の値と一致する「owner」タグが CreateSession リクエストの userId タグとして提供された場合にのみセッションの作成を許可することができます。このポリシーにより、AWS Glue インタラクティブセッションはセッションリソースを作成し、セッションの作成時にのみ、userId タグでセッションにタグ付けすることができます。さらに、CreateSession 中に渡された実行ロールに「owner」ResourceTag を持つ IAM ポリシーをアタッチすると、セッションへのアクセス (ステートメントの実行など) をセッションの作成者 (つまり ${aws:userId} という値を持つオーナータグ) のみに制限することができます。

TagOnCreate 機能を使用してセッションをセッションの作成者に対してプライベートにするのを簡単にするために、AWS Glue が専用の管理ポリシーとサービスロールを提供します。

IAM AssumeRole プリンシパルを使用して (つまり、IAM ロールを引き受けることによって提供される認証情報を使用して) AWS Glue インタラクティブセッションを作成し、セッションを作成者に対してプライベートにしたい場合は、それぞれ AWSGlueSessionUserRestrictedNotebookPolicy および AWSGlueSessionUserRestrictedNotebookServiceRole と同様のポリシーを使用する必要があります。これらのポリシーでは、AWS Glue が ${aws:PrincipalTag} を使用してオーナータグ値を抽出します。これには、${aws:userId} の値を持つ UserID タグを SessionTag としてロール引き受ける認証情報で渡す必要があります。「ID session tags」(ID セッションタグの受け渡し) を参照してください。認証情報を提供するインスタンスプロファイルで Amazon EC2 インスタンスを使用し、Amazon EC2 インスタンス内でセッションを作成したり、セッションと対話したりする場合は、引き受けるロールの認証情報で ${aws:userId} の値を持つ userID タグを SessionTag として渡す必要があります。

例えば、IAM AssumeRole プリンシパル認証情報を使用してセッションを作成していて、TagOnCreate 機能を使用してサービスをプライベートにする場合は、次の手順に従います。

1. IAM コンソールからランタイムロールを自分で作成します。この AWS Glue マネージドポリシー AwsGlueSessionUserRestrictedNotebookServiceRole をアタッチし、ロール名の前にポリシー名 AwsGlueSessionUserRestrictedNotebookServiceRole を付けてください。例えば、AwsGlueSessionUserRestrictedNotebookServiceRole-myrole という名前のロールを作成し、AWS Glue マネージドポリシー AwsGlueSessionUserRestrictedNotebookServiceRoleをアタッチできます。
2. AWS Glue が上記のロールを引き受けることができるように、以下のような信頼ポリシーを適用します。

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

3. プレフィックス AwsGlueSessionUserRestrictedNotebookPolicy で名前が付けられた別のロールを作成し、AWS Glue マネージドポリシー AwsGlueSessionUserRestrictedNotebookPolicy をアタッチして、セッションをプライベートにします。　マネージドポリシーに加えて、次のインラインポリシーをアタッチして、ステップ 1 で作成したロールに iam:PassRole を許可してください。

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

```
            }
        }
    ]
}
```

4. 上記の IAM AWS Glue に次のような信頼ポリシーをアタッチして、ロールを引き受けます。

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

> **Note**
>
> オプションで、単一のロール (ノートブックロールなど) を使用して、上記の管理ポリシー AwsGlueSessionUserRestrictedNotebookServiceRole と AwsGlueSessionUserRestrictedNotebookPolicy 両方をアタッチできます また、追加のインラインポリシーをアタッチして、自分のロールの `iam:passrole` を AWS Glue に許可します。最後に、上記の信頼ポリシーをアタッチして、`sts:AssumeRole` と `sts:TagSession` を許可します。

AWSGlueSessionUserRestrictedNotebookPolicy

AWSGlueSessionUserRestrictedNotebookPolicy は、タグキー「owner」とプリンシパル (ユーザーまたはロール) の AWS ユーザー ID が一致する場合にのみ、ノートブックから AWS Glue インタラクティブセッションを作成するためのアクセスを提供します。詳細については、「ポリシー変数を使用する場所」を参照してください。このポリシーは、AWS Glue Studio から AWS Glue インタラク

ティブセッションノートブックを作成するプリンシパル (ユーザーまたはロール) にアタッチされます。また、このポリシーは、プリンシパルの AWS ユーザー ID と一致する「owner」タグの値を使用して作成された AWS Glue Studio インタラクティブセッションのリソースと対話するための AWS Glue Studio ノートブックへの十分なアクセスを許可しています。このポリシーは、セッションが作成された後に、AWS Glue セッションリソースから「owner」タグを変更または削除する許可を拒否します。

AWSGlueSessionUserRestrictedNotebookServiceRole

AWSGlueSessionUserRestrictedNotebookServiceRole は、AWS Glue Studio ノートブックを作成したプリンシパル (ユーザーまたはロール) の AWS ユーザー ID と一致する「owner」タグの値を使用して作成された AWS Glue インタラクティブセッションのリソースと対話するための、ノートブックへの十分なアクセスを提供します。詳細については、「ポリシー変数を使用する場所」を参照してください。このサービスロールポリシーは、ノートブックにマジックとして渡されたロールや、CreateSession API に実行ロールとして渡されたロールにアタッチされます。また、このポリシーは、タグキー「owner」と値がプリンシパルの AWS ユーザー ID と一致する場合のみ、ノートブックから AWS Glue インタラクティブセッションを作成することを許可します。このポリシーは、セッションが作成された後に、AWS Glue セッションリソースから「owner」タグを変更または削除する許可を拒否します。このポリシーには、Amazon S3 バケットからの書き込みと読み取り、CloudWatch ログの書き込み、AWS Glue が使用する Amazon EC2 リソースのタグの作成と削除のアクセス許可も含まれます。

ユーザーポリシーを使用してセッションをプライベートにする

AWSGlueSessionUserRestrictedPolicy をアカウント内の各ユーザーにアタッチされた IAM ロールにアタッチして、自分の ${aws:userId} と一致する値を持つ owner タグのみを使用してセッションを作成するように制限できます。AWSGlueSessionUserRestrictedNotebookPolicy および AWSGlueSessionUserRestrictedNotebookServiceRole を使用する代わりに、それぞれ AWSGlueSessionUserRestrictedPolicy および AWSGlueSessionUserRestrictedServiceRole と同様のポリシーを使用する必要があります。詳細については、「アイデンティティベースポリシーを使用する」を参照してください。このポリシーは、作成者、つまり自分の ${aws:userId} を持つ owner タグでセッションを作成したユーザーの ${aws:userId} のみに、セッションへのアクセス範囲を絞り込みます。「ランタイムロールを設定する」の手順に従って IAM コンソールを使用して実行ロールを自分で作成した場合は、AwsGlueSessionUserRestrictedPolicy マネージドポリシーをアタッチすることに加えて、アカウント内の各ユーザーに次のインラインポリシーをアタッチして、以前に作成した実行ロールに `iam:PassRole` を許可します。

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

```
    "Statement": [{
        "Effect": "Allow",
        "Action": [
            "iam:PassRole"
        ],
        "Resource": [
            "arn:aws:iam::*:role/AwsGlueSessionUserRestrictedServiceRole*"
        ],
        "Condition": {
            "StringLike": {
                "iam:PassedToService": [
                    "glue.amazonaws.com"
                ]
            }
        }
    }]
}
```

AWSGlueSessionUserRestrictedPolicy

AWSGlueSessionUserRestrictedPolicy は、タグキー「owner」と AWS ユーザー ID に一致する値が提供されている場合にのみ、CreateSession API を使用して AWS Glue インタラクティブセッションを作成するアクセス権を付与します。このアイデンティティポリシーは、CreateSession API を呼び出すユーザーにアタッチされます。このポリシーでは、「owner」タグと AWS Glue ユーザー ID に一致する値を使用して作成された AWS インタラクティブセッションのリソースとの対話も許可されます。このポリシーは、セッションが作成された後に、AWS Glue セッションリソースから「owner」タグを変更または削除する許可を拒否します。

AWSGlueSessionUserRestrictedServiceRole

AWSGlueSessionUserRestrictedServiceRole - セッションを除くすべての AWS Glue リソースへのフルアクセス権限を提供し、ユーザーがユーザーに関連付けられているインタラクティブセッションのみを作成して使用できるようにします。このポリシーには、他の AWS Glue サービスで Glue リソースを管理するために、AWS が必要とするその他のアクセス許可も含まれています。このポリシーでは、その他の AWS Glue リソースの AWS サービスへのタグの追加も許可されます。

IAM ポリシーに関する考慮事項

インタラクティブセッションは AWS Glue の IAM リソースです。これらは IAM リソースであるため、セッションへのアクセスとインタラクションは IAM ポリシーによって管理されます。管理者によって設定されたクライアントプリンシパルまたは実行ロールに添付された IAM ポリシーに基づい

て、クライアントプリンシパル (ユーザーまたはロール) は新しいセッションを作成し、独自のセッションや他のセッションと対話することができます。

管理者が、アカウントのすべての AWS Glue リソースにアクセスできる AWSGlueConsoleFullAccess や AWSGlueServiceRole のような IAM ポリシーをアタッチした場合、クライアントプリンシパルは相互にコラボレーションできます。例えば、ポリシーで許可されていれば、ユーザーは他のユーザーによって作成されたセッションと対話することができます。

特定のニーズに合わせたポリシーを設定したい場合は、ポリシーのリソースの設定に関する IAM ドキュメントを参照してください。例えば、あるユーザーに属するセッションを分離するには、AWS Glue インタラクティブセッションがサポートしている TagOnCreate 機能を使用できます。「TagOnCreate を使用してセッションをプライベートにする」を参照してください。

インタラクティブセッションは、特定の VPC 条件に基づくセッション作成の制限をサポートします。「条件キーを使って設定を制御するポリシーを制御する」を参照してください。

AWS Glue Jupyter および AWS Glue Studio ノートブックのインタラクティブセッションの設定

Jupyter Magics の概要

Jupyter Magics は、セル先頭またはセル本文全体として実行できるコマンドです。マジックの先頭は、Line Magics の場合は %、Cell Magics の場合は %% です。%region や %connections などの Line Magics は、セル内の複数のマジック、または次の例のようにセル本文に含まれるコードを使用して実行できます。

```
%region us-east-2
%connections my_rds_connection
dy_f = glue_context.create_dynamic_frame.from_catalog(database='rds_tables',
 table_name='sales_table')
```

Cell Magics はセル全体を使用する必要があり、コマンドを複数行に渡すことができます。%%sql の例を以下に示します。

```
%%sql
select * from rds_tables.sales_table
```

Jupyter 向け AWS Glue インタラクティブセッションでサポートされているマジック

以下に、Jupyter Notebookの AWS Glue インタラクティブセッションで使用できるマジックを示します。

Sessions Magics

名前	型	説明
%help	該当なし	すべてのマジックコマンドのために説明と入力タイプのリストを返します。
%profile	文字列	AWS 認証情報プロバイダーとして使用するための設定に、プロファイルを指定します。
%region	文字列	[AWS リージョン] を指定します。; この中でセッションを初期化します。~/.aws/configure. からのデフォルト設定 例：%region us-west-1
%idle_timeout	Int	セルが実行されてからセッションがタイムアウトするまでの非アクティブ時間 (分)。Spark ETL セッションのデフォルトのアイドルタイムアウト値は、デフォルトのタイムアウトである 2,880分 (48時間) です。他のセッションタイプについては、該当するセッションタイプのマニュアルを参照してください。 例：%idle_timeout 3000
%session_id	該当なし	実行中のセッションのセッション ID をリターンします。
%session_id_prefix	文字列	[session_id_prefix]-[session_id] の形式ですべてのセッション ID に先行する文字列を

名前	型	説明
		定義します。セッション ID を指定しないと、ランダムな UUID が生成されます。AWS Glue Studio で Jupyter Notebook を実行する場合に、このマジックはサポートされていません。 例：`%session_id_prefix 001`
`%status`		現在の AWS Glue セッションのステータス (期間、構成、実行中のユーザー/ロールなど) を返します。
`%stop_session`		現在のセッションを停止します。
`%list_sessions`		名前と ID で、現在実行中のすべてのセッションをリストします。
`%session_type`	文字列	セッションタイプをストリーミング、ETL、または Ray のいずれかに設定します。 例：`%session_type Streaming`
`%glue_version`	文字列	このセッションで使用される AWS Glue のバージョン。 例：`%glue_version 3.0`

ジョブタイプを選択するためのマジック

名前	型	説明
`%streaming`	文字列	セッションタイプを AWS Glue ストリーミングに変更します。
`%etl`	文字列	セッションタイプを AWS Glue ETL に変更します。

名前	型	説明
%glue_ray	文字列	セッションタイプを AWS Glue for Ray に変更します。「Magics supported by AWS Glue Ray interactive sessions」を参照してください。

AWS Glue for Spark 設定マジック

%%configure マジックは、セッションのすべての構成パラメータで構成される JSON 形式のディクショナリです。各パラメータは、ここで指定することも、個々のマジックを使って指定することもできます。

名前	型	説明
%%configure	辞書	セッションのすべての構成パラメータで構成される JSON 形式のディクショナリを指定します。各パラメータは、ここで指定することも、個々のマジックを使って指定することもできます。 パラメータのリストと %%configure の使用方法の例については、「%%configure セルマジック引数」を参照してください。
%iam_role	文字列	セッションを実行する IAM ロール ARN を指定します。~/.aws/configure からのデフォルト設定。 例：%iam_role AWSGlueServiceRole
%number_of_workers	Int	ジョブの実行時に割り当てられる、定義された worker_type のワーカーの数。worker_type も設定する必要があります。デフォルト number_of_workers は 5 です。

名前	型	説明
		例：%number_of_workers 2
%additional_python_modules	リスト	クラスターに含める追加の Python モジュールのカンマ区切りリスト (PyPI または S3 からでも可)。 例えば、%additional_python_modules pandas, numpy などです。

名前	型	説明

%%tags	文字列	セッションにタグを追加します。タグを中かっこ {} で囲んで指定します。各タグ名のペアは二重引用符 (" ") で囲まれ、カンマ (,) で区切られます。
		
		`%%tags`
		`{"billing":"Data-Platform",`
		`"team":"analytics"}`
		
		%status マジックを使用して、セッションに関連付けられたタグを表示します。
		
		`%status`
		
		`Session ID: <sessionId>`
		`Status: READY`
		`Role: <example-role>`
		`CreatedOn: 2023-05-26 11:12:17.`
		`056000-07:00`
		`GlueVersion: 3.0`
		`Job Type: glueetl`
		`Tags: {'owner':'example-owner',`
		`'team':'analytics', 'billing'`
		`:'Data-Platform'}`
		`Worker Type: G.4X`
		`Number of Workers: 5`
		`Region: us-west-2`
		`Applying the following default`
		`arguments:`
		`--glue_kernel_version 0.38.0`
		`--enable-glue-datacatalog true`
		`Arguments Passed: ['--glue_`
		`kernel_version: 0.38.0', '--`
		`enable-glue-datacatalog: true']`

名前	型	説明

%%assume_role	辞書	JSON 形式のディクショナリまたは IAM ロールの ARN 文字列を指定して、クロスアカウントアクセス用のセッションを作成します。
		
		ARN を使用した例:
		
		`%%assume_role`
		`{`
		`  'arn:aws:iam::XXXXXXXXXXXX:`
		`role/AWSGlueServiceRole'`
		`}`
		
		認証情報の例:
		
		`%%assume_role`
		`{{`
		`  "aws_access_key_id" =`
		`"XXXXXXXXXXXX",`
		`  "aws_secret_access_key" =`
		`"XXXXXXXXXXXX",`
		`  "aws_session_token" =`
		`"XXXXXXXXXXXX"`
		`}}`

%%configure セルマジック引数

%%configure マジックは、セッションのすべての構成パラメータで構成される JSON 形式のディクショナリです。各パラメータは、ここで指定することも、個々のマジックを使って指定することもできます。%%configure セルマジックがサポートする引数の例については、以下を参照してください。ジョブに指定された実行引数に -- プレフィックスを使用します。例 :

```
%%configure
{
   "--user-jars-first": "true",
   "--enable-glue-datacatalog": "false"
```

```
}
```

ジョブパラメータの詳細については、「ジョブのパラメータ」を参照してください。

セッション設定

パラメータ	Type	説明

`max_retries`	Int	失敗した場合にこのジョブを再試行する最大回数。
		`%%configure`
		`{`
		`  "max_retries": "0"`
		`}`

`max_concurrent_runs`	Int	ジョブで許可される同時実行の最大数。
		例：
		`%%configure`
		`{`
		`  "max_concurrent_runs": "3"`
		`}`

セッションパラメータ

パラメータ	Type	説明

`--enable-spark-ui`	ブール値	Spark UI を有効にして AWS Glue ETL ジョブのモニタリングとデバッグを行います。
		`%%configure`
		`{`
		`  "--enable-spark-ui": "true"`

パラメータ	Type	説明
		})

--spark-event-logs-path	文字列	Amazon S3 パスを指定します。Spark UI のモニタリング機能を使用する場合。
		
		例：
		
		%%configure
		{
		"--spark-event-logs-path":
		"s3://path/to/event/logs/"
		}

--script_location	文字列	ジョブを実行するスクリプトへの S3 パスを指定します。
		
		例：
		
		%%configure
		{
		"script_location": "s3://new-folder-here"
		}

パラメータ	Type	説明

`--SECURITY_CONFIGURATION`	文字列	AWS Glue セキュリティ設定の名前
		
		例：
		
		`%%configure`
		`{`
		`"--security_configuration": { "encryption_type": "kms", "kms_key_id": "YOUR_KMS_KEY_ARN" }`
		`}`

`--job-language`	文字列	スクリプトプログラミング言語。'scala' または 'python' の値を使用できます。デフォルトは 'python' です。
		
		例：
		
		`%%configure`
		`{`
		`"--job-language": "scala"`
		`}`

`--class`	文字列	Scala スクリプトのエントリポイントとなる Scala クラス。デフォルトは null です。
		
		例：
		
		`%%configure`
		`{`
		`"--class": "className"`
		`}`

パラメータ	Type	説明

`--user-jars-first`	ブール値	クラスパスにあるお客様のその他の JAR ファイルに優先順位を付けます。デフォルトは null です。
		
		例：
		
		`%%configure`
		`{`
		`  "--user-jars-first": "true"`
		`}`

`--use-postgres-driver`	ブール値	Amazon Redshift JDBC ドライバーとの競合を避けるために、クラスパスにある Postgres JDBC ドライバーが優先されます。デフォルトは null です。
		
		例：
		
		`%%configure`
		`{`
		`  "--use-postgres-driver": "true"`
		`}`

パラメータ	Type	説明

`--extra-files`	List(string)	スクリプトを実行する前に、AWS Glue がスクリプトの作業ディレクトリにコピーする設定ファイルなどの追加ファイルを指す Amazon S3 のパス。
		
		例：
		
		`%%configure`
		`{`
		`  "--extra-files": "s3://path/to/additional/files/"`
		`}`

`--job-bookmark-option`	文字列	ジョブブックマークの動作を制御します。'job-bookmark-enable'、'job-bookmark-disable' または 'job-bookmark-pause' の値を使用できます。デフォルトは 'job-bookmark-disable' です。
		
		例：
		
		`%%configure`
		`{`
		`  "--job-bookmark-option": "job-bookmark-enable"`
		`}`

パラメータ	Type	説明

`--TempDir`	文字列	ジョブの一時ディレクトリとして使用できるバケットへの Amazon S3 のパスを指定します。デフォルトは null です。
		
		例：
		
		`%%configure`
		`{`
		`  "--TempDir": "s3://path/to/temp/dir"`
		`}`

`--enable-s3-parquet-optimized-committer`	ブール値	Parquet データを Amazon S3 に書き込むために、EMRFS Amazon S3 最適化コミッターを有効にします。デフォルトは 'true' です。
		
		例：
		
		`%%configure`
		`{`
		`  "--enable-s3-parquet-optimized-committer": "false"`
		`}`

パラメータ	Type	説明

`--enable-rename-algorithm-v2`	ブール値	EMRFS 名前変更アルゴリズムのバージョンをバージョン 2 に設定します。デフォルトは 'true' です。
		
		例：
		
		`%%configure`
		`{`
		`  "--enable-rename-algorithm-v2": "true"`
		`}`

`--enable-glue-datacatalog`	ブール値	AWS Glue データカタログの Apache Spark Hive メタストアとしての使用を有効にします。
		
		例：
		
		`%%configure`
		`{`
		`  "--enable-glue-datacatalog": "true"`
		`}`

`--enable-metrics`	ブール値	ジョブの実行のジョブプロファイリングに関するメトリクスの収集を有効にします。デフォルトは 'false' です。
		
		例：
		
		`%%configure`
		`{`
		`  "--enable-metrics": "true"`
		`}`

パラメータ	Type	説明

`--enable-continuous-cloudwatch-log`	ブール値	AWS Glue ジョブのリアルタイムの連続ログ記録を有効にします。デフォルトは 'false' です。
		
		例：
		
		`%%configure`
		`{`
		`  "--enable-continuous-cloudwatch-log": "true"`
		`}`

`--enable-continuous-log-filter`	ブール値	連続ログ記録が有効であるジョブを作成または編集するときに、標準フィルタまたはフィルタなしを指定します。デフォルトは 'true' です。
		
		例：
		
		`%%configure`
		`{`
		`  "--enable-continuous-log-filter": "true"`
		`}`

パラメータ	Type	説明

`--continuous-log-stream-prefix`	文字列	連続ログ記録が有効なジョブのカスタム Amazon CloudWatch ログストリームプレフィックスを指定します。デフォルトは null です。
		例：
		`%%configure`
		`{`
		`  "--continuous-log-stream-prefix": "prefix"`
		`}`

`--continuous-log-conversionPattern`	文字列	連続ログ記録を有効にしたジョブのカスタム変換ログパターンを指定します。デフォルトは null です。
		例：
		`%%configure`
		`{`
		`  "--continuous-log-conversionPattern": "pattern"`
		`}`

パラメータ	Type	説明

--conf	文字列	Spark の設定パラメータを制御します。高度なユースケース向けです。各パラメータの前に --conf を使用してください。例：
		
		%%configure
		{
		"--conf": "spark.hadoop.hive
		.metastore.glue.catalogid=1
		23456789012 --conf hive.meta
		store.client.factory.class=
		com.amazonaws.glue.catalog.
		metastore.AWSGlueDataCatalo
		gHiveClientFactory --conf
		hive.metastore.schema.verif
		ication=false"
		}

timeout	Int	Spark セッションが終了する前にステートメントの完了を待機する最大時間を決定します。
		
		%%configure
		{
		"timeout": "30"
		}

自動スケーリング	ブール値	自動スケーリングを使用するかどうかを決定します。
		
		%%configure
		{
		"--enable-auto-scaling": "true"
		}

Spark ジョブ (ETL とストリーミング) のマジック

名前	型	説明
`%worker_type`	文字列	Standard、G.1X、または G.2X を指定します。`number_of_workers` も設定する必要があります。デフォルトの worker_type は G.1X です。

`%connections`	リスト	セッションで使用する接続のカンマ区切りリストを指定します。
		
		例：
		
		`%connections my_rds_connection`
		`dy_f =`
		`glue_context.create_dynamic`
		`_frame.from_catalog(databas`
		`e='rds_tables', table_nam`
		`e='sales_table')`

`%extra_py_files`	リスト	Amazon S3 からの追加の Python ファイルのカンマ区切りリスト。
`%extra_jars`	リスト	クラスターに含める追加の jar のカンマ区切りリスト
`%spark_conf`	文字列	セッション用のカスタム Spark 設定を指定します。例えば、`%spark_conf spark.serializer=org.apache.spark.serializer.KryoSerializer` と指定します。

Ray ジョブのマジック

名前	型	説明
`%min_workers`	Int	Ray ジョブに割り当てられるワーカーの最小数。デフォルト: 1。 例：`%min_workers 2`
`%object_memory_head`	Int	ウォームスタート後のインスタンスヘッドノードの空きメモリの割合。最小値: 0。最大値: 100。 例：`%object_memory_head 100`
%object_memory_worker	Int	ウォームスタート後のインスタンスワーカーノードの空きメモリの割合。最小値: 0。最大値: 100。 例：`%object_memory_worker 100`

Action Magics

名前	型	説明
`%%sql`	文字列	SQL コードを実行します。最初の `%%sql` マジックの後のすべての行は、SQL コードの一部として渡されます。 例：`%%sql select * from rds_tables.sales_table`
`%matplot`	Matplotlib Figure	Matplotlib ライブラリを使用してデータを可視化します。 例： `import matplotlib.pyplot as plt`

名前	型	説明

		``
		# Set X-axis and Y-axis values
		x = [5, 2, 8, 4, 9]
		y = [10, 4, 8, 5, 2]
		
		# Create a bar chart
		plt.bar(x, y)
		
		# Show the plot
		%matplot plt
		``

%plotly	Plotly Figure	Plotly ライブラリを使用してデータを可視化します。
		
		例：
		
		``
		import plotly.express as px
		
		#Create a graphical figure
		fig = px.line(x=["a","b","c"],
		 y=[1,3,2], title="sample
		 figure")
		
		#Show the figure
		%plotly fig
		``

ネーミングセッション

AWS Glue インタラクティブセッションは、AWS リソースであり、名前が必要です。名前はセッションごとに固有である必要があり、IAM 管理者によって制限される場合があります。詳細については、「IAM を使用したインタラクティブセッション」を参照してください。Jupyter カーネルは固有のセッション名を自動的に生成します。ただし、セッション名は、次の 2 つの方法で手動で名前を付けることができます。

1. ~.aws/config にある AWS Command Line Interface 設定ファイルを使用する。「AWS Command Line Interface による AWS の設定」を参照してください。

2. `%session_id_prefix` マジックを使用する。「Jupyter 向け AWS Glue インタラクティブセッションでサポートされているマジック」を参照してください。

セッション名は、次のように生成されます。

- プレフィックスと session_id が指定されている場合、セッション名は {prefix}-{UUID} になります。
- 何も指定されていない場合、セッション名は {UUID} になります。

プレフィックスにセッション名を使用すると、AWS CLI またはコンソールにセッションを一覧表示した際にセッションを認識できます。

インタラクティブセッションの IAM ロールの指定

インタラクティブセッションを実行する AWS ETL コードを使って、AWS Glue Identity and Access Management (IAM)ロールを指定する必要があります。

ロールには、AWS Glue ジョブ実行に必要なアクセス許可と同じ IAM アクセス許可が付与されている必要があります。AWS Glue ジョブおよびインタラクティブセッションのロール作成の詳細については、「AWS Glue 用にIAM ロールを作成する」を参照してください。

IAM ロールは、次の 2 つの方法で指定できます。

- `~.aws/config` にある AWS Command Line Interface 設定ファイルを使用する (推奨)。詳細については、「~/.aws/config でのセッションの設定」を参照してください。

> **Note**
>
> `%profile` マジックを使用すると、そのプロファイルの `glue_iam_role` の設定が優先されます。

- %iam_role マジックを使用する。(詳しくは、「Jupyter 向け AWS Glue インタラクティブセッションでサポートされているマジック」を参照してください。)

名前付きプロファイルを使用したセッションの設定

AWS Glue インタラクティブセッションでは、AWS Command Line Interface または boto3 と同じ認証情報が使用されます。インタラクティブセッションでは、`~/.aws/config` (Linux および

MacOS) または %USERPROFILE%\.aws\config (Windows) にある AWS CLI などの名前付きプロファイルと優先的に連携します。詳細については、「名前を指定されたプロファイルを使用する」を参照してください。

AWS Glue サービスロールとセッション ID プレフィックスをプロファイルで指定できるようにすることで、インタラクティブセッションが名前付きプロファイルを利用します。プロファイルロールを設定するには、以下に示すように、名前付きプロファイルに iam_role キーまたは session_id_prefix 、あるいはその両方の行を追加します。session_id_prefix は引用符を必要としません。例えば、 session_id_prefix を追加する場合は、session_id_prefix=myprefix の値を入力します。

```
[default]
region=us-east-1
aws_access_key_id=AKIAIOSFODNN7EXAMPLE
aws_secret_access_key=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
glue_iam_role=arn:aws:iam::<AccountID>:role/<GlueServiceRole>
session_id_prefix=<prefix_for_session_names>

[user1]
region=eu-west-1
aws_access_key_id=AKIAI44QH8DHBEXAMPLE
aws_secret_access_key=je7MtGbClwBF/2Zp9Utk/h3yCo8nvbEXAMPLEKEY
glue_iam_role=arn:aws:iam::<AccountID>:role/<GlueServiceRoleUser1>
session_id_prefix=<prefix_for_session_names_for_user1>
```

認証情報を生成するカスタムメソッドがある場合は、~/.aws/config ファイルの credential_process パラメーターを使用するようにプロファイルを設定することもできます。例:

```
[profile developer]
region=us-east-1
credential_process = "/Users/Dave/generate_my_credentials.sh" --username helen
```

credential_process パラメータを使用した認証情報の調達の詳細については、「外部プロセスを使用した認証情報の調達」を参照してください。

使用しているプロファイルにリージョンまたは iam_role が設定されていない場合は、最初に実行するセルの %region と %iam_role マジックを使用して設定する必要があります。

スクリプトまたはノートブックの AWS Glue ジョブジョブへの変換

スクリプトまたはノートブックを AWS Glue ジョブに変換する方法は2つあります。

- nbconvertを使用して、Jupyter `.ipynb` Notebook ドキュメントファイルを `.py` ファイルに変換します。詳細については、「nbconvert: Convert Notebooks to other formats」(nbconvert：ノートブックを他の形式に変換する) を参照してください。
- ファイルを AWS Glue Studio ノートブックにアップロードします。
  - AWS Glue Studio コンソールで、ナビゲーションメニューから ジョブを選択します。
  - ジョブの作成セクションで、Jupyter Notebookを選択します。
  - オプションセクションで、[Upload] (アップロード) を選択し、既存のノートブックを編集します。
  - Choose file (ファイルの選択)を選択して、`.ipynb` ファイルをアップロードします。

AWS Glue インタラクティブセッションでのストリーミングオペレーションの使用

ストリーミングセッションタイプの切り替え

AWS Glue インタラクティブセッション設定マジックである `%streaming` を使用して、実行しているジョブを定義し、ストリーミングインタラクティブセッションを初期化します。

インタラクティブな開発のためのサンプリング入力ストリーム

AWS Glue インタラクティブセッションで対話型のエクスペリエンスを強化するために導入されたツール機能の 1 つは、静的な DynamicFrame でストリームのスナップショットを取得するために、`GlueContext` の下に新しいメソッドを追加する機能です。`GlueContext` を使用することで、ワークフローを検査、操作、実装できます。

`GlueContext` クラスインスタンスを使用して、メソッド `getSampleStreamingDynamicFrame` を見つけることができます。このメソッドに必要な引数は以下のとおりです。

- `dataFrame`: Spark ストリーミングデータフレーム
- `options`: 以下の利用可能なオプションを参照してください。

以下のオプションを使用できます。

- windowSize: これは Microbatch Duration とも呼ばれます。このパラメータは、前回のバッチがトリガーされてからストリーミングクエリが待機する時間を特定します。このパラメータの値は、pollingTimeInMs より小さくしてください。
- pollingTimeInM: このメソッドが実行される合計時間です。入力ストリームからサンプルレコードを取得するために、少なくとも 1 つのマイクロバッチが起動されます。
- recordPollingLimit: このパラメータを使用すると、ストリームからポーリングするレコードの総数を制限できます。
- (オプション) writeStreamFunction を使用して、このカスタム機能をすべてのレコードサンプリング機能に適用することもできます。以下は、Scala と Python のサンプルコードです。

Scala

```
val sampleBatchFunction = (batchDF: DataFrame, batchId: Long) => {//Optional but
 you can replace your own forEachBatch function here}
val jsonString: String = s"""{"pollingTimeInMs": "10000", "windowSize": "5
 seconds"}"""
val dynFrame = glueContext.getSampleStreamingDynamicFrame(YOUR_STREAMING_DF,
 JsonOptions(jsonString), sampleBatchFunction)
dynFrame.show()
```

Python

```
def sample_batch_function(batch_df, batch_id):
       //Optional but you can replace your own forEachBatch function here
options = {
            "pollingTimeInMs": "10000",
            "windowSize": "5 seconds",
        }
glue_context.getSampleStreamingDynamicFrame(YOUR_STREAMING_DF, options,
 sample_batch_function)
```

> **Note**
>
> サンプリングされた DynFrame が空の場合、いくつかの原因が考えられます。
>
> - ストリーミングソースが「Latest」(最新) に設定されており、サンプリング期間中に新しいデータは取り込まれていません。
> - ポーリング時間が十分ではなく、取り込まれたレコードを処理できません。バッチ全体が処理されない限り、データは表示されません。

インタラクティブセッションでのストリーミングアプリケーションの実行

AWS Glue インタラクティブセッションでは、AWS Glue コンソールでストリーミングアプリケーションを作成するのと同じように、AWS Glue ストリーミングアプリケーションを実行できます。インタラクティブセッションはセッションベースのため、ランタイムで例外が発生してもセッションが停止することはありません。バッチ関数を反復的に開発するというメリットが追加されました。例:

```
def batch_function(data_frame, batch_id):
    log.info(data_frame.count())
    invalid_method_call()
glueContext.forEachBatch(frame=streaming_df, batch_function = batch_function, options =
 {**})
```

上記の例にはメソッドの無効な使用が含まれており、アプリケーション全体が終了する通常の AWS Glue ジョブとは異なります。ユーザーのコーディングコンテキストと定義は完全に保持され、セッションは引き続き動作可能な状態になっています。新しいクラスターをブートストラップして、先行するすべての変換を再実行する必要はありません。これにより、バッチ関数の実装をすばやくに反復でき、望ましい結果を得ることができます。

インタラクティブセッションは、セッションが一度に 1 つのステートメントだけを実行するように、各ステートメントをブロックする方法で評価することに注意してください。ストリーミングクエリは継続的で終了することがないため、アクティブなストリーミングクエリを含むセッションは、中断されない限り後続のステートメントを処理できません。Jupyter Notebook から直接中断のコマンドを発行すると、カーネルがキャンセルを処理します。

例として、実行を待機するシーケンシャルなステートメントを考えてみましょう。

```
Statement 1:
```

```
      val number = df.count()
      #Spark Action with deterministic result
      Result: 5

Statement 2:
      streamingQuery.start().awaitTermination()
      #Spark Streaming Query that will be executing continously
      Result: Constantly updated with each microbatch

Statement 3:
      val number2 = df.count()
      #This will not be executed as previous statement will be running indefinitely
```

AWS Glue ジョブスクリプトのローカルでの開発およびテスト

AWS Glue for Spark ジョブスクリプトを開発およびテストする場合、以下のような複数のオプションが用意されています。

- AWS Glue Studio コンソール
  - Visual editor (ビジュアルエディタ)
  - スクリプトエディタ
  - AWS Glue Studio ノートブック
- インタラクティブなセッション
  - Jupyter Notebook
- Docker イメージ
  - ローカル開発
  - リモート開発

ユーザーは、要件に基づいて、上記のオプションのいずれかを選択できます。

コードを記述したくない、あるいは記述する量を減らしたい場合は、AWS Glue Studio のビジュアルエディタが便利です。

ノートブックのインタラクティブな操作を希望する場合は、AWS Glue Studio ノートブックが適しています。詳細については、「AWS Glue Studio および AWS Glue ノートブックによるコードの作成」を参照してください。独自のローカル環境を使用するユーザーには、インタラクティブセッショ

ンが適しています。詳細については、「インタラクティブセッションで AWS Glue ジョブを構築する」を参照してください。

開発をローカル/リモートの両方で行いたい場合は、Docker イメージが便利です。これにより、Spark ジョブスクリプト用の AWS Glue の開発とテストを好きな場所で行うことができ、AWS Glue のコストもかかりません。

Docker を使用せずにローカルで開発したい場合は、AWS Glue ETL ライブラリディレクトリをローカルにインストールします。

AWS Glue Studio を使用した開発

AWS Glue Studio のビジュアルエディタは、AWS Glue での抽出、変換、ロード (ETL) ジョブの作成、実行、およびモニタリングを簡単に行うことができるグラフィカルなインターフェイスです。データ変換ワークフローを視覚的に作成し、それらを AWS Glue の Apache Spark ベースのサーバーレス ETL エンジン上でシームレスに実行することができます。ジョブの各ステップでスキーマとデータの結果を調べることができます。詳細については、「AWS Glue Studio ユーザーガイド」を参照してください。

インタラクティブセッションによる開発

インタラクティブセッションでは、お好みの環境からアプリケーションを構築し、そのテストが行えます。詳細については、「インタラクティブセッションで AWS Glue ジョブを構築する」を参照してください。

Docker イメージを使用してローカルで AWS Glue ジョブを開発およびテストする

本番環境対応のデータプラットフォームの場合、AWS Glue ジョブ向けの開発プロセスや CI/CD パイプラインで検討すべきです。Docker コンテナでは、AWS Glue のジョブの開発やテストが柔軟に行えます。AWS Glue は Docker Hub に Docker イメージをホストすることで、追加のユーティリティを使用する開発環境をセットアップします。IDE、ノートブック、または、AWS Glue ETL ライブラリを使用する REPL などを、ご自分で選択して使用できます。このトピックでは、Docker コンテナ上で Docker イメージを使用して AWS Glue バージョン 5.0 のジョブを開発およびテストする方法について説明します。

利用可能な Docker イメージ

Amazon ECR の AWS Glue では、以下の Docker イメージが利用可能です。

- AWS Glue バージョン 5.0 の場合: `public.ecr.aws/glue/aws-glue-libs:5`
- AWS Glue バージョン 4.0 の場合: `public.ecr.aws/glue/aws-glue-libs:glue_libs_4.0.0_image_01`
- AWS Glue バージョン 3.0 の場合: `public.ecr.aws/glue/aws-glue-libs:glue_libs_3.0.0_image_01`
- AWS Glue バージョン 2.0 の場合: `public.ecr.aws/glue/aws-glue-libs:glue_libs_2.0.0_image_01`

> **Note**
>
> AWS Glue Docker イメージは x86_64 と arm64 の両方と互換性があります。

この例では、ローカルマシン (Mac/Windows/Linux) での `public.ecr.aws/glue/aws-glue-libs:5` の使用法とコンテナを実行するための方法を示します。このコンテナイメージは、AWS Glue バージョン 5.0 の Spark ジョブでテスト済みです。このイメージには、以下が含まれます。

- Amazon Linux 2023
- AWS Glue ETL ライブラリ
- Apache Spark 3.5.4
- オープンテーブル形式のライブラリ; Apache Iceberg 1.7.1、Apache Hudi 0.15.0、Delta Lake 3.3.0
- AWS Glue データカタログクライアント
- Amazon Redshift Apache Spark 用 コネクタ
- Amazon DynamoDB Apache Hadoop 用 コネクタ

コンテナを設定するには、ECR Public Gallery からイメージを取得して、コンテナを実行します。このトピックでは、要件に応じて以下の各方法でコンテナを実行する方法を示します。

- `spark-submit`
- REPL シェル (`pyspark`)
- `pytest`
- Visual Studio Code

前提条件

作業を開始する前に、Docker がインストール済みであり、Docker デーモンが実行中であることを確認します。インストールの手順については、Mac または Linux 向けの Docker ドキュメントを参照してください。AWS Glue コンテナは、Docker を実行しているマシンでホストされます。また、Docker を実行しているホスト上のイメージ用として、少なくとも 7 GB のディスク領域が必要です。

AWS Glue のコードをローカルで開発する際の制限については、「ローカル開発の制限」を参照してください。

AWS の設定

コンテナから AWS API コールを有効にするには、以下の手順に従って AWS 認証情報をセットアップします。以下のセクションでは、この AWS により名前が指定されたプロファイルを使用します。

1. AWS の名前付きプロファイルを作成します。
2. Windows または Mac/Linux のターミナルで cmd を開き、ターミナル上で次のコマンドを実行します。

```
PROFILE_NAME="<your_profile_name>"
```

以下のセクションでは、この AWS の名前付きプロファイルを使用します。

Windows で Docker を実行している場合は、Docker アイコンを右クリックし、イメージを取得する前に [Switch to Linux containers] (Linux コンテナに切り替える) を選択します。

ECR Public からイメージを取得するには、次のコマンドを実行します。

```
docker pull public.ecr.aws/glue/aws-glue-libs:5
```

コンテナを実行する

ここまでの操作により、取得したイメージを使用してコンテナを実行できるようになります。ご自身の要件に応じて、以下のいずれかを選択できます。

spark-submit

コンテナで `spark-submit` コマンドを使用して、AWS Glue のジョブスクリプトを実行できます。

1. 以下の例のようにスクリプトを記述して sample.py として保存し、さらに以下のコマンドを使用して /local_path_to_workspace/src/ ディレクトリに保存します。

```
$ WORKSPACE_LOCATION=/local_path_to_workspace
$ SCRIPT_FILE_NAME=sample.py
$ mkdir -p ${WORKSPACE_LOCATION}/src
$ vim ${WORKSPACE_LOCATION}/src/${SCRIPT_FILE_NAME}
```

2. これらの変数は、以下の Docker 実行コマンドで使用されます。以下の spark-submit コマンドで使用されるサンプルコード (sample.py) は、このトピックの最後にある付録に含まれています。

   以下のコマンドにより、コンテナで spark-submit コマンドを使用して、新しい Spark アプリケーションをサブミットします。

```
$ docker run -it --rm \
    -v ~/.aws:/home
    /hadoop/.aws \
    -v $WORKSPACE_LOCATION:/home/hadoop/workspace/ \
    -e AWS_PROFILE=$PROFILE_NAME \
    --name glue5_spark_submit \
    public.ecr.aws/glue/aws-glue-libs:5 \
    spark-submit /home/hadoop/workspace/src/$SCRIPT_FILE_NAME
```

3. (オプション) 環境に合わせて spark-submit を構成します。たとえば、--jars 設定を使用して依存関係を渡すことができます。詳細については、Spark ドキュメントの「Dynamically Loading Spark Properties (Spark プロパティの動的ロード)」を参照してください。

REPL シェル (Pyspark)

REPL (read-eval-print loops) シェルを実行すると、インタラクティブな開発を行うことができます。次のコマンドにより、コンテナ上で PySpark コマンドを実行し、REPL シェルを起動します。

```
$ docker run -it --rm \
    -v ~/.aws:/home/hadoop/.aws \
    -e AWS_PROFILE=$PROFILE_NAME \
    --name glue5_pyspark \
    public.ecr.aws/glue/aws-glue-libs:5 \
    pyspark
```

この場合、次の出力が返されます。

```
Python 3.11.6 (main, Jan  9 2025, 00:00:00) [GCC 11.4.1 20230605 (Red Hat 11.4.1-2)] on
 linux
Type "help", "copyright", "credits" or "license" for more information.
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use
 setLogLevel(newLevel).
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 3.5.4-amzn-0
      /_/

Using Python version 3.11.6 (main, Jan  9 2025 00:00:00)
Spark context Web UI available at None
Spark context available as 'sc' (master = local[*], app id = local-1740643079929).
SparkSession available as 'spark'.
>>>
```

この REPL シェルを使用することで、インタラクティブなコーディングとテストを実行できます。

Pytest

ユニットテストの場合、AWS Glue の Spark ジョブスクリプトで pytest を使用できます。以下のコマンドを実行し、準備を行います。

```
$ WORKSPACE_LOCATION=/local_path_to_workspace
$ SCRIPT_FILE_NAME=sample.py
$ UNIT_TEST_FILE_NAME=test_sample.py
$ mkdir -p ${WORKSPACE_LOCATION}/tests
$ vim ${WORKSPACE_LOCATION}/tests/${UNIT_TEST_FILE_NAME}
```

以下のコマンドの実行により、docker run を使用して pytest を実行します。

```
$ docker run -i --rm \
    -v ~/.aws:/home/hadoop/.aws \
    -v $WORKSPACE_LOCATION:/home/hadoop/workspace/ \
    --workdir /home/hadoop/workspace \
    -e AWS_PROFILE=$PROFILE_NAME \
    --name glue5_pytest \
    public.ecr.aws/glue/aws-glue-libs:5 \
    -c "python3 -m pytest --disable-warnings"
```

pytest がユニットテストの実行を完了すると、出力は次のようになります。

```
============================= test session starts ==============================
platform linux -- Python 3.11.6, pytest-8.3.4, pluggy-1.5.0
rootdir: /home/hadoop/workspace
plugins: integration-mark-0.2.0
collected 1 item

tests/test_sample.py .                                                   [100%]

======================== 1 passed, 1 warning in 34.28s =========================
```

Visual Studio Code を使用するためのコンテナのセットアップ

Visual Studio Code でコンテナを設定するには、次の手順を実行します。

1. Visual Studio Code をインストールします。
2. Python をインストールします。
3. Visual Studio Code Remote – コンテナをインストールします。
4. Visual Studio Code でワークスペースフォルダを開きます。
5. Ctrl+Shift+P (Windows/Linux) または Cmd+Shift+P (Mac) を押します。
6. タイプ Preferences: Open Workspace Settings (JSON)。
7. [Enter] キーを押します。
8. 次の JSON を貼り付け、保存します。

```
{
    "python.defaultInterpreterPath": "/usr/bin/python3.11",
    "python.analysis.extraPaths": [
        "/usr/lib/spark/python/lib/py4j-0.10.9.7-src.zip:/usr/lib/spark/python/:/usr/
lib/spark/python/lib/",
    ]
}
```

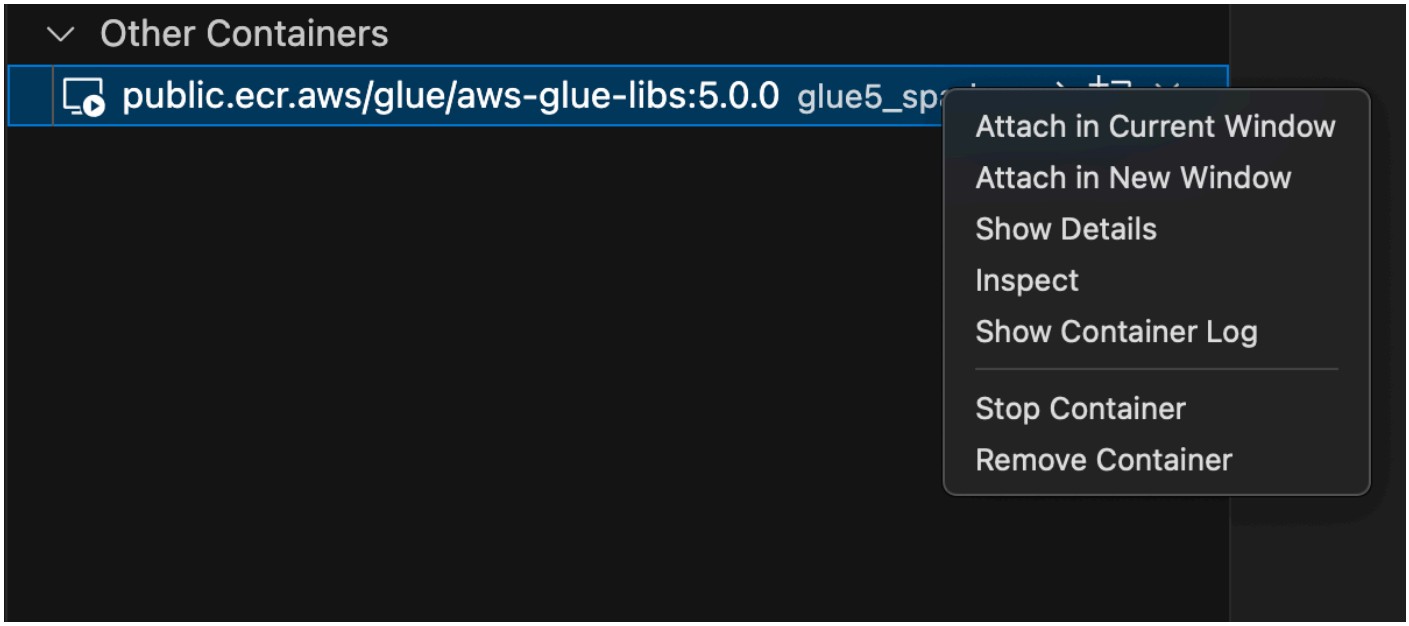
コンテナを設定するには:

1. Docker コンテナを実行します。

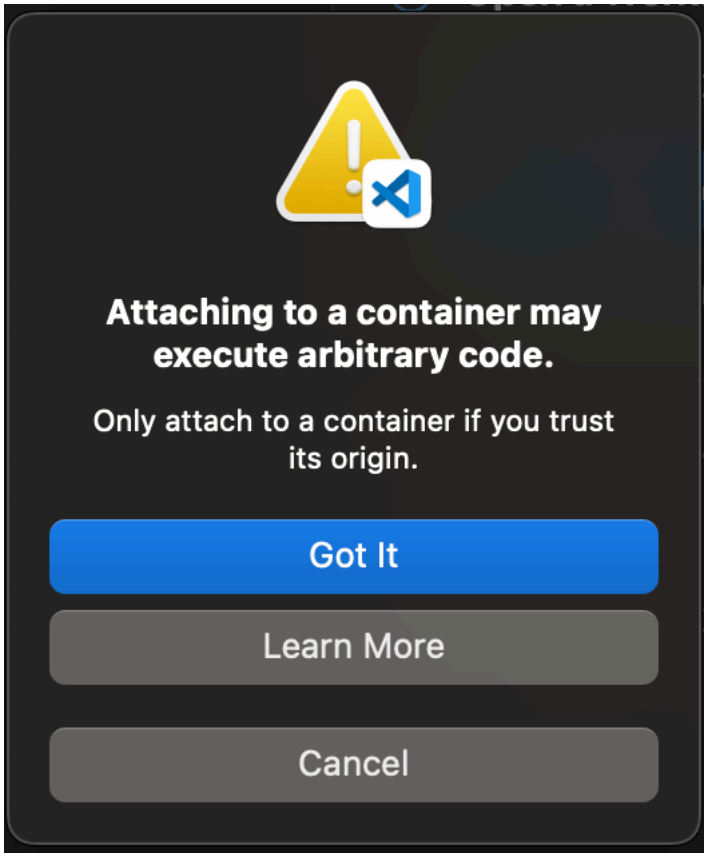
```
$ docker run -it --rm \
```

```
    -v ~/.aws:/home/hadoop/.aws \
    -v $WORKSPACE_LOCATION:/home/hadoop/workspace/ \
    -e AWS_PROFILE=$PROFILE_NAME \
    --name glue5_pyspark \
    public.ecr.aws/glue/aws-glue-libs:5 \
    pyspark
```

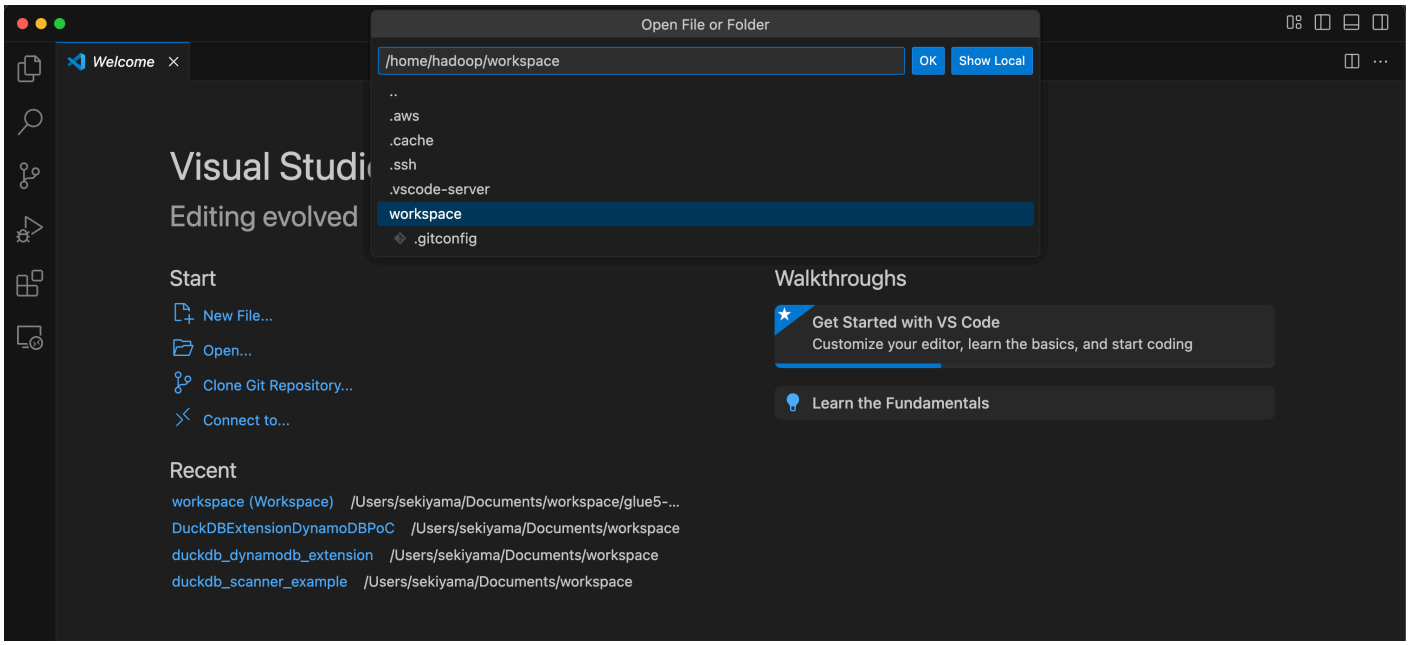
2. Visual Studio Code を起動します。
3. 左側のメニューで [Remote Explorer] (リモートエクスプローラー) を選択した上で、amazon/aws-glue-libs:glue_libs_4.0.0_image_01 を選択します。
4. 右クリックし、[Attach in Current Window] (現在のウィンドウにアタッチする) を選択します。



5. 次のダイアログが表示されたら、[Got it] (取得しました) を選択します。

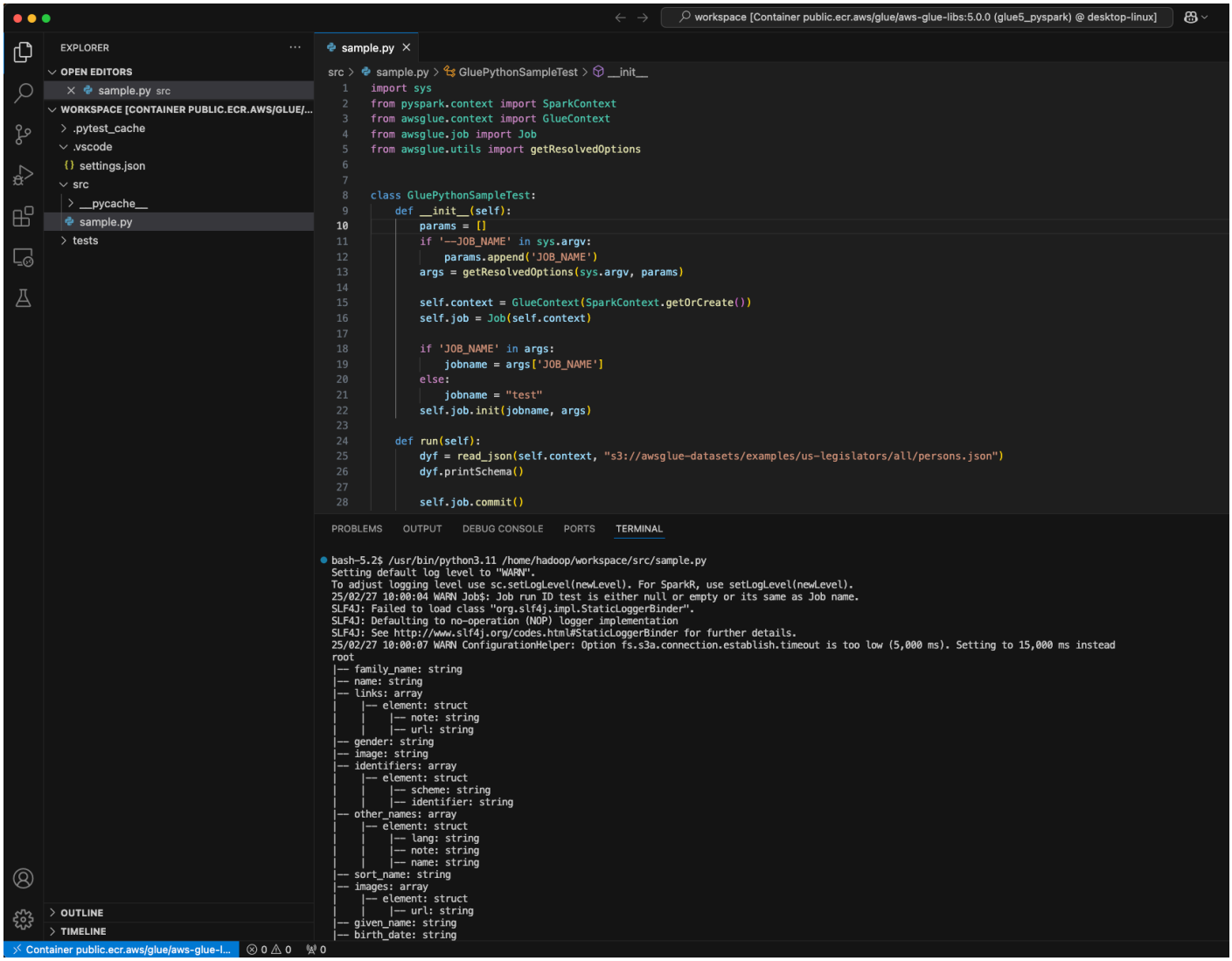


6. /home/handoop/workspace/ を開きます。



7. AWS Glue PySpark スクリプトを作成し、[Run] (実行) を選択します。

   このスクリプトは正常に実行されたことが表示されます。



AWS Glue 4.0 と AWS Glue 5.0 の Docker イメージの変更点

AWS Glue 4.0 と AWS Glue 5.0 の Docker イメージの主な変更点:

- AWS Glue 5.0 では、バッチジョブとストリーミングジョブの両方に 1 つのコンテナイメージがあります。これは、バッチ用とストリーミング用にそれぞれ別のイメージがあった Glue 4.0 とは異なります。
- AWS Glue 5.0 では、コンテナのデフォルトのユーザー名は hadoop です。AWS Glue 4.0 では、デフォルトのユーザー名は `glue_user` でした。
- AWS Glue 5.0 では、JupyterLab や Livy など、いくつかの追加ライブラリがイメージから削除されています。これらのライブラリは手動でインストールできます。

- AWS Glue 5.0 では、Iceberg、Hudi、Delta のすべてのライブラリがデフォルトでプリロードされるようになり、環境変数 DATALAKE_FORMATS は不要になりました。AWS Glue 4.0 以前は、環境変数 DATALAKE_FORMATS を使用して、ロードする特定のテーブル形式を指定していました。

上記のリストは Docker イメージ固有のものです。AWS Glue 5.0 の更新の詳細については、「Introducing AWS Glue 5.0 for Apache Spark」および「Migrating AWS Glue for Spark jobs to AWS Glue version 5.0」を参照してください。

考慮事項

AWS Glue コンテナイメージを使用してジョブスクリプトをローカルで開発する場合、以下の機能はサポートされていないことに注意してください。

- ジョブのブックマーク
- AWS Glue Parquet ライター (AWS Glue で Parquet 形式を使用する場合)
- FillMissingValues 変換
- FindMatches 変換
- ベクトル化された SIMD CSV リーダー
- Amazon S3 パスから JDBC ドライバーをロードする際の customJdbcDriverS3Path プロパティ
- AWS Glue Data Quality
- 機密データの検出
- AWS Lake Formation アクセス許可ベースの認証情報供給

付録: JDBC ドライバーと Java ライブラリの追加

コンテナで現在使用できない JDBC ドライバーを追加するには、必要な JAR ファイルを使用してワークスペースの下に新しいディレクトリを作成し、Docker 実行コマンド内でディレクトリを /opt/spark/jars/ にマウントします。コンテナ内の /opt/spark/jars/ の下にある JAR ファイルは Spark Classpath に自動的に追加され、ジョブの実行中に使用できます。

たとえば、次の Docker 実行コマンドを使用して、PySpark REPL シェルに JDBC ドライバー jar を追加できます。

```
docker run -it --rm \
    -v ~/.aws:/home/hadoop/.aws \
    -v $WORKSPACE_LOCATION:/home/hadoop/workspace/ \
```

```
    -v $WORKSPACE_LOCATION/jars/:/opt/spark/jars/ \
    --workdir /home/hadoop/workspace \
    -e AWS_PROFILE=$PROFILE_NAME \
    --name glue5_jdbc \
    public.ecr.aws/glue/aws-glue-libs:5 \
    pyspark
```

「考慮事項」で強調されているように、customJdbcDriverS3Path 接続オプションを使用して Amazon S3 から AWS Glue コンテナイメージにカスタム JDBC ドライバーをインポートすることはできません。

開発エンドポイント

> **Note**
>
> 開発エンドポイントのコンソール体験は、2023 年 3 月 31 日に削除されました。開発エンドポイントの作成、更新、モニタリングは、開発エンドポイント API および AWS Glue CLI を介して引き続きご使用いただけます。

以下に示す理由により、開発エンドポイントからインタラクティブセッションに移行することを強くお勧めします。開発エンドポイントからインタラクティブセッションへの移行に必要なアクションについては、「開発エンドポイントからインタラクティブセッションへ移行する」を参照してください。

説明	開発者エンドポイント	インタラクティブセッション
Glue バージョンサポート	AWS Glue バージョン 0.9 および 1.0 をサポートしています	AWS Glue バージョン 2.0 以降をサポートしています
開発エンドポイントは、アジアパシフィック (ジャカルタ) (ap-southeast-3)、中東 (UAE) (me-central-1)、欧州 (スペイン) (eu-south-2)、欧州 (チューリッヒ) (eu-central-2) や、今後	現在、インタラクティブセッションは中東 (UAE) (me-central-1) リージョンでは利用できませんが、今後は利用できるようになる可能性があります	

説明	開発者エンドポイント	インタラクティブセッション
開設される新しいリージョンでは利用できません。		
Spark クラスターへのアクセス方法	SSH、REPL シェル、Jupyter Notebook、および IDE (PyCharm など) をサポートしています	AWS Glue Studio ノートブック、Jupyter Notebook、さまざまな IDE (Visual Studio Code、PyCharm など)、および SageMaker AI ノートブックをサポートしています
先頭クエリまでの時間	Spark クラスターのセットアップには 10 ~ 15 分かかります	エフェメラル Spark クラスターの設定には、最大 1 分程度かかる場合があります
料金モデル	AWS は、エンドポイントがプロビジョニングされた時間と DPU の数に基づいて、開発エンドポイントの料金を請求します。開発エンドポイントはタイムアウトしません。プロビジョニングされた開発エンドポイントごとに 10 分の最小課金時間が適用されます。さらに、AWS は、Amazon EC2 インスタンスでの Jupyter Notebook と、開発エンドポイントで設定した場合の SageMaker AI ノートブックに対して料金を請求します。	AWS は、セッションがアクティブになっている時間と DPU の数に基づいて、インタラクティブセッションの料金を請求します。インタラクティブセッションには、設定可能なアイドルタイムアウトがあります。 AWS Glue Studio ノートブックには、インタラクティブなセッション用の組み込みインターフェイスが用意されており、追加費用なしで提供されます。インタラクティブセッションごとに 1 分の最小課金時間が適用されます。AWS Glue Studio ノートブックには、インタラクティブなセッション用の組み込みインターフェイスが用意されているため、追加コストなしで提供されます

説明	開発者エンドポイント	インタラクティブセッション
コンソールエクスペリエンス	CLI および API 経由でのみ使用可能	AWS Glue コンソール、CLI、および API を介して使用可能

開発エンドポイントからインタラクティブセッションへ移行する

次のチェックリストを使用して、開発エンドポイントからインタラクティブセッションに移行する適切な方法を決定してください。

スクリプトは AWS Glue 0.9 または 1.0 の特定の機能 (HDFS、YARN など) に依存していますか?

答えが「はい」の場合は、「AWS Glue ジョブの AWS Glue バージョン 3.0 への移行」を参照して、Glue 0.9 または 1.0 から Glue 3.0 以降に移行する方法を確認してください。

開発エンドポイントへのアクセスにはどの方法を使用していますか?

この方法を使用する場合	こちらを実行してください
SageMaker AI ノートブック、Jupyter Notebook、または JupyterLab	Jupyter で `.ipynb` ファイルをダウンロードして AWS Glue Studio ノートブックに移行し、`.ipynb` ファイルをアップロードして新しい AWS Glue Studio ノートブックジョブを作成します。または、SageMaker AI Studio を使用して AWS Glue カーネルを選択することもできます。
Zeppelin ノートブック	ノートブックの Jupyter Notebook への変換は、コードをコピーして貼り付ける操作を手動で行うか、ze2nb などのサードパーティコンバータを使用して自動で行います。次に、ノートブックを AWS Glue Studio ノートブックまたは SageMaker AI Studio で使用します。
IDE	「AWS Glue インタラクティブセッションを使用した PyCharm による AWS Glue ジョブのオーサリング」、または「Microsoft Visual

この方法を使用する場合	こちらを実行してください
	Studio Code によるインタラクティブセッションの使用」を参照してください。
REPL	`aws-glue-session package` をローカルでインストールしたら、次のコマンドを実行します。 • Python の場合: `jupyter console --kernal glue_pyspark` • Scala の場合: `jupyter console --kernal glue_spark`
SSH	インタラクティブセッションでは対応するオプションがありません。または、Docker イメージを使用することもできます。詳細は「Docker イメージによる開発」をご覧ください。

以下のセクションでは、デベロッパーエンドポイントを使用してジョブを開発する方法の情報を提供します。AWS Glueバージョン 1.0

トピック

- 開発エンドポイントを使用してスクリプトを開発する
- ノートブックの管理

開発エンドポイントを使用してスクリプトを開発する

> **Note**
>
> 開発エンドポイントは、AWS Glue 2.0 より前のバージョンでのみサポートされます。ETL スクリプトを作成およびテストできるインタラクティブな環境の場合は、AWS Glue Studio でノートブックを使用します。

AWS Glue では、開発エンドポイントと呼ばれる環境を作成できます。この環境を使用して、抽出、変換、ロード (ETL) スクリプトを反復的に開発およびテストできます。開発エンドポイントを作成、編集、削除するには、 AWS Glue コンソールまたは API を使用します。

開発環境の管理

開発エンドポイントを作成するときは、開発環境をプロビジョニングするための設定値を指定します。これらの値は、ユーザーが開発エンドポイントに安全にアクセスできるように、またエンドポイントがデータストアにアクセスできるように、ネットワークを設定する方法を AWS Glue に指示します。

次に、開発エンドポイントに接続するノートブックを作成し、このノートブックを使用して ETL スクリプトを作成およびテストできます。開発プロセスの結果に満足したら、スクリプトを実行する ETL ジョブを作成します。このプロセスにより、インタラクティブな方法で機能を追加してスクリプトをデバッグできます。

このセクションのチュートリアルに従って、ノートブックで開発エンドポイントを使用する方法を学習します。

トピック

- 開発エンドポイントのワークフロー
- AWS Glue 開発エンドポイントで SageMaker ノートブックを使用する方法
- 開発エンドポイントの追加
- 開発エンドポイントへのアクセス
- チュートリアル: JupyterLab で Jupyter Notebook をセットアップして ETL スクリプトをテストおよびデバッグする
- チュートリアル: 開発エンドポイントで Amazon SageMaker AI ノートブックを使用する
- チュートリアル: 開発エンドポイントで REPL シェルを使用する
- チュートリアル: 開発エンドポイントで PyCharm Professional をセットアップする
- 高度な設定: 複数のユーザー間で開発エンドポイントを共有する

開発エンドポイントのワークフロー

AWS Glue の開発エンドポイントを使用するには、次のワークフローに従います。

1. API を使用して開発エンドポイントを作成します。このエンドポイントは、定義したセキュリティグループを使用して、Virtual Private Cloud (VPC) 内で起動されます。

2. API は、開発エンドポイントがプロビジョニングされて使用可能になるまで、開発エンドポイントをポーリングします。使用可能になると、次のいずれかの方法を使用して開発エンドポイントに接続し、AWS Glue スクリプトを作成してテストします。

   - アカウントに SageMaker AI ノートブックを作成します。ノートブックの作り方の詳細については、「the section called “AWS Glue Studio ノートブックによるコードの作成”」を参照してください。
   - ターミナルウィンドウを開いて、開発エンドポイントに直接接続します。
   - JetBrains PyCharm Python IDE の Professional エディションがある場合は、これを開発エンドポイントに接続し、これを使用してインタラクティブに開発を行います。スクリプトに pydevd ステートメントを挿入すると、PyCharm でリモートブレークポイントをサポートできます。

3. 開発エンドポイントでのデバッグとテストが完了したら、削除することができます。

AWS Glue 開発エンドポイントで SageMaker ノートブックを使用する方法

開発エンドポイントにアクセスする一般的な方法の 1 つは、SageMaker ノートブックの Jupyter を使用することです。Jupyter Notebook は、可視化、分析、機械学習などで広く使用されているオープンソースのウェブアプリケーションです。AWS Glue SageMaker ノートブックにより、AWS Glue 開発エンドポイントを使用した Jupyter Notebook を使用できます。AWS Glue SageMaker ノートブックでは、Jupyter Notebook 環境は SparkMagic (リモートの Spark クラスターに Spark ジョブを送信するオープンソースの Jupyter プラグイン) で事前設定されています。Apache Livyは、REST API を介してリモート Spark クラスターとのやり取りを可能にするサービスです。AWS Glue SageMaker ノートブックでは、SparkMagic は、AWS Glue 開発エンドポイントで実行されている Livy サーバーに対して REST API を呼び出すように設定されています。

各コンポーネントの動作について、次のテキストフローで説明します。

AWS Glue SageMaker ノートブック: (Jupyter → SparkMagic) → (ネットワーク) → AWS Glue 開発エンドポイント: (Apache Livy → Apache Spark)

Jupyter Notebook 上の各段落に記述された Spark スクリプトを実行すると、Spark コードが SparkMagic 経由で Livy サーバーに送信され、「livy-session-N」という名前の Spark ジョブが Spark クラスター上で実行されます。このジョブは、Livy セッションと呼ばれます。ノートブックセッションが存続している間、Spark ジョブは実行されます。Spark ジョブは、ノートブックから Jupyter カーネルをシャットダウンしたとき、またはセッションがタイムアウトしたときに終了します。ノートブック (.ipynb) ファイルごとに 1 つの Spark ジョブが起動します。

単一の AWS Glue 開発エンドポイントで複数の SageMaker ノートブックインスタンスを使用できます。SageMaker ノートブックインスタンスごとに、複数のノートブックファイルを作成できます。各ノートブックファイルを開いて段落を実行すると、SparkMagic 経由で Spark クラスター上のノートブックファイルごとに Livy セッションが起動します。各 Livy セッションは、単一の Spark ジョブに対応します。

AWS Glue 開発エンドポイントと SageMaker ノートブックのデフォルトの動作

Spark ジョブは、[Spark configuration] (Spark の設定) に基づいて実行されます。Spark の設定には複数の方法があります (Spark クラスター設定、SparkMagic の設定など)。

デフォルトでは、Spark は Spark クラスター設定に基づいて Livy セッションにクラスターリソースを割り当てます。AWS Glue 開発エンドポイントでは、クラスター設定はワーカーのタイプによって異なります。ワーカータイプごとの共通設定について説明した表を次に示します。

	標準	G.1X	G.2X
`spark.driver.memory`	5G	10G	20G
`spark.executor.memory`	5G	10G	20G
`spark.executor.cores`	4	8	16
`spark.dynamicAllocation.enabled`	TRUE	TRUE	TRUE

Sparkエグゼキューターの最大数は、DPU (または `NumberOfWorkers`) とワーカータイプの組み合わせによって自動的に計算されます。

	標準	G.1X	G.2X
Spark エグゼキューターの最大数	`(DPU - 1) * 2 - 1`	`(NumberOfWorkers - 1)`	`(NumberOfWorkers - 1)`

例えば、開発エンドポイントに 10 ワーカーあり、ワーカータイプが `G.1X` の場合、Spark エグゼキューターは 9 つになり、各エグゼキューターは 10G のメモリを持つため、クラスター全体のエグゼキューターメモリは 90G になります。

指定されたワーカータイプに関係なく、Spark 動的リソース割り当てが有効になります。データセットが十分に大きい場合、`spark.dynamicAllocation.maxExecutors` はデフォルトで設定されていないため、Spark はすべてのエグゼキューターを単一の Livy セッションに割り当てることができます。つまり、同じ開発エンドポイント上の他の Livy セッションは、新しいエグゼキューターの起動を待つことになります。データセットが小さい場合、Spark は同時に複数の Livy セッションにエグゼキューターを割り当てることができます。

> **Note**
>
> さまざまなユースケースでリソースがどのように割り当てられるか、および動作を変更するための設定方法の詳細については、「高度な設定: 複数のユーザー間で開発エンドポイントを共有する」を参照してください。

開発エンドポイントの追加

開発エンドポイントを使用して、AWS Glue で抽出、変換、ロード (ETL) スクリプトを反復的に開発およびテストします。開発エンドポイントは、AWS Command Line Interface を介してのみ使用できます。

1. コマンドラインウィンドウで、次のようなコマンドを入力します。

```
aws glue create-dev-endpoint --endpoint-name "endpoint1" --role-arn
 "arn:aws:iam::account-id:role/role-name" --number-of-nodes "3" --glue-version
 "1.0" --arguments '{"GLUE_PYTHON_VERSION": "3"}' --region "region-name"
```

このコマンドでは、AWS Glue バージョン 1.0 を指定します。このバージョンは Python 2 と Python 3 の両方をサポートしているため、必要な Python バージョンを指定するには

arguments パラメータを使用します。glue-version パラメータを省略すると、AWS Glue バージョン 0.9 が想定されます。AWS Glue のバージョンの詳細については、「Glue version job property」を参照してください。

追加のコマンドラインパラメータについては、AWS CLI コマンドリファレンスの「create-dev-endpoint」を参照してください。

2. (オプション) 次のコマンドを入力して、開発エンドポイントのステータスを確認します。ステータスが READY に変わったら、開発エンドポイントの使用を開始できます。

```
aws glue get-dev-endpoint --endpoint-name "endpoint1"
```

開発エンドポイントへのアクセス

Virtual Private Cloud (VPC) に開発エンドポイントを作成すると、AWS Glue はプライベート IP アドレスのみを返します。パブリック IP アドレスフィールドは返されません。VPC 以外の開発エンドポイントを作成する場合、AWS Glue はパブリック IP アドレスのみを返します。

開発エンドポイントにパブリックアドレスがある場合は、次の例のように、開発エンドポイントの SSH プライベートキーを使用して、このアドレスに到達できることを確認します。

```
ssh -i dev-endpoint-private-key.pem glue@public-address
```

開発エンドポイントにプライベートアドレスがあり、VPC サブネットがパブリックインターネットからルーティング可能であり、そのセキュリティグループがクライアントからのインバウンドアクセスを許可するとします。この場合は、以下のステップに従って Elastic IP アドレスを開発エンドポイントにアタッチし、インターネットからのアクセスを許可します。

> **Note**
>
> Elastic IP アドレスを使用する場合、使用されているサブネットには、ルートテーブルを通じて関連付けられたインターネットゲートウェイが必要です。

Elastic IP アドレスをアタッチして開発エンドポイントにアクセスするには

1. https://console.aws.amazon.com/glue/ で AWS Glue コンソール を開きます。

2. ナビゲーションペインで、[開発エンドポイント] を選択し、開発エンドポイントの詳細ページに移動します。次のステップで使用するためにプライベートアドレスを書き留めます。
3. Amazon EC2 コンソール (https://console.aws.amazon.com/ec2/) を開きます。
4. ナビゲーションペインの [Network & Security] で、[ネットワークインターフェイス] を選択します。
5. AWS Glue コンソールの [development endpoint details] (開発エンドポイントの詳細) ページで、プライベートアドレスに対応するプライベート DNS (IPv4) を検索します。

   必要に応じて Amazon EC2 コンソールで表示する列を変更します。このアドレスのネットワークインターフェイス ID (ENI) を書き留めます (例: `eni-12345678`)。
6. Amazon EC2 コンソールの [Network & Security] (ネットワーク & セキュリティ) で、[Elastic IPs] (Elastic IP) をクリックします。
7. [新しいアドレスの割り当て]、[割り当て] の順に選択して、新しい Elastic IP アドレスを割り当てます。
8. [Elastic IP] ページで、新しく割り当てた Elastic IP を選択します。[アクション]、[アドレスの関連付け] の順に選択します。
9. [アドレスの関連付け] ページで、以下の操作を行います。

   - [リソースタイプ] で、[ネットワークインターフェイス] を選択します。
   - [ネットワークインターフェイス] ボックスに、プライベートアドレスのネットワークインターフェイス ID (ENI) を入力します。
   - [関連付ける] を選択します。
10. 次の例に示すように、開発エンドポイントに関連付けられた SSH プライベートキーを使用して、新しく関連付けられた Elastic IP アドレスに到達できることを確認します。

```
ssh -i dev-endpoint-private-key.pem glue@elastic-ip
```

踏み台ホストを使用して、開発エンドポイントのプライベートアドレスへの SSH アクセスを取得する方法については、AWS セキュリティブログの投稿「Securely Connect to Linux Instances Running in a Private Amazon VPC」を参照してください。

チュートリアル: JupyterLab で Jupyter Notebook をセットアップして ETL スクリプトをテストおよびデバッグする

このチュートリアルでは、ローカルマシンで実行されている JupyterLab の Jupyter Notebook を開発エンドポイントに接続します。これは、デプロイする前に、AWS Glue 抽出、変換、ロード (ETL) スクリプトを対話的に実行、デバッグ、およびテストできるようにするためです。このチュートリアルでは、Secure Shell (SSH) ポートフォワーディングを使用して、ローカルマシンを AWS Glue 開発エンドポイントに接続します。詳細については、Wikipedia の「Port forwarding」を参照してください。

ステップ1: JupyterLab と Sparkmagic をインストールする

JupyterLab は `conda` または `pip` を使用してインストールできます。`conda` は、Windows、macOS、および Linux で稼働するオープンソースのパッケージ管理システムおよび環境管理システムです。`pip` は Python のパッケージインストーラです。

macOS にインストールする場合は、Sparkmagic をインストールする前に Xcode をインストールしておく必要があります。

1. JupyterLab、Sparkmagic、および関連する拡張機能をインストールします。

```
$ conda install -c conda-forge jupyterlab
$ pip install sparkmagic
$ jupyter nbextension enable --py --sys-prefix widgetsnbextension
$ jupyter labextension install @jupyter-widgets/jupyterlab-manager
```

2. `Location` から `sparkmagic` ディレクトリをチェックします。

```
$ pip show sparkmagic | grep Location
Location: /Users/username/.pyenv/versions/anaconda3-5.3.1/lib/python3.7/site-
packages
```

3. ディレクトリを、`Location` に対して返されたディレクトリに変更し、Scala と PySpark 用のカーネルをインストールします。

```
$ cd /Users/username/.pyenv/versions/anaconda3-5.3.1/lib/python3.7/site-packages
$ jupyter-kernelspec install sparkmagic/kernels/sparkkernel
$ jupyter-kernelspec install sparkmagic/kernels/pysparkkernel
```

4. サンプルの `config` ファイルをダウンロードします。

```
$ curl -o ~/.sparkmagic/config.json https://raw.githubusercontent.com/jupyter-incubator/sparkmagic/master/sparkmagic/example_config.json
```

この設定ファイルでは、driverMemory や executorCores などの Spark 関連のパラメータを設定できます。

ステップ 2: JupyterLab を起動する

JupyterLab を起動すると、デフォルトのウェブブラウザが自動的に開き、http://localhost:8888/lab/workspaces/{workspace_name} という URL が表示されます。

```
$ jupyter lab
```

ステップ 3: 開発エンドポイントに接続するための SSH ポート転送を開始する

次に、SSH ローカルポート転送を使用して、ローカルポート (ここでは 8998) を AWS Glue で定義されたリモート送信先 (169.254.76.1:8998) に転送します。

1. SSH へのアクセスができる別のターミナルウィンドウを開きます。Microsoft Windows の場合、Git for Windows に用意されている BASH シェルを使用するか、Cygwin をインストールすることができます。
2. 次のように変更した、以下の SSH コマンドを実行します。

   - *private-key-file-path* を、開発エンドポイントを作成するのに使用したパブリックキーに対応するプライベートキーを含む .pem ファイルへのパスに置き換えます。
   - 8998 とは異なるポートを転送している場合は、8998 をローカルで実際に使用しているポート番号に置き換えます。この 169.254.76.1:8998 というアドレスはリモートポートです。変更することはできません。
   - *dev-endpoint-public-dns* を開発エンドポイントのパブリック DNS アドレスで置き換えます。このアドレスを確認するには、AWS Glue コンソールで開発エンドポイントに移動して名前を選択し、[Endpoint details] (エンドポイントの詳細) ページに一覧表示されている [Public address] (パブリックアドレス) をコピーします。

```
ssh -i private-key-file-path -NTL 8998:169.254.76.1:8998 glue@dev-endpoint-public-dns
```

以下のような警告メッセージが表示されます。

```
The authenticity of host 'ec2-xx-xxx-xxx-xx.us-west-2.compute.amazonaws.com
 (xx.xxx.xxx.xx)'
can't be established.  ECDSA key fingerprint is SHA256:4e97875Brt+1wKzRko
+JflSnp21X7aTP3BcFnHYLEts.
Are you sure you want to continue connecting (yes/no)?
```

yes と入力し、JupyterLab を使用中はターミナルウィンドウを開いたままにしておきます。

3. SSH ポート転送が開発エンドポイントで正しく機能していることを確認します。

```
$ curl localhost:8998/sessions
{"from":0,"total":0,"sessions":[]}
```

ステップ 4: ノートブックの段落でシンプルスクリプトフラグメントを実行する

JupyterLab のノートブックは、開発エンドポイントで動作するようになっています。次のスクリプトフラグメントをノートブックに入力して実行します。

1. Spark が正常に実行されていることを確認します。次のコマンドによって、Sparkが 1 を計算して値を出力します。

```
spark.sql("select 1").show()
```

2. AWS Glue Data Catalog 統合が機能しているかどうか確認します。次のコマンドで、データカタログ内のテーブルが一覧表示されます。

```
spark.sql("show tables").show()
```

3. AWS Glue ライブラリを使用するシンプルスクリプトフラグメントが機能していることを確認します。

次のスクリプトでは、AWS Glue Data Catalog の persons_json テーブルメタデータを使用して、サンプルデータから DynamicFrame を作成します。次に、このデータの項目数およびスキーマが出力されます。

```
import sys
```

```
from pyspark.context import SparkContext
from awsglue.context import GlueContext

# Create a Glue context
glueContext = GlueContext(SparkContext.getOrCreate())

# Create a DynamicFrame using the 'persons_json' table
persons_DyF = glueContext.create_dynamic_frame.from_catalog(database="legislators",
 table_name="persons_json")

# Print out information about *this* data
print("Count:  ", persons_DyF.count())
persons_DyF.printSchema()
```

スクリプトの出力は次のとおりです。

```
Count:  1961
root
|-- family_name: string
|-- name: string
|-- links: array
|    |-- element: struct
|    |    |-- note: string
|    |    |-- url: string
|-- gender: string
|-- image: string
|-- identifiers: array
|    |-- element: struct
|    |    |-- scheme: string
|    |    |-- identifier: string
|-- other_names: array
|    |-- element: struct
|    |    |-- note: string
|    |    |-- name: string
|    |    |-- lang: string
|-- sort_name: string
|-- images: array
|    |-- element: struct
|    |    |-- url: string
|-- given_name: string
|-- birth_date: string
|-- id: string
```

```
|-- contact_details: array
|    |-- element: struct
|    |    |-- type: string
|    |    |-- value: string
|-- death_date: string
```

トラブルシューティング

- JupyterLab のインストール中に、コンピュータが企業のプロキシまたはファイアウォールの内側にある場合、企業の IT 部門によって管理されるカスタムセキュリティプロファイルにより HTTP および SSL エラーが発生することがあります。

  conda が独自のリポジトリに接続できないときに、よく発生するエラーの例を次に示します。

  ```
  CondaHTTPError: HTTP 000 CONNECTION FAILED for url <https://repo.anaconda.com/pkgs/
  main/win-64/current_repodata.json>
  ```

  これは、会社が Python および JavaScript コミュニティで広く使用されているリポジトリへの接続をブロックすることがあるために発生した可能性があります。詳細については、JupyterLab ウェブサイトの「Installation Problems」を参照してください。
- 開発エンドポイントに接続しようとしたときに「connection refused (接続拒否)」エラーが発生した場合、古い開発エンドポイントを使用している可能性があります。新しい開発エンドポイントを作成して再接続してみます。

チュートリアル: 開発エンドポイントで Amazon SageMaker AI ノートブックを使用する

AWS Glue では、開発エンドポイントを作成し、ETL および機械学習スクリプトの開発に役立つ SageMaker AI ノートブックを作成できます。SageMaker AI ノートブックは、Jupyter Notebook アプリケーションを実行しているフルマネージド型機械学習コンピューティングインスタンスです。

1. AWS Glue コンソールで [Dev endpoints (開発エンドポイント)] を選択して、開発エンドポイントのリストに移動します。
2. 使用する開発エンドポイントの名前の横にあるチェックボックスをオンにし、[アクション] メニューで [Create SageMaker notebook (SageMaker ノートブックの作成)] を選択します。
3. [Create and configure a notebook (ノートブックの作成と設定)] ページに次のように入力します。

a. ノートブック名を入力します。

b. [Attach to development endpoint (開発エンドポイントにアタッチ] で、開発エンドポイントを確認します。

c. AWS Identity and Access Management (IAM) ロールを作成または選択します。

ロールを作成することをお勧めします。既存のロールを使用する場合は、必要な権限があることを確認します。(詳細については、the section called “ステップ 6: SageMaker AI ノートブック用の IAM ポリシーを作成する” を参照してください)。

d. (オプション) VPC、サブネット、および 1 つ以上のセキュリティグループを選択します。

e. (オプション) AWS Key Management Service 暗号化キーを選択します。

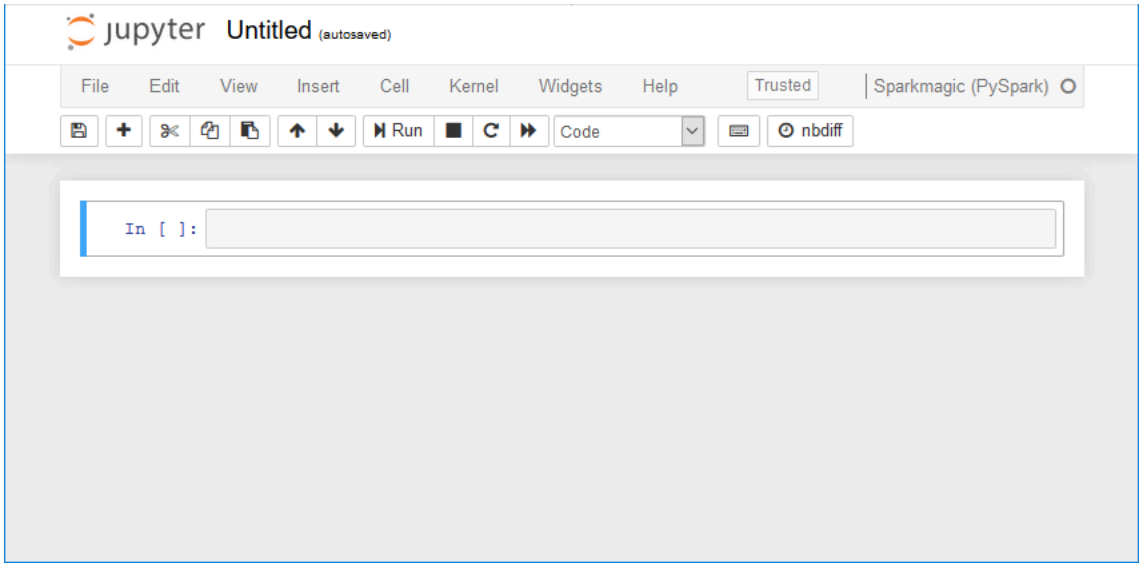
f. (オプション) ノートブックインスタンスのタグを追加します。

4. [Create notebook (ノートブックの作成)] を選択します。[ノートブック] ページで、右上にある更新アイコンを選択し、[ステータス] に Ready と表示されるまで続行します。

5. 新しいノートブック名の横にあるチェックボックスをオンにし、[ノートブックを開く] を選択します。

6. 新しいノートブックを作成する: jupyter ページで [新規] を選択し、[Sparkmagic (PySpark)] を選択します。

これで、画面は以下のようになります。

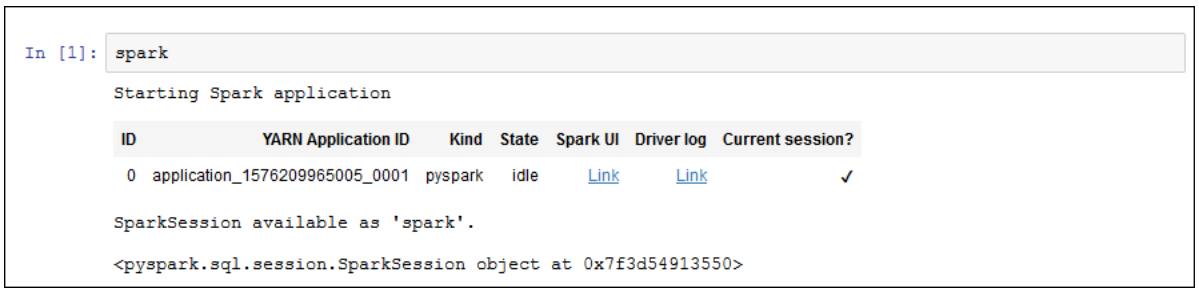


7. (オプション) ページの上部にある [Untitled (無題)] を選択し、ノートブックに名前を付けます。

8. Spark アプリケーションを起動するには、次のコマンドをノートブックに入力し、ツールバーで [実行] を選択します。

```
spark
```

しばらくすると、次のようなレスポンスが表示されます。



9. 動的フレームを作成し、それに対してクエリを実行します。persons_json テーブルのカウントとスキーマを出力する次のコードをコピー、貼り付け、実行します。

```
import sys
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.transforms import *
glueContext = GlueContext(SparkContext.getOrCreate())
persons_DyF = glueContext.create_dynamic_frame.from_catalog(database="legislators",
 table_name="persons_json")
print ("Count:  ", persons_DyF.count())
persons_DyF.printSchema()
```

チュートリアル: 開発エンドポイントで REPL シェルを使用する

AWS Glue では、開発エンドポイントを作成してから、REPL (Read-Evaluate-Print Loop) シェルを呼び出して PySpark コードを増分的に実行し、ETL スクリプトをデプロイする前にインタラクティブにデバッグできるようにします。

開発エンドポイントで REPL を使用するには、エンドポイントに SSH で接続するための承認が必要です。

1. ローカルコンピュータで、SSH コマンドを実行できるターミナルウィンドウを開き、編集した SSH コマンドを貼り付けます。コマンドを実行します。

   開発エンドポイントで AWS Glue バージョン 1.0 (Python 3 を使用) を受け入れた場合、出力は次のようになります。

```
Python 3.6.8 (default, Aug  2 2019, 17:42:44)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-28)] on linux
Type "help", "copyright", "credits" or "license" for more information.
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/share/aws/glue/etl/jars/glue-assembly.jar!/
org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/spark/jars/slf4j-log4j12-1.7.16.jar!/
org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use
 setLogLevel(newLevel).
2019-09-23 22:12:23,071 WARN  [Thread-5] yarn.Client (Logging.scala:logWarning(66))
 - Neither spark.yarn.jars nor spark.yarn.archive is set, falling back to uploading
 libraries under SPARK_HOME.
2019-09-23 22:12:26,562 WARN  [Thread-5] yarn.Client (Logging.scala:logWarning(66))
 - Same name resource file:/usr/lib/spark/python/lib/pyspark.zip added multiple
 times to distributed cache
2019-09-23 22:12:26,580 WARN  [Thread-5] yarn.Client (Logging.scala:logWarning(66))
 - Same path resource file:///usr/share/aws/glue/etl/python/PyGlue.zip added
 multiple times to distributed cache.
2019-09-23 22:12:26,581 WARN  [Thread-5] yarn.Client (Logging.scala:logWarning(66))
 - Same path resource file:///usr/lib/spark/python/lib/py4j-src.zip added multiple
 times to distributed cache.
2019-09-23 22:12:26,581 WARN  [Thread-5] yarn.Client (Logging.scala:logWarning(66))
 - Same path resource file:///usr/share/aws/glue/libs/pyspark.zip added multiple
 times to distributed cache.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 2.4.3
      /_/

Using Python version 3.6.8 (default, Aug  2 2019 17:42:44)
SparkSession available as 'spark'.
>>>
```

2. ステートメント `print(spark.version)` を入力して REPL シェルが正常に動作しているかテストします。Spark のバージョンが表示されれば、REPL を使用する準備ができたことになります。

3. シェルで次のシンプルなスクリプトを行単位で実行することができます。

```
import sys
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.transforms import *
glueContext = GlueContext(SparkContext.getOrCreate())
persons_DyF = glueContext.create_dynamic_frame.from_catalog(database="legislators",
 table_name="persons_json")
print ("Count:  ", persons_DyF.count())
persons_DyF.printSchema()
```

チュートリアル: 開発エンドポイントで PyCharm Professional をセットアップする

このチュートリアルでは、ローカルマシンで実行中の PyCharm Professional Python IDE を開発エンドポイントに接続し、AWS Glue ETL (抽出、転送、およびロード) スクリプトをデプロイ前にインタラクティブに実行、デバッグ、およびテストします。チュートリアルの手順とスクリーンショットは、PyCharm Professional バージョン 2019.3 に基づいています。

開発エンドポイントをインタラクティブに接続するには、PyCharm Professional がインストールされている必要があります。無料版を使用してこれを行うことはできません。

> **Note**
>
> このチュートリアルでは、Amazon S3 をデータソースとして使用します。代わりに JDBC データソースを使用する場合は、Virtual Private Cloud (VPC) で開発エンドポイントを実行する必要があります。SSH を使用して VPC の開発エンドポイントに接続するには、SSH トンネルを作成する必要があります。このチュートリアルには、SSH トンネルを作成する手順は含まれていません。SSH を使用して VPC の開発エンドポイントに接続する方法については、AWS セキュリティブログの「Securely Connect to Linux Instances Running in a Private Amazon VPC」を参照してください。

トピック

- PyCharm Professional を開発エンドポイントに接続する
- 開発エンドポイントにスクリプトをデプロイする
- リモートインタープリタの設定
- 開発エンドポイントでスクリプトを実行する

PyCharm Professional を開発エンドポイントに接続する

1. PyCharm に legislators という名前の新しい純粋な Python プロジェクトを作成します。
2. プロジェクトに get_person_schema.py という名前のファイルを、次の内容で作成します。

```
from pyspark.context import SparkContext
from awsglue.context import GlueContext


def main():
    # Create a Glue context
    glueContext = GlueContext(SparkContext.getOrCreate())

    # Create a DynamicFrame using the 'persons_json' table
    persons_DyF =
 glueContext.create_dynamic_frame.from_catalog(database="legislators",
 table_name="persons_json")

    # Print out information about this data
    print("Count:  ", persons_DyF.count())
    persons_DyF.printSchema()


if __name__ == "__main__":
    main()
```

3. 次のいずれかを行います。

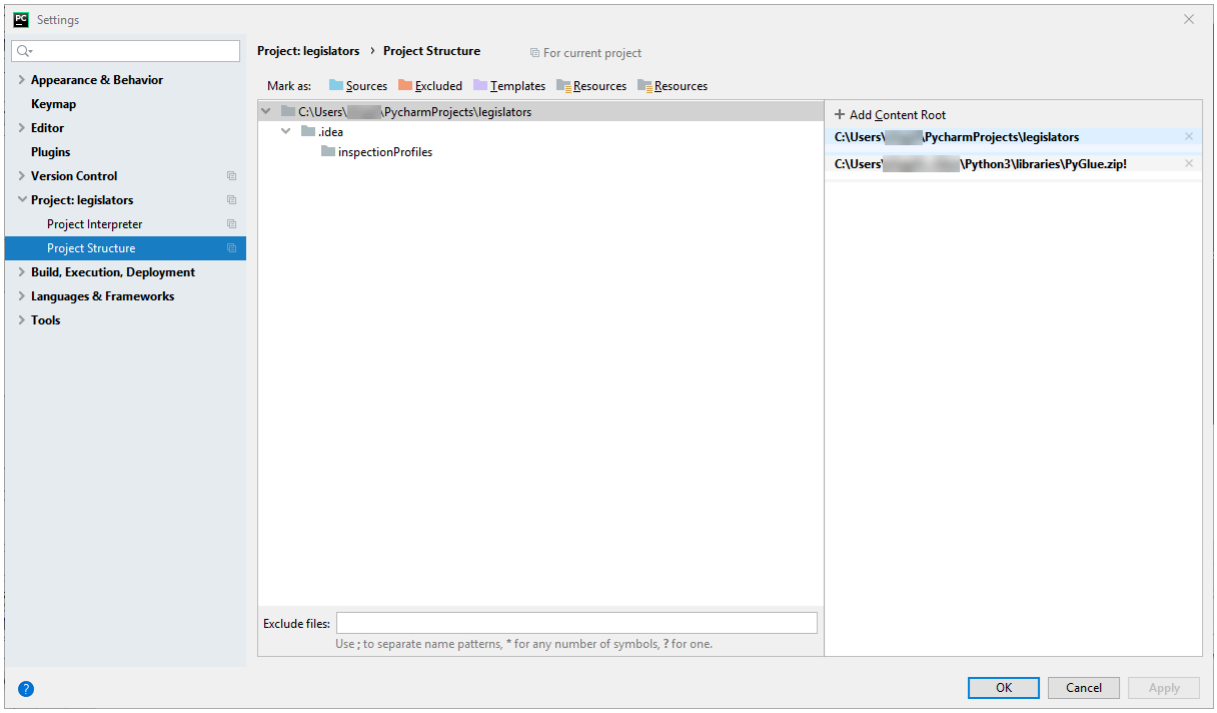
   - AWS Glue バージョン 0.9 の場合、AWS Glue Python ライブラリファイル PyGlue.zip を https://s3.amazonaws.com/aws-glue-jes-prod-us-east-1-assets/etl/python/PyGlue.zip から、ローカルマシンの便利な場所にダウンロードします。
   - AWS Glue バージョン 1.0 以降の場合、AWS Glue Python ライブラリファイル PyGlue.zip を https://s3.amazonaws.com/aws-glue-jes-prod-us-east-1-assets/etl-1.0/python/PyGlue.zip から、ローカルマシンの便利な場所にダウンロードします。

4. PyCharm のプロジェクトのコンテンツルートとして PyGlue.zip を追加します。

   - PyCharm で、[File] (ファイル)、[Settings] (設定) の順に選択し、[Settings] (設定) ダイアログボックスを開きます。(Ctrl+Alt+S キーを押す方法もあります。)

- `legislators` プロジェクトを展開し、[Project Structure] (プロジェクト構造) を選択します。次に、右ペインで [+Add Content Root] (+ コンテンツルートの追加) を選択します。
- `PyGlue.zip` を保存した場所に移動して選択し、[Apply] (適用) を選択します。

[Settings] (設定) 画面は以下のようになります。



[Apply] (適用) を選択した後は、[Settings] (設定) ダイアログボックスを開いたままにします。

5. デプロイオプションを設定し、SFTP を使用してローカルスクリプトを開発エンドポイントにアップロードします (この機能は、PyCharm Professional でのみ使用できます)。

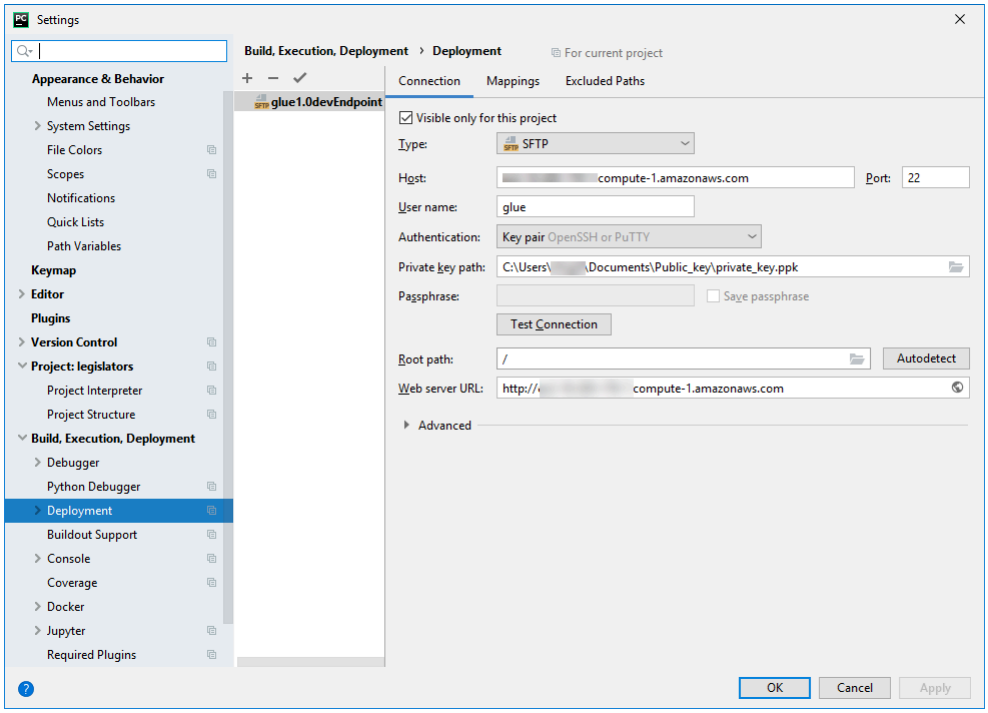
   - [Settings] (設定) ダイアログボックスで、[Build, Execution, Deployment] (ビルド、実行、デプロイ) セクションを展開します。[Deployment] (デプロイ) サブセクションを選択します。
   - 中央のペインの一番上にある [+] アイコンを選択し、新しいサーバーを追加します。[タイプ] を SFTP に設定し、名前を付けます。
   - 詳細ページにリストされているように、[SFTP ホスト] を開発エンドポイントの [パブリックアドレス] に設定します (詳細ページを表示するには、AWS Glue コンソールで開発エンドポイントの名前を選択します)。VPC で実行されている開発エンドポイントの場合、[SFTP ホスト] をホストアドレスに設定し、SSH トンネルのローカルポートを開発エンドポイントに設定します。
   - [User name] (ユーザー名) を `glue` に設定します。

- [Auth type] (認証タイプ) を [Key pair (OpenSSH or Putty)] (キーペア、OpenSSH または Putty) に設定します。開発エンドポイントのプライベートキーファイルがある場所を参照し、プライベートキーファイルを設定します。PyCharm は DSA、RSA、ECDSA の OpenSSH キータイプのみをサポートし、Putty のプライベート形式のキーは受け入れないことに注意してください。ssh-keygen の最新バージョンを使用して、PyCharm が受け入れるキーペアタイプを以下のような構文で生成できます。

```
ssh-keygen -t rsa -f <key_file_name> -C "<your_email_address>"
```

- [Test connection (接続のテスト)] を選択し、接続をテストします。接続が成功したら、[Apply] (適用) を選択します。

[Settings] (設定) 画面は以下のようになります。



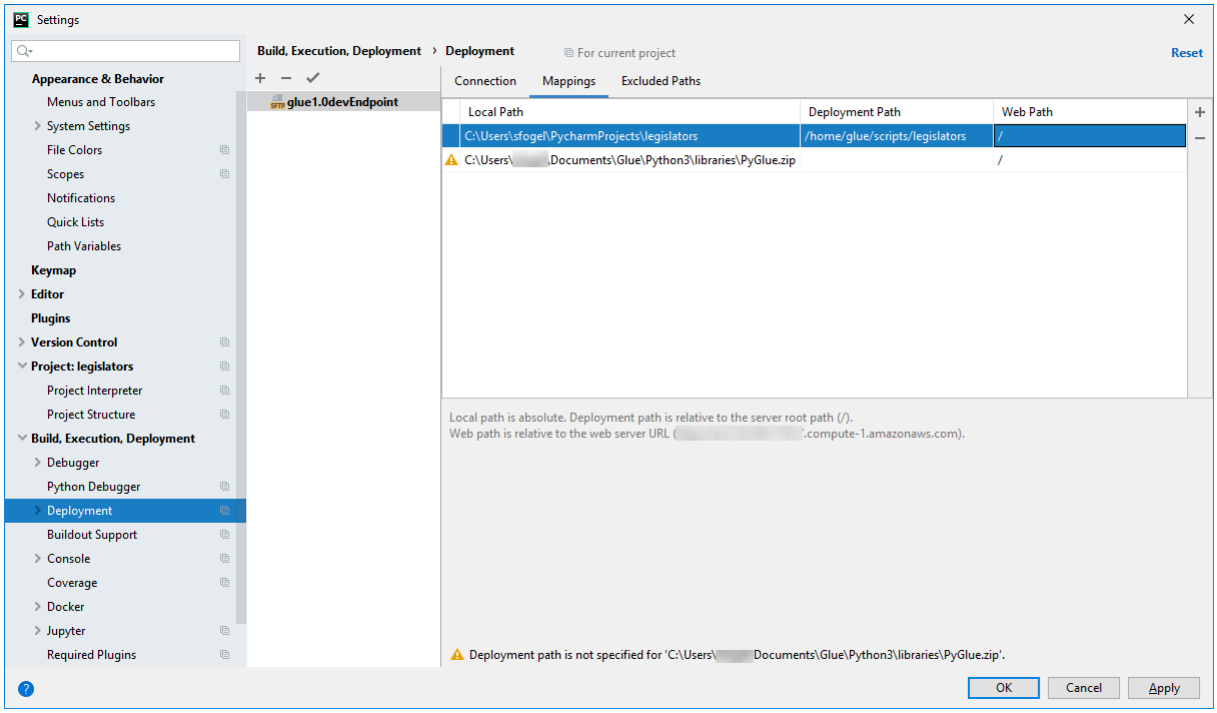
再び、[Apply] (適用) を選択した後は、[Settings] (設定) ダイアログボックスを開いたままにします。

6. ローカルディレクトリをデプロイ用のリモートディレクトリにマッピングします。

- 右のペインの [Deployment] (デプロイ) ページで、[Mappings] (マッピング) と書かれた中央上部のタブを選択します。

- [Deployment Path] (デプロイパス) 列で、プロジェクトパスのデプロイ用に `/home/glue/scripts/` の下にパスを入力します。例: `/home/glue/scripts/legislators`。
- [Apply] を選択します。

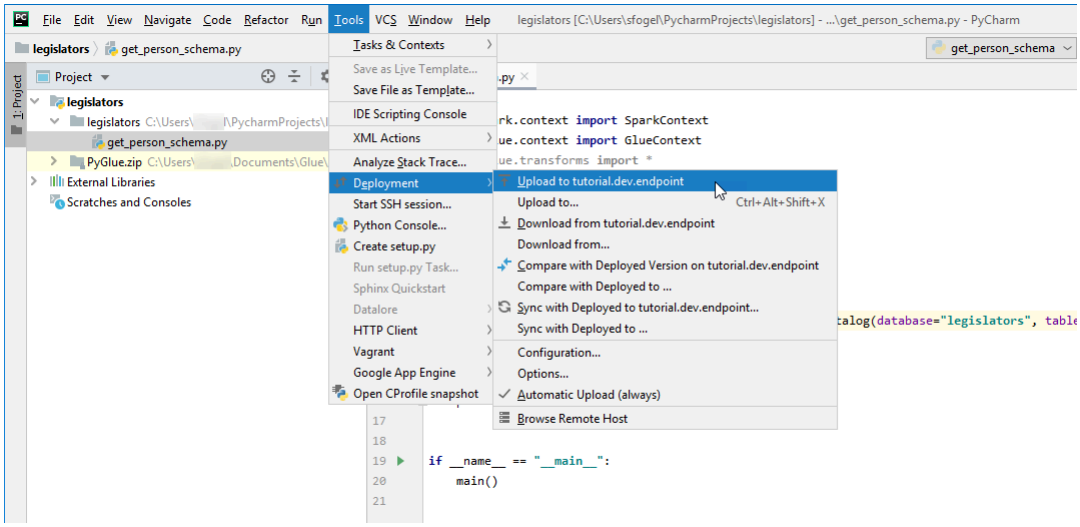
[Settings] (設定) 画面は以下のようになります。



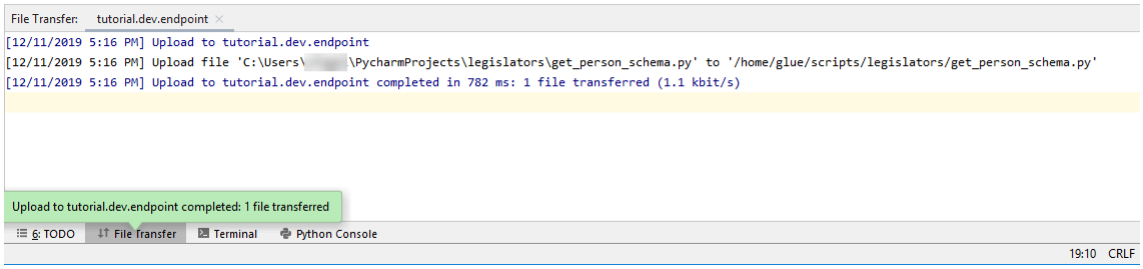
[OK] を選択し、[設定] ダイアログボックスを閉じます。

開発エンドポイントにスクリプトをデプロイする

1. [Tools (ツール)]、[Deployment (デプロイ)] の順に選択した後、次の図に示すように、開発エンドポイントをセットアップする名前を選択します。



スクリプトがデプロイされた後、画面の下部は、次のようになります。



2. メニューバーで、[Tools (ツール)]、[Deployment (デプロイ)]、[Automatic Upload (always) (自動アップロード (常時))] の順に選択します。[Automatic Upload (always) (自動アップロード (常時))] の横にチェックマークが表示されていることを確認します。

   このオプションを有効にすると、PyCharm は変更されたファイルを開発エンドポイントに自動的にアップロードします。

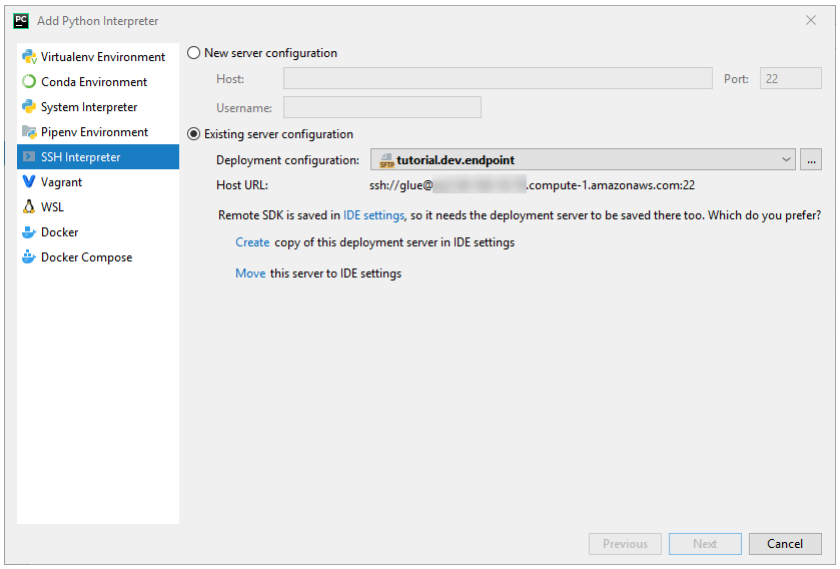
リモートインタープリタの設定

開発エンドポイントで Python インタプリタを使用するように PyCharm を設定します。

1. [ファイル] メニューの [設定] を選択します。
2. プロジェクトの [legislators (立法者)] を展開し、[Project Interpreter (プロジェクトインタープリタ)] を選択します。
3. [Project Interpreter (プロジェクトインタープリタ)] リストの横にある歯車アイコンを選択し、[追加] を選択します。
4. [Add Python Interpreter (Python インタープリタを追加)] ダイアログボックスの左ペインで、[SSH Interpreter (SSH インタープリタ)] を選択します。

5. [Existing server configuration (既存のサーバー設定)] を選択し、[Deployment configuration (展開設定)] リストで設定を選択します。

   これで、画面は以下の図のようになります。



6. [Move this server to IDE settings (このサーバーを IDE 設定に移動する)] を選択し、[次へ] を選択します。
7. [Interpreter (インタプリタ)] フィールドで、Python 2 を使用している場合は、パスを `/usr/bin/gluepython` に変更し、Python 3 を使用している場合は `/usr/bin/gluepython3` に変更します。次に、[Finish] (終了) を選択します。

開発エンドポイントでスクリプトを実行する

スクリプティングを実行するには:

- 左ペインでファイル名を右クリックし、[***<filename>*** を実行] を選択します。

  一連のメッセージの後、最終的な出力にはカウントとスキーマが表示されます。

```
Count:  1961
root
|-- family_name: string
|-- name: string
|-- links: array
|    |-- element: struct
|    |    |-- note: string
|    |    |-- url: string
|-- gender: string
```

```
|-- image: string
|-- identifiers: array
|    |-- element: struct
|    |    |-- scheme: string
|    |    |-- identifier: string
|-- other_names: array
|    |-- element: struct
|    |    |-- lang: string
|    |    |-- note: string
|    |    |-- name: string
|-- sort_name: string
|-- images: array
|    |-- element: struct
|    |    |-- url: string
|-- given_name: string
|-- birth_date: string
|-- id: string
|-- contact_details: array
|    |-- element: struct
|    |    |-- type: string
|    |    |-- value: string
|-- death_date: string


Process finished with exit code 0
```

これで開発エンドポイントでスクリプトをリモートでデバッグするためのセットアップができました。

高度な設定: 複数のユーザー間で開発エンドポイントを共有する

このセクションでは、一般的なユースケースで SageMaker ノートブックを使用して開発エンドポイントを活用し、複数のユーザー間で開発エンドポイントを共有する方法について説明します。

シングルテナンシーの設定

シングルテナントのユースケースでは、デベロッパーの作業をシンプルにし、リソースの競合を避けるために、各デベロッパーが、作業しているプロジェクトに適したサイズの独自の開発エンドポイントを使用することをお勧めします。これにより、ワーカータイプと DPU 数に関する決定がシンプルになり、デベロッパーおよび作業しているプロジェクトの裁量に任せられます。

複数のノートブックファイルを同時に実行しない限り、リソースの割り当てを検討する必要はありません。同時に複数のノートブックファイルでコードを実行すると、複数の Livy セッションが同時に起動します。複数の Livy セッションを同時に実行するために、Spark クラスター設定を分離するには、マルチテナントのユースケースで導入された手順に従います。

例えば、開発エンドポイントに 10 ワーカーあり、ワーカータイプが `G.1X` の場合、Spark エグゼキューターは 9 つになり、各エグゼキューターは 10G のメモリを持つため、クラスター全体のエグゼキューターメモリは 90G になります。

指定されたワーカータイプに関係なく、Spark 動的リソース割り当てが有効になります。データセットが十分に大きい場合、`spark.dynamicAllocation.maxExecutors` はデフォルトで設定されていないため、Spark はすべてのエグゼキューターを単一の Livy セッションに割り当てることができます。つまり、同じ開発エンドポイント上の他の Livy セッションは、新しいエグゼキューターの起動を待つことになります。データセットが小さい場合、Spark は同時に複数の Livy セッションにエグゼキューターを割り当てることができます。

> **Note**
>
> さまざまなユースケースでリソースがどのように割り当てられるか、および動作を変更するための設定方法の詳細については、「高度な設定: 複数のユーザー間で開発エンドポイントを共有する」を参照してください。

マルチテナンシーの設定

> **Note**
>
> 開発エンドポイントは、シングルテナント環境として AWS Glue ETL 環境をエミュレートするためのものであることに留意してください。マルチテナントの使用は可能ですが、これは高度なユースケースであり、ほとんどのユーザーには、開発エンドポイントごとにシングルテナンシーのパターンを維持することをお勧めします。

マルチテナントのユースケースでは、リソース割り当てを検討する必要があることがあります。重要な要因は、同時に Jupyter Notebook を使用するユーザーの数です。チームが「フォローザサン」ワークフローで作業し、各タイムゾーンに Jupyter ユーザーが 1 人しかいない場合、同時ユーザーの数は 1 人であるため、リソースの割り当てについて心配する必要はありません。ただし、ノートブックが複数のユーザー間で共有され、各ユーザーがアドホックベースでコードを送信する場合は、以下の点を考慮する必要があります。

Spark クラスターリソースを複数のユーザー間でパーティション化するには、SparkMagic の設定を使用します。Spark Magic の設定には、2 つの異なる方法があります。

(A) %%configure -f ディレクティブを使用する

ノートブックから Livy セッションごとの設定を変更したい場合は、`%%configure -f` ディレクティブをノートブックの段落で実行します。

例えば、5 つのエグゼキューターで Spark アプリケーションを実行する場合は、ノートブックの段落で次のコマンドを実行します。

```
%%configure -f
{"numExecutors":5}
```

そうすると、Spark UI 上でジョブに対して実行されている エグゼキューターが 5 つだけ表示されます。

動的リソース割り当てでは、エグゼキューターの最大数を制限することをお勧めします。

```
%%configure -f
{"conf":{"spark.dynamicAllocation.maxExecutors":"5"}}
```

(B) SparkMagic Config ファイルを変更する

SparkMagic は、Livy API に基づいて動作します。SparkMagic は、`driverMemory`、`driverCores`、`executorMemory`、`executorCores`、 `numExecutors`、`conf` などの設定で Livy セッションを作成します。これらは、Spark クラスター全体で消費されるリソースの量を決定する重要な要素です。SparkMagic を使用すると、Livy に送信されるパラメータを指定するための設定ファイルを指定することができます。この GitHub リポジトリ にサンプルの設定ファイルがあります。

ノートブックからすべての Livy セッションの設定を変更したい場合は、`/home/ec2-user/.sparkmagic/config.json` を変更して `session_config` を追加します。

SageMaker ノートブックインスタンスの設定ファイルを変更するには、以下の手順に従います。

1. SageMaker ノートブックを開きます。
2. ターミナルカーネルを開きます。
3. 以下のコマンドを実行します。

```
sh-4.2$ cd .sparkmagic
```

```
sh-4.2$ ls
config.json logs
sh-4.2$ sudo vim config.json
```

例えば、これらの行を `/home/ec2-user/.sparkmagic/config.json` に追加し、ノートブックから Jupyter カーネルを再起動します。

```
  "session_configs": {
    "conf": {
      "spark.dynamicAllocation.maxExecutors":"5"
    }
  },
```

ガイドラインとベストプラクティス

このようなリソースの競合を避けるために、次のような基本的なアプローチを使用できます。

- `NumberOfWorkers` (水平方向のスケーリング) を増やし、 `workerType` (垂直スケーリング) をアップグレードして、Spark クラスターを大きくする
- ユーザーあたりの割り当てリソースを削減 (Livy セッションあたりのリソースの削減)

アプローチはユースケースによって異なります。開発エンドポイントが大きく、大量のデータがない場合、Spark は動的割り当て戦略に基づいてリソースを割り当てることができるため、リソースの競合の可能性が大幅に低下します。

上記のように、Spark エグゼキューターの数は、DPU (または `NumberOfWorkers`) とワーカータイプの組み合わせに基づいて自動的に計算できます。各 Spark アプリケーションは、1つのドライバーと複数のエグゼキューターを起動します。計算するには、 `NumberOfWorkers` = `NumberOfExecutors` + 1 が必要です。以下のマトリックスは、同時ユーザー数に基づいて、開発エンドポイントに必要な容量を示しています。

同時に実行するノートブックユーザーの数	ユーザーごとに割り当てる Spark エグゼキューターの数	開発エンドポイントの NumberOfWorkers 合計
3	5	18
10	5	60

同時に実行するノートブックユーザーの数	ユーザーごとに割り当てるSpark エグゼキュターの数	開発エンドポイントの NumberOfWorkers 合計
50	5	300

ユーザーごとに割り当てるリソースを少なくしたい場合、spark.dynamicAllocation.maxExecutors (または numExecutors) は、Livy セッションパラメータとして設定する最も簡単なパラメータです。/home/ec2-user/.sparkmagic/config.json で以下のように設定した場合、SparkMagic は Livy セッションごとに最大 5 つのエグゼキュターを割り当てます。これは、Livy セッションごとにリソースを分離するのに役立ちます。

```
"session_configs": {
    "conf": {
      "spark.dynamicAllocation.maxExecutors":"5"
    }
  },
```

18 のワーカー (G.1X) が使用されている開発エンドポイントで、同時実行するノートブックユーザー数が 3 とします。セッション設定に　spark.dynamicAllocation.maxExecutors=5 が含まれていると、各ユーザーは 1 つのドライバーと 5 つのエグゼキュターを使用できます。複数のノートブックの段落を同時に実行しても、リソースの競合は発生しません。

トレードオフ

この "spark.dynamicAllocation.maxExecutors":"5" というセッション設定で、リソースの競合エラーを回避することができ、同時ユーザーアクセスがあるときにリソース割り当てを待つ必要はありません。ただし、多くの空きリソースがある場合 (例えば、他の同時ユーザーがいない場合) でも、Spark は Livy セッションに 5 つまでしか工グゼキュターを割り当てることができません。

その他の注意事項

ノートブックの使用を停止するときは、Jupyter カーネルを停止することをお勧めします。これにより、リソースが解放され、他のノートブックユーザーはカーネルの有効期限 (自動シャットダウン) を待たずにすぐにこれらのリソースを使用できます。

一般的な問題

ガイドラインに従っている場合でも、特定の問題が発生する可能性があります。

セッションが見つからない

Livy セッションがすでに終了しているにもかかわらず、ノートブック段落を実行しようとすると、以下のメッセージが表示されます。Livy セッションをアクティブにするには、Jupyter カーネルを再起動する必要があります。Jupyter メニューで [Kernel] (カーネル) > [Restart] (再起動) を選択し、ノートブックの段落をもう一度実行します。

```
An error was encountered:
Invalid status code '404' from http://localhost:8998/sessions/13 with error payload:
 "Session '13' not found."
```

YARN リソースが不足

Spark クラスターに新しい Livy セッションを開始するのに十分なリソースがないにもかかわらず、ノートブックの段落を実行しようとすると、以下のメッセージが表示されます。多くの場合、ガイドラインに従うことでこの問題を回避できますが、この問題に直面する可能性があります。この問題を回避するには、不要なアクティブな Livy セッションがあるかどうかを確認します。不要な Livy セッションがある場合、クラスターリソースを解放するためにセッションを終了する必要があります。詳細については、次のセクションを参照ください。

```
Warning: The Spark session does not have enough YARN resources to start.
The code failed because of a fatal error:
    Session 16 did not start up in 60 seconds..

Some things to try:
a) Make sure Spark has enough available resources for Jupyter to create a Spark
 context.
b) Contact your Jupyter administrator to make sure the Spark magics library is
 configured correctly.
c) Restart the kernel.
```

モニタリングとデバッグ

このセクションでは、リソースとセッションをモニタリングするための手法について説明します。

クラスターリソース割り当てのモニタリングとデバッグ

Spark UI を見て、Livy セッションごとに割り当てられているリソースの数と、ジョブで有効な Spark 設定をモニタリングできます。Spark UI をアクティブ化するには、「Enabling the Apache Spark Web UI for Development Endpoints」を参照してください。

(オプション) Spark UI のリアルタイムビューが必要な場合は、Spark クラスターで実行されている Spark 履歴サーバーに対して SSH トンネルを設定できます。

```
ssh -i <private-key.pem> -N -L 8157:<development endpoint public address>:18080
 glue@<development endpoint public address>
```

ブラウザで http://localhost:8157 を開き、Spark UI を表示できます。

不要な Livy セッションの解放

ノートブックまたは Spark クラスターから不要な Livy セッションをすべてシャットダウンするには、以下の手順を確認してください。

(a)。ノートブックから Livy セッションを終了する

Jupyter Notebook でカーネルをシャットダウンして、不要な Livy セッションを終了できます。

(b)。Spark クラスターから Livy セッションを終了する

不要な Livy セッションがまだ実行されている場合は、Spark クラスターで Livy セッションをシャットダウンできます。

この手順を実行するための前提条件として、開発エンドポイントの SSH パブリックキーを設定する必要があります。

Spark クラスターにログインするには、次のコマンドを実行します。

```
$ ssh -i <private-key.pem> glue@<development endpoint public address>
```

次のコマンドを実行して、アクティブな Livy セッションを表示できます。

```
$ yarn application -list
20/09/25 06:22:21 INFO client.RMProxy: Connecting to ResourceManager at
 ip-255-1-106-206.ec2.internal/172.38.106.206:8032
Total number of applications (application-types: [] and states: [SUBMITTED, ACCEPTED,
 RUNNING]):2
Application-Id Application-Name Application-Type User Queue State Final-State Progress
 Tracking-URL
application_1601003432160_0005 livy-session-4 SPARK livy default RUNNING UNDEFINED 10%
 http://ip-255-1-4-130.ec2.internal:41867
application_1601003432160_0004 livy-session-3 SPARK livy default RUNNING UNDEFINED 10%
 http://ip-255-1-179-185.ec2.internal:33727
```

その後、次のコマンドを使用して Livy セッションをシャットダウンできます。

```
$ yarn application -kill application_1601003432160_0005
20/09/25 06:23:38 INFO client.RMProxy: Connecting to ResourceManager at
 ip-255-1-106-206.ec2.internal/255.1.106.206:8032
Killing application application_1601003432160_0005
20/09/25 06:23:39 INFO impl.YarnClientImpl: Killed application
 application_1601003432160_0005
```

ノートブックの管理

> **Note**
>
> 開発エンドポイントは、AWS Glue 2.0 より前のバージョンでのみサポートされます。ETL スクリプトを作成およびテストできるインタラクティブな環境の場合は、AWS Glue Studio でノートブックを使用します。

ノートブックを使用すると、開発エンドポイントでの ETL (抽出、変換、ロード) スクリプトのインタラクティブな開発とテストを行うことができます。AWS Glue には、SageMaker AI Jupyter Notebook に対するインターフェイスが用意されています。AWS Glue を使用して、SageMaker AI ノートブックを作成および管理します。AWS Glue コンソールから SageMaker AI ノートブックを開くこともできます。

また、SageMaker AI (AWS Glue ETL ジョブではない) をサポートする AWS Glue 開発エンドポイント上の SageMaker AI で Apache Spark を使用できます。SageMaker Spark は、オープンソースの SageMaker AI 用 Apache Spark ライブラリです。詳細については、「Amazon SageMaker で Apache Spark を使う」を参照してください。

> **Important**
>
> AWS Glue 開発エンドポイントでの SageMaker AI ノートブックの管理は、以下の AWS リージョンで利用できます。

リージョン	コード
米国東部(オハイオ)	us-east-2

リージョン	コード
米国東部 (バージニア北部)	us-east-1
米国西部 (北カリフォルニア)	us-west-1
米国西部 (オレゴン)	us-west-2
アジアパシフィック (東京)	ap-northeast-1
アジアパシフィック (ソウル)	ap-northeast-2
アジアパシフィック (ムンバイ)	ap-south-1
アジアパシフィック (シンガポール)	ap-southeast-1
アジアパシフィック (シドニー)	ap-southeast-2
カナダ (中部)	ca-central-1
欧州 (フランクフルト)	eu-central-1
欧州 (アイルランド)	eu-west-1
欧州 (ロンドン)	eu-west-2

AWS Glue Studio でビジュアル ETL ジョブを作成する

AWS Glue ジョブには、ソースデータに接続して処理し、データターゲットに書き出すスクリプトがカプセル化されています。通常、ジョブは、抽出、変換、ロード (ETL) スクリプトを実行します。ジョブは、Apache Spark および Ray ランタイム環境向けに設計されたスクリプトを実行できます。ジョブでは、汎用 Python スクリプト (Python シェルジョブ) を実行することもできます。AWS Glue トリガーでは、スケジュールまたはイベントに基づいて、またはオンデマンドでジョブを開始できます。ジョブ実行をモニタリングすると、完了ステータス、継続時間、開始時間などのランタイムメトリクスを知ることができます。

AWS Glue で生成されたスクリプトを使用することも、独自のスクリプトを使用することもできます。ソーススキーマとターゲット位置またはスキーマを指定すると、AWS Glue Studio コードジェネレーターで Apache Spark API (PySpark) スクリプトを自動的に作成できます。このスクリプトを出発点として使用し、目標に合わせて編集できます。

AWS Glue では出力ファイルを複数のデータ形式で書き込むことができます。各ジョブタイプは異なる出力形式をサポートしている場合があります。一部のデータ形式では、一般的な圧縮形式を記述できます。

AWS Glue コンソールにサインインする

AWS Glue でのジョブは、抽出、変換、ロード (ETL) 作業を実行するビジネスロジックで構成されます。 コンソールの [ETLAWS Glue] セクションでジョブを作成できます。

既存のジョブを表示するには、AWS Management Console にサインインし、AWS Glue コンソール (https://console.aws.amazon.com/glue/) を開きます。その後、 の [ジョブAWS Glue] タブを選択します。[Jobs] (ジョブ) リストは、ジョブが最後に変更されたとき各ジョブに関連付けられたスクリプトの場所、および現在のジョブのブックマークオプションを表示します。

新しいジョブの作成中またはジョブの保存後、can AWS Glue Studio を使用して、ETL ジョブを変更できます。これを行うには、ビジュアルエディタでノードを編集するか、デベロッパーモードでジョブスクリプトを編集します。ビジュアルエディタでノードを追加および削除して、より複雑な ETL ジョブを作成することもできます。

AWS Glue Studio でジョブを作成するための次のステップ

ジョブのノードを設定するには、ビジュアルジョブエディタを使用します。各ノードは、ソース位置からのデータの読み取り、データへの変換の適用などのアクションに対応しています。ジョブに追加する各ノードには、データの場所または変換に関する情報を指定するためのプロパティがあります。

ジョブを作成および管理するための次のステップは、以下のとおりです。

- AWS Glue Studio でビジュアル ETL ジョブを開始する
- ジョブスクリプトの表示
- ジョブのプロパティを変更する
- ジョブの保存
- ジョブの実行の開始
- 最近のジョブの実行の情報を表示する
- ジョブモニタリングダッシュボードにアクセスする

AWS Glue Studio でビジュアル ETL ジョブを開始する

AWS Glue Studio ではシンプルなビジュアルインターフェイスを使用して、ETL ジョブを作成できます。新しいジョブは、[Jobs] (ジョブ) ページを使用して作成します。また、スクリプトエディタを使用して、AWS Glue Studio ETL ジョブスクリプト内のコードを直接操作することも可能です。

AWS Glue Studio または AWS Glue で作成したすべてのジョブは、[Jobs] (ジョブ) ページで確認することができます。このページでは、ジョブを表示、管理、および実行できます。

AWS Glue Studio で ETL ジョブを作成する方法の別の例については、ブログチュートリアルも参照してください。

AWS Glue Studio でのジョブの開始

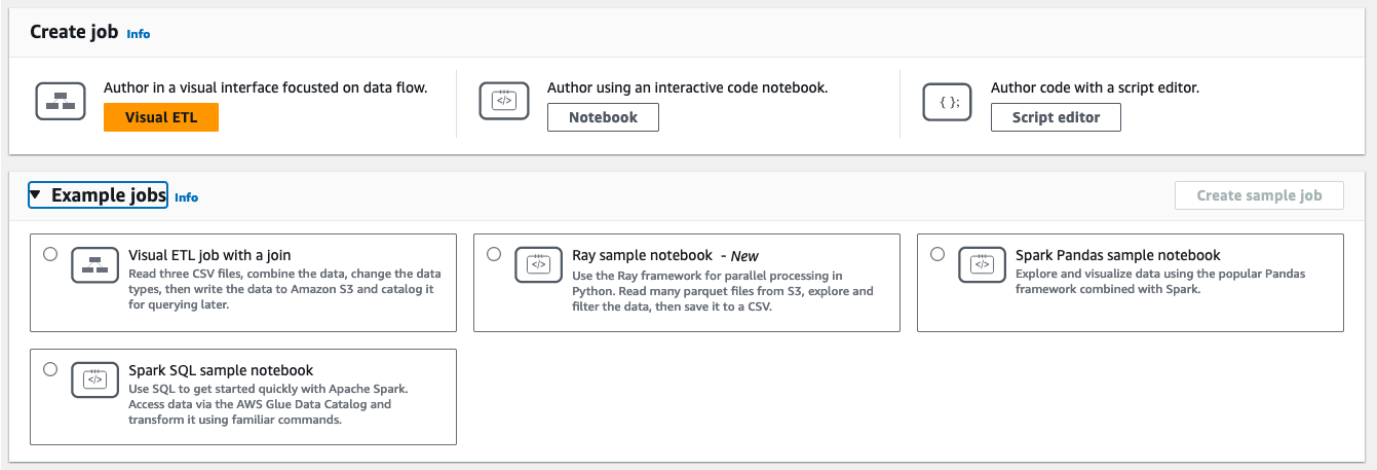
AWS Glue では、ビジュアルインターフェイスやインタラクティブなコードの Notebook を通じて、またはスクリプトエディタを使用してジョブを作成できます。オプションのいずれかをクリックしてジョブを開始することも、サンプルジョブに基づいて新しいジョブを作成することもできます。

サンプルジョブでは、選択したツールでジョブが作成されます。例えば、サンプルジョブを使用すると、CSV ファイルをカタログテーブルに結合するビジュアル ETL ジョブを作成したり、Pandas を操作するときに AWS Glue for Ray または AWS Glue for Spark を使用してインタラクティブな

コードの Notebook でジョブを作成したり、SparkSQL を使用してインタラクティブなコードの Notebook でジョブを作成したりできます。

AWS Glue Studio でゼロからのジョブの作成

1. AWS Management Console にサインインして、AWS Glue Studio コンソール (https://console.aws.amazon.com/gluestudio/) を開きます。
2. ナビゲーションペインで、[ETL ジョブ] を選択します。
3. [ジョブを作成する] セクションで、ジョブの設定オプションを選択します。



ジョブを最初から作成するためのオプション:

- ビジュアル ETL – データフローに重点を置いたビジュアルインターフェイスでの作成
- インタラクティブなコードの Notebook を使用して作成 – Jupyter Notebook に基づいて、Notebook インターフェイスでジョブをインタラクティブに作成

  このオプションを選択した場合、Notebook 作成セッションを作成する前に、追加情報を指定する必要があります。この情報の指定方法の詳細については、AWS Glue Studio 中でのノートブックの使用開始 を参照してください。
- スクリプトエディタを使用してコードを作成 – プログラミングと ETL スクリプトの記述をよく知っている場合には、このオプションを選択して、新しい Spark ETL ジョブを作成します。エンジン (Python シェル、Ray、Spark (Python)、または Spark (Scala)) を選択します。次に、[新規に開始] または [スクリプトをアップロード] を選択し、ローカルファイルから既存のスクリプトをアップロードします。スクリプトエディタの使用を選択した場合は、ビジュアルジョブエディタを使用してジョブを設計または編集することはできません。

Spark ジョブは、AWS Glue によって管理される Apache Spark 環境で実行されます。デフォルトでは、新しいスクリプトは Python でコーディングします。新しい Scala スクリプトを作成する場合は、「AWS Glue Studio 中の Scala スクリプトの作成および編集」を参照してください。

AWS Glue Studio で、サンプルジョブからジョブの作成

サンプルジョブからジョブを作成できます。[サンプルジョブ] セクションでオプションを選択し、[サンプルジョブの作成] を選択します。いずれかのオプションから作成したサンプルジョブを手軽なテンプレートとして作業を開始できます。

1. AWS Management Console にサインインして、AWS Glue Studio コンソール (https://console.aws.amazon.com/gluestudio/) を開きます。
2. ナビゲーションペインで、[ETL ジョブ] を選択します。
3. サンプルジョブからジョブを作成するためのオプションを次の中から選択します。
   - 複数のソースを結合するビジュアル ETL ジョブ – 3 つの CSV ファイルを読み取り、データを結合し、データ型を変更し、データを Amazon S3 に書き込み、後でクエリできるようにカタログ化します。
   - Pandas を使用した Spark Notebook – 一般的な Pandas フレームワークと Spark を組み合わせてデータを探索および可視化します。
   - SQL を使用した Spark Notebook – SQL を使用して Apache Spark をすぐに開始できます。AWS Glue データカタログからデータにアクセスし、使い慣れたコマンドを使用してデータを変換します。
4. [サンプルジョブの作成] を選択します。

ジョブエディタの機能

ジョブエディタでは、ジョブの作成と編集のための次の機能が提供されています。

- ジョブの視覚的な図。各ジョブタスクのノード (データを読み取るデータソースノード、データを修正する変換ノード、データを書き込むデータターゲットノード) が示されます。

  ジョブ図では、各ノードのプロパティを表示および設定できます。また、各ノードのスキーマとサンプルデータを表示することもできます。これらの機能により、ジョブを実行することなく、ジョブでデータが正しい方法で変更および変換されていることを確認できます。

- [Script viewing and editing] (スクリプトの表示と編集) タブ。ジョブ用に生成されたコードを変更できます。
- [Job details] (ジョブの詳細) タブ。さまざまな設定を行い、AWS Glue ETL ジョブの実行環境をカスタマイズできます。
- [Runs] (実行) タブ。現在実行中のジョブや以前実行したジョブの表示、ジョブの実行のステータスの表示、ジョブの実行用のログへのアクセスを行えます。
- [データ品質] タブでは、データ品質ルールをジョブに適用できます。
- [Schedules] (スケジュール) タブ。ジョブの開始時間を設定したり、ジョブの実行を定期的に設定できます。
- バージョン管理タブ。ジョブで使用する Git サービスを設定できます。

ビジュアルジョブエディタでスキーマのプレビューを使用する

ジョブの作成中または編集中に、[Output schema] (出力スキーマ) タブをクリックして、データのスキーマを表示できます。

スキーマを表示するには、ジョブエディタにデータソースへのアクセス許可が必要です。エディタの [Job details] (ジョブの詳細) タブ、またはノードの [Output schema] (出力スキーマ) タブで IAM ロールを指定できます。データソースへのアクセスに必要なすべてのアクセス許可が IAM ロールにある場合は、ノードの [Output schema] (出力スキーマ) タブのスキーマを表示できます。

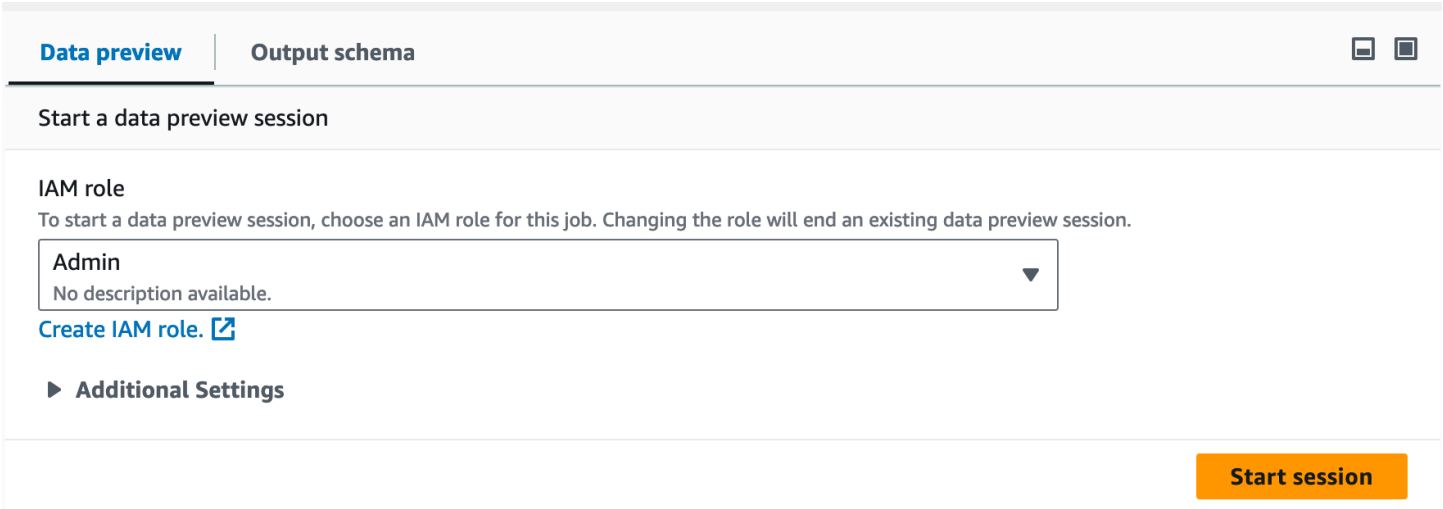
ビジュアルジョブエディタでデータのプレビューを使用する

データのプレビューを使用することで、ジョブを繰り返し実行することなく、データのサンプルを使用してジョブを作成およびテストできます。データプレビューを使用すると、次を実行できます。

- IAM ロールをテストして、データソースまたはデーターゲットにアクセスできることを確認します。
- 変換により、意図した方法でデータが変更されていることを確認します。例えば、フィルター変換を使用する場合、フィルターでデータの適切なサブセットが選択されていることを確認できます。
- データを確認する。データセットに複数のタイプの値を持つ列が含まれている場合、データのプレビューには、これらの列のタプルのリストが表示されます。各タプルには、データ型とその値が含まれています。

> **Note**
>
> データプレビューセッションおよびカスタム SQL またはカスタムコードノードを使用する場合、データプレビューセッションはデータセット全体に SQL またはコードブロックをそのまま実行します。

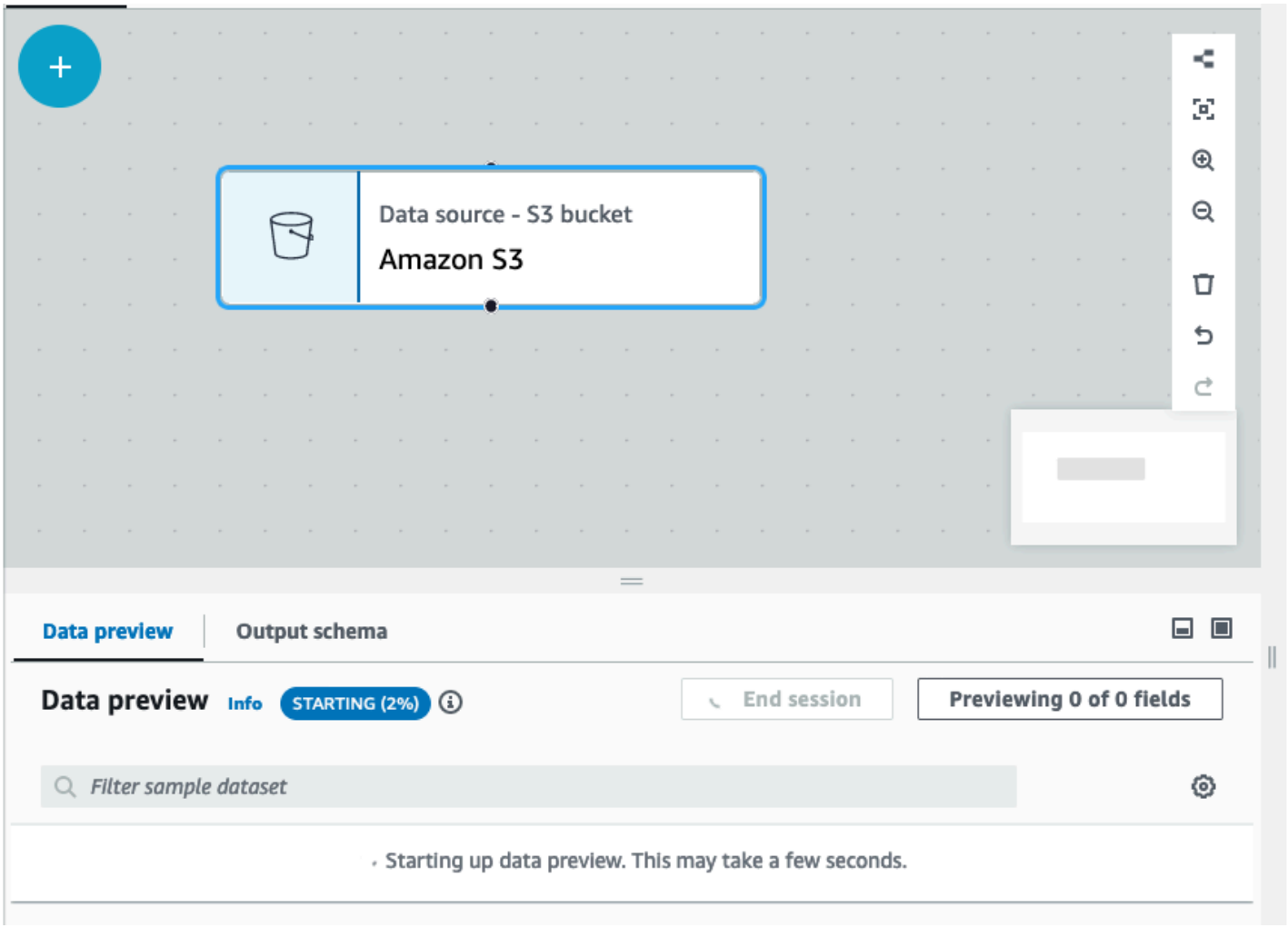
ジョブの作成中または編集中に、[Data preview] タブをクリックして、データのサンプルを表示できます。ジョブでロールが既に設定されている場合、またはアカウントでデフォルトの IAM ロールが設定されている場合、新しいデータプレビューセッションが自動的に開始されます。ロールが事前に設定されていない場合は、ロールを選択してセッションを開始できます。



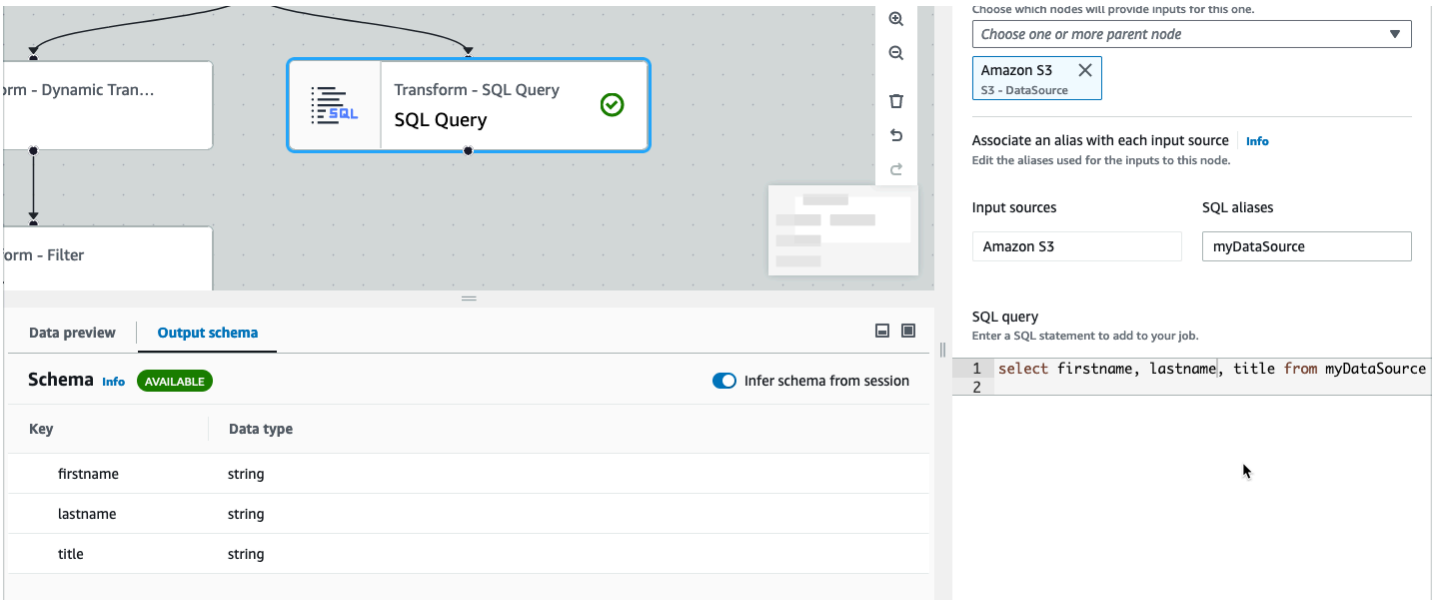
> **Note**
>
> データプレビューセッション用に選択したロールは、ジョブにも使用されます。

情報アイコンをクリックすると、セッションのステータスと進行状況、およびセッションの詳細を確認できます。

セッションの準備が完了すると、AWS Glue Studio は選択したノードのためにデータをロードします。進捗に伴う [完了率 (%)] を表示できます。



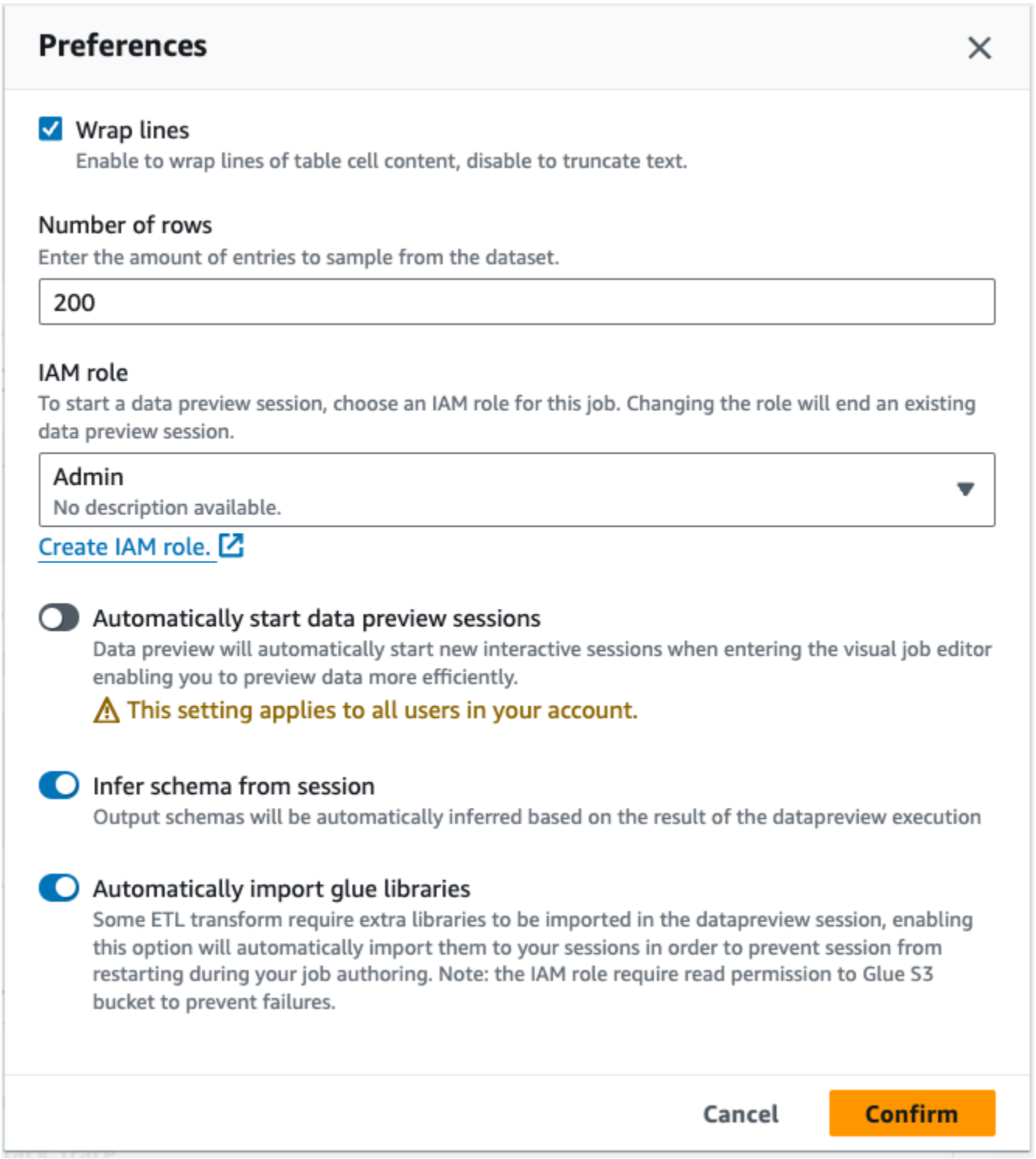
ビジュアルジョブを作成する際に、[出力スキーマ] タブで [セッションからスキーマを推測] を切り替えると、AWS Glue Studio は選択したノードのスキーマを自動的に更新します。



データプレビューの設定を行うには:

設定アイコン (歯車の記号) を選択して、データのプレビューの設定を行えます。これらの設定は、ジョブ図のすべてのノードに適用されます。次のようにできます。

- テキストを 1 行から次の行に折り返す場合に選択します。デフォルトでは、このオプションは有効になっています
- 行数を変更する (デフォルトは 200)
- IAM ロールを選択するか、または、必要であれば新しいロールを作成
- ジョブの作成時に新しいセッションを自動的に開始することを選択します。これにより、ジョブの作成時に新しいインタラクティブセッションがプロビジョニングされます。この設定はアカウントレベルで適用されます。一度設定すると、ジョブの編集時にアカウント内のすべてのユーザーに適用されます。
- スキーマを自動的に推論する場合に選択します。出力スキーマは、選択したノードのために自動的に推論されます
- AWS Glue ライブラリを自動的にインポートする場合に選択します。これは、セッションの再起動を必要とする新しい変換を追加する際に、データプレビューが新しいセッションを再起動するのを防ぐため、有益です。

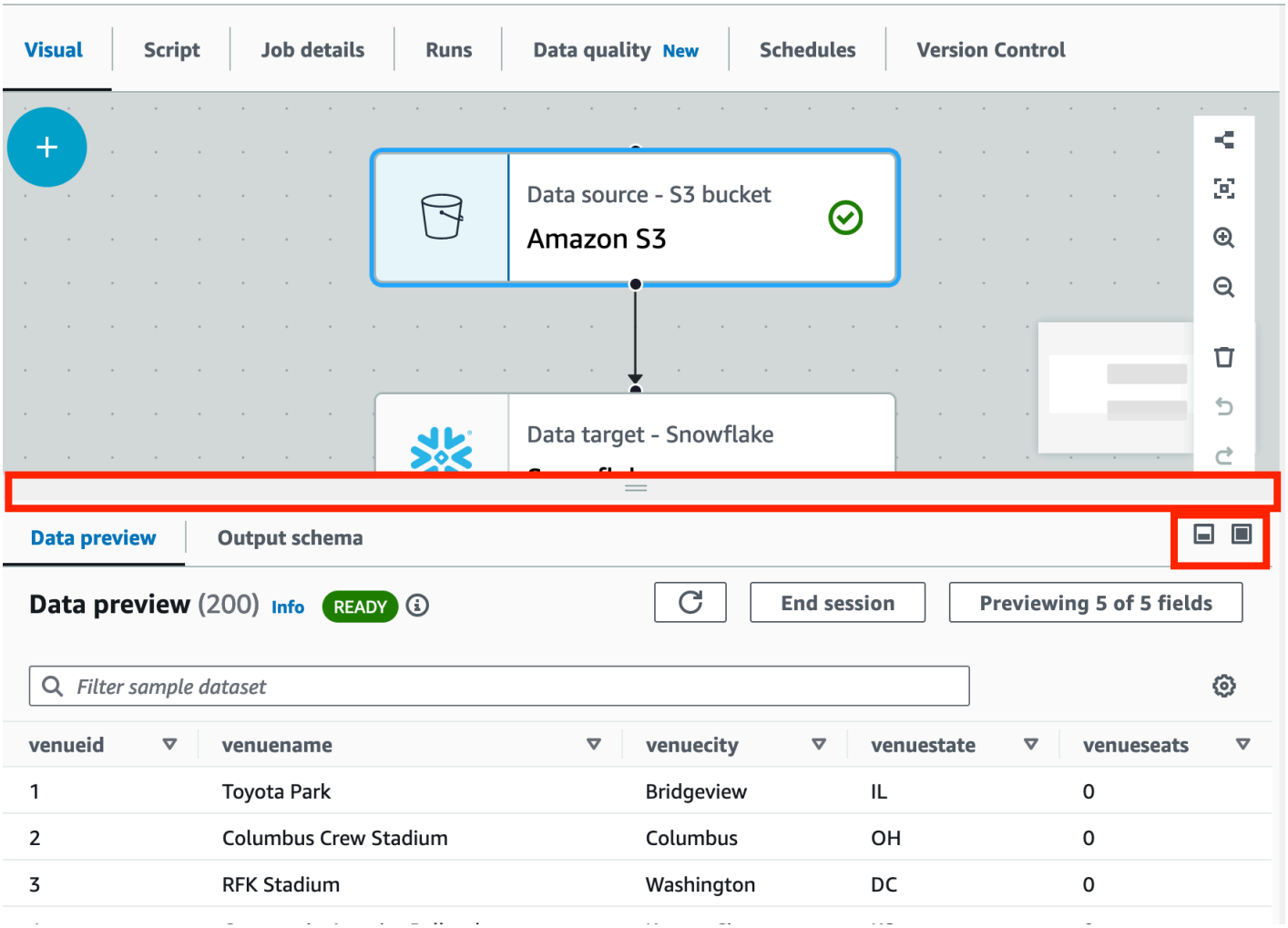


追加機能には次の機能が含まれます。

- [Previewing x of y fields] (X/Y フィールドのプレビュー) ボタンをクリックして、プレビューする列 (フィールド) を選択できます。デフォルトの設定を使用してデータをプレビューすると、ジョブ

エディタにはデータセットの最初の 5 列が表示されます。これを、[show all] (すべて表示) または [none] (なし) (非推奨) に変更できます。

- データのプレビューウィンドウは、水平方向と垂直方向の両方にスクロールします。
- 最大化ボタンを使用して、[Data preview] タブをジョブグラフ画面全体に展開し、データおよびデータ構造をより詳しく表示できます。同様に、最小化ボタンを使用して [Data preview] タブを最小化します。ハンドルペインをつかんで上にドラッグして [Data preview] タブを展開することもできます。



- [セッションの終了] を使用してデータプレビューを停止します。セッションを停止すると、新しい IAM ロールを選択し、新しいセッションを自動的に開始したり、スキーマを推測したり、AWS Glue ライブラリをインポートしたりするための追加設定 (オン/オフなど) を設定して、セッションを再開できます。

データのプレビューを使用する際の制限事項

データのプレビューを使用する際、次の制約または制限事項が発生する場合があります。

- 初めて [Data preview] (データのプレビュー) タブを選択する際、IAM ロールを選択する必要があります。このロールには、そのデータおよびデータのプレビューの作成に必要なその他のリソースへのアクセス許可が必要です。
- IAM ロールを指定した後、データを表示できるようになるまでには時間がかかります。データが 1 GB 未満のデータセットの場合、最大で 1 分かかることがあります。大きなデータセットがある場合は、パーティションを使用してロード時間を改善する必要があります。Amazon S3 からデータを直接ロードすると、パフォーマンスが最も良くなります。
- 非常に大きなデータセットがあり、データのプレビュー用のデータのクエリに 15 分以上かかる場合、リクエストはタイムアウトします。データプレビューには 30 分の IDLE タイムアウトがあります。これを軽減するために、データセットのサイズを小さくして、データプレビューを使用できます。
- デフォルトでは、[Data preview] タブには最初の 50 列が表示されます。列にデータの値がない場合は、表示するデータはありませんというメッセージが表示されます。サンプリングされる行数を増やすか、別の列を選択してデータの値を表示できます。
- 現在、データのプレビューは、ストリーミングデータソースまたはカスタムコネクタを使用するデータソースではサポートされていません。
- 1 つのノードでのエラーは、ジョブ全体に影響します。いずれかのノードでデータのプレビューのエラーがある場合、エラーは修正されるまですべてのノードに表示されます。
- ジョブのデータソースを変更する場合、そのデータソースの子ノードは、新しいスキーマに合わせて更新する必要があります。例えば、列を変更する ApplyMapping ノードがあり、その列が代替データソースに存在しない場合、ApplyMapping 変換ノードを更新する必要があります。
- SQL クエリ変換ノードの [Data preview] (データのプレビュー) タブを表示する際に SQL クエリで誤ったフィールド名が使用されている場合、[Data preview] (データのプレビュー)タブにエラーが表示されます。

スクリプトコードの生成

ビジュアルエディタを使用してジョブを作成する場合、ETL コードが自動的に生成されます。AWS Glue Studio は機能的で完全なジョブスクリプトを作成し、Amazon S3 の場所に保存します。

AWS Glue Studio によって生成されるコードには 2 つの形式があります。: オリジナルバージョン、またはクラシックバージョン、およびより新しいストリームラインされたバージョンです。デフォル

トでは、新しいコードジェネレーターがジョブスクリプトの作成に使用されます。ジョブスクリプトは、[Generate classic script]トグルボタンを選択することにより、[Script]タブ上のクラシックコードジェネレーターを使用して生成できます。

生成されたコードの新しいバージョンでは、いくつかの違いを含みます。:

- 大きなコメントブロックは、もうスクリプトに追加されなくなりました。
- コード内の出力のデータ構造は、ビジュアルエディタで指定したノード名を使用します。クラススクリプトでは、出力のデータ構造は単に`DataSource0`,`DataSource1`,`Transform0`,`Transform1`,`DataSink0`,`DataSink1` などと名前が付けられます。
- 長いコマンドは複数の行に分割されるため、ページ全体をスクロールしてコマンド全体を表示する必要がなくなります。

AWS Glue Studio 中の新機能には新しいバージョンのコード生成が必要で、クラシックコードスクリプトでは動作しません。これらのジョブを実行しようとすると、これらのジョブを更新するように求められます。

AWS Glue マネージド変換によるデータの変換

AWS Glue Studio には、次の 2 種類の変換が用意されています。

- AWS Glue ネイティブ変換 - すべてのユーザーが利用でき、AWS Glue によって管理されます。
- カスタムビジュアル変換 - 自分のカスタムビジュアル変換をアップロードして、AWS Glue Studio で使用できます。

AWS Glue マネージドデータ変換ノード

AWS Glue Studio には、データを処理するために使用できる一式の組み込み変換が用意されています。データは、ジョブ図内のあるノードから `DynamicFrame` と呼ばれるデータ構造の別のノードに渡されます。これは、Apache Spark SQL `DataFrame` の拡張機能です。

事前設定されたジョブ図では、データソースとデーターゲットノード間で、スキーマ変更変換ノードを使用できます。この変換ノードを設定すると、データを修正したり、他の変換を使用できます。

AWS Glue Studio を用いて、次のbuilt-in 変換が利用可能です。

- ChangeSchema: データソースのデータプロパティキーを、データターゲットのデータプロパティキーにマッピングします。キーの名前を変更したり、データ型を変更したり、データセットから削除するキーを選択できます。
- SelectFields: 保持したいデータプロパティキーを選択します。
- DropFields: 削除したいデータプロパティキーを選択します。
- RenameField: 単一のデータプロパティキーの名前を変更します。
- Spigot:Amazon S3 バケットにデータのサンプルを書き込みます。
- Join: 指定したデータプロパティキー上の比較フレーズを使用して、2 つのデータセットを 1 つに結合します。結合タイプは、内部結合、外部結合、左結合、右結合、左半結合、左反結合を使用できます。
- Union: 同じスキーマを持つ複数のデータソースの行を結合します。
- SplitFields: データプロパティキーを 2 つの DynamicFrames に分割します。出力は DynamicFrames のコレクションです。一方は選択したデータプロパティキー、他方は残っている方のデータプロパティキーを持ちます。
- SelectFromCollection： DynamicFrame のコレクションから を 1 つの DynamicFrames を選択します。出力は選択された DynamicFrame です。
- [FillMissingValues]：欠落値があるデータセットの記録を配置し、代入によって決定された提案値を用いた新しいフィールドを追加します。
- Filter: フィルター条件に基づいて、データセットを 2 つに分割します。
- [Drop Null Fields]: 列のすべての値が「null」の場合、データセットから列を削除します。
- [Drop Duplicates]: 行全体を照合するか、キーを指定するかを選択して、データソースから行を削除します。
- SQL:SparkSQL コードをテキスト入力フィールドに入力して、データを変換するために SQL クエリを使用します。出力は、単一の DynamicFrame です。
- Aggregate: 選択したフィールドと行で計算 (平均、合計、最小、最大など) を実行し、計算された値を使って新しいフィールドを作成します。
- Flatten: Struct 内のフィールドを最上位のフィールドに抽出します。
- UUID: 各行にユニバーサル一意識別子を含む列を追加します。
- Identifier: 各行に数値識別子を含む列を追加します。
- To timestamp: 列をタイムスタンプタイプに変換します。
- Format timestamp: タイムスタンプ列をフォーマットされた文字列に変換します。

- Conditional Router transform: 受信データに複数の条件を適用します。受信データの行はそれぞれグループフィルター条件により評価され、対応するグループに取り入れて処理されます。
- [列の連結] 変換: オプションのスペーサーを使用して、他の列の値を使った新しい文字列の列を作成します。
- [文字列の分割] 変換: 正規表現を使用して文字列をトークンの配列に分割し、分割方法を定義します。
- [配列から列へ] 変換: 配列タイプの列の一部またはすべての要素を抽出し、新しい列に追加します。
- [現在のタイムスタンプを追加] 変換: データが処理された時刻で行をマークします。これは、監査目的やデータパイプラインでのレイテンシーの追跡に役立ちます。
- [行から列へのピボット] 変換: 選択した列の固有の値を回転させて数値列を集約し、新しい列にします。複数の列を選択した場合、値が連結されて新しい列に名前が付けられます。
- [列から行へのピボット解除] 変換: 列を新しい列の値に変換し、固有の値ごとに行を生成します。
- [オートバランス処理] 変換: データをワーカー間でより適切に再配分します。これは、データのバランスが取れていない場合や、ソースから取得したデータでは十分に並行処理ができない場合に役立ちます。
- [派生列] 変換: 数式または SQL 式に基づいて新しい列を定義します。これらの式では、データ内の他の列および定数やリテラルを使用できます。
- [ルックアップ] 変換: キーがデータ内の定義済みのルックアップ列と一致する場合、定義済みのカタログテーブルから列を追加します。
- [配列またはマップを行に分解] 変換: ネストされたデータ構造から値を抽出し、操作しやすい個々の行に値を追加します。
- レコードマッチング変換: 既存のレコードマッチング機械学習データ分類変換を呼び出します。
- null 行変換を削除: すべての列が null または空の行をデータセットから削除します。
- JSON 列変換の解析: JSON データを含む文字列の列を解析し、JSON がオブジェクトか配列かに応じて、それぞれを構造体または配列の列に変換します。
- JSON パス変換の抽出: JSON 文字列の列から新しい列を抽出します。
- 正規表現から文字列の断片を抽出: 正規表現を使用して文字列の断片を抽出し、そこから新しい列を作成します。正規表現グループを使用する場合は、複数の列を作成します。
- Custom transform: カスタム transforms を使用するために、テキスト入力フィールドにコードを入力します。出力は、`DynamicFrames` のコレクションです。

AWS Glue Studio でのデータ準備レシピの使用

[データ準備レシピ] 変換を使用すると、インタラクティブなグリッドスタイルオーサリングインターフェイスを使用して、データ準備レシピを最初から作成できます。また、既存の AWS Glue DataBrew レシピをインポートし、AWS Glue Studio で編集することもできます。

データ準備レシピノードは、リソースパネルから使用できます。データ準備レシピノードは、データソースノードであるか別の変換ノードであるかに関らず、ビジュアルワークフロー内の別のノードに接続できます。AWS Glue DataBrew レシピとバージョンを選択すると、レシピに適用されたステップがノードのプロパティタブに表示されます。

前提条件

- AWS Glue DataBrew レシピをインポートする場合、「AWS Glue Studio で AWS Glue DataBrew レシピをインポートする」で説明されているように、必要な IAM アクセス許可があります。
- データプレビューセッションを作成する必要があります。

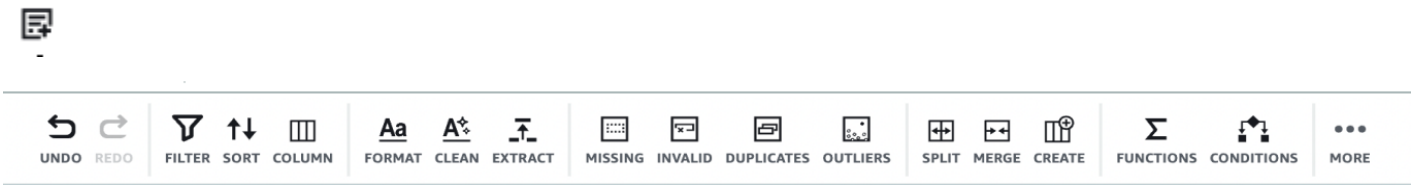
制約事項

- AWS Glue DataBrew レシピは、商用 DataBrew リージョンでのみサポートされています。
- すべての AWS Glue DataBrew レシピが AWS Glue でサポートされているわけではありません。一部のレシピは AWS Glue Studio では実行できません。
  - `UNION` および `JOIN` 変換を使用するレシピはサポートされていませんが、AWS Glue Studio には、データ準備レシピノードの前後で使用できる「Union」および「Join」変換ノードが既に存在します。
- データ準備レシピノードは AWS Glue バージョン 4.0 以降のジョブでサポートされています。このバージョンは、データ準備レシピノードがジョブに追加されると自動選択されます。
- データ準備レシピノードには Python が必要です。Python は、データ準備レシピノードがジョブに追加されると自動設定されます。
- ビジュアルグラフに新しいデータ準備レシピノードを追加すると、データ準備レシピノードを使用する正しいライブラリでデータプレビューセッションが自動的に再開されます。
- データ準備レシピノードでのインポートまたは編集では、`GROUP_BY`、`PIVOT`、`UNPIVOT`、および `TRANSPOSE` の変換はサポートされていません。

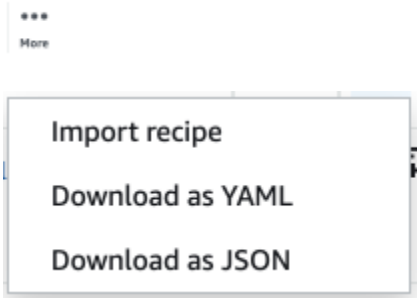
その他の機能

[データ準備レシピ] 変換を選択すると、[レシピの作成] を選択した後に追加のアクションを実行できます。

- ステップの追加 – ステップの追加アイコンを選択するか、[プレビュー] ペインのツールバーを使用してアクションを選択して、必要に応じてレシピにステップを追加できます。



- レシピのインポート — [その他] を選択し、AWS Glue Studio ジョブで使用するレシピをインポートします。



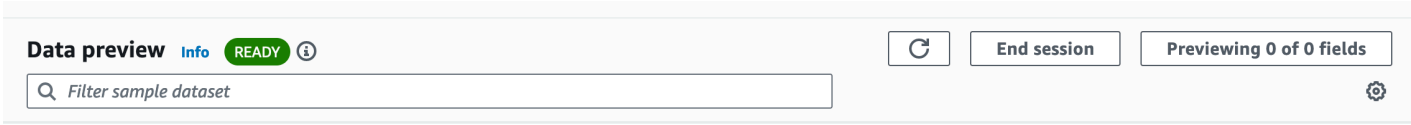
- YAML としてダウンロード – [その他]、次に [YAML としてダウンロード] を選択してレシピをダウンロードし、AWS Glue Studio の外部に保存します。
- JSON としてダウンロード – [その他]、次に [JSON としてダウンロード] を選択してレシピをダウンロードし、AWS Glue Studio の外部に保存します。
- レシピステップの元に戻すとやり直す – グリッドでデータを操作するときに、[プレビュー] ペインでレシピステップを元に戻したりやり直したりすることができます。



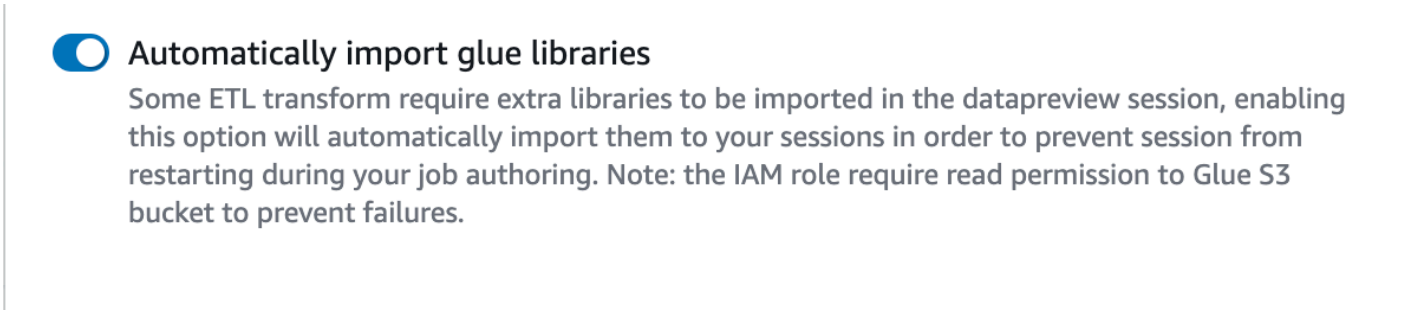
ビジュアル ETL AWS Glue ジョブにおけるデータ準備レシピの作成および実行

このシナリオでは、DataBrew で最初に作成しなくても、データ準備レシピを作成できます。レシピの作成を開始する前に、以下を行う必要があります。

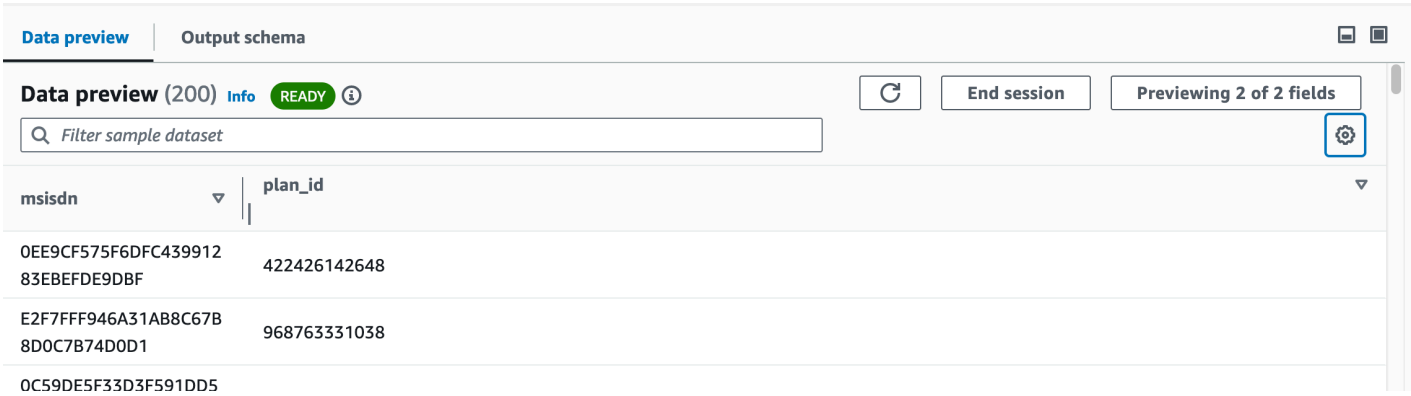
- アクティブなデータプレビューセッションが実行されている。データプレビューセッションが準備完了になると、[レシピの作成] がアクティブになり、レシピの作成または編集を開始できます。



- [Glue ライブラリを自動的にインポート] のトグルが有効になっていることを確認します。

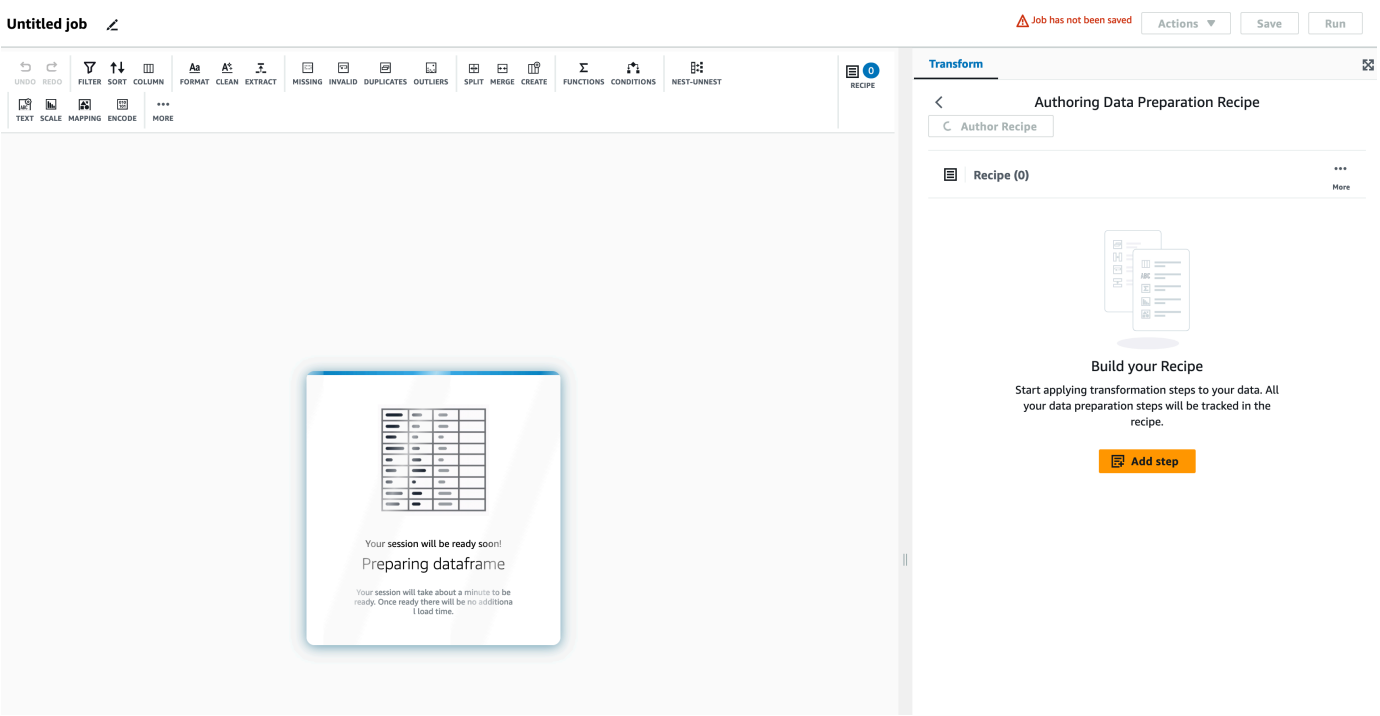


これを行うには、[データプレビュー] ペインで歯車アイコンを選択します。

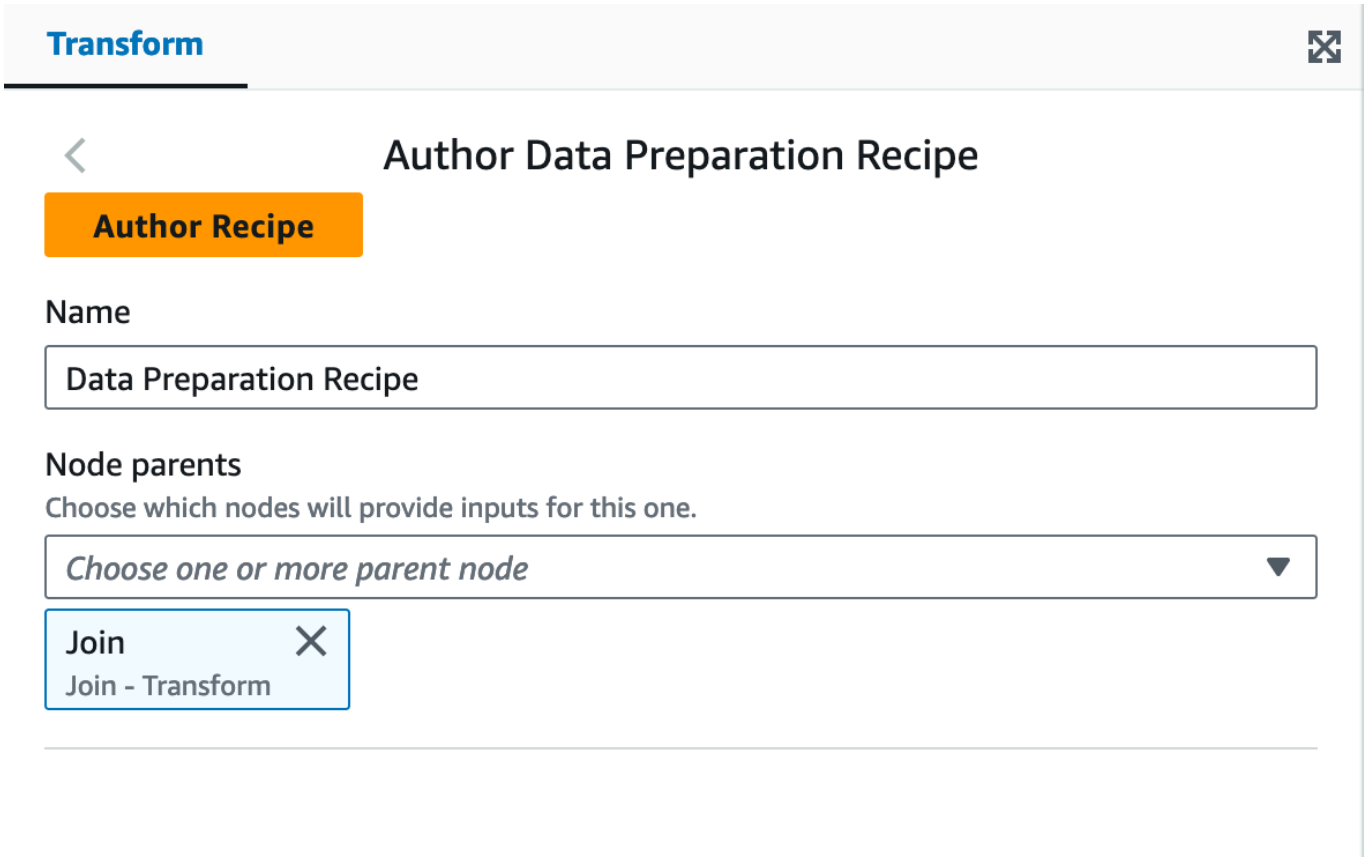


AWS Glue Studio でデータ準備レシピを作成するには:

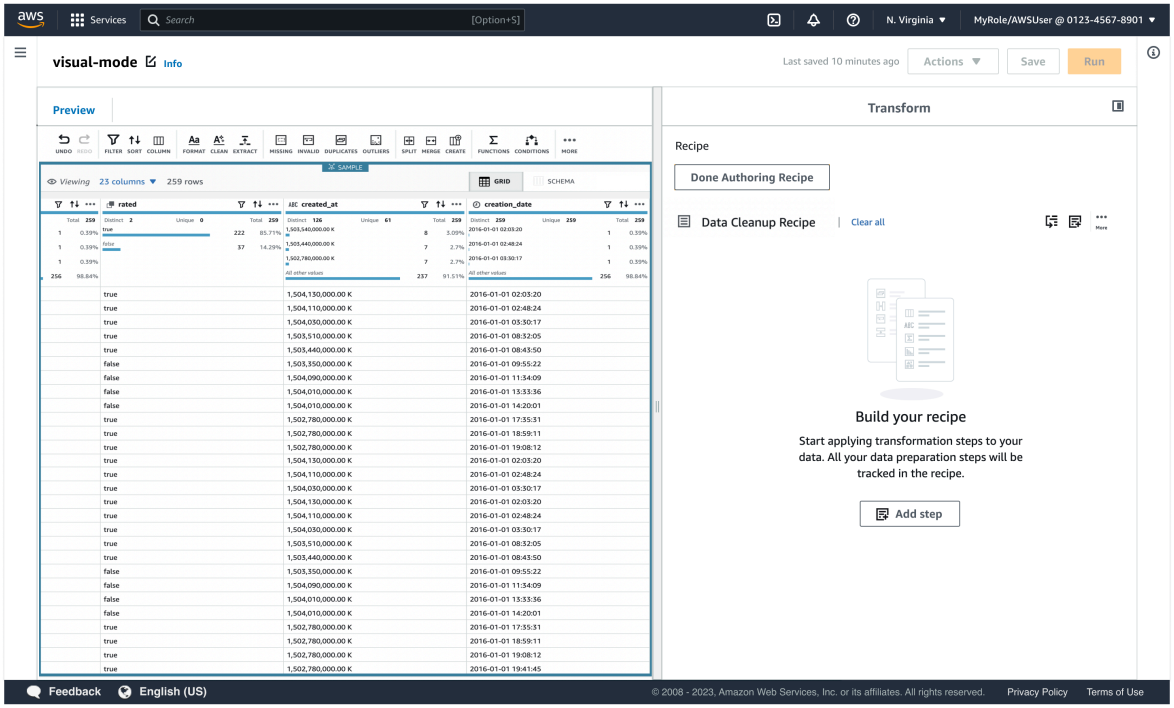
1. [データ準備レシピ] 変換をジョブキャンバスに追加します。変換はデータソースノードの親に接続する必要があります。[データ準備レシピ] ノードを追加すると、ノードが適切なライブラリで再起動し、データフレームが準備中であることがわかります。



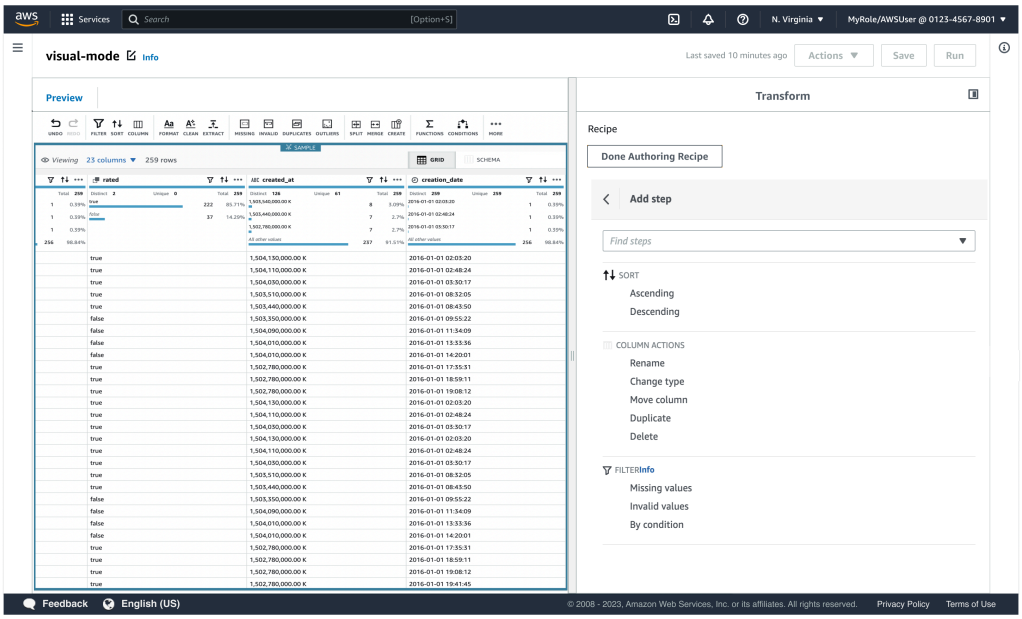
2. データプレビューセッションが準備完了になると、以前に適用されたステップを含むデータが画面の下部に表示されます。
3. [レシピの作成] を選択します。これにより、AWS Glue Studio で新しいレシピを開始できます。



4. ジョブキャンバスの右側にある [変換] パネルに、データ準備レシピの名前を入力します。
5. 左側では、キャンバスはデータのグリッドビューに置き換えられます。右側には、[変換] パネルが変化し、レシピのステップが表示されます。[ステップの追加] を選択して、レシピの最初のステップを追加します。

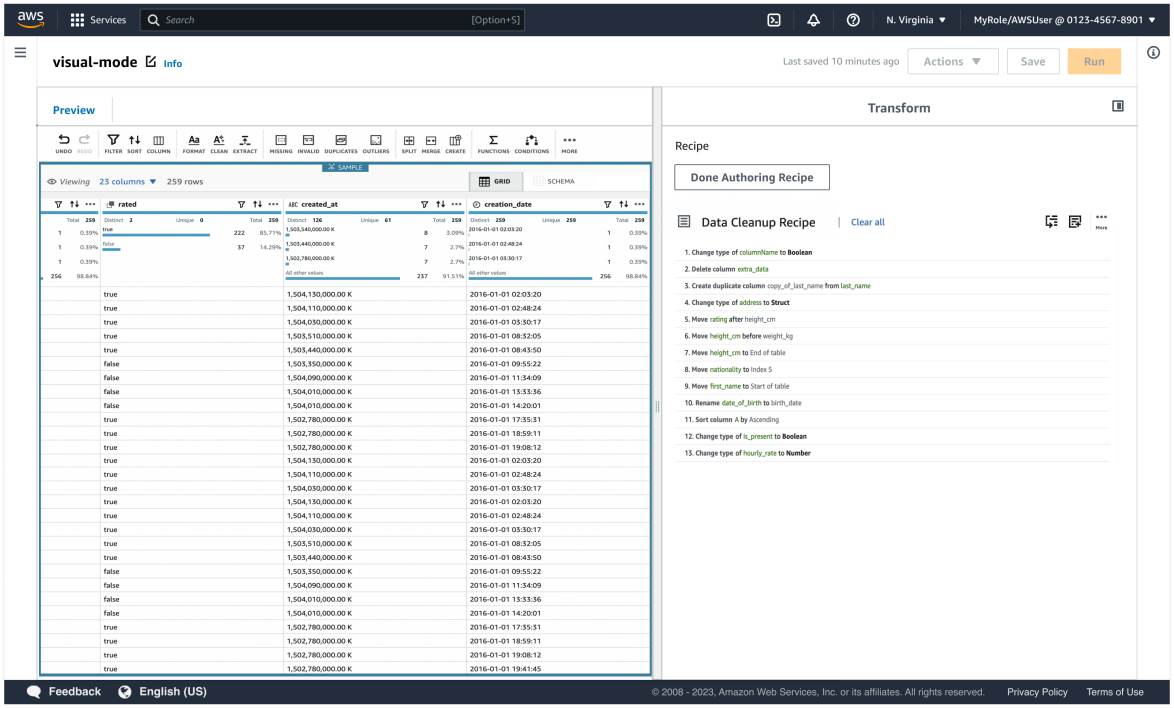


6. [変換] パネルで、ソート、列に対するアクションの実行、値のフィルタリングを選択します。例えば、[列の名前変更] を選択します。



7. 右側の [変換] パネルで、列の名前を変更するオプションを使用して、名前を変更するソース列を選択し、新しい列名を入力できます。完了したら、[適用] を選択します。

   各ステップをプレビューしたり、ステップを元に戻したり、ステップを並べ替えたり、フィルター、ソート、分割、マージなどのアクションアイコンを使用できます。データグリッドでアクションを実行すると、[変換] パネルのレシピにステップが追加されます。



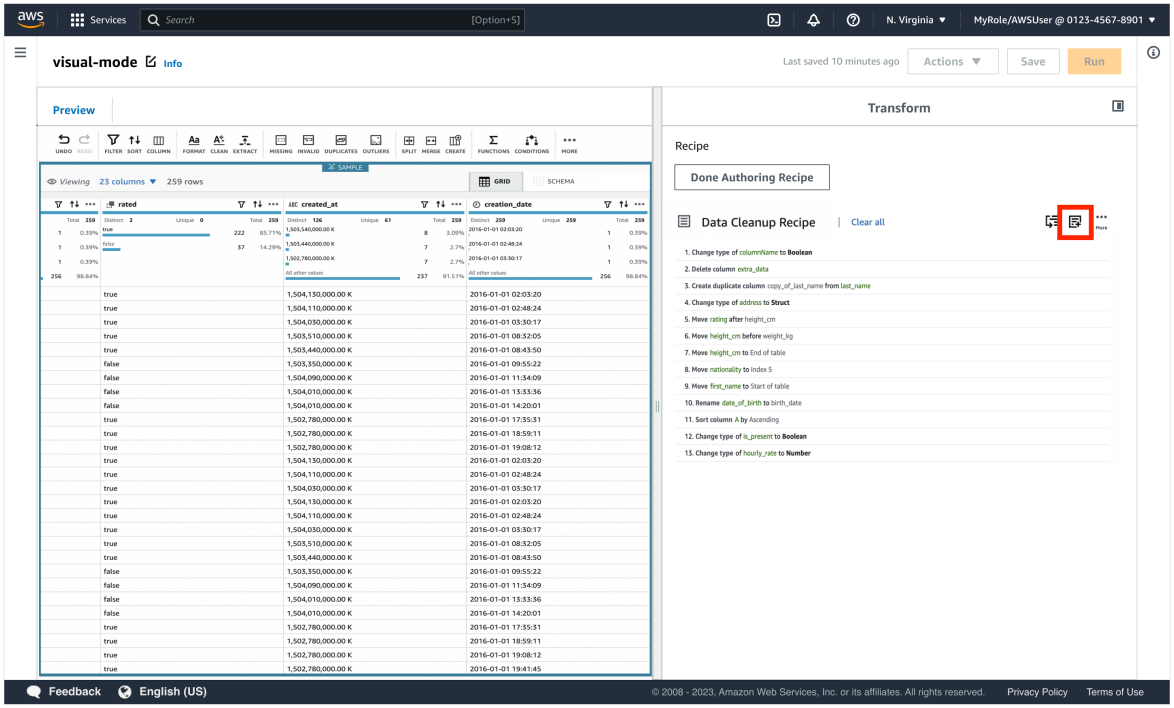
変更を加える必要がある場合は、各ステップの結果をプレビューし、ステップを元に戻し、ステップの順序を変更することで、[プレビュー] ペインでこれを行うことができます。例:

- ステップを元に戻す/やり直す – [元に戻す] アイコンを選択してステップを元に戻します。[やり直す] アイコンを選択すると、ステップを繰り返すことができます。



- ステップの順序を変更する – ステップの順序を変更すると、AWS Glue Studio は各ステップを検証し、そのステップが無効かどうかを知らせます。

8. ステップを適用すると、[変換] パネルにレシピのすべてのステップが表示されます。すべてのステップをクリアして最初からやり直したり、追加アイコンを選択してさらにステップを追加したり、[レシピの作成を完了] を選択したりできます。



9. 画面の右上にある [保存] を選択します。レシピステップは、ジョブを保存するまで保存されません。

AWS Glue Studio で AWS Glue DataBrew レシピをインポートする

AWS Glue DataBrew では、一連のデータ変換ステップをレシピといいます。AWS Glue DataBrew レシピには読み取り済みデータの変換方法は規定されていますが、データを読み取る場所と方法、およびデータを書き込む場所と方法は記載されていません。これは、AWS Glue Studio のソースノードとターゲットノードで設定します。レシピの詳細については、「Creating and using AWS Glue DataBrew recipes」を参照してください。

AWS Glue Studio で AWS Glue DataBrew レシピを使用するには、最初に AWS Glue DataBrew でレシピを作成します 使用するレシピがある場合は、このステップを省略できます。

AWS Glue DataBrew に対する IAM アクセス許可

このトピックでは、IAM 管理者がデータ準備レシピ変換の AWS Identity and Access Management (IAM) ポリシーで使用できるアクションとリソースを理解するための情報について説明します。

AWS Glue のセキュリティに関する追加情報については、「Access Management」を参照してください。

> **Note**
>
> 次の表に、既存の AWS Glue DataBrew レシピをインポートするときにユーザーが必要とするアクセス許可を示します。

データ準備レシピ変換アクション

[アクション]	説明
`databrew:ListRecipes`	AWS Glue DataBrew レシピを取得するアクセス許可を付与します。
`databrew:ListRecipeVersions`	AWS Glue DataBrew レシピバージョンを取得するアクセス許可を付与します。
`databrew:DescribeRecipe`	AWS Glue DataBrew レシピの説明を取得するアクセス許可を付与します。

この機能へのアクセスに使用するロールには、複数の AWS Glue DataBrew アクションを許可するポリシーが必要です。これを実現するには、必要なアクションを含む `AWSGlueConsoleFullAccess` ポリシーを使用するか、次のインラインポリシーをロールに追加します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "databrew:ListRecipes",
                "databrew:ListRecipeVersions",
                "databrew:DescribeRecipe"
            ],
            "Resource": [
                "*"
            ]
```

```
        }
    ]
}
```

データ準備レシピ変換を使用するには、アクセス許可ポリシーに IAM:PassRole アクションを追加する必要があります。

追加で必要な許可

[アクション]	説明
iam:PassRole	承認済みのロールをユーザーが渡すことを許可するアクセス許可を、IAM に付与します。

これらのアクセス許可がないと、次のエラーが発生します。

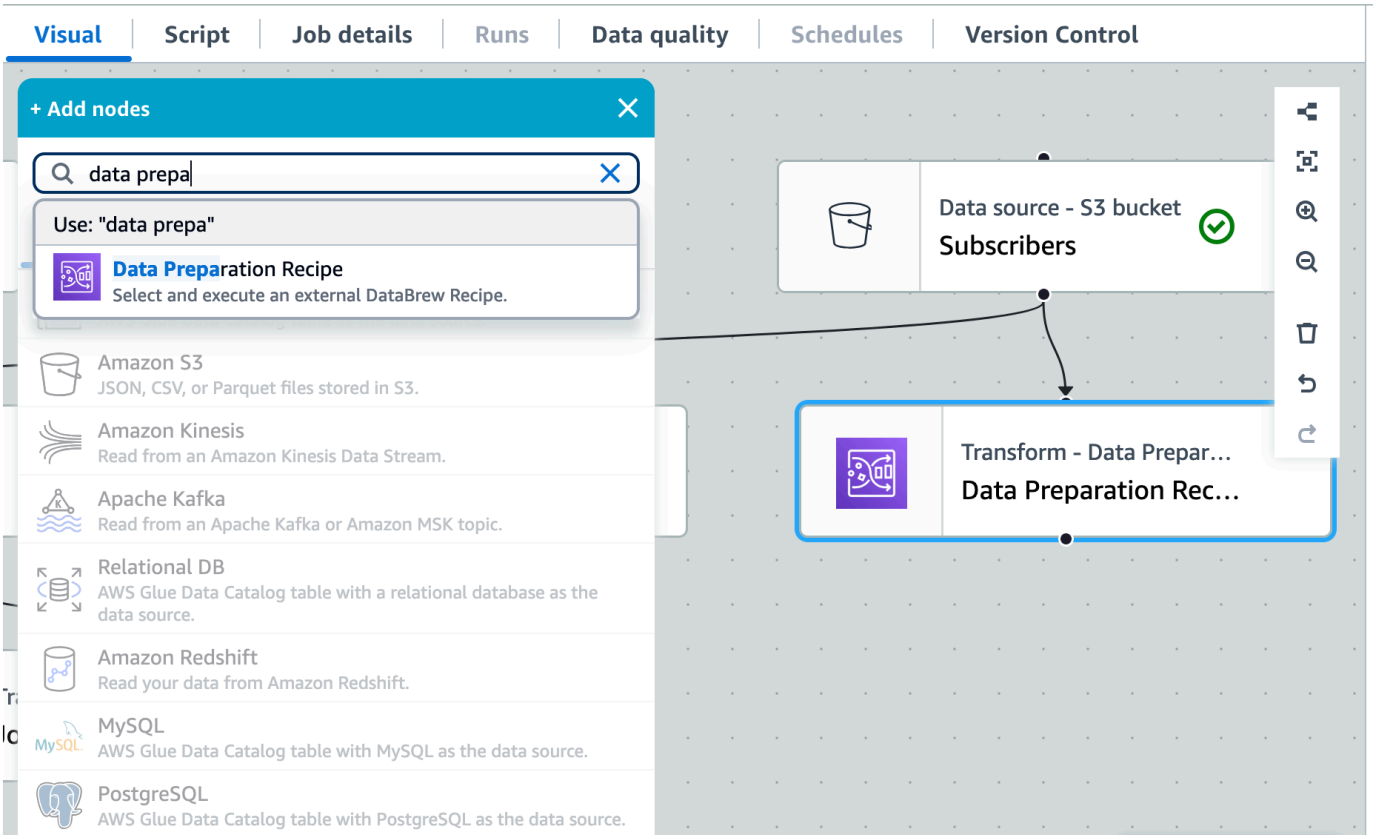
```
"errorCode": "AccessDenied"
"errorMessage": "User: arn:aws:sts::account_id:assumed-role/AWSGlueServiceRole is not
authorized to perform: iam:PassRole on resource: arn:aws:iam::account_id:role/service-
role/AWSGlueServiceRole
because no identity-based policy allows the iam:PassRole action"
```

AWS Glue DataBrew レシピのインポート

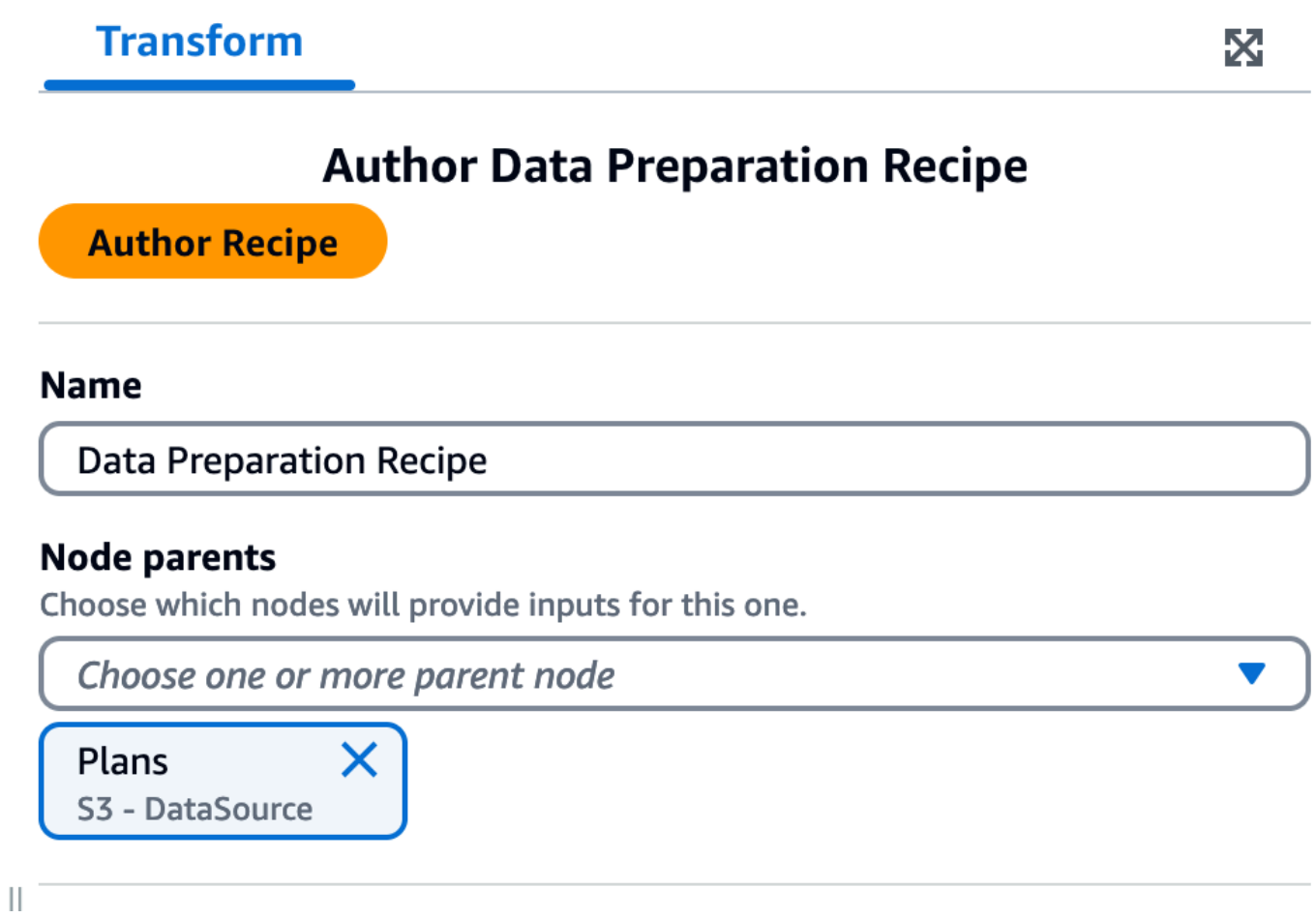
AWS Glue DataBrew レシピをインポートして AWS Glue Studio で使用するには:

既存のデータ準備レシピノードがあり、AWS Glue Studio でレシピステップを直接編集する場合は、レシピステップを AWS Glue Studio ジョブにインポートする必要があります。

1. データソースを使用して、AWS Glue Studio で AWS Glue ジョブを開始します。
2. [データ準備レシピ] ノードをジョブキャンバスに追加します。

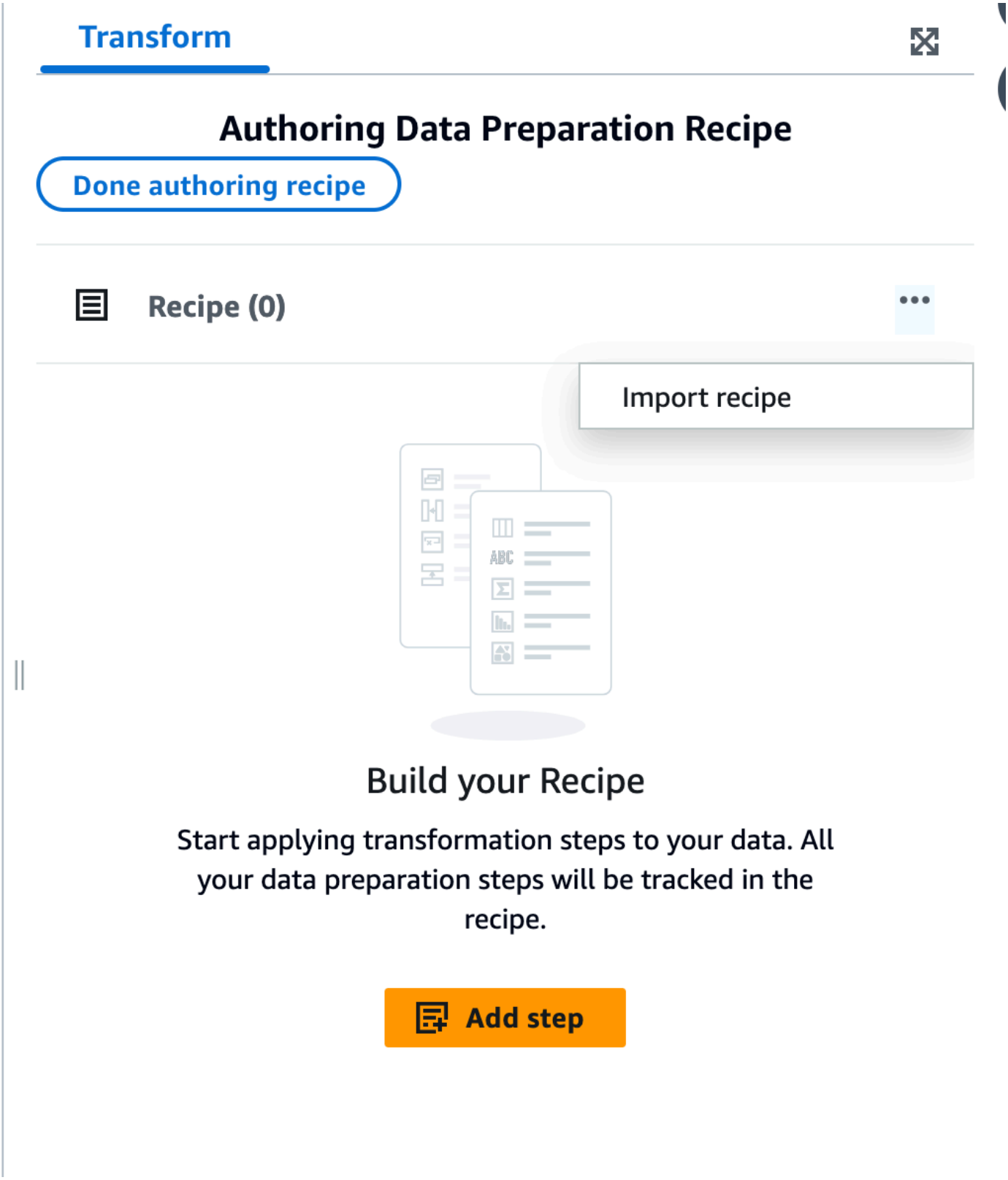


3. [変換] パネルで、レシピの名前を入力します。
4. ドロップダウンリストからキャンバスで使用可能なノードを選択して、1 つ以上の親ノードを選択します。
5. [レシピの作成] を選択します。[作成者レシピ] が灰色の場合、ノードの親が選択され、データプレビューセッションが完了するまで使用できません。

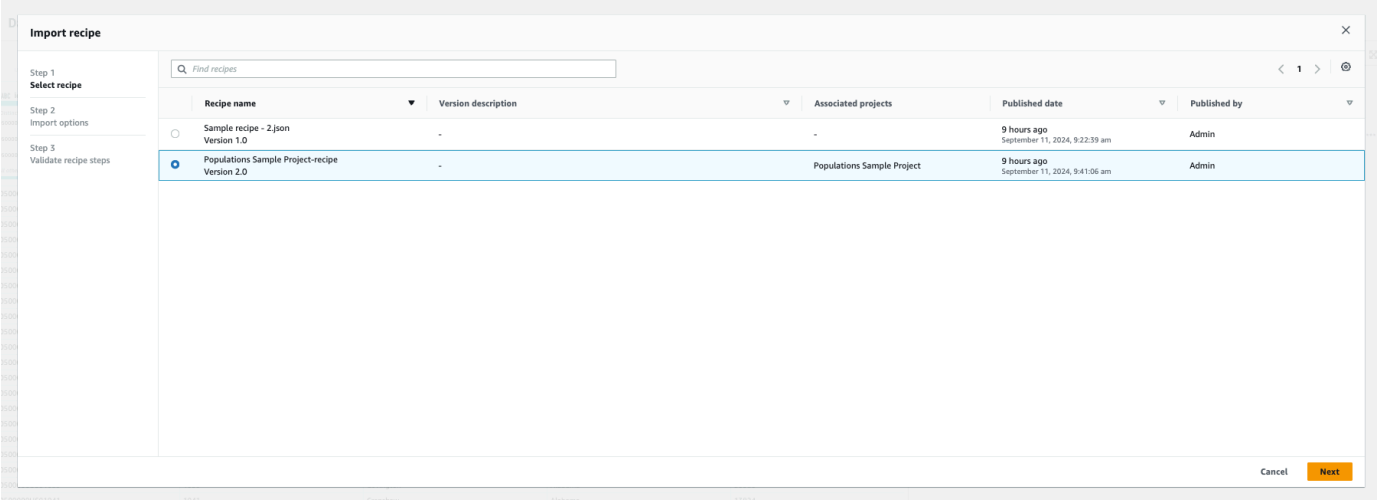


6. データフレームはロードされ、ソースデータに関する詳細情報が表示されます。

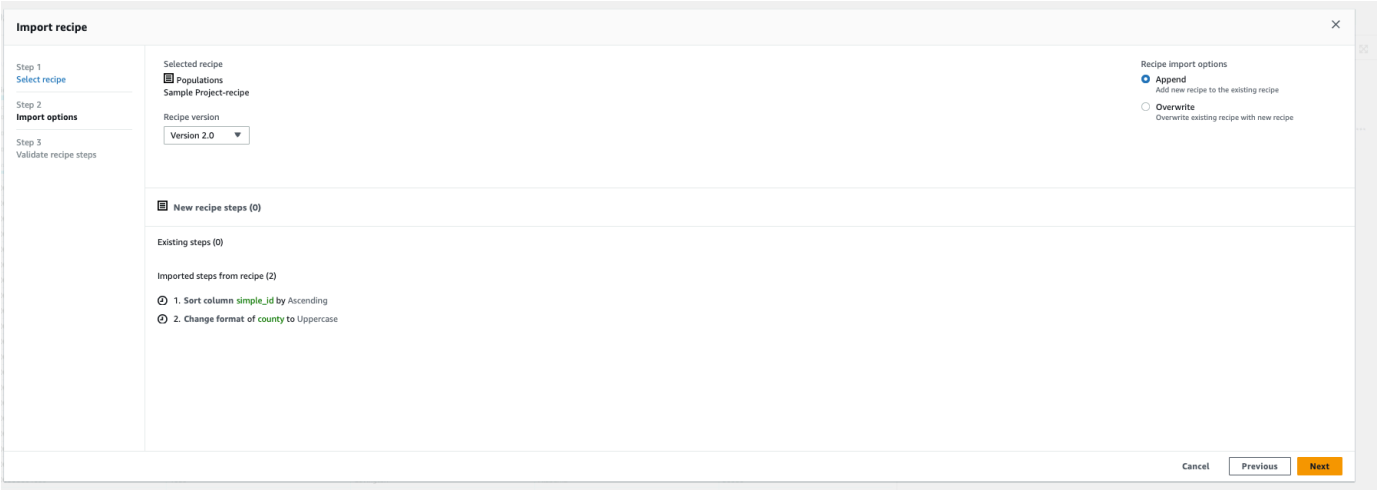
   [その他のアクション] アイコンを選択し、[レシピのインポート] を選択します。



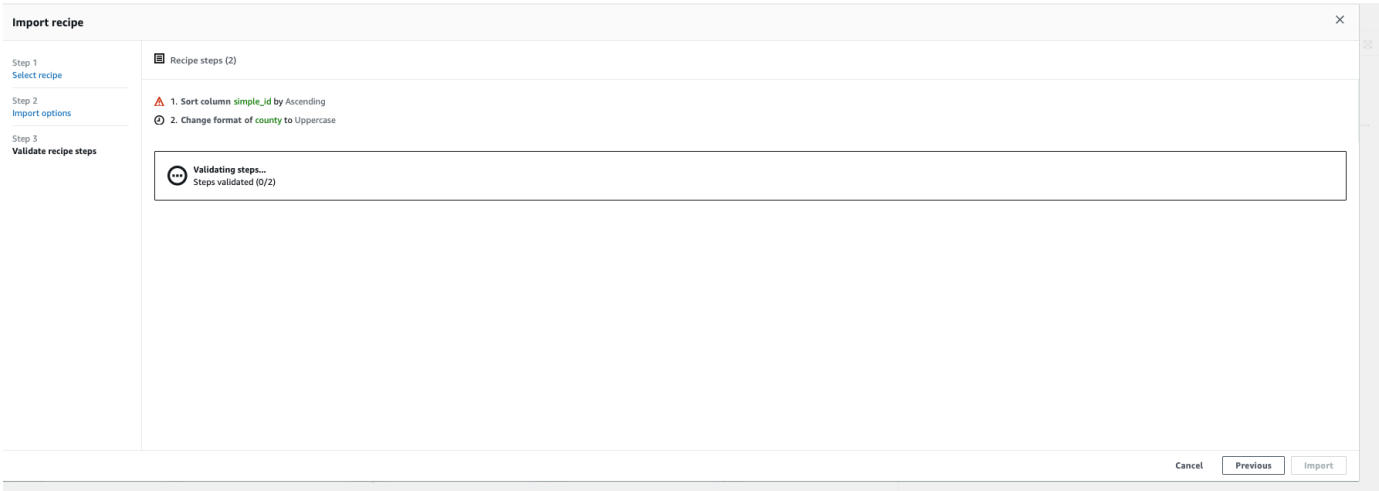
7. レシピのインポートウィザードを使用して、ステップを完了します。ステップ 1 で、レシピを検索して選択し、[次へ] を選択します。

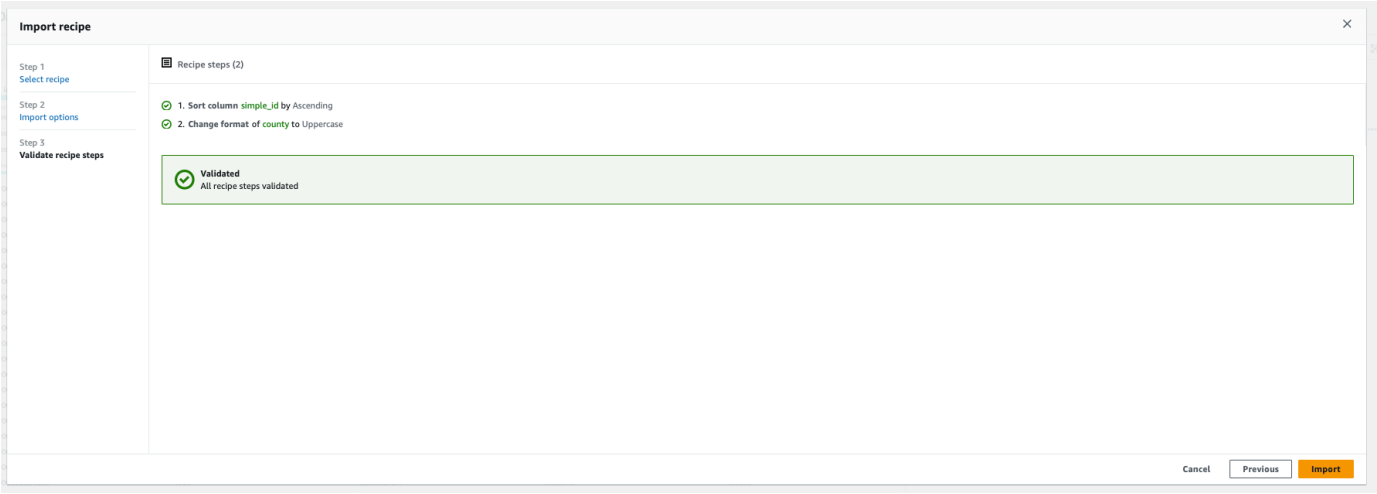


8. ステップ 2 で、インポートオプションを選択します。新しいレシピを既存のレシピに追加するか、既存のレシピを上書きするかを選択できます。[Next] を選択します。



9. ステップ 3 で、レシピステップを検証します。AWS Glue DataBrew レシピをインポートしたら、このレシピを AWS Glue Studio で直接編集できます。





10. その後、ステップは AWS Glue ジョブの一部としてインポートされます。必要に応じてジョブに名前を付けたり、割り当てられた容量を調整したりして、[ジョブの詳細] タブで必要な設定変更を行います。[保存] を選択して、ジョブとレシピを保存します。

    > **Note**
    >
    > JOIN、UNION、GROUP_BY、PIVOT、UNPIVOT、TRANSPOSE はレシピのインポートではサポートされず、レシピ作成モードでも使用できません。

11. 必要に応じて、その他の変換ノードを追加してジョブ作成を終了し、データターゲットノードを追加できます。

    レシピをインポートした後にステップの順序を変更すると、AWS Glue はそれらのステップの検証を実行します。例えば、列の名前を変更してから削除し、削除ステップを先頭に移動した場合、名前変更ステップは無効です。その後、ステップを編集して検証エラーを修正できます。

AWS Glue DataBrew から AWS Glue Studio への移行

AWS Glue DataBrew にレシピがある場合は、次のチェックリストを使用してレシピを AWS Glue Studio に移行します。

目的	こちらを実行してください
ユーザーに AWS Glue DataBrew レシピ、レシピバージョン、レシピの説明の取得を許可します。	ロールが必要なアクションにアクセスできるようにするポリシーに IAM アクセス許可を追加

目的	こちらを実行してください
	します。「AWS Glue DataBrew に対する IAM アクセス許可」を参照してください。
既存の AWS Glue DataBrew レシピを AWS Glue Studio にインポートします。	「AWS Glue DataBrew レシピのインポート」の手順を実行します。
JOIN と UNION を使用してレシピをインポートします。	UNION 変換と JOIN 変換を含むレシピはサポートされていません。Data Preparation Recipe ノードの前後に AWS Glue Studio で [結合] 変換と [ユニオン] 変換を使用します。

スキーマの変更によるデータプロパティキーの再マッピング

スキーマ変更変換を使用して、ソースデータのプロパティキーをターゲットデータに適切な設定に再マッピングします。スキーマ変更変換ノードでは、次のことができます。

- 複数のデータプロパティキーの名前の変更。
- 新しいデータ型がサポートされており、2 つのデータ型間の変換パスがある場合、データプロパティキーのデータ型の変更。
- 削除するデータプロパティキーの指定によるデータプロパティキーのサブセットの選択。

また、必要に応じて、ジョブ図にスキーマ変更ノードを追加できます。例えば、追加のデータソースを変更する場合や、Join 変換の後に続く場合などです。

10 進データ型で Change Schema を使用する

10 進データ型で Change Schema 変換を使用する場合、Change Schema 変換は精度をデフォルト値 (10,2) に変更します。これを変更してユースケースの精度を設定するには、SQL クエリ変換を使用して特定の精度で列をキャストできます。

例えば、「DecimalCol」という名前の入力列が 10 進数タイプで、それを特定の精度 (18,6) で「OutputDecimalCol」という名前の出力列に再マップする場合、次のようになります。

1. Change Schema 変換後に後続の SQL クエリ変換を追加します。
2. SQL クエリ変換では、SQL クエリを使用して、再マッピングされた列を希望の精度にキャストします。SQL クエリは次のようになります。

```
SELECT col1, col2, CAST(DecimalCol AS DECIMAL(18,6)) AS OutputDecimalCol
FROM __THIS__
```

上記の SQL クエリでは、次の操作を行います。

- 「col1」と「col2」は、変更せずに渡すデータ内の他の列です。
- `DecimalCol` は、入力データの元の列名です。
- 「CAST(DecimalCol AS DECIMAL(18,6))」は、DecimalCol」を 18 桁と 6 桁の小数点以下の精度を持つ 10 進数型にキャストします。
- `AS OutputDecimalCol ` は、キャストされた列の名前を `OutputDecimalCol ` に変更します。

SQL クエリ変換を使用すると、Change Schema 変換で設定されたデフォルトの精度を上書きし、10 進数の列を目的の精度に明示的にキャストできます。このアプローチにより、Change Schema 変換を活用してデータの名前変更と再構築を行い、その後の SQL クエリ変換を通じて 10 進数列の精度要件を処理できます。

Change Schema 変換をジョブに追加

> **Note**
>
> スキーマ変更変換は、大文字と小文字を区別しません。

スキーマ変更変換ノードをジョブ図に追加するには

1. (オプション) リソースパネルを開いて、[スキーマを変更する] を選択し、必要に応じてジョブ図に新しい変換を追加します。
2. ノードのプロパティパネルで、ジョブ図にノード名を入力します。ノードの親がまだ選択されていない場合、[Node parents] (ノードの親) リストから、変換の入力ソースとして使用するノードを選択します。
3. ノードのプロパティパネルで、[変換] タブを選択します。
4. その後、入力スキーマを変更します。

   - データプロパティキーの名前を変更するには、キーの新しい名前を[Target key] (ターゲットキー) フィールドに入力します。

- データプロパティキーのデータ型を変更するには、[Data type] (データ型) リストから、新しいデータ型を選択します。
- ターゲットスキーマからデータプロパティキーを削除するには、そのキーの [Drop] (削除) チェックボックスをオンにします。

5. (オプション) 変換ノードのプロパティを設定した後、ノードの詳細パネルの [Output schema] (出力スキーマ) タブを選択して、データ用に変更されたスキーマを表示できます。ジョブ内の任意のノードに対してこのタブを初めて選択すると、データにアクセスする IAM ロールを指定するよう求められます。[Job details] (ジョブの詳細) タブで IAM ロールをまだ指定していない場合、ここで IAM ロールを入力するよう求められます。
6. (オプション) ノードおよび変換のプロパティを設定した後、ノードの詳細パネルの [Data preview] (データのプレビュー) タブを選択して、変更されたデータセットをプレビューできます。ジョブ内の任意のノードに対してこのタブを初めて選択すると、データにアクセスする IAM ロールを指定するよう求められます。この機能を使用するには費用がかかり、IAM ロールを指定するとすぐに請求が開始します。

Drop Duplicates を使用する

Drop Duplicates 変換では、2 つのオプションによってデータソースから行を削除します。完全に一致した重複する行を削除するか、照合するフィールドを選択して、選択したフィールドに基づいて重複する行のみを削除するかのどちらかを選択できます。

例えば、次のデータセットには重複する行があり、行内の一部の値が一致していたり異なっていたりします。また、いくつかの行は、別の行とすべての値が完全に一致しています。

行	名前	Email(メール)	年齢	都道府県	注記
1	Joy	joy@gmail	33	NY	
2	Tim	tim@gmail	45	OH	
3	Rose	rose@gmail	23	NJ	
4	Tim	tim@gmail	42	OH	
5	Rose	rose@gmail	23	NJ	
6	Tim	tim@gmail	42	OH	これは重複した行であり、

行	名前	Email(メール)	年齢	都道府県	注記
					行 4 とすべての値が完全に一致しています
7	Rose	rose@gmail	23	NJ	これは重複した行であり、行 5 とすべての値が完全に一致しています

行全体で照合することを選択した場合、行 6 と 7 はデータセットから削除されます。データセットは次のようになります。

行	名前	Email(メール)	年齢	都道府県
1	Joy	joy@gmail	33	NY
2	Tim	tim@gmail	45	OH
3	Rose	rose@gmail	23	NJ
4	Tim	tim@gmail	42	OH
5	Rose	rose@gmail	23	NJ

キーを指定することを選択した場合、「名前」と「Email (メール)」が一致する行を削除するように選択できます。これにより、データセットの「重複する行」をより細かく制御できます。「名前」と「Email (メール)」を指定すると、データセットは次のようになります。

行	名前	Email(メール)	年齢	都道府県
1	Joy	joy@gmail	33	NY

行	名前	Email(メール)	年齢	都道府県
2	Tim	tim@gmail	45	OH
3	Rose	rose@gmail	23	NJ

留意すべき点

- 行の重複を認識するため、値では大文字と小文字が区別されます。行のすべての値は、大文字と小文字も一致する必要があります。これは、選択したオプション (行全体での一致/指定キーでの一致) のどちらにも当てはまります。
- すべての値は文字列として読み込まれます。
- Drop Duplicates 変換は、Spark の dropDuplicates コマンドを使用します。
- Drop Duplicates 変換を使用すると、最初の行は保持され、他の行は削除されます。
- Drop Duplicates 変換は、データフレームのスキーマを変更しません。キーの指定を選択した場合、生成されたデータフレームにはすべてのフィールドが保持されます。

SelectFields を使用して大多数のデータプロパティキーを削除する

SelectFields 変換を使用して、データセットからデータプロパティキーのサブセットを作成できます。保持するデータプロパティキーを指定します。残りはデータセットから削除されます。

> **Note**
>
> SelectFields 変換では、大文字と小文字が区別されます。大文字と小文字を区別しない方法でフィールドを選択する必要がある場合は、ApplyMapping を使用します。

SelectFields 変換ノードをジョブ図に追加するには

1. (オプション) リソースパネルを開いて、[SelectFields] を選択し、必要に応じてジョブ図に新しい変換を追加します。
2. [Node properties] (ノードのプロパティ) タブで、ジョブ図のノードの名前を入力します。ノードの親がまだ選択されていない場合は、[Node parents] (ノードの親) リストから、変換の入力ソースとして使用するノードを選択します。

3. ノードの詳細パネルで [Transform] (変換) タブを選択します。
4. 見出し [SelectFields] の下にある、保持するデータセット内のデータプロパティキーを選択します。選択されていないデータプロパティキーは、データセットから削除されます。

   列見出し [Field] (フィールド) の横にあるチェックボックスをオンにして、データセット内のすべてのデータプロパティキーを自動的に選択することもできます。その後、個々のデータプロパティキーの選択を解除して、それらをデータセットから削除できます。
5. (オプション) 変換ノードのプロパティを設定した後、ノードの詳細パネルの [Output schema] (出力スキーマ) タブを選択して、データ用に変更されたスキーマを表示できます。ジョブ内の任意のノードに対してこのタブを初めて選択すると、データにアクセスする IAM ロールを指定するよう求められます。[Job details] (ジョブの詳細) タブで IAM ロールをまだ指定していない場合、ここで IAM ロールを入力するよう求められます。
6. (オプション) ノードおよび変換のプロパティを設定した後、ノードの詳細パネルの [Data preview] (データのプレビュー) タブを選択して、変更されたデータセットをプレビューできます。ジョブ内の任意のノードに対してこのタブを初めて選択すると、データにアクセスする IAM ロールを指定するよう求められます。この機能を使用するには費用がかかり、IAM ロールを指定するとすぐに請求が開始します。

DropFields を使用して大多数のデータプロパティキーを保持する

DropFields 変換を使用して、データセットからデータプロパティキーのサブセットを作成できます。データセットから削除するデータプロパティキーを指定します。残りのキーは保持されます。

> **Note**
>
> DropFields 変換では、大文字と小文字が区別されます。大文字と小文字を区別しない方法でフィールドを選択する必要がある場合は、スキーマ変更を使用します。

ジョブ図に DropFields 変換ノードを追加するには

1. (オプション) リソースパネルを開いて、[DropFields] を選択し、必要に応じてジョブ図に新しい変換を追加します。
2. [Node properties] (ノードのプロパティ) タブで、ジョブ図のノードの名前を入力します。ノードの親がまだ選択されていない場合は、[Node parents] (ノードの親) リストから、変換の入力ソースとして使用するノードを選択します。
3. ノードの詳細パネルで [Transform] (変換) タブを選択します。

4. 見出し [DropFields] の下で、データソースから削除するデータプロパティキーを選択します。

   列見出し [Field] (フィールド) の横にあるチェックボックスをオンにして、データセット内のすべてのデータプロパティキーを自動的に選択することもできます。その後、個々のデータプロパティキーの選択を解除して、それらをデータセット内に保持できます。

5. (オプション) 変換ノードのプロパティを設定した後、ノードの詳細パネルの [Output schema] (出力スキーマ) タブを選択して、データ用に変更されたスキーマを表示できます。ジョブ内の任意のノードに対してこのタブを初めて選択すると、データにアクセスする IAM ロールを指定するよう求められます。[Job details] (ジョブの詳細) タブで IAM ロールをまだ指定していない場合、ここで IAM ロールを入力するよう求められます。

6. (オプション) ノードおよび変換のプロパティを設定した後、ノードの詳細パネルの [Data preview] (データのプレビュー) タブを選択して、変更されたデータセットをプレビューできます。ジョブ内の任意のノードに対してこのタブを初めて選択すると、データにアクセスする IAM ロールを指定するよう求められます。この機能を使用するには費用がかかり、IAM ロールを指定するとすぐに請求が開始します。

データセット内のフィールドの名前を変更する

RenameField 変換を使用して、データセット内の個々のプロパティキーの名前を変更します。

> **Note**
>
> RenameField 変換では、大文字と小文字が区別されます。大文字と小文字を区別しないで変換する必要がある場合は、ApplyMapping を使用します。

> **Tip**
>
> スキーマ変更変換では、1 回の変換でデータセット内に存在する複数のデータプロパティキーの名前を変更できます。

RenameField 変換ノードをジョブ図に追加するには

1. (オプション) リソースパネルを開いて、[RenameField] を選択し、必要に応じてジョブ図に新しい変換を追加します。

2. [Node properties] (ノードのプロパティ) タブで、ジョブ図のノードの名前を入力します。ノードの親がまだ選択されていない場合は、[Node parents] (ノードの親) リストから、変換の入力ソースとして使用するノードを選択します。
3. [Transform] (変換) タブを選択します。
4. 見出し [Data field] (データフィールド) の下で、ソースデータからプロパティキーを選択し、その後 [New field name] (新しいフィールド名) フィールドに新しい名前を入力します。
5. (オプション) 変換ノードのプロパティを設定した後、ノードの詳細パネルの [Output schema] (出力スキーマ) タブを選択して、データ用に変更されたスキーマを表示できます。ジョブ内の任意のノードに対してこのタブを初めて選択すると、データにアクセスする IAM ロールを指定するよう求められます。[Job details] (ジョブの詳細) タブで IAM ロールをまだ指定していない場合、ここで IAM ロールを入力するよう求められます。
6. (オプション) ノードおよび変換のプロパティを設定した後、ノードの詳細パネルの [Data preview] (データのプレビュー) タブを選択して、変更されたデータセットをプレビューできます。ジョブ内の任意のノードに対してこのタブを初めて選択すると、データにアクセスする IAM ロールを指定するよう求められます。この機能を使用するには費用がかかり、IAM ロールを指定するとすぐに請求が開始します。

Spigot を使用してデータセットをサンプリングする

ジョブで実行される変換をテストするには、データのサンプルを取得して、変換が意図したとおりに機能することを確認します。Spigot 変換では、データセットから Amazon S3 バケットの JSON ファイルにレコードのサブセットが書き出されます。データのサンプリングには、ファイルの最初からの特定のレコード数、またはレコードの選択に使用される確率係数を使用します。

Spigot 変換ノードをジョブ図に追加するには

1. (オプション) リソースパネルを開いて、[スピゴット] を選択し、必要に応じてジョブ図に新しい変換を追加します。
2. [Node properties] (ノードのプロパティ) タブで、ジョブ図のノードの名前を入力します。ノードの親がまだ選択されていない場合は、[Node parents] (ノードの親) リストから、変換の入力ソースとして使用するノードを選択します。
3. ノードの詳細パネルで [Transform] (変換) タブを選択します。
4. Amazon S3 パスを入力するか、[Browse S3] (S3 をブラウズ) を選択して、Amazon S3 内の場所を選択します。これは、ジョブによりデータサンプルを含む JSON ファイルが書き込まれる場所です。

5. サンプリング方法に関する情報を入力します。データセットの最初から書き込むレコード数の値、および任意のレコードを選択する確率のしきい値 (最大値が 1 の 10 進値として入力) を指定できます。

   例えば、データセットから最初の 50 レコードを書き込むには、レコード数を 50、確率のしきい値を 1 (100%) に設定します。

データセットの結合

結合 変換を使用すると、2 つのデータセットを 1 つに結合できます。比較する各データセットのスキーマのキー名を指定します。出力 DynamicFrame には、キーが結合条件を満たす行が含まれます。結合条件を満たす各データセットの行は、いずれかのデータセット内で見つかったすべての列を含む出力 DynamicFrame の単一の行に結合されます。

Join (結合) 変換ノードをジョブ図に追加するには

1. 使用可能なデータソースが 1 つしかない場合は、新しいデータソースノードをジョブ図に追加する必要があります。
2. 結合するソースノードを 1 つ選択します。リソースパネルを開いて、[Join] を選択し、ジョブ図に新しい変換を追加します。
3. [Node properties] (ノードのプロパティ) タブで、ジョブ図のノードの名前を入力します。
4. [Node properties] (ノードのプロパティ) タブの、見出し [Node parents] (ノードの親) の下で、結合の入力を提供するデータセットが 2 つ存在するように親ノードを追加します。親ノードは、データソースノードまたは変換ノードです。

   > **Note**
   >
   > 結合すると、親ノードを 2 つだけ持つことができます。

5. [Transform] (変換) タブを選択します。

   競合するキー名があることを示すメッセージが表示された場合は、次のいずれかの操作を行います。

   - [Resolve it] (解決する) を選択して、ApplyMapping 変換ノードをジョブ図に自動的に追加します。ApplyMapping ノードにより、他のデータセットのキーと同じ名前を持つデータセット内のキーにプレフィックスが追加されます。例えば、デフォルト値 **`right`** を使用している

場合、左側のデータセットと同じキー名を持つ右側のデータセットのキー名は `(right)`*key name* に変更されます。

- ジョブ図に変換ノードを早期に手動で追加して、競合するキーを削除するか、名前を変更できます。

6. [Join type] (結合タイプ) のリストで、結合のタイプを選択します。

   - Inner join (内部結合): 結合条件に基づくすべての一致に対して、両方のデータセットの列を含む行を返します。結合条件を満たさない行は返されません。
   - Left join (左結合): 左側のデータセットのすべての行と、結合条件を満たす右側のデータセットの行の結合です。
   - Right join (右結合): 右側のデータセットのすべての行と、結合条件を満たす左側のデータセットの行の結合です。
   - Outer join (外部結合): 両方のデータセットのすべての行の結合です。
   - Left semi join (左半結合): 右側のデータセットで結合条件に基づく一致がある、左側のデータセットのすべての行の結合です。
   - Left anti join (左反結合): 右側のデータセットで結合条件に基づく一致がない、左側のデータセットのすべての行の結合です。

7. [Transform] (変換) タブで、見出し [Join conditions] (結合条件) の下にある、[Add condition] (条件の追加)を選択します。比較する各データセットからプロパティキーを選択します。比較演算子の左側にあるプロパティキーを左側のデータセット、右側にあるプロパティキーを右側のデータセットと呼びます。

   より複雑な結合条件の場合、[Add condition] (条件の追加) を複数回クリックして、
   他の一致するキーを追加できます。誤って条件を追加した場合は、削除アイコン
   (🗑)
   をクリックして削除できます。

8. (オプション) 変換ノードのプロパティを設定した後、ノードの詳細パネルの [Output schema] (出力スキーマ) タブを選択して、データ用に変更されたスキーマを表示できます。ジョブ内の任意のノードに対してこのタブを初めて選択すると、データにアクセスする IAM ロールを指定するよう求められます。[Job details] (ジョブの詳細) タブで IAM ロールをまだ指定していない場合、ここで IAM ロールを入力するよう求められます。

9. (オプション) ノードおよび変換のプロパティを設定した後、ノードの詳細パネルの [Data preview] (データのプレビュー) タブを選択して、変更されたデータセットをプレビューできます。ジョブ内の任意のノードに対してこのタブを初めて選択すると、データにアクセスする

IAM ロールを指定するよう求められます。この機能を使用するには費用がかかり、IAM ロールを指定するとすぐに請求が開始します。

結合出力スキーマの例では、次のプロパティキーを持つ 2 つのデータセット間の結合を検討します。

```
Left: {id, dept, hire_date, salary, employment_status}
Right: {id, first_name, last_name, hire_date, title}
```

結合は、= 比較演算子を使用して、id および hire_date キーで一致するように設定されます。

両方のデータセットに id および hire_date キーが含まれているため、[Resolve it] (解決する) を選択して、右側のデータセットのキーにプレフィックス **right** が自動で追加されるようにします。

出力スキーマのキーは次のようになります。

```
{id, dept, hire_date, salary, employment_status,
(right)id, first_name, last_name, (right)hire_date, title}
```

Union を使用して行を結合する

Union 変換ノードは、同じスキーマを持つ複数のデータソースの行を結合する場合に使用します。

Union 変換には次の 2 種類があります。

1. ALL – ALL を適用すると、Union の結果として、重複する行は削除されません。
2. DISTINCT – DISTINCT を適用すると、Union の結果として、重複する行は削除されます。

Union と Join

行を結合するには Union を使用します。列を結合するには Join を使用します。

Visual ETL キャンバスでの Union 変換の使用

1. Union 変換を実行するには、複数のデータソースを追加します。データソースを追加するには、リソースパネルを開き、[ソース] タブからデータソースを選択します。Union 変換を使用する前に、Union の対象となるすべてのデータソースのスキーマと構造が同じであることを確認する必要があります。

2. Union 変換を使用して結合したいデータソースが 2 つ以上ある場合は、キャンバスに追加して Union 変換を作成します。キャンバス上のリソースパネルを開き、「Union」を検索します。リソースパネルの [変換] タブを選択し、Union 変換が見つかるまで下にスクロールして、[Union] を選択することもできます。
3. ジョブキャンバスの Union ノードを選択します。ノードプロパティウィンドウで、Union 変換に接続する親ノードを選択します。
4. AWS Glue は互換性をチェックして、Union 変換がすべてのデータソースに適用できることを確認します。データソースのスキーマが同じ場合、実行できます。データソースのスキーマが同じでない場合は、無効であるというエラーメッセージが表示されます。「この Union の入力スキーマは一致していません。ApplyMapping を使用してスキーマを一致させることを検討してください」 これを修正するには、[ApplyMapping を使用] を選択します。
5. Union の種類を選択します。
    1. All – デフォルトでは、Union の種類として All が選択されています。これにより、データを結合したときに、重複する行は重複したままになります。
    2. Distinct – データを結合したときに、重複する行を削除したい場合は、Distinct を選択します。

SplitFields を使用してデータセットを 2 つに分割する

SplitFields 変換を使用すると、入力データセット内のデータプロパティキーをいくつか選択し、それらを 1 つのデータセットに配置したり、選択されていないキーを別のデータセットに配置できます。この変換からの出力は、`DynamicFrames` のコレクションです。

> **Note**
> 出力をターゲットの場所に送信するには、SelectFromCollection 変換を使用して、`DynamicFrames` のコレクションを単一の `DynamicFrame` に変換する必要があります。

SplitFields 変換では、大文字と小文字が区別されます。大文字と小文字を区別しないプロパティキー名が必要な場合は、ApplyMapping 変換を親ノードとして追加します。

SplitFields 変換ノードをジョブ図に追加するには

1. (オプション) リソースパネルを開いて、[SplitFields] を選択し、必要に応じてジョブ図に新しい変換を追加します。
2. [Node properties] (ノードのプロパティ) タブで、ジョブ図のノードの名前を入力します。ノードの親がまだ選択されていない場合は、[Node parents] (ノードの親) リストから、変換の入力ソースとして使用するノードを選択します。
3. [Transform] (変換) タブを選択します。
4. 最初のデータセットに入れるプロパティキーを選択します。選択しなかったキーは、2 番目のデータセットに配置されます。
5. (オプション) 変換ノードのプロパティを設定した後、ノードの詳細パネルの [Output schema] (出力スキーマ) タブを選択して、データ用に変更されたスキーマを表示できます。ジョブ内の任意のノードに対してこのタブを初めて選択すると、データにアクセスする IAM ロールを指定するよう求められます。[Job details] (ジョブの詳細) タブで IAM ロールをまだ指定していない場合、ここで IAM ロールを入力するよう求められます。
6. (オプション) ノードおよび変換のプロパティを設定した後、ノードの詳細パネルの [Data preview] (データのプレビュー) タブを選択して、変更されたデータセットをプレビューできます。ジョブ内の任意のノードに対してこのタブを初めて選択すると、データにアクセスする IAM ロールを指定するよう求められます。この機能を使用するには費用がかかり、IAM ロールを指定するとすぐに請求が開始します。
7. SelectFromCollection 変換ノードを設定して、結果のデータセットを処理します。

SelectFromCollection 変換の概要

SplitFields のように、単一ではなく、複数のデータセットが出力される変換もあります。SelectFromCollection 変換では、データセットのコレクション (`DynamicFrames` の配列) から、1 つのデータセット (`DynamicFrame`) が選択されます。変換の出力は、選択された `DynamicFrame` です。

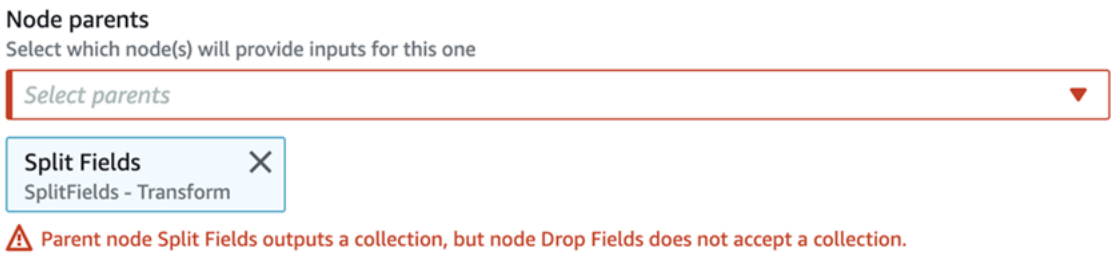
この変換は、次のような `DynamicFrames` のコレクションを作成する変換を使用した後に使用する必要があります。

- カスタムコード変換
- SplitFields

変換後に SelectFromCollection 変換ノードをジョブ図に追加しないと、ジョブでエラーが発生します。

この変換の親ノードは、DynamicFrames のコレクションを返すノードである必要があります。この変換ノードに、Join (結合) 変換などの単一の DynamicFrame を返す親ノードを選択すると、ジョブによりエラーが返されます。

同様に、ジョブ図の SelectFromCollection ノードを、入力として単一の DynamicFrame を想定している変換の親ノードとして使用する場合、ジョブによりエラーが返されます。



SelectFromCollection を使用して保持するデータセットを選択する

SelectFromCollection 変換を使用して、DynamicFrames のコレクションを単一の DynamicFrame に変換できます。

SelectFromCollection 変換ノードをジョブ図に追加するには

1. (オプション) リソースパネルを開いて、[SelectFromCollection] を選択し、必要に応じてジョブ図に新しい変換を追加します。
2. [Node properties] (ノードのプロパティ) タブで、ジョブ図のノードの名前を入力します。ノードの親がまだ選択されていない場合は、[Node parents] (ノードの親) リストから、変換の入力ソースとして使用するノードを選択します。
3. [Transform] (変換) タブを選択します。
4. 見出し [Frame index] (フレームインデックス)の下で、DynamicFrames のコレクションから選択する DynamicFrame に対応する配列のインデックス番号を選択します。

   例えば、この変換の親ノードが SplitFields 変換である場合、そのノードの [Output schema] (出力スキーマ) タブで各 DynamicFrame のスキーマを表示できます。[出力 2] のスキーマに関連付けられている DynamicFrame を保持する場合、フレームインデックスの値は **1** を選択します。これは、リストの 2 番目の値です。

   選択した DynamicFrame だけが出力に含まれます。

5. (オプション) 変換ノードのプロパティを設定した後、ノードの詳細パネルの [Output schema] (出力スキーマ) タブを選択して、データ用に変更されたスキーマを表示できます。ジョブ内の任意のノードに対してこのタブを初めて選択すると、データにアクセスする IAM ロールを指定するよう求められます。[Job details] (ジョブの詳細) タブで IAM ロールをまだ指定していない場合、ここで IAM ロールを入力するよう求められます。
6. (オプション) ノードおよび変換のプロパティを設定した後、ノードの詳細パネルの [Data preview] (データのプレビュー) タブを選択して、変更されたデータセットをプレビューできます。ジョブ内の任意のノードに対してこのタブを初めて選択すると、データにアクセスする IAM ロールを指定するよう求められます。この機能を使用するには費用がかかり、IAM ロールを指定するとすぐに請求が開始します。

データセット内の欠落値を検索して埋める

FillMissingValues 変換を使用して、データセット内に欠落値があるレコードを検索し、補完により決定する値を持つ新しいフィールドを追加します。入力データセットは、欠落値を決定する機械学習 (ML) モデルのトレーニングに使用されます。増分のデータセットを使用する場合、増分の各セットが ML モデルのトレーニングデータとして使用されるため、結果はそれほど正確ではない可能性があります。

ジョブ図で FillMissingValues 変換ノードを使用するには

1. (オプション) リソースパネルを開いて、[FillMissingValues] を選択し、必要に応じてジョブ図に新しい変換を追加します。
2. [Node properties] (ノードのプロパティ) タブで、ジョブ図のノードの名前を入力します。ノードの親がまだ選択されていない場合、[Node parents] (ノードの親) リストから、変換の入力ソースとして使用するノードを選択します。
3. [Transform] (変換) タブを選択します。
4. [Data field] (データフィールド) で、欠落値を分析するソースデータから、列名またはフィールド名を選択します。
5. (オプション) [New field name] (新しいフィールド名) フィールドに、フィールドの名前を入力します。このフィールドは、分析したフィールドの推定置換値を保持する各レコードに追加されます。分析したフィールドに欠落値がない場合、その分析したフィールドの値が新しいフィールドにコピーされます。

   フィールドの名前を指定しない場合、デフォルトの名前は、分析した列に `_filled` を追加した名前になります。例えば、[Data field] (データフィールド) に「**`Age`**」と入力し、[New field

name] (新しいフィールド名) に値を指定しない場合、**Age_filled** という名前の新しいフィードが各レコードに追加されます。

6. (オプション) 変換ノードのプロパティを設定した後、ノードの詳細パネルの [Output schema] (出力スキーマ) タブを選択して、データ用に変更されたスキーマを表示できます。ジョブ内の任意のノードに対してこのタブを初めて選択すると、データにアクセスする IAM ロールを指定するよう求められます。[Job details] (ジョブの詳細) タブで IAM ロールをまだ指定していない場合、ここで IAM ロールを入力するよう求められます。
7. (オプション) ノードおよび変換のプロパティを設定した後、ノードの詳細パネルの [Data preview] (データのプレビュー) タブを選択して、変更されたデータセットをプレビューできます。ジョブ内の任意のノードに対してこのタブを初めて選択すると、データにアクセスする IAM ロールを指定するよう求められます。この機能を使用するには費用がかかり、IAM ロールを指定するとすぐに請求が開始します。

データセット内のキーのフィルタリング

フィルター変換を使用して、正規表現に基づいて入力データセットのレコードをフィルタリングすることで、新しいデータセットを作成できます。フィルター条件を満たさない行は、出力から削除されます。

- 文字列のデータ型の場合、キー値と指定された文字列が一致する行をフィルタリングできます。
- 数値のデータ型の場合、比較演算子 <、>、=、!=、<=、>= を使用してキー値と指定された値を比較することで、行をフィルタリングできます。

複数のフィルター条件を指定した場合、デフォルトでは AND 演算子を使用して結果が結合されますが、代わりに OR を使用することもできます。

フィルター変換では、大文字と小文字が区別されます。大文字と小文字を区別しないプロパティキー名が必要な場合は、ApplyMapping 変換を親ノードとして追加します。

フィルター変換ノードをジョブ図に追加するには

1. (オプション) リソースパネルを開いて、[フィルター] を選択し、必要に応じてジョブ図に新しい変換を追加します。
2. [Node properties] (ノードのプロパティ) タブで、ジョブ図のノードの名前を入力します。ノードの親がまだ選択されていない場合は、[Node parents] (ノードの親) リストから、変換の入力ソースとして使用するノードを選択します。
3. [Transform] (変換) タブを選択します。

4. [Global AND] または [Global OR] を選択します。これにより、複数のフィルター条件の結合方法が決定します。すべての条件は、AND または OR オペレーションで結合されます。フィルター条件が 1 つしかない場合は、どちらかを選択できます。
5. [Filter condition] (フィルター条件) セクションで [Add condition] (条件の追加) ボタンをクリックして、フィルター条件を追加します。

   [Key] (キー) フィールドで、データセットからプロパティキー名を選択します。[Operation] (オペレーション) フィールドで、比較演算子を選択します。[Value] (値) フィールドに、比較値を入力します。フィルター条件のいくつかの例を次に示します。

   - `year >= 2018`
   - `State matches 'CA*'`

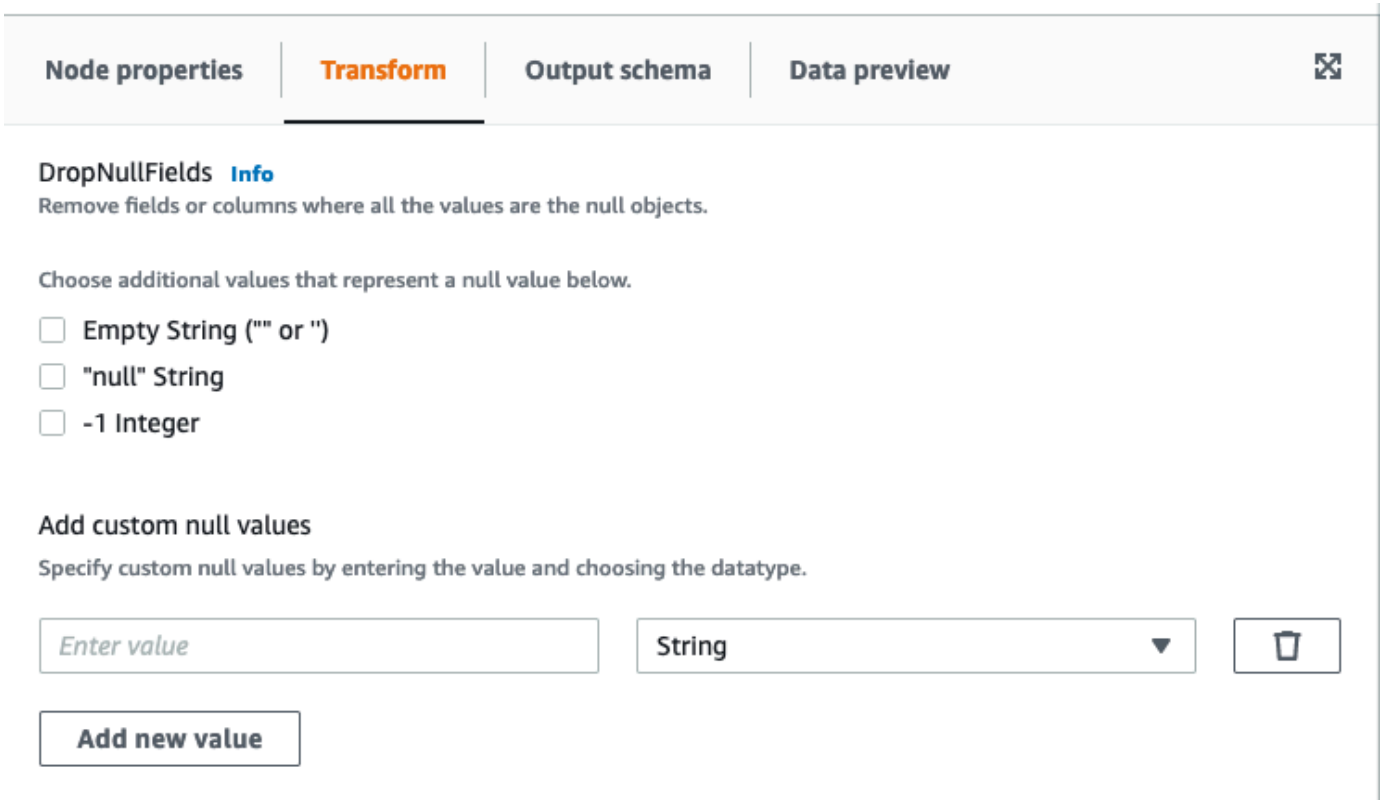
   文字列の値をフィルタリングする場合は、比較値に、ジョブのプロパティ (Python または Scala) で選択したスクリプト言語と一致する正規表現の形式が使用されていることを確認します。
6. 必要に応じて、他のフィルター条件を追加します。
7. (オプション) 変換ノードのプロパティを設定した後、ノードの詳細パネルの [Output schema] (出力スキーマ) タブを選択して、データ用に変更されたスキーマを表示できます。ジョブ内の任意のノードに対してこのタブを初めて選択すると、データにアクセスする IAM ロールを指定するよう求められます。[Job details] (ジョブの詳細) タブで IAM ロールをまだ指定していない場合、ここで IAM ロールを入力するよう求められます。
8. (オプション) ノードおよび変換のプロパティを設定した後、ノードの詳細パネルの [Data preview] (データのプレビュー) タブを選択して、変更されたデータセットをプレビューできます。ジョブ内の任意のノードに対してこのタブを初めて選択すると、データにアクセスする IAM ロールを指定するよう求められます。この機能を使用するには費用がかかり、IAM ロールを指定するとすぐに請求が開始します。

NULL 値を持つフィールドを削除するDropNullFields の使用

フィールド内のすべての値が 'null' の場合、[DropNullFields]transform を使用し、データセットからフィールドを削除します。デフォルトでは、AWS Glue Studio は null オブジェクトを認識しますが、空の文字列、“null”の列、-1 の整数、または 0 などのプレースホルダなどでは、自動的に「null」として認識されません。

DropNullFields を使用するには

1. ジョブダイアグラムに DropNullFields ノードを追加します。
2. [Node properties]タブで、NULL 値を表す追加の値を選択します。値を選択しないか、すべての値を選択するかを選択できます。



   - 空の文字列("" or '') - 空の文字列を含むフィールドは削除されます。
   - "null string"-単語「null」の文字列を含むフィールドは削除されます。
   - a-1 (負の 1) integerを含む-1 整数 - フィールドは削除されます。
3. 必要に応じて、カスタム NULL 値を指定することもできます。これらは、データセットに一意である可能性がある NULL 値です。カスタム NULL 値を追加するには、[Add new value]を選択します。
4. カスタム Null 値を入力します。たとえば、ゼロを指定したり、データセット内の NULL を表すために使用されている任意の値を指定できます。
5. ドロップダウンフィールドでデータタイプを選択します。データ型は、文字列または整数のいずれかです。

> **Note**
>
> カスタムの NULL 値とそのデータ型は、フィールドが NULL 値として認識され、フィールドを削除するために、正確に一致する必要があります。カスタムの NULL 値のみが一致しているがデータ型が一致しない部分的な一致については、フィールドが削除されません。

SQL クエリを使用してデータを変換する

[SQL] 変換を使用して、SQL クエリの形式で独自の変換を記述できます。

SQL 変換ノードでは、入力として複数のデータセットを持つことができますが、出力として生成されるデータセットは 1 つだけです。これには、Apache SparkSQL クエリを入力するテキストフィールドが含まれています。入力として使用する各データセットにエイリアスを割り当てることで、SQL クエリを簡単に実行できます。SQL 構文の詳細については、Spark SQL ドキュメントを参照してください。

> **Note**
>
> VPC にあるデータソースで Spark SQL 変換を使用する場合は、AWS Glue VPC エンドポイントを、データソースを含む VPC に追加します。開発エンドポイントの設定の詳細については、AWS Glue デベロッパーガイドの「Adding a Development Endpoint」、「Setting Up Your Environment for Development Endpoints」、「Accessing Your Development Endpoint」を参照してください。

ジョブ図で SQL 変換ノードを使用するには

1. (オプション) 必要な場合、ジョブ図に変換ノードを追加します。ノードタイプに [SQL クエリ] を選択します。

> **Note**
>
> データプレビューセッションおよびカスタム SQL またはカスタムコードノードを使用する場合、データプレビューセッションはデータセット全体に SQL またはコードブロックをそのまま実行します。

2. [Node properties] (ノードのプロパティ) タブで、ジョブ図のノードの名前を入力します。ノードの親がまだ選択されていない場合、または SQL 変換に複数の入力が必要な場合は、[Node parents] (ノードの親) リストから、変換の入力ソースとして使用するノードを選択します。必要な場合、他の親ノードを追加します。
3. ノードの詳細パネルで [Transform] (変換) タブを選択します。
4. SQL クエリのソースデータセットは、各ノードの [Name] (名前) フィールドで指定する名前により識別されます。これらの名前を使用しない場合、または名前が SQL クエリに適していない場合は、各データセットに名前を関連付けることができます。コンソールでは、`MyDataSource` などのデフォルトのエイリアスが提供されています。

   例えば、SQL 変換ノードの親ノードの名前が `Rename Org PK field` の場合、このデータセットに `org_table` という名前を関連付けることができます。その後、このエイリアスをノード名の代わりに SQL クエリで使用できます。
5. 見出し [Code block] (コードブロック) の下にある テキスト入力フィールドで、SQL クエリを貼り付けるか、入力します。テキストフィールドには、SQL 構文のハイライトとキーワードの候補が表示されます。
6. SQL 変換ノードを選択した状態で、[Output schema] (出力スキーマ) タブを選択してから、[Edit] (編集) を選択します。SQL クエリの出力フィールドを示す、列とデータ型を指定します。

   そのページの [Output schema] (出力スキーマ) セクションにある次のアクションを使用して、スキーマを指定します。

   - 列の名前を変更するには、その列の [Key] (キー) テキストボックス ([field] (フィールド) または [property key] (プロパティキー) とも示されます) にカーソルを置き、新しい名前を入力します。
   - 列のデータ型を変更するには、ドロップダウンリストから列の新しいデータ型を選択します。
   - トップレベルの新しい列をスキーマに追加するには、[Overflow] (オーバーフロー) (⋯) ボタンを選択して、[Add root key] (ルートキーの追加) をクリックします。新しい列がスキーマの先頭に追加されます。
   - スキーマから列を削除するには、キー名の右端にある削除アイコン (🗑) をクリックします。
7. 出力スキーマの指定が完了したら、[Apply] (適用) を選択して変更を保存し、スキーマエディタを終了します。変更を保存しない場合は、[Cancel] (キャンセル) を選択して、スキーマエディタを編集します。

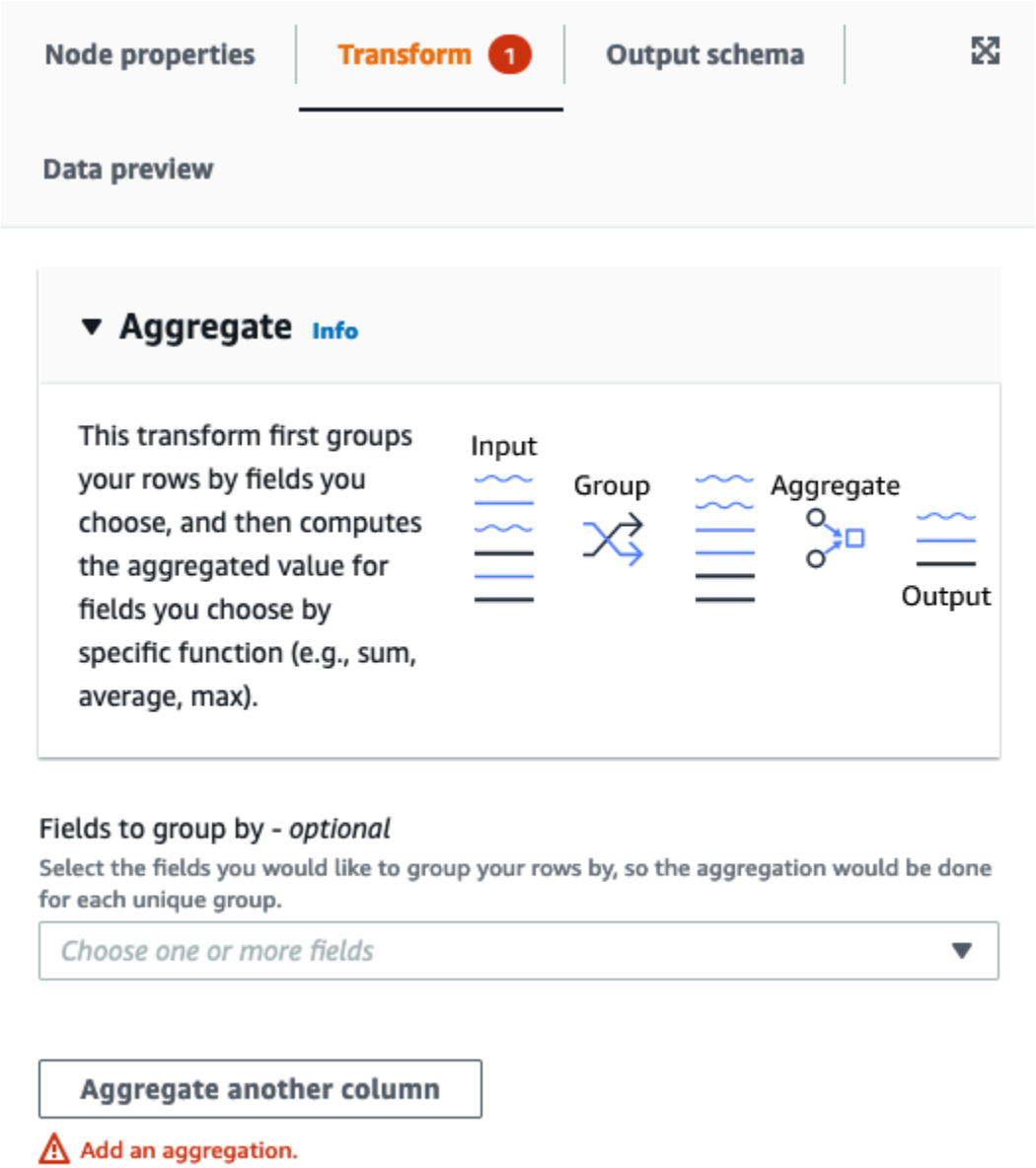
8. (オプション) ノードおよび変換のプロパティを設定した後、ノードの詳細パネルの [Data preview] (データのプレビュー) タブを選択して、変更されたデータセットをプレビューできます。ジョブ内の任意のノードに対してこのタブを初めて選択すると、データにアクセスする IAM ロールを指定するよう求められます。この機能を使用するには費用がかかり、IAM ロールを指定するとすぐに請求が開始します。

[集計] を使用して選択したフィールドに対して集約計算を実行する

Aggregate transform を使用するには

1. Aggregate ノードをジョブダイアグラムに追加します。
2. Node propertiesタブ上で、ドロップダウンフィールド (オプション) を選択して、一緒にグループ化するフィールドを選択します。一度に複数のフィールドを選択するか、検索バーに入力してフィールド名を検索できます。

   フィールドを選択すると、名前とデータ型が表示されます。フィールドを削除するには、フィールドで [X] を選択します。



3. [Aggregate another column]を選択します。少なくとも 1 つのフィールドを選択する必要があります。



4. [Field to aggregate]ドロップダウン中のフィールドを選択します。

5. 選択したフィールドに適用する集計関数を選択します：

   - avg-平均を計算します。
   - CountDistinct-一意の非ヌル値の数を計算します。
   - count-非 null 値の数を計算します。
   - first-「グループ別」の基準を満たす最初の値を返します。
   - last-「グループ別」の基準を満たす最後の値を返します。
   - kurtosis-周波数ディストリビューション曲線のピークのシャープネスを計算します。
   - max-「グループ別」の基準を満たす最高値を返します。
   - min-「グループ別」の基準を満たす最小値を返します。
   - skewness-正規ディストリビューションの確率ディストリビューションの非対称性の尺度
   - stddev_pop-母標準偏差を計算し、母分散の平方根を返します。
   - sum-グループ内のすべての値の合計
   - sumDistinct-グループ内の個別の値の合計
   - var_samp-グループのサンプル分散 (ヌルは無視されます)
   - var_pop-グループの母分散 (ヌルは無視されます)

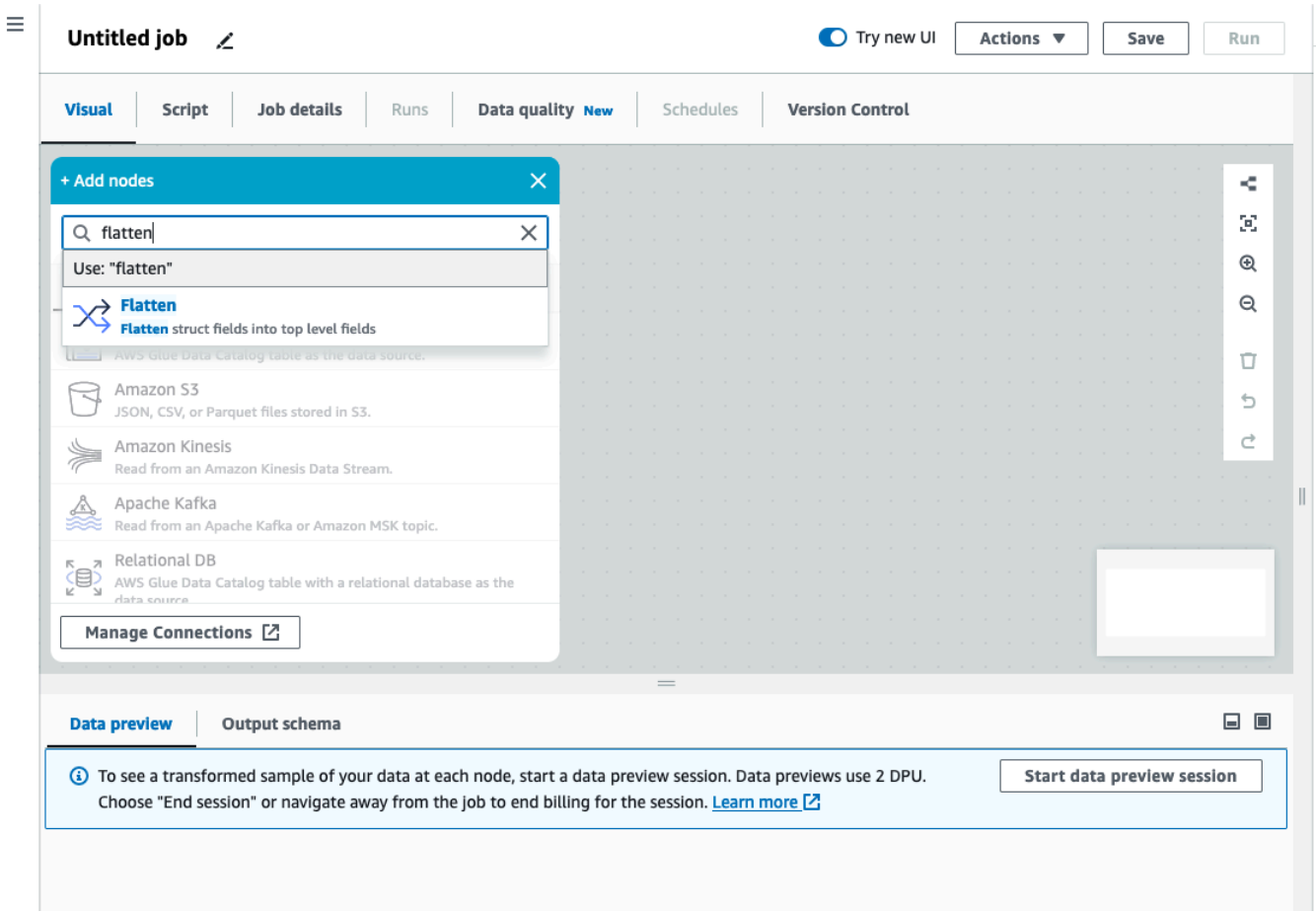
ネストされた Struct のフラット化

データ内のネストされた Struct のフィールドをフラット化し、最上位のフィールドにします。新しいフィールドの名前には、そこに届く Struct フィールドの名前が接頭辞として付いたフィールド名が、ドットで区切られて使用されます。

例えば、データに「phone_numbers」という Struct 型のフィールドがあり、他のフィールドでの「home_phone」という名前の Struct 型の 1 つとともに、「country_code」および「number」という 2 つのフィールドを伴っているとします。フラット化されると、この 2 つのフィールドはそれぞれ「phone_numbers.home_phone.country_code」および「phone_numbers.home_phone.number」という名前の最上位のフィールドになります。

フラット化された変換ノードをジョブ図に追加するには

1. リソースパネルを開いて、[変換] タブの [フラット化] 選択し、ジョブ図に新しい変換を追加します。検索バーを使用して、「Flatten」と入力し、フラット化ノードをクリックすることもできます。ノードを追加する際に選択したノードが、その親になります。



2. (オプション) [Node properties] (ノードのプロパティ) タブで、ジョブ図にノードの名前を入力できます。ノードの親がまだ選択されていない場合は、[Node parents] (ノードの親) リストから、変換の入力ソースとして使用するノードを選択します。
3. (オプション) [Transform] (変換) タブでは、フラット化するネストレベルの上限を設定できます。例えばその値を 1 に設定すると、最上位の Struct のみをフラット化できます。上限を 2 に設定すると、最上位とその次の Struct をフラット化できます。

UUID カラムを追加する

UUID (ユニバーサル一意識別子) 列を追加すると、各行に 36 字の一意の文字列が割り当てられます。

UUID 変換ノードをジョブ図に追加するには

1. リソースパネルを開いて、[UUID] を選択し、ジョブ図に新しい変換を追加します。ノードを追加する際に選択したノードが、その親になります。

2. (オプション) [Node properties] (ノードのプロパティ) タブで、ジョブ図にノードの名前を入力できます。ノードの親がまだ選択されていない場合は、[Node parents] (ノードの親) リストから、変換の入力ソースとして使用するノードを選択します。
3. (オプション) 新しい列の名前は [Transform] (変換) タブでカスタマイズできます。デフォルトでは「uuid」という名前が付けられます。

識別子列を追加する

データセット内の各行に数値識別子を割り当てます。

ジョブ図に識別子変換ノードを追加するには

1. リソースパネルを開いて、[Identifier] を選択し、ジョブ図に新しい変換を追加します。ノードを追加する際に選択したノードが、その親になります。
2. (オプション) [Node properties] (ノードのプロパティ) タブで、ジョブ図にノードの名前を入力できます。ノードの親がまだ選択されていない場合は、[Node parents] (ノードの親) リストから、変換の入力ソースとして使用するノードを選択します。
3. (オプション) 新しい列の名前は [Transform] (変換) タブでカスタマイズできます。デフォルトでは、「id」という名前が付けられます。
4. (オプション) ジョブがデータを段階的に処理して保存する場合は、ジョブの実行時に同じ ID が再利用されないよう注意する必要があります。

   [Transform] (変換) タブで、チェックボックスのオプション [unique] (一意) をチェックします。これで識別子にジョブのタイムスタンプが追加されるため、複数の実行間で一意になります。数字を増やせるようにするには、long の代わりに列を 10 進数にします。

列をタイムスタンプタイプに変換する

To timestamp 変換を使用すると、数値または文字列の列のデータ型をタイムスタンプに変更できます。これで、そのデータ型で保存したり、タイムスタンプを必要とする他の変換に適用したりできるようになります。

To timestamp 変換ノードをジョブ図に追加するには

1. リソースパネルを開いて、[To timestamp] を選択し、ジョブ図に新しい変換を追加します。ノードを追加する際に選択したノードが、その親になります。

2. (オプション) [Node properties] (ノードのプロパティ) タブで、ジョブ図にノードの名前を入力できます。ノードの親がまだ選択されていない場合は、[Node parents] (ノードの親) リストから、変換の入力ソースとして使用するノードを選択します。
3. [Transform] (変換) タブで、変換する列の名前を入力します。
4. [Transform] (変換) タブで、タイプを選択し、選択した列の解析方法を定義します。

   値が数値の場合は、秒 (Unix/Python タイムスタンプ)、ミリ秒、マイクロ秒のいずれかで表すことができます。その際は対応するオプションを選択します。

   値がフォーマットされた文字列の場合は、「iso」タイプを選択します。文字列は ISO 形式のいずれかに準拠している必要があります (例:「2022-11-02T14:40:59.915Z」)。

   この時点でタイプが不明な場合や、行によってタイプが異なる場合は、[autodetect] (自動検出) を選択すると、システムが最小限の実行コストでベストな推測を行います。
5. (オプション) [Transform] タブでは、選択した列を変換する代わりに、新しい列の名前を入力すれば、新しい列を作成し、元の列を残すことができます。

タイムスタンプ列をフォーマットされた文字列に変換する

タイムスタンプ列をパターンに基づいた文字列にフォーマットします。Format timestamp を使用すると、日付と時刻を希望する形式の文字列で取得できます。形式の定義には、Spark の日付構文とほとんどの Python 日付コードを使用できます。

例えば、日付文字列を「2023-01-01 00:00」のような形式にしたいときは、Spark 構文を「yyyy-MM-dd HH:mm」として使用するか、Python の同様の日付コード「%Y-%m-%d %H:%M」を使用することで、形式を定義できます。

Format timestamp 変換ノードをジョブ図に追加するには

1. リソースパネルを開いて、[Format timestamp] を選択し、ジョブ図に新しい変換を追加します。ノードを追加する際に選択したノードが、その親になります。
2. (オプション) [Node properties] (ノードのプロパティ) タブで、ジョブ図にノードの名前を入力できます。ノードの親がまだ選択されていない場合は、[Node parents] (ノードの親) リストから、変換の入力ソースとして使用するノードを選択します。
3. [Transform] (変換) タブで、変換する列の名前を入力します。
4. [Transform] タブに、使用するタイムスタンプ形式のパターンを、Spark 日付構文または Python 日付コードを使って入力します。

5. (オプション) [Transform] タブでは、選択した列を変換する代わりに、新しい列の名前を入力すれば、新しい列を作成し、元の列を残すことができます。

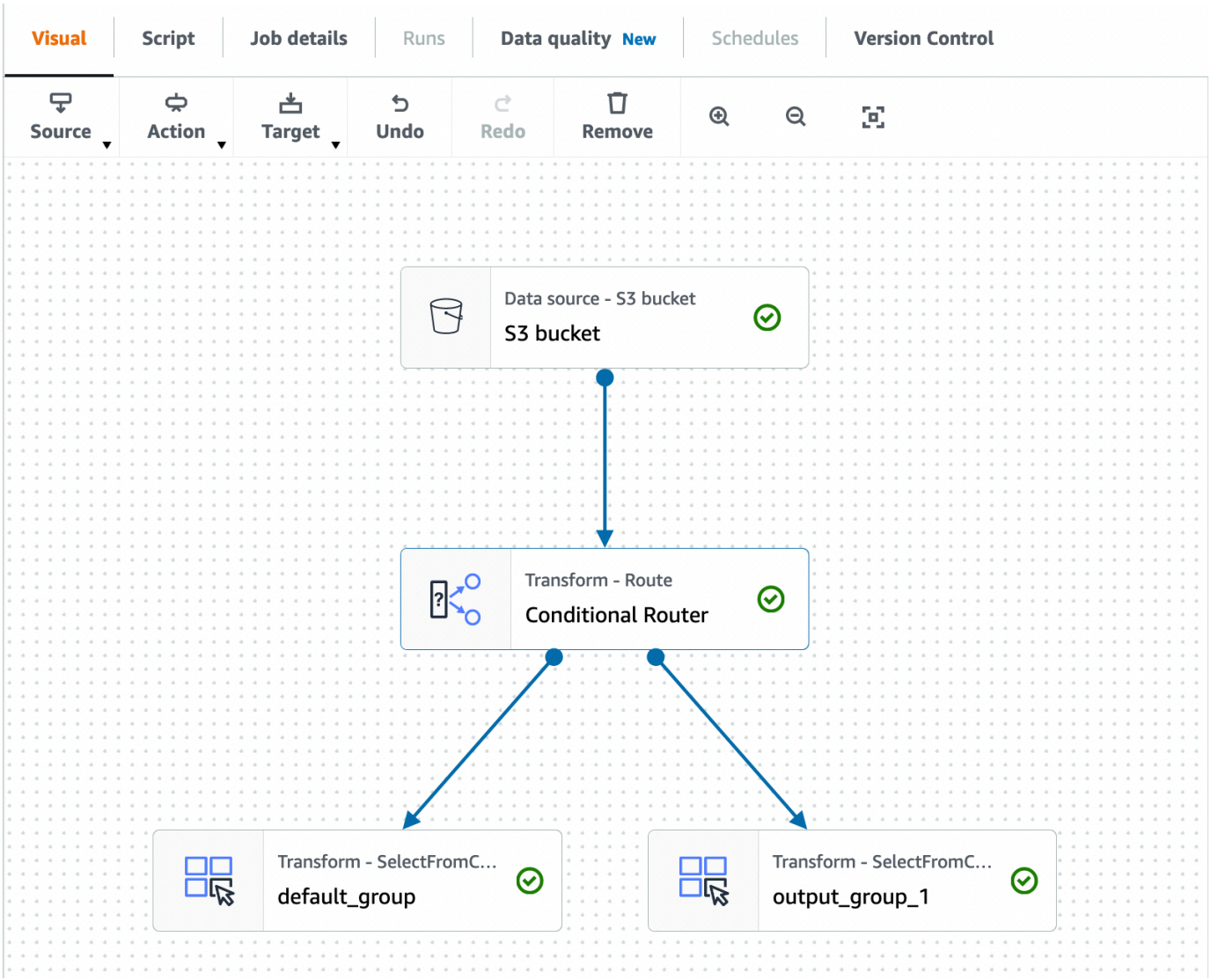
条件付きルーター変換の作成

条件付きルーター変換を使うと、着信データに複数の条件を適用することができます。着信データの行はそれぞれグループフィルター条件により評価され、対応するグループに取り入れて処理されます。行が複数のグループフィルター条件を満たしている場合、変換により、その行が複数のグループに渡されます。行がどの条件も満たしていない場合は、その行を削除するか、デフォルトの出力グループにルーティングします。

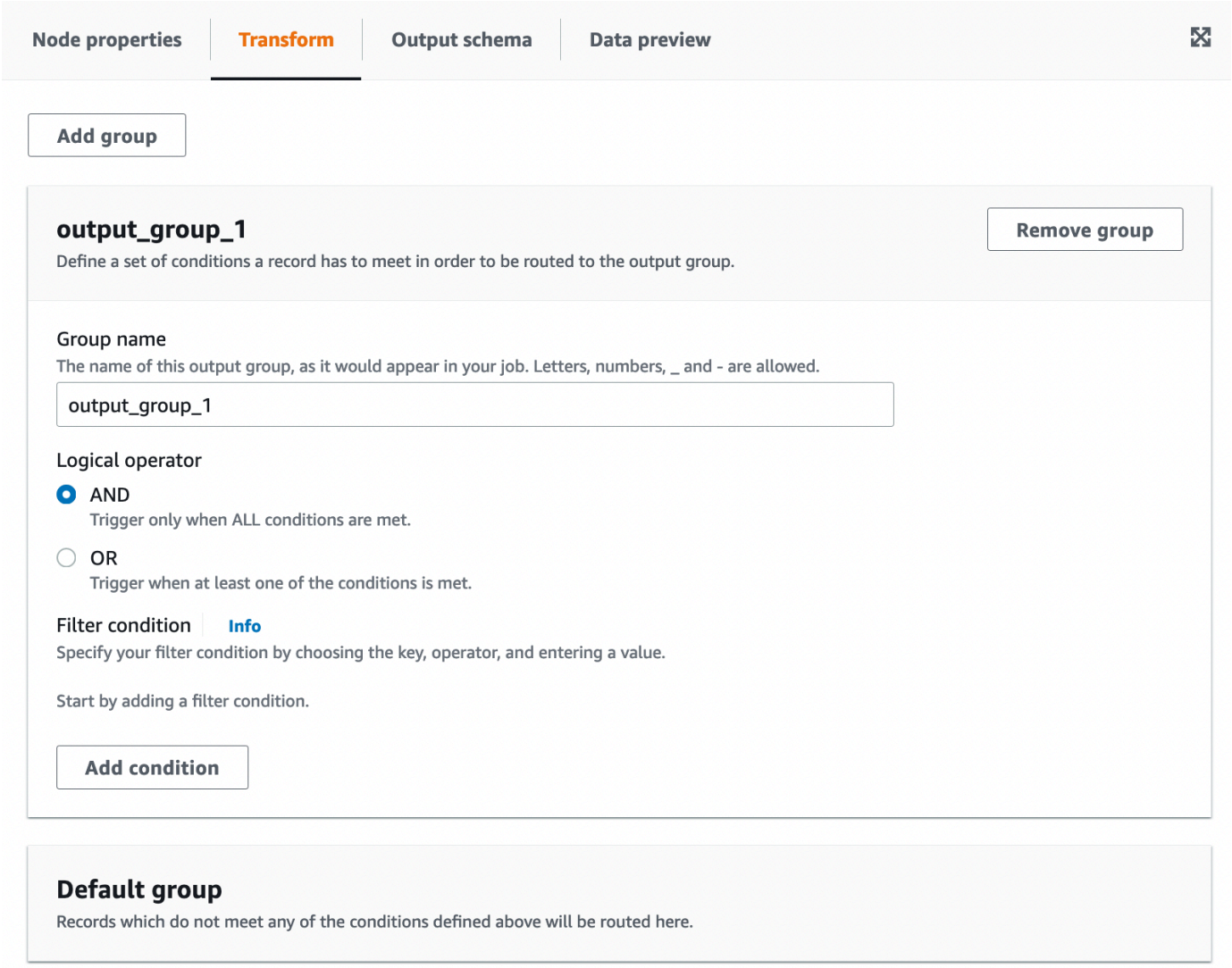
この変換はフィルター変換に似ていますが、同じ入力データを複数の条件でテストしたいユーザーにとって便利です。

条件付きルーター変換を追加するには

1. 条件付きルーター変換を実行するノードを選択します。これは、ソースノードでも別の変換でもかまいません。
2. [Action] (アクション) を選択し、検索バーで [Conditional Router] (条件付きルーター) を検索し、選択します。条件付きルーター変換が、2 つの出力ノードと共に追加されます。一方の出力ノード [Default group] (デフォルトグループ) には、もう一方の出力ノードで定義された条件のいずれをも満たさないレコードが含まれます。デフォルトのグループは編集できません。

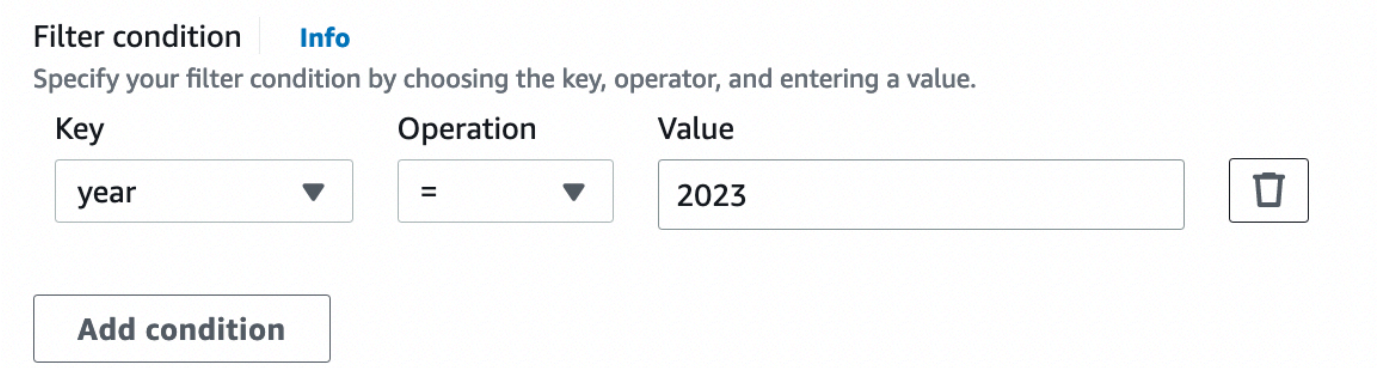


[Add group] (グループを追加) を選択すると、出力グループを追加できます。各出力グループで、グループに名前を付けてフィルタ条件と論理演算子を追加できます。



3. 出力グループの名前を変更するときは、グループに新しい名前を入力します。AWS Glue Studio は、ユーザーに代わって自動的にグループに名前を付けます (例:「output_group_1」)。
4. 論理演算子 (AND、OR) を選択し、[Key] (キー)、[Operation] (演算)、[Value] (値) を指定して[Filter condition] (フィルター条件) を追加します。論理演算子を使用すると、複数のフィルター条件を実装し、指定した各フィルター条件に対して論理演算子を実行できます。

   キーを指定するときは、スキーマに表示されているキーの中から選択します。その後、選択したキーの種類に応じて使用可能な演算を選択します。例えば、キーの種類が「string」の場合、選択できる演算は「match」です。



5. [Value] (値) フィールドに値を入力します。その他の条件を追加するときは、[Add condition] (条件を追加) を選択します。フィルター条件を削除するときは、ごみ箱のアイコンを選択します。

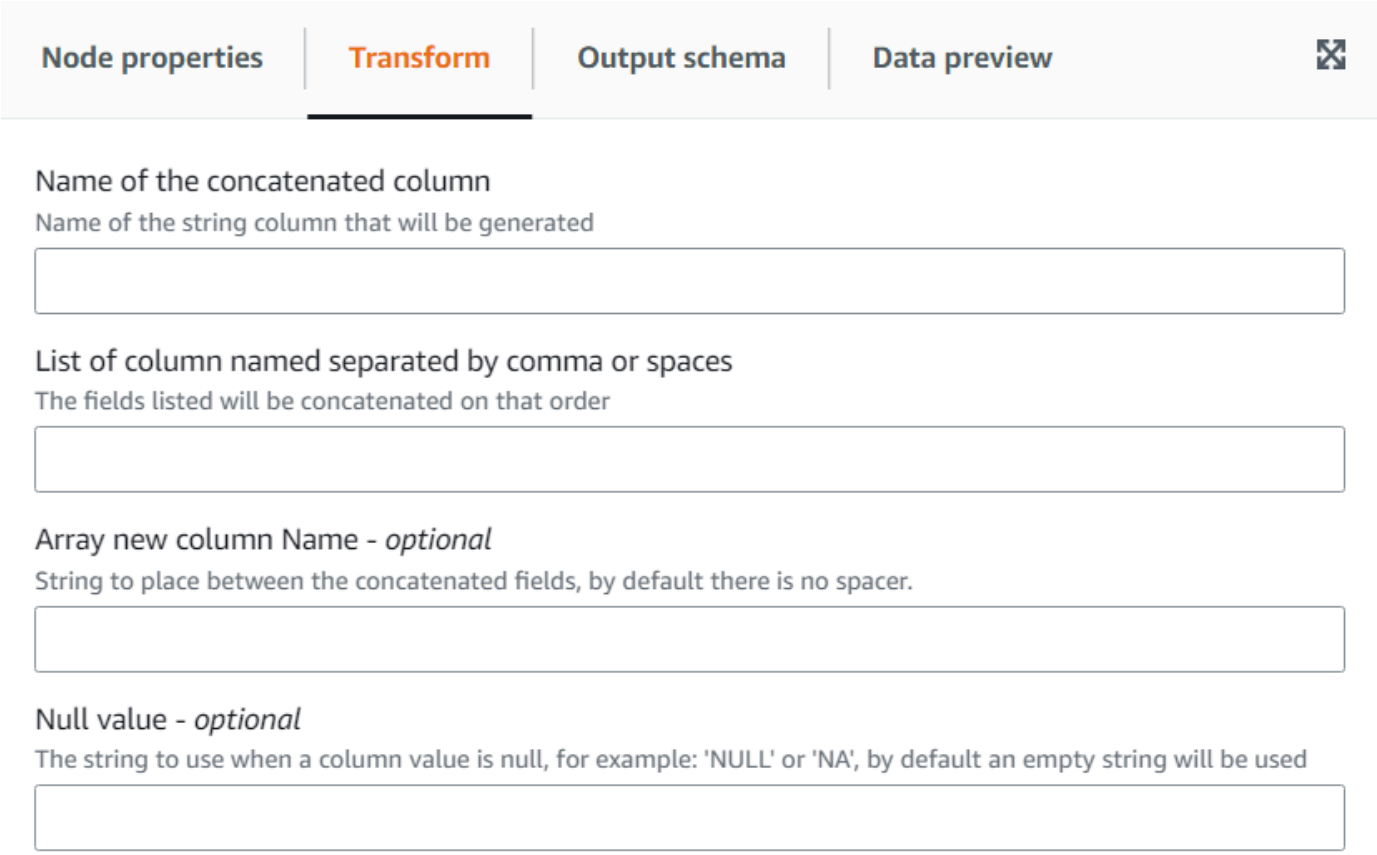
[列の連結] 変換を使用して列を追加する

[連結] 変換では、オプションのスペーサーを使用して、他の列の値を使った新しい文字列の列を作成できます。「year」、「month」、「day」を順にスペーサー「-」で連結して、連結した列「date」を定義する例を以下に示します。

day	month	year	date
01	01	2020	2020-01-01
02	01	2020	2020-01-02
03	01	2020	2020-01-03
04	01	2020	2020-01-04

[連結] 変換を追加するには

1. リソースパネルを開きます。次に [列の連結] を選択して、ジョブ図に新しい変換を追加します。ノードを追加する際に選択したノードが、その親になります。
2. (オプション) [Node properties] (ノードのプロパティ) タブで、ジョブ図にノードの名前を入力できます。ノードの親がまだ選択されていない場合は、[Node parents] (ノードの親) リストから、変換の入力ソースとして使用するノードを選択します。
3. [変換] タブで、連結する列および連結した文字列を保持する列の名前を入力します。ドロップダウンで列にチェックを入れた順序が、使用される順序になります。



4. [スペーサー - オプション] — 連結したフィールドの間に配置する文字列を入力します。デフォルトでは、スペーサーはありません。
5. [NULL 値 - オプション] — 列の値が「NULL」の場合に使用する文字列を入力します。デフォルトでは、列の値が「NULL」または「NA」の場合、空の文字列が使用されます。

[文字列の分割] 変換を使用して文字列の列を分割する

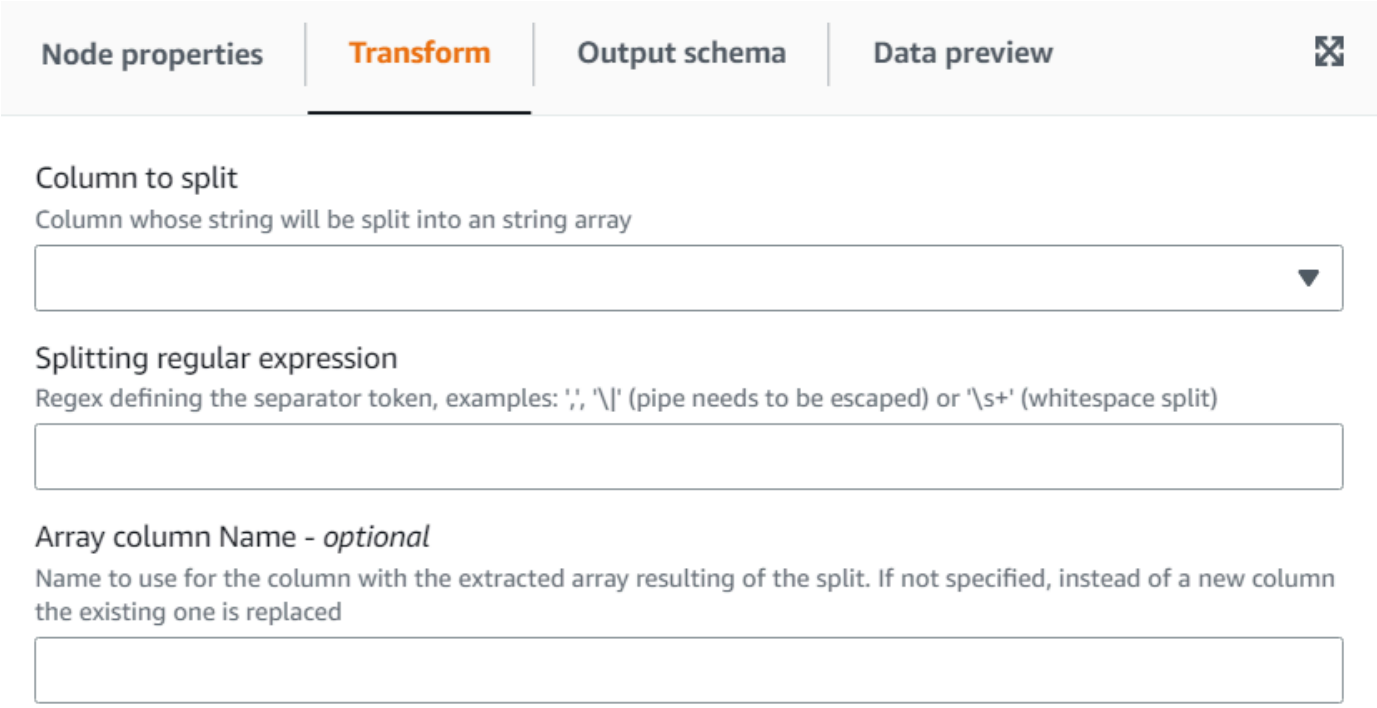
[文字列の分割] 変換を使用して、正規表現の文字列をトークンの配列に分割し、分割方法を定義します。その後、列を配列型のままにするか、この後に [配列から列へ] 変換を適用して、配列の値を抽出し最上位のフィールドに追加できます。ただし、各トークンの意味が事前にわかっていることが前提です。また、カテゴリのセットなどトークンの順序が関係ない場合、[分解] 変換を使用して値ごとに個別の行を生成できます。

例えば、カンマを分割のパターンとして使用して「categories」列を分割し、「categories_arr」列を追加できます。

product_id	カテゴリ	categories_arr
1	sports,winter	[sports, winter]
2	garden,tools	[garden, tools]
3	videogames	[videogames]
4	game,boardgame,social	[game, boardgame, social]

[文字列の分割] 変換を追加するには：

1. リソースパネルを開いて、[Split String] を選択し、ジョブ図に新しい変換を追加します。ノードを追加する際に選択したノードが、その親になります。
2. (オプション) [Node properties] (ノードのプロパティ) タブで、ジョブ図にノードの名前を入力できます。ノードの親がまだ選択されていない場合は、[Node parents] (ノードの親) リストから、変換の入力ソースとして使用するノードを選択します。
3. [変換] タブで、分割する列を選択して文字列の分割に使用するパターンを入力します。ほとんどの場合、正規表現として特別な意味があり エスケープする必要がある場合を除いて、文字を入力するだけで済みます。エスケープする必要がある文字は `\.[]{}()<>*+-=!?^$|` で、文字の前にバックスラッシュを追加することで エスケープできます。例えば、ドット (「.」) で区切る場合は、「`\.`」と入力する必要があります。ただし、カンマには特別な意味はなく、「`,`」のようにそのまま指定できます。



4. (オプション) 元の文字列の列を保持したい場合は、新しい配列の列の名前を入力できます。これにより、元の文字列の列と新しいトークン化された配列の列の両方が保持できます。

[配列から列へ] 変換を使用して、配列の要素を抽出して最上位の列に追加する

[配列から列へ] 変換では、配列型の列の一部またはすべての要素を抽出して新しい列に追加できます。配列に抽出に十分な数の値がある場合、変換によって新しい列が可能な限り埋め込まれ、オプションで指定された位置の要素を取得できます。

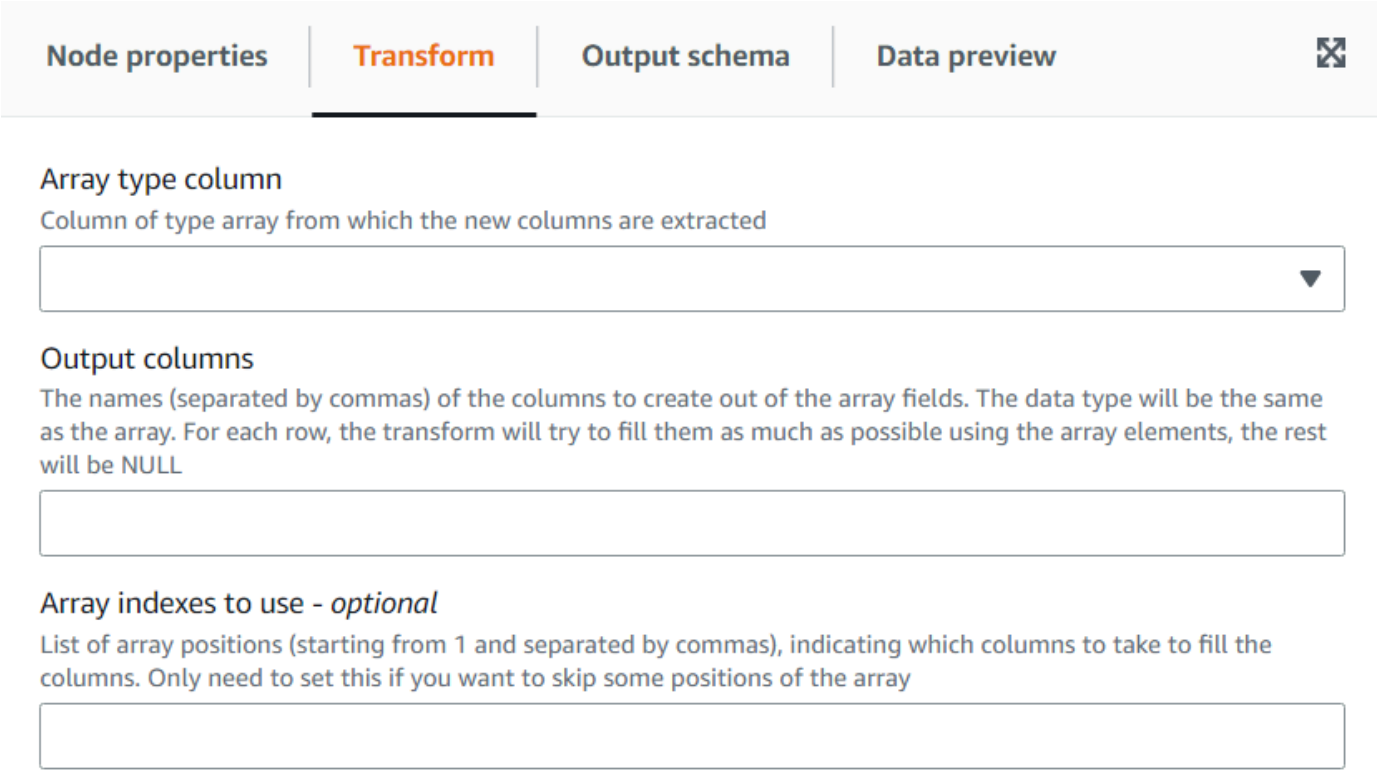
例えば、ip v4 サブネットに「文字列の分割」変換を適用した結果である配列の列「subnet」がある場合、最初と 4 番目の要素を抽出して、新しい列「first_octect」と「forth_octect」に追加できます。この例の変換の出力は次のようになります (最後の 2 行の配列が期待する長さよりも短いことに注意してください)。

サブネット	first_octect	fourth_octect
[54, 240, 197, 238]	54	238
[192, 168, 0, 1]	192	1
[192, 168]	192	

サブネット	first_octect	fourth_octect
[]		

[配列から列へ] 変換を追加するには:

1. リソースパネルを開いて、[Array To Columns] を選択し、ジョブ図に新しい変換を追加します。ノードを追加する際に選択したノードが、その親になります。
2. (オプション) [Node properties] (ノードのプロパティ) タブで、ジョブ図にノードの名前を入力できます。ノードの親がまだ選択されていない場合は、[Node parents] (ノードの親) リストから、変換の入力ソースとして使用するノードを選択します。
3. [変換] タブで、抽出する配列の列を選択し、抽出されたトークンの新しい列のリストを入力します。



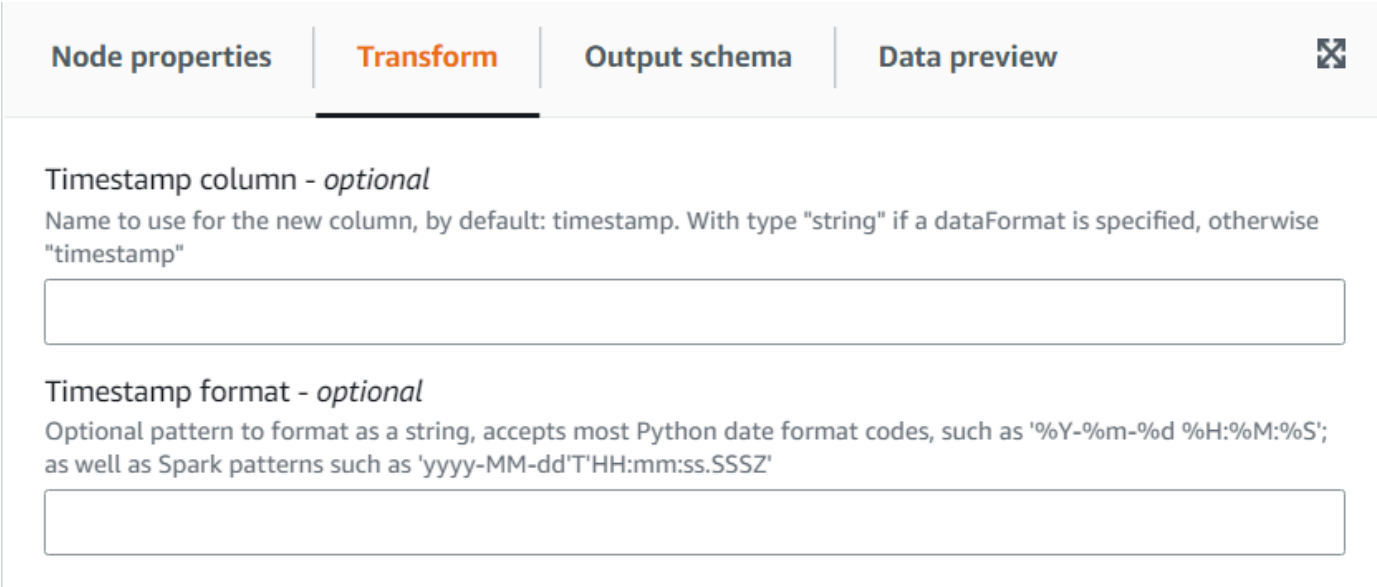
4. (オプション) 列を割り当てるために配列のトークンを取得しない場合は、指定した同じ順序で列のリストに割り当てるインデックスを指定できます。例えば、出力列が「column1, column2, column3」でインデックスが「4, 1, 3」の場合、配列の 4 番目の要素は column1 に、最初の要素は column2 に、3 番目の要素は column3 に指定できます (配列がインデックスの番号より短い場合は NULL 値が設定されます)。

[現在のタイムスタンプを追加] 変換の使用

[現在のタイムスタンプを追加] 変換では、データが処理された時刻で行をマークできます。これは、監査目的やデータパイプラインでのレイテンシーの追跡に役立ちます。この新しい列は、タイムスタンプのデータ型またはフォーマットされた文字列として追加できます。

[現在のタイムスタンプを追加] 変換を追加するには:

1. リソースパネルを開いて、[Add Current Timestamp] を選択し、ジョブ図に新しい変換を追加します。ノードを追加する際に選択したノードが、その親になります。
2. (オプション) [Node properties] (ノードのプロパティ) タブで、ジョブ図にノードの名前を入力できます。ノードの親がまだ選択されていない場合は、[Node parents] (ノードの親) リストから、変換の入力ソースとして使用するノードを選択します。



3. (オプション) 列をフォーマット済の日付の文字列にしたい場合は、[変換] タブで、新しい列のカスタム名とフォーマットを入力します。

[行から列へのピボット] 変換の使用

[行から列へのピボット] 変換では、選択した列の固有の値を回転させて数値列を集約し、新しい列にすることができます (複数の列を選択した場合は、値が連結され新しい列に名前が付けられます)。このように、それぞれの固有の値ごとに部分的な集計を含む列を増やしながら、行を統合します。例えば、月別および国別の売上に関するデータセットがあるとします (わかりやすいように並べ替えられています)。

年	month	country	amount
2020	Jan	uk	32
2020	Jan	de	42
2020	Jan	us	64
2020	Feb	uk	67
2020	Feb	de	4
2020	Feb	de	7
2020	Feb	us	6
2020	Feb	us	12
2020	Jan	us	90

金額と国を集計列としてピボットすると、元の国列から新しい列が作成されます。次の表では、国列の代わりに de、uk、us の新しい列が作成されています。

年	month	de	uk	us
2020	Jan	42	32	64
2020	Jan	11	67	18
2021	Jan			90

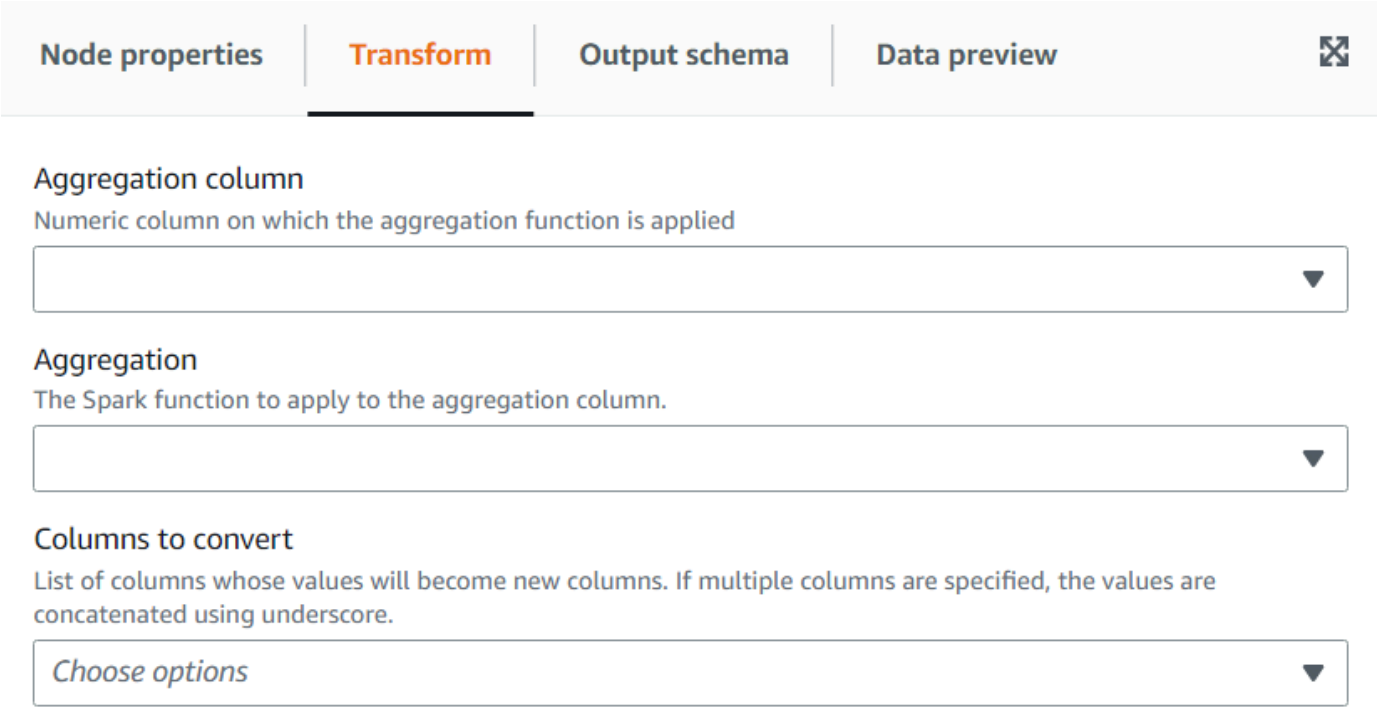
代わりに、月と国の両方をピボットする場合は、それらの列の値の組み合わせごとに列が表示されます。

year	Jan_de	Jan_uk	Jan_us	Feb_de	Feb_uk	Feb_us
2020	42	32	64	11	67	18

year	Jan_de	Jan_uk	Jan_us	Feb_de	Feb_uk	Feb_us
2021			90			

[行から列へのピボット] 変換を追加するには:

1. リソースパネルを開いて、[Pivot Rows To Columns] を選択し、ジョブ図に新しい変換を追加します。ノードを追加する際に選択したノードが、その親になります。
2. (オプション) [Node properties] (ノードのプロパティ) タブで、ジョブ図にノードの名前を入力できます。ノードの親がまだ選択されていない場合は、[Node parents] (ノードの親) リストから、変換の入力ソースとして使用するノードを選択します。
3. [変換] タブで、新しい列の値を生成するために集約される数値列、適用する集計関数、および固有の値を新しい列に変換する列を選択します。



[列から行へのピボット解除] 変換の使用

[ピボット解除] 変換では、列を新しい列の値に変換して、固有の値ごとに行を生成できます。この変換はピボットの逆ですが、集約された同じ値の行を分解したり、組み合わせを元の列に分割できないため、同等ではないことに注意してください (これらは、後で [分割] 変換を使用して行うことができます)。例えば、次のようなテーブルがあるとします。

年	month	de	uk	us
2020	Jan	42	32	64
2020	Feb	11	67	18
2021	Jan			90

値「amount」を使用して「de」、「uk」、「us」の列を「country」列にピボット解除すると、次のようになります (ここでは分かりやすくするために並べ替えられています)。

年	month	country	amount
2020	Jan	uk	32
2020	Jan	de	42
2020	Jan	us	64
2020	Feb	uk	67
2020	Feb	de	11
2020	Feb	us	18
2021	Jan	us	90

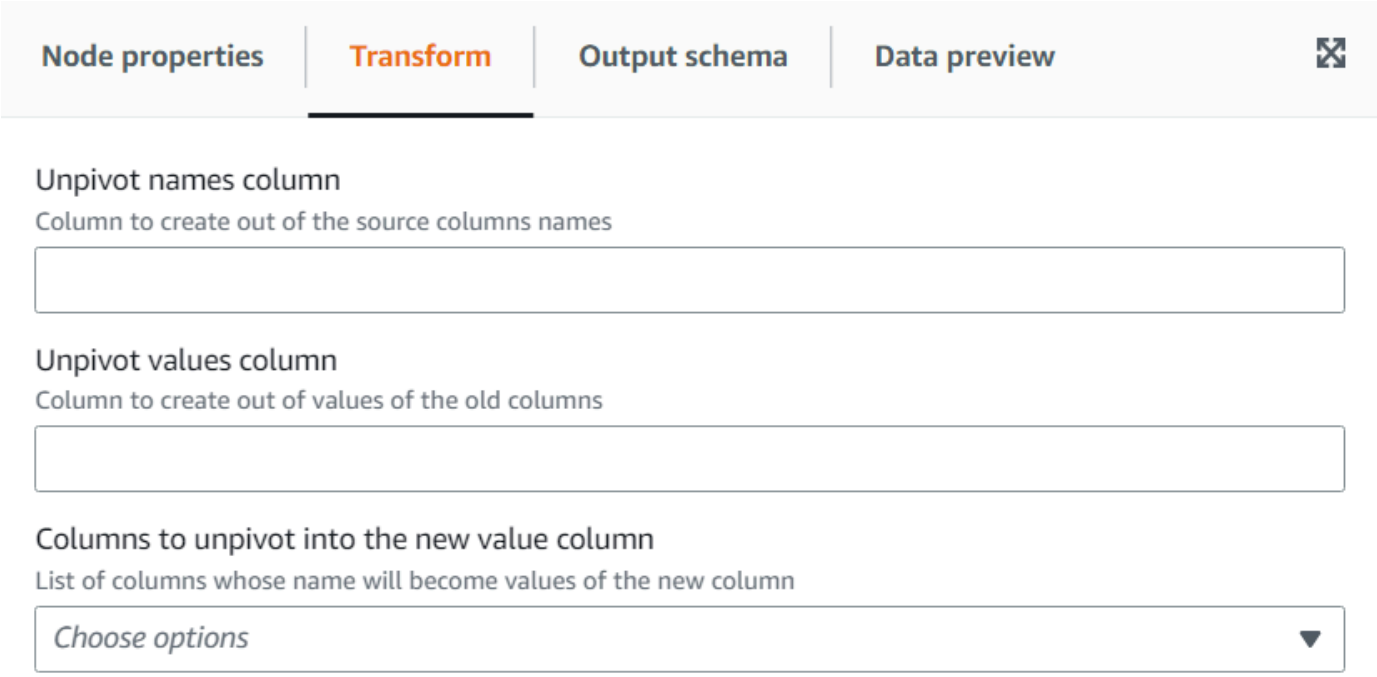
NULL 値の列 (Jan 2021 の「de」と「uk」) はデフォルトでは生成されません。このオプションを有効にすると、次のようになります。

年	month	country	amount
2020	Jan	uk	32
2020	Jan	de	42
2020	Jan	us	64

年	month	country	amount
2020	Feb	uk	67
2020	Feb	de	11
2020	Feb	us	18
2021	Jan	us	90
2021	Jan	de	
2021	Jan	uk	

[列から行へのピボット解除] 変換を追加するには:

1. リソースパネルを開いて、[Unpivot Columns to Rows] を選択し、ジョブ図に新しい変換を追加します。ノードを追加する際に選択したノードが、その親になります。
2. (オプション) [Node properties] (ノードのプロパティ) タブで、ジョブ図にノードの名前を入力できます。ノードの親がまだ選択されていない場合は、[Node parents] (ノードの親) リストから、変換の入力ソースとして使用するノードを選択します。
3. [変換] タブで、ピボット解除する列の名前と値を保持するために作成する新しい列を入力します。

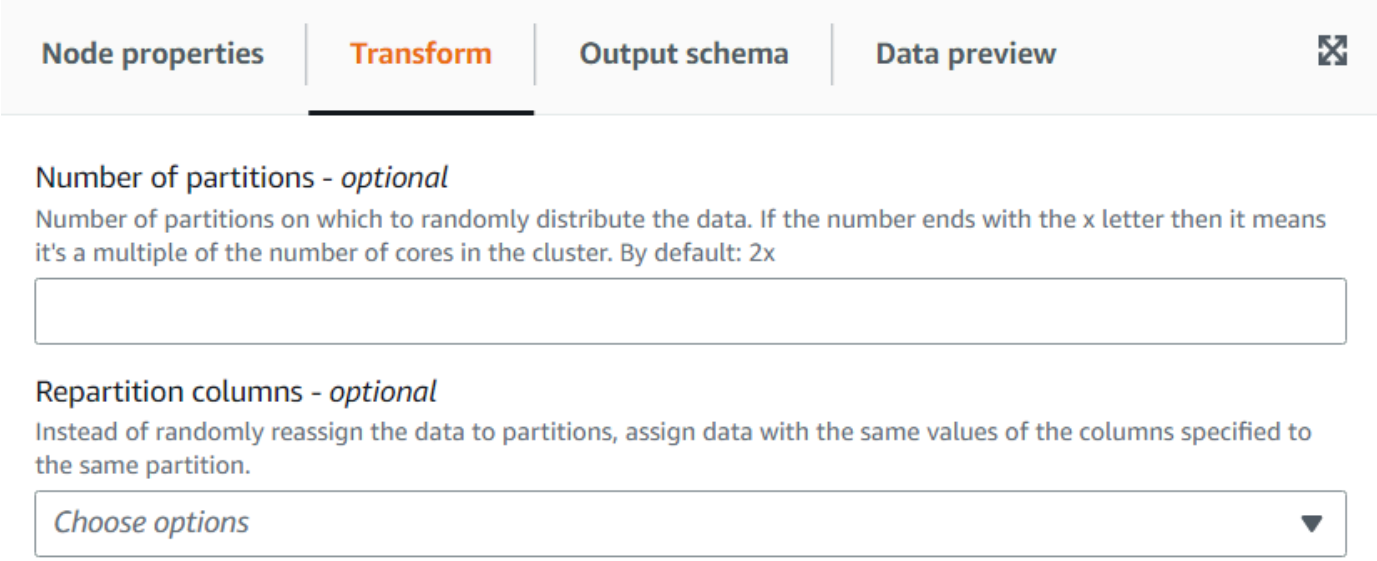


[オートバランス処理] 変換を使用してランタイムを最適化する

[オートバランス処理] 変換は、パフォーマンスを向上させるためにデータをワーカー間で再配分します。これは、データのバランスが取れていない場合や、ソースからのデータでは十分に並行処理ができない場合に役立ちます。これは、ソースが gzip 圧縮されている場合や JDBC である場合に一般的です。データの再配分には中程度のパフォーマンスコストがかかるため、データのバランスが既に取れている場合は、最適化により必ずしもコストが補われるとは限りません。変換で Apache Spark の再パーティション化を使用して、クラスターのキャパシティに最適な複数のパーティション間でデータをランダムに再度割り当てできます。上級ユーザーの場合は、複数のパーティションを手動で入力できます。さらに、指定した列に基づいてデータを再編成することで、分割テーブルの書き込みを最適化することもできます。これにより、出力ファイルがより統合されます。

1. リソースパネルを開いて、[Autobalance Processing] を選択し、ジョブ図に新しい変換を追加します。ノードを追加する際に選択したノードが、その親になります。
2. (オプション) [Node properties] (ノードのプロパティ) タブで、ジョブ図にノードの名前を入力できます。ノードの親がまだ選択されていない場合は、[Node parents] (ノードの親) リストから、変換の入力ソースとして使用するノードを選択します。
3. (オプション) [変換] タブで、複数のパーティションを入力できます。一般的に、この値はシステムに決定させることをお勧めしますが、値を制御する必要がある場合は乗数を調整したり特定の値を入力できます。列で分割されたデータを保存する場合は、再パーティションの列と同じ列

を選択できます。これにより、各パーティションのファイルの数を最小限に抑え、パーティションごとに多数のファイルが作成されるのを防ぐことができます。ファイルが多数作成されると、データをクエリするツールのパフォーマンスが低下します。



[派生列] 変換を使用して他の列を結合する

[派生列] 変換では、定数やリテラル、データ内の他の列も使用できる数式または SQL 式に基づいて新しい列を定義できます。例えば「success」および「count」列から「percentage」列を派生させるには、「success * 100 / count || '%'」という SQL 式を入力します。

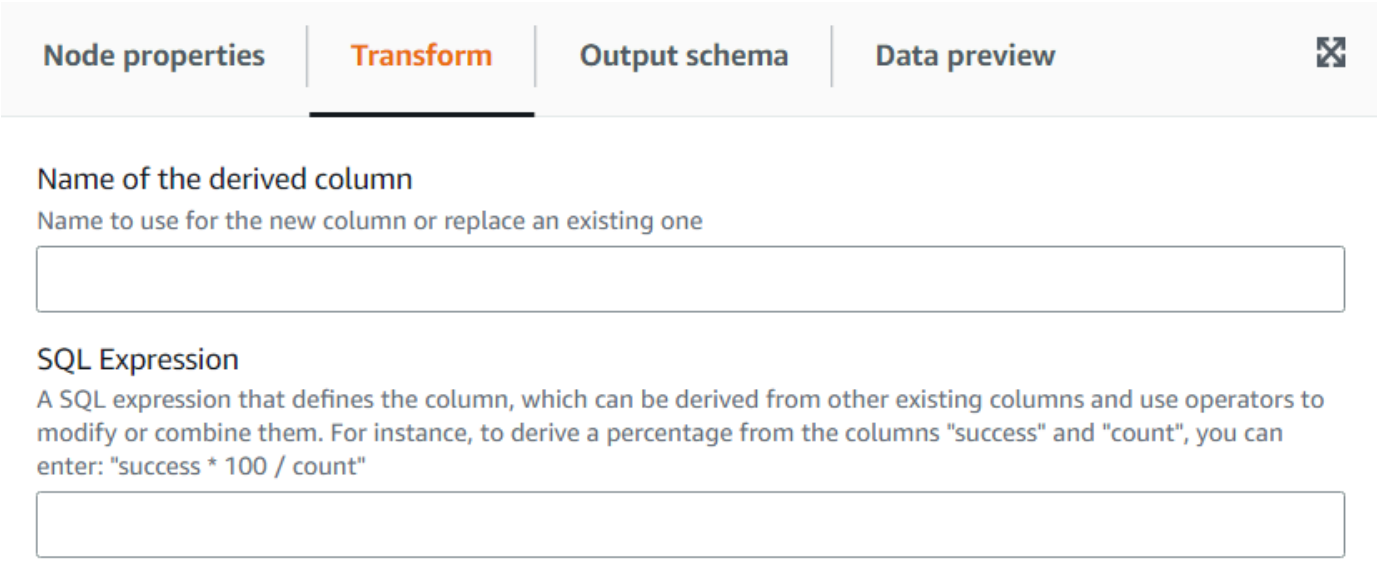
結果の例:

success	count	割合(%)
14	100	14%
6	20	3%
3	40	7.5%

[派生列] 変換を追加するには

1. リソースパネルを開いて、[Derived Column] を選択し、ジョブ図に新しい変換を追加します。ノードを追加する際に選択したノードが、その親になります。

2. (オプション) [Node properties] (ノードのプロパティ) タブで、ジョブ図にノードの名前を入力できます。ノードの親がまだ選択されていない場合は、[Node parents] (ノードの親) リストから、変換の入力ソースとして使用するノードを選択します。
3. [変換] タブで、列の名前とその内容の式を入力します。



[ルックアップ] 変換を使用してカタログテーブルから一致するデータを追加する

[ルックアップ] 変換では、キーがデータ内の定義済みのルックアップ列と一致する場合、定義済みのカタログテーブルから列を追加できます。これは、データとルックアップテーブル間を左外部結合し、条件に一致する列として使用するのと同じです。

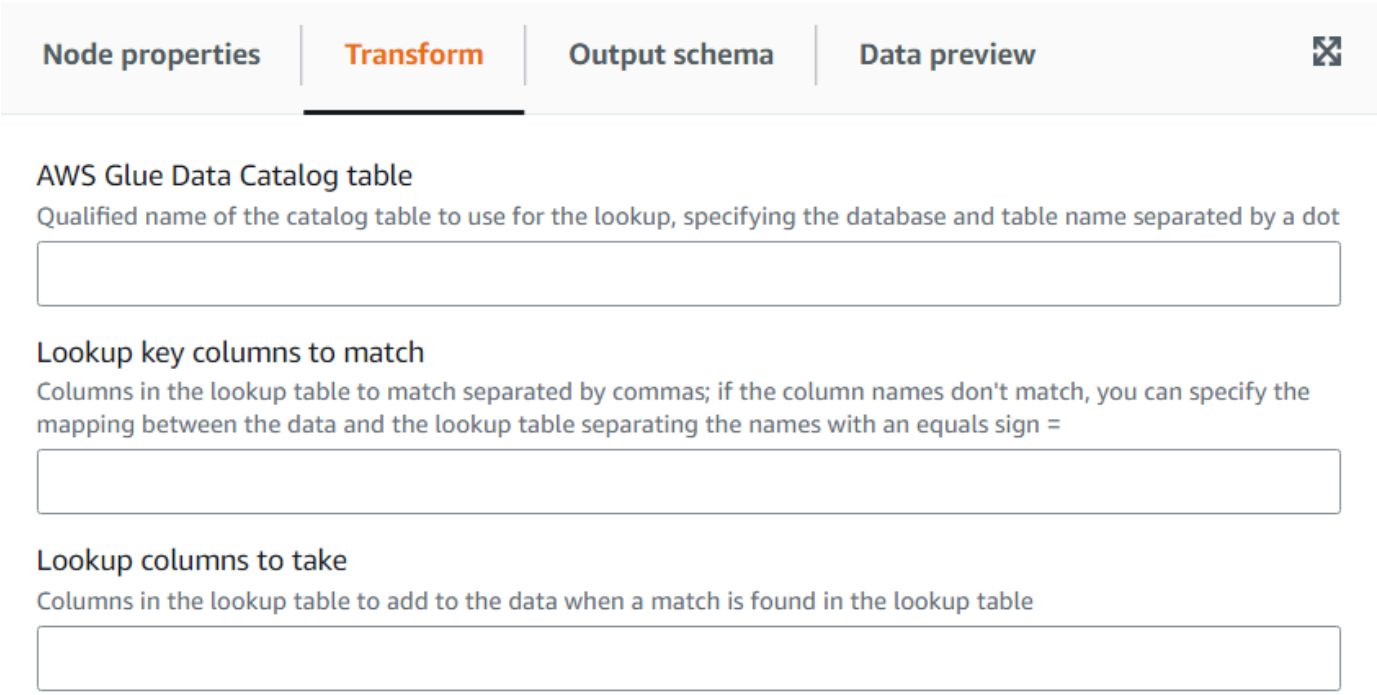
[ルックアップ] 変換を追加するには

1. リソースパネルを開いて、[Lookup] を選択し、ジョブ図に新しい変換を追加します。ノードを追加する際に選択したノードが、その親になります。
2. (オプション) [Node properties] (ノードのプロパティ) タブで、ジョブ図にノードの名前を入力できます。ノードの親がまだ選択されていない場合は、[Node parents] (ノードの親) リストから、変換の入力ソースとして使用するノードを選択します。
3. [変換] タブで、ルックアップの実行に使用するカタログテーブルの完全修飾名を入力します。例えば、データベースが「mydb」で、テーブルが「mytable」の場合は、「mydb.mytable」と入力します。次に、ルックアップキーが作成されている場合は、ルックアップテーブルで一致するものを見つけるための基準を入力します。キー列のリストは、カンマで区切って入力します。1つ以上のキー列に同じ名前がない場合は、マッチマッピングを定義する必要があります。

例えば、データ列が「user_id」と「region」で、ユーザーテーブルで対応する列の名前が「id」と「region」の場合、[一致する列] フィールドに「user_id=id, region」と入力します。region=region と入力することもできますが、これらは同じなので必要ありません。

4. 最後に、ルックアップテーブルで一致した行から取得する列を入力して、それらをデータに組み込みます。一致するものが見つからなかった場合、それらの列は NULL に設定されます。

> **Note**
>
> [ルックアップ] 変換では、効率を上げるため左結合を使用しています。ルックアップテーブルに複合キーがある場合は、一致する列がすべてのキーの列と一致するように設定されていることを確認し、一致が 1 つだけになるようにしてください。そうしないと、複数のルックアップ行が一致し、一致するごとに余分な行が追加されます。



[配列またはマップを行に分解] 変換の使用

[分解] 変換では、ネストされたデータ構造から値を抽出し、操作しやすい個々の行を追加できます。配列の場合、変換により配列の各値に対して行が生成され、その行に他の列の値が複製されます。マップの場合、変換によりキーと値を列として持つ各エントリに対して行が生成されます。各エントリには列としてのキーと値があり、その行に他の列も複製されます。

例えば、このデータセットに複数の値がある「category」配列の列があるとします。

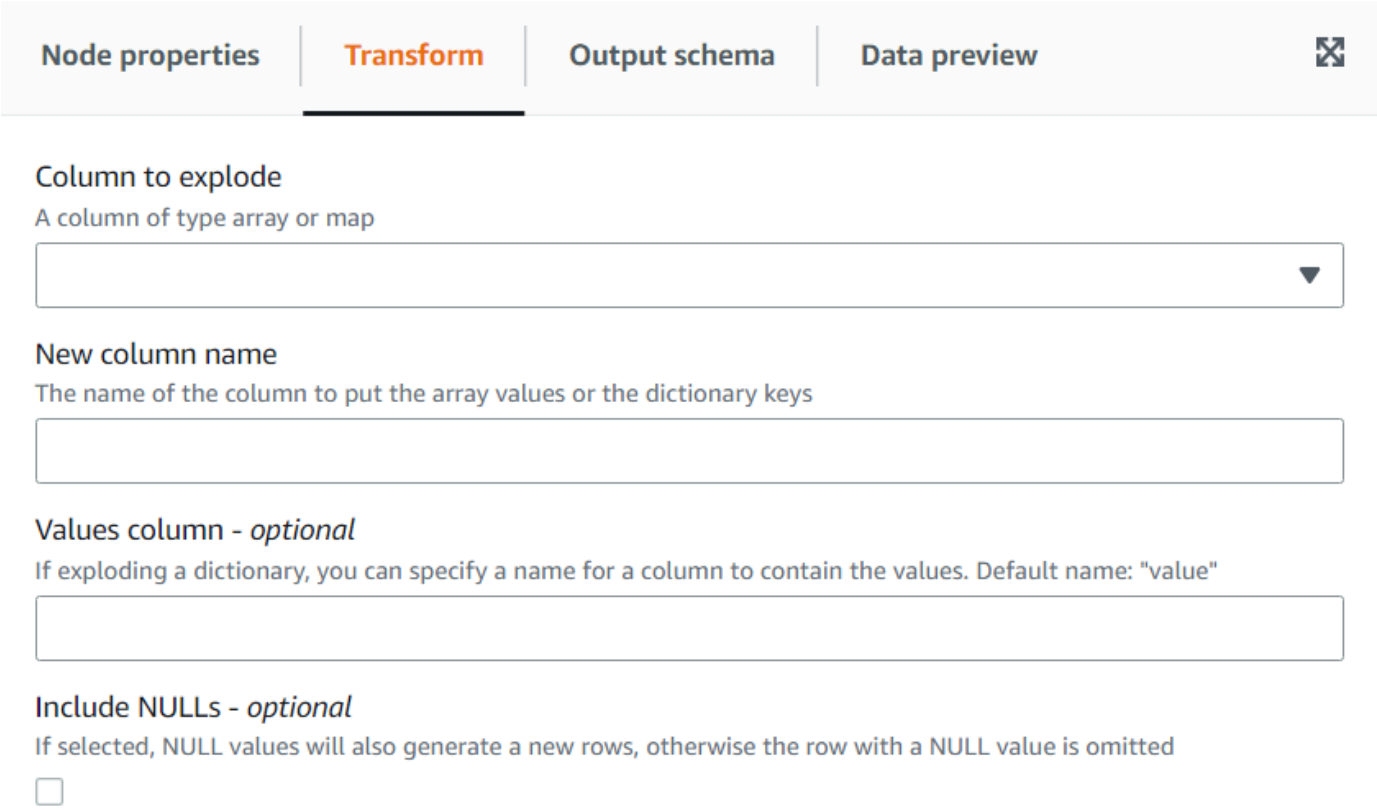
product_id	category
1	[sports, winter]
2	[garden, tools]
3	[videogames]
4	[game, boardgame, social]
5	[]

「category」列が同じ名前の列に分解されると、その列が上書きされます。NULL を含めるように選択すると、次のようになります (分かりやすいように順番になっています)。

product_id	category
1	sports
1	winter
2	garden
2	tool
3	videogames
4	ゲーム
4	boardgame
4	social
5	

[配列またはマップを行に分解] 変換を追加するには:

1. リソースパネルを開いて、[Explode Array Or Map Into Rows] を選択し、ジョブ図に新しい変換を追加します。ノードを追加する際に選択したノードが、その親になります。
2. (オプション) [Node properties] (ノードのプロパティ) タブで、ジョブ図にノードの名前を入力できます。ノードの親がまだ選択されていない場合は、[Node parents] (ノードの親) リストから、変換の入力ソースとして使用するノードを選択します。
3. [変換] タブで、分解する列を選択します (タイプが配列またはマップである必要があります)。次に、配列の項目を表す列の名前を入力するか、マップを分解する場合はキーと値を表す列の名前を入力します。
4. (オプション) [変換] タブでは、デフォルトで分解する列が NULL、またはデータ構造が空である場合、分解されたデータセットではその列が省略されます。(新しい列を NULL として) 行を残しておきたい場合は、[NULL を含む] にチェックを入れます。

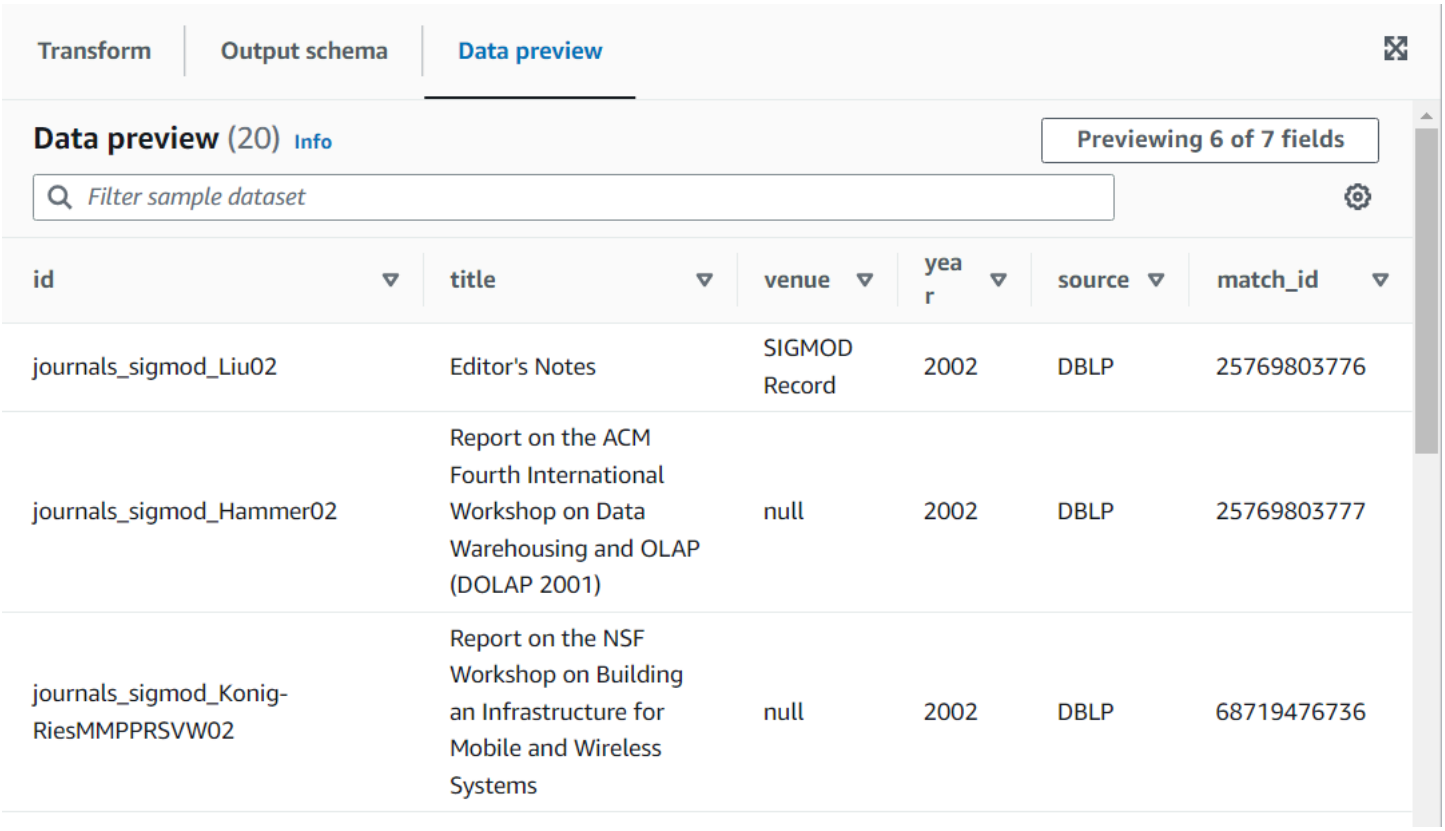


レコードマッチング変換を使用して既存のデータ分類変換を呼び出す

この変換は、既存のレコードマッチング機械学習データ分類変換を呼び出します。

この変換は、ラベルに基づいて、トレーニング済みモデルに対して現在のデータを評価します。列「match_id」が追加され、アルゴリズムトレーニングに基づいて、同等とみなされる項目のグループに各行が割り当てられます。詳細については、「AWS Lake Formation FindMatches によるレコードのマッチング」を参照してください。

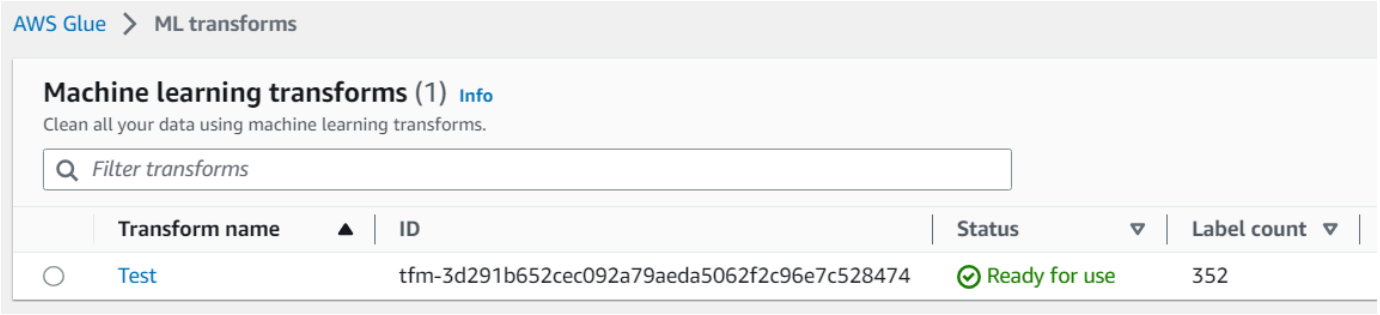
> **Note**
>
> ビジュアルジョブで使用する AWS Glue のバージョンは、レコードマッチング変換の作成時に AWS Glue で使用するバージョンと一致する必要があります。



id	title	venue	year	source	match_id
journals_sigmod_Liu02	Editor's Notes	SIGMOD Record	2002	DBLP	25769803776
journals_sigmod_Hammer02	Report on the ACM Fourth International Workshop on Data Warehousing and OLAP (DOLAP 2001)	null	2002	DBLP	25769803777
journals_sigmod_Konig-RiesMMPPRSVW02	Report on the NSF Workshop on Building an Infrastructure for Mobile and Wireless Systems	null	2002	DBLP	68719476736

レコードマッチング変換ノードをジョブ図に追加するには

1. リソースパネルを開いて、[Record Matching] を選択し、ジョブ図に新しい変換を追加します。ノードを追加する際に選択したノードが、その親になります。
2. ノードのプロパティパネルで、ジョブ図にノード名を入力します。ノードの親がまだ選択されていない場合、[Node parents] (ノードの親) リストから、変換の入力ソースとして使用するノードを選択します。
3. [変換] タブに、[機械学習の変換] ページから取得した ID を入力します。



4. (オプション) [変換] タブで、信頼スコアを追加するオプションを確認できます。計算量は増えますが、モデルは各マッチングの信頼スコアを追加の列として推定します。

null 行の削除

この変換は、すべての列が null である行をデータセットから削除します。さらに、この条件を拡張して空のフィールドを含め、1 つ以上の列が空でない行を保持することもできます。

null 行削除変換ノードをジョブ図に追加するには

1. リソースパネルを開いて、[Remove Null Rows] を選択し、ジョブ図に新しい変換を追加します。ノードを追加する際に選択したノードが、その親になります。
2. ノードのプロパティパネルで、ジョブ図にノード名を入力します。ノードの親がまだ選択されていない場合、[Node parents] (ノードの親) リストから、変換の入力ソースとして使用するノードを選択します。
3. (オプション) 行が null でもなく空でもないようにする必要がある場合は、[変換] タブで [Extended] オプションをオンにします。この変換により、空の文字列、配列、マップは null とみなされます。

JSON データを含む文字列の列の解析

この変換は、JSON データを含む文字列の列を解析し、JSON がオブジェクトか配列かに応じて、それぞれを構造体または配列の列に変換します。(オプション) 解析された列と元の列の両方を保持できます。

JSON スキーマは、オプションのサンプリングを使用して提供または推測できます (JSON オブジェクトの場合)。

JSON 列解析変換ノードをジョブ図に追加するには

1. リソースパネルを開いて、[Parse JSON Column] を選択し、ジョブ図に新しい変換を追加します。ノードを追加する際に選択したノードが、その親になります。
2. ノードのプロパティパネルで、ジョブ図にノード名を入力します。ノードの親がまだ選択されていない場合、[Node parents] (ノードの親) リストから、変換の入力ソースとして使用するノードを選択します。
3. [変換] タブで、JSON 文字列を含む列を選択します。
4. (オプション) [変換] タブで、JSON データが従うスキーマを、SQL 構文を使用して入力します (例: オブジェクトの場合は「field1 STRING, field2 INT」、配列の場合は「ARRAY<STRING>」)。

   配列の場合はスキーマが必要ですが、オブジェクトの場合はデータを使用して推測されます (スキーマが指定されていない場合)。スキーマの推論による影響を軽減するため (特に大規模なデータセットの場合)、[Ratio of samples to use to infer schema] を入力することで、データ全体を 2 回読み取る必要がなくなります。値が 1 より小さい場合は、対応するランダムサンプルの比率を使用してスキーマが推測されます。データに信頼性があり、行間でオブジェクトが一貫している場合は、0.1 などの小さな比率を使用してパフォーマンスを高めることができます。
5. (オプション) 元の文字列の列と解析された列の両方を保持する場合は、[変換] タブで新しい列名を入力します。

JSON パスの抽出

この変換は、JSON 文字列の列から新しい列を抽出します。この変換は、必要なデータ要素が少なく、JSON コンテンツ全体をテーブルスキーマにインポートしない場合に便利です。

JSON パス抽出変換ノードをジョブ図に追加するには

1. リソースパネルを開いて、[Extract JSON Path] を選択し、ジョブ図に新しい変換を追加します。ノードを追加する際に選択したノードが、その親になります。
2. ノードのプロパティパネルで、ジョブ図にノード名を入力します。ノードの親がまだ選択されていない場合、[Node parents] (ノードの親) リストから、変換の入力ソースとして使用するノードを選択します。
3. [変換] タブで、JSON 文字列を含む列を選択します。1 つ以上の JSON パス式をカンマで区切って入力します。各式は、JSON 配列またはオブジェクトから値を抽出する方法を示しています。例えば、JSON 列に「prop_1」と「prop2」というプロパティを持つオブジェクトが含まれている場合、「prop_1, prop_2」と名前を指定して両方を抽出できます。

JSON フィールドに特殊文字が含まれていて、この JSON {"a．a": 1} からプロパティを抽出する場合などには、パス $['a．a'] を使用できます。例外は、パスを区切るためのカンマです。続いて、各パスに対応する列名をカンマで区切って入力します。

4. (オプション) [変換] タブでは、抽出された JSON 列を削除するようにチェックできます。これは、必要な JSON データを抽出した後、残りのデータが不要な場合に便利です。

正規表現による文字列の断片の抽出

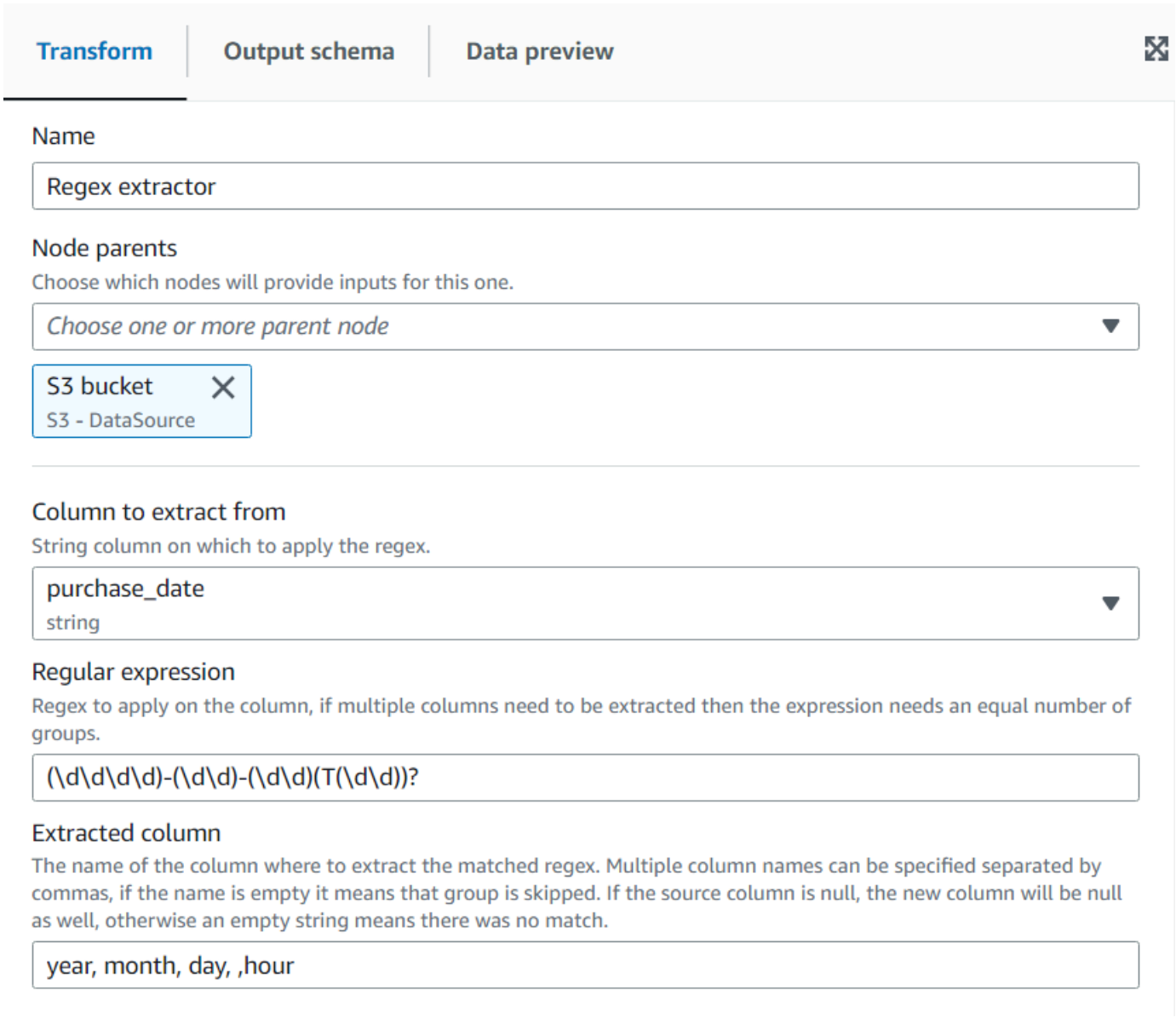
この変換は、正規表現を使用して文字列の断片を抽出し、そこから新しい列を作成します。正規表現グループを使用する場合は複数の列を作成します。

正規表現抽出変換ノードをジョブ図に追加するには

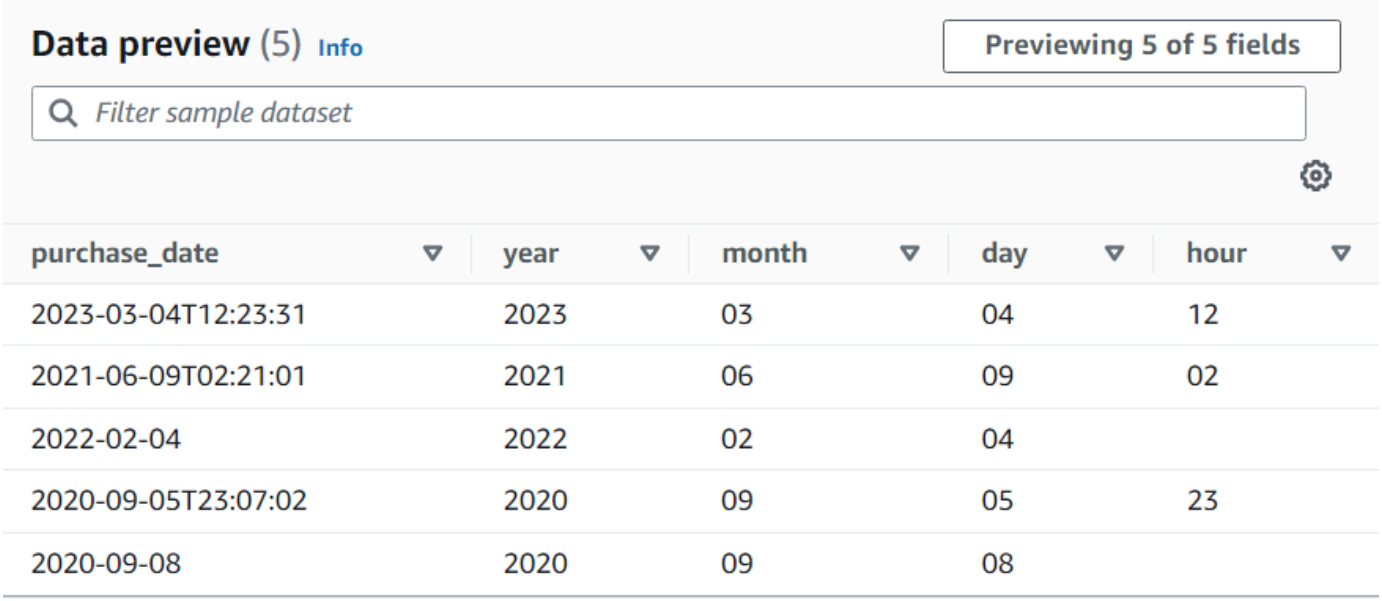
1. リソースパネルを開いて、[Regex Extractor] を選択し、ジョブ図に新しい変換を追加します。ノードを追加する際に選択したノードが、その親になります。
2. ノードのプロパティパネルで、ジョブ図にノード名を入力します。ノードの親がまだ選択されていない場合、[Node parents] (ノードの親) リストから、変換の入力ソースとして使用するノードを選択します。
3. [変換] タブに、正規表現とそれを適用する必要のある列を入力します。続いて、一致する文字列を格納する新しい列の名前を入力します。新しい列は、ソース列が null の場合にのみ null になり、正規表現が一致しない場合は空になります。

   正規表現がグループを使用する場合、対応する列名はカンマで区切られますが、列名を空のままにすることでグループをスキップできます。

   例えば、長い ISO 日付形式と短い ISO 日付形式の両方を使用する文字列を含む列「purchase_date」があり、使用可能な場合は、年、月、日、時間を抽出します。時間グループはオプションであり、使用できない場合、(正規表現が一致しないため) 抽出されたすべてのグループは空の文字列になります。グループで、時間をオプションではなく含めたい場合は、名前を空のままにすることで抽出されなくなります (グループには T 文字が含まれます)。



データプレビューの結果:



カスタム変換を作成する

データに対してより複雑な変換を実行する必要がある場合、またはデータプロパティキーをデータセットに追加する場合は、ジョブ図にカスタムコード変換を追加します。カスタムコードノードを使用すると、変換を実行するスクリプトを入力できます。

カスタムコードを使用する場合は、スキーマエディタを使用して、カスタムコードを通して出力に加えらえる変更を指定する必要があります。スキーマの編集時に、次のアクションを実行できます。

- データプロパティキーの追加または削除
- データプロパティキーのデータ型の変更
- データプロパティキーの名前の変更
- ネストされたプロパティキーの再構築

出力をターゲットの場所に送信するには、SelectFromCollection 変換を使用して、カスタム変換ノードの結果から単一の `DynamicFrame` を選択する必要があります。

次のタスクを実行して、カスタム変換ノードをジョブ図に追加できます。

カスタムコード変換ノードをジョブ図に追加する

カスタム変換ノードをジョブ図に追加するには

1. (オプション) リソースパネルを開いて、[Custom transform] を選択し、ジョブ図にカスタム変換を追加します。
2. [Node properties] (ノードのプロパティ) タブで、ジョブ図のノードの名前を入力します。ノードの親がまだ選択されていない場合、またはカスタム変換に複数の入力が必要な場合は、[Node parents] (ノードの親) リストから、変換の入力ソースとして使用するノードを選択します。

カスタム変換ノードのコードを入力する

入力フィールドにコードを入力またはコピーできます。ジョブでは、データ変換の実行にこのコードを使用します。Python または Scala のいずれかでコードのスニペットを指定できます。コードは、入力として 1 つ以上の `DynamicFrames` を取り、`DynamicFrames` のコレクションを返す必要があります。

カスタム変換ノードのスクリプトを入力するには

1. ジョブ図でカスタム変換ノードを選択した状態で、[Transform] (変換) タブを選択します。
2. 見出し [Code block] (コードブロック) の下のテキスト入力フィールドで、変換用のコードを貼り付けるか、入力します。使用するコードは、[Job details] (ジョブの詳細) タブのジョブで指定されている言語と一致する必要があります。

   コード内の入力ノードを参照する場合、AWS Glue Studio は、作成された順序に基づきジョブの図表ノードによって連続的に返される `DynamicFrames` に名前を付けます。コード内で次のいずれかのネーミングメソッドを使用します。

   - Classic code generation — 関数名を使用して、ジョブダイアグラム内のノードを参照します。
     - データソースノード: `DataSource0`、`DataSource1`、`DataSource2` などです。
     - 変換ノード: `Transform0`、`Transform1`、`Transform2` などです。
   - New code generation — '_node1','_node2'などが添付されたノードの[Node properties]タブ上で指定された名前を使用します。例えば、`S3bucket_node1`、`ApplyMapping_node2`、`S3bucket_node2`、`MyCustomNodeName_node`

各エラーコード generator のさらなる詳細については、「スクリプトコードの生成」を参照してください。

次の例は、コードボックスに入力するコードの形式を示しています。

Python

次の例では、まず DynamicFrame を受信し、それを DataFrame に変換してネイティブフィルターメソッド (1000 票を超えるレコードのみを保持)を適用しています。その後、返す前に DynamicFrame に再び変換しています。

```
def FilterHighVoteCounts (glueContext, dfc) -> DynamicFrameCollection:
    df = dfc.select(list(dfc.keys())[0]).toDF()
    df_filtered = df.filter(df["vote_count"] > 1000)
    dyf_filtered = DynamicFrame.fromDF(df_filtered, glueContext, "filter_votes")
    return(DynamicFrameCollection({"CustomTransform0": dyf_filtered}, glueContext))
```

Scala

次の例では、まず DynamicFrame を受信し、それを DataFrame に変換してネイティブフィルターメソッド (1000 票を超えるレコードのみを保持)を適用しています。その後、返す前に DynamicFrame に再び変換しています。

```
object FilterHighVoteCounts {
  def execute(glueContext : GlueContext, input : Seq[DynamicFrame]) :
 Seq[DynamicFrame] = {
    val frame = input(0).toDF()
    val filtered = DynamicFrame(frame.filter(frame("vote_count") > 1000),
 glueContext)
    Seq(filtered)
  }
}
```

カスタム変換ノードでスキーマを編集する

カスタム transform ノードを使用する場合、AWS Glue Studio は、transform によって作成された出力スキーマを自動で推測することはできません。スキーマエディタを使用して、カスタム変換コードによって実装されるスキーマの変更を記述できます。

カスタムコードノードでは、カスタムコードの入力として DynamicFrame を提供する任意の数の親ノードを持つことができます。カスタムコードノードでは、DynamicFrames のコレクションが返されます。入力として使用される各 DynamicFrame には、スキーマが関連付けられています。カスタムコードノードによって返される各 DynamicFrame を記述するスキーマを追加する必要があります。

> **Note**
>
> カスタム変換で独自のスキーマを設定した場合、AWS Glue Studio は、前のノードからのスキーマ継承を行いません。スキーマを更新するには、カスタム変換のノードを選択した上で、[Data preview] (データプレビュー) タブを開きます。プレビューが生成されたら、[Use Preview Schema] (プレビュースキーマを使用) をクリックします。その後、このスキーマがプレビューデータを使用するスキーマに置き換えられます。

カスタム変換ノードの入力スキーマを編集するには

1. ジョブ図でカスタム変換ノードを選択した状態で、ノードの詳細パネルの [Output schema] (出力スキーマ) タブを選択します。
2. [Edit] (編集) を選択して、スキーマを変更します。

   配列やオブジェクトなどのネストされたデータのプロパティキーがある場合、各スキーマパネルの右上部にある [Expand-Rows] (行の展開) アイコン () を選択して、子データのプロパティキーのリストを展開します。このアイコンは、選択すると [Collapse-Rows] (列の折りたたみ) アイコン () に変わり、子のプロパティキーのリストを折りたたむことができます。
3. ページ右側のセクションにある次のアクションを使用して、スキーマを変更します。

   - プロパティキーの名前を変更するには、プロパティキーの [Key] (キー) テキストボックスにカーソルを置き、新しい名前を入力します。
   - プロパティキーのデータ型を変更するには、リストを使用して、プロパティキーの新しいデータ型を選択します。
   - トップレベルの新しいプロパティキーをスキーマに追加するには、[Cancel] (キャンセル) ボタンの左側にある [Overflow] (オーバーフロー)

(⋯)
アイコンをクリックして、[Add root key] (ルートキーの追加) を選択します。

- スキーマに子のプロパティキーを追加するには、親キーに関連付けられている [Add-Key] (キーの追加)
(⮒)
アイコンをクリックします。子のキーの名前を入力し、データ型を選択します。
- スキーマからプロパティキーを削除するには、キー名の右端にある [Remove] (削除) アイコン
(🗑)
をクリックします。

4. カスタム変換コードで DynamicFrames を複数使用していない場合、他の出力スキーマを追加できます。

   - 新しい空のスキーマを追加するには、[Overflow] (オーバーフロー)
   (⋯)
   アイコンをクリックし、[Add output schema] (出力スキーマの追加) を選択します。
   - 既存のスキーマを新しい出力スキーマにコピーするには、コピーするスキーマがスキーマセレクタに表示されていることを確認します。[Overflow] (オーバーフロー)
   (⋯)
   アイコンをクリックして、[Duplicate] (複製) を選択します。

   出力スキーマを削除する場合は、コピーするスキーマがスキーマセレクタに表示されていることを確認します。[Overflow] (オーバーフロー)
   (⋯)
   アイコンをクリックして、[Delete] (削除)を選択します。

5. 新しいスキーマに新しいルートキーを追加するか、複製したキーを編集します。
6. 出力スキーマを変更する場合は、[Apply] (適用) ボタンをクリックして変更を保存し、スキーマエディタを終了します。

   変更を保存しない場合は、[Cancel] (キャンセル) ボタンをクリックします。

カスタム変換の出力を設定する

カスタムコード変換では、1 つの結果セットに DynamicFrame が 1 つだけであっても、DynamicFrames のコレクションが返されます。

カスタム変換ノードからの出力を処理するには

1. カスタム変換ノードを親ノードとして持つ SelectFromCollection 変換ノードを追加します。この変換を更新して、使用するデータセットを指定します。詳細については、「SelectFromCollection を使用して保持するデータセットを選択する」を参照してください。
2. カスタム変換ノードによって生成される他の `DynamicFrames` を使用する場合は、SelectFromCollection 変換をジョブ図に追加します。

   カスタム変換ノードを追加してフライトのデータセットを複数のデータセットに分割し、フライトの日付や番号などの各出力スキーマで識別プロパティキーを複製するシナリオを考えてみましょう。各出力スキーマに、カスタム変換ノードを親として持つ SelectFromCollection 変換ノードを追加します。
3. (オプション) その後、各 SelectFromCollection 変換ノードをジョブ内の他のノードの入力として、またはデータターゲットノードの親として使用できます。

カスタムビジュアル変換によるデータの変換

カスタムビジュアル変換を使用すると、変換を作成して AWS Glue Studio ジョブで使用できるようになります。ETL 開発者は、コーディングに慣れていない場合がありますが、カスタムビジュアル変換により、AWS Glue Studio インターフェイスを使用して増え続ける変換のライブラリを検索して使用することができます。

カスタムのビジュアル変換を作成し、Amazon S3 にアップロードして、AWS Glue Studio のビジュアルエディタでこれらのジョブを処理できるようにすることができます。

トピック

- カスタムビジュアル変換を開始する
- ステップ 1. JSON 設定ファイルを作成する
- ステップ 2. 変換ロジックを実装する
- ステップ 3. AWS Glue Studio でのカスタムビジュアル変換の検証とトラブルシューティング
- ステップ 4. 必要に応じてカスタムビジュアル変換を更新する
- Step 5. AWS Glue Studio でカスタムビジュアル変換を使用する
- 使用例
- カスタムビジュアルスクリプトの例
- 動画

カスタムビジュアル変換を開始する

カスタムビジュアル変換を作成するには、次の手順を実行します。

- ステップ 1. JSON 設定ファイルを作成する
- ステップ 2. 変換ロジックを実装する
- ステップ 3. カスタムビジュアル変換を検証する
- ステップ 4. 必要に応じてカスタムビジュアル変換を更新する
- Step 5. AWS Glue Studio でカスタムビジュアル変換を使用する

Amazon S3 バケットをセットアップして開始し、ステップ 1。JSON 設定ファイルを作成するに進みます。

前提条件

お客様提供の変換はお客様の AWS アカウント内にあります。そのアカウントは変換を所有しているため、変換を表示 (検索、使用)、編集、削除するすべてのアクセス許可を持っています。

AWS Glue Studio でカスタム変換を使用するには、その AWS アカウントの Amazon S3 アセットバケットに次の 2 つのファイルを作成してアップロードする必要があります。

- Python ファイル – 変換関数が含まれています
- JSON ファイル – 変換を記述します。これは、設定ファイルとも呼ばれ、変換の定義に必要です。

ファイルをペアリングするには、両方に同じ名前を使用します。例:

- myTransform.json
- myTransform.py

必要に応じて、アイコンを含む SVG ファイルを指定することで、カスタムビジュアル変換にカスタムアイコンを付加できます。ファイルをペアリングするには、次のように、アイコンに同じ名前を使用します。

- myTransform.svg

AWS Glue Studio では、それぞれのファイル名を使用して自動的に照合されます。ファイル名を既存のモジュールと同じにすることはできません。

推奨される変換ファイル名の命名規則

AWS Glue Studio では、ファイルはモジュール (例えば、`import myTransform`) としてジョブスクリプトにインポートされます。そのため、ファイル名は Python の変数名 (識別子) に設定されているのと同じ命名規則に従う必要があります。具体的には、文字またはアンダースコアで始まり、文字、数字、および/またはアンダースコアのみで構成されている必要があります。

> **Note**
>
> 予期しない実行時の問題を避けるため、変換ファイル名がロードされている既存の Python モジュール (`sys`, `array`, `copy` など) と競合していないことを確認してください。

Amazon S3 バケットのセットアップ

作成した変換は Amazon S3 に保存され、AWS アカウントによって所有されます。すべてのジョブスクリプトが現在保存されている Amazon S3 アセットフォルダ (例えば、`s3://aws-glue-assets-<accountid>-<region>/transforms`) にファイル (json と py) をアップロードするだけで、新しいカスタムビジュアル変換を作成できます。カスタムアイコンを使用している場合は、それもアップロードします。デフォルトでは、AWS Glue Studio は同じ S3 バケット内の /transforms フォルダからすべての .json ファイルを読み取ります。

ステップ 1. JSON 設定ファイルを作成する

カスタムビジュアル変換を定義して記述するには、JSON 設定ファイルが必要です。設定ファイルのスキーマは次の通りです。

JSON ファイル構造

フィールド

- `name: string` – (必須) 変換の識別に使用される変換システム名。Python の変数名 (識別子) に設定されているのと同じ命名規則に従ってください。具体的には、文字またはアンダースコアで始まり、文字、数字、および/またはアンダースコアのみで構成されている必要があります。
- `displayName: string` – (オプション) AWS Glue Studio ビジュアルジョブエディタに表示される変換の名前。`displayName` を指定しない場合は、AWS Glue Studio では `name` が変換の名前として使用されます。
- `description: string` – (オプション) 変換の説明が AWS Glue Studio に表示され、検索できます。

- `functionName: string` – (必須) Python 関数名が Python スクリプトで呼び出す関数を識別するために使用されます。
- `path: string` – (オプション) Python ソースファイルへの Amazon S3 のフルパス。指定しない場合は、AWS Glue では、ファイル名マッチングを使用して .json ファイルと .py ファイルがペアリングされます。例えば、JSON ファイルの名前が `myTransform.json` の場合、同じ Amazon S3 ロケーションにある `myTransform.py` という Python ファイルとペアになります。
- `parameters: Array of TransformParameter object` – (オプション) AWS Glue Studio ビジュアルエディタで設定するときに表示されるパラメータのリスト。

TransformParameter フィールド

- `name: string` – (必須) ジョブスクリプト内で名前付き引数として Python 関数に渡されるパラメータ名。Python の変数名 (識別子) に設定されているのと同じ命名規則に従ってください。具体的には、文字またはアンダースコアで始まり、文字、数字、および/またはアンダースコアのみで構成されている必要があります。
- `displayName: string` – (オプション) AWS Glue Studio ビジュアルジョブエディタに表示される変換の名前。`displayName` を指定しない場合は、AWS Glue Studio では `name` が変換の名前として使用されます。
- `type: string` – (必須) 一般的な Python データ型を受け入れるパラメータ型。有効な値: 'str' | 'int' | 'float' | 'list' | 'bool'。
- `isOptional: boolean` – (オプション) パラメータがオプションかどうかを決定します。デフォルトでは、すべてのパラメータが必須です。
- `description: string` – (オプション) ユーザーによる変換パラメータの設定に役立つ説明が AWS Glue Studio に表示されます。
- `validationType: string` – (オプション) このパラメータの検証方法を定義します。現在、正規表現のみをサポートしています。デフォルトで検証タイプは `RegularExpression` に設定されています。
- `validationRule: string` – (オプション) `validationType` が `RegularExpression` に設定されている場合、送信前にフォーム入力を検証するために使用される正規表現。正規表現の構文は RegExp Ecmascript の仕様に合致している必要があります。
- `validationMessage: string` – (オプション) 検証が失敗したときに表示するメッセージ。
- `listOptions: An array of TransformParameterListOption object` または `string`、もしくは「column」という文字列の値 — (オプション) 選択、または複数選択の UI コントロールに表示するためのオプション。カンマ区切り値 (CSV) のリスト、または強く型付けさ

れた TransformParameterListOption 型の JSON オブジェクトを受け入れます。文字列の値「column」を指定すると、親ノードのスキーマから、列のリストを動的に入力することもできます。

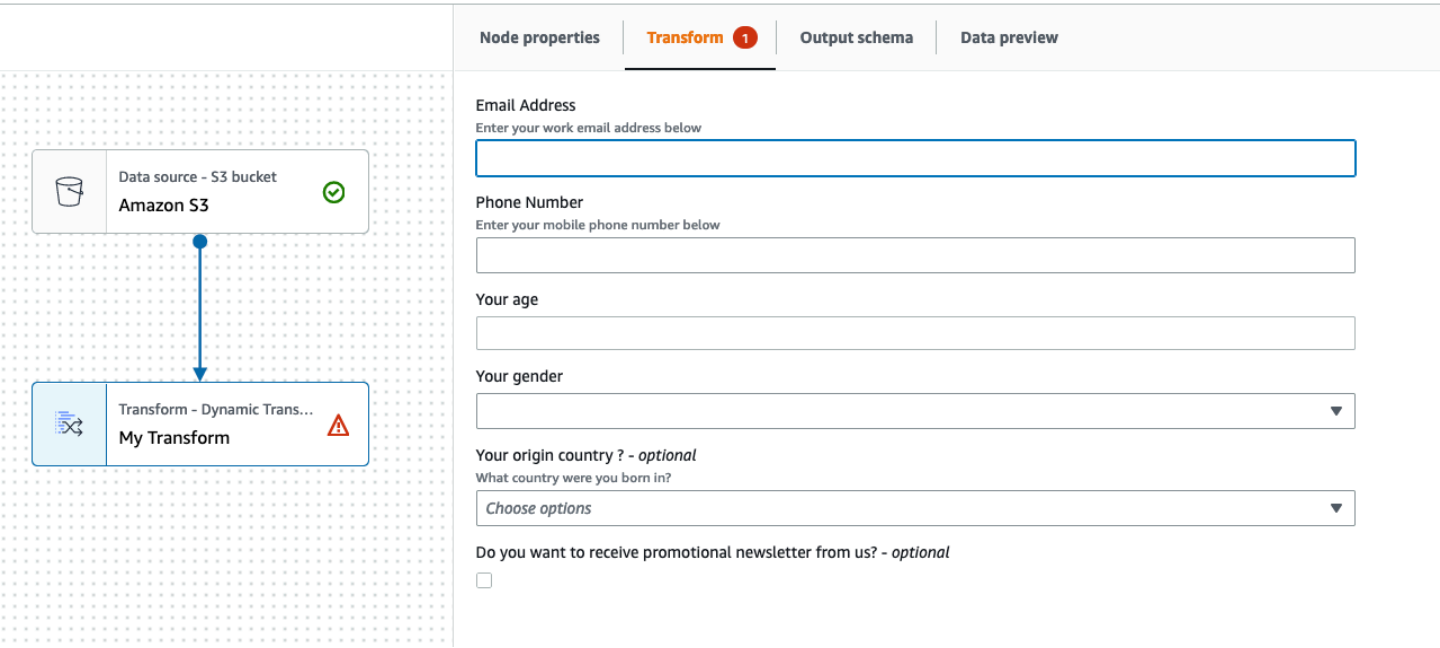
- `listType: string` – (オプション) type = 'list' のオプションタイプを定義します。有効な値: 'str' | 'int' | 'float' | 'list' | 'bool'。一般的な Python データ型を受け入れるパラメータ型。

TransformParameterListOption フィールド

- `value: string | int | float | bool` – (必須) オプション値。
- `label: string` – (オプション) 選択ドロップダウンに表示されるオプションラベル。

AWS Glue Studio での変換パラメータ

.json ファイルで `isOptional` と指定しない限り、デフォルトでパラメータは必須です。AWS Glue Studio では、パラメータが [Transform] (変換) タブに表示されます。この例は、E メールアドレス、電話番号、年齢、性別、出身国などのユーザー定義パラメータを示しています。

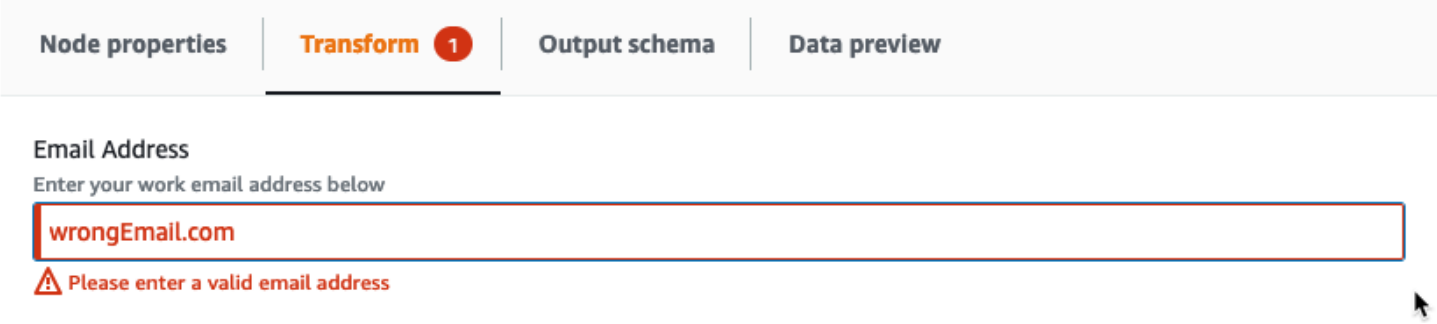


`validationRule` パラメータを指定し、`validationMessage` で検証メッセージを指定することで、JSON ファイルで正規表現を使用する際に AWS Glue Studio で一部の検証を必須にできます。

```
    "validationRule": "^\\(?(\\d{3})\\)?[- ]?(\\d{3})[- ]?(\\d{4})$",
    "validationMessage": "Please enter a valid US number"
```

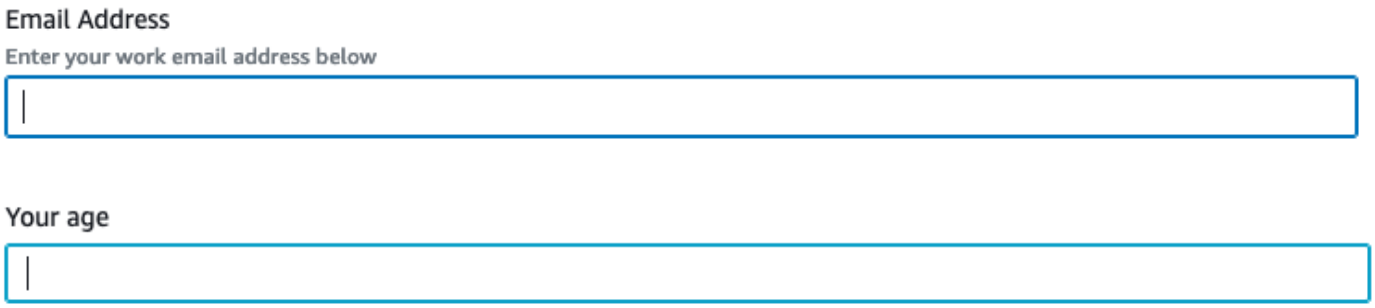
> **Note**
>
> 検証はブラウザで行われるため、正規表現の構文は RegExp Ecmascript の仕様に合致している必要があります。Python 構文は、これらの正規表現ではサポートされていません。

検証を追加することで、ユーザーが誤ったユーザー入力でジョブを保存するのを防ぐことができます。次の例のように AWS Glue Studio に検証メッセージが表示されます。

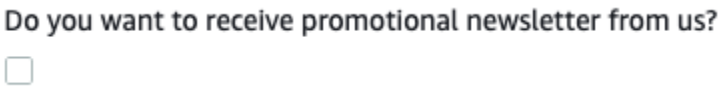


パラメータは、パラメータ設定に基づいて AWS Glue Studio に表示されます。

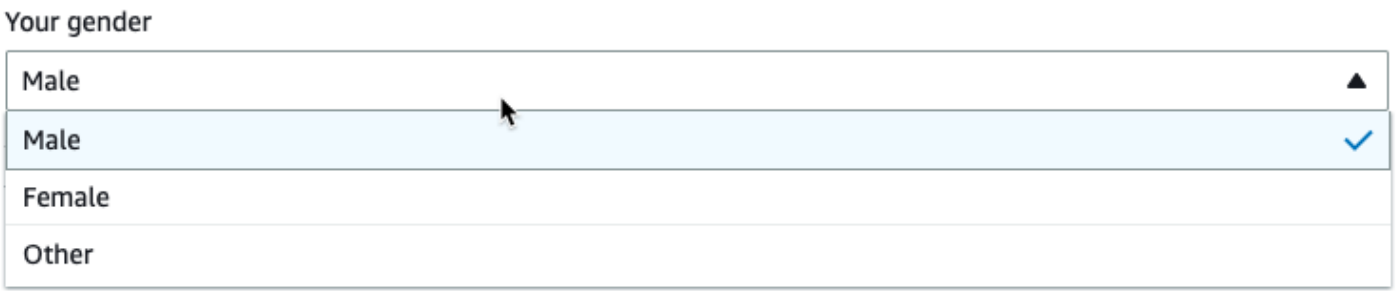
- `type` が `str`、`int`、または `float` の場合、テキスト入力フィールドが表示されます。例えば、スクリーンショットは「E メールアドレス」と「年齢」パラメータの入力フィールドを示しています。



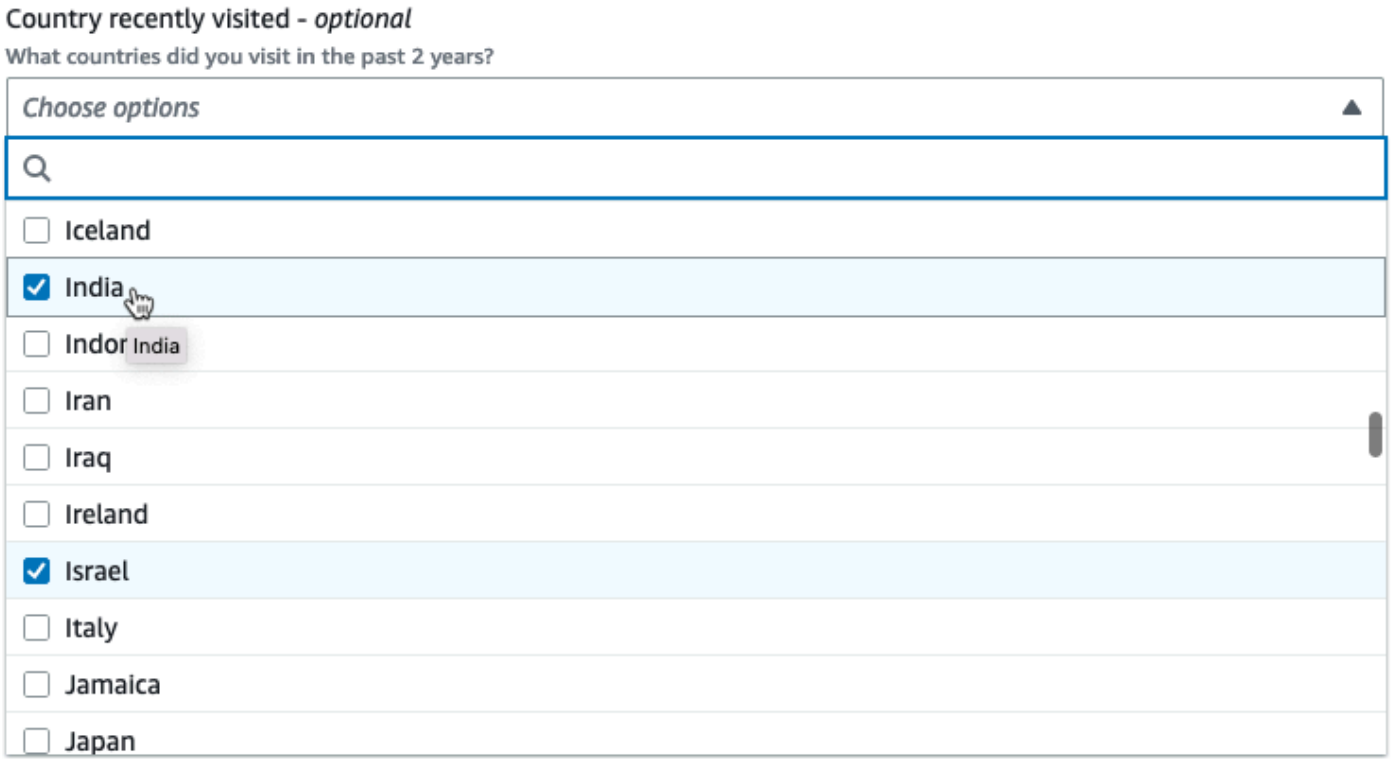
- `type` が `bool` の場合、チェックボックスが表示されます。



- `type` が `str` であり、`listOptions` が指定されていると、単一選択リストが表示されます。



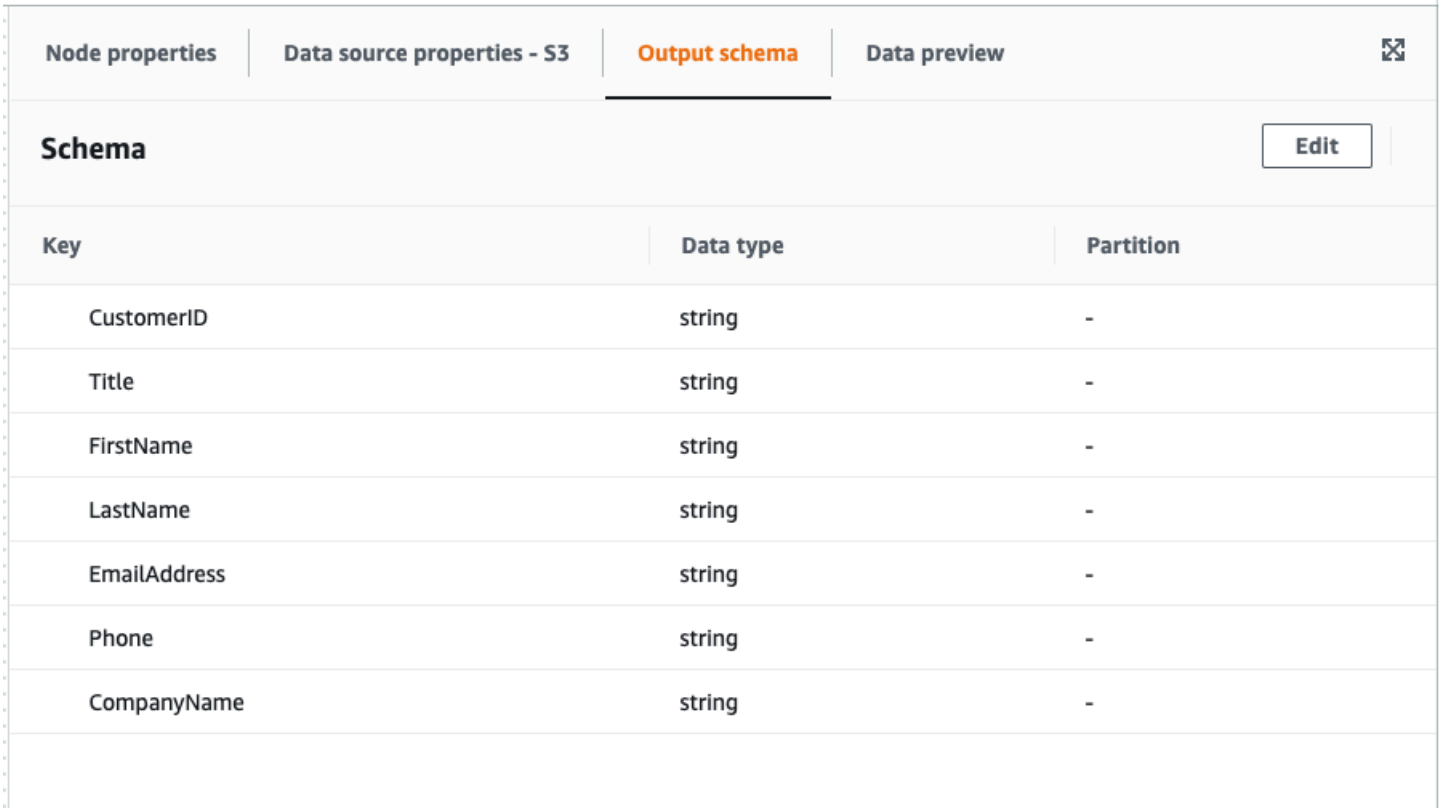
- `type` が `list` であり、`listOptions` と `listType` が指定されていると、複数選択リストが表示されます。



列のセレクタをパラメータとして表示

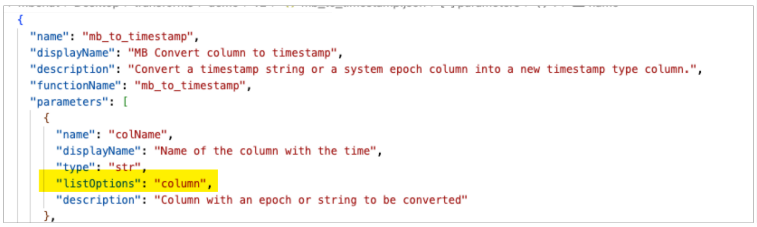
スキーマから列をユーザーが選択するように設定されている場合は、列セレクタを表示することで、ユーザーは列の名前を入力する必要がなくなります。`listOptions` フィールドを「column」に設定すると、親ノードの出力スキーマに基づいて、列セレクタを AWS Glue Studio が動的に表示します。AWS Glue Studio は単一の列、または複数列のセレクタを表示できます。

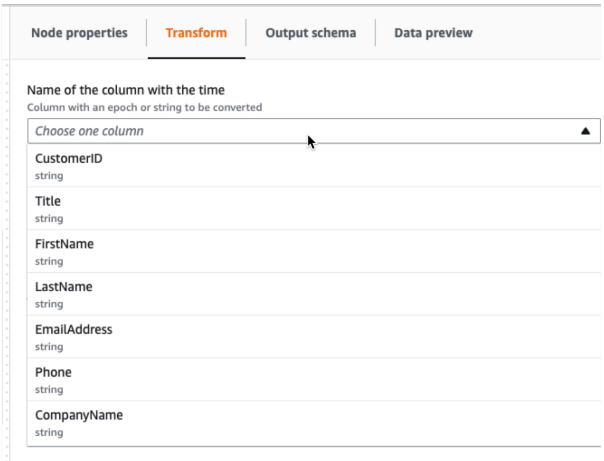
次の例でスキーマを使用しています。



単一の列を表示するように、Custom Visual Transform パラメータを定義するには

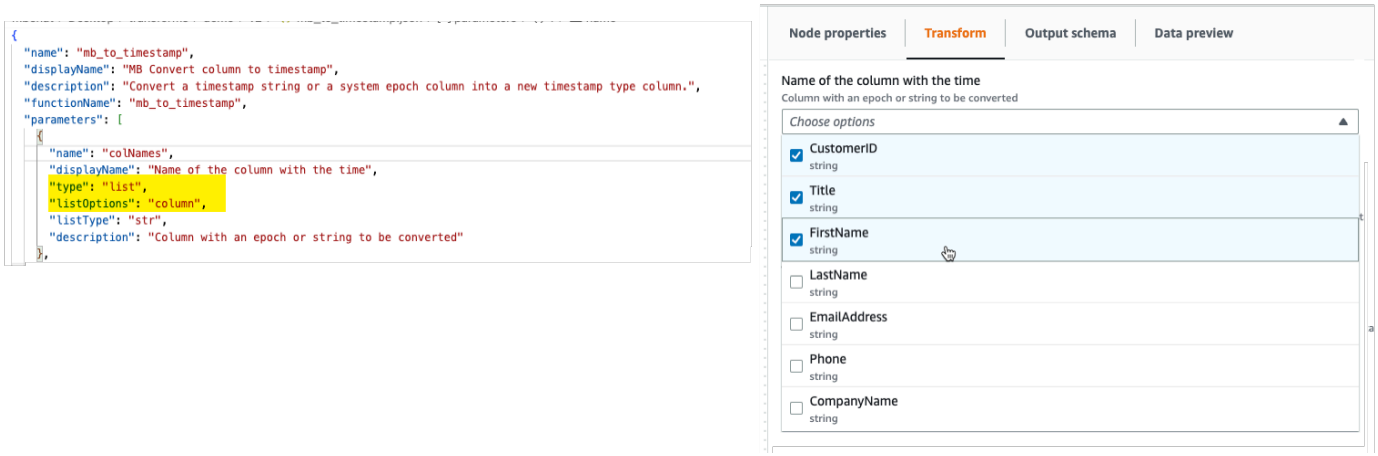
1. JSON ファイルで、`parameters` オブジェクトの `listOptions` の値を、「column」に設定します。これにより、ユーザーが AWS Glue Studio の選択リストから列を選択できるようになります。





2. 次のようにパラメータを定義すると、複数の列を選択することもできます。

   - `listOptions: "column"`
   - `type: "list"`



ステップ 2. 変換ロジックを実装する

> **Note**
>
> カスタムビジュアル変換は Python スクリプトのみをサポートしています。Scala はサポートされていません。

.json 設定ファイルで定義された関数を実装するコードを追加するには、Python ファイルを .json ファイルと同じ場所に同じ名前で「.py」拡張子を付けて配置することをお勧めします。AWS Glue Studio では、.json ファイルと .py ファイルが自動的にペアリングされるため、設定ファイルで Python ファイルのパスを指定する必要はありません。

Python ファイルに、指定されたパラメータを設定した宣言済み関数を追加し、その関数を DynamicFrame で使用するために登録します。Python ファイルの例を次に示します。

```
from awsglue import DynamicFrame

# self refers to the DynamicFrame to transform,
# the parameter names must match the ones defined in the config
# if it's optional, need to provide a default value
def myTransform(self, email, phone, age=None, gender="",
                      country="", promotion=False):
   resulting_dynf = # do some transformation on self
   return resulting_dynf

DynamicFrame.myTransform = myTransform
```

Python コードの開発とテストを最も早く行うために、AWS Glue ノートブックの使用をお勧めします。「AWS Glue Studio 中でのノートブックの使用開始」を参照してください。

変換ロジックの実装方法を説明するために、以下の例でのカスタムビジュアル変換は、入力データをフィルタリングして特定の米国の州に関連するデータのみを保持する変換になっています。.json ファイルには functionName のパラメータが custom_filter_state として含まれ、2 つの引数 (「str」型の「state」と「colName」) が含まれています。

.json 設定ファイルの例は次のとおりです。

```
{
"name": "custom_filter_state",
"displayName": "Filter State",
"description": "A simple example to filter the data to keep only the state indicated.",
"functionName": "custom_filter_state",
"parameters": [
   {
    "name": "colName",
    "displayName": "Column name",
    "type": "str",
    "description": "Name of the column in the data that holds the state postal code"
   },
   {
    "name": "state",
    "displayName": "State postal code",
    "type": "str",
    "description": "The postal code of the state whole rows to keep"
   }
  ]
}
```

Python でコンパニオンスクリプトを実装するには

1. AWS Glue ノートブックを起動し、セッションを開始するために提供された最初のセルを実行します。最初のセルを実行すると、必要な基本コンポーネントが作成されます。
2. 例で説明されているようにフィルタリングを実行する関数を作成し、DynamicFrame に登録します。以下のコードをコピーして、AWS Glue ノートブックのセルに貼り付けます。

```
from awsglue import DynamicFrame

def custom_filter_state(self, colName, state):
    return self.filter(lambda row: row[colName] == state)

DynamicFrame.custom_filter_state = custom_filter_state
```

3. サンプルデータを作成またはロードして、同じセルまたは新しいセルでコードをテストします。サンプルデータを新しいセルに追加する場合は、セルを実行することを忘れないでください。例:

```
# A few of rows of sample data to test
data_sample = [
    {"state": "CA", "count": 4},
    {"state": "NY", "count": 2},
    {"state": "WA", "count": 3}
]
df1 = glueContext.sparkSession.sparkContext.parallelize(data_sample).toDF()
dynf1 = DynamicFrame.fromDF(df1, glueContext, None)
```

4. さまざまな引数で次のようにテストして、「custom_filter_state」を検証します。

```
[14]: dynf1.custom_filter_state("state", "NY").show()
      {"count": 2, "state": "NY"}
```

5. いくつかのテストを実行した後、.py 拡張子を付けてコードを保存し、.py ファイルに .json ファイル名と同じ名前を付けます。.py ファイルと .json ファイルは同じ変換フォルダにある必要があります。

   次のコードをコピーしてファイルに貼り付け、.py ファイル拡張子に変更します。

```
from awsglue import DynamicFrame

def custom_filter_state(self, colName, state):
    return self.filter(lambda row: row[colName] == state)

DynamicFrame.custom_filter_state = custom_filter_state
```

6. AWS Glue Studio でビジュアルジョブを開き、使用可能な [Transforms] (変換) のリストから変換を選択して変換をジョブに追加します。

   この変換を Python スクリプトコードで再利用するには、Amazon S3 パスを [Referenced files path] (参照されるファイルパス) の下のジョブの .py ファイルに追加し、スクリプト内の python ファイルの名前 (拡張子なし) をファイルの先頭に追加してインポートします。例えば、`import <ファイル名 (拡張子なし)>`

ステップ 3. AWS Glue Studio でのカスタムビジュアル変換の検証とトラブルシューティング

カスタムビジュアル変換が AWS Glue Studio にロードされる前に AWS Glue Studio で JSON 設定ファイルが検証されます。検証には次の項目が含まれます。

- 必須フィールドの有無
- JSON 形式の検証
- パラメータの誤りまたは無効
- 同じ Amazon S3 パスに .py ファイルと .json ファイルの両方が存在すること
- .py ファイル名と .json のファイル名の一致

検証が成功すると、変換がビジュアルエディタの利用可能な [Actions] (アクション) のリストに表示されます。カスタムアイコンが指定されている場合は、[アクション] の横に表示されているはずです。

検証に失敗した場合、AWS Glue Studio では、カスタムビジュアル変換はロードされません。

ステップ 4. 必要に応じてカスタムビジュアル変換を更新する

一度作成して使用すると、次のように変換が対応する JSON 定義に従っている限り、変換スクリプトを更新できます。

- DynamicFrame に割り当てるときに使用される名前は、JSON の `functionName` と一致している必要があります。
- 関数の引数は、ステップ 1. JSON 設定ファイルを作成する で説明されているように JSON ファイルで定義する必要があります。

- Python ファイルの Amazon S3 パスは、ジョブが直接それに依存しているため、変更できません。

> **Note**
>
> 更新が必要な場合は、スクリプトと .json ファイルの更新を整合させ、ビジュアルジョブを新しい変換で正しく再保存します。更新後にビジュアルジョブを保存しないと、更新の適用と検証が行われません。Python スクリプトファイルの名前が変更されたり、.json ファイルの横に配置されなかったりする場合は、.json ファイルにフルパスを指定する必要があります。

カスタムアイコン

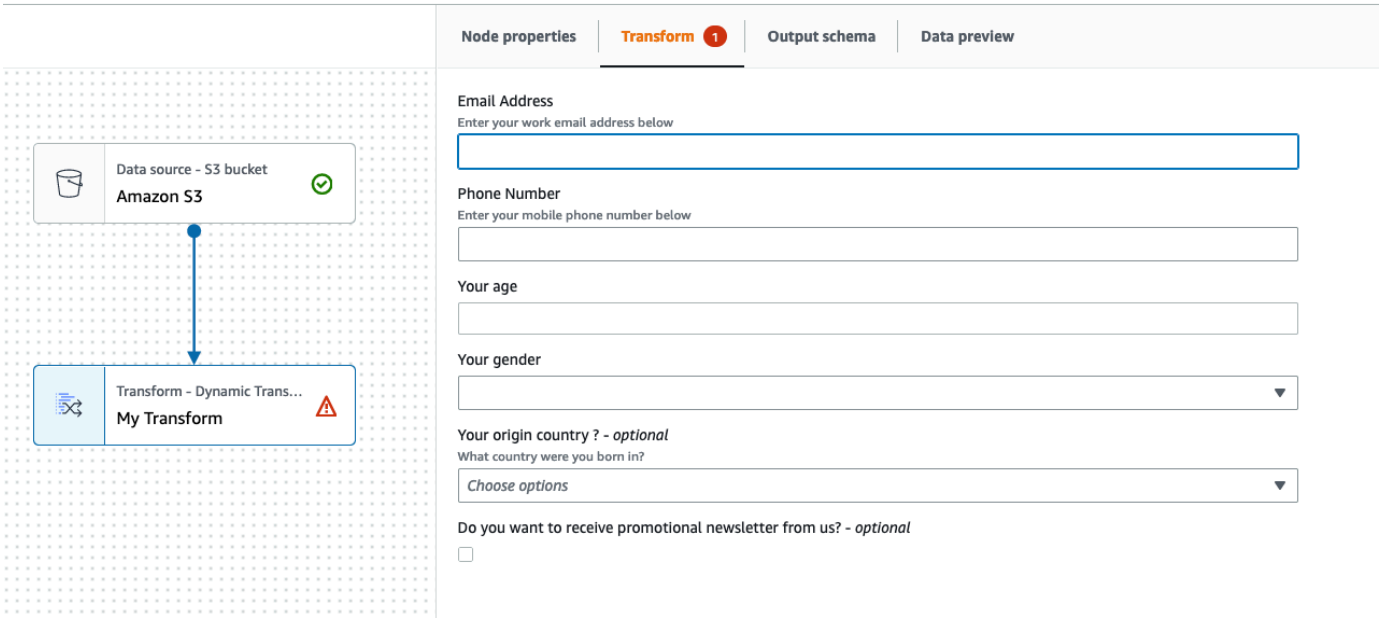
[アクション] のデフォルトアイコンではワークフローの一部として視覚的に区別できないと判断した場合は、「the section called “ カスタムビジュアル変換を開始する ”」で説明されているようにカスタムアイコンを指定できます。Amazon S3 でホストされている対応する SVG を更新することで、アイコンを更新できます。

最良の結果を得るには、Cloudscape Design System のガイドラインに従って、32 x 32 ピクセルで表示されるように画像をデザインしてください。Cloudscape ガイドラインの詳細については、Cloudscape のドキュメントを参照してください。

Step 5. AWS Glue Studio でカスタムビジュアル変換を使用する

AWS Glue Studio でカスタムビジュアル変換を使用するには、設定ファイルとソースファイルをアップロードし、[Action] (アクション) メニューから変換を選択します。値や入力が必要なパラメータはどれも [Transform] (変換) タブで利用できます。

1. 2 つのファイル (Python ソースファイルと JSON 設定ファイル) を、ジョブスクリプトが保存されている Amazon S3 アセットフォルダにアップロードします。デフォルトでは、AWS Glue は同じ Amazon S3 バケット内の /transforms フォルダからすべての JSON ファイルを取得します。
2. [Action] (アクション) メニューから、カスタムビジュアル変換を選択します。これの名前は、変換の `displayName` または .json 設定ファイルで指定した名前になります。
3. 設定ファイルに設定されたパラメータの値を入力します。



使用例

.json 設定ファイルに含めることが可能なすべてのパラメータの例を次に示します。

```
{
  "name": "MyTransform",
  "displayName": "My Transform",
  "description": "This transform description will be displayed in UI",
  "functionName": "myTransform",
  "parameters": [
      {
      "name": "email",
      "displayName": "Email Address",
      "type": "str",
      "description": "Enter your work email address below",
      "validationType": "RegularExpression",
      "validationRule": "^\\w+([\\.-]?\\w+)*@\\w+([\\.-]?\\w+)*(\\.\\w{2,3})+$",
      "validationMessage": "Please enter a valid email address"
    },
    {
      "name": "phone",
      "displayName": "Phone Number",
      "type": "str",
      "description": "Enter your mobile phone number below",
      "validationRule": "^\\(?(\\d{3})\\)?[- ]?(\\d{3})[- ]?(\\d{4})$",
      "validationMessage": "Please enter a valid US number"
```

```
    },
    {
      "name": "age",
      "displayName": "Your age",
      "type": "int",
      "isOptional": true
    },
    {
      "name": "gender",
      "displayName": "Your gender",
      "type": "str",
      "listOptions": [
            {"label": "Male", "value": "male"},
            {"label": "Female", "value": "female"},
            {"label": "Other", "value": "other"}
        ],
      "isOptional": true
    },
    {
      "name": "country",
      "displayName": "Your origin country ?",
      "type": "list",
      "listOptions": "Afghanistan,Albania,Algeria,American
 Samoa,Andorra,Angola,Anguilla,Antarctica,Antigua and
 Barbuda,Argentina,Armenia,Aruba,Australia,Austria,Azerbaijan,Bahamas,Bahrain,Bangladesh,Barbad
 and Herzegovina,Botswana,Bouvet Island,Brazil,British Indian Ocean Territory,Brunei
 Darussalam,Bulgaria,Burkina Faso,Burundi,Cambodia,Cameroon,Canada,Cape
 Verde,Cayman Islands,Central African Republic,Chad,Chile,China,Christmas
 Island,Cocos (Keeling Islands),Colombia,Comoros,Congo,Cook Islands,Costa
 Rica,Cote D'Ivoire (Ivory Coast),Croatia (Hrvatska,Cuba,Cyprus,Czech
 Republic,Denmark,Djibouti,Dominica,Dominican Republic,East Timor,Ecuador,Egypt,El
 Salvador,Equatorial Guinea,Eritrea,Estonia,Ethiopia,Falkland Islands (Malvinas),Faroe
 Islands,Fiji,Finland,France,France,Metropolitan,French Guiana,French Polynesia,French
 Southern
 Territories,Gabon,Gambia,Georgia,Germany,Ghana,Gibraltar,Greece,Greenland,Grenada,Guadeloupe,G
Bissau,Guyana,Haiti,Heard and McDonald Islands,Honduras,Hong
 Kong,Hungary,Iceland,India,Indonesia,Iran,Iraq,Ireland,Israel,Italy,Jamaica,Japan,Jordan,Kazak
 (North),Korea
 (South),Kuwait,Kyrgyzstan,Laos,Latvia,Lebanon,Lesotho,Liberia,Libya,Liechtenstein,Lithuania,Lu
 Islands,Martinique,Mauritania,Mauritius,Mayotte,Mexico,Micronesia,Moldova,Monaco,Mongolia,Mont
 Antilles,New Caledonia,New Zealand,Nicaragua,Niger,Nigeria,Niue,Norfolk
 Island,Northern Mariana Islands,Norway,Oman,Pakistan,Palau,Panama,Papua
 New Guinea,Paraguay,Peru,Philippines,Pitcairn,Poland,Portugal,Puerto
 Rico,Qatar,Reunion,Romania,Russian Federation,Rwanda,Saint Kitts and Nevis,Saint
```

```
 Lucia,Saint Vincent and The Grenadines,Samoa,San Marino,Sao Tome and Principe,Saudi
 Arabia,Senegal,Seychelles,Sierra Leone,Singapore,Slovak Republic,Slovenia,Solomon
 Islands,Somalia,South Africa,S. Georgia and S. Sandwich Isls.,Spain,Sri
 Lanka,St. Helena,St. Pierre and Miquelon,Sudan,Suriname,Svalbard and Jan Mayen
 Islands,Swaziland,Sweden,Switzerland,Syria,Tajikistan,Tanzania,Thailand,Togo,Tokelau,Tonga,Tri
 and Tobago,Tunisia,Turkey,Turkmenistan,Turks and Caicos
 Islands,Tuvalu,Uganda,Ukraine,United Arab Emirates,United Kingdom
 (Britain / UK),United States of America (USA),US Minor Outlying
 Islands,Uruguay,Uzbekistan,Vanuatu,Vatican City State (Holy See),Venezuela,Viet
 Nam,Virgin Islands (British),Virgin Islands (US),Wallis and Futuna Islands,Western
 Sahara,Yemen,Yugoslavia,Zaire,Zambia,Zimbabwe",
      "description": "What country were you born in?",
      "listType": "str",
      "isOptional": true
    },
    {
      "name": "promotion",
      "displayName": "Do you want to receive promotional newsletter from us?",
      "type": "bool",
      "isOptional": true
    }
  ]
}
```

カスタムビジュアルスクリプトの例

以下の例で行われる変換はどれも等価です。ただし、2 番目の例 (SparkSQL) が最もクリーンで効率的です。その次が Pandas UDF で、最後は最初の例の低レベルのマッピングです。次の例は、2 つの列を合計する単純な変換の完全な例です。

```
from awsglue import DynamicFrame

# You can have other auxiliary variables, functions or classes on this file, it won't
 affect the runtime
def record_sum(rec, col1, col2, resultCol):
    rec[resultCol] = rec[col1] + rec[col2]
    return rec


# The number and name of arguments must match the definition on json config file
# (expect self which is the current DynamicFrame to transform
# If an argument is optional, you need to define a default value here
```

```
#  (resultCol in this example is an optional argument)
def custom_add_columns(self, col1, col2, resultCol="result"):
    # The mapping will alter the columns order, which could be important
    fields = [field.name for field in self.schema()]
    if resultCol not in fields:
        # If it's a new column put it at the end
        fields.append(resultCol)
    return self.map(lambda record: record_sum(record, col1, col2,
 resultCol)).select_fields(paths=fields)


# The name we assign on DynamicFrame must match the configured "functionName"
DynamicFrame.custom_add_columns = custom_add_columns
```

次の例は、SparkSQL API を利用した同等の変換です。

```
from awsglue import DynamicFrame

# The number and name of arguments must match the definition on json config file
# (expect self which is the current DynamicFrame to transform
# If an argument is optional, you need to define a default value here
#  (resultCol in this example is an optional argument)
def custom_add_columns(self, col1, col2, resultCol="result"):
    df = self.toDF()
    return DynamicFrame.fromDF(
        df.withColumn(resultCol, df[col1] + df[col2]) # This is the conversion logic
        , self.glue_ctx, self.name)


# The name we assign on DynamicFrame must match the configured "functionName"
DynamicFrame.custom_add_columns = custom_add_columns
```

次の例では同じ変換を使用していますが、Pandas UDF を使用しているため、プレーン UDF を使用するよりも効率的です。Pandas UDF を書く方法の詳細については、Apache Spark SQL のドキュメントを参照してください。

```
from awsglue import DynamicFrame
import pandas as pd
from pyspark.sql.functions import pandas_udf

# The number and name of arguments must match the definition on json config file
```

```
# (expect self which is the current DynamicFrame to transform
# If an argument is optional, you need to define a default value here
#  (resultCol in this example is an optional argument)
def custom_add_columns(self, col1, col2, resultCol="result"):
    @pandas_udf("integer")  # We need to declare the type of the result column
    def add_columns(value1: pd.Series, value2: pd.Series) # pd.Series:
        return value1 + value2

    df = self.toDF()
    return DynamicFrame.fromDF(
        df.withColumn(resultCol, add_columns(col1, col2)) # This is the conversion
 logic
        , self.glue_ctx, self.name)

# The name we assign on DynamicFrame must match the configured "functionName"
DynamicFrame.custom_add_columns = custom_add_columns
```

動画

次の動画では、ビジュアルカスタム変換の概要と使用方法を紹介しています。

データレイクフレームワークを AWS Glue Studio で使用する

概要

オープンソースのデータレイクフレームワークは、Amazon S3 上に構築されたデータレイクに保存されたファイルのインクリメンタルデータ処理を簡素化します。AWS Glue 3.0 以降では、次のオープンソースのデータレイクストレージフレームワークをサポートしています。

- Apache Hudi
- Linux Foundation Delta Lake
- Apache Iceberg

AWS Glue 4.0 では、AWS Glue はこれらのフレームワークをネイティブにサポートしているため、トランザクションが一貫した方法で、Amazon S3 に保存したデータを読み書きできます。AWS Glue ジョブでこれらのフレームワークを使用する場合でも、別のコネクタをインストールしたり、設定手順を追加で実行したりする必要はありません。

データレイクフレームワークは、Spark Script Editor ジョブから AWS Glue Studio 内のソースまたはターゲットとして使用できます。Apache Hudi、Apache Iceberg、Delta Lake の使用方法の詳細に

ついては、「AWS Glue ETL ジョブでデータレイクフレームワークを使用する」を参照してください。

AWS Glue ストリーミングソースからのオープンテーブルフォーマットの作成

AWS Glue ストリーミング ETL ジョブは、ストリーミングソースからのデータを継続的に消費し、転送中のデータをクリーンアップおよび変換して、数秒で分析できるようにします。

AWS は、お客様のニーズをサポートする幅広いサービスを提供します。AWS Database Migration Service などのデータベースレプリケーションサービスは、ソースシステムから Amazon S3 にデータをレプリケートできます。Amazon S3 は通常、データレイクのストレージレイヤーをホストします。オンラインソースアプリケーションをサポートするリレーショナルデータベース管理システム (RDBMS) に更新を適用するのは簡単ですが、この CDC プロセスをデータレイクに適用するのは困難です。オープンソースのデータ管理フレームワークは、増分データ処理とデータパイプライン開発を簡素化し、この問題を解決するための優れたオプションです。

詳細については、以下を参照してください。

- AWS Glue ストリーミングを使用して、Apache Hudi ベースのほぼリアルタイムのトランザクションデータレイクを作成する
- Build a real-time GDPR-aligned Apache Iceberg data lake

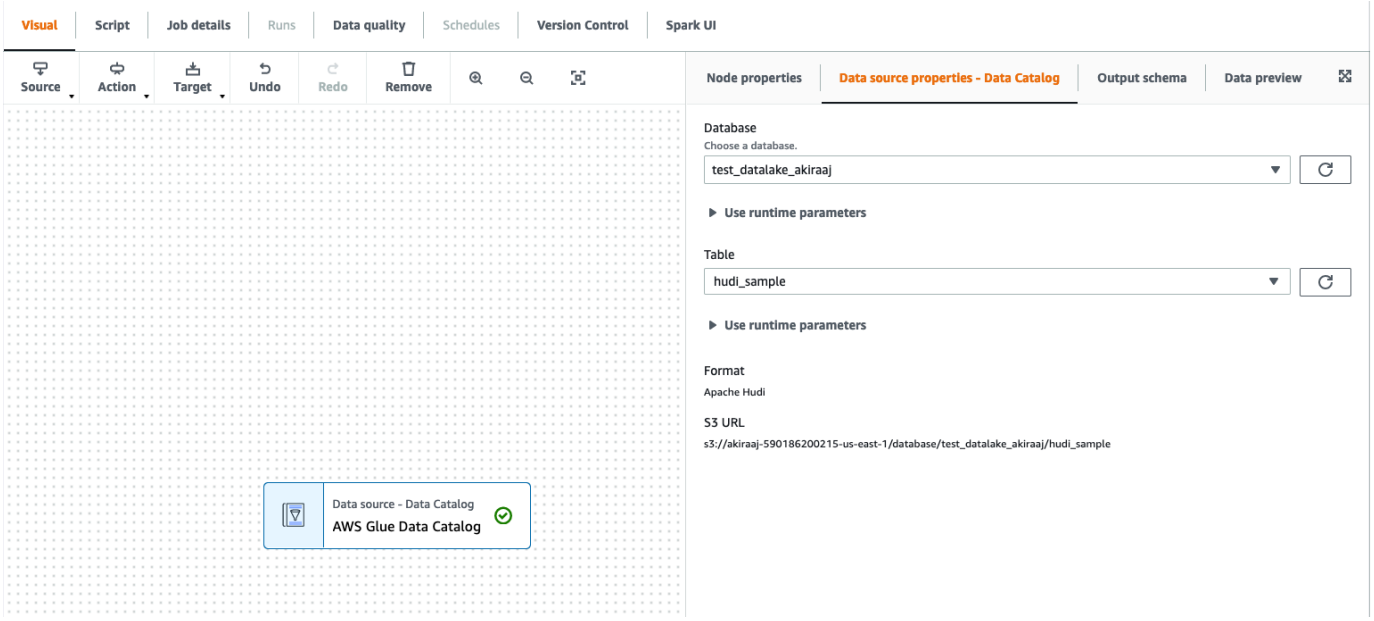
AWS Glue Studio Glue で Hudi フレームワークを使用する

ジョブを作成または編集すると、AWS Glue Studio では、使用している AWS Glue のバージョンに応じて対応する Hudi ライブラリが自動的に追加されます。詳細は、「AWS Glue での Hudi フレームワークの使用」を参照してください。

データカタログのデータソースで Apache Hudi フレームワークを使用する

Hudi データソースフォーマットをジョブに追加するには:

1. [ソース] メニューで、[AWS Glue Studio データカタログ] を選択します。
2. [Data source properties] (データソースのプロパティ) タブで、データベースとテーブルを選択します。
3. AWS Glue Studio に、フォーマットタイプとして Apache Hudi と Amazon S3 URL が表示されます。



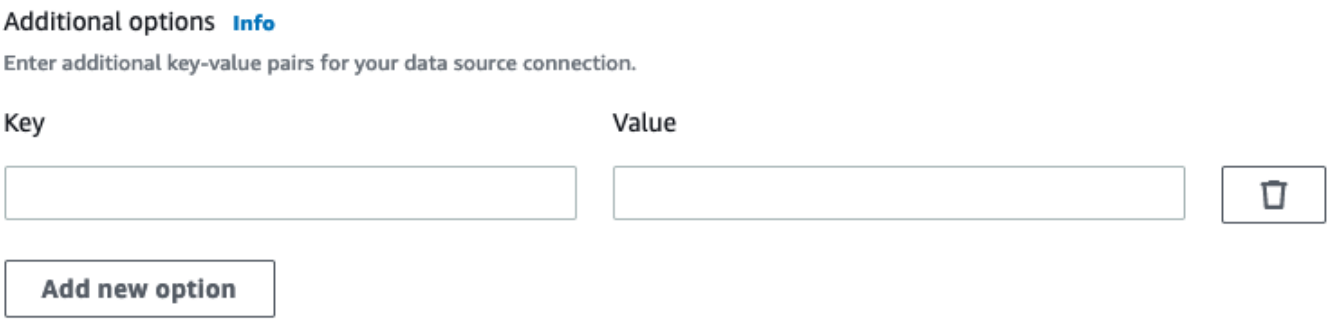
Amazon S3 データソースで Hudi フレームワークを使用する

1. [Source] (ソース) メニューで、[Amazon S3] を選択します。
2. Amazon S3 ソースタイプとしてデータカタログテーブルを選択する場合は、データベースとテーブルを選択します。
3. AWS Glue Studio に、Apache Hudi と Amazon S3 URL がフォーマットとして表示されます。
4. Amazon S3 ソースタイプとして Amazon S3 の場所を選択する場合は、[Browse Amazon S3] (Amazon S3 を参照) をクリックして Amazon S3 URL を選択します。
5. [Data format] (データフォーマット) で、[Apache Hudi] を選択します。

> **Note**
>
> AWS Glue Studio が、選択した Amazon S3 フォルダまたはファイルからスキーマを推測できない場合は、[Additional options] (追加オプション) を選択し、新しいフォルダまたはファイルを選択します。
> [Additional options] (追加オプション) の [Schema inference] (スキーマ推論) から、次のオプションを選択します。
>
> - [AWS Glue Studio にサンプルファイルを自動選択させる] — スキーマを推測できるよう、AWS Glue Studio が Amazon S3 の場所にあるサンプルファイルを選択します。自動選択されたファイルは [Auto-sampled file] (自動サンプル化ファイル) フィールドで確認できます。

- [Choose a sample file from Amazon S3] (Amazon S3 からサンプルファイルを選択) - [Browse Amazon S3] (Amazon S3 を参照) をクリックし、使用する使用する Amazon S3 ファイルを選択します。

6. [Infer schema] (スキーマを推測) をクリックします。続いて、[Output schema] (出力スキーマ) タブをクリックすると、出力スキーマを確認できます。
7. [Additional options] (追加オプション) をクリックし、キー値ペアを入力します。



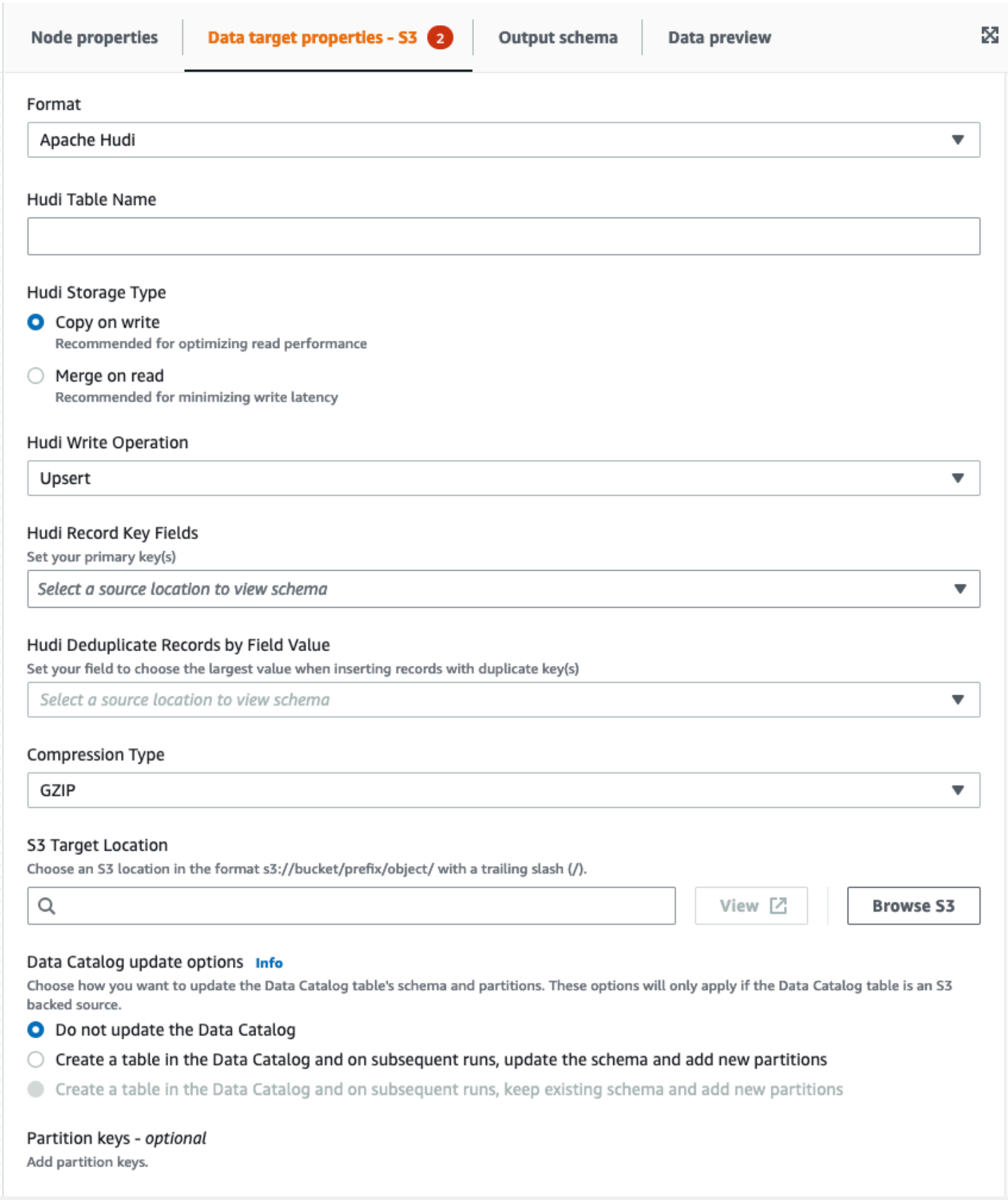
データターゲットで Apache Hudi フレームワークを使用する

データカタログのデータターゲットで Apache Hudi フレームワークを使用する

1. [ターゲット] メニューで [AWS Glue Studio データカタログ] を選択します。
2. [Data source properties] (データソースのプロパティ) タブで、データベースとテーブルを選択します。
3. AWS Glue Studio に、フォーマットタイプとして Apache Hudi と Amazon S3 URL が表示されます。

Amazon S3 データターゲットで Apache Hudi フレームワークを使用する

値を入力するか、使用可能なオプションの中から選択し、Apache Hudi 形式を設定します。Apache Hudi の詳細については、「Apache Hudi のドキュメント」を参照してください。


Node properties
Data target properties - S3 2
Output schema
Data preview
Format
Apache Hudi
Hudi Table Name
Hudi Storage Type
Copy on write
Recommended for optimizing read performance
Merge on read
Recommended for minimizing write latency
Hudi Write Operation
Upsert
Hudi Record Key Fields
Set your primary key(s)
Select a source location to view schema
Hudi Deduplicate Records by Field Value
Set your field to choose the largest value when inserting records with duplicate key(s)
Select a source location to view schema
Compression Type
GZIP
S3 Target Location
Choose an S3 location in the format s3://bucket/prefix/object/ with a trailing slash (/).
View
Browse S3
Data Catalog update options Info
Choose how you want to update the Data Catalog table's schema and partitions. These options will only apply if the Data Catalog table is an S3 backed source.
Do not update the Data Catalog
Create a table in the Data Catalog and on subsequent runs, update the schema and add new partitions
Create a table in the Data Catalog and on subsequent runs, keep existing schema and add new partitions
Partition keys - optional
Add partition keys.

- [Hudi Table Name] (Hudi テーブル名) — お使いの Hudi Table の名前です。
- [Hudi Storage Type] (Hudi ストレージタイプ) — 次の 2 つのうちいずれかを選択します。
  - [Copy on write] (書き込み時にコピー) — 読み取りパフォーマンスの最適化のため、推奨されています。こちらはデフォルトの Hudi ストレージタイプです。書き込み中、更新するたびに新しいバージョンのファイルが作成されます。
  - [Merge on read] (読み取り時に結合) — 書き込みレイテンシーの最小化のため、推奨されています。更新は、行形式の差分ファイルに記録され、必要に応じて圧縮されて、新しいバージョンの列形式のファイルが作成されます。
- [Hudi Write Operation] (Hudi 書き込みオペレーション) - 次の中から選択します。
  - [Upsert] (アップサート) — デフォルトのオペレーションで、入力レコードはインデックスを検索して挿入または更新として最初にタグ付けされます。既存データを更新する場合に推奨されています。
  - [Insert] (インサート) — レコードが挿入されます。ただし、既存のレコードはチェックされないため重複する可能性があります。
  - [Bulk Insert] (一括インサート) — レコードを挿入します。データの量が多い場合に推奨されています。
- [Hudi Record Key Fields] (Hudi レコードキーフィールド) — 検索バーを使ってプライマリレコードキーを検索し選択します。Hudi のレコードは、レコードキーとレコードが属しているパーティションパスのペアから成るプライマリキーで識別されます。
- [Hudi Precombine Field] (Hudi 事前結合フィールド) — 実際に書き込む前の事前結合で使用されるフィールドです。2 つのレコードのキー値が同じである場合、AWS Glue Studio は、事前結合フィールドの値が大きい方のレコードを選択します。増分値 (例: updated_at) が属するフィールドを設定します。
- [Compression Type] (圧縮タイプ) — 圧縮タイプ (非圧縮、GZIP、LZO、Snappy) の中から 1 つ選択します。
- [Amazon S3 Target Location] (Amazon S3 ターゲットの場所) — [Browse S3] (S3 を参照) をクリックして Amazon S3 ターゲットの場所を選択します。
- [Data Catalog update options] (データカタログ更新オプション) — 次の中から選択します。
  - Do not update the Data Catalog(データカタログを更新しない): (デフォルト) スキーマが変更されたり、新しいパーティションが追加された場合、ジョブでデータカタログを更新したくない場合は、このオプションを選択します。
  - [Create a table in the Data Catalog and on subsequent runs, update the schema and add new partitions] (データカタログにテーブルを作成し、それ以降の実行時にスキーマを更新して新しいパーティションを追加する): このオプションを選択すると、最初のジョブの実行時に、ジョブに

よりデータカタログにテーブルが作成されます。それ以降のジョブの実行時にスキーマが変更されたり、新しいパーティションが追加されたりすると、ジョブによりデータカタログテーブルが更新されます。

また、データカタログからデータベースを選択し、テーブル名を入力する必要があります。

- [Create a table in the Data Catalog and on subsequent runs, keep existing schema and add new partitions] (データカタログでテーブル作成し、それ以降の実行時に既存のスキーマの保持して新しいパーティションを追加する): このオプションを選択すると、最初のジョブの実行時に、ジョブによりデータカタログにテーブルが作成されます。それ以降のジョブの実行時に、新しいパーティションを追加するため、ジョブでデータカタログテーブルが更新されます。

  また、データカタログからデータベースを選択し、テーブル名を入力する必要があります。

- Partition keys (パーティションキー): 出力でパーティションキーとして使用する列を選択します。さらにパーティションキーを追加するには、[Add a partition key] (パーティションキーの追加) を選択します。
- [Addtional options] (追加オプション) — 必要に応じてキーと値のペアを入力します。

AWS Glue Studio によるコード生成

ジョブを保存すると、Hudi のソースまたはターゲットが検出された場合に、次のジョブパラメータがジョブに追加されます。

- `--datalake-formats` — ビジュアルジョブで、([Format] (形式) を選択することで、直接的またはデータレイクでバックアップされたカタログテーブルを選択することで間接的に) 検出されたデータレイク形式の個別リスト。
- `--conf` — `--datalake-formats` の値に基づいて生成される。例えば、`--datalake-formats` の値が「hudi」である場合、AWS Glue はこのパラメータの `spark.serializer=org.apache.spark.serializer.KryoSerializer –conf spark.sql.hive.convertMetastoreParquet=false` の値を生成します。

AWS Glue が提供したライブラリのオーバーライド

AWS Glue でサポートされていない Hudi のバージョンを使用するには、独自の Hudi ライブラリ JAR ファイルを指定します。独自の JAR ファイルを使用するには:

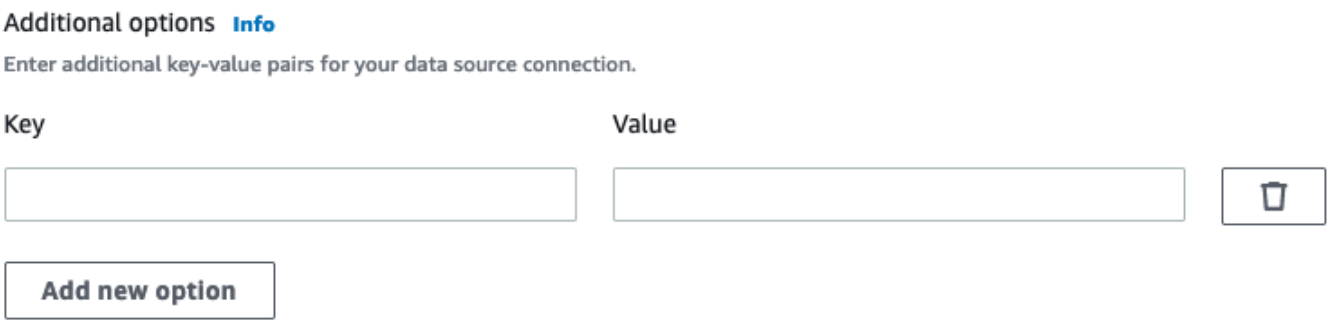
- `--extra-jars` ジョブパラメータを使用します。例えば、`'--extra-jars': 's3pathtojarfile.jar'` と指定します。詳細については、「AWS Glue ジョブのパラメータ」を参照してください。
- `--datalake-formats` ジョブパラメータの値として、`hudi` を含めることはできません。値として空白の文字列を入力すると、AWS Glue はデータレイクライブラリを自動的に提供することができません。詳細は、「AWS Glue での Hudi フレームワークの使用」を参照してください。

AWS Glue Studio でデータレイクフレームワークを使用する

データソースでデータレイクフレームワークを使用する

Amazon S3 データソースでデータレイクフレームワークを使用する

1. [Source] (ソース) メニューで、[Amazon S3] を選択します。
2. Amazon S3 ソースタイプとしてデータカタログテーブルを選択する場合は、データベースとテーブルを選択します。
3. AWS Glue Studio に、Delta Lake と Amazon S3 URL がフォーマットとして表示されます。
4. [Additional options] (追加オプション) をクリックし、キー値ペアを入力します。例えば、キーと値のペアは次のようになります。キー: timestampAsOf、値: 2023-02-24 14:16:18。



5. Amazon S3 ソースタイプとして Amazon S3 の場所を選択する場合は、[Browse Amazon S3] (Amazon S3 を参照) をクリックして Amazon S3 URL を選択します。
6. [Data format] (データ形式) で [Delta Lake] を選択します。

> **Note**
>
> AWS Glue Studio が、選択した Amazon S3 フォルダまたはファイルからスキーマを推測できない場合は、[Additional options] (追加オプション) を選択し、新しいフォルダまたはファイルを選択します。
> [Additional options] (追加オプション) の [Schema inference] (スキーマ推論) から、次のオプションを選択します。
>
> - [AWS Glue Studio にサンプルファイルを自動選択させる] — スキーマを推測できるよう、AWS Glue Studio が Amazon S3 の場所にあるサンプルファイルを選択します。自動選択されたファイルは [Auto-sampled file] (自動サンプル化ファイル) フィールドで確認できます。
> - [Choose a sample file from Amazon S3] (Amazon S3 からサンプルファイルを選択) - [Browse Amazon S3] (Amazon S3 を参照) をクリックし、使用する使用する Amazon S3 ファイルを選択します。

7. [Infer schema] (スキーマを推測) をクリックします。続いて、[Output schema] (出力スキーマ) タブをクリックすると、出力スキーマを確認できます。

データカタログのデータソースで Delta Lake フレームワークを使用する

1. [ソース] メニューで、[AWS Glue Studio データカタログ] を選択します。
2. [Data source properties] (データソースのプロパティ) タブで、データベースとテーブルを選択します。
3. AWS Glue Studio に、フォーマットタイプとして Delta Lake と Amazon S3 URL が表示されます。

> **Note**
>
> Delta Lake ソースが AWS Glue データカタログテーブルとしてまだ登録されていない場合は、次の 2 つの方法があります。
>
> 1. Delta Lake データストアの AWS Glue クローラーを作成します。詳細は、「Delta Lake データストアの設定オプションを指定する方法」を参照してください。

2. Amazon S3 データソースを使用して Delta Lake のデータソースを選択します。「Amazon S3 データソースでデータレイクフレームワークを使用する」を参照してください。

データターゲットで Delta Lake フォーマットを使用する

データカタログのデータターゲットで Delta Lake フォーマットを使用する

1. [ターゲット] メニューで [AWS Glue Studio データカタログ] を選択します。
2. [Data source properties] (データソースのプロパティ) タブで、データベースとテーブルを選択します。
3. AWS Glue Studio に、フォーマットタイプとして Delta Lake と Amazon S3 URL が表示されます。

Amazon S3 データソースで Delta Lake フォーマットを使用する

値を入力するか使用可能なオプションの中から選択し、Delta Lake 形式を設定します。

- [Compression Type] (圧縮タイプ) — 圧縮タイプ (非圧縮、Snappy) の中から 1 つ選択します。
- [Amazon S3 Target Location] (Amazon S3 ターゲットの場所) — [Browse S3] (S3 を参照) をクリックして Amazon S3 ターゲットの場所を選択します。
- [Data Catalog update options] (データカタログ更新オプション) — Glue Studio ビジュアルエディターでは、この形式のデータカタログの更新はサポートされていません。
  - Do not update the Data Catalog(データカタログを更新しない): (デフォルト) スキーマが変更されたり、新しいパーティションが追加された場合、ジョブでデータカタログを更新したくない場合は、このオプションを選択します。
  - AWS Glue ジョブの実行後にデータカタログを更新するには、AWS Glue クローラーを実行またはスケジュールします。詳細は、「Delta Lake データストアの設定オプションを指定する方法」を参照してください。
- [Partition keys] (パーティションキー): 出力でパーティションキーとして使用する列を選択します。さらにパーティションキーを追加するには、[Add a partition key] (パーティションキーの追加) を選択します。
- [Additional options] (追加オプション) をクリックして、キー値ペアを入力します。例えば、キーと値のペアは次のようになります。キー: timestampAsOf、値: 2023-02-24 14:16:18。

AWS Glue Studio での Apache Iceberg フレームワークの使用

データターゲットで Apache Iceberg フレームワークを使用する

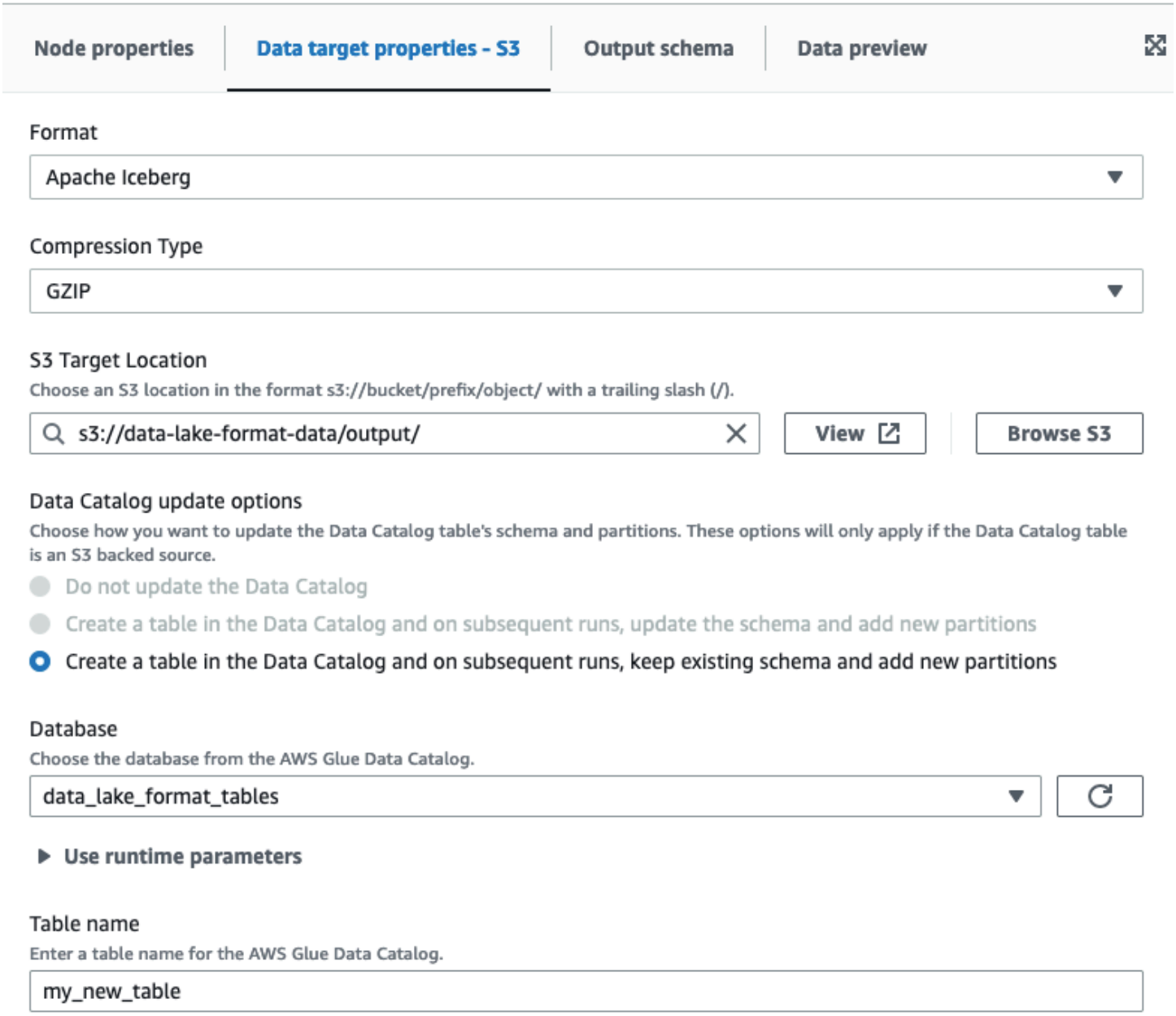
データカタログのデータターゲットで Apache Iceberg フレームワークを使用する

1. [ターゲット] メニューで [AWS Glue Studio データカタログ] を選択します。
2. [Data source properties] (データソースのプロパティ) タブで、データベースとテーブルを選択します。
3. AWS Glue Studio に、フォーマットタイプとして Apache Iceberg と Amazon S3 URL が表示されます。

Amazon S3 データターゲットで Apache Iceberg フレームワークを使用する

値を入力するか、使用可能なオプションの中から選択し、Apache Iceberg 形式を設定します。

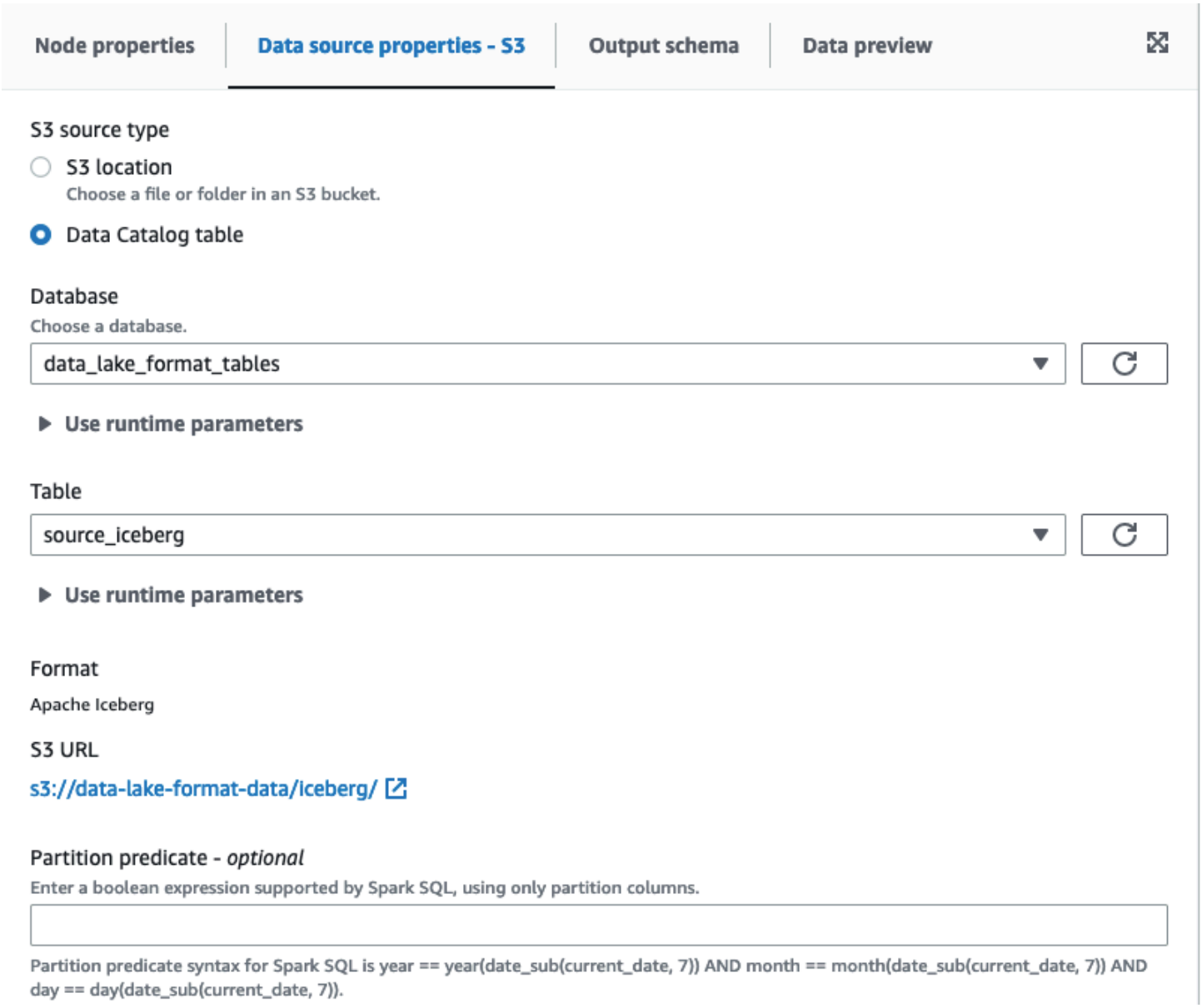
- 形式 — ドロップダウンメニューから [Apache Iceberg] を選択します。
- [Amazon S3 ターゲットの場所] — [S3 を参照] をクリックして Amazon S3 ターゲットの場所を選択します。
- [データカタログの更新オプション] — 続けるには、[データカタログでテーブル作成し、それ以降の実行時に既存のスキーマを保持して新しいパーティションを追加する] を選択する必要があります。AWS Glue を使用して新しい Iceberg テーブルを作成するには、Iceberg テーブルのカタログとして Data Catalog を設定する必要があります。Data Catalog に登録されている既存の Iceberg テーブルを更新するには、ターゲットとして Data Catalog を選択します。
  - [データベース] — Data Catalog からデータベースを選択します。
  - [テーブル名] – テーブルの名前を値として入力します。Apache Iceberg のテーブル名は、すべて小文字でなければなりません。スペースは使用できないため、必要な場合はアンダースコアを使用してください。例えば、「data_lake_format_tables」のようにします。



Amazon S3 データソースで Apache Iceberg フレームワークを使用する

データカタログのデータソースで Apache Iceberg フレームワークを使用する

1. [ソース] メニューで、[AWS Glue Studio データカタログ] を選択します。
2. [Data source properties] (データソースのプロパティ) タブで、データベースとテーブルを選択します。
3. AWS Glue Studio に、フォーマットタイプとして Apache Iceberg と Amazon S3 URL が表示されます。



Amazon S3 データソースで Apache Iceberg フレームワークを使用する

Apache Iceberg は、AWS Glue Studio の Amazon S3 ソースノードのデータオプションとしては使用できません。

Visual ETL ジョブを使用してデータソースに接続する

新しいジョブの作成中、AWS Glue でのビジュアル ETL ジョブ作成時に接続を使用して、データに接続できます。これには、コネクタを使用してデータを読み込むソースノードと、データを書き出す場所を指定するターゲットノードを追加します。

トピック

- データソースノードのプロパティを変更する
- データソースにデータカタログテーブルを使用する
- データソースにコネクタを使用する
- データソースに Amazon S3 内のファイルを使用する
- ストリーミングデータソースの使用
- リファレンス

データソースノードのプロパティを変更する

データソースのプロパティを指定するには、初めにジョブ図でデータソースノードを選択します。次に、ノードの詳細パネルの右側で、ノードのプロパティを設定します。

データソースノードのプロパティを変更するには

1. 新規または保存済みのジョブのビジュアルエディタに移動します。
2. ジョブ図でデータソースノードを選択します。
3. ノードの詳細パネルの [Node properties] (ノードのプロパティ) タブを選択して、次の情報を入力します。
   - Name (名前): (オプション) ジョブ図でノードに関連付ける名前を入力します。この名前は、このジョブのすべてのノードにおいて一意である必要があります。
   - Node type (ノードタイプ): ノードタイプによって、ノードで実行されるアクションが決定します。[Node type] (ノードタイプ) のオプションのリストで、見出し [Data source] (データソース) の下にリストされている値のいずれかを選択します。
4. データソースのプロパティの情報を設定します。詳細については、次のセクションを参照してください。
   - データソースにデータカタログテーブルを使用する
   - データソースにコネクタを使用する
   - データソースに Amazon S3 内のファイルを使用する
   - ストリーミングデータソースの使用
5. (オプション) ノードおよびデータソースのプロパティを設定した後、ノードのパネルの [Output schema] (出力スキーマ) タブをクリックすると、データソースのスキーマを表示できます。ジョブ内の任意のノードに対してこのタブを初めて選択すると、データにアクセスする IAM ロール

を指定するよう求められます。[Job details] (ジョブの詳細) タブで IAM ロールをまだ指定していない場合、ここで IAM ロールを入力するよう求められます。

6. (オプション) ノードおよびデータソースのプロパティを設定した後、ノードの詳細パネルの [Data preview] (データのプレビュー) タブを選択して、データソースからのデータセットをプレビューできます。ジョブ内の任意のノードに対してこのタブを初めて選択すると、データにアクセスする IAM ロールを指定するよう求められます。この機能を使用するには費用がかかり、IAM ロールを指定するとすぐに請求が開始します。

データソースにデータカタログテーブルを使用する

Amazon S3 とコネクタを除くすべてのデータソースでは、選択するソースタイプの AWS Glue Data Catalog にテーブルが存在する必要があります。AWS Glue はデータカタログテーブルを生成しません。

データカタログテーブルに基づいてデータソースノードを設定するには

1. 新規または保存済みのジョブのビジュアルエディタに移動します。
2. ジョブ図でデータソースノードを選択します。
3. [Data source properties] (データソースのプロパティ) タブを選択して、次の情報を入力します。

   - S3 source type (S3 ソースタイプ): (Amazon S3 データソースのみ) [Select a Catalog table] (Catalog テーブルを選択) オプションを選択して、既存の AWS Glue Data Catalog テーブルを使用します。
   - Database (データベース): このジョブに使用するソーステーブルを含むデータカタログのデータベースを選択します。検索フィールドを使用して、名前でデータベースを検索できます。
   - Table (テーブル): ソースデータに関連付けられたテーブルをリストから選択します。このテーブルは、既に AWS Glue Data Catalog に存在している必要があります。検索フィールドを使用して、名前でテーブルを検索できます。
   - Partition predicate (パーティション述語):(Amazon S3 データソースのみ) パーティション列のみを含む Spark SQL に基づいてブール式を入力します。例: `"(year=='2020' and month=='04')"`
   - Temporary directory (一時ディレクトリ): (Amazon Redshift データソースのみ) ETL ジョブが一時的な中間結果を書き込める Amazon S3 の作業ディレクトリの場所のパスを入力します。
   - Role associated with the cluster (クラスターに関連付けられたロール): (Amazon Redshift データソースのみ) Amazon Redshift クラスターのアクセス許可を含む、使用する ETL ジョブの

ロールを入力します。詳細については、「the section called “データソースとデータターゲットのアクセス許可”」を参照してください。

データソースにコネクタを使用する

ノードタイプにコネクタを選択する場合、カスタムコネクタを使用したジョブのオーサリング の説明に従って、データソースのプロパティの設定を完了します。

データソースに Amazon S3 内のファイルを使用する

データソースとして Amazon S3 を選択する場合、次のいずれかを選択できます。

- データカタログデータベースおよびテーブル。
- Amazon S3 内のバケット、フォルダ、またはファイル。

Amazon S3 バケットをデータソースとして使用する場合、AWS Glue は、ファイルの一つ、あるいはサンプルファイルとして指定したファイルを使うことによって、指定した場所にあるデータのスキーマを検出します。[Infer schema] (スキーマを推測) ボタンをクリックすると、スキーマの検出が行われます。Amazon S3 の場所またはサンプルファイルを変更する場合は、[Infer schema] (スキーマを推測) を再度クリックして、新しい情報を使用しスキーマの検出を実行します。

Amazon S3 内のファイルから直接読み取るデータソースノードを設定するには

1. 新規または保存済みのジョブのビジュアルエディタに移動します。
2. Amazon S3 ソース用に、ジョブ図でデータソースノードを選択します。
3. [Data source properties] (データソースのプロパティ) タブを選択して、次の情報を入力します。

   - S3 source type (S3 ソースタイプ): (Amazon S3 データソースのみ) [S3 location] (S3 の場所) オプションを選択します。
   - S3 URL (S3 の URL): ジョブのデータを含む Amazon S3 バケット、フォルダ、またはファイルへのパスを入力します。[Browse S3] (S3 をブラウズ) を選択すると、アカウントで利用可能な場所からパスを選択できます。
   - Recursive : S3 の場所にある子フォルダ内のファイルからデータをAWS Glue に読ませたい場合、このオプションを選択します。

     子フォルダにパーティション化されたデータが含まれている場合、AWS Glue は、フォルダ名中で指定されているパーティション情報をデータカタログに追加しません。例えば、Amazon S3 で、次のフォルダについて考えてみます。

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

[Recursive] を選択し、`sales` フォルダを S3 の場所として選択した場合、AWS Glue は、すべての子フォルダーのデータを読みますが、年、月、日のパーティションは作成しません。

- Data format (データ形式): データを保存する形式を選択します。JSON、CSV、または Parquet を選択できます。選択した値により、ソースファイルからデータを読み取る方法が AWS Glue ジョブに伝えられます。

> **Note**
>
> データのためにに正しい形式を選択していない場合、AWS Glue はスキーマは正しく推測する可能性はありますが、ジョブはソースファイルからのデータを正しく解析することはでないかもしれません。

選択した形式に応じて、追加の設定オプションを入力できます。

- JSON (JavaScript Object Notation)
  - JsonPath: テーブルスキーマの定義に使用されるオブジェクトを指す JSON パス。JSON パス式では、XPath 式が XML ドキュメントと組み合わせて使用されるのと同じように、常に JSON 構造を参照します。JSON パスの「ルートメンバーオブジェクト」は、オブジェクトや配列であっても、常に $ として参照されます。JSON パスは、ドット表記またはブラケット表記で記述できます。

    JSON パスの詳細については、GitHub ウェブサイトの「JsonPath」を参照してください。
  - Records in source files can span multiple lines (ソースファイル内のレコードが複数行にまたがることができる): CSV ファイル内の単一のレコードが複数行にまたがることができる場合は、このオプションを選択します。
- CSV (カンマ区切り値)
  - Delimiter (区切り記号): 行内の各列エントリの区切りを示す文字 (;、, など) を入力します。
  - Escape character (エスケープ文字): エスケープ文字として使用する文字を入力します。この文字は、その直後の文字は文字通りに解釈され、区切り文字として解釈しないことを示します。

- Quote character (引用符文字): 区切られた文字列を単一の値にグループ化するのに使用する文字を入力します。例えば、CSV ファイルに `"This is a single value"` などの値がある場合、ダブルクォート (") を選択します。
- Records in source files can span multiple lines (ソースファイル内のレコードが複数行にまたがることができる): CSV ファイル内の単一のレコードが複数行にまたがることができる場合は、このオプションを選択します。
- First line of source file contains column headers (ソースファイルの最初の行に列ヘッダーが含まれる): CSV ファイルの最初の行にデータではなく列ヘッダーが含まれている場合は、このオプションを選択します。

- Parquet (Apache Parquet 列指向ストレージ)

  Parquet 形式で保存されたデータに対する追加の設定はありません。

- Apache Hudi

  Apache Hudi 形式で保存されたデータに対する追加の設定はありません。

- Delta Lake

  Delta Lake 形式で保存されたデータに対する追加の設定はありません。

- Excel

  Excel 形式で保存されたデータに対する追加の設定はありません。

- Partition predicate (パーティション述語): データソースから読み取られたデータをパーティション分割するには、パーティション列のみを含む Spark SQL に基づいてブール式を入力します。例: `"(year=='2020' and month=='04')"`
- Advanced options： 特定のファイルに基づくデータのスキーマの検出を AWS Glue にさせたい場合は、このセクションを展開します。
  - Schema inference : AWS Glue にファイルを選択させる代わりに、特定のダイルを使用したい場合は、 オプションの 「Choose a sample file from S3」 を選択します。スキーマ推論は Excel ソースでは使用できません。
  - Auto-sampled file (自動サンプリングファイル): スキーマの推測に使用する Amazon S3 内のファイルへのパスを入力します。

  データソースノードを編集中に、選択したサンプルファイルを変更する場合は、[Reload schema] (スキーマの再ロード) を選択して、新しいサンプルファイルを使用してスキーマを検出します。

4. [Infer schema] (スキーマを推測) ボタンをクリックして、Amazon S3 内のソースファイルからスキーマを検出します。Amazon S3 の場所またはサンプルファイルを変更する場合は、[Infer schema] (スキーマを推測) を再度選択して、新しい情報を使用してスキーマを推測します。

ストリーミングデータソースの使用

Amazon Kinesis Data Streams、Apache Kafka、および Amazon Managed Streaming for Apache Kafka (Amazon MSK) で、ストーミングからのデータを消費し、連続的に実行されるストリーミング抽出、変換、ロード (ETL) ジョブを作成できます。

ストリーミングデータソースのプロパティを設定するには

1. 新規または保存済みのジョブのビジュアルグラフエディタに移動します。
2. Kafka または Kinesis Data Streams のグラフでデータソースノードを選択します。
3. [Data source properties] (データソースのプロパティ) タブを選択して、次の情報を入力します。

   Kinesis

   - Kinesis source type: ストリームの詳細への直接のアクセスを使うためにオプションの[Stream details]を選択するか、、そこに格納されている情報を使用するために[Data Catalog table] (データカタログテーブル) を選択します。

     [Stream details]を選択すると、以下の追加情報を指定します。

     - Location of data stream：ストリームを現在のユーザーに関連付けるかどうか、または別のユーザーに関連付けるかどうかを選択します。
     - Region: ストリームが存在する[AWS リージョン] を選択します。この情報は、データストリームにアクセスするための ARN を構築するために使用されます。
     - Stream ARN: Kinesis Data Streams の Amazon リソースネーム (ARN)を入力します。ストリームが現在のアカウント内にある場合は、ドロップダウンリストからストリーム名を選択します。検索フィールドを使用して、名前、あるいは ARN でデータベースを検索できます。
     - Data format: リストから、データストリームで使用される形式を選択します。

       AWS Glue は、ストリーミングデータからスキーマを自動的に検出します。

     [Data Catalog table] (データカタログテーブル) を選択すると、以下の追加情報を指定します。

- Database : (オプション) ストリーミングデータソースに関連付けられたテーブルを含む AWS Glueデータカタログ内のデータベースを選択します。検索フィールドを使用して、名前でデータベースを検索できます。
- Table (テーブル): (オプション) ソースデータに関連付けられたテーブルをリストから選択します。このテーブルは、AWS Glue のデータカタログに既に存在している必要があります。検索フィールドを使用して、名前でテーブルを検索できます。
- Detect schema: このオプションを選択すると、スキーマの情報がデータカタログテーブルに格納されるのではなく、AWS Glue にストリーミングデータからスキーマを検出させます。[Stream details] オプションを選択すると、このオプションは自動的に有効になります。

- Starting position: デフォルトでは、ETL ジョブは[Earliest]オプションを使います。これは、ストリーム内の使用可能な最も古いレコードから始まるデータを読み取ることを意味します。代わりに[Latest]選択することができます。これは、ETL ジョブがストリーム内の最新のレコードの直後から読み出しを開始する必要があることを示します。
- Window size (ウィンドウサイズ): デフォルトでは、ETL ジョブによるデータ処理と書き出しは 100 秒ウィンドウ単位で行われます。これにより、データを効率的に処理しつつ、想定より遅く到着するデータに対して集計を実行できます。ウィンドウサイズを変更することで、適時性や集計の精度を高めることができます。

  AWS Glue ストリーミングジョブでは、読み込んだデータの追跡には、ジョブのブックマークではなくチェックポイントが使用されます。
- Connection options : 追加の接続オプションを指定するために、このセクションを展開してキーバリューペアを追加します。ここで指定できるオプションの詳細については、[AWS Glue Developer Guide]中の 「""connectionType": "kinesis"」を参照してください。

Kafka

- Apache Kafka source: オプションの[Stream details]を選択し、ストリームの詳細ストリーミングソースへの直接アクセスを使用するか、[Data Catalog table] を選択して代わりに、そこに格納されている情報を使用します。

  [Data Catalog table] を選択すると、以下の追加情報を指定します。
  - Database : (オプション) ストリーミングデータソースに関連付けられたテーブルを含む AWS Glueデータカタログ内のデータベースを選択します。検索フィールドを使用して、名前でデータベースを検索できます。

- Table (テーブル): (オプション) ソースデータに関連付けられたテーブルをリストから選択します。このテーブルは、AWS Glue のデータカタログに既に存在している必要があります。検索フィールドを使用して、名前でテーブルを検索できます。
- Detect schema : このオプションを選択し、スキーマの情報がデータカタログテーブルに格納されるのではなく、ストリーミングデータからスキーマを AWS Glue に検出させます。Stream details オプションを選択すると、このオプションは自動的に有効になります。

Stream detailsを選択すると、以下の追加情報を指定します。

- Connection name: Kafka データストリームのためのアクセス情報と認証情報を含む AWS Glue の接続を選択します。Kafka ストリーミングデータソースとの接続を使用する必要があります。接続が存在しない場合は、AWS Glue のコンソールを使用して Kafka データストリームのための接続を作成します。
- Topic name: 読み取り元のトピック名を入力します。
- Data format : Kafka イベントストリームからデータを読み取るときに使用する形式を選択します。

- Starting position: デフォルトでは、ETL ジョブは[Earliest] オプションを使用します。これは、ストリーム内の使用可能な最も古いレコードから始まるデータを読み取ることを意味します。代わりに[Latest]選択することができます。これは、ETL ジョブがストリーム内の最新のレコードの直後から読み出しを開始する必要があることを示します。
- Window size (ウィンドウサイズ): デフォルトでは、ETL ジョブによるデータ処理と書き出しは 100 秒ウィンドウ単位で行われます。これにより、データを効率的に処理しつつ、想定より遅く到着するデータに対して集計を実行できます。ウィンドウサイズを変更することで、適時性や集計の精度を高めることができます。

  AWS Glue ストリーミングジョブでは、読み込んだデータの追跡には、ジョブのブックマークではなくチェックポイントが使用されます。
- Connection options : 追加の接続オプションを指定するために、このセクションを展開してキーバリューペアを追加します ここで指定できるオプションの詳細については、 [AWS Glue Developer Guide]内の「"connectionType": "kafka"」を参照してください。

> **Note**
>
> 現在、ストリーミングデータソースでのデータのプレビューはサポートされていません。

リファレンス

ベストプラクティス

- AWS Glue を使用して Amazon S3 から Amazon Redshift にデータを段階的にロードする ETL サービスパイプラインを構築します。

ETL プログラミング

- AWS Glue での ETL の接続タイプとオプション
- JDBC connectionType の値
- Amazon Redshift との間でのデータ移動に関する詳細オプション

データターゲットノードの設定

データターゲットは、変換されたデータがジョブにより書き込まれる場所です。

データターゲットでのオプションの概要

データターゲット (データシンクとも呼ばれます) は、次のようになります。

- S3 – ジョブにより、選択した Amazon S3 の場所にある、指定した形式のファイルにデータが書き込まれます。

  データターゲットにパーティション列を設定すると、ジョブによりパーティションキーに基づいて Amazon S3 へのデータセットがディレクトリに書き込まれます。
- AWS Glue Data Catalog –データカタログ内のテーブルに関連付けられた情報を使用して、ジョブによりターゲットの場所に出力データが書き込まれます。

  テーブルは、手動で、またはクローラーを使用して作成できます。また、AWS CloudFormation テンプレートを使用して、データカタログにテーブルを作成できます。
- コネクタ – コネクタは、データストアと AWS Glue 間の通信を容易にするコードの一部です。コネクタや関連付けられた接続を使用して、ジョブにより出力データがターゲットの場所に書き込まれます。で提供されるコネクタをサブスクライブするには、AWS Marketplaceまたは、独自のカスタムコネクタを作成することもできます。詳細については、「AWS Glue Studio にコネクタを追加する」を参照してください。

ジョブにより Amazon S3 のデータターゲットに書き込みが行われる際、データカタログを更新するように選択できます。このオプションを使用すると、スキーマまたはパーティションの変更時に、クローラーをリクエストしなくてもデータカタログが更新されます。これにより、テーブルを最新の状態に保つことが容易になります。また、このオプションではデータカタログに新しいテーブルを追加したり、テーブルパーティションを更新したり、ジョブから直接テーブルのスキーマを更新できます。これにより、データを分析に使用できるようにするプロセスが簡略化されます。

データターゲットノードの編集

データターゲットは、変換されたデータがジョブにより書き込まれる場所です。

ジョブ図でデータターゲットノードを追加または設定するには

1. (オプション) ターゲットノードを追加する必要がある場合、ビジュアルエディタの上部のツールバーで [Target] (ターゲット) を選択し、その後 [S3] または [Glue Data Catalog] (Glue データカタログ) を選択します。

   - ターゲットに [S3] を選択した場合、ジョブにより指定した Amazon S3 の場所にある 1 つ以上のファイルにデータセットが書き込まれます。
   - ターゲットに [AWS Glue Data Catalog] を選択した場合、ジョブによりデータカタログから選択したテーブルによって記述された場所に書き込みが行われます。
2. ジョブ図でデータターゲットノードを選択します。ノードを選択すると、ページの右側にノードの詳細パネルが表示されます。
3. [Node properties] (ノードのプロパティ) タブを選択して、次の情報を入力します。

   - Name (名前): ジョブ図でノードに関連付ける名前を入力します。
   - Node type (ノードタイプ): 値は既に選択されていますが、必要に応じて変更できます。
   - Node parents (ノードの親): 親ノードは、ターゲットの場所に書き込む出力データを提供するジョブ図内のノードです。事前に設定されたジョブ図では、ターゲットノードで既に親ノードが選択されている必要があります。親ノードが表示されていない場合は、リストから親ノードを選択します。

     ターゲットノードは、単一の親ノードを持ちます。
4. データターゲットのプロパティ情報を設定する 詳細については、次のセクションを参照してください。

   - データターゲットに Amazon S3 を使用する
   - データターゲットにデータカタログテーブルを使用する

- データターゲットにコネクタを使用する

5. (オプション) データターゲットノードのプロパティの設定後、ノードの詳細パネルの [Output schema] (出力スキーマ) タブを選択して、データの出力スキーマを表示できます。ジョブ内の任意のノードに対してこのタブを初めて選択すると、データにアクセスする IAM ロールを指定するよう求められます。[Job details] (ジョブの詳細) タブで IAM ロールをまだ指定していない場合、ここで IAM ロールを入力するよう求められます。

データターゲットに Amazon S3 を使用する

Amazon S3 とコネクタを除くすべてのデータソースでは、選択するソースタイプの AWS Glue Data Catalog にテーブルが存在する必要があります。AWS Glue Studio はデータカタログテーブルを生成しません。

Amazon S3 に書き込むデータターゲットノードを設定するには

1. 新規または保存済みのジョブのビジュアルエディタに移動します。
2. ジョブ図でデータソースノードを選択します。
3. [Data source properties] (データソースのプロパティ) タブを選択して、次の情報を入力します。

   - Format (形式): リストから形式を選択します。データ結果に使用できる形式のタイプは次のとおりです。
     - JSON: JavaScript Object Notation。
     - CSV: カンマで区切られた値。
     - Avro: Apache Avro JSON バイナリ。
     - Parquet: データ形式として `DynamicFrames` 用に最適化されたカスタム Parquet ライターのタイプ。データに対して事前に計算されたスキーマをリクエストしなくても、スキーマを動的に計算して変更します。
     - ORC: Apache Optimized Row Columnar (ORC) 形式。
     - Apache Hudi: 増分データ処理とデータパイプラインの開発を簡素化する、オープンソースのデータレイク用ストレージフレームワーク。
     - Apache Iceberg: SQL テーブルと同様な機能を持つ、高性能なテーブルフォーマット。
     - Delta Lake: ACID トランザクションの実行、メタデータ処理のスケーリング、さらにストリーミングとバッチデータ処理の統合を支援する、オープンソースのデータレイクストレージフレームワーク。
     - XML: 拡張マークアップ言語 (XML)。

- Tableau Hyper: Tableau のインメモリデータエンジンテクノロジー。

これらの形式のオプションについての詳細は、AWS Glue デベロッパーガイドの「Format Options for ETL Inputs and Outputs in AWS Glue」を参照してください。

- 圧縮タイプ: CSV、JSON または Parquet のファイルタイプを使用して、任意でデータを圧縮するかどうかを選択できます。デフォルトでは圧縮なし、または [None] (なし) です。

ファイルタイプ	圧縮
JSON/CSV/XML	GZIP、BZIP2、BROTLI、DEFLATE、LZ4、Snappy
Parquet	Snappy、LZ4、LZO、BROTLI、GZIP
ORC	Snappy、ZLIB、非圧縮、LZO
Avro	GZIP、BZIP2、BROTLI、DEFLATE、LZ4、Snappy
Delta Lake	GZIP、BZIP2、BROTLI、DEFLATE、LZ4、Snappy
Apache Hudi	GZIP、LZO、Snappy
Apache Iceberg	GZIP、LZO、Snappy
Tableau Hyper	なし

- S3 Target Location (S3 ターゲットの場所): Amazon S3 バケットおよびデータを出力する場所。[Browse S3] (S3 をブラウズ) ボタンをクリックして、アクセスできる Amazon S3 バケットを表示し、バケットの 1 つをターゲットの宛先として選択できます。
- データカタログの更新オプション
  - Do not update the Data Catalog(データカタログを更新しない): (デフォルト) スキーマが変更されたり、新しいパーティションが追加された場合、ジョブでデータカタログを更新したくない場合は、このオプションを選択します。
  - Create a table in the Data Catalog and on subsequent runs, update the schema and add new partitions (データカタログにテーブルを作成し、それ以降の実行時にスキーマを更新して新しいパーティションを追加する): このオプションを選択すると、最初のジョブの実行時

に、ジョブによりデータカタログにテーブルが作成されます。それ以降のジョブの実行時にスキーマが変更されたり、新しいパーティションが追加されたりすると、ジョブによりデータカタログテーブルが更新されます。

また、データカタログからデータベースを選択し、テーブル名を入力する必要があります。

- [Create a table in the Data Catalog and on subsequent runs, keep existing schema and add new partitions] (データカタログでテーブル作成し、それ以降の実行時に既存のスキーマの保持して新しいパーティションを追加する): このオプションを選択すると、最初のジョブの実行時に、ジョブによりデータカタログにテーブルが作成されます。それ以降のジョブの実行時に、新しいパーティションを追加するため、ジョブでデータカタログテーブルが更新されます。

  また、データカタログからデータベースを選択し、テーブル名を入力する必要があります。

- ファイルのパーティショニング: 出力を保存するパーティショニングのタイプを選択します。
  - ファイルの自動生成 (推奨): これは生成されたファイル数のデフォルト値です。
  - 複数のファイル出力: 必要なファイル出力の数を指定します。最適なパフォーマンスを得るには、デフォルトで自動生成されたファイル数の値を使用します。
- Partition keys (パーティションキー): 出力でパーティションキーとして使用する列を選択します。さらにパーティションキーを追加するには、[Add a partition key] (パーティションキーの追加) を選択します。

ファイルのパーティショニングは、ターゲット形式としての Tableau Hyper ではサポートされていません。

データターゲットにデータカタログテーブルを使用する

Amazon S3 とコネクタを除くすべてのデータソースでは、選択するターゲットタイプの AWS Glue Data Catalog にテーブルが存在する必要があります。AWS Glue Studio はデータカタログテーブルを生成しません。

データカタログテーブルを使用するターゲットのデータプロパティを設定するには

1. 新規または保存済みのジョブのビジュアルエディタに移動します。
2. ジョブ図でデータターゲットノードを選択します。
3. [Data target properties] (データターゲットのプロパティ) タブを選択して、次の情報を入力します。

- Database (データベース): ターゲットとして使用するテーブルを含むデータベースをリストから選択します。このデータベースは、データカタログに既に存在している必要があります。
- Table (テーブル): 出力データのスキーマを定義するテーブルをリストから選択します。このテーブルは、データカタログに既に存在している必要があります。

  データカタログのテーブルは、列名、データ型の定義、パーティション情報、およびターゲットのデータセットに関するその他のメタデータで構成されています。ジョブにより、データカタログのこのテーブルで示されている場所に書き込みが行われます。

  データカタログでのテーブルの作成についての詳細は、AWS Glue デベロッパーガイドの「Defining Tables in the Data Catalog」を参照してください。
- データカタログの更新オプション
  - Do not change table definition (テーブルの定義を変更しない) : (デフォルト) スキーマが変更されたり、新しいパーティションが追加された場合、ジョブでデータカタログを更新したくない場合は、このオプションを選択します。
  - [Update schema and add new partitions] (スキーマを更新して新しいパーティションを追加する): このオプションを選択すると、スキーマが変更されたり新しいパーティションが追加される際に、ジョブによりデータカタログテーブルが更新されます。
  - [Keep existing schema and add new partitions] (既存のスキーマを保持して新しいパーティションを追加する): このオプションを選択すると、ジョブにより データカタログテーブルが更新され、新しいパーティションが追加されます。
  - Partition keys (パーティションキー): 出力でパーティションキーとして使用する列を選択します。さらにパーティションキーを追加するには、[Add a partition key] (パーティションキーの追加) を選択します。

データターゲットにコネクタを使用する

ノードタイプにコネクタを選択する場合、カスタムコネクタを使用したジョブのオーサリング の説明に従って、データターゲットのプロパティ設定を完了します。

ジョブスクリプトの編集またはアップロード

AWS Glue Studio のビジュアルエディタを使用して、ジョブスクリプトを編集したり、独自のスクリプトをアップロードします。

ジョブが AWS Glue Studio で作成された場合にのみ、ビジュアルエディタを使用してジョブノードを編集できます。 AWS Glue コンソールをAPI コマンドを通して、またはコマンドラインインターフェイス (CLI) を使用してジョブが作成された場合、AWS Glue Studio 中のスクリプトエディタを使用して、ジョブスクリプト、パラメータ、スケジュールを編集できます。ジョブをスクリプト専用モードに変換して、AWS Glue Studio 内に作成されたジョブのスクリプトを編集することもできます。

ジョブスクリプトを編集または独自のスクリプトをアップロードするには

1. 新しいジョブを作成する場合、[Jobs] (ジョブ) ページで、[Spark script editor] (Spark スクリプトエディタ) オプションを選択して Spark ジョブを作成するか、[Python Shell script editor] (Python シェルスクリプトエディタ) を選択して Python シェルジョブを作成します。新しいスクリプトを作成するか、既存のスクリプトをアップロードできます。[Spark script editor] (Spark スクリプトエディタ) を選択した場合、Scala スクリプトまたは Python スクリプトのいずれかを作成またはアップロードできます。[Python Shell script editor] (Python シェルスクリプトエディタ) を選択した場合、Python スクリプトの作成またはアップロードのみが可能です。

   新しいジョブを作成するオプションを選択した後、表示される [Options] (オプション) セクションで、スタータースクリプト (定型コードを含む新しいスクリプトを作成) から開始するか、ジョブスクリプトとして使用するローカルファイルをアップロードするかを選択できます。

   [Spark script editor] (Spark スクリプトエディタ) を選択した場合、Python または Scala スクリプトファイルのどちらかをアップロードできます。Scala スクリプトには、ファイル拡張子 `.scala` が必要です。Python スクリプトは、Python タイプのファイルとして認識される必要があります。[Python Shell script editor] (Python シェルスクリプトエディタ) を選択した場合、Python スクリプトファイルのみをアップロードできます。

   選択が完了したら、[Create] (作成) をクリックしてジョブを作成し、ビジュアルエディタを開きます。
2. 新規または保存済みのジョブのビジュアルジョブエディタに移動して、[Script] (スクリプト) タブを選択します。
3. スクリプトエディタのオプションを使用して新しいジョブを作成しておらず、既存のジョブのスクリプトを編集したことがない場合、[Script] (スクリプト) タブには、[Script (Locked)] (スクリプト (ロックされています)) という見出しが表示されます。これは、スクリプトエディタが読み取り専用モードであることを意味します。[Edit script] (スクリプトの編集) を選択して、編集用のスクリプトのロックを解除します。

スクリプトを編集可能にするには、AWS Glue Studio がジョブをビジュアルジョブからスクリプト専用ジョブに変換します。編集用にスクリプトをロック解除すると、保存後にこのジョブでビジュアルエディタを使用できなくなります。

確認ウィンドウで、[Confirm] (確認) を選択して続行するか、[Cancel] (キャンセル) を選択して引き続きビジュアルの編集にジョブを使用します。

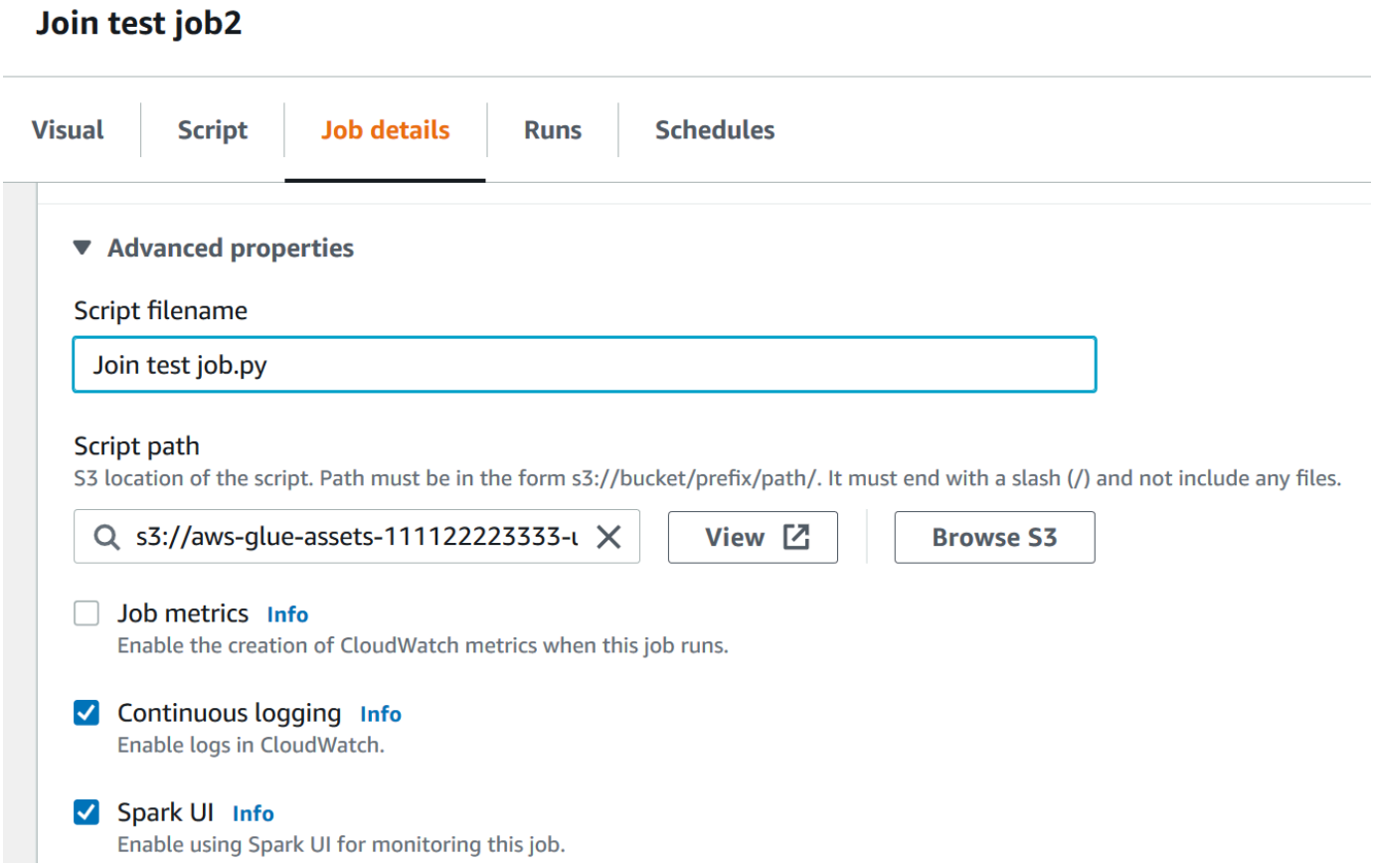
[Confirm] (確認) を選択すると、[Visual] (ビジュアル) タブはエディタに表示されなくなります。AWS Glue Studio を使って、スクリプトエディタを使用したスクリプトを変更したり、ジョブの詳細やスケジュールを変更したり、実行中のジョブを表示したりできます。

> **Note**
>
> ジョブを保存するまで、スクリプト専用ジョブへの変換は永続的ではありません。コンソールのウェブページを更新したり、ビジュアルエディタでジョブを保存する前に閉じて再度開いても、ビジュアルエディタで個々のノードを編集できます。

4. 必要に応じて、スクリプトを編集します。

   スクリプトの編集が終了したら、[Save] (保存) をクリックしてジョブを保存し、ジョブをビジュアルからスクリプト専用に永続的に変換します。

5. (オプション)[Script] タブ 上の[Download]ボタンを選択することにより、AWS Glue Studio コンソールからのスクリプトをダウンロードできます。このボタンをクリックすると、新しいブラウザウィンドウが開き、Amazon S3 内にあるその場所からのスクリプトが表示されます。[Job details] (ジョブの詳細) タブの [Script filename] (スクリプトファイル名) および [Script path] (スクリプトパス) パラメータにより、Amazon S3 のスクリプトファイルの名前と場所が決定します。



ジョブを保存すると、AWS Glue によりこれらのフィールドで指定された場所にジョブスクリプトが保存されます。Amazon S3 内のこの場所にあるスクリプトファイルを変更すると、AWS Glue Studio は次回ジョブを編集するときに、変更されたスクリプトをロードします。

AWS Glue Studio 中の Scala スクリプトの作成および編集

ジョブの作成のためにスクリプトエディタを選択する場合、デフォルトでは、ジョブのプログラミング言語は Python 3 に設定されています。スクリプトをアップロードするのではなく、新しいスクリプトの記述を選択した場合、AWS Glue Studio は、Python 内に記述された定型テキストを用いて新しいスクリプトを開始します。Scala スクリプトを記述する場合は、Scala を使用するよう最初にスクリプトエディタを設定する必要があります。

> **Note**
>
> ジョブのプログラミング言語として Scala を選択し、ビジュアルエディタを使用してジョブを設計する場合、生成されるジョブスクリプトは Scala で記述されます。それ以上のアクションは必要ありません。

AWS Glue Studio 中の新しい Scala スクリプトの作成

1. [Spark script editor] (Spark スクリプトエディタ) オプションを選択して、新しいジョブを作成します。
2. [Options] (オプション) の、[Create a new script with boilerplate code] (定型コードで新しいスクリプトを作成する) を選択します。
3. [Job details] (ジョブの詳細) タブを選択し、[Language] (言語) を [Scala] (Python 3 ではなく) に設定します。

   > **Note**
   >
   > [Spark script editor] (Spark スクリプトエディタ) オプションを選択してジョブを作成すると、ジョブの [Type] (タイプ) プロパティは Spark に自動的に設定されます。

4. [Script] (スクリプト) タブを選択します。
5. Python の定型テキストを削除します。これを、次の Scala の定型テキストに置き換えることができます。

```
import com.amazonaws.services.glue.{DynamicRecord, GlueContext}
import org.apache.spark.SparkContext
import com.amazonaws.services.glue.util.JsonOptions
import com.amazonaws.services.glue.util.GlueArgParser
import com.amazonaws.services.glue.util.Job

object MyScript {
  def main(args: Array[String]): Unit = {
    val sc: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(sc)

    }
}
```

6. エディタで Scala ジョブスクリプトを記述します。必要に応じて、他の import ステートメントを追加します。

AWS Glue Studio 中の Python シェルジョブの作成および編集

ジョブの作成に Python シェルスクリプトエディタを選択すると、既存の Python スクリプトをアップロードするか、新しい Python スクリプトを記述できます。新しいスクリプトを作成する場合、定型コードが新しい Python ジョブスクリプトに追加されます。

新しい Python シェルジョブを作成するには

「AWS Glue Studio でのジョブの開始」の手順を参照してください。

Python シェルジョブでサポートされているジョブプロパティは、Spark ジョブでサポートされているものとは異なります。次のリストは、[Job details] (ジョブの詳細) タブでの、Python シェルジョブで使用可能なジョブパラメータの変更について示しています。

- ジョブの [Type] (タイプ) プロパティは自動的に [Python Shell] に設定され、変更はできません。
- [Language] (言語) の代わりに、ジョブ用の [Python version] (Python バージョン) プロパティがあります。現在、AWS Glue Studio で作成される Python シェルジョブは Python 3.6 を使用します。
- [Glue version] (Glue バージョン) プロパティは Python シェルジョブには適用されないため、使用できません。
- [Worker type] (ワーカータイプ) および [Number of workers] (ワーカー数) の代わりに、[Data processing units] (データ処理ユニット) プロパティが表示されます。このジョブプロパティにより、ジョブの実行時に Python シェルによって消費されるデータ処理ユニット (DPU) の数が決定します。
- [Job bookmark] (ジョブのブックマーク) プロパティは Python シェルジョブではサポートされていないため、使用できません。
- [Advanced properties] (詳細プロパティ) の次のプロパティは Python シェルジョブでは使用できません。
  - ジョブのメトリクス
  - 連続ログ記録
  - [Spark UI] および [Spark UI logs path] (Spark UI ログパス)
  - 見出し [Libraries] (ライブラリ) の下の [依存 JARS パス]

ジョブ図でノードの親ノードを変更する

ノードの親を変更して、ジョブ図内のノードを移動したり、ノードのデータソースを変更できます。

親ノードを変更するには

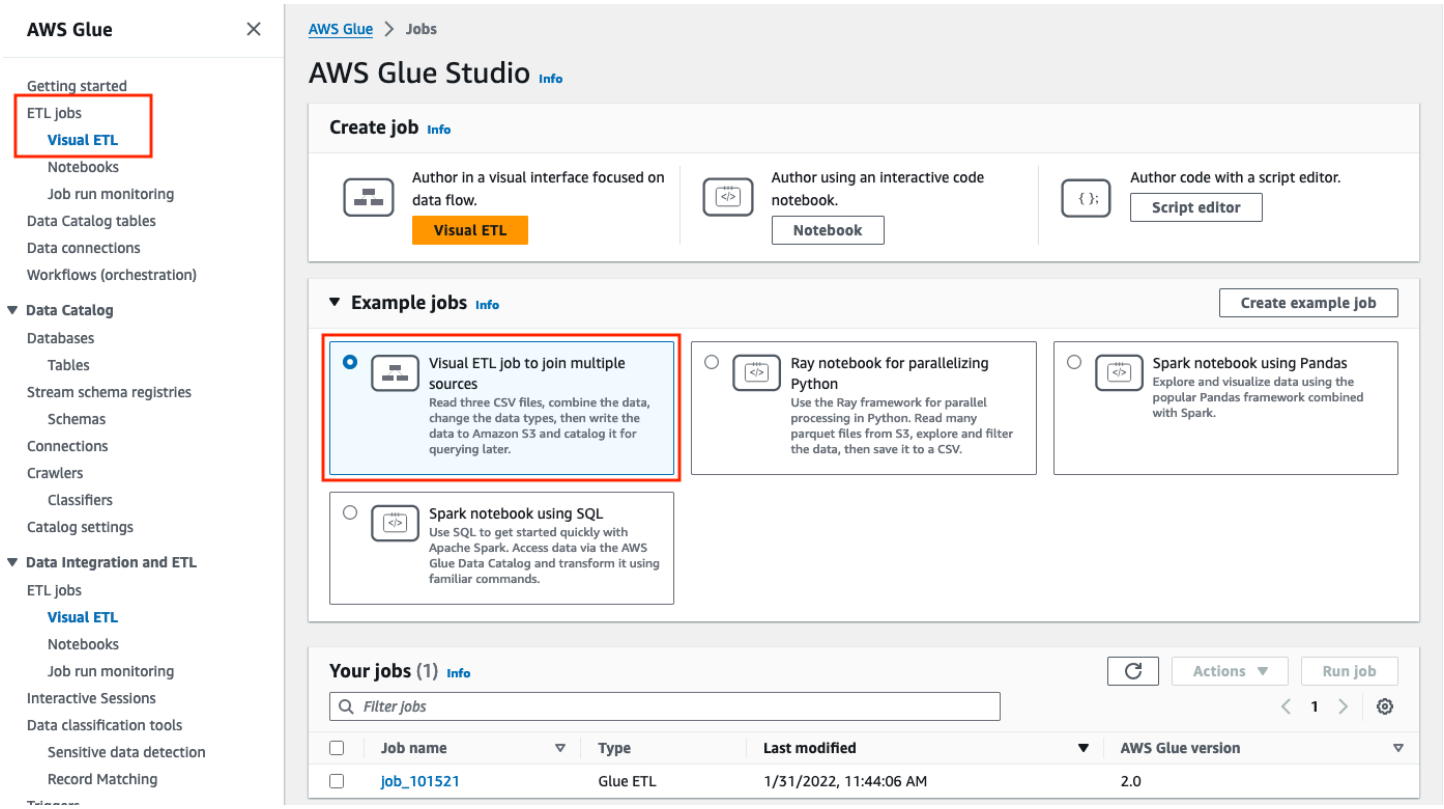
1. ジョブ図で変更するノードを選択します。
2. ノードの詳細パネルの [Node properties] (ノードのプロパティ) タブの、見出し [Node parents] (ノードの親) の下で、そのノードの現在の親を削除します。
3. リストから新しい親ノードを選択します。
4. 必要に応じて、新しく選択した親ノードと一致するように、ノードのその他のプロパティを変更します。

誤ってノードを変更した場合は、[Undo] (元に戻す) ボタンをクリックして、アクションを元に戻します。

ジョブ図からノードを削除する

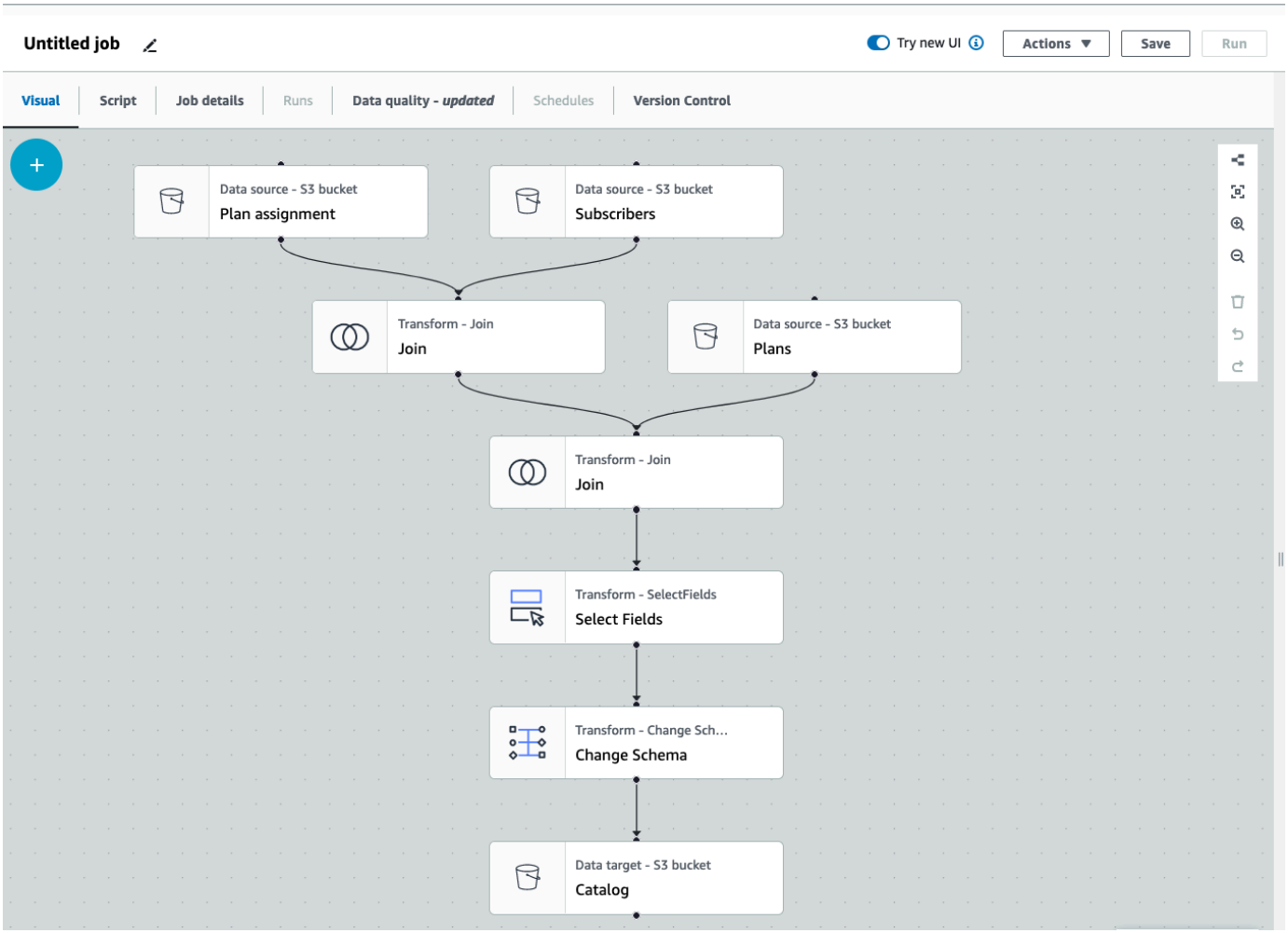
Visual ETL ジョブを使用するとき、削除されたノードに接続されているノードを再追加または再構築せずに、キャンバスからノードを削除できます。

以下の例では、[ETL ジョブ > Visual ETL] を選択し、次に [サンプルジョブ] で [複数のソースを統合する Visual ETL ジョブ] を選択して手順に従います。[サンプルジョブの作成] を選択してジョブを作成し、以下の手順に従います。



キャンバスからノードを削除する方法

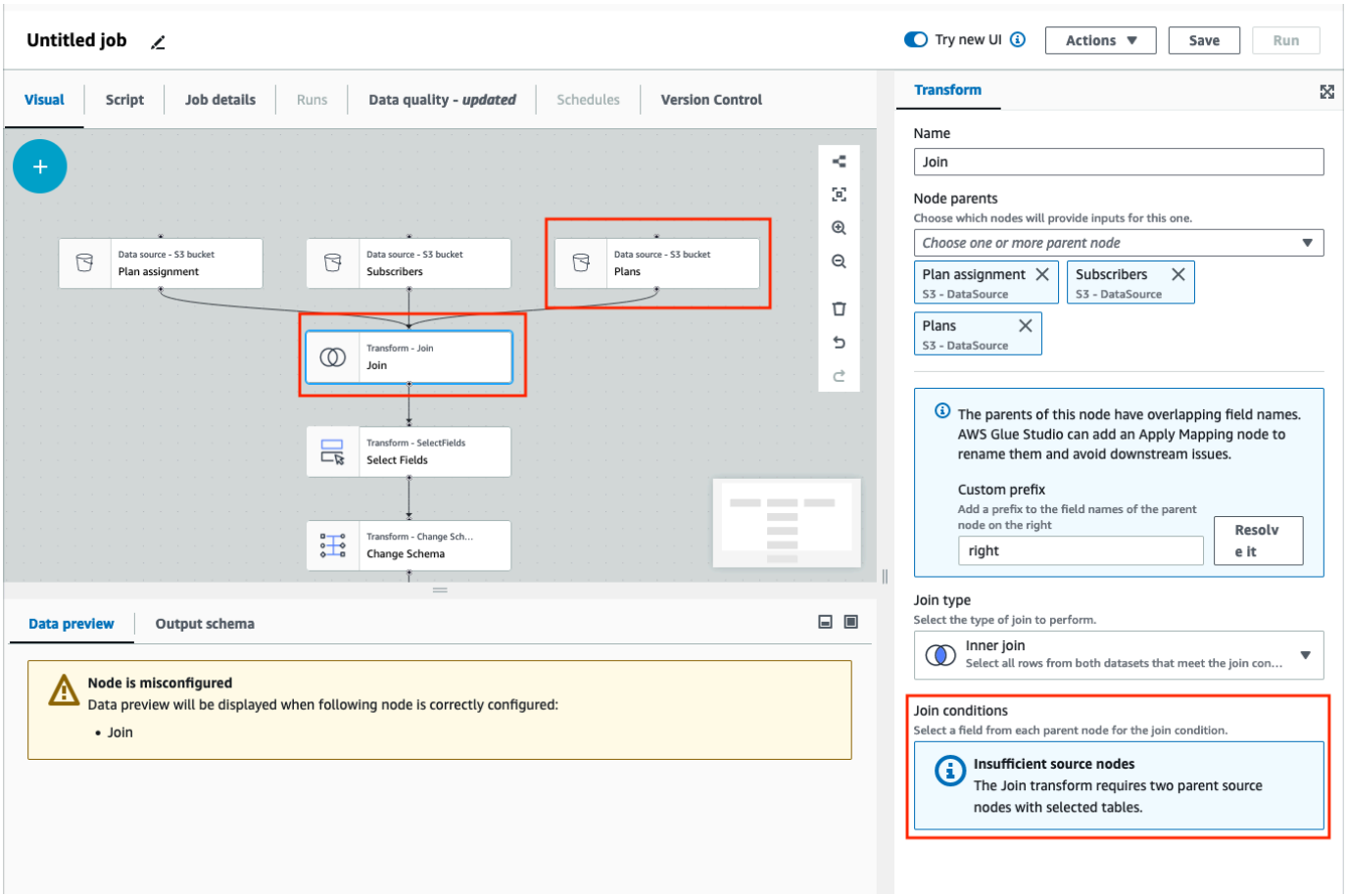
1. AWS Glue コンソールで、ナビゲーションメニューから [Visual ETL] を選択し、既存のジョブを選択します。ジョブキャンバスは、以下のように示されるサンプルジョブを表示します。



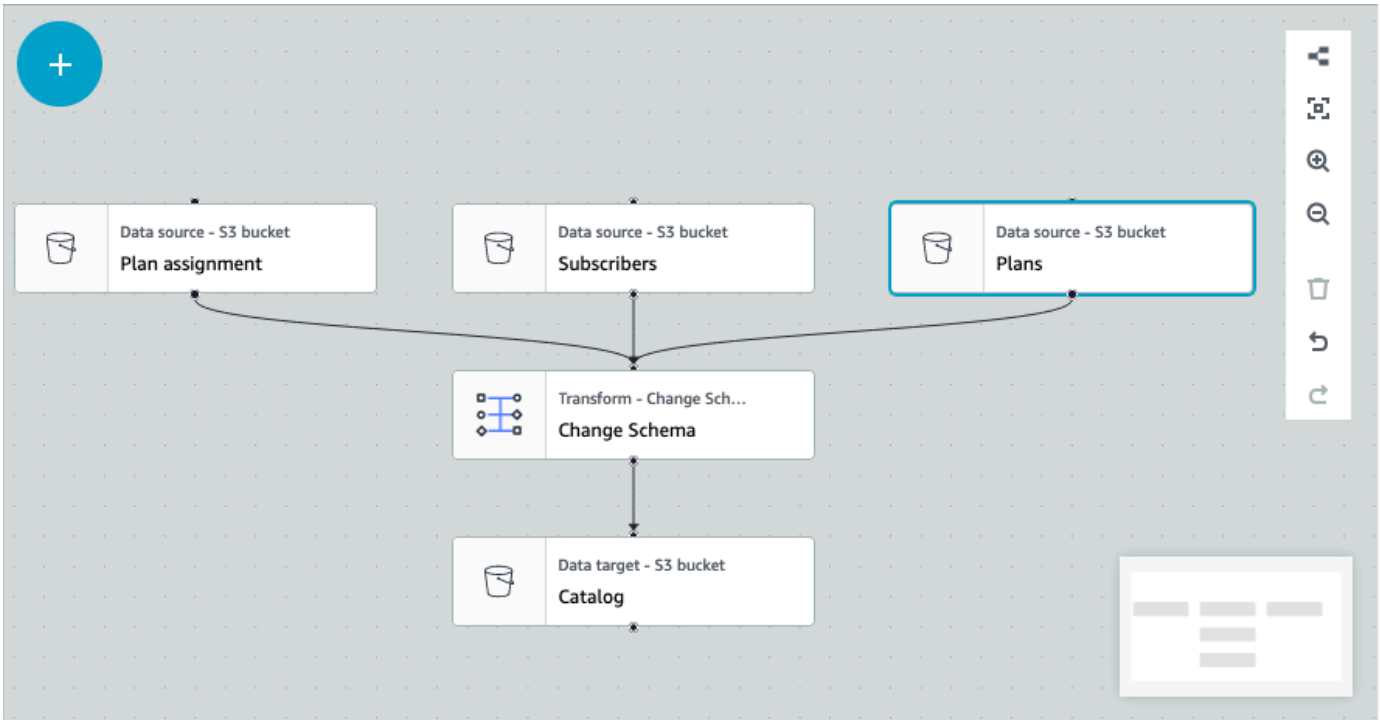
2. 削除するノードを選択します。キャンバスがノードにズームインします。キャンバスの右側のツールバーで [ゴミ箱] アイコンを選択します。これによってノードが削除され、そのノードに接続されているすべてのノードが移動し、ワークフローで削除されたノードを置き換えます。この例では、最初の [結合] ノードがキャンバスから削除されました。

   ワークフローでノードを削除した場合、ワークフローが無効にならないように AWS Glue がノードを再配置します。それでもノードの設定を修正する必要がある場合があります。

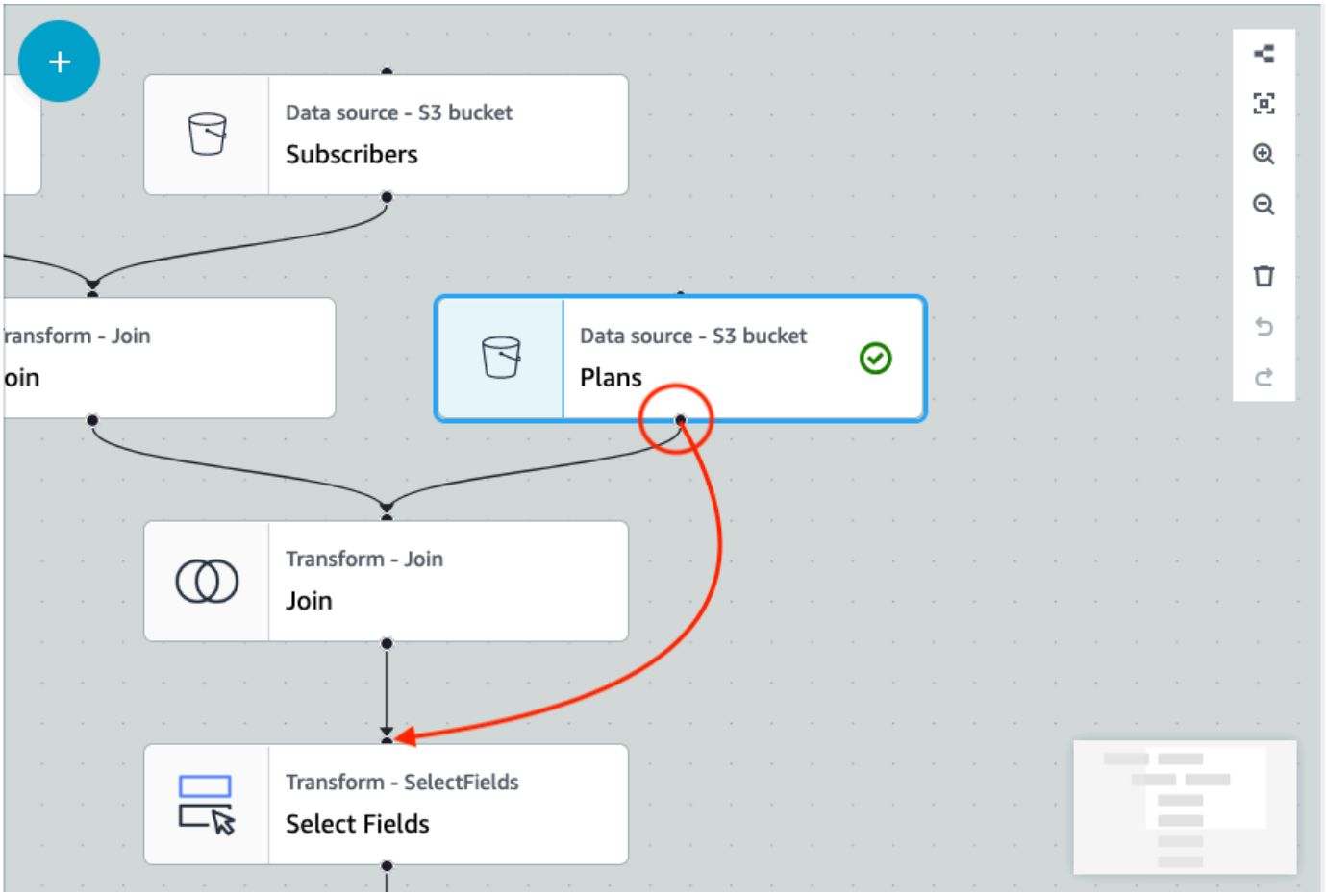
   この例では、[サブスクライバー] ノードの下にある [結合] ノードは削除されています。その結果、[プラン] ソースノードは最上位に移動され、子の [結合] ノードにまだ接続されています。[結合] は選択したテーブルを持つ 2 つの親ソースのノードが必要なため、[結合] ノードには追加の設定が必要になりました。キャンバスの右側の [変換] タブは、[結合条件] に不足している要件を表示しています。



3. 2 つ目の [結合] ノードおよび [フィールド選択] ノードを削除します。ノードが削除されると、ワークフローは以下の例のようになります。



4. ノード接続を変更するには、ノードのハンドルをクリックして接続を新しいノードにドラッグします。ノードを削除したり、論理フローでノードを再配置したりできるようにします。この例では、[プラン] ノードのハンドルをクリックし、赤い矢印で示されているように [結合] ノードに接続をドラッグすることにより、新しい接続を確立しています。



5. 操作を取り消す必要がある場合、キャンバスの右側のツールバーにある [ゴミ箱] アイコンの真下にある [元に戻す] アイコンを選択します。

AWS Glue データカタログノードにソースパラメータとターゲットパラメータを追加する

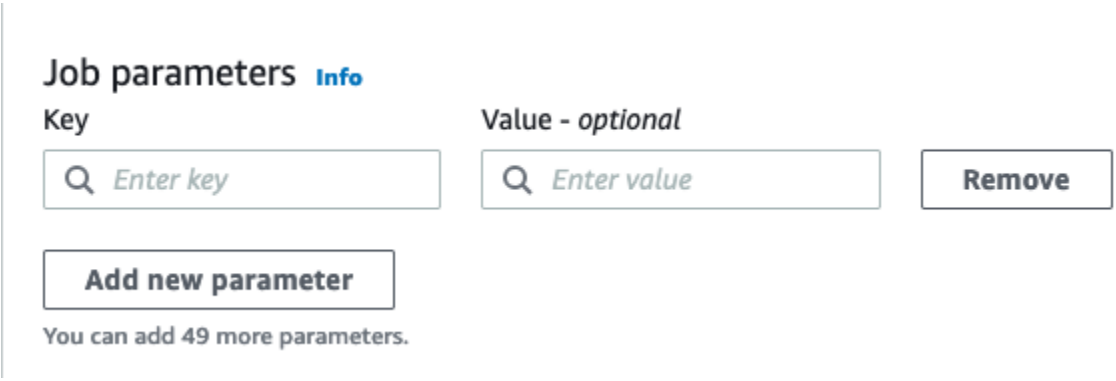
AWS Glue Studio によって、ビジュアルジョブをパラメータ化できます。本番環境と開発環境ではカタログテーブル名が異なる場合があるため、ジョブの実行時に実行されるデータベースとテーブルのランタイムパラメータを定義して選択できます。

ジョブのパラメータ化により、ソースとターゲットをパラメータ化して、AWS Glue データカタログノードを使用する際にそれらのパラメータをジョブに保存できます。ソースとターゲットをパラメータとして指定すると、特に同じジョブを複数の環境で使用する場合に、ジョブの再利用が可能になります。ソースとターゲットを管理する時間と労力を節約して、デプロイメント環境全体でコードをプロモートする際に役立ちます。さらに、指定したカスタムパラメータは、特定の実行のデフォルト引数をオーバーライドします。AWS Glue仕事。

ソースパラメータとターゲットパラメータを追加するには

AWS Glue データカタログノードをソースまたはターゲットとして使用しているかどうかにかかわらず、[Job details] (ジョブの詳細) タブの [Advanced properties] (詳細プロパティ) セクションでランタイムパラメータを定義できます。

1. ソースノードまたはターゲットノードとして AWS Glue データカタログノードを選択します。
2. [Job details] (ジョブの詳細) タブを選択します。
3. [Advanced properties] (詳細プロパティ) を選択します。
4. [Job parameters] (ジョブパラメータ) セクションで、キー値を入力します。例えば、`--db.source` はデータベースソースのパラメータになります。キー名の後に '-' (ダッシュ) が続いている限り、任意の名前をキーに入力できます。



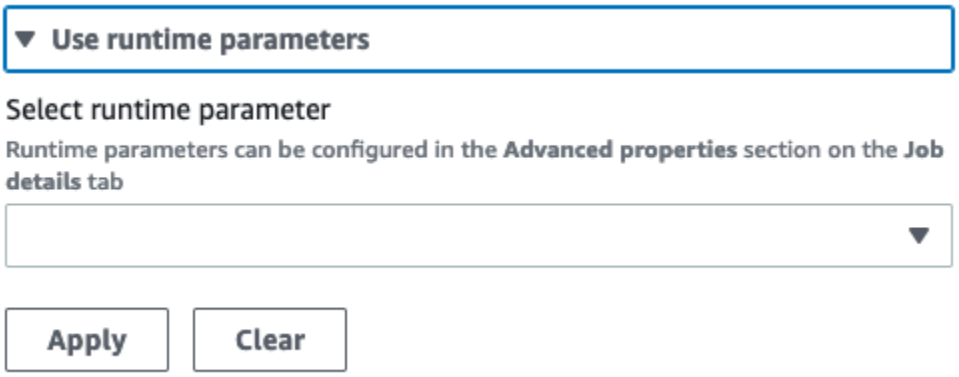
5. 値を入力します。例えば、`databasename` はパラメータ化されるデータベースの値になります。
6. さらにパラメータを追加する場合は、[Add new parameter] (新しいパラメータを追加) を選択します。最大 50 個のパラメータを使用できます。キーと値のペアを定義したら、AWS Glue データカタログノードでパラメータを使用できます。

ランタイムパラメータを選択するには

> **Note**
>
> データベースとテーブルのランタイムパラメータを選択するプロセスは、AWS Glue データカタログノードがソースであっても、ターゲットであっても同じです。

1. ソースノードまたはターゲットノードとして AWS Glue データカタログノードを選択します。

2. [Data source properties - Data Catalog] (データソースのプロパティ - データカタログ) タブの [Database] (データベース) で、[Use runtime parameters] (ランタイムパラメータを使用) を選択します。



3. ドロップダウンメニューからパラメータを選選択します。例えば、ソースデータベースに定義したパラメータを選択し、[Apply] (適用) を選択すると、データベースのドロップダウンメニューにデータベースが自動的に入力されます。
4. [Table] (テーブル) セクション で、ソーステーブルとして定義済みのパラメータを選択します。[Apply] (適用) を選択した場合は、使用するテーブルとしてテーブルが自動的に入力されます。
5. ジョブを保存して実行すると、AWS Glue Studio はジョブの実行中に選択されたパラメータを参照します。

AWS Glue で Git のバージョン管理システムを使用する

> **Note**
>
> ノートブックは現在、AWS Glue Studio のバージョン管理には対応していません。ただし、AWS Glue ジョブスクリプトとビジュアル ETL ジョブのバージョン管理はサポートされています。

リモートリポジトリがあり、そのリポジトリを使用して AWS Glue ジョブを管理する場合は、AWS Glue Studio または AWS CLI を使用して、リポジトリと AWS Glue のジョブへの変更を同期できます。この方法で変更を同期すると、ジョブは AWS Glue Studio からリポジトリにプッシュされるか、リポジトリから AWS Glue Studio にプルされます。

AWS Glue Studio での Git との統合により、次のことができるようになります。

- AWS CodeCommit、GitHub、GitLab、 Bitbucket など、Git のバージョン管理システムとの統合
- ビジュアルジョブやスクリプトジョブを使用するかどうかにかかわらず、AWS Glue Studio で AWS Glue ジョブを編集し、リポジトリに同期する
- ジョブのソースとターゲットをパラメータ化する
- リポジトリからジョブをプルし、AWS Glue Studio で編集する
- AWS Glue Studio のマルチブランチワークフローを利用して、ブランチからプルしたり、ブランチにプッシュしたりして、ジョブをテストする
- リポジトリからファイルをダウンロードし、ジョブを AWS Glue Studio にアップロードしてクロスアカウントジョブを作成する
- 選択した自動化ツール (Jenkins、AWS CodeDeploy など) を使用する

このビデオでは、AWS Glue を Git と統合し、継続的な共有コードパイプラインを構築する方法について紹介します。

IAM アクセス許可

ジョブに次の IAM アクセス許可のいずれかがあることを確認します。IAM アクセス許可の設定方法の詳細については、「AWS Glue Studio に対する IAM のアクセス許可の設定」を参照してください。

- `AWSGlueServiceRole`
- `AWSGlueConsoleFullAccess`

Git との統合では、少なくとも次のアクションが必要です。

- `glue:UpdateJobFromSourceControl` - バージョン管理システムに存在するジョブで AWS Glue を更新できるようにする
- `glue:UpdateSourceControlFromJob` - AWS Glue に保存されているジョブでバージョン管理システムを更新できるようにする
- `s3:GetObject` - バージョン管理システムにプッシュしている間にジョブのスクリプトを取得できるようにする
- `s3:PutObject` - ソース管理システムからジョブをプルする際にスクリプトを更新できるようにする

前提条件

ジョブをソース管理リポジトリにプッシュするには、以下が必要です。

- 管理者によってすでに作成されているリポジトリ
- リポジトリ内のブランチ
- 個人アクセストークン (Bitbucket の場合、これはリポジトリアクセストークン)
- リポジトリの所有者のユーザー名
- AWS Glue Studio にリポジトリの読み取りと書き込みを許可するためのアクセス許可をリポジトリに設定
  - [GitLab] — トークンスコープを api、read_repository、write_repository に設定
  - [Bitbucket] — アクセス許可を次のように設定します。
    - [Workspace membership] — 読み取り、書き込み
    - [プロジェクト] — 書き込み、管理者による読み取り
    - [リポジトリ] — 読み取り、書き込み、管理、削除

> **Note**
>
> AWS CodeCommit の使用時には、個人アクセストークンとリポジトリの所有者は必要ありません。「Git および AWS CodeCommit の開始方法」を参照してください。

AWS Glue Studio のソースコントロールリポジトリのジョブを使用する

AWS Glue Studio 以外にあるソース管理リポジトリからジョブを取得し、そのジョブを AWS Glue Studio で使用するための前提条件は、ジョブのタイプによって異なります。

ビジュアルジョブの場合:

- ジョブ名と一致するジョブ定義のフォルダと JSON ファイルが必要です

  例えば、以下のジョブ定義を参照してください。リポジトリのブランチには、フォルダと JSON ファイルの両方がジョブ名と一致する `my-visual-job/my-visual-job.json` パスが含まれている必要があります。

```
{
  "name" : "my-visual-job",
```

```
  "description" : "",
  "role" : "arn:aws:iam::aws_account_id:role/Rolename",
  "command" : {
    "name" : "glueetl",
    "scriptLocation" : "s3://foldername/scripts/my-visual-job.py",
    "pythonVersion" : "3"
  },
  "codeGenConfigurationNodes" : "{\"node-nodeID\":{\"S3CsvSource\":
{\"AdditionalOptions\":{\"EnableSamplePath\":false,\"SamplePath\":\"s3://notebook-
test-input/netflix_titles.csv\"},\"Escaper\":\"\",\"Exclusions\":[],\"Name\":\"Amazon
 S3\",\"OptimizePerformance\":false,\"OutputSchemas\":[{\"Columns\":[{\"Name\":
\"show_id\",\"Type\":\"string\"},{\"Name\":\"type\",\"Type\":\"string\"},{\"Name\":
\"title\",\"Type\":\"choice\"},{\"Name\":\"director\",\"Type\":\"string\"},{\"Name\":
\"cast\",\"Type\":\"string\"},{\"Name\":\"country\",\"Type\":\"string\"},{\"Name\":
\"date_added\",\"Type\":\"string\"},{\"Name\":\"release_year\",\"Type\":\"bigint\"},
{\"Name\":\"rating\",\"Type\":\"string\"},{\"Name\":\"duration\",\"Type\":\"string
\"},{\"Name\":\"listed_in\",\"Type\":\"string\"},{\"Name\":\"description\",\"Type
\":\"string\"}]}],\"Paths\":[\"s3://dalamgir-notebook-test-input/netflix_titles.csv
\"],\"QuoteChar\":\"quote\",\"Recurse\":true,\"Separator\":\"comma\",\"WithHeader
\":true}}}"
}
```

スクリプトジョブの場合:

- フォルダ、ジョブ定義の JSON ファイル、スクリプトが必要です。
- フォルダと JSON ファイルはジョブ名と一致している必要があります。スクリプト名は、ファイル拡張子と共にジョブ定義の `scriptLocation` と一致している必要があります

  例えば、以下のジョブ定義では、リポジトリ内のブランチにパス `my-script-job/my-script-job.json` および `my-script-job/my-script-job.py` が含まれている必要があります。スクリプト名は、スクリプトの拡張子を含む `scriptLocation` の名前と一致している必要があります。

```
{
  "name" : "my-script-job",
  "description" : "",
  "role" : "arn:aws:iam::aws_account_id:role/Rolename",
  "command" : {
    "name" : "glueetl",
    "scriptLocation" : "s3://foldername/scripts/my-script-job.py",
```

```
    "pythonVersion" : "3"
  }
}
```

制約事項

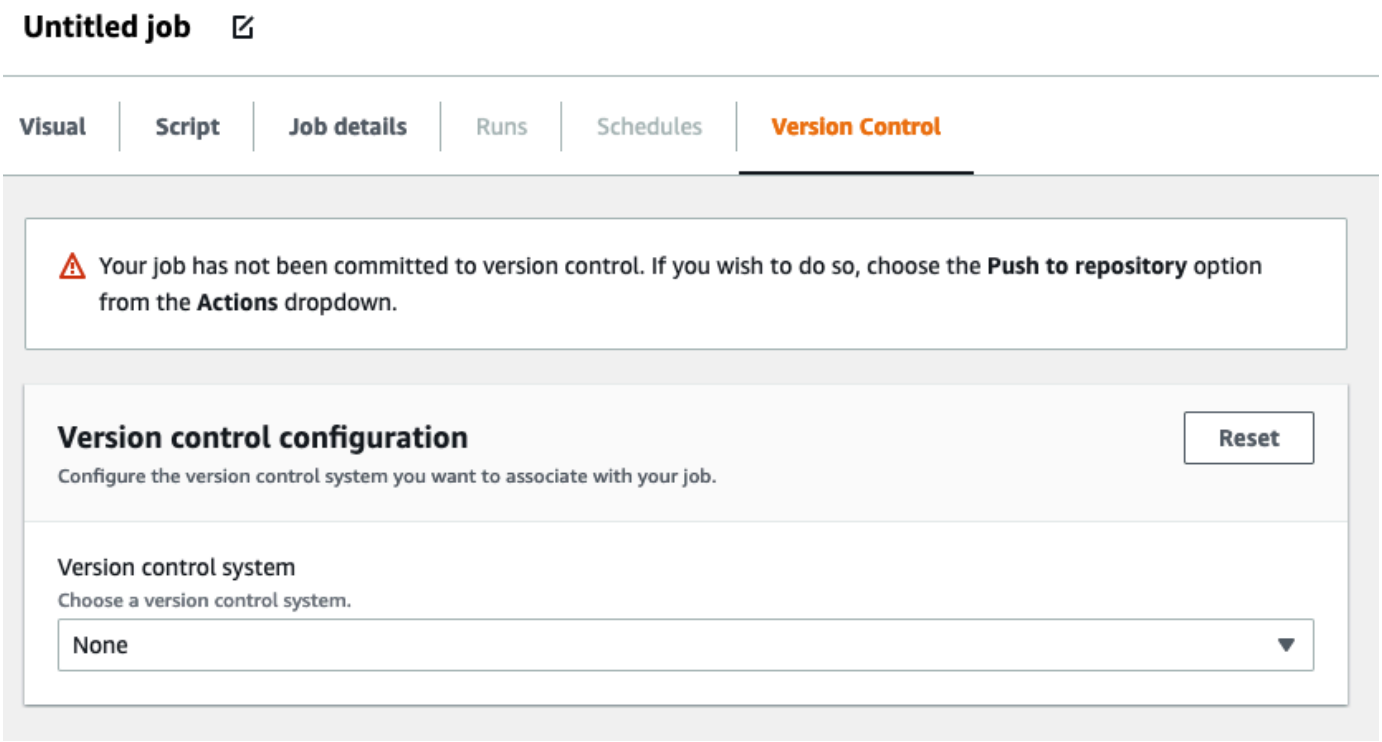
- 現在、AWS Glue は「GitLab-Groups」からのプッシュ・プルをサポートしていません。

バージョン管理リポジトリと AWS Glue の接続

バージョン管理リポジトリの詳細を入力して、AWS Glue Studio ジョブエディタの [Version Control] (バージョン管理) タブで管理できます。Git のリポジトリと統合するには、AWS Glue Studio にログインするたびにリポジトリに接続する必要があります。

Git バージョン管理システムに接続するには:

1. AWS Glue Studio で、新しいジョブを開始し、[Version Control] (バージョン管理) タブを選択します。



2. [Version control system] で、ドロップダウンメニューをクリックし、利用可能なオプションから Git サービスを選択します。

- AWS CodeCommit
- GitHub
- GitLab
- Bitbucket

3. 選択した Git バージョン管理システムによって、入力するフィールドは異なります。

   AWS CodeCommit の場合

   ジョブのリポジトリとブランチを選択して、リポジトリの設定を完了します。

   - [Repository] (リポジトリ) - AWS CodeCommit でリポジトリを設定した場合は、ドロップダウンメニューからリポジトリを選択します。リストに自動的にリポジトリが入力されます
   - [Branch] (ブランチ) - ドロップダウンメニューからブランチを選択します。
   - [Folder] (フォルダ) - オプション - ジョブを保存するフォルダの名前を入力します。空のままにすると、フォルダが自動的に作成されます。フォルダ名はデフォルトでジョブ名になります。

   GitHub の場合:

   次のフィールドに入力して GitHub の設定を完了します。

   - [Personal access token] (個人用のアクセストークン) - GitHub リポジトリによって提供されるトークンです。個人用のアクセストークンの詳細については、「GitHub Docs」を参照してください。
   - [Repository owner] (リポジトリ所有者) - GitHub リポジトリの所有者です。

   GitHub からリポジトリとブランチを選択して、リポジトリの設定を完了します。

   - [Repository] (リポジトリ) - GitHub でリポジトリを設定している場合は、ドロップダウンメニューからリポジトリを選択します。リストに自動的にリポジトリが入力されます
   - [Branch] (ブランチ) - ドロップダウンメニューからブランチを選択します。
   - [Folder] (フォルダ) - オプション - ジョブを保存するフォルダの名前を入力します。空のままにすると、フォルダが自動的に作成されます。フォルダ名はデフォルトでジョブ名になります。

GitLab の場合:

> **Note**
>
> 現在、AWS Glue は「GitLab-Groups」からのプッシュ・プルをサポートしていません。

- [個人用アクセストークン] - これは GitLab リポジトリによって提供されるトークンです。個人用のアクセストークンの詳細については、「GitLab の個人アクセストークン」を参照してください。
- [リポジトリの所有者] - これは GitLab リポジトリの所有者です。

GitLab からリポジトリとブランチを選択して、リポジトリの設定を完了します。

- [リポジトリ] - GitLab でリポジトリを設定している場合は、ドロップダウンメニューからリポジトリを選択します。リストに自動的にリポジトリが入力されます
- [Branch] (ブランチ) - ドロップダウンメニューからブランチを選択します。
- [Folder] (フォルダ) - オプション - ジョブを保存するフォルダの名前を入力します。空のままにすると、フォルダが自動的に作成されます。フォルダ名はデフォルトでジョブ名になります。

Bitbucket の場合:

- [アプリパスワード] – Bitbucket はリポジトリアクセストークンではないく、アプリパスワードを使用します。アプリパスワードの詳細については、「アプリパスワード」を参照してください。
- [リポジトリの所有者] - これは Bitbucket リポジトリの所有者です。Bitbucket では、所有者はリポジトリの作成者です。

Bitbucket からワークスペース、リポジトリ、ブランチ、フォルダを選択することで、リポジトリの設定を完了します。

- [ワークスペース] — Bitbucket にワークスペースを設定している場合は、ドロップダウンメニューからワークスペースを選択します。ワークスペースは自動的に入力されます

- [リポジトリ] - Bitbucket でリポジトリを設定した場合は、ドロップダウンメニューからリポジトリを選択します。リポジトリは自動的に入力されます
- [分岐] - ドロップダウンメニューから分岐を選択します。分岐は自動的に入力されます
- [Folder] (フォルダ) - オプション - ジョブを保存するフォルダの名前を入力します。空のままにすると、フォルダがジョブ名で自動的に作成されます。

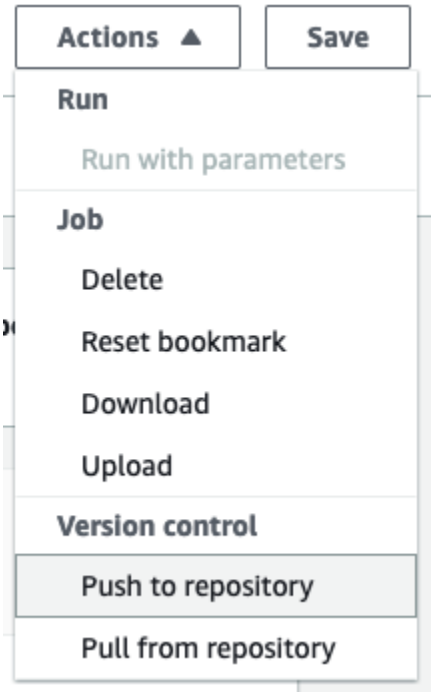
4. AWS Glue Studio ジョブの上部で、[Save] (保存) を選択します。

ソースリポジトリに AWS Glue ジョブをプッシュする

バージョン管理システムの詳細を入力したら、AWS Glue Studio でジョブを編集してソースリポジトリにプッシュできます。プッシュやプルなどの Git の概念に慣れていない場合は、「Git および AWS CodeCommit の開始方法」で、このチュートリアルを参照してください。

ジョブをリポジトリにプッシュするには、バージョン管理システムの詳細を入力してジョブを保存する必要があります。

1. AWS Glue Studio ジョブで、[Actions] (アクション) を選択します。これにより、追加のメニューオプションが開きます。



2. [Push to repository] (リポジトリにプッシュ) を選択します。

   このアクションによってジョブが保存されます。リポジトリにプッシュすると、AWS Glue Studio は最後に保存した変更をプッシュします。リポジトリ内のジョブが自分または別のユー

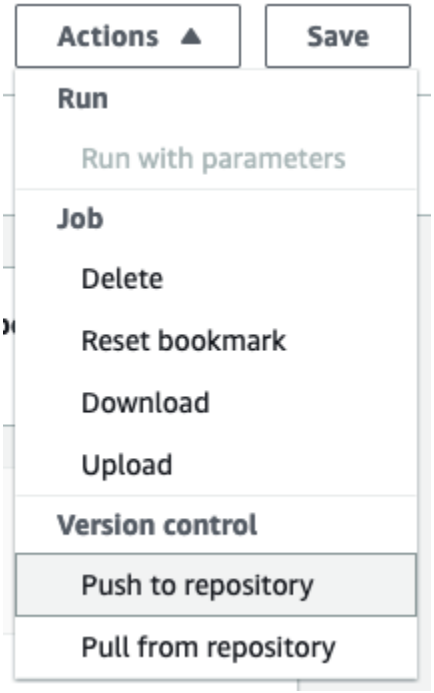
ザーによって変更され、AWS Glue Studio のジョブと同期していない場合は、AWS Glue Studio からジョブをプッシュする際にリポジトリ内のジョブが AWS Glue Studio に保存されたジョブで上書きされます。

3. [Confirm] (確認) を選択してアクションを完了します。これにより、リポジトリに新しいコミットが作成されます。AWS CodeCommit を使用している場合、AWS CodeCommit での最新のコミットへのリンクが確認メッセージに表示されます。

ソースリポジトリからの AWS Glue ジョブのプル

Git リポジトリの詳細を [Version control] (バージョン管理) タブに入力したら、リポジトリからジョブを取得して AWS Glue Studio で編集することもできます。

1. AWS Glue Studio ジョブで、[Actions] (アクション) を選択します。これにより、追加のメニューオプションが開きます。



2. [Pull from repository] (リポジトリからプル) を選択します。
3. [確認] を選択します。これにより、リポジトリから最新のコミットが取得され、AWS Glue Studio のジョブが更新されます。
4. AWS Glue Studio でジョブを編集します。変更を加えた場合は、[Actions] (アクション) ドロップダウンメニューから [Push to repository] (リポジトリにプッシュ) を選択して、ジョブをリポジトリに同期できます。

AWS Glue Studio ノートブックによるコードの作成

データエンジニアは、AWS Glue Studio のインタラクティブノートブックインターフェイス、または AWS Glue のインタラクティブセッションを使用して、従来よりも迅速かつ簡単に AWS Glue ジョブを作成できます。

制約事項

- AWS Glue Studio ノートブックは Scala をサポートしていません。

トピック

- ノートブックの使用の概要
- AWS Glue Studio の中のノートブックを使用した ETL ジョブの作成
- ノートブックエディタコンポーネント
- ノートブックとジョブスクリプトの保存
- ノートブックセッションの管理
- AWS Glue Studio ノートブックでの Amazon Q Developer の使用

ノートブックの使用の概要

AWS Glue Studio は、Jupyter Notebooks に基づいて、ノートブックインターフェイス中でジョブをインタラクティブにオーサリングできるようにします。AWS Glue Studio 中のノートブックを通じて、完全なジョブを実行せずにジョブスクリプトを編集して出力を表示できます。また、完全なジョブを実行せずにデータ統合コードを編集して出力を表示できます。また、マークダウンを追加してノートブックを [.ipynb] ファイルとジョブスクリプトとして保存できます。ソフトウェアをローカルにインストールしたり、サーバーを管理したりすることなく、ノートブックを起動できます。コードに満足したら、ボタンをクリックするだけで、AWS Glue Studio はノートブックを Glue ジョブに変換できます。

ノートブックを使用する利点のいくつかを以下に示します：

- プロビジョニングまたは管理するクラスターがない
- 支払うべきアイドルクラスターがない
- 事前設定は不要
- Jupyter Notebook のインストールは不要

- AWS Glue の ETL と同じランタイム/プラットフォーム

AWS Glue Studio を通してスルーノートブックを起動すると、ほんの数秒後にデータを調査し、ジョブスクリプトの開発を開始できるように、すべての設定手順が実行されます。AWS Glue Studio は、AWS Glue Jupyter カーネルによって Jupyter Notebook を設定します。このノートブックを使用するために VPC、ネットワーク接続、または開発エンドポイントを構成する必要はありません。

ノートブックインターフェイスを使用してジョブを作成するには、次の手順を実行します:

- 必要な IAM 許可を設定します。
- ノートブックセッションを開始してジョブを作成します
- ノートブックのセルにコードを記述します。
- コードを実行してテストし、出力を表示します。
- ジョブを保存します。

ノートブックを保存すると、ノートブックは満杯の AWS Glue のジョブ です。スケジュールジョブの実行、の設定、ジョブパラメータの設定、およびノートブックのすぐ横にあるジョブ実行履歴の表示など、ジョブのすべての側面を管理できます。

AWS Glue Studio の中のノートブックを使用した ETL ジョブの作成

AWS Glue Studio のコンソール中でノートブックの使用を開始するには

1. AWS Identity and Access Management のポリシーを AWS Glue Studio ユーザーにアタッチし、ETL ジョブとノートブックの IAM ロールを作成します。
2. IAM ロールに対するアクセス許可の付与 の説明に従って、ノートブックに追加の IAM セキュリティを設定します。
3. https://console.aws.amazon.com/gluestudio/ で AWS Glue Studio コンソールを開きます。

> **Note**
>
> ブラウザがサードパーティの Cookie をブロックしていないことを確認してください。ブラウザのデフォルトまたはユーザーの設定でサードパーティのCookieをブロックしている場合、ノートブックは起動しません。Cookie の管理の詳細については、以下を参照してください。

- Chrome
- Firefox
- Safari

4. ナビゲーションメニューの左側の [ジョブ] リンクを選択します。
5. [Jupyter notebook] を選択し、[Create] をクリックして、新しいノートブックセッションを開始します。
6. [Create job in Jupyter notebook] (Jupyter Notebook でジョブを作成する) ページで、ジョブ名を指定して、使用する IAM ロールを選択します。[ジョブの作成]を選択します。

   短時間経過後に、ノートブックエディタが表示されます。
7. コードを追加したら、セルを実行してセッションを開始します。セルを実行するには複数の方法があります。

   - 再生ボタンを押します。
   - キーボードショートカットを使用します。
     - MacOS では、Command + Enter キーでセルを実行します。
     - Windows では、Shift + Enter キーでセルを実行します。

   Jupyter ノートブックインターフェースを使用したコードの記述については、「The Jupyter Notebook User Documentation」を参照してください。
8. スクリプトをテストするには、スクリプト全体、または個々のセルを実行します。コマンド出力は、セルの下の領域に表示されます。
9. スクリプトの開発が完了したら、ジョブを保存して実行できます。スクリプトは、[Script] (スクリプト) タブにあります。ノートブックに追加したマジックはすべて取り除かれ、生成された AWS Glue ジョブのスクリプトの一部として保存されません。AWS Glue Studioは、ノートブックの内容から生成されたスクリプトの最後に `job.commit()` を自動的に追加します。

   ジョブの実行のさらなる詳細については、「ジョブの実行の開始」を参照してください。

ノートブックエディタコンポーネント

ノートブックエディタのインターフェイスには、次の主要なセクションがあります。

- ノートブックインターフェイス (メインパネル) とツールバー
- Job 編集タブ

ノートブックエディタ

AWS Glue Studio のノートブックエディタは Jupyter Notebook Application に基づいています。AWS Glue Studio のノートブックのインターフェースは、Jupyter Notebook が提供するものと似ています。これについては、セクション [Notebook user interface] で説明されしています。インタラクティブセッションにより使用されるノートブックは Jupyter Notebook です。

AWS Glue Studio のノートブックは Juptyer Notebooks に似ていますが、いくつかの重要な点で異なります。

- 現在、AWS Glue Studio のノートブックには拡張をインストールできません。
- 複数のタブを使用することはできません。ジョブとノートブックの間には 1:1 の関係があります
- AWS Glue Studio のノートブックには、Jupyter Notebooks 中に存在する、同じトップファイルメニューがありません。
- 現在、AWS Glue Studio のノートブックでは、AWS Glue のカーネルを用いてのみ動作します。カーネルは、ご自分では更新できないことに注意してください。

AWS Glue Studio のジョブ編集タブ

ETL ジョブを操作するために使用するタブは、ノートブックページの上部にあります。これらは、AWS Glue Studio のビジュアルジョブエディタに表示されるタブに似ていて、同じアクションを実行します。

- Notebook — このタブを使用して、ノートブックインターフェイスを用いてジョブスクリプトを表示します。
- Job details — ジョブ実行の環境とプロパティを設定します。
- Runs — このジョブの以前の実行に関する情報を表示します。
- Schedules — 特定の時間にジョブを実行するためのスケジュールを設定します。

ノートブックとジョブスクリプトの保存

ノートブックと作成中のジョブスクリプトは、いつでも保存できます。右上隅の [Save] ボタンを単に選択し、ビジュアルエディタまたはスクリプトエディタを使用しているかのように同じものを選択してください。

[Save] (保存) を選択した場合、ノートブックファイルはデフォルトの場所に保存されます。

- デフォルトでは、ジョブスクリプトは、[Job Details] (ジョブの詳細) タブの [Advanced properties] (詳細プロパティ) における、[Job details] (ジョブの詳細) プロパティの [Script path] (スクリプトパス) に示される Amazon S3 ロケーションに保存されます。ジョブスクリプトは、`Scripts` というサブフォルダに保存されます。
- デフォルトでは、ノートブックファイル (`.ipynb`) は、[Job Details] (ジョブの詳細) タブの [Advanced properties] (詳細プロパティ) における、[Job details] (ジョブの詳細) の [Script path] (スクリプトパス) に示される Amazon S3 ロケーションに保存されます。ノートブックファイルは、`Notebooks` というサブフォルダに保存されます。

> **Note**
>
> ジョブを保存すると、ジョブスクリプトにはノートブックのコードセルのみが含まれます。Markdown セルとマジックはジョブスクリプトに含まれません。ただし、`.ipynb` ファイルには Markdown とマジックが含まれます。

ジョブを保存したら、ノートブックで作成したスクリプトを使用してジョブを実行できます。

ノートブックセッションの管理

AWS Glue Studio 中のノートブックは、AWS Glue のインタラクティブセッション機能に基づいています。インタラクティブセッションの使用にはコストがかかります。コストの管理に役立つように、アカウント用に作成されたセッションをモニタリングし、すべてのセッションのデフォルト設定を構成できます。

すべてのノートブックセッションのデフォルトのタイムアウトを変更します。

デフォルトでは、プロビジョニングされた AWS Glue Studio ノートブックの起動の後に実行されたセルがない場合、そのノートブックは 12 時間後にタイムアウトします。これに関連して料金が発生することはありません。また、このタイムアウトは設定できません。

セルを実行することにより、インタラクティブなセッションが開始されます。このセッションのデフォルトのタイムアウトは 48 時間です。このタイムアウトは、セルを実行する前に `%idle_timeout` マジックを渡すことで設定が可能です。

AWS Glue Studio の中のノートブックのデフォルトのセッションタイムアウトを変更するには

1. ノートブックに、セル内に `%idle_timeout` のマジックを入力し、タイムアウト値を分単位で指定します。
2. 例: `%idle_timeout 15` はデフォルトのタイムアウトを 15 分に変更します。15 分以内にセッションを使用しない場合、セッションは自動的に停止します。

追加 Python モジュールのインストール

[pip] を使用してセッションに追加のモジュールをインストールする場合は、`%additional_python_modules` を使用してセッションに追加します。

```
%additional_python_modules awswrangler, s3://amzn-s3-demo-bucket/mymodule.whl
```

[additional_python_modules] へのすべての引数が `pip3 install -m <>` に渡されます。

利用可能な Python モジュールのリストについては、「AWS Glue での Python ライブラリの使用」を参照してください。

AWS Glue の設定の変更

マジックを使ってAWS Glue ジョブの設定値を管理できます 。ジョブ設定値を変更する場合は、ノートブックで適切なマジックを使用する必要があります。「Magics supported by AWS Glue interactive sessions for Jupyter」を参照してください。

> **Note**
>
> 実行中のセッションのプロパティを上書きできなくなりました。セッションの設定を変更する場合は、セッションを停止し、新しい構成を設定してから、新しいセッションを開始します。

AWS Glue はさまざまなワーカータイプをサポートしています。ワーカータイプは、`%worker_type` を用いて設定できます。例: `%worker_type G.2X` 。デフォルトは G.1X です。

`%number_of_workers` を使用してワーカー数を指定することもできます。例えば、40 人のワーカーを指定するには: `%number_of_workers 40`。

さらなる詳細については、「Defining Job Properties」を参照してください。

ノートブックセッションを停止します。

ノートブックセッションを停止するには、マジックの `%stop_session` を使用します。

AWS コンソールでノートブックから離れると、セッションの停止を選択できる警告メッセージが表示されます。

AWS Glue Studio ノートブックでの Amazon Q Developer の使用

AWS Glue Studio は、Jupyter Notebooks に基づいて、ノートブックインターフェイス中でジョブをインタラクティブにオーサリングできるようにします。Amazon Q Developer を使用すると、AWS Glue Studio ノートブック内でのオーサリング体験が向上します。

Amazon Q Developer の拡張機能は推奨コードを生成したり、コードの問題に関連する改善を提案したりすることにより、コードの記述をサポートします。Amazon Q Developer は、Python と Scala の両方をサポートしています。この 2 つの言語は、AWS Glue Studio ノートブックの Spark ジョブの ETL スクリプトをコーディングするために使用されます。

Amazon Q Developer とは?

Amazon Q Developer は機械学習を活用したサービスであり、デベロッパーの生産性の向上に貢献します。Amazon Q Developer は、自然言語によるデベロッパーのコメントおよび IDE のコードに基づいて推奨コードを生成することによって実現します。このサービスは、JupyterLab、Amazon SageMaker AI Studio、Amazon SageMaker AI ノートブックインスタンス、およびその他の統合開発環境 (IDE) と統合されています。

詳細については、「AWS Glue Studio を備えた Amazon Q Developer の使用」を参照してください。

コンソール上の AWS Glue ジョブ実行のステータス

AWS Glue 抽出、変換、ロード (ETL) ジョブのステータスは、実行中または停止後に表示できます。AWS Glue コンソールを使用してステータスを表示できます。ジョブ実行のステータスの詳細については、「the section called “ジョブ実行ステータス”」を参照してください。

ジョブモニタリングダッシュボードにアクセスする

ジョブモニタリングダッシュボードにアクセスするには、[ETL ジョブ] の下の AWS Glue ナビゲーションペインで [ジョブ実行のモニタリング] を選択します。

ジョブモニタリングダッシュボードの概要

ジョブモニタリングダッシュボードでは、ジョブの実行に関する全体的な概要と、ステータスが [Running] (実行中)、[Canceled] (キャンセル済み)、[Success] (成功)、または [Failed] (失敗) となっているジョブの合計を表示します。追加のタイルには、ジョブの実行の全体的な成功率、ジョブの DPU 使用率、ジョブタイプ、ワーカータイプ、日別のジョブステータスの内訳が表示されます。

タイル内のグラフはインタラクティブです。グラフ内の任意のブロックを選択して、ページの下部にある [Job runs] (ジョブの実行) テーブルで、それらのジョブのみを表示するフィルターを実行します。

[Date range] (日付範囲) セレクタを使用して、このページに表示される情報の日付範囲を変更できます。日付範囲を変更すると、現在の日付より前の指定した日数の値を表示するよう情報のタイルが調整されます。また、[Date range] (日付範囲) セレクタから [Custom] (カスタム) を選択して、特定の日付範囲を使用することもできます。

ジョブの実行ビュー

> **Note**
>
> ジョブ実行履歴には、ワークフローとジョブ実行のために 90 日間アクセスできます。

[Job runs] (ジョブの実行) リソースリストには、指定した日付範囲とフィルターのジョブが表示されます。

ステータス、ワーカータイプ、ジョブタイプ、ジョブ名などの追加の基準でジョブをフィルタリングできます。テーブルの上部にあるフィルターボックスに、フィルターとして使用するテキストを入力できます。テキストを入力すると、一致するテキストを含む行でテーブルの結果が更新されます。

ジョブモニタリングダッシュボードのグラフから要素を選択すると、ジョブのサブセットを表示できます。例えば、[Job runs summary] (ジョブの実行のサマリー) タイルで実行中のジョブの数を選択する場合、[Job runs] (ジョブの実行) リストには、その時点で Running のステータスにあるジョブのみが表示されます。[Worker type breakdown] (ワーカータイプの内訳) の棒グラフでいずれかのバーを選択する場合、ワーカータイプとステータスが一致するジョブの実行のみが [Job runs] (ジョブの実行) リストに表示されます。

[Job runs] (ジョブの実行) リソースリストには、ジョブの実行の詳細が表示されます。列見出しを選択して、テーブル内の行を並べ替えることができます。テーブルには次の情報が含まれます。

プロパティ	説明
ジョブ名	ジョブの名前。
タイプ	ジョブの環境のタイプ • Glue ETL: AWS Glue が管理する Apache Spark 環境で実行します。 • Glue Streaming: Apache Spark 環境で実行し、データストリームで ETL を実行します。 • Python シェル: Python スクリプトをシェルとして実行します。
開始時間	このジョブ実行が開始された日付と時刻。
終了時間	このジョブ実行が完了した日付と時刻。
実行ステータス	現在のジョブ実行の状態。値は次のようになります。 • STARTING • RUNNING • STOPPING • STOPPED • SUCCEEDED • FAILED

プロパティ	説明
	• `TIMEOUT`
実行時間	ジョブの実行でリソースを消費した時間。
容量	このジョブの実行に割り当てられる AWS Glue データ処理ユニット (DPU) の数。容量計画の詳細については、AWS Glue デベロッパーガイドの「Monitoring for DPU Capacity Planning」を参照してください。

プロパティ	説明
ワーカータイプ	ジョブの実行時に割り当てられた事前定義済みのワーカーのタイプ。値は G.1X、G.2X、G.4X または G.8X になります。 • **G.1X** – このタイプを選択する場合は、[Number of workers] (ワーカー数) の値も指定します。各ワーカーは、84 GB のディスク (約 34 GB の空き) を備えた 1 DPU (4 vCPU、16 GB のメモリ) にマッピングされます。メモリを大量に消費するジョブには、このワーカータイプをお勧めします。これが AWS Glue バージョン 2.0 以降のジョブの [Worker type] (ワーカータイプ) のデフォルトです。 • **G.2X** – このタイプを選択する場合は、[Number of workers] (ワーカー数) の値も指定します。各ワーカーは、128 GB のディスク (約 77 GB の空き) を備えた 2 DPU (8 vCPU、32 GB のメモリ) にマッピングされます。メモリを大量に消費するジョブには、機械学習変換を実行するこのワーカータイプをお勧めします。 • **G.4X** – このタイプを選択する場合は、[Number of workers] (ワーカー数) の値も指定します。各ワーカーは、256 GB のディスク (約 235 GB の空き) を備えた 4 DPU (16 vCPU、64 GB のメモリ) にマッピングされます。ワークロードに含まれる変換、集約、結合、クエリへの要求が非常に厳しいジョブには、このワーカータイプをお勧めします。このワーカータイプは、AWS Glue バージョン 3.0 以降の Spark ETL ジョブでのみ使用可能で、以下の AWS リージョンで使用でき

プロパティ	説明
	ます。米国東部 (オハイオ)、米国東部 (バージニア北部)、米国西部 (オレゴン)、アジアパシフィック (シンガポール)、アジアパシフィック (シドニー)、アジアパシフィック (東京)、カナダ (中部)、欧州 (フランクフルト)、欧州 (アイルランド)、欧州 (ストックホルム)。 • **G.8X** – このタイプを選択する場合は、[Number of workers] (ワーカー数) の値も指定します。各ワーカーは、512 GB のディスク (約 487 GB の空き) を備えた 8 DPU (32 vCPU、128 GB のメモリ) にマッピングされます。ワークロードに含まれる変換、集約、結合、クエリへの要求が非常に厳しいジョブには、このワーカータイプをお勧めします。このワーカータイプは、AWS Glue バージョン 3.0 以降の Spark ETL ジョブでのみ使用可能で、G.4X ワーカータイプでサポートされているのと同じ AWS リージョンで使用できます。
DPU 時間	ジョブの実行に使用される DPU の推定数。DPU は、処理能力の相対的な尺度です。DPU は、ジョブの実行のコストを割り出すために使用されます。詳細については、AWS Glue 料金表ページを参照してください。

リストから任意のジョブの実行を選択し、追加の情報を表示できます。ジョブの実行を選択して、次のいずれかを実行します。

- [Actions] (アクション) メニューから [View job] (ジョブの表示) オプションを選択して、ビジュアルエディタでジョブを表示します。

- [Actions] (アクション) メニューから [Stop run] (実行の停止) オプションを使用して、現在行われているジョブ実行を停止します。
- [View CloudWatch logs] (CloudWatch ログを表示) ボタンをクリックして、そのジョブのジョブの実行ログを表示します。
- [詳細を表示] を選択して、ジョブの実行の詳細ページを表示します。

ジョブの実行ログの表示

ジョブのログは、さまざまな方法で表示できます。

- [Monitoring] (モニタリング) ページの [Job runs] (ジョブの実行) テーブルで、ジョブの実行を選択し、[View CloudWatch logs] (CloudWatch ログを表示) を選択します。
- ビジュアルジョブエディタの、ジョブの [Runs] (実行) タブで、ハイパーリンクを選択してログを表示できます。
  - Logs (ログ) – ジョブの実行で継続的なログ記録が有効であるときに書き込まれる Apache Spark ジョブのログへのリンクです。このリンクを選択すると、`/aws-glue/jobs/logs-v2` ロググループ 内の Amazon CloudWatch ログに移動します。デフォルトでは、不要な Apache Hadoop YARN ハートビート、Apache Spark ドライバー、エグゼキューターログメッセージはログから除外されています。継続的なログ記録の詳細については、AWS Glue デベロッパーガイドの「Continuous Logging for AWS Glue Jobs」を参照してください。
  - Error logs (エラーログ) – このジョブの実行で `stderr` に書き込まれるログへのリンクです。このリンクを選択すると、`/aws-glue/jobs/error` ロググループ 内の Amazon CloudWatch ログに移動します。これらのログを使用して、ジョブの実行中に発生したエラーに関する詳細を表示できます。
  - Output logs (出力ログ) – このジョブの実行で `stdout` に書き込まれるログへのリンクです。このリンクを選択すると、`/aws-glue/jobs/output` ロググループ 内の Amazon CloudWatch ログに移動します。これらのログを使用して、AWS Glue Data Catalog で作成されたテーブルに関する詳細と、発生したエラーをすべて確認することができます。

ジョブの実行の詳細を表示する

[Monitoring] (モニタリング) ページの [Job runs] (ジョブの実行) リストでジョブを選択し、[View run details] (実行の詳細を表示する) をクリックして、ジョブの実行に関する詳細情報を表示できます。

ジョブの実行の詳細ページには、次の情報が表示されます。

プロパティ	説明
ジョブ名	ジョブの名前。
実行ステータス	現在のジョブ実行の状態。値は次のようになります。 • STARTING • RUNNING • STOPPING • STOPPED • SUCCEEDED • FAILED • TIMEOUT
Glue バージョン	ジョブ実行に使用される AWS Glue バージョン
最近の試行	ジョブ実行時の自動再試行回数
開始時間	このジョブ実行が開始された日付と時刻。
終了時間	このジョブ実行が完了した日付と時刻。
起動時間	ジョブ実行の準備にかかった時間
実行時間	ジョブスクリプトの実行にかかった時間
トリガー名	ジョブに関連付けられているトリガーの名前
最終更新日	ジョブが最後に変更された日付
セキュリティ設定	ジョブのセキュリティ設定。これには、Amazon S3 の暗号化、CloudWatch の暗号化、ジョブブックマークの暗号化設定が含まれます。
タイムアウト	ジョブ実行のタイムアウトのしきい値

プロパティ	説明
割り当てられた容量	このジョブの実行に割り当てられる AWS Glue データ処理ユニット (DPU) の数。容量計画の詳細については、AWS Glue デベロッパーガイドの「Monitoring for DPU Capacity Planning」を参照してください。
最大容量	ジョブの実行に使用可能な最大の容量。
ワーカー数	ジョブ実行に使用されるワーカーの数
ワーカータイプ	ジョブの実行に割り当てられる定義済みのワーカータイプ。値は、`G.1X` または `G.2X` になります。 • **`G.1X`** – このタイプを選択する場合は、[Number of workers] (ワーカー数) の値も指定します。各ワーカーは 1 DPU (4 vCPU、16 GB のメモリ、64 GB のディスク) にマッピングされており、ワーカーごとに 1 個のエグゼキューターがあります。メモリを大量に消費するジョブには、このワーカータイプをお勧めします。これが AWS Glue バージョン 2.0 以降のジョブの [Worker type] (ワーカータイプ) のデフォルトです。 • **`G.2X`** – このタイプを選択する場合は、[Number of workers] (ワーカー数) の値も指定します。各ワーカーは 2 DPU (8 vCPU、32 GB のメモリ、128 GB のディスク) にマッピングされており、ワーカーごとに 1 個のエグゼキューターがあります。メモリを大量に消費するジョブには、機械学習変換を実行するこのワーカータイプをお勧めします。

プロパティ	説明
ログ	継続的なログ記録のためのジョブログへのリンク (/aws-glue/jobs/logs-v2)。
出力ログ	ジョブの出力ログファイルへのリンク (/aws-glue/jobs/output)。
エラーログ	ジョブのエラーログファイルへのリンク (/aws-glue/jobs/error)。

次の追加項目も表示できます。これらは、最近のジョブ実行の情報を表示する際に表示できます。詳細については、「the section called “最近のジョブの実行の情報を表示する”」を参照してください。

- [入力引数]
- 連続ログ
- [メトリクス] – 基本的なメトリクスのビジュアライゼーションを確認できます。含まれるメトリクスの詳細については、「the section called “Spark ジョブ実行の Amazon CloudWatch メトリクスを表示”」を参照してください。
- [Spark UI] – Spark UI でジョブの Spark ログを視覚化できます。Spark Web UI の使用の詳細については、「the section called “Spark UI を使用したモニタリング”」を参照してください。the section called “ジョブ用の Spark UI の有効化” のステップに従って、この機能を有効にします。

Spark ジョブ実行の Amazon CloudWatch メトリクスを表示

ジョブの実行の詳細ページの [Run details] (実行の詳細) セクションでは、ジョブのメトリクスを表示できます。AWS Glue Studio はジョブ実行のたびにジョブメトリクスを Amazon CloudWatch に送信します。

AWS Glue では、30 秒ごとに Amazon CloudWatch にメトリクスが報告されます。AWS Glue メトリクスは、以前に報告された値からデルタ値を表示します。必要に応じて、メトリクスのダッシュボードにより 30 秒の値が集計 (合計) され、直近 1 分間分の値を取得できます。一方、AWS Glue によって Amazon CloudWatch に渡される Apache Spark メトリクスは、一般的に報告された時点での状態を表す絶対値です。

> **Note**
>
> Amazon CloudWatch にアクセスできるようアカウントを設定する必要があります。

メトリクスには、次のようなジョブの実行に関する情報が表示されます。

- ETL Data Movement (ETL データ移動) – Amazon S3 から読み取られたバイト数またはそこに書き込まれたバイト数。
- Memory Profile: Heap used (プロファイル: ヒープ使用) – Java 仮想マシン (JVM) ヒープによって使用されたメモリのバイト数。
- Memory Profile: heap usage (メモリプロファイル: ヒープ使用量) – JVM ヒープによって使用されたメモリの割合 (スケール: 0 ~ 1)。百分率で示されます。
- CPU Load (CPU 負荷) – 使用された CPU システムの負荷の割合 (スケール: 0 ~ 1)。百分率で示されます。

Ray ジョブ実行の Amazon CloudWatch メトリクスを表示

ジョブの実行の詳細ページの [Run details] (実行の詳細) セクションでは、ジョブのメトリクスを表示できます。AWS Glue Studio はジョブ実行のたびにジョブメトリクスを Amazon CloudWatch に送信します。

AWS Glue では、30 秒ごとに Amazon CloudWatch にメトリクスが報告されます。AWS Glue メトリクスは、以前に報告された値からデルタ値を表示します。必要に応じて、メトリクスのダッシュボードにより 30 秒の値が集計 (合計) され、直近 1 分間分の値を取得できます。一方、AWS Glue によって Amazon CloudWatch に渡される Apache Spark メトリクスは、一般的に報告された時点での状態を表す絶対値です。

> **Note**
>
> 次に示されているように、Amazon CloudWatch にアクセスできるようアカウントを設定する必要があります。

Ray ジョブでは、次の集約メトリクスグラフを表示できます。これらを使用すると、クラスターとタスクのプロファイルを作成し、各ノードの詳細情報にアクセスできます。これらのグラフを裏付ける時系列データは、CloudWatch でさらに分析できます。

タスクプロファイル: タスクの状態

システム内の Ray タスクの数を表示します。各タスクのライフサイクルには、独自の時系列が割り当てられます。

タスクプロファイル: タスク名

システム内の Ray タスクの数を表示します。保留中のタスクとアクティブなタスクのみが表示されます。タスクのタイプ (名前別) ごとに独自の時系列が割り当てられます。

クラスタープロファイル: 使用中の CPU

使用中の CPU コアの数を表示します。ノードごとに独自の時系列が割り当てられます。ノードは IP アドレスで識別されます。IP アドレスは一時的なものであり、識別にのみ使用されます。

クラスタープロファイル: オブジェクトストアのメモリ使用量

Ray オブジェクトキャッシュによるメモリ使用量を表示します。各メモリの場所 (物理メモリ、ディスクへのキャッシュ、Amazon S3 への流出) には、独自の時系列が割り当てられます。オブジェクトストアは、クラスター内に存在するすべてのノードのデータストレージを管理します。詳細については、Ray ドキュメントの「Objects」を参照してください。

クラスタープロファイル: ノード数

クラスターにプロビジョニングされたノード数を表示します。

ノードの詳細: CPU 使用量

各ノードの CPU 使用率をパーセンテージで表示します。各シリーズは、ノード上に存在する全コアの CPU 使用率の合計パーセンテージを表示します。

ノードの詳細: メモリ使用量

各ノードのメモリ使用量を GB 単位で表示します。各シリーズは、Ray タスクと Plasma ストアプロセスを含む、ノード上のすべてのプロセス間で集約されたメモリを表示します。これには、ディスクに保存されたオブジェクトや Amazon S3 に流出したオブジェクトは反映されません。

ノードの詳細: ディスク使用量

各ノードのディスク使用量を GB 単位で表示します。

ノードの詳細: ディスク I/O 速度

各ノードのディスク I/O を KB/秒単位で表示します。

ノードの詳細: ネットワーク I/O スループット

各ノードのネットワーク I/O を KB/秒単位で表示します。

ノードの詳細: Ray コンポーネントによる CPU 使用量

CPU 使用量をコアの分数で表示します。各ノードの Ray コンポーネントごとに独自の時系列が割り当てられます。

ノードの詳細: Ray コンポーネントによるメモリ使用量

メモリ使用量を GiB 単位で表示します。各ノードの Ray コンポーネントごとに独自の時系列が割り当てられます。

機密データを検出して処理する

Detect PII transform は、データソース内の個人識別情報 (PII) を識別します。エンティティを選択し、データのスキャン方法、Detect PII transform によって識別されてきた PII エンティティで何を行うかを識別します。

Detect PII transform は、定義したエンティティ、または AWS によって事前定義されたエンティティを検出、マスク、削除する機能を提供します。これにより、コンプライアンスを高め、責任を軽減できます。例えば、個人を特定でき読み取りが可能な情報がデータ内に存在しないことを確認した上で、固定文字列の社会保障番号 (例: xxx-xx-xxxx)、電話番号、または住所をマスクすることが考えられます。

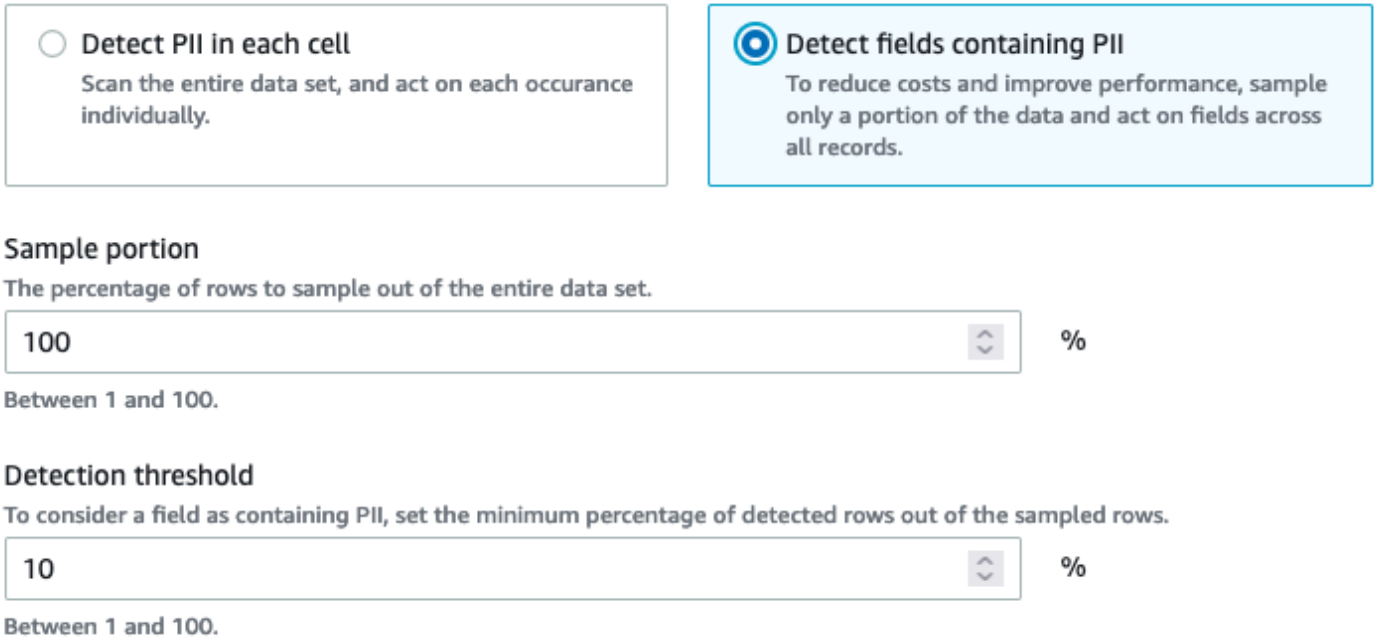
AWS Glue Studio の外部で機密データを使用するには、「AWS Glue Studio 外での機密データ検出の使用」を参照してください。

トピック

- データのスキャン方法の選択
- 検出する PII エンティティの選択
- 検出感度のレベルの指定
- 特定された PII データによる対処方法の選択
- 詳細なアクションオーバーライドの追加

データのスキャン方法の選択

データセットをスキャンして個人を特定できる情報 (PII) などの機密データを探す際に、各行で PII を検出するか、または PII データを含む列を検出するかを選択できます。



Detect PII in each cell を選択する場合、データソース内のすべての行をスキャンすることを選択しています。これは、PII エンティティを識別するための包括的なスキャンです。

Detect fields containing PII を選択する場合、PII エンティティの行のサンプルをスキャンすることを選択しています。これは、PII エンティティが見つかったフィールドを特定しながら、コストとリソースを低く抑える方法です。

PII を含むフィールドを検出することを選択した場合、行の一部をサンプリングすることで、コストを削減し、パフォーマンスを向上させることができます。このオプションを選択すると、追加のオプションを指定できます。

- Sample portion: これにより、サンプリングする行の割合を指定できます。例えば、50 と入力すると、PII エンティティのためにスキャンされた行の 50% を指定したいことになります。
- Detection threshold: これにより、列全体が PII エンティティを持つものとして識別されるように、PII エンティティを含む行の割合を指定できます。例えば、10 と入力した場合、フィールドに PII エンティティである米国電話機があると識別されるためには、スキャンされる行の PII エンティティの US Phone の数が 10% 以上になるように指定します。PII エンティティを含む行の割合が 10% 未満の場合、そのフィールドに PII エンティティ (US Phone) が含まれているというラベル付けできません。

検出する PII エンティティの選択

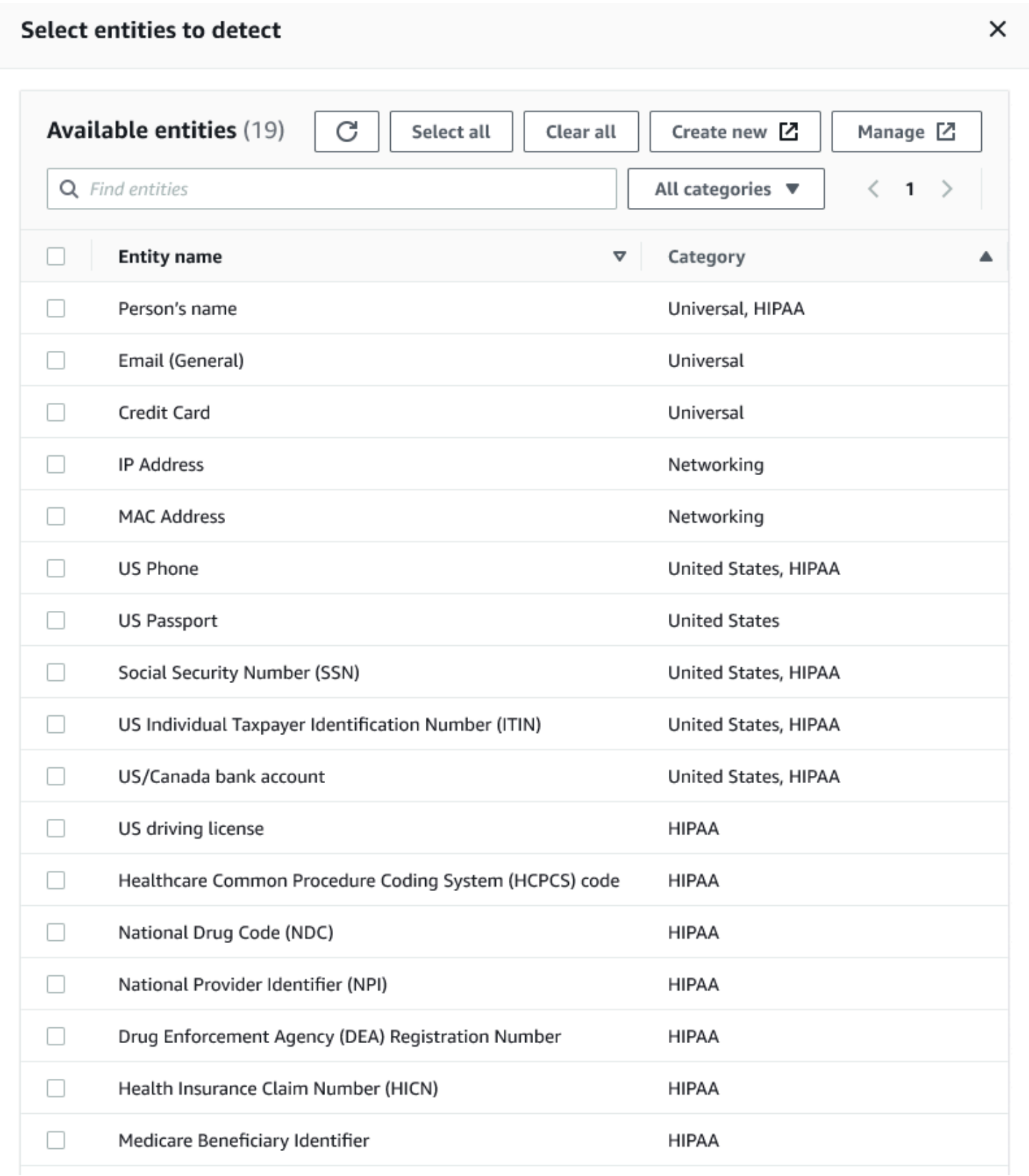
[Detect PII in each cell] (各セルの PII を検出する) を選択した場合は、次の 3 つのオプションのいずれかを選択できます。

- 利用可能なすべての PII パターン - これには AWS エンティティが含まれます。
- カテゴリを選択する - カテゴリを選択すると、PII パターンには、選択したカテゴリのパターンが自動的に含まれます。
- 特定のパターンを選択する - 選択したパターンのみが検出されます。

マネージド機密データタイプの全リストについては、「Managed data types」を参照してください。

利用可能なすべての PII パターンから選択する

[All available PII patterns] (使用可能なすべての PII パターン) を選択する場合は、AWS で事前定義されたエンティティを選択します。エンティティは、1 つでも、複数でも、すべてでも選択できます。


Select entities to detect
Available entities (19)
Select all
Clear all
Create new
Manage
Find entities
All categories
1
Entity name
Category
Person's name
Universal, HIPAA
Email (General)
Universal
Credit Card
Universal
IP Address
Networking
MAC Address
Networking
US Phone
United States, HIPAA
US Passport
United States
Social Security Number (SSN)
United States, HIPAA
US Individual Taxpayer Identification Number (ITIN)
United States, HIPAA
US/Canada bank account
United States, HIPAA
US driving license
HIPAA
Healthcare Common Procedure Coding System (HCPCS) code
HIPAA
National Drug Code (NDC)
HIPAA
National Provider Identifier (NPI)
HIPAA
Drug Enforcement Agency (DEA) Registration Number
HIPAA
Health Insurance Claim Number (HICN)
HIPAA
Medicare Beneficiary Identifier
HIPAA

カテゴリを選択する

PII パターンとして [Select categories] (カテゴリを選択する) を選択して検出する場合は、ドロップダウンメニューのオプションから選択します。一部のエンティティは複数のカテゴリに属する場合があることに注意してください。例えば、[Person's name] (人名) は、[Universal] (ユニバーサル) および [HIPAA] カテゴリに属するエンティティです。

- [Universal] (ユニバーサル) (例: [Email] (E メール)、[Credit Card] (クレジットカード))
- [HIPAA] (例: [US driving license] (米国の運転免許証)、[Healthcare Common Procedure Coding System (HCPCS) code] (Healthcare Common Procedure Coding System (HCPCS) コード))
- [Networking] (ネットワーキング) (例: [IP Address] (IP アドレス)、[MAC Address] (MAC アドレス))
- アルゼンチン
- オーストラリア
- オーストリア
- ベルギー
- ボスニア
- ブルガリア
- カナダ
- チリ
- コロンビア
- クロアチア
- キプロス
- チェコ共和国
- デンマーク
- エストニア
- フィンランド
- フランス
- ドイツ
- ギリシャ
- ハンガリー
- アイルランド

- 韓国
- 日本
- メキシコ
- オランダ
- ニュージーランド
- ノルウェー
- ポルトガル
- ルーマニア
- シンガポール
- スロバキア
- スロベニア
- スペイン
- スウェーデン
- スイス
- トルコ
- ウクライナ
- アメリカ
- 英国
- ベネズエラ

特定のパターンを選択する

検出する PII パターンとして [Select specific patterns] (特定のパターンを選択する) を選択すると、作成済みのパターンのリストから検索や参照したり、新しい検出エンティティパターンを作成したりできます。

次のステップでは、機密データを検出する新しいカスタムパターンを作成する方法について説明します。カスタムパターンの名前を入力して、カスタムパターンを作成し、正規表現を追加して、オプションでコンテキスト単語を定義します。

1. 新しいパターンを作成するには、[Create new] (新規作成) ボタンをクリックします。



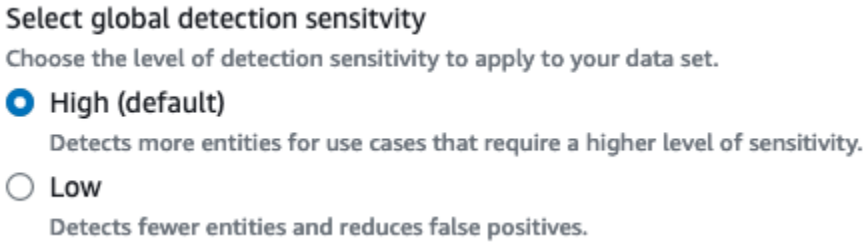
2. [Create detection entity] (検出エンティティの作成) ページで、エンティティ名と正規表現を入力します。正規表現 (Regex) は、AWS Glue がエンティティを照合するために使用するものです。
3. [Validate] (検証) をクリックします。検証が成功すると、文字列が有効な正規表現であることを示す確認メッセージが表示されます。検証に失敗した場合は、文字列が適切なフォーマット、および許容される文字リテラル、演算子、構文のいずれかに準拠していないことを示すメッセージが表示されます。
4. 正規表現にコンテキスト単語を追加することもできます。コンテキスト単語によって一致する可能性が高くなることがあります。コンテキスト単語は、フィールド名がエンティティを説明していない場合に有効です。例えば、社会保障番号には「SSN」または「SS」という名前が付けられます。これらのコンテキスト単語を追加すると、エンティティの照合に役立ちます。
5. [Create] (作成) をクリックして、検出エンティティを作成します。作成されたエンティティは、AWS Glue Studio コンソールに表示されます。左側のナビゲーションメニューの [Detection entities] (検出エンティティ) をクリックします。

   [Detection entities] (検出エンティティ) ページから、検出エンティティの編集、削除、作成ができます。検索フィールドを使用してパターンを検索することもできます。

検出感度のレベルの指定

機密データの検出を使用する場合の感度レベルを設定できます。

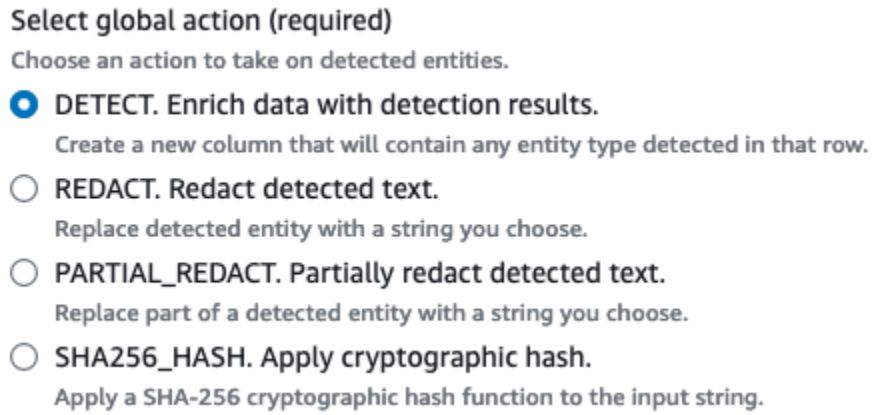
- [高] – (デフォルト) より高いレベルの感度が必要なユースケースのために、より多くのエンティティを検出します。2023 年 11 月よりも後に作成されたすべての AWS Glue ジョブは、この設定を自動的にオプトインします。
- [低] - 検出するエンティティの数を減らし、誤検知を減らします。



特定された PII データによる対処方法の選択

データソース全体で PII を検出することを選択した場合は、適用するグローバルアクションを選択できます。

- Enrich data with detection results: 各セルで Detect PII を選択した場合、検出されたエンティティを新しい列に保存できます。
- Redact detected text: 検出された PII 値を、オプションの置換テキスト入力フィールド中に指定した文字列に置き換えることができます。文字列を指定しない場合、検出された PII エンティティは '********' に指定されます。
- [検出されたテキストを部分的にマスキング]: 検出された PII の値の一部を、選択した文字列に置き換えることができます。可能なオプションは 2 つあります。すなわち、端をマスキングしないままにするか、または明示的な正規表現パターンを指定してマスキングするかのいずれかです。AWS Glue 2.0 では、この機能は使用できません。
- Apply cryptographic hash: 検出された PII 値を SHA-256 暗号化ハッシュ関数に渡し、その値を関数からの出力に置き換えることができます。



AWS Glue バージョン 2.0 と 3.0 以降の相違点

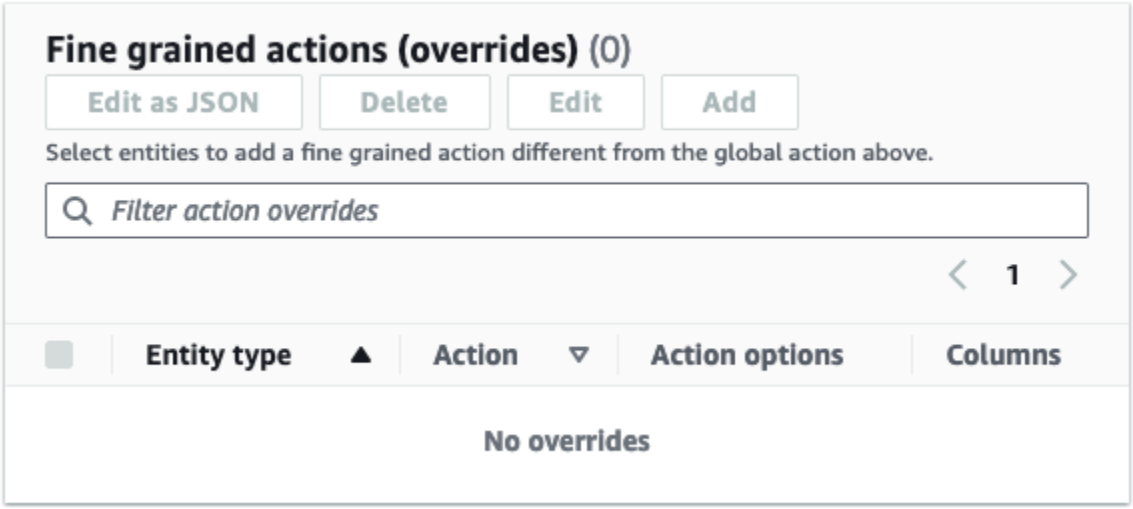
AWS Glue 2.0 ジョブは、補足列の各列について検出された PII 情報を含む新しい DataFrame を返します。マスキングまたはハッシュ作業は、ビジュアルタブの AWS Glue スクリプト内に表示されます。

AWS Glue 3.0 および 4.0 ジョブは、これと同じ補足列を持つ新しい DataFrame を返します。「actionUsed」の新しいキーが存在し、DETECT、REDACT、PARTIAL_REDACT、または SHA256_HASH のいずれかになります。マスキングアクションが選択されている場合、DataFrame は機密データをマスキングした状態でデータを返します。

詳細なアクションオーバーライドの追加

追加の検出およびアクションの設定を、詳細なアクションオーバーライドテーブルに追加できます。これにより、次のことが可能になります。

- [検出から特定の列を包含または除外] – データソース上の推論されたスキーマによって、使用可能な列がテーブルに入力されます。
- [グローバルアクションを使用するよりも詳細な特定の設定を指定] – 例えば、エンティティタイプごとに異なるマスキングテキストの設定を指定できます。
- [グローバルアクションとは異なるアクションを指定] – 異なる機密データタイプに異なるアクションを適用する場合は、ここで実行できます。2 つの異なるその場編集アクション (マスキングとハッシュ) は同じ列では使用できませんが、検出はいつでも使用できることに留意してください。



AWS Glue Studio による ETL ジョブの管理

AWS Glue Studio ではシンプルでグラフィカルなインタフェースを使用して、ETL ジョブを管理できます。ナビゲーションメニューで、[Jobs] (ジョブ) を選択し、[Jobs] (ジョブ) ページを表示します。このページには、AWS Glue Studio または AWS Glue コンソールで作成したすべてのジョブが表示されます。このページでは、ジョブを表示、管理、および実行できます。

またこのページでは、次のタスクを実行できます。

- ジョブの実行の開始
- ジョブの実行のスケジュール
- ジョブスケジュールの管理
- ジョブの実行の停止
- ジョブの表示
- 最近のジョブの実行の情報を表示する
- ジョブスクリプトの表示
- ジョブのプロパティを変更する
- ジョブの保存
- ジョブのクローンを作成する
- ジョブの削除

ジョブの実行の開始

AWS Glue Studio ではオンデマンドでジョブを実行できます。ジョブは複数回実行でき、ジョブを実行するたびに、AWS Glue によりジョブのアクティビティとパフォーマンスに関する情報が収集されます。この情報は [job run] (ジョブの実行) として参照され、ジョブの実行 ID によって識別されます。

AWS Glue Studio では次の方法でジョブの実行を開始できます。

- [Jobs] (ジョブ) ページで、開始するジョブを選択してから、[Run job] (ジョブの実行) ボタンをクリックします。
- ビジュアルエディタでジョブを表示していて、そのジョブが保存されている場合は、[Run] (実行) ボタンをクリックして、ジョブの実行を開始します。

ジョブの実行についての詳細は、AWS Glue デベロッパーガイドの「Working with Jobs on the AWS Glue Console」を参照してください。

ジョブの実行のスケジュール

AWS Glue Studio ではスケジュールを作成し、特定の時間にジョブを実行できます。ジョブが実行される回数、曜日、時間などの制約を指定できます。これらの制約は `cron` に基づいており、`cron` と同じ制限があります。例えば、毎月 31 日にジョブを実行することを選択した場合、31 日がない月があることに注意してください。`cron` の詳細については、AWS Glue デベロッパーガイドの「cron 式」を参照してください。

スケジュールに従ってジョブを実行するには

1. 次のいずれかの方法を使用して、ジョブのスケジュールを作成します。
   - [Jobs] (ジョブ) ページで、スケジュールを作成するジョブを選択し、[Actions] (アクション) を選択してから、[Schedule job] (ジョブのスケジュール) を設定します。
   - ビジュアル エディタでジョブを表示していて、そのジョブが保存されている場合は、[Schedules] (スケジュール) タブを選択します。次に [Create Schedule] (スケジュールの作成) を選択します。
2. [Schedule job run] (ジョブの実行のスケジュール) ページで、次の情報を入力します。
   - Name (名前): ジョブのスケジュールの名前を入力します。
   - Frequency (頻度): ジョブのスケジュールの頻度を入力します。以下のオプションを選択できます。
     - Hourly (毎時): ジョブは 1 時間ごとに実行され、特定の時間に始まります。ジョブを実行する時間の分を指定できます。デフォルトでは、[hourly] (毎時) を選択すると、ジョブは時間の初め (0 分) に実行されます。
     - Daily (毎日): ジョブは毎日実行され、一度に開始します。ジョブを実行する時間の分とジョブの開始時間を指定できます。時間は 23 時間制で指定され、午後の時間には 13 から 23 の数字を使用します。分と時間のデフォルトの値は 0 です。つまり、[Daily] (毎日) を選択している場合、ジョブはデフォルトで午前 0 時に実行されます。
     - Weekly (毎週): ジョブは毎週 1 日以上実行されます。前述の [Daily] (毎日) と同じ設定に加えて、ジョブを実行する曜日を選択できます。1 日以上選択できます。
     - Monthly (毎月): ジョブは毎月、特定の日に実行されます。前述の [Daily] (毎日) と同じ設定に加えて、その月のジョブを実行する日を選択できます。その日を 1 から 31 の数値で指定

します。2 月 30 日などの、1 か月で存在しない日を選択した場合、その月にジョブは実行されません。

- Custom (カスタム): cron 構文を使用して、ジョブのスケジュールの式を入力します。cron 式を使用すると、その月の特定の日ではなく最終日、または 3 か月ごとに 7 日と 21 日などのより複雑なスケジュールを作成できます。

  AWS Glue デベロッパーガイドの「cron 式」を参照してください。

- Description (説明): オプションで、ジョブのスケジュールについての説明を入力できます。複数のジョブに同じスケジュールを使用する場合、説明があることでジョブのスケジュールがどうなっているのかを簡単に確認できます。

3. [Create schedule] (スケジュールの作成) をクリックして、ジョブのスケジュールを保存します。
4. スケジュールを作成すると、コンソールページの上部に成功のメッセージが表示されます。このバナーの [Job details] (ジョブの詳細) をクリックして、ジョブの詳細を表示できます。これにより、[Schedules] (スケジュール) タブが選択されているビジュアルジョブエディタのページが開きます。

ジョブスケジュールの管理

ジョブのスケジュールを作成したら、ビジュアルエディタでジョブを開き、[Schedules] (スケジュール) タブをクリックして、スケジュールを管理できます。

[Schedules] (スケジュール) タブでは、次のタスクを実行できます。

- 新しいスケジュールの作成。

  [Create Schedule] (スケジュールの作成) を選択し、the section called “ジョブの実行のスケジュール” に示されているようにスケジュールの情報を入力します。

- 既存のスケジュールの編集。

  編集するスケジュールを選択して、[Action] (アクション)、[Edit schedule] (スケジュールの編集) を選択します。既存のスケジュールの編集を選択すると、[Frequency] (頻度) は [Custom] (カスタム)、スケジュールは cron 式と表示されます。cron 式を変更するか、[Frequency] (頻度) ボタンを使用して新しいスケジュールを指定できます。変更が完了したら、[Update schedule] (スケジュールの更新) を選択します。

- アクティブなスケジュールの一時停止。

アクティブなスケジュールを選択して、[Action] (アクション)、[Pause schedule] (スケジュールの一時停止) を選択します。スケジュールは、すぐに非アクティブ化されます。更新されたジョブスケジュールのステータスを表示するには、[refresh (reload)] (更新 (再ロード)) ボタンをクリックします。

- 一時停止したスケジュールの再開。

  非アクティブ化したスケジュールを選択して、[Action] (アクション)、[Resume schedule] (スケジュールの再開) を選択します。スケジュールはすぐにアクティブになります。更新されたジョブスケジュールのステータスを表示するには、[refresh (reload)] (更新 (再ロード)) ボタンをクリックします。

- スケジュールの削除。

  削除するスケジュールを選択して、[Action] (アクション)、[Delete schedule] (スケジュールの削除) を選択します。スケジュールはすぐに削除されます。更新したジョブスケジュールのリストを表示するには、[refresh (reload)] (更新 (再ロード)) ボタンを選択します。スケジュールには、完全に削除されるまで [Deleting] (削除中) のステータスが表示されます。

ジョブの実行の停止

ジョブの実行が完了する前に、ジョブを停止できます。ジョブが正しく設定されていないことがわかっている場合、またはジョブの完了に時間がかかりすぎる場合は、このオプションを選択できます。

[Monitoring] (モニタリング) ページの [Job runs] (ジョブの実行) リストで停止するジョブを選択し、[Action] (アクション) を選択してから、[Stop run] (実行の停止) をクリックします。

ジョブの表示

すべてのジョブは、[Jobs] (ジョブ) ページで確認できます。このページにアクセスするには、ナビゲーションペインで [Jobs] (ジョブ) を選択します。

[Jobs] (ジョブ) ページでは、アカウントで作成されたすべてのジョブを確認できます。[Your jobs] (ジョブ) リストには、ジョブ名、タイプ、ジョブの前回の実行のステータス、およびジョブが作成されて最後に変更された日付が表示されます。ジョブの名前を選択して、そのジョブの詳細情報を表示できます。

また、モニタリングダッシュボードを使用して、すべてのジョブを表示することもできます。ダッシュボードにアクセスするには、ナビゲーションペインで、[Monitoring] (モニタリング) を選択します。

ジョブの表示のカスタマイズ

[Jobs] (ジョブ) ページの [Your jobs] (ジョブ) セクションで、ジョブの表示方法をカスタマイズできます。また、検索テキストフィールドにテキストを入力して、そのテキストを含む名前のジョブのみを表示することもできます。

[Your jobs] (ジョブ) セクションの設定アイコン
( )
を選択すると、AWS Glue Studio でテーブルに情報を表示する方法をカスタママイズできます。表示内のテキストの行を折り返したり、ページに表示されるジョブの数を変更したり、表示する列を指定できます。

最近のジョブの実行の情報を表示する

新しいデータをソースの場所に追加する際、ジョブは複数回実行されます。ジョブを実行するたびに、一意の ID が割り当てられ、それに関する情報が収集されます。この情報は、次の方法で表示できます。

- [Runs] (実行) タブをクリックして、現在表示されているジョブの実行の情報を表示します。

  [Recent job runs] (最近のジョブの実行) ページの [Runs] (実行) タブ には、ジョブの実行ごとにカードがあります。[Runs] (実行) タブで表示される情報には、次の情報が含まれます。

  - ジョブの実行 ID
  - このジョブの実行の試行回数
  - ジョブの実行のステータス
  - ジョブの実行開始および終了時刻
  - ジョブの実行のランタイム
  - ジョブのログファイルへのリンク
  - ジョブのエラーログファイルへのリンク
  - 失敗したジョブに返されたエラー
- ジョブ実行を選択すると、以下のようにジョブに関する追加情報を表示できます。
  - [入力引数]

- 連続ログ
- [メトリクス] – 基本的なメトリクスのビジュアライゼーションを確認できます。含まれるメトリクスの詳細については、「the section called “Spark ジョブ実行の Amazon CloudWatch メトリクスを表示”」を参照してください。
- [Spark UI] – Spark UI でジョブの Spark ログを視覚化できます。Spark Web UI の使用の詳細については、「the section called “Spark UI を使用したモニタリング”」を参照してください。the section called “ジョブ用の Spark UI の有効化” のステップに従って、この機能を有効にします。

[詳細を表示] を選択すると、ジョブ実行の詳細ページに同じような情報が表示されます。あるいは、[モニタリング] ページからジョブ実行の詳細ページに移動することもできます。ナビゲーションペインで、[モニタリング] を選択します。[Job runs] (ジョブの実行) リストまで下にスクロールします。ジョブを選択し、[View run details] (実行の詳細を表示する) を選択します。内容は、ジョブの実行の詳細を表示する に記述されます。

このジョブのログについての詳細は、「ジョブの実行ログの表示」を参照してください。

ジョブスクリプトの表示

ジョブ内のすべてのノードの情報を入力した後、AWS Glue Studio により、スクリプトが生成されます。このスクリプトは、ジョブによるソースからのデータの読み取り、変換、ターゲットの場所への書き込みに使用されます。ジョブを保存すると、このスクリプトをいつでも表示できます。

ジョブ用に生成されたスクリプトを表示するには

1. ナビゲーションペインで [Jobs] (ジョブ) を選択します。
2. [Jobs] (ジョブ) ページの [Your Jobs] (ジョブ) リストで、レビューするジョブの名前を選択します。または、リストからジョブを選択し、[Actions] (アクション) メニューを選択して、[Edit job] (ジョブの編集) を選択することもできます。
3. ビジュアルエディタで、ページ上部の [Script] (スクリプト) タブを選択して、ジョブスクリプトを表示します。

   ジョブスクリプトを編集する場合は、「AWS Glue プログラミングガイド」を参照してください。

ジョブのプロパティを変更する

ジョブで実行されるアクションは、ジョブ図のノードにより定義されますが、お客様がジョブに設定できるプロパティもいくつかあります。これらのプロパティでは、ジョブが実行される環境、使用するリソース、しきい値の設定、セキュリティ設定などを決定できます。

ジョブの実行環境をカスタマイズするには

1. ナビゲーションペインで [Jobs] (ジョブ) を選択します。
2. [Jobs] (ジョブ) ページの [Your Jobs] (ジョブ) リストで、レビューするジョブの名前を選択します。
3. ビジュアルエディタのページで、ジョブ編集ペインの上部にある [Job details] (ジョブの詳細) タブを選択します。
4. 必要に応じて、ジョブのプロパティを変更します。

   ジョブのプロパティの詳細については、AWS Glue デベロッパーガイドの「Defining Job Properties」を参照してください。
5. 次のようなジョブのプロパティを指定する必要がある場合、[Advanced properties] (詳細プロパティ) セクションを展開します。

   - Script filename (スクリプトファイル名) – Amazon S3 でジョブスクリプトを保存するファイルの名前です。
   - Script path (スクリプトパス) – ジョブスクリプトが保存される Amazon S3 の場所です。
   - Job metrics (ジョブのメトリクス) – (Python シェルジョブでは使用不可) ジョブの実行時に Amazon CloudWatch メトリクスの作成をオンにします。
   - Continuous logging (継続的なログ記録) – (Python シェルジョブでは使用不可) CloudWatch への連続的なログ記録をオンにし、ジョブが完了する前にログを表示できるようにします。
   - Spark UI および Spark UI logs path (Spark UI ログパス) – (Python シェルジョブでは使用不可) このジョブをモニタリングするための Spark UI の使用をオンにし、Spark UI ログの場所を指定します。
   - Maximum concurrency (同時実行の最大数) – このジョブで許可される同時実行の最大数を設定します。
   - Temporary path (一時的なパス) – AWS Glue でジョブスクリプトが実行されるときに一時的な中間結果が書き込まれる Amazon S3 の作業ディレクトリの場所です。

- Delay notification threshold (minutes) (遅延通知のしきい値 (分)) – ジョブの遅延のしきい値を指定します。ジョブがしきい値で指定された時間よりも長い時間実行された場合、AWS Glue によりジョブの遅延通知が CloudWatch に送信されます。
- Security configuration (セキュリティ設定) および Server-side encryption (サーバー側の暗号化) – これらのフィールドを使用して、ジョブの暗号化オプションを選択します。
- Use Glue Data Catalog as the Hive metastore (Glue Data Catalog を Hive メタストアとして使用する) – Apache Hive Metastore の代わりに AWS Glue Data Catalog を使用する場合、このオプションを選択します。
- Additional network connection (追加のネットワーク接続) – VPC 内のデータソース向けに、タイプ `Network` の接続を指定して、ジョブが VPC 経由でデータにアクセスできるようにします。
- Python library path (Python ライブラリパス)、Dependent jars path (依存 JARS パス) (Python シェルジョブでは使用不可)、Referenced files path (参照されるファイルパス) – これらのフィールドを使用して、スクリプトの実行時にジョブが使用する追加のファイルの場所を指定できます。
- Job Parameters (ジョブパラメータ) – スクリプトに名前付きパラメータとして渡される一連のキーと値のペアを追加できます。AWS Glue API の Python 呼び出しでは、明示的に名前でパラメータを渡すことが最善です。ジョブスクリプトでのパラメータの使用の詳細については、AWS Glue デベロッパーガイドの「Passing and Accessing Python Parameters in AWS Glue」を参照してください。
- Tags (タグ) – ジョブにタグを追加することで、ジョブの整理および識別が容易になります。

6. ジョブのプロパティを変更した後、ジョブを保存します。

Spark のシャッフルファイルを Amazon S3 に保存する

一部の ETL ジョブでは、複数のパーティションから情報を読み込んで結合する必要があります。例えば、結合変換を使用する場合などです。この操作はシャッフリングと呼ばれます。シャッフル中、データはディスクに書き込まれ、ネットワーク経由で転送されます。AWS Glue バージョン 3.0 では、これらのファイルの保存場所として Amazon S3 を設定できます。AWS Glue では、Amazon S3 との間でシャッフルファイルの書き込みおよび読み込みを行うシャッフルマネージャーを利用できます。Amazon S3 からのシャッフルファイルの書き込みおよび読み込みは、ローカルディスク (または Amazon EC2 用に最適化された Amazon EBS) に比べて 5% ~ 20% 遅くなります。ただし、Amazon S3 には無制限のストレージ容量があるため、ジョブを実行する際に「`No space left on device`」エラーについて心配する必要はありません。

ファイルのシャッフルに Amazon S3 を使用するようにジョブを設定するには

1. [Jobs] (ジョブ) ページの [Your Jobs] (ジョブ) リストで、変更するジョブの名前を選択します。
2. ビジュアルエディタのページで、ジョブ編集ペインの上部にある [Job details] (ジョブの詳細) タブを選択します。

   [Job parameters] (ジョブパラメータ) セクションまで下にスクロールします。
3. 次のキーと値のペアを指定します。

   - `--write-shuffle-files-to-s3` — `true`

     これは、AWS Glue でシャッフルマネージャーを構成する主要なパラメータです。シャッフルデータの書き込みおよび読み取りには Amazon S3 バケットを使用します。デフォルトでは、このパラメータの値は `false` です。
   - (オプション) `--write-shuffle-spills-to-s3` — `true`

     このパラメータを使用すると、流出ファイルを Amazon S3 バケットにオフロードできます。これにより、AWS Glue での Spark ジョブの耐障害性が強化されます。これは、大量のデータをディスクに退避させる大規模なワークロードにのみ必要です。デフォルトでは、このパラメータの値は `false` です。
   - (オプション) `--conf spark.shuffle.glue.s3ShuffleBucket` — `S3://<shuffle-bucket>`

     このパラメータを使用すると、シャッフルファイルを書き込む際に使用する Amazon S3 バケットを指定できます。このパラメータを設定しない場合、その場所は一時パス (`--TempDir`) 用に指定された場所にある `shuffle-data` フォルダになります。

     > **Note**
     >
     > シャッフルバケットの場所がジョブが実行されるのと同じ AWS リージョン にあることを確認します。
     > また、ジョブの実行の終了後、ファイルはシャッフルサービスによりクリーンアップされないため、シャッフルバケットの場所で Amazon S3 ストレージライフサイクルポリシーを設定する必要があります。詳細については、Amazon S3 ユーザーガイドの「ストレージのライフサイクルの管理」をご参照ください。

ジョブの保存

ジョブが保存されるまで、[Save] (保存) ボタンの左側に、赤い [Job has not been saved] (ジョブは保存されていません) という吹き出しが表示されます。



ジョブを保存するには

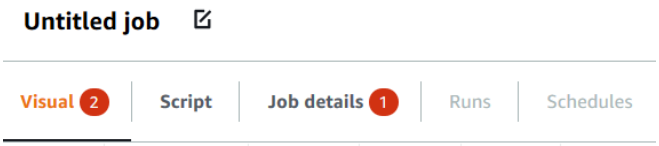
1. [Visual] (ビジュアル) および [Job details] (ジョブの詳細) タブで、必要なすべての情報を入力します。
2. [保存] ボタンを選択します。

   ジョブを保存すると、「保存されていません」という吹き出しが、ジョブが最後に保存された日時の表示に変わります。

ジョブを保存する前に AWS Glue Studio を終了した場合、次回の AWS Glue Studio の起動時に通知が表示されます。通知では、未保存のジョブがあり、それを復元するかどうかを尋ねられます。ジョブの復元を選択すると、そのジョブの編集を続行できます。

ジョブ保存時のエラーのトラブルシューティング

[Save] (保存) ボタンをクリックすると、ジョブに必要な情報が欠落している場合は、情報が欠落しているタブに赤い吹き出しが表示されます。吹き出しの数字は、欠落している検出されたフィールドの数を示します。

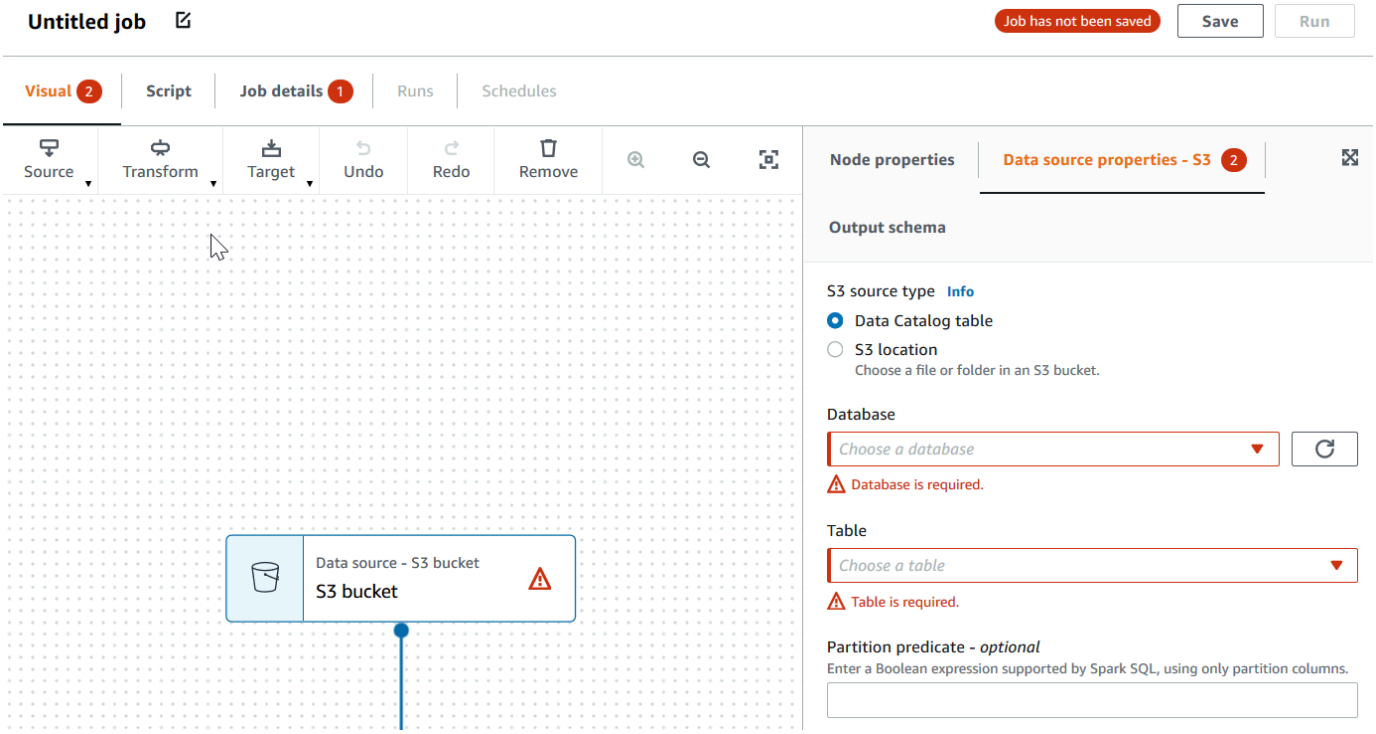


- ビジュアルエディタのノードが正しく設定されていない場合、[Visual] (ビジュアル) タブに赤い吹き出しが表示され、エラーのあるノードに警告記号

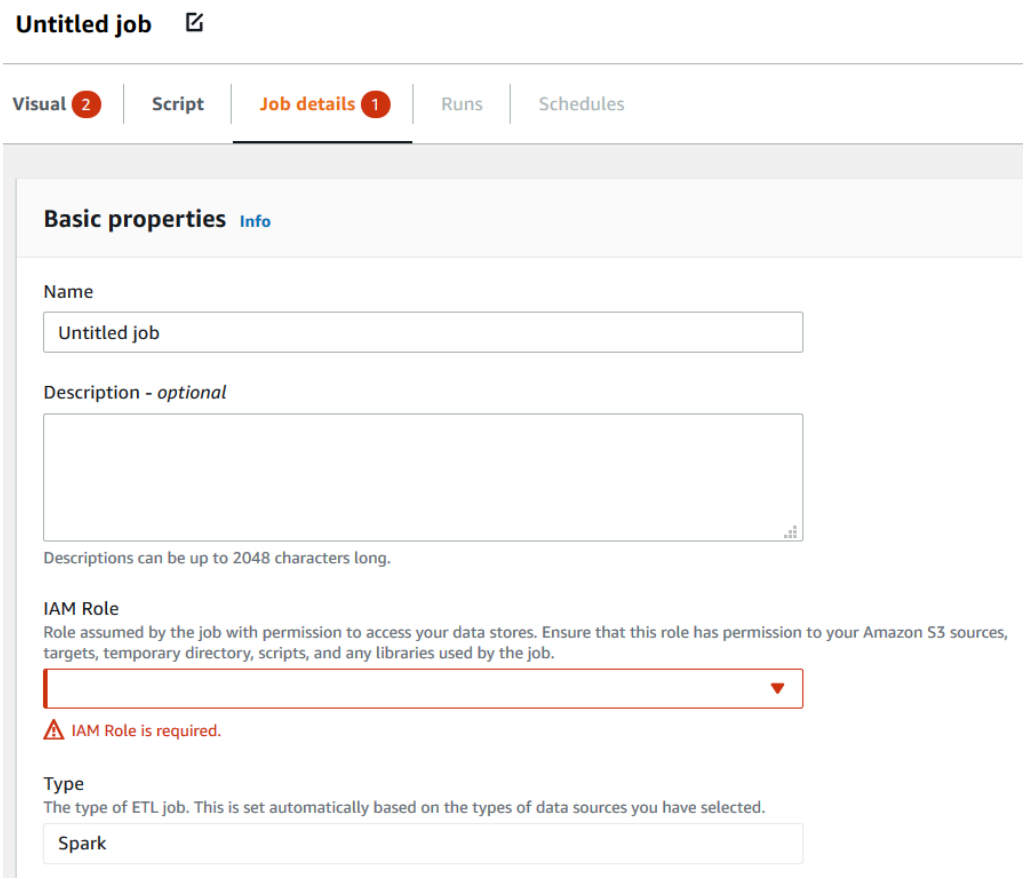
  が表示されます。

  1. ノードを選択します。ノードの詳細パネルで、不足または誤った情報があるタブに赤い吹き出しが表示されます。

2. ノードの詳細パネルで赤い吹き出しが表示されるタブを選択し、ハイライトされている問題のフィールドを見つけます。フィールドの下のエラーメッセージには、問題に関する追加情報が表示されます。



- ジョブのプロパティに問題がある場合、[Job details] (ジョブの詳細) タブには、赤い吹き出しが表示されます。そのタブを選択し、ハイライトされている問題のフィールドを見つけます。フィールドの下のエラーメッセージにより、問題に関する追加情報が表示されます。



ジョブのクローンを作成する

[Clone job] (ジョブのクローン) アクションを使用して、既存のジョブを新しいジョブにコピーします。

既存のジョブをコピーして新しいジョブを作成するには

1. [Jobs] (ジョブ) ページの [Your Jobs] (ジョブ) リストで、複製するジョブを選択します。
2. [Actions] (アクション) メニューから、[Clone job] (ジョブのクローン) を選択します。
3. 新しいジョブの名前を入力します。その後、ジョブを保存または編集できます。

ジョブの削除

不要になったジョブを削除できます。1 回の操作で 1 つ以上のジョブを削除できます。

AWS Glue Studio からジョブを削除する方法

1. [Jobs] (ジョブ) ページの [Your Jobs] (ジョブ) リストで、削除するジョブを選択します。
2. [Actions] (アクション) メニューから、[Delete job] (ジョブの削除) を選択します。
3. 「**delete**」を入力して、ジョブを削除することを確認します。

ビジュアルエディタで、そのジョブの [Job details] (ジョブの詳細) タブを表示して、保存したジョブを削除することもできます。

AWS Glue でのジョブの使用

AWS Glue ジョブには、ソースデータに接続して処理し、データターゲットに書き出すスクリプトがカプセル化されています。通常、ジョブは、抽出、変換、ロード (ETL) スクリプトを実行します。ジョブでは、汎用 Python スクリプト (Python シェルジョブ) を実行することもできます。AWS Glue トリガーでは、スケジュールまたはイベントに基づいて、またはオンデマンドでジョブを開始できます。ジョブ実行をモニタリングすると、完了ステータス、継続時間、開始時間などのランタイムメトリクスを知ることができます。

AWS Glue で生成されたスクリプトを使用することも、独自のスクリプトを使用することもできます。ソーススキーマとターゲット位置またはスキーマを指定すると、AWS Glue コードジェネレーターで Apache Spark API (PySpark) スクリプトを自動的に作成できます。このスクリプトを出発点として使用し、目標に合わせて編集できます。

AWS Glue は、JSON、CSV、ORC (Optimized Row Columnar)、Apache Parquet、Apache Avro などのいくつかのデータ形式で出力ファイルを書き込むことができます。一部のデータ形式では、一般的な圧縮形式を記述できます。

AWS Glue は次のジョブのタイプをサポートします。

- Spark ジョブは、AWS Glue によって管理される Apache Spark 環境で実行されます。データはバッチで処理されます。
- ストリーミング ETL ジョブは Spark ジョブに似ていますが、データストリームで ETL を実行する点が異なります。このタイプのジョブでは Apache Spark 構造化ストリーミングフレームワークが使用されます。一部の Spark ジョブ機能は、ストリーミング ETL ジョブでは使用できません。
- Python シェルジョブは Python スクリプトをシェルとして実行し、使用している AWS Glue バージョンに応じた Python バージョンをサポートします。このタイプのジョブでは、Apache Spark 環境を必要としないタスクをスケジュールして実行できます。
- Ray は、Python を中心としたワークロードのスケールアップに使用できるオープンソースの分散計算フレームワークです。AWS GlueRay ジョブおよびインタラクティブセッションにより、AWS Glue 内で Ray を使用できます。

以下のセクションでは、AWS Glue の ETL ジョブと Ray ジョブについて説明します。

トピック

- AWS Glue バージョン
- AWS Glue での Spark ジョブの使用

- AWS Glue での Ray ジョブの使用
- AWS Glue での Python シェルジョブに関するジョブプロパティの設定
- AWS Glue のモニタリング
- AWS Glue ジョブ実行ステータス

AWS Glue バージョン

AWS Glue バージョンパラメータは、ジョブを追加または更新するときに設定できます。AWS Glue バージョンにより、AWS Glueがサポートする Apache Spark と Python のバージョンが決定されます。Python バージョンは、Spark タイプのジョブでサポートされるバージョンを示します。次の表に、使用可能な AWS Glue のバージョン、対応する Spark と Python のバージョン、および機能上のその他の変更点を示します。

AWS Glue バージョン

AWS Glue バージョン	サポートされているランタイム環境バージョン	サポートされている Java バージョン	機能の変更
AWS Glue 5.0	• Spark 3.5.4 • Python 3.11 • Scala 2.12.18	Java 17	フレームワークへのアップグレードに加えて、この AWS Glue リリースでは、以下の最適化とアップグレードが組み込まれています。 • Amazon SageMaker Unified Studio のサポート • Amazon SageMaker Lakehouse のサポート • オープンテーブルフォーマット (OTF)

AWS Glue バージョン	サポートされているランタイム環境バージョン	サポートされている Java バージョン	機能の変更
			を Hudi 0.15.0、Iceberg 1.7.1、Delta Lake 3.3.0 に更新 • Lake Formation を使用した Spark ネイティブのきめ細かなアクセスコントロール • Access Grants のサポート • 追加の Python ライブラリをインストールするための `requirements.txt` をサポート • Amazon DataZone のデータリネージュのサポート • Amazon S3 テーブルバケットのサポート • AWS Glue Data Catalog のマルチダイアレクトビューのサポート 制限 AWS Glue 5.0 での制限事項は以下のとおりです。

AWS Glue バージョン	サポートされているランタイム環境バージョン	サポートされている Java バージョン	機能の変更
			• Glue 4.0 以前でサポートされていた、AWS Lake Formation のアクセス許可を持つ Glue Dynamic Frame/`GlueContext` ベースのテーブルレベルのアクセスコントロールは、Glue 5.0 ではサポートされません。Glue 5.0 では新しい Spark ネイティブのきめ細かなアクセスコントロール (FGAC) を使用します。 AWS Glue バージョン 5.0 への移行に関する詳細については、「Spark ジョブの AWS Glue の AWS Glue バージョン 5.0 への移行」を参照してください。

AWS Glue バージョン	サポートされているランタイム環境バージョン	サポートされている Java バージョン	機能の変更

AWS Glue 4.0	Spark 環境バージョン
	• Spark 3.3.0
	• Python 3.10
		Java 8	AWS Glue 4.0 では、AWS Glue リリースで次のような多くの最適化とアップグレードが組み込まれています。
			• Spark 3.1 から Spark 3.3 になり、Spark の機能の多くがアップグレードされています。
			  • Pandas と組み合わせたときに改善されるいくつかの機能。詳細については、「What's New in Spark 3.3」(Spark 3.3 の新機能) を参照してください。
			  • Amazon EMR で開発されたその他の最適化。
			  • EMR ファイルシステム (EMRFS) 2.53 へのアップグレード。
			• Log4j 1.x から Log4j 2 への移行

AWS Glue バージョン	サポートされているランタイム環境バージョン	サポートされている Java バージョン	機能の変更
			• Boto のアップグレード版など、いくつかの Python モジュールの AWS Glue 3.0 からの更新。 • デフォルトの Amazon Redshift コネクタを含むいくつかのコネクタのアップグレード。「付録 C: コネクタのアップグレード」を参照してください。 • いくつかの JDBC ドライバーのアップグレード。「付録 B: JDBC ドライバーのアップグレード」を参照してください。 • 新しい Amazon Redshift コネクタと JDBC ドライバーで更新されました。 • Apache Hudi、Delta Lake、および Apache Iceberg によるオープンデータレイクフレーム

AWS Glue バージョン	サポートされているランタイム環境バージョン	サポートされている Java バージョン	機能の変更
			ワークのネイティブサポート。 • Amazon S3 ベースのクラウドシャッフルストレージプラグイン (Apache Spark プラグイン) のネイティブサポート。これにより、Amazon S3 によるシャッフルと伸縮自在なストレージ容量が使用できるようになります。 制限 AWS Glue 4.0 での制限事項を次に示します。 • AWS Glue 機械学習変換と個人を特定できる情報 (PII) 変換は AWS Glue 4.0 ではまだ使用できません。 AWS Glue バージョン 4.0 への移行に関する詳細については、「Spark ジョブ

AWS Glue バージョン	サポートされているランタイム環境バージョン	サポートされているJava バージョン	機能の変更
			の AWS Glue の AWS Glue バージョン 4.0 への移行」を参照してください。

AWS Glue バージョン	サポートされているランタイム環境バージョン	サポートされている Java バージョン	機能の変更
	Ray 環境バージョン • Ray 2.4.0 Python 3.9	該当なし	AWS Glue for Ray を使用して分散型 Python アプリケーションをビルドして実行します。 • Python 3.9 で Ray-2.4.0 データ分散 (`ray[data]`) をサポートします。この Ray リリースの詳細については、Ray GitHub リポジトリの Ray-2.4.0 を参照してください。 • `Ray2.4` ランタイム環境への追加の Python ライブラリのインストールをサポートします。詳細については、「the section called “Ray ジョブ用の Python モジュールを追加する”」を参照してください。 • Ray ジョブのログとメトリクスを Amazon CloudWatch と統合します。詳細については、

AWS Glue バージョン	サポートされているランタイム環境バージョン	サポートされている Java バージョン	機能の変更
			「the section called “Ray エラーをトラブルシューティングする”」および「the section called “Ray ジョブメトリクス”」を参照してください。 • 各ジョブ実行ページでは、AWS Glue Studio の Ray ジョブのメトリクスを集約して視覚化します。 • クラスター内の各作業ディレクトリへのファイルの配布、Ray オブジェクトストアから Amazon S3 へのオブジェクトのスピル、Ray ジョブに割り当てられるワーカーノードの最小数の制御をサポートします。詳細については、「the section called “Ray ジョブのパラメータ”」を参照してください。

AWS Glue バージョン	サポートされているランタイム環境バージョン	サポートされている Java バージョン	機能の変更
			AWS Glue 4.0 での Ray ジョブの制限事項 • Ray の AWS Glue インタラクティブセッションは、このリリースでもまだプレビュー段階です。 • AWS Glue for Ray の Amazon VPC との統合は、現在ご利用いただけません。AWS にある VPC 内のリソースには、パブリックルートがないとアクセスできません。Amazon VPC での AWS Glue の使用に関する詳細は、「the section called “AWS Glue のインターフェイス VPC エンドポイント (AWS PrivateLink) の設定”」を参照してください。 • AWS Glue for Ray は、米国東部 (バージニア北部)、米国

AWS Glue バージョン	サポートされているランタイム環境バージョン	サポートされているJava バージョン	機能の変更
			東部 (オハイオ)、米国西部 (オレゴン)、アジアパシフィック (東京)、欧州 (アイルランド) でご利用いただけます。

AWS Glue バージョン	サポートされているランタイム環境バージョン	サポートされている Java バージョン	機能の変更

AWS Glue 3.0	• Spark 3.1.1
	• Python 3.7
		Java 8	Spark エンジン 3.0 へのアップグレードに加えて、この AWS Glue リリースでは、以下の最適化とアップグレードが行われています。
			• Spark のメジャーリリースである Spark 3.0 に対するAWS Glue ETL ライブラリの構築。
			• AWS Glue 3.0 によるストリーミングジョブのサポート
			• パフォーマンスと信頼性のための、以下の新しい AWS Glue Spark ランタイム最適化。
			  • CSV データ読み取りのための、より高速化された Apache Arrow ベースのインメモリ列指向処理。
			  • CSV データに関するベクトル化された読み取り

AWS Glue バージョン	サポートされているランタイム環境バージョン	サポートされているJava バージョン	機能の変更
			のための SIMD ベースの実行。 • Spark のアップグレードには、Amazon EMR で開発された他の最適化も含まれます。 • EMRFS 2.38 から 2.46 へのアップグレードにともなう、Amazon S3 アクセスのための新機能追加とバグ修正。 • 新しい Spark バージョンに必要ないくつかの依存関係のアップグレード。 • ネイティブにサポートされているデータソース用の、JDBC ドライバのアップグレード。 制限 以下に、AWS Glue 3.0での制限事項を示します。

AWS Glue バージョン	サポートされているランタイム環境バージョン	サポートされているJava バージョン	機能の変更
			• 現在、AWS Glue の 機械学習変換は AWS Glue 3.0 で使用できません。 • Spark 2.4 に依存しており、Spark 3.1との互換性がない一部のカスタム Spark コネクタは、AWS Glue 3.0 では動作しません。

AWS Glue バージョン	サポートされているランタイム環境バージョン	サポートされている Java バージョン	機能の変更
AWS Glue 2.0 (2026 年 4 月 1 日にライフサイクル終了)	• Spark 2.4.3 • Python 3.7	該当なし	AWS Glue バージョン 2.0 では、AWS Glue バージョン 1.0 で提供される機能に加えて次の機能も提供されます。 • AWS Glue で Apache Spark ETL ジョブを実行するための、アップグレードされたインフラストラクチャにより、起動時間が大幅に短縮されています。 • デフォルトの口グ記録機能はリアルタイムになり、ストリームがドライバー用とエグゼキュター用、出力用と工ラー用の間でそれぞれ分離されています。 • ジョブレベルで追加の Python モジュールまたは異なるバージョンを指定するためのサポート。

AWS Glue バージョン	サポートされているランタイム環境バージョン	サポートされている Java バージョン	機能の変更
			Note 基礎となるアーキテクチャの変更に伴い、AWS Glue バージョン 2.0 での一部の依存関係とバージョンが、AWS Glue バージョン 1.0 から変更されています。AWS Glue のメジャーバージョンリリース間での移行の前に、AWS Glue ジョブの検証を行ってください。

AWS Glue バージョン	サポートされているランタイム環境バージョン	サポートされている Java バージョン	機能の変更
AWS Glue 1.0 (2026 年 4 月 1 日にライフサイクル終了)	• Spark 2.4.3 • Python 2.7 • Python 3.6	該当なし	AWS Glue ETL ジョブ (AWS Glue バージョン 1.0 を使用) には、Parquet 形式と ORC 形式のジョブのブックマークを維持できます。これまで AWS Glue ETL ジョブでは、JSON、CSV、Apache Avro、XML などの一般的な Amazon S3 ソース形式のみのブックマークが可能でした。 ETL 入力および出力の形式オプションを設定する際に、Apache Avro のリーダー/ライター形式 1.8 を使用して Avro 論理型の読み取りと書き込みをサポートするように指定できます (AWS Glue バージョン 1.0 を使用)。以前は、バージョン 1.7 Avro リーダー/ライター形式のみがサポートされていました。

AWS Glue バージョン	サポートされているランタイム環境バージョン	サポートされている Java バージョン	機能の変更
			DynamoDB 接続タイプは、(AWS Glue バージョン 1.0 を使用する) ライターオプションをサポートします。 制限 AWS Glue 1.0 での制限事項を次に示します。 • AWS Glue バージョン 0.9 と 1.0 は、今後、アジアパシフィック (ジャカルタ) (`ap-southeast-3`)、中東 (UAE) (`me-central-1`)、その他の新しい地域ではご利用いただけなくなります。

AWS Glue バージョン	サポートされているランタイム環境バージョン	サポートされている Java バージョン	機能の変更
AWS Glue 0.9 (2026 年 4 月 1 日にライフサイクル終了)	• Spark 2.2.1 • Python 2.7	該当なし	AWS Glue バージョンを指定せずに作成されたジョブは、デフォルトで AWS Glue 0.9 に設定されます。 制限 AWS Glue 0.9 での制限事項を次に示します。 • AWS Glue バージョン 0.9 と 1.0 は、今後、アジアパシフィック (ジャカルタ) (ap-southeast-3)、中東 (UAE) (me-central-1)、その他の新しい地域ではご利用いただけなくなります。

Note

次の Glue バージョンは、PythonShell のこれらのバージョンをサポートしています。

- PythonShell v3.6 は Glue バージョン 1.0 でサポートされています。
- PythonShell v3.9 は Glue バージョン 3.0 でサポートされています。

さらに、開発エンドポイントは Glue バージョン 1.0 および 0.9 でのみサポートされています。

AWS Glue バージョンサポートポリシー

AWS Glue は、分析、機械学習、アプリケーション開発のためのデータの検出、準備、結合を容易にするサーバーレスデータ統合サービスです。AWS Glue ジョブでは、AWS Glue でのデータ統合作業を実行するビジネスロジックが含まれています。AWS Glue のジョブには、Spark (バッチとストリーミング)、Ray および Python シェルの 3 つのタイプがあります。ジョブを定義するときは、基盤となるランタイム環境の Spark、Ray および Python のバージョンを構成する AWS Glue バージョンを指定します。例: AWS Glue バージョン 5.0 の Spark のジョブでは、Spark 3.5.4 と Python 3.11 がサポートされています。

サポートポリシー

各 AWS Glue バージョンは、メンテナンスとセキュリティの更新の対象となるオペレーティングシステム、プログラミング言語、およびソフトウェアライブラリの組み合わせを中心に構築されています。AWS Glue のバージョンサポートポリシーは、あるバージョンの主要コンポーネントのいずれかに対するコミュニティ長期サポート (LTS) 期間が満了し、セキュリティアップデートが利用できなくなった時点でそのバージョンのサポートを終了するというものです。AWS Glue のバージョンサポートポリシーは以下の各ステータスを含みます。

サポート終了 (EOS) - AWS Glue バージョンが EOS に達すると:

- AWS Glue は、EOS バージョンにセキュリティパッチやその他の更新を適用しなくなります。
- EOS バージョンの AWS Glue ジョブは、テクニカルサポートの対象外です。
- ジョブが EOS バージョンで実行されている場合、AWS Glue は SLA を尊重しません。

ライフサイクル終了 (EOL) - AWS Glue バージョンが EOL に達すると:

- EOL バージョンでは新しい AWS Glue ジョブやインタラクティブセッションを作成できなくなります。
- これらの AWS Glue バージョンではジョブ実行を開始できなくなります。
- EOL バージョンでは、AWS Glue は既存のジョブ実行とインタラクティブセッションを停止します。

- EOL バージョンは AWS Glue SDK および API から削除されます。

以下の AWS Glue バージョンはサポートが終了しており、今後 EOL 期日を過ぎると使用できなくなります。バージョンのサポートステータスの変更は、記載の日付の午前 0 時 (太平洋標準時) に開始します。

Type	Glue バージョン	サポート終了	ライフサイクル終了
Spark	Glue バージョン 0.9 (Spark 2.2、Scala 2、Python 2)	2022 年 6 月 1 日	2026 年 4 月 1 日
Spark	Glue バージョン 1.0 (Spark 2.4、Python 2)	2022 年 6 月 1 日	2026 年 4 月 1 日
Spark	Glue バージョン 1.0 (Spark 2.4、Scala 2、Python 3)	2022 年 9 月 30 日	2026 年 4 月 1 日
Spark	Glue バージョン 2.0 (Spark 2.4、Python 3)	2024 年 1 月 31 日	2026 年 4 月 1 日
Type	Python バージョン	サポート終了	ライフサイクル終了
Python シェル	Python 2 (Glue バージョン 1.0)	2022 年 6 月 1 日	2026 年 4 月 1 日
Python シェル	PythonShell 3.6 (Glue バージョン 1.0)	2026 年 3 月 31 日	NA
Type	ノートブックのバージョン	サポート終了	ライフサイクル終了
開発エンドポイント	Zeppelin ノートブック	2022 年 9 月 30 日	NA

AWS は、ジョブをサポートしているバージョンに移行することを強く推奨します。

Spark ジョブを最新の AWS Glue バージョンへ移行する方法の詳細については、「AWS Glue ジョブの AWS Glue バージョン 5.0 への移行」を参照してください。

Python シェルのジョブを最新のものに移行するため AWS Glue バージョン:

- コンソールで、[Python 3 (Glue Version 4.0)] を選択します。
- CreateJob/UpdateJob API で、GlueVersion パラメータを 2.0 に、Command パラメータで PythonVersion を 3 に設定します。GlueVersion 設定は Python シェルジョブの動作に影響しないため、GlueVersion をインクリメントするメリットはありません。
- ジョブスクリプトを Python 3 と互換性を持たせる必要があります。

Spark ジョブの AWS Glue の AWS Glue バージョン 5.0 への移行

このトピックでは、AWS Glue バージョン 0.9、1.0、2.0、3.0、4.0 の間の変更点について説明し、Spark アプリケーションと ETL ジョブを AWS Glue 5.0 に移行できるようにします。また、AWS Glue 5.0 の機能とそれを使用する利点についても説明します。

この機能を AWS Glue ETL ジョブに使用するには、ジョブを作成するとき、Glue version として **5.0** を選択します。

トピック

- 新機能
- AWS Glue 5.0 に移行するためのアクション
- 移行チェックリスト
- AWS Glue 5.0 の機能
- AWS Glue 4.0 から AWS Glue 5.0 への移行
- AWS Glue 3.0 から AWS Glue 5.0 への移行
- AWS Glue 2.0 から AWS Glue 5.0 への移行
- AWS Glue 5.0 でのログ記録動作の変更
- AWS Glue 5.0 向けのコネクタと JDBC ドライバーの移行

新機能

このセクションでは、AWS Glue バージョン 5.0 の新機能と利点について説明します。

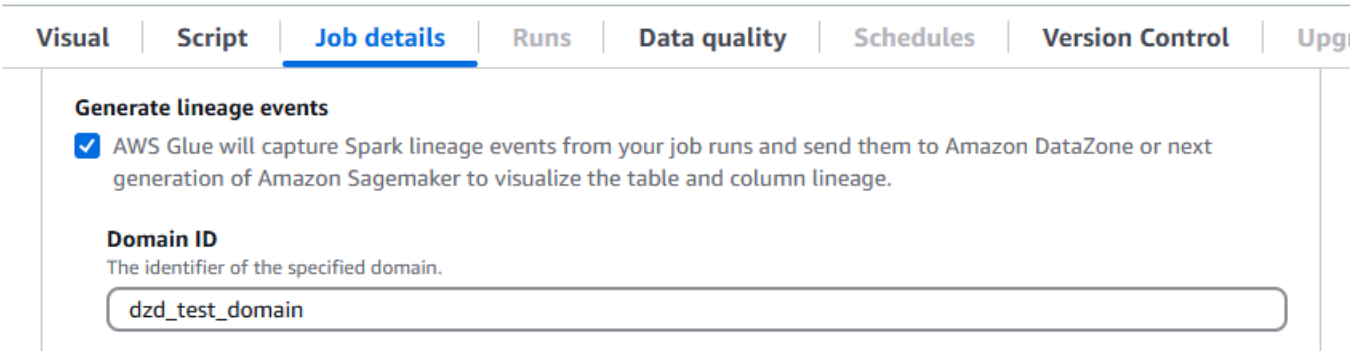
- Apache Spark が AWS Glue 4.0 の 3.3.0 から AWS Glue 5.0 の 3.5.4 に更新されました。「Spark 3.3.0 から Spark 3.5.4 への主な機能強化」を参照してください。
- Lake Formation を使用した Spark ネイティブの詳細なアクセスコントロール (FGAC) これには、FGAC for Iceberg、Delta、Hudi テーブルが含まれます。詳細については、「Using AWS Glue with AWS Lake Formation for fine-grained access control」を参照してください。

  Spark ネイティブの FGAC に関する以下の考慮事項または制限事項にご注意ください。

  - 現在、データの書き込みはサポートされていません
  - Lake Formation を使用して `GlueContext` から Iceberg に書き込むには、代わりに IAM アクセスコントロールを使用する必要があります

  Spark ネイティブ FGAC を使用する際の制限事項と考慮事項の完全なリストについては、「the section called “考慮事項”」を参照してください。
- AWS Glue から Amazon S3 データへのスケーラブルなアクセスコントロールソリューションとして Amazon S3 Access Grants をサポート。詳細については、「AWS Glue での Amazon S3 Access Grants の使用」を参照してください。
- オープンテーブルフォーマット (OTF) を Hudi 0.15.0、Iceberg 1.7.1、Delta Lake 3.3.0 に更新
- Amazon SageMaker Unified Studio のサポート
- Amazon SageMaker Lakehouse とデータ抽象化の統合。詳細については、「AWS Glue ETL からのメタストアデータカタログのクエリ」を参照してください。
- `requirements.txt` を使用して追加の Python ライブラリのインストールをサポート。詳細については、「requirements.txt を使用して AWS Glue 5.0 に追加の Python ライブラリをインストールする」を参照してください。
- AWS Glue 5.0 は Amazon DataZone のデータ系列をサポートしています。Spark ジョブの実行中に系列情報を自動収集し、系列イベントを送信して Amazon DataZone に表示されるように AWS Glue を設定できます。詳細については、「Amazon DataZone のデータ統合」を参照してください。

  AWS Glue コンソールでこれを設定するには、[系列イベントを生成] をオンにし、Amazon DataZone ドメイン ID を [ジョブの詳細] タブに入力します。



あるいは、次のジョブパラメータを指定することもできます (DataZone ドメイン ID を指定します)。

- キー: `--conf`
- 値:

```
extraListeners=io.openlineage.spark.agent.OpenLineageSparkListener
—conf spark.openlineage.transport.type=amazon_datazone_api
-conf spark.openlineage.transport.domainId=<your-domain-ID>
```

- コネクタと JDBC ドライバーの更新。詳細については、「付録 B: JDBC ドライバーのアップグレード」および「付録 C: コネクタのアップグレード」を参照してください。
- Java が 8 から 17 に更新。
- AWS Glue `G.1X` と `G.2X` ワーカーのストレージを増やし、ディスク容量をそれぞれ 94GB と 138GB に引き上げました。詳細については、ジョブを参照してください。
- AWS SDK for Java バージョン 2 をサポート - ジョブが v2 をサポートしている場合、AWS Glue 5.0 ジョブは Java バージョン 1.12.569 または 2.28.8 を使用できます。AWS SDK for Java 2.x は、1.x コードベースより大幅に変更されています。今回のバージョンは Java 8 以降をベースに構築されており、これまで要望の声が高かった機能が複数追加されています。このバージョンでは、ノンブロッキング I/O がサポートされており、ランタイム時にさまざまな HTTP 実装をプラグインすることができます。SDK for Java v1 から v2 への移行ガイドを含む詳細については、「AWS SDK for Java バージョン 2」ガイドを参照してください。
-

重要な変更

以下の最新の変更点に注意してください。

- AWS Glue 5.0 では、S3A ファイルシステムを使用する場合、および「fs.s3a.endpoint」と「fs.s3a.endpoint.region」の両方が設定されていない場合、S3A で使用されるデフォルトのリージョンは「us-east-2」です。これにより、特に VPC ジョブの場合、S3 アップロードタイムアウトエラーなどの問題が発生する可能性があります。この変更によって発生する問題を軽減するには、AWS Glue 5.0 で S3A ファイルシステムを使用するときに「fs.s3a.endpoint.region」Spark 設定を設定します。
- Lake Formation のきめ細かなアクセスコントロール (FGAC)
  - AWS Glue 5.0 は、Spark DataFrames を使用した新しい Spark ネイティブ FGAC のみをサポートしています。AWS Glue DynamicFrames を使用した FGAC はサポートされません。
    - 5.0 で FGAC を使用するには、AWS Glue DynamicFrames から Spark DataFrames に移行する必要があります。
    - FGAC が不要であれば、Spark DataFrame に移行にする必要はなく、ジョブのブックマークやプッシュダウン述語などの GlueContext 機能は引き続き機能します。
  - Spark ネイティブ FGAC を使用するジョブには少なくとも 4 つのワーカー (ユーザードライバー、システムドライバー、システムエグゼキューター、スタンバイユーザーエグゼキューターをそれぞれ 1 つずつ) が必要です。
  - 詳細については、「Using AWS Glue with AWS Lake Formation for fine-grained access control」を参照してください。
- Lake Formation フルテーブルアクセス (FTA)
  - AWS Glue 5.0 は、Spark ネイティブ DataFrames (新規) と GlueContext DynamicFrames (レガシー、制限付き) で FTA をサポートします。
  - Spark ネイティブ FTA
    - 4.0 スクリプトが GlueContext を使用している場合は、ネイティブ Spark の使用に移行します。
    - この機能は Hive テーブルと Iceberg テーブルに限定されます。
    - Spark ネイティブ FTA を使用するように 5.0 ジョブを設定する方法の詳細については、以下を参照してください。
  - GlueContext DynamicFrame FTA
    - コードの変更は不要
    - この機能は OTF 以外のテーブルに限定されており、Iceberg、Delta Lake、Hudi では機能しません。
- ベクトル化された SIMD CSV リーダーはサポートされません。

- 出力ロググループへの継続的なログ記録はサポートされません。代わりに `error` ロググループを使用します。
- AWS Glue ジョブ実行インサイト `job-insights-rule-driver` は廃止されました。`job-insights-rca-driver` ログストリームがエラーロググループに配置されるようになりました。
- Athena ベースのカスタム/マーケットプレイスコネクタはサポートされません。
- Adobe Marketo Engage、Facebook Ads、Google Ads、Google Analytics 4、Google Sheets、Hubspot、Instagram Ads、Intercom、Jira Cloud、Oracle NetSuite、Salesforce、Salesforce Marketing Cloud、Salesforce Marketing Cloud Account Engagement、SAP OData、ServiceNow、Slack、Snapchat Ads、Stripe、Zendesk、Zoho CRM のコネクタはサポートされません。
- カスタム log4j プロパティは AWS Glue 5.0 ではサポートされていません。

Spark 3.3.0 から Spark 3.5.4 への主な機能強化

次の機能強化があります。

- Spark Connect 用の Python クライアント (SPARK-39375)。
- テーブルの列の DEFAULT 値に対するサポートを実装 (SPARK-38334)。
- 「リテラル列のエイリアス参照」をサポート (SPARK-27561)。
- エラークラスで SQLSTATE の使用を強化 (SPARK-41994)。
- Bloom フィルターの結合をデフォルトで有効化 (SPARK-38841)。
- 大規模アプリケーションの Spark UI のスケーラビリティとドライバーの安定性を向上 (SPARK-41053)。
- 構造化ストリーミングでの非同期進行状況の追跡 (SPARK-39591)。
- 構造化ストリーミングでの Python の任意のステートフル処理 (SPARK-40434)。
- Pandas API カバレッジの改善 (SPARK-42882) と PySpark での NumPy 入力サポート (SPARK-39405)。
- PySpark ユーザー定義関数にメモリプロファイラーを提供 (SPARK-40281)。
- PyTorch ディストリビューターを実装 (SPARK-41589)。
- SBOM アーティファクトを公開 (SPARK-41893)。
- IPv6 限定の環境をサポート (SPARK-39457)。
- カスタマイズされた K8 スケジューラ (Apache YuniKorn と Volcano) GA (SPARK-42802)。

- Spark Connect での Scala および Go クライアントのサポート (SPARK-42554) および (SPARK-43351)。
- Spark Connect 用の PyTorch ベースの分散 ML サポート (SPARK-42471)。
- Python および Scala での Spark Connect の構造化ストリーミングのサポート (SPARK-42938)。
- Python Spark Connect クライアントの Pandas API のサポート (SPARK-42497)。
- Arrow Python UDF を導入 (SPARK-40307)。
- Python のユーザー定義テーブル関数をサポート (SPARK-43798)。
- PySpark エラーをエラークラスに移行 (SPARK-42986)。
- PySpark テストフレームワーク (SPARK-44042)。
- Datasketches HllSketch のサポートを追加 (SPARK-16484)。
- 組み込み SQL 関数の改善 (SPARK-41231)。
- IDENTIFIER 句 (SPARK-43205)。
- SQL 関数を Scala、Python、R API に追加 (SPARK-43907)。
- SQL 関数の名前付き引数のサポートを追加 (SPARK-43922)。
- シャッフルデータを移行した場合に、廃止済みエグゼキューターの喪失に対して不要なタスクが再実行されるのを回避 (SPARK-41469)。
- 分散 ML <> のスパーク接続 (SPARK-42471)。
- DeepSpeed ディストリビューター (SPARK-44264)。
- RocksDB ステートストアの変更ログチェックポイントを実装 (SPARK-43421)。
- オペレーター間のウォーターマークの伝播を導入 (SPARK-42376)。
- dropDuplicatesWithinWatermark を導入 (SPARK-42931)。
- RocksDB ステートストアプロバイダーのメモリ管理を強化 (SPARK-43311)。

AWS Glue 5.0 に移行するためのアクション

既存のジョブについては、ジョブ設定で、`Glue version` を以前のバージョンから `Glue 5.0` に変更します。

- AWS Glue Studio では、`Glue version` で `Glue 5.0 - Supports Spark 3.5.4, Scala 2, Python 3` を選択します。
- API では、`UpdateJob` API オペレーションの `GlueVersion` パラメータで **`5.0`** を選択します。

新しいジョブについては、ジョブを作成するときに Glue 5.0 を選択します。

- コンソールでは、Glue version で Spark 3.5.4, Python 3 (Glue Version 5.0) or Spark 3.5.4, Scala 2 (Glue Version 5.0) を選択します。
- AWS Glue Studio では、Glue version で Glue 5.0 - Supports Spark 3.5.4, Scala 2, Python 3 を選択します。
- API では、CreateJob API オペレーションの GlueVersion パラメータで **5.0** を選択します。

AWS Glue 2.0 以前で生成された AWS Glue 5.0 Spark イベントログを表示するには、AWS CloudFormation または Docker を使用して AWS Glue 5.0 用にアップグレードされた Spark 履歴サーバーを起動します。

移行チェックリスト

移行については、次のチェックリストを確認してください。

- Java 17 の更新
- [Scala] AWS SDK 呼び出しを v1 から v2 にアップグレード
- Python 3.10 から 3.11 への移行
- [Python] boto リファレンスを 1.26 から 1.34 に更新

AWS Glue 5.0 の機能

このセクションでは AWS Glue の機能を詳しく説明します。

AWS Glue ETL からのメタストアデータカタログのクエリ

AWS Glue ジョブを登録すると AWS Glue Data Catalog にアクセスできます。これにより、さまざまなコンシューマーがテーブルやその他のメタストアリソースを利用できるようになります。データカタログは、Amazon S3 データレイク全体のすべてのデータを統合するマルチカタログ階層をサポートしています。また、データにアクセスするための Hive メタストア API とオープンソースの Apache Iceberg API の両方が用意されています。これらの機能は、AWS Glue とその他のデータ指向サービス (Amazon EMR、Amazon Athena、Amazon Redshift など) で使用できます。

Data Catalog でリソースを作成すると、Apache Iceberg REST API をサポートしている SQL エンジンからそれらのリソースにアクセスできます。アクセス許可は AWS Lake Formation によって管理されます。設定後、AWS Glue の機能を活用して、使い慣れたアプリケーションでこれらのメタスト

アリソースに対してクエリを実行することで、さまざまなデータを照会することができます。これには Apache Spark と Trino も含まれます。

メタデータリソースの整理方法

データは、AWS Glue Data Catalog を使用して、カタログ、データベース、テーブルの論理階層に整理されます。

- カタログ – スキーマやテーブルなどデータストアのオブジェクトを保持する論理コンテナ。
- データベース – テーブルやビューなどのデータオブジェクトをカタログ内で整理します。
- テーブルとビュー – 抽象化レイヤーと理解可能なスキーマを提供するデータベース内のデータオブジェクト。これにより、さまざまな場所にあるさまざまな形式の参照元データに簡単にアクセスすることができます。

AWS Glue 4.0 から AWS Glue 5.0 への移行

機械学習の変換を除き、AWS Glue 4.0 にある既存のジョブパラメータと主要な機能はすべて AWS Glue 5.0 に移行します。

以下の新しいパラメータが追加されました。

- `--enable-lakeformation-fine-grained-access`: AWS Lake Formation テーブルのきめ細かなアクセスコントロール (FGAC) 機能を有効にします。

Spark の移行ドキュメントを参照してください。

- Migration Guide: Spark Core
- Migration Guide: SQL, Datasets and DataFrame
- Migration Guide: Structured Streaming
- Upgrading PySpark

AWS Glue 3.0 から AWS Glue 5.0 への移行

> **Note**
>
> AWS Glue 4.0 に関連する移行ステップについては、「AWS Glue 3.0 から AWS Glue 4.0 への移行」を参照してください。

機械学習の変換を除き、AWS Glue 3.0 にある既存のジョブパラメータと主要な機能はすべて AWS Glue 5.0 に移行します。

AWS Glue 2.0 から AWS Glue 5.0 への移行

> **Note**
>
> AWS Glue 4.0 に関連する移行ステップと、AWS Glue バージョン 3.0 と 4.0 の移行の相違点をまとめたリストについては、「AWS Glue 3.0 から AWS Glue 4.0 への移行」を参照してください。

また、AWS Glue バージョン 3.0 と 2.0 の移行では、以下の相違点があります。

- 機械学習の変換を除き、AWS Glue 2.0 にある既存のジョブパラメータと主要な機能はすべて AWS Glue 5.0 に移行します。
- 一部の Spark の変更のみについてですが、削除された機能が参照されないように、スクリプトの修正が必要になる場合があります。例えば、Spark 3.1.1 以降では Scala 型が指定されていない UDF は有効になりませんが、Spark 2.4 では有効になります。
- Python 2.7 はサポートされていません。
- 既存の AWS Glue 2.0 ジョブで提供されている追加の jar は、いくつかの依存関係がアップグレードされたため、依存関係の競合を引き起こす可能性があります。クラスパスの競合は `--user-jars-first` ジョブパラメータによって回避することができます。
- パーケットファイルから、またはパーケットファイルへのタイムスタンプの読み込み/保存の動作が変更されています。詳細は「Upgrading from Spark SQL 3.0 to 3.1」を参照してください。
- ドライバー/エグゼキューター設定における異なる Spark タスクの並列処理。タスクの並列処理は `--executor-cores` ジョブ引数を渡すことで調整できます。

AWS Glue 5.0 でのログ記録動作の変更

以下は、AWS Glue 5.0 のログ記録動作の変更点です。詳細については、「AWS Glue ジョブのログ記録」を参照してください。

- すべてのログ (システムログ、Spark デーモンログ、ユーザーログ、Glue Logger ログ) がデフォルトで `/aws-glue/jobs/error` ロググループに書き込まれるようになりました。
- 以前のバージョンの継続的なログ記録に使用される `/aws-glue/jobs/logs-v2` ロググループは使用されなくなりました。

- 削除された継続的なログ記録の引数を使用して、ロググループ名またはログストリーム名を変更またはカスタマイズすることはできなくなりました。代わりに、AWS Glue 5.0 の新しいジョブ引数を参照してください。

AWS Glue 5.0 で 2 つの新しいジョブ引数が導入されました

- `--custom-logGroup-prefix`: `/aws-glue/jobs/error` および `/aws-glue/jobs/output` ロググループのカスタムプレフィックスを指定できます。
- `--custom-logStream-prefix`: ロググループ内のログストリーム名にカスタムプレフィックスを指定できます。

  カスタムプレフィックスの検証ルールと制限は次のとおりです。

  - ログストリーム名全体は、1 ~ 512 文字である必要があります。
  - ログストリーム名のカスタムプレフィックスは、400 文字に制限されています。
  - プレフィックスで使用できる文字には、英数字、アンダースコア (`_`)、ハイフン (`-`)、スラッシュ (`/`) などがあります。

AWS Glue 5.0 で廃止された継続的なログ記録の引数

継続的なログ記録の次のジョブ引数は、AWS Glue 5.0 で廃止されました。

- `--enable-continuous-cloudwatch-log`
- `--continuous-log-logGroup`
- `--continuous-log-logStreamPrefix`
- `--continuous-log-conversionPattern`
- `--enable-continuous-log-filter`

AWS Glue 5.0 向けのコネクタと JDBC ドライバーの移行

アップグレードされた JDBC コネクタとデータレイクコネクタのバージョンについては、次を参照してください。

- 付録 B: JDBC ドライバーのアップグレード
- 付録 C: コネクタのアップグレード
- 付録 D: オープンテーブル形式のアップグレード

Glue 5.0 の付録に記載されているコネクタまたはドライバーのバージョンに、次の変更が適用されます。

Amazon Redshift

次の変更があります。

- コネクタが Redshift データ共有テーブルにクエリできるように、3 つの要素で構成されるテーブル名のサポートを追加。
- `INTEGER` ではなく Redshift `SMALLINT` を使用するように Spark `ShortType` のマッピングを修正し、予想されるデータサイズとのマッチングを改善。
- Amazon Redshift Serverless のカスタムクラスター名 (CNAME) のサポートを追加。

Apache Hudi

次の変更があります。

- レコードレベルのインデックスをサポート。
- レコードキーの自動生成をサポート。レコードキーフィールドの指定が不要になりました。

Apache Iceberg

次の変更があります。

- AWS Lake Formation ではきめ細かなアクセスコントロールはサポートされません。
- 独立した独自のライフサイクルを持つスナップショットへの名前付き参照となる分岐とタグ付けをサポート。
- 指定した期間内または特定のスナップショット間で、テーブルに加えられた変更を含む表示を生成する変更ログの表示手順を追加。

Delta Lake

次の変更があります。

- Apache Iceberg と Apache Hudi を介したシームレスなアクセスを可能にする Delta Universal Format (UniForm) をサポート。
- Merge-on-Read パラダイムを実装する削除ベクトルをサポート。

AzureCosmos

次の変更があります。

- 階層パーティションキーのサポートを追加。
- ネストされたプロパティに StringType (raw json) でカスタムのスキーマを使用するオプションを追加。
- クライアントシークレットの代わりに証明書で SPN (ServicePrincipal name) 認証を使用できるようにする設定オプション `spark.cosmos.auth.aad.clientCertPemBase64` を追加。

詳細については、「Azure Cosmos DB Spark connector change log」を参照してください。

Microsoft SQL Server

次の変更があります。

- TLS 暗号化をデフォルトで有効化。
- encrypt = false であるがサーバーが暗号化を要求している場合、証明書は `trustServerCertificate` 接続設定に基づいて検証されます。
- `aadSecurePrincipalId` と `aadSecurePrincipalSecret` を廃止。
- `getAADSecretPrincipalId` API を削除。
- 領域が指定されている場合の CNAME 解決を追加。

MongoDB

次の変更があります。

- Spark 構造化ストリーミングによるマイクロバッチモードをサポート。
- BSON データ型をサポート。
- マイクロバッチモードまたは連続ストリーミングモードを使用する際に複数のコレクションを読み取るためのサポートを追加。
  - `collection` 設定オプションで使用される 1 つのコレクションの名前にカンマが含まれると、Spark Connector は 2 つの異なるコレクションとして処理します。これを回避するには、カンマの前にバックスラッシュ (\) を挿入して エスケープする必要があります。
  - `collection` 設定オプションで使用されるコレクションの名前が「*」である場合、Spark Connector はこれをすべてのコレクションをスキャンするための仕様として解釈します。これを

回避するには、このアスタリスクの前にバックスラッシュ (\) を挿入してエスケープする必要があります。

- `collection` 設定オプションで使用されるコレクションの名前にバックスラッシュ (\) が含まれると、Spark Connector はこれをエスケープ文字として処理するため、値の解釈方法が変わる可能性があります。これを回避するには、上記バックスラッシュの前に別のバックスラッシュを挿入してこれをエスケープする必要があります。

詳細については、「MongoDB connector for Spark release notes」を参照してください。

Snowflake

次の変更があります。

- Snowflake テーブルに保存する際に `StringType` 列の値を自動でトリミングできる新しい `trim_space` パラメータを導入。デフォルト: `false`。
- セッションレベルの `abort_detached_query` パラメータをデフォルトで無効化。
- OAUTH を使用する際の `SFUSER` パラメータの要件を削除。
- Advanced Query Pushdown 機能を削除。代替の機能が用意されています。例えば、Snowflake テーブルからデータをロードする代わりに Snowflake SQL クエリから直接データをロードできます。

詳細については、「Snowflake Connector for Spark release notes」を参照してください。

付録 A: 注目すべき依存関係のアップグレード

依存関係のアップグレードは次のとおりです。

依存関係	AWS Glue 5.0 でのバージョン	AWS Glue 4.0 でのバージョン	AWS Glue 3.0 でのバージョン	AWS Glue 2.0 でのバージョン	AWS Glue 1.0 でのバージョン
Java	17	8	8	8	8
Spark	3.5.4	3.3.0-amzn-1	3.1.1-amzn-0	2.4.3	2.4.3
Hadoop	3.4.1	3.3.3-amzn-0	3.2.1-amzn-3	2.8.5-amzn-5	2.8.5-amzn-1
Scala	2.12.18	2.12	2.12	2.11	2.11

依存関係	AWS Glue 5.0 でのバージョン	AWS Glue 4.0 でのバージョン	AWS Glue 3.0 でのバージョン	AWS Glue 2.0 でのバージョン	AWS Glue 1.0 でのバージョン
Jackson	2.15.2	2.12	2.12	2.11	2.11
Hive	2.3.9-amzn-4	2.3.9-amzn-2	2.3.7-amzn-4	1.2	1.2
EMRFS	2.69.0	2.54.0	2.46.0	2.38.0	2.30.0
Json4s	3.7.0-M11	3.7.0-M11	3.6.6	3.5.x	3.5.x
Arrow	12.0.1	7.0.0	2.0.0	0.10.0	0.10.0
AWS Glue データカタログクライアント	4.5.0	3.7.0	3.0.0	1.10.0	該当なし
AWS SDK for Java	2.29.52	1.12	1.12		
Python	3.11	3.10	37	2.7 と 3.6	2.7 と 3.6
Boto	1.34.131	1.26	1.18	1.12	該当なし
EMR DynamoDB コネクタ	5.6.0	4.16.0			

付録 B: JDBC ドライバーのアップグレード

JDBC ドライバーのアップグレードは次のとおりです。

ドライバー	AWS Glue 5.0 での JDBC ドライバーのバージョン	AWS Glue 4.0 での JDBC ドライバーのバージョン	AWS Glue 3.0 での JDBC ドライバーのバージョン	過去の AWS Glue バージョンでの JDBC ドライバーのバージョン
MySQL	8.0.33	8.0.23	8.0.23	5.1
Microsoft SQL Server	10.2.0	9.4.0	7.0.0	6.1.0
Oracle Database	23.3.0.23.09	21.7	21.1	11.2
PostgreSQL	42.7.3	42.3.6	42.2.18	42.1.0
Amazon Redshift	redshift-jdbc42-2.1.0.29	redshift-jdbc42-2.1.0.16	redshift-jdbc41-1.2.12.1017	redshift-jdbc41-1.2.12.1017
SAP Hana	2.20.17	2.17.12		
Teradata	20.00.00.33	20.00.00.06		

付録 C: コネクタのアップグレード

コネクタのアップグレードは次のとおりです。

ドライバー	AWS Glue 5.0 のコネクタバージョン	AWS Glue 4.0 のコネクタバージョン	AWS Glue 3.0 のコネクタバージョン
EMR DynamoDB コネクタ	5.6.0	4.16.0	
Amazon Redshift	6.4.0	6.1.3	
OpenSearch	1.2.0	1.0.1	
MongoDB	10.4.0	10.0.4	3.0.0

ドライバー	AWS Glue 5.0 のコネクタバージョン	AWS Glue 4.0 のコネクタバージョン	AWS Glue 3.0 のコネクタバージョン
Snowflake	3.0.0	2.12.0	
Google BigQuery	0.32.2	0.32.2	
AzureCosmos	4.33.0	4.22.0	
AzureSQL	1.3.0	1.3.0	
Vertica	3.3.5	3.3.5	

付録 D: オープンテーブル形式のアップグレード

オープンテーブル形式のアップグレードは次のとおりです。

OTF	AWS Glue 5.0 のコネクタバージョン	AWS Glue 4.0 のコネクタバージョン	AWS Glue 3.0 のコネクタバージョン
Hudi	0.15.0	0.12.1	0.10.1
Delta Lake	3.3.0	2.1.0	1.0.0
Iceberg	1.7.1	1.0.0	0.13.1

Spark ジョブの AWS Glue の AWS Glue バージョン 4.0 への移行

このトピックでは、AWS Glue バージョン 0.9、1.0、2.0、および 3.0 の間の変更について説明し、Spark アプリケーションと ETL ジョブを AWS Glue 4.0 に移行できるようにします。また、AWS Glue 4.0 の機能とそれを使用する利点についても説明します。

この機能を AWS Glue ETL ジョブに使用するには、ジョブを作成するとき、`Glue version` として **`4.0`** を選択します。

トピック

- サポートされる新機能
- AWS Glue 4.0 に移行するためのアクション

- 移行チェックリスト
- AWS Glue 3.0 から AWS Glue 4.0 への移行
- AWS Glue 2.0 から AWS Glue 4.0 への移行
- AWS Glue 1.0 から AWS Glue 4.0 への移行
- AWS Glue 0.9 から AWS Glue 4.0 への移行
- AWS Glue 4.0 向けのコネクタと JDBC ドライバーの移行
- 付録 A: 注目すべき依存関係のアップグレード
- 付録 B: JDBC ドライバーのアップグレード
- 付録 C: コネクタのアップグレード

サポートされる新機能

このセクションでは、AWS Glue バージョン 4.0 の新機能と利点について説明します。

- これは Apache Spark 3.3.0 に基づいていますが、アダプティブクエリ実行、ベクトル化されたリーダー、最適化されたシャッフルとパーティション結合など、AWS Glue での最適化や Amazon EMR が含まれています。
- MySQL、Microsoft SQL Server、Oracle、PostgreSQL、MongoDB、アップグレードされた Spark ライブラリなど、すべての AWS Glue ネイティブソース用に JDBC ドライバーをアップグレードしました。依存関係は Spark 3.3.0 によって取り込まれています。
- 新しい Amazon Redshift コネクタと JDBC ドライバーで更新されました。
- EMR File System (EMRFS) をアップグレードして Amazon S3 アクセスを最適化し、デフォルトで Amazon S3 に最適化された出力コミッターが有効になりました。
- パーティションインデックス、プッシュダウン述語、パーティションリスト、アップグレードされた Hive メタストアクライアントにより、データカタログアクセスを最適化しました。
- セルレベルのフィルタリングとデータレイクトランザクションを備えた管理されたカタログテーブル用の Lake Formation と統合しました。
- スタートアップレイテンシーが短縮され、ジョブ全体の完了時間と対話性が改善されます。
- Spark ジョブは、1 秒単位で課金され、最小課金時間は 1/10 になります。つまり、最小 10 分が最小 1 分になります。
- Apache Hudi、Delta Lake、および Apache Iceberg によるオープンデータレイクフレームワークのネイティブサポート。

- Amazon S3 ベースのクラウドシャッフルストレージプラグイン (Apache Spark プラグイン) のネイティブサポート。これにより、Amazon S3 によるシャッフルと伸縮自在なストレージ容量が使用できるようになります。

Spark 3.1.1 から Spark 3.3.0 への主な機能強化

次の機能強化があります。

- 行レベルのランタイムフィルタリング (SPARK-32268)。
- ANSI 拡張機能 (SPARK-38860)。
- エラーメッセージの改善 (SPARK-38781)。
- Parquet ベクトル化リーダーの複合型のサポート (SPARK-34863)。
- Spark SQL の隠しファイルメタデータのサポート (SPARK-37273)。
- Python/Pandas UDF 用のプロファイラーを提供 (SPARK-37443)。
- Trigger.Once のようなストリーミングクエリを複数のバッチで実行できるように、Trigger.AvailableNow を導入 (SPARK-36533)。
- より包括的な Datasource V2 プッシュダウン機能 (SPARK-38788)。
- log4j 1 から log4j 2 への移行 (SPARK-37814)。

その他の注目すべき変更

次の変更があります。

- 重要な変更
  - Python 3.6 サポートへの参照を docs と Python/docs から削除 (SPARK-36977)。
  - 組み込みの pickle を cloudpickle に置き換えることで、名前付きタプルハックを削除 (SPARK-32079)。
  - Pandas の最小バージョンを 1.0.5 に引き上げ (SPARK-37465)。

AWS Glue 4.0 に移行するためのアクション

既存のジョブについては、ジョブ設定で、`Glue version` を以前のバージョンから `Glue 4.0` に変更します。

- AWS Glue Studio では、Glue version で Glue 4.0 - Supports Spark 3.3, Scala 2, Python 3 を選択します。
- API では、UpdateJob API オペレーションの GlueVersion パラメータで **4.0** を選択します。

新しいジョブについては、ジョブを作成するときに Glue 4.0 を選択します。

- コンソールでは、Glue version で Spark 3.3, Python 3 (Glue Version 4.0) or Spark 3.3, Scala 2 (Glue Version 3.0) を選択します。
- AWS Glue Studio では、Glue version で Glue 4.0 - Supports Spark 3.3, Scala 2, Python 3 を選択します。
- API では、CreateJob API オペレーションの GlueVersion パラメータで **4.0** を選択します。

AWS Glue 2.0 以前で生成された AWS Glue 4.0 Spark イベントログを表示するには、AWS CloudFormation または Docker を使用して AWS Glue 4.0 用にアップグレードされた Spark 履歴サーバーを起動します。

移行チェックリスト

- ジョブの外部 Python ライブラリが Python 2.7/3.6 に依存していますか。
  - Spark 3.3.0 では Python 2.7 と 3.6 のサポートが完全に削除されたため、依存ライブラリを Python 2.7/3.6 から Python 3.10 に更新してください。

AWS Glue 3.0 から AWS Glue 4.0 への移行

移行時には、次の変更点に注意してください。

- AWS Glue 3.0 に存在する既存のすべてのジョブパラメータと主要な機能は、AWS Glue 4.0 に存在します。
- AWS Glue 3.0 では、Amazon EMR に最適化された Spark 3.1.1 を使用していますが、AWS Glue 4.0 では、Amazon EMR に最適化された Spark 3.3.0 を使用しています。

  一部の Spark の変更のみについてですが、削除された機能が参照されないように、スクリプトの修正が必要になる場合があります。
- AWS Glue 4.0 には EMRFS と Hadoop のアップデートも含まれています。特定のバージョンについては、「付録 A: 注目すべき依存関係のアップグレード」を参照してください。
- ETL ジョブで提供される AWS SDK が 1.11 から 1.12 にアップグレードされました。

- すべての Python ジョブは Python バージョン 3.10 を使用します。以前、AWS Glue 3.0 では、Python 3.7 が使用されていました。

  その結果、AWS Glue に用意されている pymodule の一部はアップグレードされます。
- Log4j は Log4j2 にアップグレードされました。
  - Log4j2 マイグレーションパスについては、「Log4j のドキュメント」を参照してください。
  - カスタム log4j.properties ファイルの名前を log4j2.properties ファイルに変更し、適切な log4j2 プロパティを使用する必要があります。
- 特定のコネクタの移行については、「AWS Glue 4.0 向けのコネクタと JDBC ドライバーの移行」を参照してください。
- AWS Encryption SDK が 1.x から 2.x にアップグレードされました。AWS Glue セキュリティ設定を使用する AWS Glue ジョブや、AWS Encryption SDK の実行時に提供される依存関係に依存するジョブが影響を受けます。AWS Glue ジョブの移行については、次の説明を参照してください。

  AWS Glue 2.0/3.0 には既に AWS Encryption SDK ブリッジバージョンが含まれているため、AWS Glue 2.0/3.0 ジョブを AWS Glue 4.0 ジョブに安全にアップグレードできます。

Spark の移行ドキュメントを参照してください。

- 「Upgrading from Spark SQL 3.1 to 3.2」(Spark SQL 3.1 から 3.2 へのアップグレード)
- 「Upgrading from Spark SQL 3.2 to 3.3」(Spark SQL 3.2 から 3.3 へのアップグレード)

AWS Glue 2.0 から AWS Glue 4.0 への移行

移行時には、次の変更点に注意してください。

> **Note**
>
> AWS Glue 3.0 に関連する移行手順については、「AWS Glue 3.0 から AWS Glue 4.0 への移行」を参照してください。

- AWS Glue 2.0 に存在する既存のすべてのジョブパラメータと主要な機能は、AWS Glue 4.0 に存在します。

- Parquet データを Amazon S3 に書き込むための EMRFS S3 最適化コミッターは、AWS Glue 3.0 以降デフォルトで有効になっています。ただし、`--enable-s3-parquet-optimized-committer` を `false` にすることで無効化することができます。
- AWS Glue 2.0 では、オープンソースの Spark 2.4 を使用していますが、AWS Glue 4.0 では、Amazon EMR で最適化された Spark 3.3.0 を使用しています。
  - 一部の Spark の変更のみについてですが、削除された機能が参照されないように、スクリプトの修正が必要になる場合があります。
  - 例えば、Spark 3.3.0 では Scala 型指定されていない UDF は有効になりませんが、Spark 2.4 では有効になります。
- ETL ジョブで提供される AWS SDK が 1.11 から 1.12 にアップグレードされました。
- また、AWS Glue 4.0 の特徴として、EMRFS への更新や JDBC ドライバーの更新が行われ、AWS Glue により Spark 自体に追加された最適化も組み込まれています。
- Scala は 2.11 から 2.12 に更新されており、Scala 2.12 には Scala 2.11 との下位互換性がありません。
- AWS Glue 2.0 は Python 3.7 と 2.7 しか利用していませんでしたが、Python スクリプトで使用されるデフォルトのバージョンは Python 3.10 になりました。
  - Python 2.7 は、Spark 3.3.0 ではサポートされていません。ジョブ設定で Python 2 をリクエストするジョブはすべて、IllegalArgumentException が発生して失敗します。
  - AWS Glue 2.0 以降、追加の Python モジュールをインストールする新しいメカニズムが利用可能です。
- 依存関係の更新については、付録 A: 注目すべき依存関係のアップグレード を参照してください。
- 既存の AWS Glue 2.0 ジョブで使用されている追加の JAR ファイルは、4.0 で一部の依存関係が 2.0 からアップグレードされたため、依存関係の競合を引き起こす可能性があります。`--user-jars-first` AWS Glue ジョブパラメータによって、AWS Glue 4.0 でのクラスパスの競合を避けることができます。
- AWS Glue 4.0 は Spark 3.3 を使用します。Spark 3.1 以降、parquet ファイルから、または parquet ファイルへのタイムスタンプのロード/保存の動作が変更されています。詳細は「Upgrading from Spark SQL 3.0 to 3.1」を参照してください。

  タイムスタンプ列を含むパーケットデータを読み書きする場合は、次のパラメータを設定することをお勧めします。これらのパラメータを設定すると、Spark 2 から Spark 3 へのアップグレード中に発生するカレンダーの非互換性の問題を、AWS Glue ダイナミックフレームとパークデータフレームの両方で解決することができます。CORRECTED オプションは、datetime 値をそのまま読

み込むために使用し、LEGACY オプションは読み込む際にカレンダーの違いを考慮して datetime 値をリベースするために使用します。

```
- Key: --conf
- Value: spark.sql.legacy.parquet.int96RebaseModeInRead=[CORRECTED|LEGACY] --
conf spark.sql.legacy.parquet.int96RebaseModeInWrite=[CORRECTED|LEGACY] --conf
 spark.sql.legacy.parquet.datetimeRebaseModeInRead=[CORRECTED|LEGACY]
```

- 特定のコネクタの移行については、「AWS Glue 4.0 向けのコネクタと JDBC ドライバーの移行」を参照してください。
- AWS Encryption SDK が 1.x から 2.x にアップグレードされました。AWS Glue セキュリティ設定を使用する AWS Glue ジョブや、AWS Encryption SDK の実行時に提供される依存関係に依存するジョブが影響を受けます。AWS Glue ジョブの移行については、次の説明を参照してください。
  - AWS Glue 2.0 には既に AWS Encryption SDK ブリッジバージョンが含まれているため、AWS Glue 2.0 ジョブを AWS Glue 4.0 ジョブに安全にアップグレードできます。

Spark の移行ドキュメントを参照してください。

- 「Upgrading from Spark SQL 2.4 to 3.0」(Spark SQL 2.4 から 3.0 へのアップグレード)
- 「Upgrading from Spark SQL 3.1 to 3.2」(Spark SQL 3.1 から 3.2 へのアップグレード)
- 「Upgrading from Spark SQL 3.2 to 3.3」(Spark SQL 3.2 から 3.3 へのアップグレード)
- 「Changes in Datetime behavior to be expected since Spark 3.0」(Spark 3.0 以降で予想される日時に関する動作の変化)

AWS Glue 1.0 から AWS Glue 4.0 への移行

移行時には、次の変更点に注意してください。

- AWS Glue 1.0 では、オープンソースの Spark 2.4 を使用していますが、AWS Glue 4.0 では、Amazon EMR で最適化された Spark 3.3.0 を使用しています。
  - 一部の Spark の変更のみについてですが、削除された機能が参照されないように、スクリプトの修正が必要になる場合があります。
  - 例えば、Spark 3.3.0 では Scala 型指定されていない UDF は有効になりませんが、Spark 2.4 では有効になります。

- AWS Glue 4.0 のすべてのジョブの実行におけるスタートアップ時間は、大幅に改善されています。スタートアップのレイテンシーが最大 10 分から最大 1 分になるため、Spark ジョブは 1 秒単位で課金され、最小課金時間は 1/10 になります。
- AWS Glue 4.0 で口グ記録の動作は大幅に変更されました。Spark 3.3.0 では最低でも Log4j2 が必須です。
- 一部の依存関係の更新があり、付録に記載されています。
- Scala も 2.11 から 2.12 に更新されており、Scala 2.12 には Scala 2.11 との下位互換性がありません。
- また、AWS Glue 0.9 は Python 2 しか使用していませんでしたが、Python スクリプトで使用されるデフォルトのバージョンは Python 3.10 になりました。

  Python 2.7 は、Spark 3.3.0 ではサポートされていません。ジョブ設定で Python 2 をリクエストするジョブはすべて、IllegalArgumentException が発生して失敗します。
- AWS Glue 2.0 以降、追加の Python モジュールを pip を通じてインストールする新しいメカニズムが利用可能です。詳細については、「AWS Glue 2.0+ の pip を使用した追加 Python モジュールのインストール」を参照してください。
- AWS Glue 4.0 は Apache YARN では実行されないため、YARN の設定は適用されません。
- AWS Glue 4.0 には、Hadoop 分散ファイルシステム (HDFS) はありません。
- 既存の AWS Glue 1.0 ジョブで使用されている追加の JAR ファイルは、4.0 で一部の依存関係が 1.0 からアップグレードされたため、依存関係の競合を引き起こす可能性があります。この問題を避けるため、デフォルトで `--user-jars-first` AWS Glue ジョブパラメータで AWS Glue 4.0 を有効にしています。
- AWS Glue 4.0 は Auto Scaling をサポートしています。そのため、Auto Scaling が有効になっていると、ExecutorAllocationManager メトリクスが使用可能になります。
- AWS Glue バージョン 4.0 ジョブでは、ワーカーの数とワーカータイプを指定しますが、`maxCapacity` は指定しません。
- AWS Glue 4.0 はまだ機械学習変換をサポートしていません。
- 特定のコネクタの移行については、「AWS Glue 4.0 向けのコネクタと JDBC ドライバーの移行」を参照してください。
- AWS Encryption SDK が 1.x から 2.x にアップグレードされました。AWS Glue セキュリティ設定を使用する AWS Glue ジョブや、AWS Encryption SDK の実行時に提供される依存関係に依存するジョブが影響を受けます。AWS Glue ジョブの移行については、次の説明を参照してください。
  - AWS Glue 0.9/1.0 ジョブを AWS Glue 4.0 ジョブに直接移行することはできません。バージョン 2.x 以降に直接アップグレードしてすべての新機能をすぐに有効にすると、AWS Encryption

SDK では、AWS Encryption SDK の古いバージョンで暗号化された暗号文の復号ができないからです。

- 安全にアップグレードするには、AWS Encryption SDK ブリッジバージョンが含まれている AWS Glue 2.0/3.0 ジョブにまず移行することをお勧めします。ジョブを 1 回実行して AWS Encryption SDK ブリッジバージョンを利用します。
- 完了したら、AWS Glue 2.0/3.0 ジョブを AWS Glue 4.0 に安全に移行できます。

Spark の移行ドキュメントを参照してください。

- 「Upgrading from Spark SQL 2.4 to 3.0」(Spark SQL 2.4 から 3.0 へのアップグレード)
- 「Upgrading from Spark SQL 3.0 to 3.1」(Spark SQL 3.0 から 3.1 へのアップグレード)
- 「Upgrading from Spark SQL 3.1 to 3.2」(Spark SQL 3.1 から 3.2 へのアップグレード)
- 「Upgrading from Spark SQL 3.2 to 3.3」(Spark SQL 3.2 から 3.3 へのアップグレード)
- 「Changes in Datetime behavior to be expected since Spark 3.0」(Spark 3.0 以降で予想される日時に関する動作の変化)

AWS Glue 0.9 から AWS Glue 4.0 への移行

移行時には、次の変更点に注意してください。

- AWS Glue 0.9 では、オープンソースの Spark 2.2.1 を使用していますが、AWS Glue 4.0 では、Amazon EMR で最適化された Spark 3.3.0 を使用しています。
  - 一部の Spark の変更のみについてですが、削除された機能が参照されないように、スクリプトの修正が必要になる場合があります。
  - 例えば、Spark 3.3.0 では Scala 型指定されていない UDF は有効になりませんが、Spark 2.2 では有効になります。
- AWS Glue 4.0 のすべてのジョブの実行におけるスタートアップ時間は、大幅に改善されています。スタートアップのレイテンシーが最大 10 分から最大 1 分になるため、Spark ジョブは 1 秒単位で課金され、最小課金時間は 1/10 になります。
- AWS Glue 4.0 以降、ログ記録の動作は大幅に変更されました。ここ (https://spark.apache.org/docs/latest/core-migration-guide.html#upgrading-from-core-32-to-33) に記載されているように、Spark 3.3.0 では最低でも Log4j2 が必須です。
- 一部の依存関係の更新があり、付録に記載されています。

- Scala も 2.11 から 2.12 に更新されており、Scala 2.12 には Scala 2.11 との下位互換性がありません。
- また、AWS Glue 0.9 は Python 2 しか使用していませんでしたが、Python スクリプトで使用されるデフォルトのバージョンは Python 3.10 になりました。
  - Python 2.7 は、Spark 3.3.0 ではサポートされていません。ジョブ設定で Python 2 をリクエストするジョブはすべて、IllegalArgumentException が発生して失敗します。
  - 追加の Python モジュールを pip を通じてインストールする新しいメカニズムが利用可能です。
- AWS Glue 4.0 は Apache YARN では実行されないため、YARN の設定は適用されません。
- AWS Glue 4.0 には、Hadoop 分散ファイルシステム (HDFS) はありません。
- 既存の AWS Glue 0.9 ジョブで使用されている追加の JAR ファイルは、3.0 で一部の依存関係が 0.9 からアップグレードされたため、依存関係の競合を引き起こす可能性があります。`--user-jars-first` AWS Glue ジョブパラメータによって、AWS Glue 3.0 でのクラスパスの競合を避けることができます。
- AWS Glue 4.0 は Auto Scaling をサポートしています。そのため、Auto Scaling が有効になっていると、ExecutorAllocationManager メトリクスが使用可能になります。
- AWS Glue バージョン 4.0 ジョブでは、ワーカーの数とワーカータイプを指定しますが、`maxCapacity` は指定しません。
- AWS Glue 4.0 はまだ機械学習変換をサポートしていません。
- 特定のコネクタの移行については、「AWS Glue 4.0 向けのコネクタと JDBC ドライバーの移行」を参照してください。
- AWS Encryption SDK が 1.x から 2.x にアップグレードされました。AWS Glue セキュリティ設定を使用する AWS Glue ジョブや、AWS Encryption SDK の実行時に提供される依存関係に依存するジョブが影響を受けます。AWS Glue ジョブの移行については、次の説明を参照してください。
  - AWS Glue 0.9/1.0 ジョブを AWS Glue 4.0 ジョブに直接移行することはできません。バージョン 2.x 以降に直接アップグレードしてすべての新機能をすぐに有効にすると、AWS Encryption SDK では、AWS Encryption SDK の古いバージョンで暗号化された暗号文の復号ができないからです。
  - 安全にアップグレードするには、AWS Encryption SDK ブリッジバージョンが含まれている AWS Glue 2.0/3.0 ジョブにまず移行することをお勧めします。ジョブを 1 回実行して AWS Encryption SDK ブリッジバージョンを利用します。
  - 完了したら、AWS Glue 2.0/3.0 ジョブを AWS Glue 4.0 に安全に移行できます。

Spark の移行ドキュメントを参照してください。

- 「Upgrading from Spark SQL 2.2 to 2.3」(Spark SQL 2.2 から 2.3 へのアップグレード)
- 「Upgrading from Spark SQL 2.3 to 2.4」(Spark SQL 2.3 から 2.4 へのアップグレード)
- 「Upgrading from Spark SQL 2.4 to 3.0」(Spark SQL 2.4 から 3.0 へのアップグレード)
- 「Upgrading from Spark SQL 3.0 to 3.1」(Spark SQL 3.0 から 3.1 へのアップグレード)
- 「Upgrading from Spark SQL 3.1 to 3.2」(Spark SQL 3.1 から 3.2 へのアップグレード)
- 「Upgrading from Spark SQL 3.2 to 3.3」(Spark SQL 3.2 から 3.3 へのアップグレード)
- 「Changes in Datetime behavior to be expected since Spark 3.0」(Spark 3.0 以降で予想される日時に関する動作の変化)

AWS Glue 4.0 向けのコネクタと JDBC ドライバーの移行

アップグレードされた JDBC コネクタとデータレイクコネクタのバージョンについては、次を参照してください。

- 付録 B: JDBC ドライバーのアップグレード
- 付録 C: コネクタのアップグレード

Hudi

- Spark SQL サポートの強化:
  - `Call Procedure` コマンドにより、アップグレード、ダウングレード、ブートストラップ、クリーニング、および修復のサポートが追加されます。Spark SQL で `Create/Drop/Show/Refresh Index` 構文が可能です。
  - Spark DataSource を経由して使用する場合、Spark SQL に対してパフォーマンスのギャップがありましたが、解消されました。過去には、データソースの書き込みは、SQL よりも高速でした。
  - 組み込みキージェネレーターはすべて、より高性能な Spark 固有の API 操作を実装しています。
  - SerDe の使用コストを削減するため、一括 `insert` 操作の UDF 変換を RDD 変換に置き換えました。
  - Hudi の Spark SQL では、SQL ステートメントの `tblproperites` またはオプションで指定される `primaryKey` が必要です。更新と削除のオペレーションには、`preCombineField` も必要です。

- バージョン 0.10.0 より前に `primaryKey` なしで作成された Hudi テーブルは、バージョン 0.10.0 以降、`primaryKey` フィールドつきで再作成する必要があります。

[PostgreSQL]

- いくつかの脆弱性 (CVE) に対応しました。
- Java 8 はネイティブにサポートされています。
- ジョブが配列の配列を使用している場合、バイト配列を除き、この状況は多次元配列として扱うことができます。

MongoDB

- 現在の MongoDB コネクタは Spark バージョン 3.1 以降と MongoDB バージョン 4.0 以降をサポートしています。
- コネクタのアップグレードにより、いくつかのプロパティ名が変更されました。例えば、URI プロパティ名が `connection.uri` に変更されました。現在のオプションの詳細については、「MongoDB Spark Connector blog」(MongoDB Spark コネクタブログ) を参照してください。
- Amazon DocumentDB によってホストされている MongoDB 4.0 を使用することには、いくつかの機能上の違いがあります。詳細については、以下のトピックを参照してください。
  - 機能の違い: Amazon DocumentDB と MongoDB
  - サポートされている MongoDB API、オペレーション、およびデータ型
- 「partitioner」オプションは `ShardedPartitioner`、`PaginateIntoPartitionsPartitioner`、および `SinglePartitionPartitioner` に制限されています。ステージ演算子は MongoDB API をサポートしていないため、Amazon DocumentDB にはデフォルトの `SamplePartitioner` と `PaginateBySizePartitioner` を使用できません。詳細については、「サポートされている MongoDB API、オペレーション、およびデータ型」を参照してください。

Delta Lake

- Delta Lake は SQL でのタイムトラベルをサポートし、古いデータを簡単にクエリできるようになりました。今回の更新により、タイムトラベルは Spark SQL と DataFrame API の両方で利用できるようになりました。SQL の現在のバージョンの TIMESTAMP のサポートが追加されました。

- Spark 3.3 では、バッチクエリに対する `Trigger.Once` と同様に、ストリーミングクエリを実行するための Trigger.AvailableNow が導入されています。このサポートは、Delta テーブルをストリーミングソースとして使用する場合にも利用できます。
- テーブル内の列のリストを返す SHOW COLUMNS のサポート。
- Scala と Python の DeltaTable API での DESCRIBE DETAIL のサポート。DeltaTable API または Spark SQL のいずれかを使用して、Delta テーブルに関する詳細情報を取得します。
- SQL Delete、Merge、および Update コマンドからオペレーションメトリクスを返すことのサポート。以前は、これらの SQL コマンドは空の DataFrame を返していましたが、実行されたオペレーションに関する有用なメトリクスを含む DataFrame を返すようになりました。
- パフォーマンス向上の最適化:
  - Optimize コマンドの設定オプションを `spark.databricks.delta.optimize.repartition.enabled=true` に設定して `coalesce(1)` ではなく `repartition(1)` を使用するようにすると、多数の小さなファイルを圧縮するときのパフォーマンスが向上します。
  - キューベースのアプローチを使用して圧縮ジョブを並列化すると、パフォーマンスが向上します。
- その他の注目すべき変更点:
  - VACUUM および OPTIMIZE SQL コマンドでの変数の使用をサポート。
  - カタログテーブルによる CONVERT TO DELTA の改善:
    - パーティションスキーマが指定されていない場合は、カタログからパーティションスキーマを自動入力します。
    - ディレクトリ全体をスキャンせずに、カタログのパーティション情報を使用してコミットするデータファイルを検索します。テーブルディレクトリ内のすべてのデータファイルをコミットせずに、アクティブなパーティションのディレクトリにあるデータファイルのみがコミットされます。
  - DROP COLUMN と RENAME COLUMN が使用されていない場合、列マッピングが有効なテーブルでの変更データフィード (CDF) バッチ読み取りをサポート。詳細については、Delta Lake のドキュメントを参照してください。
  - 最初のパスでスキーマのプルーニングを有効にすることで、Update コマンドのパフォーマンスを改善。

Apache Iceberg

- スキャンプランニングと Spark クエリのパフォーマンス改善をいくつか追加。

- 変更ベースのコミットを使用してサービス側のコミット競合を解決する一般的な REST カタログクライアントを追加。
- SQL タイムトラベルクエリの AS OF 構文をサポート。
- MERGE クエリと UPDATE クエリの読み取り時マージのサポートを追加。
- Z オーダーを使用してパーティションを書き換えるサポートを追加。
- Theta スケッチやブルームフィルターのような大きな統計やインデックス blob 用の形式である Puffin の仕様と実装を追加。
- データを段階的に使用するための新しいインターフェイスを追加 (追加スキャンと変更ログスキャンの両方)。
- FileIO インターフェイスへの一括オペレーションと範囲読み取りのサポートを追加。
- メタデータツリーに削除ファイルを表示するメタデータテーブルをさらに追加。
- ドロップテーブルの動作を変更。Iceberg 0.13.1 では、DROP TABLE を実行するとカタログからテーブルが削除され、テーブルの内容も削除されます。Iceberg 1.0.0 では、DROP TABLE を実行してもカタログからテーブルが削除されるだけです。テーブルの内容を削除するには、DROP TABLE PURGE を使用します。
- Iceberg 1.0.0 では、Parquet のベクトル化された読み取りがデフォルトで有効になっています。ベクトル化された読み取りを無効にする場合は、`read.parquet.vectorization.enabled` を `false` に設定します。

Oracle

変更は軽微です。

MySQL

変更は軽微です。

Amazon Redshift

AWS Glue 4.0 には、新しい JDBC ドライバーを備えた新しい Amazon Redshift コネクタが搭載されています。機能強化と以前の AWS Glue バージョンからの移行方法については、「the section called “Redshift 接続”」を参照してください。

付録 A: 注目すべき依存関係のアップグレード

依存関係のアップグレードは次のとおりです。

依存関係	AWS Glue 4.0 でのバージョン	AWS Glue 3.0 でのバージョン	AWS Glue 2.0 でのバージョン	AWS Glue 1.0 でのバージョン
Spark	3.3.0-amzn-1	3.1.1-amzn-0	2.4.3	2.4.3
Hadoop	3.3.3-amzn-0	3.2.1-amzn-3	2.8.5-amzn-5	2.8.5-amzn-1
Scala	2.12	2.12	2.11	2.11
Jackson	2.13.3	2.10.x	2.7.x	2.7.x
[Hive]	2.3.9-amzn-2	2.3.7-amzn-4	1.2	1.2
EMRFS	2.54.0	2.46.0	2.38.0	2.30.0
Json4s	3.7.0-M11	3.6.6	3.5.x	3.5.x
Arrow	7.0.0	2.0.0	0.10.0	0.10.0
AWS Glue データカタログクライアント	3.7.0	3.0.0	1.10.0	該当なし
Python	3.10	37	2.7 と 3.6	2.7 と 3.6
Boto	1.26	1.18	1.12	該当なし

付録 B: JDBC ドライバーのアップグレード

JDBC ドライバーのアップグレードは次のとおりです。

ドライバー	過去の AWS Glue バージョンでの JDBC ドライバーのバージョン	AWS Glue 3.0 での JDBC ドライバーのバージョン	AWS Glue 4.0 での JDBC ドライバーのバージョン
MySQL	5.1	8.0.23	8.0.23
Microsoft SQL Server	6.1.0	7.0.0	9.4.0

ドライバー	過去の AWS Glue バージョンでの JDBC ドライバーのバージョン	AWS Glue 3.0 での JDBC ドライバーのバージョン	AWS Glue 4.0 での JDBC ドライバーのバージョン
Oracle Database	11.2	21.1	21.7
[PostgreSQL]	42.1.0	42.2.18	42.3.6
MongoDB	2.0.0	4.0.0	4.7.2
Amazon Redshift	redshift-jdbc41-1.2.12.1017	redshift-jdbc41-1.2.12.1017	redshift-jdbc42-2.1.0.16

付録 C: コネクタのアップグレード

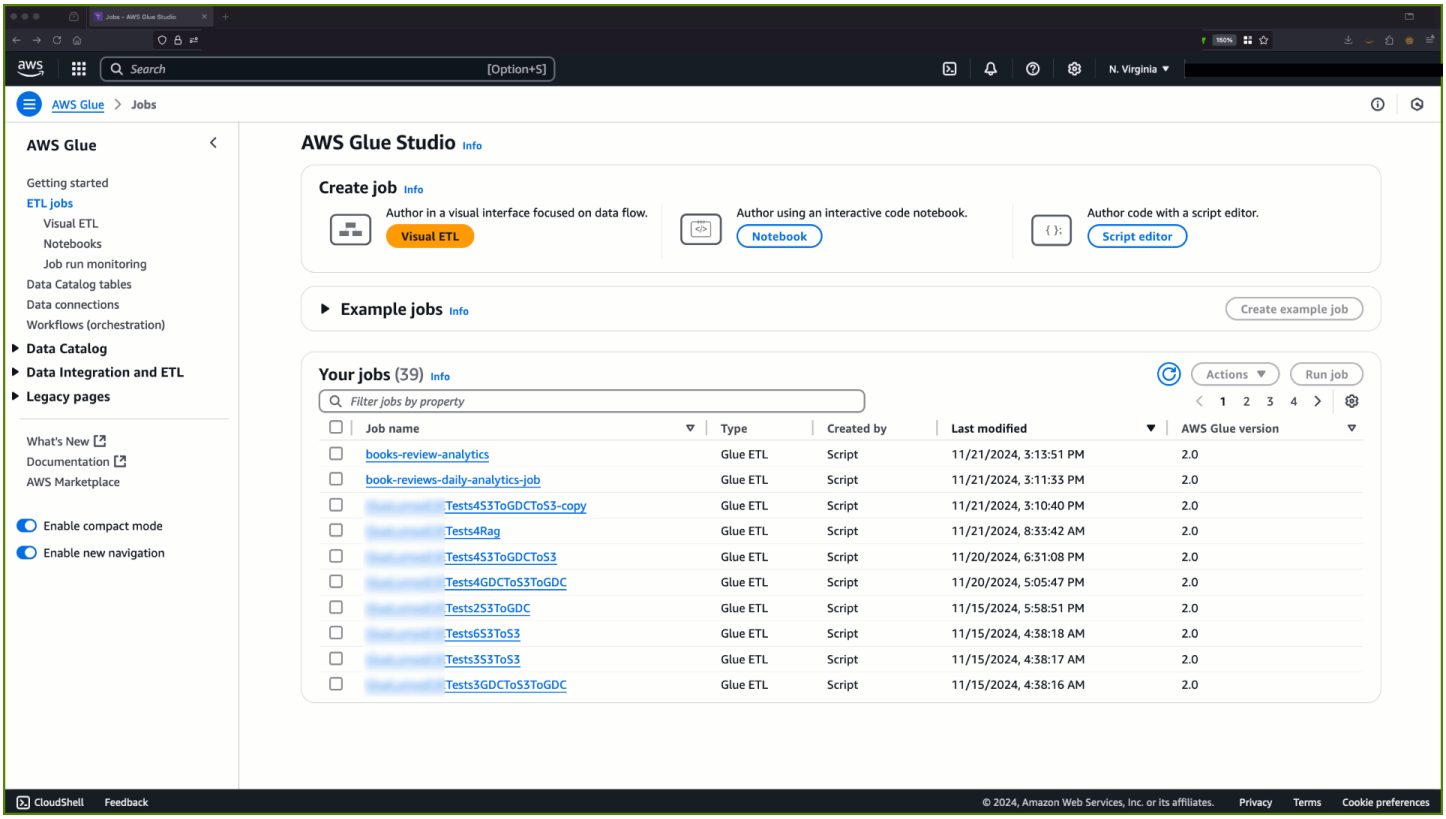
コネクタのアップグレードは次のとおりです。

ドライバー	AWS Glue 3.0 のコネクタバージョン	AWS Glue 4.0 のコネクタバージョン
MongoDB	3.0.0	10.0.4
Hudi	0.10.1	0.12.1
Delta Lake	1.0.0	2.1.0
Iceberg	0.13.1	1.0.0
DynamoDB	1.11	1.12

AWS Glue での Apache Spark の生成 AI アップグレード

Apache Spark プレビューの生成 AI アップグレードは、米国東部 (オハイオ)、米国東部 (バージニア北部)、米国西部 (オレゴン)、アジアパシフィック (東京)、アジアパシフィック (シドニー) の各 AWS リージョンの AWS Glue バージョン 4.0 および 5.0 で利用できます。プレビュー機能は変更される可能性があります。

AWS Glue の Spark アップグレードにより、データエンジニアや開発者は生成 AI を使用して既存の AWS Glue Spark ジョブを最新の Spark リリースにアップグレードして移行できます。データエンジニアはこれを使用して、AWS Glue Spark ジョブのスキャン、アップグレードプランの生成、プランの実行、出力の検証を行うことができます。Spark スクリプト、設定、依存関係、メソッド、機能の特定と更新に関する区別されていない作業を自動化することで、Spark のアップグレードにかかる時間とコストを削減します。



仕組み

アップグレード分析を使用すると、AWS Glue はジョブのコードのバージョンと設定の違いを特定して、アップグレードプランを生成します。アップグレードプランには、すべてのコード変更と必要な移行ステップの詳細が記載されています。次に、AWS Glue はアップグレードされたアプリケーションを環境で構築して実行することで、変更を検証し、ジョブを移行するためのコード変更のリストを生成します。更新されたスクリプトと、提案された変更の詳細を示す概要を表示できます。独自のテストを実行したら、変更を受け入れると、AWS Glue ジョブは新しいスクリプトで最新バージョンに自動的に更新されます。

ジョブとワークロードの複雑さによっては、アップグレード分析プロセスが完了するまでに時間がかかる場合があります。アップグレード分析の結果は、指定された Amazon S3 パスに保存されます。このパスを確認して、アップグレードと潜在的な互換性の問題を理解できます。アップグレード分析

結果を確認したら、アップグレード前に実際のアップグレードを続行するか、ジョブに必要な変更を加えるかを決定できます。

前提条件

生成 AI を使用して AWS Glue のジョブをアップグレードするには、次の前提条件が必要です:

- AWS Glue 2 PySpark ジョブ – AWS Glue 5 にアップグレードできるのは AWS Glue 2 ジョブのみです。
- 分析の開始、結果の確認、ジョブのアップグレードには IAM アクセス許可が必要です。詳細については、下記の アクセス許可 セクションの例を参照してください。
- AWS KMS を使用して分析アーティファクトまたはサービスを暗号化して分析に使用されるデータを暗号化する場合は、追加の AWS KMS アクセス許可が必要です。詳細については、下記の AWS KMS ポリシー セクションの例を参照してください。

アクセス許可

新しいアップグレード分析を開始するには、次のアクセス許可が必要です:

1. 次のアクセス許可を使用して、発信者の IAM ポリシーを更新します:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["glue:StartJobUpgradeAnalysis",
                 "glue:StartJobRun",
                 "glue:GetJobRun",
                 "glue:GetJob",
                 "glue:BatchStopJobRun"
                 ],
      "Resource": [
        "arn:aws:glue:us-east-1:123456789012:job/jobName"
      ]
    },
    {
      "Effect": "Allow",
      "Action": ["s3:GetObject"],
      "Resource": [
        "<s3 script location associated with the job>"
```

```
      ]
    },
    {
      "Effect": "Allow",
      "Action": ["s3:PutObject"],
      "Resource": [
        "<result s3 path provided on API>"
      ]
    },
    {
    "Effect": "Allow",
     "Action": [
       "kms:Decrypt",
       "kms:GenerateDataKey",
      ],
     "Resource": "<key-arn-passed in the API>"
    }
  ]
}
```

> **Note**
>
> 結果アーティファクトの暗号化とサービスメタデータの暗号化の 2 つの異なる AWS KMS キーを使用している場合、ポリシーには両方のキーに同様のポリシーを含める必要があります。

2. アップグレードするジョブの実行ロールを更新して、次のインラインポリシーを含めます:

```
    {
      "Effect": "Allow",
      "Action": ["s3:GetObject"],
      "Resource": [
        "ARN of the Amazon S3 path provided on API",
        "ARN of the Amazon S3 path provided on API/*"
      ]
    }
```

例えば、Amazon S3 パス `s3://amzn-s3-demo-bucket/upgraded-result` を使用している場合、ポリシーは次のようになります:

```
{
```

```
      "Effect": "Allow",
      "Action": ["s3:GetObject"],
      "Resource": [
        "arn:aws:s3:::amzn-s3-demo-bucket/upgraded-result/",
        "arn:aws:s3:::amzn-s3-demo-bucket/upgraded-result/*"
      ]
    }
```

分析の詳細を取得するには、次のアクセス許可が必要です:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["glue:GetJobUpgradeAnalysis"],
      "Resource": [
        "arn:aws:glue:us-east-1:123456789012:job/jobName"
      ]
    }
  ]
}
```

進行中の分析を停止するには、次のアクセス許可が必要です:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["glue:StopJobUpgradeAnalysis",
                 "glue:BatchStopJobRun"
                 ],
      "Resource": [
        "arn:aws:glue:us-east-1:123456789012:job/jobName"
      ]
    }
  ]
}
```

特定のジョブで送信されたすべての分析を一覧表示するには、次のアクセス許可が必要です:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["glue:ListJobUpgradeAnalyses"],
      "Resource": [
        "arn:aws:glue:us-east-1:123456789012:job/jobName"
      ]
    }
  ]
}
```

分析からの変更を受け入れてジョブをアップグレードするには、次のアクセス許可が必要です:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["glue:UpdateJob",
                 "glue:UpgradeJob"
                 ],
      "Resource": [
        "arn:aws:glue:us-east-1:123456789012:job/jobName"
      ]
    },
    {
      "Effect": "Allow",
      "Action": ["iam:PassRole"],
      "Resource": [
        "<Role arn associated with the job>"
      ]
    }
  ]
}
```

AWS KMS ポリシー

分析の開始時に独自のカスタム AWS KMS キーを渡すには、次のセクションを参照して、AWS KMS キーに対する適切なアクセス許可を設定してください。

サービスメタデータ暗号化 AWS KMS キーの設定:

キーを渡すにはアクセス許可 (暗号化/復号化) が必要です。以下のポリシー例では、<IAM Customer caller ARN> が指定した AWS アカウントまたはロールが、許可されたアクションを実行できます。

- kms:Decrypt は、指定された AWS KMS キーを使用した復号を許可します。
- kms:GenerateDataKey では、指定された AWS KMS キーを使用してデータキーを生成できます。

```
{
    "Effect": "Allow",
    "Principal":{
        "AWS": "<IAM Customer caller ARN>"
    },
    "Action": [
      "kms:Decrypt",
      "kms:GenerateDataKey",
    ],
    "Resource": "<key-arn-passed-on-start-api>"
}
```

AWS KMS キーの暗号化と復号化の両方にキーを使用するには、AWS Glue にアクセス許可を付与する必要があります。

```
{
    "Effect": "Allow",
    "Principal":{
       "Service": "glue.amazonaws.com"
     },
     "Action": [
       "kms:Decrypt",
       "kms:GenerateDataKey",
      ],
     "Resource": "<key-arn>",
     "Condition": {
       "StringLike": {
         "aws:SourceArn": "arn:aws:glue:<region>:<aws_account_id>:job/job-name"
       }
     }
 }
```

AWS KMS キーを使用した結果アーティファクトの暗号化の設定:

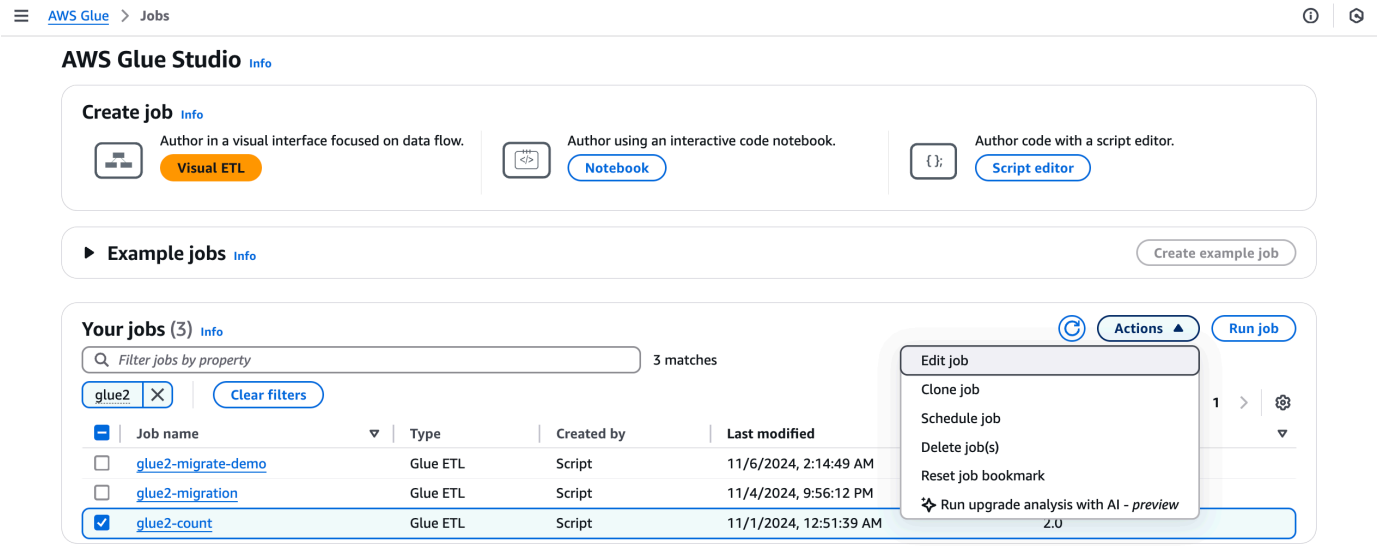
このポリシーにより、AWS KMS キーに対する暗号化と復号の両方のアクセス許可が付与されます。

```
{
    "Effect": "Allow",
    "Principal":{
        "AWS": "<IAM Customer caller ARN>"
    },
    "Action": [
      "kms:Decrypt",
      "kms:GenerateDataKey",
    ],
    "Resource": "<key-arn-passed-on-start-api>"
}
```

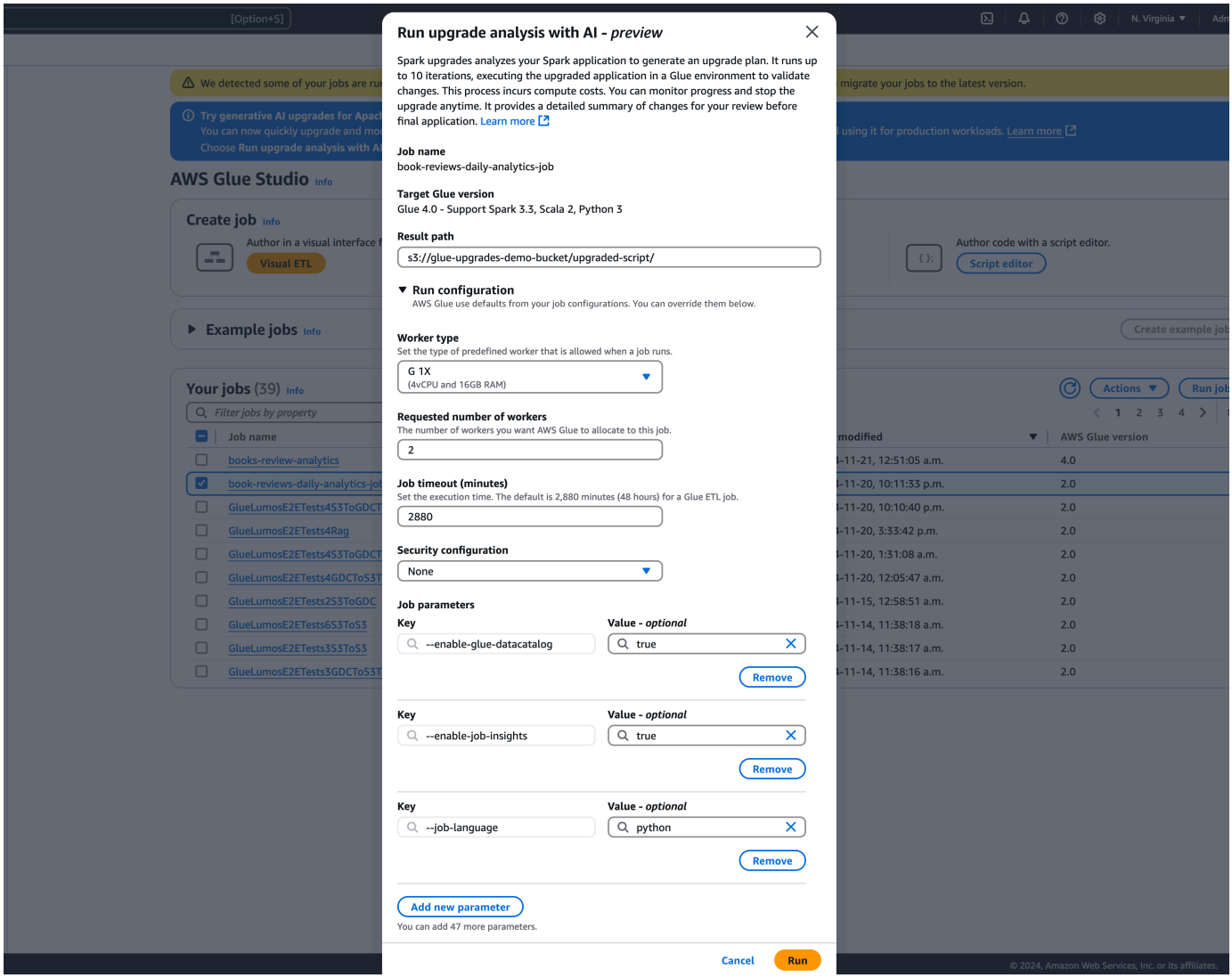
アップグレード分析の実行とアップグレードスクリプトの適用

アップグレード分析を実行すると、[ジョブ] ビューから選択したジョブでアップグレードプランが生成されます。

1. [ジョブ] から AWS Glue 2.0 ジョブを選択し、[アクション] メニューから [アップグレード分析の実行] を選択します。



2. モーダルで、生成されたアップグレードプランを [結果パス] に保存するためのパスを選択します。これは、アクセスおよび書き込みが可能な Amazon S3 バケットである必要があります。



3. 必要に応じて、追加オプションを設定します:

- 実行設定 – オプション: 実行設定は、アップグレード分析中に実行される検証実行のさまざまな側面をカスタマイズできるオプション設定です。この設定は、アップグレードされたスクリプトの実行に使用され、コンピューティング環境のプロパティ (ワーカータイプ、ワーカー数など) を選択できます。変更を確認して受け入れ、本番環境に適用する前に、本番稼働用以外の開発者アカウントを使用してサンプルデータセットで検証を実行する必要があります。実行設定には、以下のカスタマイズ可能なパラメータが含まれます:
  - ワーカータイプ: 検証の実行に使用するワーカーのタイプを指定できるため、要件に基づいて適切なコンピューティングリソースを選択できます。
  - ワーカーの数: 検証実行用にプロビジョニングするワーカーの数を定義できるため、ワークロードのニーズに応じてリソースをスケーリングできます。

- ジョブタイムアウト (分単位): このパラメータを使用すると、検証実行の時間制限を設定して、リソースの過剰な消費を防ぐために、指定された期間後にジョブが終了するようにできます。
- セキュリティ設定: 暗号化やアクセスコントロールなどのセキュリティ設定を設定して、検証実行中のデータやリソースを確実に保護できます。
- 追加のジョブパラメータ: 必要に応じて、新しいジョブパラメータを追加して、検証実行の実行環境をさらにカスタマイズできます。

実行設定を活用することで、特定の要件に合わせて検証実行を調整できます。例えば、より小さなデータセットを使用するように検証実行を設定できます。これにより、分析をより迅速に完了し、コストを最適化できます。この方法により、検証フェーズ中のリソース使用率と関連コストを最小限に抑えながら、アップグレード分析を効率的に実行できます。

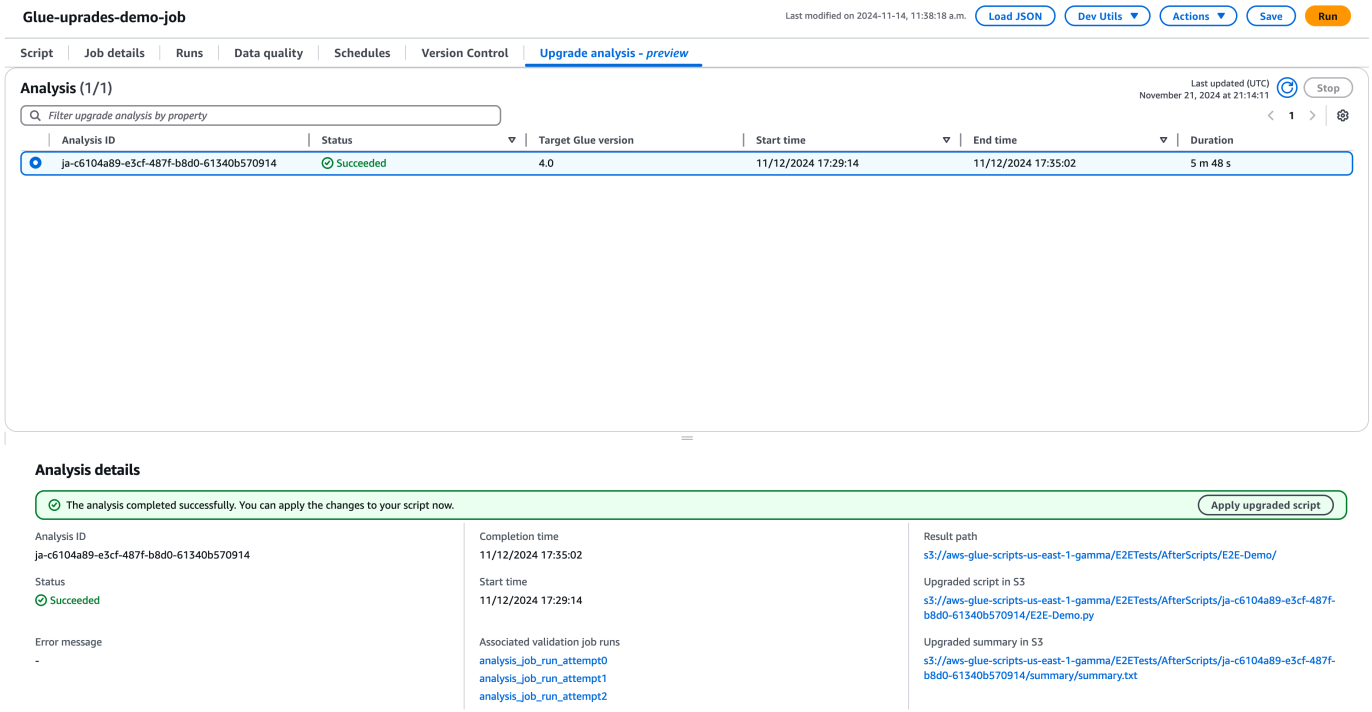
- 暗号化設定 – オプション:
  - アップグレードアーティファクトの暗号化を有効にする: 結果パスにデータを書き込むときに、保存時の暗号化を有効にします。アップグレードアーティファクトを暗号化しない場合は、このオプションはオフのままにします。
  - サービスメタデータの暗号化をカスタマイズする: サービスメタデータは、デフォルトで AWS 所有キーを使用して暗号化されます。暗号化に独自のキーを使用する場合は、このオプションを選択します。

4. [実行] を選択してアップグレード分析を開始します。分析の実行中に、アップグレード分析タブで結果を表示できます。分析の詳細ウィンドウには、分析に関する情報とアップグレードアーティファクトへのリンクが表示されます。

   - 結果パス — 結果の概要とアップグレードスクリプトが格納されます。
   - Amazon S3 のアップグレードされたスクリプト – Amazon S3 のアップグレードスクリプトの場所。アップグレードを適用する前にスクリプトを表示できます。
   - Amazon S3 のアップグレード概要 – Amazon S3 のアップグレード概要の場所。アップグレードを適用する前に、アップグレードの概要を表示できます。

5. アップグレード分析が正常に完了したら、アップグレードスクリプトを適用して、[アップグレードスクリプトを適用] を選択することでジョブを自動的にアップグレードできます。

   適用すると、AWS Glue バージョンが 4.0 に更新されます。新しいスクリプトは、[スクリプト] タブに表示されます。



アップグレードの概要について

この例では、AWS Glue ジョブをバージョン 2.0 からバージョン 4.0 にアップグレードするプロセスを示します。サンプルジョブは Amazon S3 バケットから製品データを読み取り、Spark SQL を使用してデータにいくつかの変換を適用し、変換された結果を Amazon S3 バケットに保存します。

元のコード (AWS Glue 2.0) - アップグレード前

```
from awsglue.transforms import *
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from pyspark.sql.types import *
from pyspark.sql.functions import *
from awsglue.job import Job
import json
from pyspark.sql.types import StructType

sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
```

```
gdc_database = "s3://aws-glue-scripts-us-east-1-gamma/demo-database/"
schema_location = (
    "s3://aws-glue-scripts-us-east-1-gamma/DataFiles/"
)

products_schema_string = spark.read.text(
    f"{schema_location}schemas/products_schema"
).first()[0]

product_schema = StructType.fromJson(json.loads(products_schema_string))

products_source_df = (
    spark.read.option("header", "true")
    .schema(product_schema)
    .option(
        "path",
        f"{gdc_database}products/",
    )
    .csv(f"{gdc_database}products/")
)

products_source_df.show()
products_temp_view_name = "spark_upgrade_demo_product_view"
products_source_df.createOrReplaceTempView(products_temp_view_name)

query = f"select {products_temp_view_name}.*, format_string('%0$s-%0$s', category,
 subcategory) as unique_category from {products_temp_view_name}"
products_with_combination_df = spark.sql(query)
products_with_combination_df.show()

products_with_combination_df.createOrReplaceTempView(products_temp_view_name)
product_df_attribution = spark.sql(
    f"""
SELECT *,
unbase64(split(product_name, ' ')[0]) as product_name_decoded,
unbase64(split(unique_category, '-')[1]) as subcategory_decoded
FROM {products_temp_view_name}
"""
)
product_df_attribution.show()


product_df_attribution.write.mode("overwrite").option("header", "true").option(
    "path", f"{gdc_database}spark_upgrade_demo_product_agg/"
```

```
).saveAsTable("spark_upgrade_demo_product_agg", external=True)

spark_upgrade_demo_product_agg_table_df = spark.sql(
    f"SHOW TABLE EXTENDED in default like 'spark_upgrade_demo_product_agg'"
)
spark_upgrade_demo_product_agg_table_df.show()
job.commit()
```

新しいコード (Glue 4.0) - アップグレード後

```
from awsglue.transforms import *
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from pyspark.sql.types import *
from pyspark.sql.functions import *
from awsglue.job import Job
import json
from pyspark.sql.types import StructType

sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
# change 1
spark.conf.set("spark.sql.adaptive.enabled", "false")
# change 2
spark.conf.set("spark.sql.legacy.pathOptionBehavior.enabled", "true")
job = Job(glueContext)

gdc_database = "s3://aws-glue-scripts-us-east-1-gamma/demo-database/"
schema_location = (
    "s3://aws-glue-scripts-us-east-1-gamma/DataFiles/"
)

products_schema_string = spark.read.text(
    f"{schema_location}schemas/products_schema"
).first()[0]

product_schema = StructType.fromJson(json.loads(products_schema_string))

products_source_df = (
    spark.read.option("header", "true")
    .schema(product_schema)
    .option(
```

```
        "path",
        f"{gdc_database}products/",
    )
    .csv(f"{gdc_database}products/")
)

products_source_df.show()
products_temp_view_name = "spark_upgrade_demo_product_view"
products_source_df.createOrReplaceTempView(products_temp_view_name)

# change 3
query = f"select {products_temp_view_name}.*, format_string('%1$s-%1$s', category,
 subcategory) as unique_category from {products_temp_view_name}"
products_with_combination_df = spark.sql(query)
products_with_combination_df.show()

products_with_combination_df.createOrReplaceTempView(products_temp_view_name)
# change 4
product_df_attribution = spark.sql(
    f"""
SELECT *,
try_to_binary(split(product_name, ' ')[0], 'base64') as product_name_decoded,
try_to_binary(split(unique_category, '-')[1], 'base64') as subcategory_decoded
FROM {products_temp_view_name}
"""
)
product_df_attribution.show()


product_df_attribution.write.mode("overwrite").option("header", "true").option(
    "path", f"{gdc_database}spark_upgrade_demo_product_agg/"
).saveAsTable("spark_upgrade_demo_product_agg", external=True)

spark_upgrade_demo_product_agg_table_df = spark.sql(
    f"SHOW TABLE EXTENDED in default like 'spark_upgrade_demo_product_agg'"
)
spark_upgrade_demo_product_agg_table_df.show()
job.commit()
```

分析の概要の説明

```
In Spark 3.2, spark.sql.adaptive.enabled is enabled by default. To restore the behavior before
Spark 3.2,
you can set spark.sql.adaptive.enabled to false.


In Spark 3.1, path option cannot coexist when the following methods are called with path
parameter(s): DataFrameReader.load(), DataFrameWriter.save(), DataStreamReader.load(), or
DataStreamWriter.start(). In addition, paths option cannot coexist for DataFrameReader.load().
For example, spark.read.format(csv).option(path, /tmp).load(/tmp2) or spark.read.option(path, /
tmp).csv(/tmp2) will throw org.apache.spark.sql.AnalysisException. In Spark version 3.0 and
below, path option is overwritten if one path parameter is passed to above methods; path option
is added to the overall paths if multiple path parameters are passed to DataFrameReader.load().
To restore the behavior before Spark 3.1, you can set
spark.sql.legacy.pathOptionBehavior.enabled to true.


Since Spark 3.3, the `strfmt` in `format_string(strfmt, obj, ...)` and `printf(strfmt,
obj, ...)` will no longer support to use 0$ to specify the first argument, the first argument
should always reference by 1$ when use argument index to indicating the position of the argument
in the argument list.


Since Spark 3.3, the unbase64 function throws error for a malformed str input. Use
try_to_binary(<str>, 'base64') to tolerate malformed input and return NULL instead. In Spark 3.2
and earlier, the unbase64 function returns a best-efforts result for a malformed str input.
```

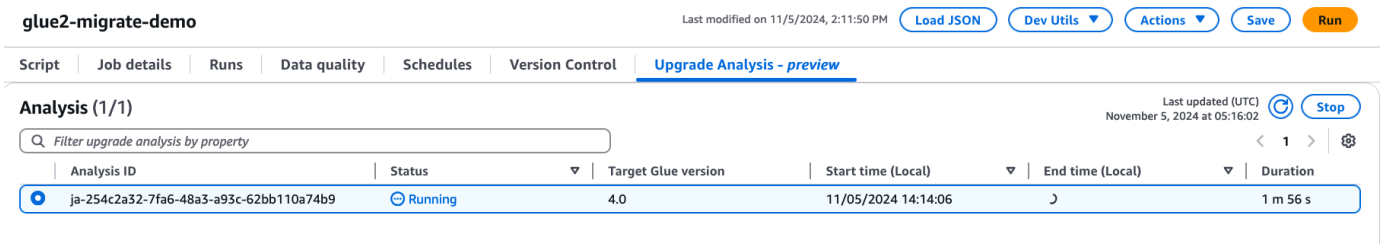
概要に基づいて、スクリプトを AWS Glue 2.0 から AWS Glue 4.0 に正常にアップグレードするために、AWS Glue が提案した 4 つの変更があります。

1. Spark SQL 設定 (spark.sql.adaptive.enabled): この変更は、Spark SQL アダプティブクエリ実行の新しい機能が Spark 3.2 から導入されたため、アプリケーションの動作を復元するためのものです。この設定の変更を検査し、必要に応じてさらに有効または無効にできます。
2. DataFrame API の変更: パスオプションは、`load()` などの他の DataFrameReader オペレーションと共存できません。前の動作を維持するために、AWS Glue はスクリプトを更新して新しい SQL 設定 (spark.sql.legacy.pathOptionBehavior.enabled) を追加しました。
3. Spark SQL API の変更: 最初の引数として `0$` を禁止するように、`format_string(strfmt, obj, ...)` の `strfmt` の動作が更新されました。互換性を確保するために、AWS Glue は代わりに最初の引数として `1$` を使用するようにスクリプトを変更しました。
4. Spark SQL API の変更: `unbase64` 関数は不正な形式の文字列入力を許可しません。前の動作を維持するために、AWS Glue は `try_to_binary` 関数を使用するようにスクリプトを更新しました。

進行中のアップグレード分析の停止

進行中のアップグレード分析をキャンセルすることも、分析を停止することもできます。

1. [アップグレード分析] タブを選択します。
2. 実行中のジョブを選択し、[停止] を選択します。これにより、分析が停止します。その後、同じジョブで別のアップグレード分析を実行できます。



考慮事項

プレビュー期間中に Spark Upgrades の使用を開始する際には、サービスの最適な使用について考慮すべき重要な点がいくつかあります。

- サービスの範囲と制限: プレビューリリースでは、PySpark コードを AWS Glue バージョン 2.0 からバージョン 5.0 にアップグレードすることに重点を置いています。現時点では、サービスは追加のライブラリ依存関係に依存しない PySpark コードを処理します。AWS アカウントで最大 10 個のジョブの自動アップグレードを同時に実行できるため、システムの安定性を維持しながら複数のジョブを効率的にアップグレードできます。
  - PySpark ジョブのみがサポートされています。
  - アップグレード分析は 24 時間後にタイムアウトします。
  - 1 つのジョブに対して一度に実行できるアクティブなアップグレード分析は 1 つだけです。アカウントレベルでは、最大 10 個のアクティブなアップグレード分析を同時に実行できます。
- アップグレードプロセス中のコストの最適化: Spark アップグレードでは生成 AI を使用して複数の反復でアップグレードプランを検証するため、各反復はアカウントで AWS Glue ジョブとして実行されるため、コスト効率を高めるためには、検証ジョブの実行設定を最適化することが不可欠です。これを実現するには、アップグレード分析を開始するときに次のように実行設定を指定することをお勧めします:
  - 本番稼働用以外のデベロッパーアカウントを使用して、本番稼働用データを表すが、Spark Upgrades での検証にはサイズが小さいサンプルモックデータセットを選択します。

  - G.1X ワーカーなどの適切なサイズのコンピューティングリソースを使用し、サンプルデータを処理するための適切なワーカー数を選択します。
  - 該当する場合は AWS Glue ジョブの自動スケーリングを有効にして、ワークロードに応じてリソースを自動的に調整します。

  例えば、本番稼働用ジョブが 20 G.2X ワーカーでテラバイトのデータを処理する場合、アップグレードジョブを、2 G.2X ワーカーで数ギガバイトの代表的なデータを処理するように設定し、検証のために自動スケーリングを有効にする可能性があります。
- プレビューのベストプラクティス: プレビュー期間中は、非本番環境のジョブでアップグレードジャーニーを開始することを強くお勧めします。この方法により、アップグレードワークフローに慣れ、サービスがさまざまなタイプの Spark コードパターンをどのように処理するかを理解することができます。
- アラームと通知: ジョブで生成 AI アップグレード機能を利用する場合は、失敗したジョブ実行のアラーム/通知がオフになっていることを確認します。アップグレードプロセス中に、アップグレードされたアーティファクトが提供される前に、アカウントで最大 10 件のジョブ実行が失敗することがあります。
- 異常検出ルール: アップグレード中のジョブでも異常検出ルールをオフにします。中間ジョブの実行中に出力フォルダに書き込まれたデータが、アップグレード検証が進行している間、想定どおりの形式ではない可能性があります。
- べき等ジョブでアップグレード分析を使用する: べき等ジョブでアップグレード分析を使用して、後続の各検証ジョブ実行試行が前のものと類似し、問題が発生しないようにします。べき等ジョブは、同じ入力データで複数回実行できるジョブであり、毎回同じ出力を生成します。AWS Glue で Apache Spark の生成 AI アップグレードを使用する場合、サービスは検証プロセスの一環としてジョブの反復を複数回実行します。各イテレーション中に、Spark コードと設定を変更してアップグレードプランを検証します。Spark ジョブが「べき等」でない場合、同じ入力データで複数回実行すると問題が発生する可能性があります。

Spark アップグレードのクロスリージョン推論

Spark アップグレードは Amazon Bedrock を利用しており、クロスリージョン推論 (CRIS) を活用します。CRIS により、Spark アップグレードは地理的に最適なリージョンを自動的に選択して推論リクエストを処理し (詳細はこちら)、利用可能なコンピューティングリソースとモデルの可用性を最大化し、最高のカスタマーエクスペリエンスを提供します。クロスリージョン推論の使用に追加コストはかかりません。

クロスリージョン推論リクエストは、データが最初に存在していた地域の一部である AWS リージョン内に保持されます。例えば、米国内で行われたリクエストは、米国の AWS リージョン内に保持されます。データはプライマリリージョンにのみ保存されますが、クロスリージョン推論を使用する場合、入力プロンプトと出力結果がプライマリリージョン外に移動することがあります。すべてのデータは Amazon の安全なネットワーク経由で暗号化されて送信されます。

AWS Glue での Spark ジョブの使用

AWS Glue for Spark の ETL ジョブについて説明します。

トピック

- AWS Glue ジョブでジョブパラメータを使用する
- AWS GlueSpark と PySpark のジョブ
- AWS Glue でのストリーミング ETL ジョブ
- AWS Lake Formation FindMatches によるレコードのマッチング
- Apache Spark プログラムを AWS Glue に移行する

AWS Glue ジョブでジョブパラメータを使用する

AWS Glue ジョブを作成するときは、`Role` や `WorkerType` などの標準フィールドを設定します。`[Argument]` フィールド (コンソールの [ジョブパラメータ]) で追加の設定情報を指定できます。これらのフィールドでは、AWS Glue ジョブに対して、このトピックに記載されている引数 (パラメータ) を指定できます。

AWS Glue Job API の詳細については、「the section called “ジョブ”」を参照してください。

> **Note**
>
> ジョブ引数の最大サイズの制限は 260KB です。検証チェックでは、引数のサイズが 260KB を超える場合にエラーが発生します。

ジョブのパラメータを設定する

コンソールの [Job details] (ジョブの詳細) タブの [Job Parameters] (ジョブパラメータ) の見出しの下でジョブを設定できます。また、AWS CLI で、ジョブに `DefaultArguments` または

NonOverridableArguments を設定するか、ジョブ実行に Arguments を設定することで、ジョブを設定できます。ジョブに設定された引数は、ジョブが実行されるたびに渡されますが、ジョブの実行に設定された引数は、その個々の実行に対してのみ渡されます。

次の例は、ジョブのパラメータを設定するために --arguments を使用してジョブを実行する構文です。

```
$ aws glue start-job-run --job-name "CSV to CSV" --arguments='--scriptLocation="s3://
my_glue/libraries/test_lib.py"'
```

ジョブのパラメータへのアクセス

AWS Glue スクリプトを作成する際に、ジョブのパラメータ値にアクセスして作成したコードの動作を変更したい場合があります。そのためのヘルパーメソッドをライブラリに用意しています。これらのメソッドは、ジョブパラメータ値をオーバーライドするジョブ実行パラメータ値を解決します。複数の場所で設定されたパラメータを解決する場合、ジョブの NonOverridableArguments はジョブ実行の Arguments をオーバーライドしますが、ジョブの DefaultArguments はオーバーライドされます。

Python では:

Python ジョブでは、getResolvedParameters という名前の関数を用意しています。詳細については、「the section called “getResolvedOptions”」を参照してください。ジョブパラメータは sys.argv 変数から取得できます。

Scala では:

Scala ジョブでは、GlueArgParser という名前のオブジェクトを用意しています。詳細については、「the section called “GlueArgParser”」を参照してください。ジョブパラメータは sysArgs 変数から取得できます。

ジョブパラメータリファレンス

AWS Glue は、ジョブおよびジョブ実行のスクリプト環境を設定するために使用できる次の引数名を認識します。

--additional-python-modules

インストールする Python パッケージのセットを表すカンマ区切りのリスト。パッケージのインストールは、PyPI からでも、カスタムディストリビューションを提供することでも行

えます。PyPI パッケージのエントリは、ターゲットパッケージの PyPI 名とバージョンを含む、*package*==*version* のような形式になります。カスタムディストリビューションエントリとは S3 のパスであり、ディストリビューションを指しています。

エントリは Python のバージョンマッチングを使用して、パッケージとバージョンの照合を行います。つまり、2 つの等号 (==) を使用する必要があります。バージョンマッチングには他の演算子もあります。詳細については「PEP 440」を参照してください。

モジュールインストールオプションを `pip3` に渡すには、--python-modules-installer-option パラメータを使用します。

--auto-scale-within-microbatch

デフォルト値は True です。このパラメータは、ストリーミングデータを一連のマイクロバッチで処理する AWS Glue ストリーミングジョブにのみ使用でき、自動スケーリングを有効にする必要があります。この値を false に設定すると、完了したマイクロバッチのバッチ持続時間の指数平滑移動平均線を計算し、この値をウィンドウサイズと比較して、エグゼキューターの数をスケールアップするかスケールダウンするかを決定します。スケーリングはマイクロバッチが完了したときにのみ行われます。この値を true に設定すると、マイクロバッチの実行中に Spark タスクの数が 30 秒間変わらない場合や、現在のバッチ処理がウィンドウサイズを超えると、スケールアップします。エグゼキューターが 60 秒以上アイドル状態であったり、バッチ持続時間の指数平滑移動平均線が短い場合、エクゼキューターの数は減少します。

--class

Scala スクリプトのエントリポイントとなる Scala クラス。これは、`--job-language` を `scala` に設定した場合にのみ適用されます。

--continuous-log-conversionPattern

連続ログ記録を有効にしたジョブのカスタム変換ログパターンを指定します。変換パターンは、ドライバーログとエグゼキューターログにのみ適用されます。この設定は、AWS Glue の進行状況バーには影響しません。

--continuous-log-logGroup

連続ログ記録が有効なジョブに対し、カスタム Amazon CloudWatch ロググループ名を指定します。

--continuous-log-logStreamPrefix

連続ログ記録が有効なジョブに対し、カスタム CloudWatch ログストリームのプレフィックスを指定します。

--customer-driver-env-vars および --customer-executor-env-vars

これらのパラメータは、各ワーカー (ドライバーまたはエグゼキューター) のオペレーティングシステムの環境変数をそれぞれ設定します。AWS Glue 上でプラットフォームやカスタムフレームワークを構築するときにこれらのパラメータを使用し、ユーザーが Glue 上にジョブを作成できるようします。これら 2 つのフラグを有効にすると、ジョブスクリプト自体に同じロジックを注入しなくても、ドライバーおよびエグゼキューターにそれぞれ異なる環境変数を設定できます。

使用例

次の内容は、これらのパラメータを使用する例を示します。

```
"–customer-driver-env-vars", "CUSTOMER_KEY1=VAL1,CUSTOMER_KEY2=\"val2,val2 val2\"",
"–customer-executor-env-vars", "CUSTOMER_KEY3=VAL3,KEY4=VAL4"
```

これらをジョブ実行の引数に設定することは、次のコマンドを実行することと同じです。

ドライバーに含まれるもの:

- export CUSTOMER_KEY1=VAL1
- export CUSTOMER_KEY2="val2,val2 val2"

エグゼキューターに含まれるもの:

- export CUSTOMER_KEY3=VAL3

その後、ジョブスクリプト自体で `os.environ.get("CUSTOMER_KEY1")` または `System.getenv("CUSTOMER_KEY1")` を使用して環境変数を取得できます。

強制構文

環境変数を定義するとき、次の基準に従ってください。

- 各キーには、`CUSTOMER_ prefix` が必要です。

  たとえば `"CUSTOMER_KEY3=VAL3,KEY4=VAL4"` の場合、`KEY4=VAL4` は無視されて設定されません。
- キーおよび値のペアは、それぞれ 1 つのカンマで区切る必要があります。

  例: `"CUSTOMER_KEY3=VAL3,CUSTOMER_KEY4=VAL4"`
- 「値」にスペースやカンマが含まれている場合、引用符で囲んで定義する必要があります。

  例: `CUSTOMER_KEY2=\"val2,val2 val2\"`

この構文は、bash 環境変数の設定基準を忠実に模倣しています。

--datalake-formats

AWS Glue 3.0 以降のバージョンでサポートされます。

使用するデータレイクフレームワークを指定します。AWSGlue は、指定したフレームワークに必要な JAR ファイルを classpath に追加します。詳細については、「AWS Glue ETL ジョブでのデータレイクフレームワークの使用」を参照してください。

次の中から 1 つまたは複数の値をカンマで区切って指定できます。

- hudi
- delta
- iceberg

例えば、3 つのフレームワークすべてを指定するには、次の引数を渡します。

```
'--datalake-formats': 'hudi,delta,iceberg'
```

--disable-proxy-v2

サービスプロキシを無効にして、VPC を介してスクリプトから由来する Amazon S3、CloudWatch、AWS Glue への AWS サービス呼び出しを許可します。詳細については、「すべての AWS コールを VPC を経由するように設定する」を参照してください。サービスプロキシを無効にするには、このパラメータの値を true に設定します。

--enable-auto-scaling

値を true に設定した場合、Auto Scaling およびワーカーごとの課金がオンになります。

--enable-continuous-cloudwatch-log

AWS Glue ジョブのリアルタイムの連続ログ記録を有効にします。Apache Spark ジョブに関するリアルタイムのログは、CloudWatch で表示できます。

--enable-continuous-log-filter

連続ログ記録が有効であるジョブを作成または編集するときに、標準フィルタ (true) またはフィルタなし (false) を指定します。標準フィルタを選択すると、無用な Apache Spark ドライバー/エグゼキュータや Apache Hadoop YARN ハートビートのログメッセージは除外されます。フィルタなしを選択すると、すべてのログメッセージが表示されます。

--enable-glue-datacatalog

AWS Glue データカタログの、Apache Spark Hive メタストアとしての使用を有効にします。この機能を有効にするには、値を `true` に設定します。

--enable-job-insights

AWS Glue のジョブ実行インサイトによる追加のエラー分析のモニタリングを有効にします。詳細については、「the section called “AWS Glue ジョブ実行インサイトでのモニタリング”」を参照してください。デフォルトでは、値は `true` に設定されており、ジョブ実行インサイトは有効になっています。

このオプションは、AWS Glue バージョン 2.0 および 3.0 で利用できます。

--enable-lakeformation-fine-grained-access

Glue AWS ジョブのきめ細かなアクセスコントロールを有効にします。詳細については、「the section called “FGAC のための Lake Formation”」を参照してください。

--enable-metrics

このジョブの実行のジョブプロファイリングに関するメトリクスの収集を有効にします。これらのメトリクスは、AWS Glue コンソールおよび Amazon CloudWatch コンソールから利用が可能です。このパラメータの値は関係ありません。この機能を有効にするには、このパラメータに任意の値を指定できますが、わかりやすくするために `true` が推奨されます。この機能を無効にするには、このパラメータをジョブ設定から削除します。

--enable-observability-metrics

AWS Glue コンソールおよび Amazon CloudWatch コンソールの「ジョブ実行モニタリング」ページで、各ジョブ実行内で起こっている内容に関する情報を生成するため、一連のオブザーバビリティメトリクスを有効にします。この機能を有効にするには、パラメータの値を true に設定します。この機能を無効にするには、`false` に設定するか、ジョブ設定からこのパラメーターを削除します。

--enable-rename-algorithm-v2

EMRFS 名前変更アルゴリズムのバージョンをバージョン 2 に設定します。Spark ジョブがパーティションの動的な上書きモードを使用している場合、パーティションが重複して作成される可能性があります。例えば、`s3://bucket/table/location/p1=1/p1=1` のような重複したパーティション作成されます。ここで、P1 は上書きされているパーティションです。名前変更アルゴリズムのバージョン 2 では、この問題が修正されています。

このオプションは、AWS Glue バージョン 1.0 でのみ使用できます。

--enable-s3-parquet-optimized-committer

Parquet データを Amazon S3 に書き込むために、EMRFS S3 最適化コミッターを有効にします。AWS Glue ジョブの作成または更新時に、AWS Glue コンソールからパラメータ/値のペアを指定できます。値を **true** に設定すると、コミッターが有効になります。デフォルトでは、このフラグは AWS Glue 3.0 ではオン、AWS Glue 2.0 ではオフになっています。

詳細については、「EMRFS S3 向けに最適化されたコミッターの使用」を参照してください。

--enable-spark-ui

`true` に設定すると、Spark UI を使用して AWS Glue ETL ジョブのモニタリングおよびデバッグを行う機能が有効になります。

--executor-cores

並列に実行できる Spark タスクの数。このオプションは、AWS Glue 3.0 以降でサポートされています。この値は、ワーカータイプの vCPU の数の 2 倍を超えないようにします。`G.1X` では 8、`G.2X` では 16、`G.4X` では 32、`G.8X` では 64 です。タスクの並列処理が増えるとメモリやディスクに負荷がかかるだけでなく、ソースシステムとターゲットシステムがスロットリングされる (例えば、Amazon RDS での同時接続数が増える) ため、この構成を更新するときは注意が必要です。

--extra-files

スクリプトを実行する前に、AWS Glue がドライバーノードのスクリプトの作業ディレクトリにコピーする構成ファイルなど、追加ファイルへの Amazon S3 パス。複数の値はコンマ (`,`) で区切られた完全なパスでなければなりません。ディレクトリパスではなく、個別のファイルのみがサポートされています。このオプションは Python Shell ジョブタイプではサポートされていません。

--extra-jars

AWS Glue がドライバーおよびエグゼキューターにコピーする追加ファイルへの Amazon S3 パス。AWSGlue は、スクリプトを実行する前にこれらのファイルを Java クラスパスにも追加します。複数の値はコンマ (`,`) で区切られた完全なパスでなければなりません。拡張機能は `.jar` である必要はありません

--extra-py-files

スクリプトを実行する前に、AWS Glue がドライバーノードの Python パスに追加する追加の Python モジュールへの Amazon S3 パス。複数の値はコンマ (`,`) で区切られた完全なパスでなければなりません。ディレクトリパスではなく、個別のファイルのみがサポートされています。

--job-bookmark-option

ジョブブックマークの動作を制御します。次のオプション値を設定できます。

--job-bookmark-option 値	説明
job-bookmark-enable	以前に処理されたデータを追跡します。ジョブが実行されると、最後のチェックポイントから新しいデータを処理します。
job-bookmark-disable	常にデータセット全体を処理します。以前のジョブからの出力の管理は、ユーザーが行います。
job-bookmark-pause	最後のブックマークの状態は更新せずに、最後に正常に実行された後の増分データ、または次のサブオプションで識別される範囲内のデータを処理します。以前のジョブからの出力の管理は、ユーザーが行います。2 つのサブオプションは以下のとおりです。 • **job-bookmark-from** <from-value> は、指定された実行 ID を含む最後に成功した実行までに処理されたすべての入力を表す実行 ID です。対応する入力は無視されます。 • **job-bookmark-to** <to-value> は、指定された実行 ID を含む最後に成功した実行までに処理されたすべての入力を表す実行 ID です。<from-value> によって識別される入力を除く対応する入力は、ジョブによって処理されます。この入力より後の入力も処理対象から除外されます。 このオプションが設定されている場合、ジョブのブックマークの状態は更新されません。 サブオプションはオプションです。ただし、使用する場合は、両方のサブオプションを指定する必要があります。

例えば、ジョブブックマークを有効にするには、以下の引数を渡します。

```
'--job-bookmark-option': 'job-bookmark-enable'
```

--job-language

スクリプトプログラミング言語。この値は scala または python のいずれかである必要があります。このパラメータが存在しない場合、デフォルトで python が使用されます。

--python-modules-installer-option

--additional-python-modules でモジュールをインストールする場合に、pip3 に渡されるオプションを定義するプレーンテキストの文字列。コマンドラインの場合と同様に、スペースで区切り、先頭にダッシュを付けてオプションを指定します。使用方法の詳細については、「the section called “pip を使用して追加の Python モジュールを AWS Glue 2.0 以降にインストールする”」(Python モジュールのサポートの追加) を参照してください。

> **Note**
>
> Python 3.9 を使用する場合、このオプションは AWS Glue ジョブではサポートされません。

--scriptLocation

ETL スクリプトが (s3://path/to/my/script.py 形式で) 置かれている Amazon Simple Storage Service (Amazon S3) の場所。このパラメータは、JobCommand オブジェクトで設定されているスクリプトの場所を上書きします。

--spark-event-logs-path

Amazon S3 パスを指定します。Spark UI モニタリング機能を使用する場合、AWS Glue は Spark イベントログを、30 秒ごとに、この Amazon S3 パスの (Spark UI イベントを保存するための一時ディレクトリとして使用可能な) バケットにフラッシュします。

--TempDir

ジョブの一時ディレクトリとして使用できるバケットへの Amazon S3 のパスを指定します。

例えば、一時ディレクトリを設定するには、以下の引数を渡します。

```
'--TempDir': 's3-path-to-directory'
```

> **Note**
>
> バケットがリージョンに既に存在しない場合、AWS Glue はジョブに一時的なバケットを作成します。このバケットは、パブリックアクセスを許可する場合があります。Amazon S3 のバケットは、パブリックアクセスブロックを設定するように変更するか、そのリージョンのすべてのジョブが完了した後に削除するかのいずれかが可能です。

`--use-postgres-driver`

この値を `true` に設定した場合、Amazon Redshift の JDBC ドライバーとの競合を避けるために、クラスパスにある Postgres JDBC ドライバーが優先されます。このオプションは、AWS Glue バージョン 2.0 以降でのみ使用可能です。

`--user-jars-first`

この値を `true` に設定した場合、クラスパス内にあるお客様の追加 JAR ファイルが優先されます。このオプションは、AWS Glue バージョン 2.0 以降でのみ利用できます。

`--conf`

Spark の設定パラメータを制御します。高度なユースケース向けです。

`--encryption-type`

レガシーパラメータ。対応する動作は、セキュリティ設定を使用して設定する必要があります。セキュリティ設定の詳細については、「the section called “AWS Glue によって書き込まれたデータの暗号化”」を参照してください。

AWS Glue は次の引数を内部的に使用するため、ユーザーは使用しません。

- `--debug` — AWS Glue 内部用。設定する必要はありません。
- `--mode` — AWS Glue 内部用。設定する必要はありません。
- `--JOB_NAME` — AWS Glue 内部用。設定する必要はありません。
- `--endpoint` — AWS Glue 内部用。設定する必要はありません。

AWS Glue は、サイト固有のカスタマイズを実行するための `sitecustomize` を使用した Python の `site` モジュールによる環境のブートストラップをサポートします。独自の初期化関数のブートスト

ラップは高度なユースケースにのみ推奨され、AWS Glue 4.0 ではベストエフォートベースでサポートされています。

環境変数のプレフィックス GLUE_CUSTOMER は、お客様専用です。

AWS GlueSpark と PySpark のジョブ

AWS Glue は Spark および PySpark のジョブをサポートします。Spark ジョブは、AWS Glue によって管理される Apache Spark 環境で実行されます。データはバッチで処理されます。ストリーミング ETL ジョブは Spark ジョブに似ていますが、データストリームで ETL を実行する点が異なります。このタイプのジョブでは Apache Spark 構造化ストリーミングフレームワークが使用されます。一部の Spark ジョブ機能は、ストリーミング ETL ジョブでは使用できません。

以下のセクションでは、AWS GlueSpark と PySpark のジョブについて説明します。

トピック

- AWS Glue で Spark ジョブに関するジョブプロパティの構成
- AWS Glueコンソールで Spark スクリプトの編集
- 仕事 (レガシー)
- ジョブのブックマークを使用した処理済みデータの追跡
- Spark のシャッフルデータの保存
- モニタリングAWS GlueSpark ジョブ
- AWS Glue での Apache Spark の生成 AI トラブルシューティング

AWS Glue で Spark ジョブに関するジョブプロパティの構成

AWS Glue コンソールでジョブを定義するときに、AWS Glue ランタイム環境をコントロールするためのプロパティの値を指定します。

Spark ジョブのジョブプロパティの定義

次のリストは、Spark ジョブのプロパティについて説明しています。Python シェルジョブのプロパティについては、「Python シェルジョブのジョブプロパティの定義」を参照してください。ストリーミング ETL ジョブのプロパティについては、「the section called “ストリーミング ETL ジョブのジョブプロパティの定義”」を参照してください。

プロパティは、AWS Glue コンソールの [Add job] (ジョブの追加) ウィザードに表示された順に一覧表示されます。

名前

UTF-8 文字を 255 文字以内で入力します。

説明

オプションとして最大 2,048 文字の説明を提示します。

IAM ロール

ジョブ実行とデータストアへのアクセスに使用されるリソースの認可に使用する IAM ロールを指定します。AWS Glue でジョブを実行するためのアクセス権限の詳細については、AWS Glue のアイデンティティとアクセスの管理 を参照してください。

タイプ

ETL ジョブの種類。これは、選択するデータソースの種類に基づいて自動的に設定されます。

- [Spark] は、`glueetl` のジョブコマンドで Apache Spark ETL スクリプトを実行します。
- [Spark ストリーミング] は、`gluestreaming` のジョブコマンドで Apache Spark ストリーミング ETL スクリプトを実行します。詳細については、「the section called “ストリーミング ETLジョブ”」を参照してください。
- [Python シェル] は、`pythonshell` のジョブコマンドで Python スクリプトを実行します。詳細については、「AWS Glue での Python シェルジョブに関するジョブプロパティの設定」を参照してください。

AWS Glue バージョン

AWS Glue のバージョンによって、ジョブで使用できる Apache Spark および Python のバージョンが次の表に指定されているように決まります。

AWS Glue バージョン	サポートされている Spark および Python のバージョン
5.0	• Spark 3.5.4 • Python 3.11
4.0	• Spark 3.3.0 • Python 3.10
3.0	• Spark 3.1.1 • Python 3.7

言語

ETL スクリプト内のコードでジョブのロジックを定義します。Python または Scala でスクリプトを記述できます。ジョブが実行するスクリプトを AWS Glue によって生成するのか、それとも自分で提供するのかを選択できます。Amazon Simple Storage Service (Amazon S3) でスクリプト名と場所を指定します。パスのスクリプトディレクトリと同じ名前のファイルが存在していないことを確認します。スクリプトの記述の詳細については、「AWS Glue プログラミングガイド」を参照してください。

ワーカータイプ

以下のワーカータイプを使用できます。

AWS Glue ワーカーで利用可能なリソースは DPU 単位で測定されます。DPU は処理能力を相対的に測定するもので、4 個の vCPU のコンピューティング性能と 16 GB のメモリで構成されています。

- **G.1X** – このタイプを選択する場合は、[Number of workers] (ワーカー数) の値も指定します。各ワーカーは、94 GB のディスクを備えた 1 DPU (4 vCPU、16 GB のメモリ) にマッピングされます。データ変換、結合、クエリなどのワークロードには、ほとんどのジョブを実行するためのスケーラブルで費用対効果の高い方法として、このワーカータイプをお勧めします。
- **G.2X** – このタイプを選択する場合は、[Number of workers] (ワーカー数) の値も指定します。各ワーカーは、138 GB のディスクを備えた 2 DPU (8 vCPU、32 GB のメモリ) にマッピングされます。データ変換、結合、クエリなどのワークロードには、ほとんどのジョブを実行するためのスケーラブルで費用対効果の高い方法として、このワーカータイプをお勧めします。
- **G.4X** – このタイプを選択する場合は、[Number of workers] (ワーカー数) の値も指定します。各ワーカーは、256 GB のディスク (約 235 GB の空き) を備えた 4 DPU (16 vCPU、64 GB のメモリ) にマッピングされます。ワークロードに含まれる変換、集約、結合、クエリへの要求が非常に厳しいジョブには、このワーカータイプをお勧めします。
- **G.8X** – このタイプを選択する場合は、[Number of workers] (ワーカー数) の値も指定します。各ワーカーは、512 GB のディスク (約 487 GB の空き) を備えた 8 DPU (32 vCPU、128 GB のメモリ) にマッピングされます。ワークロードに含まれる変換、集約、結合、クエリへの要求が非常に厳しいジョブには、このワーカータイプをお勧めします。
- **G.025X** – このタイプを選択する場合は、[Number of workers] (ワーカー数) の値も指定します。各ワーカーは、84 GB のディスク (約 34 GB の空き) を備えた 0.25 DPU (2 vCPU、4 GB のメモリ) にマッピングされます。少量のストリーミングジョブには、このワーカータイプをお勧めします。このワーカータイプは、AWS Glue バージョン 3.0 以降のストリーミングジョブでのみ使用できます。

ETL ジョブの実行に使用された DPU の数に基づいて時間あたりの料金が請求されます。詳細については、「AWS Glue 料金表ページ」を参照してください。

AWS Glue バージョン 1.0 以前のジョブでは、コンソールを使用してジョブを設定し、[Worker type] (ワーカータイプ) に [Standard] (標準) を指定する場合は、[Maximum capacity] (最大キャパシティー) が設定され、[Number of workers] (ワーカー数) は [Maximum capacity] (最大キャパシティー) から 1 を引いた値になります。AWS Command Line Interface (AWS CLI) または AWS SDK を使用する場合は、[Max capacity] (最大キャパシティー) パラメータを指定することも、[Worker type] (ワーカータイプ) と [Number of workers] (ワーカー数) の両方を指定することもできます。

AWS Glue バージョン 2.0 以降のジョブでは、[最大キャパシティ] の指定はできません。代わりに、[Worker type] (ワーカータイプ) と [Number of workers] (ワーカー数) を指定します。

G.4X と **G.8X** のワーカータイプは、米国東部 (オハイオ)、米国東部 (バージニア北部)、米国西部 (北カリフォルニア)、米国西部 (オレゴン)、アジアパシフィック (ムンバイ)、アジアパシフィック (ソウル)、アジアパシフィック (シンガポール)、アジアパシフィック (シドニー)、アジアパシフィック (東京)、カナダ (中部)、欧州 (フランクフルト)、欧州 (アイルランド)、欧州 (ロンドン)、欧州 (スペイン)、欧州 (ストックホルム)、南米 (サンパウロ) の AWS リージョンで、AWS Glue バージョン 3.0 以降の Spark ETL に対してのみ使用できます。

リクエストしたワーカーの人数

ほとんどのワーカータイプで、ジョブの実行時に割り当てられるワーカーの数を指定する必要があります。

ジョブのブックマーク

ジョブ実行時に AWS Glue が状態情報を処理する方法を指定します。以前に処理されたデータの記憶、状態情報の更新、または状態情報の無視を指定できます。詳細については、「the section called “ジョブのブックマークを使用した処理済みデータの追跡”」を参照してください。

ジョブ実行キューイング

サービスクォータが原因でジョブをすぐに実行できないとき、ジョブを後で実行するようにキューに入れるかどうかを指定します。

チェックすると、ジョブ実行キューイングがジョブ実行に対して有効になります。入力されていない場合、ジョブ実行はキューイングの対象になりません。

この設定がジョブ実行で設定された値と一致しない場合、ジョブ実行フィールドの値が使用されます。

Flex 実行

AWS Studio や API を使用してジョブを設定する際、標準のジョブ実行クラスあるいは柔軟なジョブ実行クラスを指定することができます。ジョブには、さまざまな度合いの優先順位や時間的な制約を設定することができます。標準の実行クラスは、素早くジョブを起動する必要があり、専用のリソースが必要な時間的な制約のあるワークロードに最適です。

柔軟な実行クラスは、実稼働前のジョブ、テスト、1 回限りのデータの読み込みなど、緊急性のないジョブに適しています。柔軟なジョブの実行は、AWS Glue バージョン 3.0 以降と `G.1X` または `G.2X` ワーカータイプを使用するジョブでサポートされています。

Flex ジョブの実行は、任意の時点で実行されているワーカーの数に基づいて課金されます。柔軟なジョブの実行では、ジョブ数を追加または削除することができます。`Max Capacity * Execution Time` という単純な計算で課金されるのではなく、各ワーカーがジョブ実行中に実行された時間が含まれます。請求は (`Number of DPUs per worker * time each worker ran`) の合計です。

詳細は、「AWS Studio のヘルプパネル」、または「ジョブ」や「ジョブ実行」を参照してください。

再試行回数

失敗した場合に AWS Glue がジョブを自動的に再起動する回数を 0〜10 の間で指定します。タイムアウト制限に達したジョブは再起動されません。

ジョブのタイムアウト

最大の実行時間 (分) を設定します。最大値は 7 日 (10,080 分) です。それ以外の場合は、ジョブは例外をスローします。

値を空白のままにすると、タイムアウトはデフォルトで 2,880 分に設定されます。

タイムアウト値が 7 日を超える既存の AWS Glue ジョブは、デフォルトで 7 日に設定されます。たとえば、バッチジョブに 20 日間のタイムアウトを指定した場合、7 日目に停止します。

> **ジョブタイムアウトのベストプラクティス**
>
> ジョブは実行時間に基づいて課金されます。予期しない課金を避けるには、ジョブの予想実行時間に適切なタイムアウト値を設定してください。

詳細プロパティ

スクリプトのファイル名

ジョブの一意のスクリプト名。[タイトルがないジョブ] という名前を付けられません。

スクリプトパス

スクリプトの Amazon S3 ロケーション。パスは `s3://bucket/prefix/path/` の形式で指定する必要があります。末尾にスラッシュ (/) を付けてファイルは含めないでください。

ジョブのメトリクス

このジョブの実行時に Amazon CloudWatch メトリクスの作成を有効または無効にします。プロファイリングデータを表示するには、このオプションを有効にする必要があります。メトリクスを有効にして表示する方法の詳細については、「ジョブのモニタリングとデバッグ」を参照してください。

ジョブのオブザーバビリティメトリクス

このジョブの実行時に追加のオブザーバビリティ CloudWatch メトリクスの作成を有効にします。詳細については、「the section called “AWS Glue オブザーバビリティメトリクスを使用したモニタリング”」を参照してください。

連続ログ記録

Amazon CloudWatch への連続ログ記録を有効にします。このオプションが有効になっていない場合、ログはジョブの完了後にのみ使用できます。詳細については、the section called “AWS Glue ジョブのログ記録” を参照してください。

Spark UI

このジョブをモニタリングするために Spark UI の使用を有効にします。詳細については、「AWS Glue ジョブ用の Apache Spark ウェブ UI の有効化」を参照してください。

Spark UI ログパス

Spark UI が有効になっているときにログを書き込むパス。

Spark UI のログ記録とモニタリングの設定

以下のオプションのいずれかを選択してください：

- スタンダード: AWS Glue ジョブの実行 ID をファイル名として使用してログを書き込みます。AWS Glue コンソールで Spark UI モニタリングを有効にします。

- レガシー: 「spark-application-{timestamp}」をファイル名として使用してログを書き込みます。Spark UI モニタリングをオンにしないでください。
- スタンダードとレガシー: スタンダードロケーションとレガシーロケーションの両方にログを書き込みます。AWS Glue コンソールで Spark UI モニタリングを有効にします。

最大同時実行数

このジョブで許可される同時実行の最大数を設定します。デフォルトは 1 です。このしきい値に達すると、エラーが返されます。指定できる最大値は、サービスの制限によってコントロールされます。たとえば、新しいインスタンスの開始時に前回のジョブがまだ実行されている場合、同じジョブの 2 つのインスタンスが同時に実行されないようにエラーを戻すことができます。

一時的なパス

AWS Glue がスクリプトを実行するときに一時的な中間結果が書き込まれる Amazon S3 の作業ディレクトリの場所を指定します。パスの一時ディレクトリと同じ名前のファイルが存在していないことを確認します。このディレクトリは、Amazon Redshift に対して AWS Glue が読み取りと書き込みをするときに使用します。また、特定の AWS Glue 変換で使用します。

> **Note**
>
> AWS Glue では、リージョンにバケットが存在しない場合、ジョブの一時バケットが作成されます。このバケットは、パブリックアクセスを許可する場合があります。この Amazon S3 に置かれたバケットは、パブリックアクセスブロックを構成するために設定することができます。また、そのリージョンのすべてのジョブが完了した後に削除することが可能です。

遅延通知のしきい値 (分)

遅延通知を送信するまでのしきい値 (分単位) を設定します。RUNNING、STARTING、または STOPPING ジョブの実行が想定時間 (分単位) を超えると通知を送信するように、このしきい値を設定できます。

セキュリティ設定

リストからセキュリティ設定を選択します。セキュリティ設定では、Amazon S3 ターゲットのデータの暗号化方法として、暗号化なし、AWS KMS で管理されたキー (SSE-KMS) を使用したサーバー側の暗号化、または Amazon S3 で管理された暗号化キー (SSE-S3) を使用したサーバー側の暗号化を指定します。

サーバー側の暗号化

このオプションを選択すると、ETL ジョブが Amazon S3 に書き込むときに、データは SSE-S3 暗号化を使用して保管時に暗号化されます。Amazon S3 のデータターゲットと、Amazon S3 の一時ディレクトリに書き込まれるデータは、両方とも暗号化されています。このオプションはジョブパラメータとして渡されます。詳細については、Amazon Simple Storage Service ユーザーガイドの「Amazon S3 が管理する暗号化キーによるサーバー側の暗号化 (SSE-S3) を使用したデータの保護」を参照してください。

> **Important**
> セキュリティ設定を指定している場合、このオプションは無視されます。

Glue データカタログを Hive メタストアとして使用する

AWS Glue データカタログを Hive メタストアとして使用することを選択します。ジョブに使用される IAM ロールには、`glue:CreateDatabase` アクセス許可が必要です。「default」という名前のデータベースが存在しない場合はデータカタログに作成されます。

Connections

VPC 設定を選択し、仮想プライベートクラウド (VPC) にある Amazon S3 データソースにアクセスします。AWS Glue でネットワーク接続を作成および管理できます。詳細については、「データへの接続」を参照してください。

[Libraries] (ライブラリ)

Python ライブラリパス、依存 JAR パス、参照されるファイルパス

これらのオプションはスクリプトで必要に応じて指定します。ジョブを定義するときに、これらのオプションのカンマ区切りの Amazon S3 パスを定義できます。ジョブ実行時にこれらのパスを上書きできます。詳細については、「独自のカスタムスクリプトの提供」を参照してください。

ジョブのパラメータ

スクリプトに名前付きパラメータとして渡される一連のキーと値のペア。これらは、スクリプトの実行時に使用されるデフォルト値ですが、トリガーまたはジョブの実行時に上書きできます。キー名の前に `--` を付ける必要があります (`--myKey` など)。AWS Command Line Interface を使用するときに、ジョブパラメータをマッピングとして渡します。

例については、「AWS Glue の Python パラメータの受け渡しとアクセス」で Python パラメータを参照してください。

[タグ]

[タグキー] とオプションの [タグ値] を使用してジョブにタグを付けます。作成されたタグキーは読み取り専用になります。リソースを整理、識別しやすいように、いくつかのリソースでタグを使用します。詳細については、「AWSAWS Glue のタグ」を参照してください。

Lake Formation 管理対象テーブルにアクセスするジョブの制約事項

AWS Lake Formation で管理されるテーブルに対して読み書きするジョブを作成する場合は、次の注意事項と制約事項に注意してください。

- 次の機能は、セルレベルフィルタを使用してテーブルにアクセスするジョブではサポートされません。
  - ジョブのブックマークと境界実行の実行
  - プッシュダウン述語
  - サーバーサイドのカタログパーティション述語
  - UpdateCatalog を有効にする

AWS Glueコンソールで Spark スクリプトの編集

スクリプトには、ソースからデータを抽出して、変換し、ターゲットにロードするコードが含まれています。AWS Glue はジョブを開始するときにスクリプトを実行します。

Python または Scala で AWS Glue ETL スクリプトを記述できます。Python スクリプトは、抽出、変換、ロード (ETL) ジョブのための PySpark Python ダイアレクトの拡張機能である言語を使用します。スクリプトには ETL 変換を処理する拡張構造が含まれます。自動でジョブのソースコードロジックを生成するときに、スクリプトが作成されます。このスクリプトを編集するか、または、独自のスクリプトを指定して ETL 作業を処理することができます。

AWS Glue のスクリプトの定義と編集の詳細については、「AWS Glue プログラミングガイド」を参照してください。

その他のライブラリまたはファイル

スクリプトに追加のライブラリやファイルが必要な場合は、次のように指定できます。

Python ライブラリパス

スクリプトで必要とされる Python ライブラリへのカンマ区切りの Amazon Simple Storage Service (Amazon S3) パス。

> **Note**
>
> 純粋な Python ライブラリのみを使用できます。pandas Python データ解析ライブラリなど、C 拡張機能に依存するライブラリはまだサポートされていません。

依存 JARS パス

スクリプトで必要とされる JAR ファイルへのカンマ区切りの Amazon S3 パスです。

> **Note**
>
> 現在、純粋な Java または Scala (2.11) ライブラリのみを使用できます。

参照されるファイルパス

スクリプトに必要な追加のファイル (例えば、設定ファイル) への、カンマ区切りの Amazon S3 パス。

仕事 (レガシー)

スクリプトには、抽出、変換、ロード (ETL) ワークを実行するコードが含まれます。独自のスクリプトを提供することもできますし、お客様のガイダンスで AWS Glue がスクリプトを生成することもできます。独自のスクリプトの作成については、「独自のカスタムスクリプトの提供」を参照してください。

スクリプトは、AWS Glue コンソールで編集できます。スクリプトを編集する場合、ソース、ターゲット、および変換を追加することができます。

スクリプトを編集するには

1. AWS Management Consoleにサインインし、AWS Glue コンソール (https://console.aws.amazon.com/glue/) を開きます。その後 [Jobs] (ジョブ) タブを選択します。

2. リストでジョブを選択し、次に [Action] (アクション)、[Edit script] (スクリプトの編集) を選択してスクリプトエディタを開きます。

   ジョブ詳細ページからスクリプトエディタにアクセスすることもできます。[Script] (スクリプト) タブを選択し、次に [Edit script] (スクリプトの編集) を選択します。

スクリプトエディタ

AWS Glue スクリプトエディタを使用して、スクリプトのソース、ターゲット、変換を挿入、変更、および削除できます。スクリプトエディタにはスクリプトとダイアグラムの両方が表示され、データの流れを可視化しやすくなります。

スクリプトのダイアグラムを作成するには、[ダイアグラムの生成] を選択します。AWS Glue は ## で始まるスクリプト内の注釈行を使用してダイアグラムをレンダリングします。ダイアグラムでスクリプトを正しく表すために、注釈のパラメータと Apache Spark コードのパラメータの同期を保つ必要があります。

スクリプトエディタを使用して、スクリプトのカーソルが置かれている任意の場所にコードテンプレートを追加することができます。エディタの上部で、次のオプションから選択します。

- ソーステーブルをスクリプトに追加するには、[Source] (ソース) を選択します。
- ターゲットテーブルをスクリプトに追加するには、[Target] (ターゲット) を選択します。
- ターゲット位置をスクリプトに追加するには、[Target location] (ターゲット位置) を選択します。
- 変換をスクリプトに追加するには、[Transform] (変換) を選択します。スクリプトで呼び出される関数については、「PySpark で AWS Glue ETL スクリプトをプログラムする」を参照してください。
- スピゴット変換をスクリプトに追加するには、[Spigot] (スピゴット) を選択します。

挿入されたコードで、注釈および Apache Spark コード両方の `parameters` を変更します。たとえば、スピゴット変換を追加したら、`path` が `@args` 注釈行および `output` コード行の両方で置き換えられていることを検証します。

[Logs] (ログ) タブでは、実行されるジョブに関連するログが表示されます。最新の 1,000 行が表示されます。

[Schema] (スキーマ) タブでは、Data Catalog で使用可能な場合、選択されたソースとターゲットのスキーマが表示されます。

ジョブのブックマークを使用した処理済みデータの追跡

AWS Glue ではジョブの実行による状態情報を保持することで、ETL ジョブの以前の実行中にすでに処理されたデータを追跡します。この継続状態の情報はジョブのブックマークと呼ばれています。ジョブのブックマークは、AWS Glue で状態情報を保持して、古いデータを再処理しないために役立ちます。ジョブのブックマークを使用すると、スケジュールされた間隔で再実行する際に新しいデータを処理できます。ジョブのブックマークは、ソース、変換、ターゲットなど、さまざまなジョブの要素で構成されています。例えば、ETL ジョブが Amazon S3 ファイルで新しいパーティションを読み込むとします。AWS Glue は、そのジョブにより正常に処理されたのはどのパーティションなのかを追跡し、処理の重複およびジョブのターゲットデータストアにデータが重複するのを防ぎます。

ジョブのブックマークは、JDBC データソース、Relationalize 変換、および一部の Amazon Simple Storage Service (Amazon S3) ソースに実装されています。次の表に、AWS Glue がジョブのブックマークに対してサポートする Amazon S3 ソース形式を示します。

AWS Glue バージョン	Amazon S3 ソース形式
バージョン 0.9	JSON、CSV、Apache Avro、XML
バージョン 1.0 以降	JSON、CSV、Apache Avro、XML、Parquet、ORC

AWS Glue バージョンの詳細に関しては、「Spark ジョブのジョブプロパティの定義」を参照してください。

ジョブのブックマーク機能には、AWS Glue スクリプトからアクセスする際の追加機能があります。生成されたスクリプトを参照すると、この機能に関連する変換コンテキストが表示される場合があります。詳細については、「the section called “ジョブのブックマークを使用する”」を参照してください。

トピック

- AWS Glue でジョブのブックマークを使用する
- ジョブブックマーク機能の運用に関する詳細

AWS Glue でジョブのブックマークを使用する

ジョブのブックマークオプションは、ジョブが開始したときにパラメータとして渡されます。AWS Glue コンソールでジョブのブックマークを設定するためのオプションを次の表に示します。

ジョブのブックマーク	説明
有効	ジョブの実行後に状態を更新させて以前に処理されたデータを追跡します。ジョブのブックマークをサポートしているソースのあるジョブの場合、ジョブは処理されたデータを追跡し、ジョブが実行されると、最後のチェックポイント以降の新しいデータを処理します。
[無効]	ジョブのブックマークは使用されず、ジョブは常にデータセット全体を処理します。以前のジョブからの出力の管理は、ユーザーが行います。これがデフォルトです。
[Pause] (一時停止)	最後のブックマークの状態は更新せずに、最後に正常に実行された後の増分データ、または次のサブオプションで識別される範囲内のデータを処理します。以前のジョブからの出力の管理は、ユーザーが行います。次の 2 つのサブオプションがあります。 • job-bookmark-from <from-value> は、指定された実行 ID を含む最後に成功した実行までに処理されたすべての入力を表す実行 ID です。対応する入力は無視されます。 • job-bookmark-to <to-value> は、指定された実行 ID を含む最後に成功した実行までに処理されたすべての入力を表す実行 ID です。<from-value> によって識別される入力を除く対応する入力は、ジョブによって処理されます。この入力より後の入力も処理対象から除外されます。 このオプションが設定されている場合、ジョブのブックマークの状態は更新されません。 サブオプションはオプションですが、使用する場合、両方のサブオプションを提供する必要があります。

コマンドラインでジョブに渡されるパラメータ、および特にジョブブックマークの詳細については、「AWS Glue ジョブでジョブパラメータを使用する」を参照してください。

Amazon S3 入力ソースの場合、AWS Glue ジョブのブックマークではオブジェクトの最終更新日時を確認して、どのオブジェクトを再処理する必要があるのかを確認します。入力ソースデータが最後のジョブ実行以降に変更されている場合、ジョブを再度実行すると、ファイルが再度処理されます。

JDBC ソースでは、次のルールが適用されます。

- AWS Glue では、テーブルごとに 1 つ以上の列をブックマークキーとして使用して、新しいデータおよび処理済みのデータを決定します。ブックマークキーは、結合して単一の複合キーを形成します。
- AWS Glue はデフォルトでプライマリキーをブックマークキーとして使用します。ただし、プライマリキーが連続して (ギャップなく) 増減することが条件です。
- AWS Glue スクリプトでブックマークキーとして使用する列を指定できます。AWS Glue スクリプトでのジョブブックマーク使用の詳細については、「the section called “ジョブのブックマークを使用する”」を参照してください。
- AWS Glue は、ジョブブックマークキーとして名前の大文字と小文字を区別する列の使用をサポートしていません。

AWS Glue Spark ETL ジョブのジョブブックマークを以前のジョブ実行に巻き戻すことができます。ジョブのブックマークを以前のジョブ実行に巻き戻すことで、データのバックフィルシナリオをより適切にサポートできます。その結果、後続のジョブ実行ではブックマークされたジョブ実行からのデータだけが再処理されます。

同じジョブを使用してすべてのデータを再処理する場合は、ジョブのブックマークをリセットします。ジョブのブックマークの状態をリセットするには、AWS Glue コンソール、ResetJobBookmark アクション (Python: reset_job_bookmark) API オペレーション、または AWS CLI を使用します。例えば、AWS CLI を使用して以下のコマンドを入力します。

```
aws glue reset-job-bookmark --job-name my-job-name
```

ブックマークを巻き戻したりリセットしたりしたとき、複数のターゲットが存在する可能性があり、ターゲットがジョブブックマークで追跡されないため、AWS Glue はターゲットファイルをクリーンアップしません。ジョブブックマークで追跡されるのは、ソースファイルだけです。ソースファイルの巻き戻しと再処理時に異なる出力ターゲットを作成して、出力内のデータが重複しないようにすることができます。

AWS Glue では、ジョブでジョブのブックマークを管理します。ジョブを削除すると、ジョブのブックマークは削除されます。

場合によっては、AWS Glue ジョブのブックマークを有効にしてあっても、以前の実行で処理したデータを ETL ジョブで再処理することがあります。このエラーの一般的な原因を解決する方法の詳細については、「エラーのトラブルシューティング Spark エラー」を参照してください。

ジョブブックマーク機能の運用に関する詳細

このセクションでは、ジョブのブックマークを使用する運用の詳細を説明します。

ジョブのブックマークはジョブの状態を保存します。状態の各インスタンスは、ジョブ名とバージョン番号がキーになっています。スクリプトで呼び出された `job.init` では、その状態を取得し、常に最新バージョンを取得します。1 つの状態の中に複数の状態要素が存在します。要素は、スクリプト内のソース、変換、シンクインスタンスごとに固有です。これらの状態要素は、スクリプト内の対応する要素 (ソース、変換、またはシンク) にアタッチされている変換コンテキストによって識別されます。状態要素はユーザースクリプトから `job.commit` が呼び出されたときにアトミックに保存されます。スクリプトは引数からジョブのブックマークのジョブ名およびコントロールオプションを取得します。

ジョブのブックマーク内の状態要素は、ソース、変換、またはシンク固有のデータです。例えば、アップストリームジョブまたはプロセスによって継続的に書き込まれている Amazon S3 の場所から増分データを読み取るとします。この場合、スクリプトではそれまでに処理されている部分を判別する必要があります。Amazon S3 ソースのジョブのブックマーク実装では情報を保存するため、再度ジョブを実行するときに、保存された情報を使用して新しいオブジェクトのみをフィルターでき、次にジョブを実行するための状態を再計算できます。新しいファイルをフィルタリングするためにタイムスタンプが使用されます。

ジョブのブックマークは、状態要素に加えて、実行番号、試行番号、バージョン番号 を持ちます。実行番号はジョブの実行を追跡し、試行番号はジョブ実行の試行回数を記録します。ジョブ実行番号は正常な実行ごとに増分される、一定間隔で増加する番号です。試行番号は各実行の試行回数を追跡し、失敗した試行後のに実行がある場合にのみ増分されます。バージョン番号は、一定間隔で増加し、ジョブのブックマークの更新を追跡します。

AWS Glue サービスデータベースでは、すべての変換のブックマーク状態がキーと値のペアとして一緒に格納されます。

```
{
  "job_name" : ...,
  "run_id": ...,
```

```
  "run_number": ..,
  "attempt_number": ...
  "states": {
    "transformation_ctx1" : {
      bookmark_state1
    },
    "transformation_ctx2" : {
      bookmark_state2
    }
  }
}
```

ベストプラクティス

以下は、ジョブブックマークを使用するためのベストプラクティスです。

- ブックマークを有効にした状態でデータソースプロパティを変更しないでください。例えば、Amazon S3 入力パス A を指している datasource0 があります。ジョブはブックマークを有効にして数ラウンド実行されているソースから読み出しています。`transformation_ctx` を変更せずに datasource0 の入力パスを Amazon S3 パス B に変更すると、AWS Glue ジョブは保存されている古いブックマークの状態を使用します。これによって、入力パス B のファイルは、見つからないか、スキップされます。AWS Glue は、これらのファイルが以前の実行で処理されたと仮定します。
- パーティション管理を改善するために、ブックマーク付きのカタログテーブルを使用してください。ブックマークは、データカタログのデータソースとオプションの両方に対して機能します。ただし、from オプションのアプローチでは、新しいパーティションを削除/追加するのは困難です。クローラーでカタログテーブルを使用すると、新しく追加された パーティション を追跡するための自動化が向上し、プッシュダウン述語を使用して特定のパーティションを柔軟に選択できるようになります。
- 大規模なデータセット向けのAWS GlueAmazon S3 ファイルリスターの使用 ブックマークは、各入力パーティションの下のすべてのファイルをリストし、ファイリングを実行します。したがって、単一のパーティションの下にファイルが多すぎると、ブックマークがドライバ OOM で実行される可能性があります。AWS Glue Amazon S3 ファイルリスターを使用して、メモリ内のすべてのファイルを一度にリストしないようにします。

Spark のシャッフルデータの保存

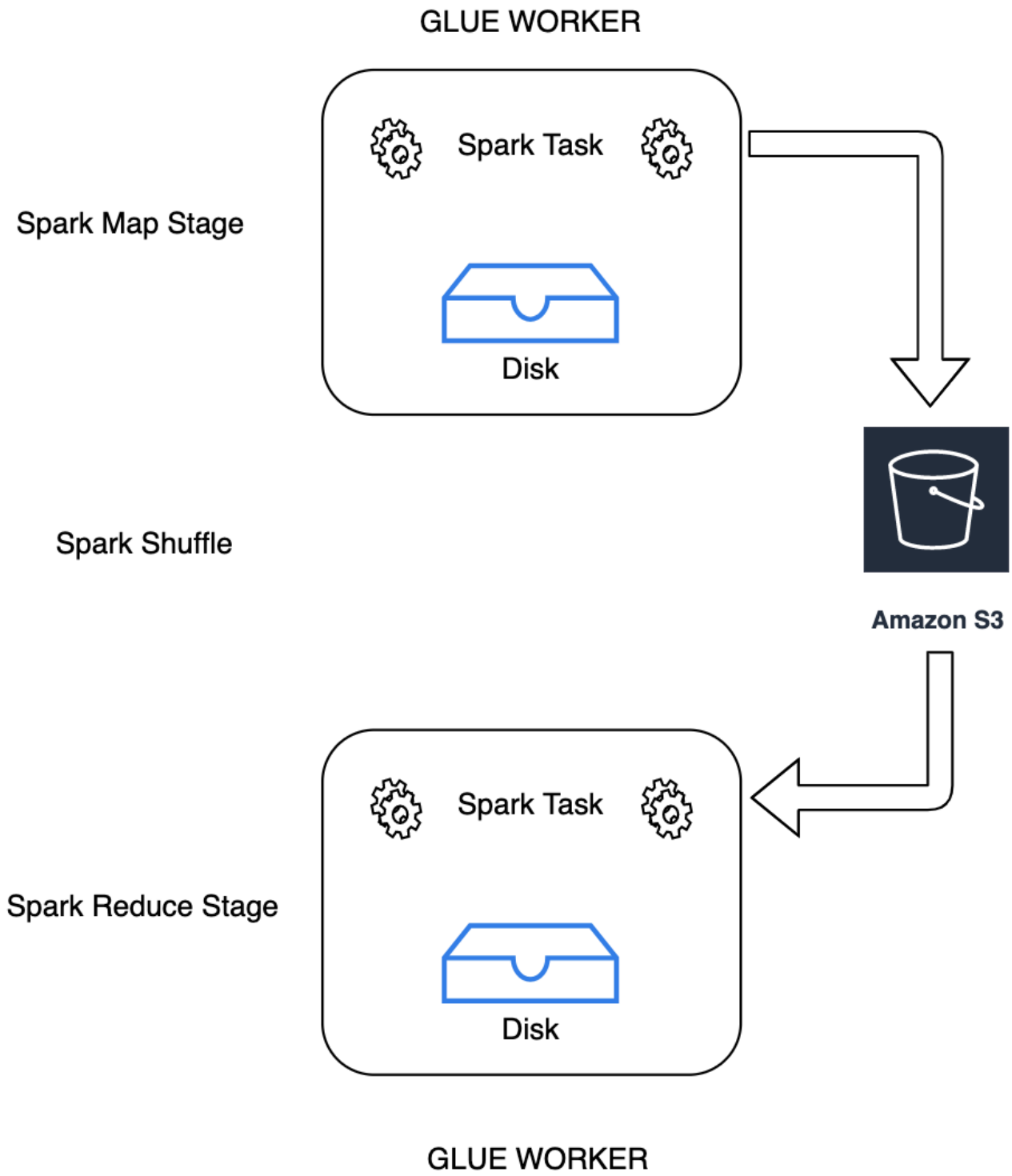
シャッフルは、データがパーティション間で再配置されるたびに行われる Spark ジョブの重要なステップです。これが必要なのは、`join`、 `groupByKey`、`reduceByKey`、`repartition` などのさ

まざまな変換に、処理を完了するために他のパーティションからの情報が必要なためです。Spark は各パーティションから必要なデータを収集し、新しいパーティションに結合します。シャッフル中、データはディスクに書き込まれ、ネットワーク経由で転送されます。その結果、シャッフルオペレーションはローカルディスク容量に制約されます。Spark の `No space left on device` または `MetadataFetchFailedException` エラーは、エグゼキューターに十分なディスク領域がなく、リカバリがない場合に発生します。

> **Note**
>
> Amazon S3 での AWS Glue Spark シャッフルプラグインは、AWS Glue ETL ジョブについてのみサポートされています。

ソリューション

AWS Glue では、Amazon S3 を使用して Spark シャッフルデータを保存できるようになりました。Amazon S3 は、業界をリードするスケーラビリティ、データ可用性、セキュリティ、およびパフォーマンスを提供するオブジェクトストレージサービスです。このソリューションで、Spark ジョブ用のコンピューティングとストレージが非集約化され、完全な伸縮自在性と低コストのシャッフルストレージが得られ、シャッフル負荷の高いワークロードが高い信頼性で実行できるようになります。



Amazon S3 を使用するための Cloud Shuffle Storage Plugin for Apache Spark を導入します。大きなシャッフルオペレーションのためにローカルディスク容量の制約を受けることがわかっている場合、Amazon S3 シャッフルを有効にすると、AWS Glue ジョブを失敗することなく高い信頼性で実行できます。小さなパーティションが多数ある場合や、Amazon S3 に書き込まれたファイルを

シャッフルする場合、Amazon S3 へのシャッフルはローカルディスク (または EBS) よりもわずかに遅くなる場合があります。

クラウドシャッフルストレージプラグインを使用するための前提条件

AWS Glue ETL ジョブにクラウドシャッフルストレージプラグインを使用するには、以下が必要です。

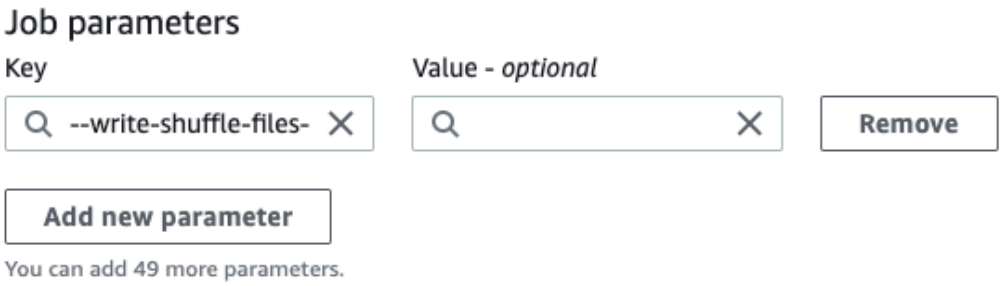
- 途中のシャッフルデータやスピルしたデータを保存するための、ジョブが実行されているのと同じリージョンにある Amazon S3 バケット。シャッフルストレージの Amazon S3 プレフィックスは、`--conf spark.shuffle.glue.s3ShuffleBucket=s3://shuffle-bucket/prefix/` で次の例のように指定できます。

```
--conf spark.shuffle.glue.s3ShuffleBucket=s3://glue-shuffle-123456789-us-east-1/glue-shuffle-data/
```

- シャッフルマネージャーはジョブの終了後にファイルをクリーンアップしないため、プレフィックス (`glue-shuffle-data` など) に Amazon S3 ストレージライフサイクルポリシーを設定します。途中のシャッフルデータやスピルしたデータは、ジョブの終了後に削除する必要があります。ユーザーはプレフィックスに短いライフサイクルポリシーを設定できます。Amazon S3 ライフサイクルポリシーのセットアップ手順については、「Amazon Simple Storage Service ユーザーガイド」の「バケットのライフサイクル設定の指定」を参照してください。

AWS コンソールからの AWS Glue Spark シャッフルマネージャーの使用

ジョブを設定するときに AWS Glue コンソールまたは AWS Glue Studio を使用して AWS Glue Spark シャッフルマネージャーをセットアップするには、--write-shuffle-files-to-s3 ジョブパラメータを選択して、ジョブの Amazon S3 シャッフルを有効にします。



AWS Glue Spark シャッフルプラグインの使用

次のジョブパラメータで、AWS Glue シャッフルマネージャーが有効になり、チューニングされます。これらのパラメータはフラグなので、指定された値は考慮されません。

- `--write-shuffle-files-to-s3` – メインフラグです。Amazon S3 バケットを使用してシャッフルデータの書き込みと読み込みを行う AWS Glue Spark シャッフルマネージャーが有効になります。フラグを指定しない場合、シャッフルマネージャーは使用されません。
- `--write-shuffle-spills-to-s3` – (AWS Glue バージョン 2.0 でのみサポートされています)。オプションのフラグです。スピルファイルを Amazon S3 バケットにオフロードできます。これにより、Spark ジョブの耐障害性が強化されます。これは、大量のデータをディスクに退避させる大規模なワークロードにのみ必要です。フラグが指定されていない場合、中間スピルファイルは書き込まれません。
- `--conf spark.shuffle.glue.s3ShuffleBucket=s3://<shuffle-bucket>` – 別のオプションのフラグです。シャッフルファイルを書き込む Amazon S3 バケットを指定します。デフォルトでは、`--TempDir/shuffle data` です。AWS Glue 3.0 以降では、`--conf spark.shuffle.glue.s3ShuffleBucket=s3://`*`shuffle-bucket-1`*`/`*`prefix`*`,s3://`*`shuffle-bucket-2`*`/`*`prefix`*`/` のようにバケットをカンマ区切りで指定することで、シャッフルファイルを複数のバケットに書き込むことができます。複数のバケットを使用するとパフォーマンスが向上します。

シャッフルデータの保存時の暗号化を有効にするには、セキュリティ構成設定を提供する必要があります。　セキュリティの構成の詳細については、「the section called “暗号化のセットアップ”」を参照してください。AWS Glue は、Spark が提供する他のすべてのシャッフル関連の構成をサポートします。

クラウドシャッフルストレージプラグインのソフトウェアバイナリ

また、Apache 2.0 ライセンスで Cloud Shuffle Storage Plugin for Apache Spark のソフトウェアバイナリをダウンロードして、任意の Spark 環境で実行することもできます。新しいプラグインは Amazon S3 をそのまま使えるようにサポートしており、Google クラウドストレージや Microsoft Azure Blob ストレージなどの他の形式のクラウドストレージを使用するように簡単に設定することもできます。詳細については、「Cloud Shuffle Storage Plugin for Apache Spark」を参照してください。

注意事項と制限事項

AWS Glue シャッフルマネージャーには次の注意事項や制限事項があります。

- AWS Glue シャッフルマネージャーは、ジョブの完了後に Amazon S3 バケットに保存されている (一時的な) シャッフルデータファイルを自動的に削除しません。データを確実に保護するには、Cloud Shuffle ストレージプラグインを有効にする前に、クラウドシャッフルストレージプラグインを使用するための前提条件 の手順に従ってください。
- データに偏りがある場合、この機能が使用できます。

Cloud Shuffle Storage Plugin for Apache Spark

Cloud Shuffle ストレージプラグインは `ShuffleDataIO API` と互換性のある Apache Spark プラグインで、シャッフルデータをクラウドストレージシステム (Amazon S3 など) に保存することを可能にします。この機能では、Spark アプリケーションにおいて `join`、`reduceByKey`、`groupByKey`、および `repartition` などの変換によってトリガーされることが多い、大規模なシャッフル操作用のローカルディスクストレージ容量を補充または置換できます。このため、サーバーレスデータ分析ジョブやパイプラインでの、一般的な障害やコスト対パフォーマンスの不整合を軽減するのに役立ちます。

AWS Glue

AWS Glue バージョン 3.0 と 4.0 には、このプラグインが事前インストールされており、特別な手順なしで Amazon S3 でシャッフルが利用可能になります。Spark アプリケーションでこの機能を有効にする方法については、「AWS Glue シャッフルマネージャーと Amazon S3 」を参照してください。

その他の Spark 環境

他の Spark 環境では、このプラグインに以下の Spark 構成を設定する必要があります。

- `--conf spark.shuffle.sort.io.plugin.class=com.amazonaws.spark.shuffle.io.cloud.Choppe`
  こSpark に対し、Shuffle IO 用としてこのプラグインを使用することを通知します。
- `--conf spark.shuffle.storage.path=s3://`*`bucket-name`*`/`*`shuffle-file-dir`*: シャッフルファイルが保存されるパスです。

> **Note**
>
> このプラグインは、Spark コアクラスの 1 つを上書きします。そのため、プラグインの jar ファイルは、Spark jar に先行してロードする必要があります。プラグインを AWS Glue の外

部で使用する場合は、オンプレミスの YARN 環境で `userClassPathFirst` を使用して、この処理を実行できます。

Spark アプリケーションへのプラグインのバンドル

Spark アプリケーションをローカルで開発する際、Maven `pom.xml` にプラグインの依存関係を追加することで、Spark アプリケーションと Spark ディストリビューション (バージョン 3.1 以降) にプラグインをバンドルできます。このプラグインと Spark バージョンの詳細については、「プラグインのバージョン」を参照してください。

```
<repositories>
   ...
    <repository>
        <id>aws-glue-etl-artifacts</id>
        <url>https://aws-glue-etl-artifacts.s3.amazonaws.com/release/ </url>
    </repository>
</repositories>
...
<dependency>
    <groupId>com.amazonaws</groupId>
    <artifactId>chopper-plugin</artifactId>
    <version>3.1-amzn-LATEST</version>
</dependency>
```

別の方法として、以下のように AWS Glue Maven アーティファクトからバイナリを直接ダウンロードして、Spark アプリケーションにインクルードすることも可能です。

```
#!/bin/bash
sudo wget -v https://aws-glue-etl-artifacts.s3.amazonaws.com/release/com/amazonaws/
chopper-plugin/3.1-amzn-LATEST/chopper-plugin-3.1-amzn-LATEST.jar -P /usr/lib/spark/
jars/
```

spark-submit の例

```
spark-submit --deploy-mode cluster \
--conf spark.shuffle.storage.s3.path=s3://<ShuffleBucket>/<shuffle-dir> \
--conf spark.driver.extraClassPath=<Path to plugin jar> \
--conf spark.executor.extraClassPath=<Path to plugin jar> \
--class <your test class name> s3://<ShuffleBucket>/<Your application jar> \
```

オプションの設定

これらは Amazon S3 シャッフルの動作を制御するための、オプションの設定値です。

- `spark.shuffle.storage.s3.enableServerSideEncryption`: ファイルのシャッフルおよびスピルのために、S3 SSE を有効化/無効化します。デフォルト値は `true` です。
- `spark.shuffle.storage.s3.serverSideEncryption.algorithm`: 使用される SSE アルゴリズム。デフォルト値は `AES256` です。
- `spark.shuffle.storage.s3.serverSideEncryption.kms.key`: aws:kms が有効になっている場合の KMS キーの ARN。

これらの構成に加え、ユースケースに応じて適切な暗号化が適用されるように `spark.hadoop.fs.s3.enableServerSideEncryption`、その他の環境固有の構成などの設定が必要になる場合があります。

プラグインのバージョン

このプラグインは、AWS Glue のバージョンごとに関連付けられている Spark バージョンで、それぞれサポートされています。次の表に、AWS Glue のバージョン、Spark バージョン、およびプラグインのソフトウェアバイナリ用の Amazon S3 ロケーションと関連付けたプラグインバージョンを示します。

AWS Glue バージョン	Spark バージョン	プラグインバージョン	Amazon S3 ロケーション
3.0	3.1	3.1-amzn-LATEST	s3://aws-glue-etl-artifacts/release/com/amazonaws/chopper-plugin/3.1-amzn-0/chopper-plugin-3.1-amzn-LATEST.jar
4.0	3.3	3.3-amzn-LATEST	s3://aws-glue-etl-artifacts/release/com/amazonaws/chopper-plugin/3.3-amzn-0/chopper-plugin-3.3-amzn-LATEST.jar

ライセンス

このプラグインのソフトウェアバイナリは、Apache-2.0 License の下でライセンスされています。

モニタリングAWS GlueSpark ジョブ

トピック

- AWS Glue Studio で使用可能な Spark メトリクス
- Apache Spark ウェブ UI を使用したジョブのモニタリング
- AWS Glue ジョブ実行インサイトでのモニタリング
- Amazon CloudWatch によるモニタリング
- ジョブのモニタリングとデバッグ

AWS Glue Studio で使用可能な Spark メトリクス

[Metrics] (メトリクス) タブには、ジョブが実行され実行とプロファイリングが有効になっているときに収集されるメトリクスが表示されます。以下のグラフが Spark ジョブに表示されます。

- ETL データ移動
- メモリプロファイル: ドライバーおよびエグゼキューター

[View additional metrics (追加のメトリクスの表示)] を選択し、以下のグラフを表示します。

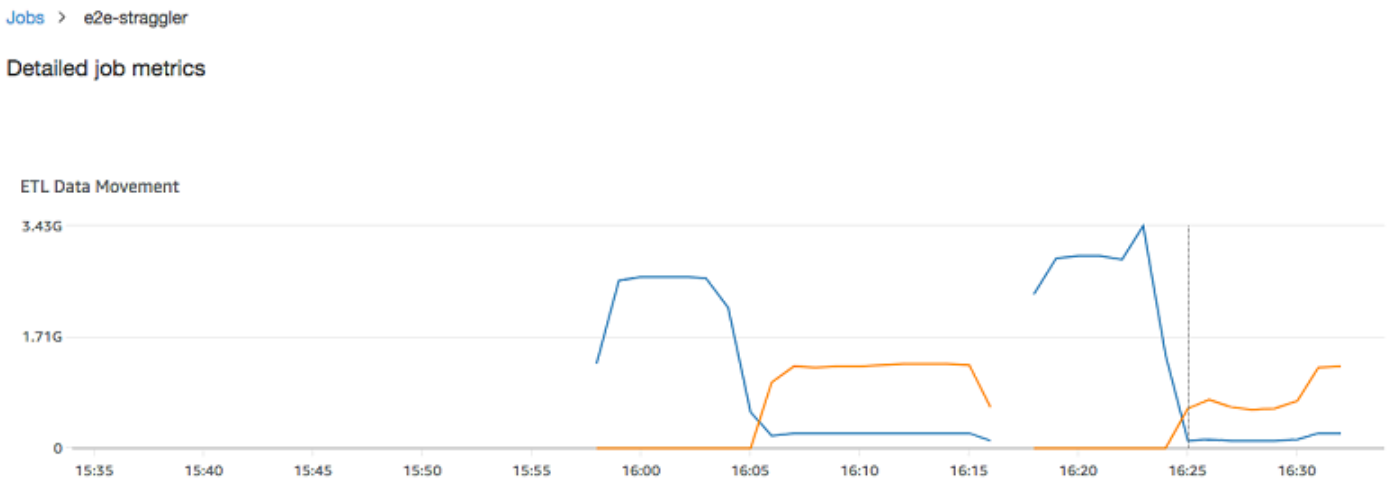
- ETL データ移動
- メモリプロファイル: ドライバーおよびエグゼキューター
- エグゼキューター間のデータシャッフル
- CPU 負荷: ドライバーおよびエグゼキューター
- ジョブの実行: アクティブなエグゼキューター、完了したステージ、必要な最大エグゼキューター

これらのグラフのデータは、ジョブでメトリクスの収集が有効になっている場合に CloudWatch メトリクスにプッシュされます。メトリクスを有効にしてグラフを解釈する方法の詳細については、「ジョブのモニタリングとデバッグ」を参照してください。

Example ETL データ移動グラフ

ETL データ移動グラフには、以下のメトリクスが表示されます。

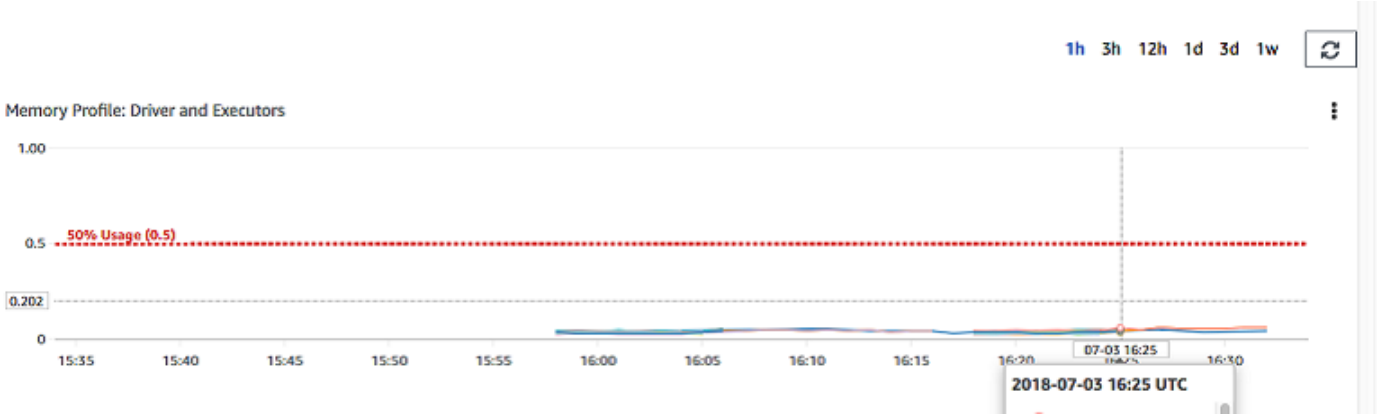
- すべてのエグゼキューターにより Amazon S3 から読み取られたバイト数 – glue.ALL.s3.filesystem.read_bytes
- すべてのエグゼキューターにより Amazon S3 に書き込まれたバイト数 – glue.ALL.s3.filesystem.write_bytes



Example メモリプロファイルグラフ

メモリプロファイルグラフには、以下のメトリクスが表示されます。

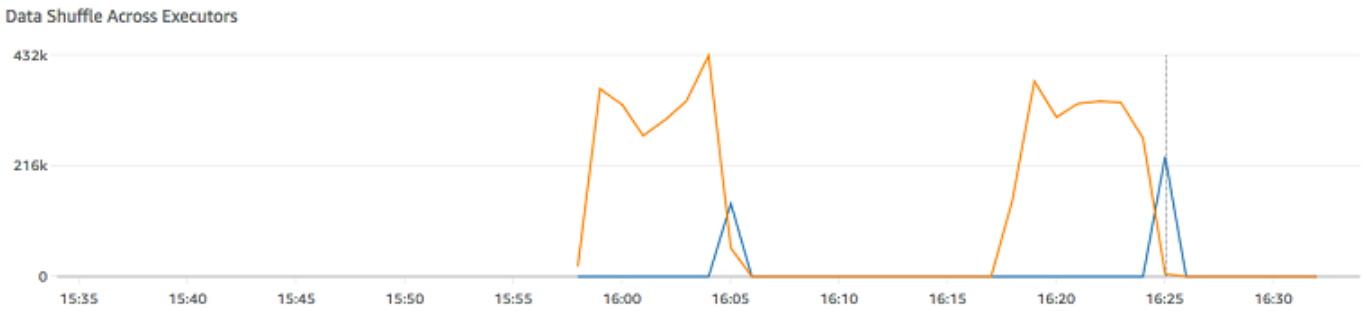
- このドライバーの JVM ヒープにより使用されるメモリの割合を、スケール 0 ~ 1 で、ドライバーごと、executorId により識別されるエグゼキューターごと、またはすべてのエグゼキューターごとに表示 –
  - glue.driver.jvm.heap.usage
  - glue.*executorId*.jvm.heap.usage
  - glue.ALL.jvm.heap.usage



Example エグゼキュター間のデータシャッフルグラフ

エグゼキュター間でのデータシャッフルグラフには、以下のメトリクスが表示されます。

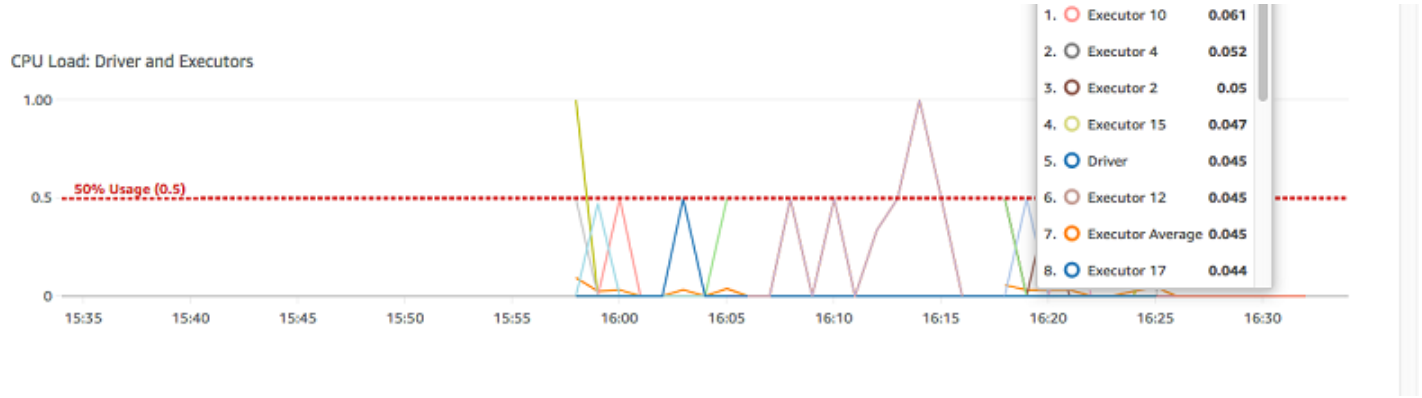
- すべてのエグゼキュター間でデータをシャッフルするためにそれらのエグゼキューターにより読み取られたバイト数 — glue.driver.aggregate.shuffleLocalBytesRead
- すべてのエグゼキュター間でデータをシャッフルするためにそれらのエグゼキューターにより書き込まれたバイト数 — glue.driver.aggregate.shuffleBytesWritten



Example CPU ロードグラフ

CPU ロードグラフには、以下のメトリクスが表示されます。

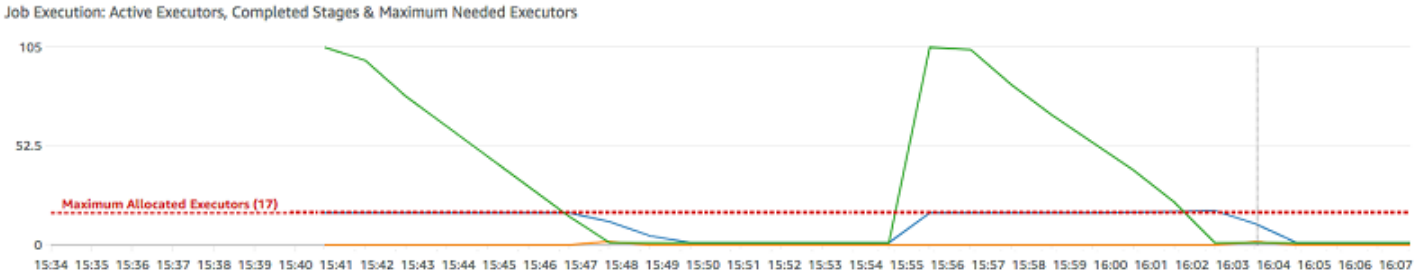
- 使用された CPU システムロードの割合を、スケール 0 ~ 1 で、ドライバーごと、executorId により識別されるエグゼキューターごと、またはすべてのエグゼキューターごとに表示 –
  - glue.driver.system.cpuSystemLoad
  - glue.*executorId*.system.cpuSystemLoad
  - glue.ALL.system.cpuSystemLoad



Example ジョブの実行グラフ

ジョブの実行グラフには、以下のメトリクスが表示されます。

- アクティブに実行されているエグゼキューターの数 — glue.driver.ExecutorAllocationManager.executors.numberAllExecutors
- 完了したステージの数 — glue.aggregate.numCompletedStages
- 必要なエグゼキューターの最大数 — glue.driver.ExecutorAllocationManager.executors.numberMaxNeededExecutors



Apache Spark ウェブ UI を使用したジョブのモニタリング

Apache Spark ウェブ UI を使用して、AWS Glue ジョブシステムで実行されている AWS Glue ETL ジョブと、 AWS Glue 開発エンドポイントで実行されている Spark アプリケーションをモニタリングおよびデバッグできます。Spark UI では、ジョブごとに以下の項目を確認できます。

- 各 Spark ステージのイベントタイムライン
- ジョブの Directed Acyclic Graph (DAG)
- SparkSQL クエリの物理プランと論理プラン
- 各ジョブの基盤となる Spark 環境変数

Spark Web UI の使用の詳細については、Spark ドキュメントの「Web UI」を参照してください。Spark UI の結果を解釈してジョブのパフォーマンスを向上させる方法のガイダンスについては、「AWS 規範的ガイダンス」の「Best practices for performance tuning AWS Glue for Apache Spark jobs」を参照してください。

AWS Glue コンソールで Spark UI を表示できます。これは、AWS Glue ジョブが AWS Glue 3.0 以降のバージョンで実行され、新しいジョブのデフォルトである (レガシーではなく) 標準形式でログが生成される場合に使用できます。　ログファイルが 0.5 GB を超える場合は、AWS Glue 4.0 以降のバージョンのジョブ実行のローリングログサポートを有効にして、ログのアーカイブ、分析、トラブルシューティングを簡素化できます。

Spark UI を有効にするには、AWS Glue コンソールまたは AWS Command Line Interface (AWS CLI) を使用します。Spark UI を有効にすると、AWS Glue ETL ジョブと、AWS Glue 開発エンドポイントの Spark アプリケーションの Spark イベントログを Amazon Simple Storage Service (Amazon S3) の指定した場所にバックアップできます。Amazon S3 にバックアップされたイベントログは、ジョブの実行中 (リアルタイム) でも、ジョブの完了後でも Spark UI で使用できます。ログが Amazon S3 に残っている間は、AWS Glue コンソールの Spark UI でログを表示できます。

アクセス許可

AWS Glue コンソールで Spark UI を使用するには、`UseGlueStudio` を使用するか、個々のサービス API をすべて追加できます。Spark UI を完全に使用するには、すべての API が必要です。ただし、ユーザーはきめ細かくアクセスするために、IAM 許可にサービス API を追加して SparkUI の機能にアクセスできます。

`RequestLogParsing` はログの解析を行うため、最も重要です。残りの API は、それぞれの解析済みデータを読み取るためのものです。たとえば、`GetStages` は Spark ジョブのすべてのステージに関するデータへのアクセスを提供します。

`UseGlueStudio` にマッピングされた Spark UI サービス API のリストは、以下のサンプルポリシーに記載されています。以下のポリシーは、Spark UI 機能のみを使用するためのアクセスを提供します。Amazon S3 や IAM などの許可をさらに追加するには、「AWS Glue Studio のカスタム IAM ポリシーの作成」を参照してください。

`UseGlueStudio` にマッピングされた Spark UI サービス API のリストは、以下のサンプルポリシーに記載されています。Spark UI サービス API を使用するとき、`glue:<ServiceAPI>` という名前空間を使用してください。

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

```
 "Statement": [
  {
   "Sid": "AllowGlueStudioSparkUI",
   "Effect": "Allow",
            "Action": [
                "glue:RequestLogParsing",
                "glue:GetLogParsingStatus",
                "glue:GetEnvironment",
                "glue:GetJobs",
                "glue:GetJob",
                "glue:GetStage",
                "glue:GetStages",
                "glue:GetStageFiles",
                "glue:BatchGetStageFiles",
                "glue:GetStageAttempt",
                "glue:GetStageAttemptTaskList",
                "glue:GetStageAttemptTaskSummary",
                "glue:GetExecutors",
                "glue:GetExecutorsThreads",
                "glue:GetStorage",
                "glue:GetStorageUnit",
                "glue:GetQueries",
                "glue:GetQuery"
            ],
   "Resource": [
    "*"
   ]
  }
 ]
}
```

制約事項

- AWS Glue コンソールの Spark UI は、レガシーログ形式であるため、2023 年 11 月 20 日よりも前に発生したジョブ実行には使用できません。
- AWS Glue コンソールの Spark UI は、ストリーミング ジョブでデフォルトで生成されるログなど、AWS Glue 4.0 のローリング ログをサポートしています。生成されたすべてのロールされたログファイルの最大合計は 2 GB です。ロールログがサポートされていない AWS Glue ジョブの場合、SparkUI でサポートされるログイベントファイルの最大サイズは 0.5 GB です。
- サーバーレス Spark UI は、VPC のみがアクセスできる Amazon S3 バケットに保存されている Spark イベントログでは使用できません。

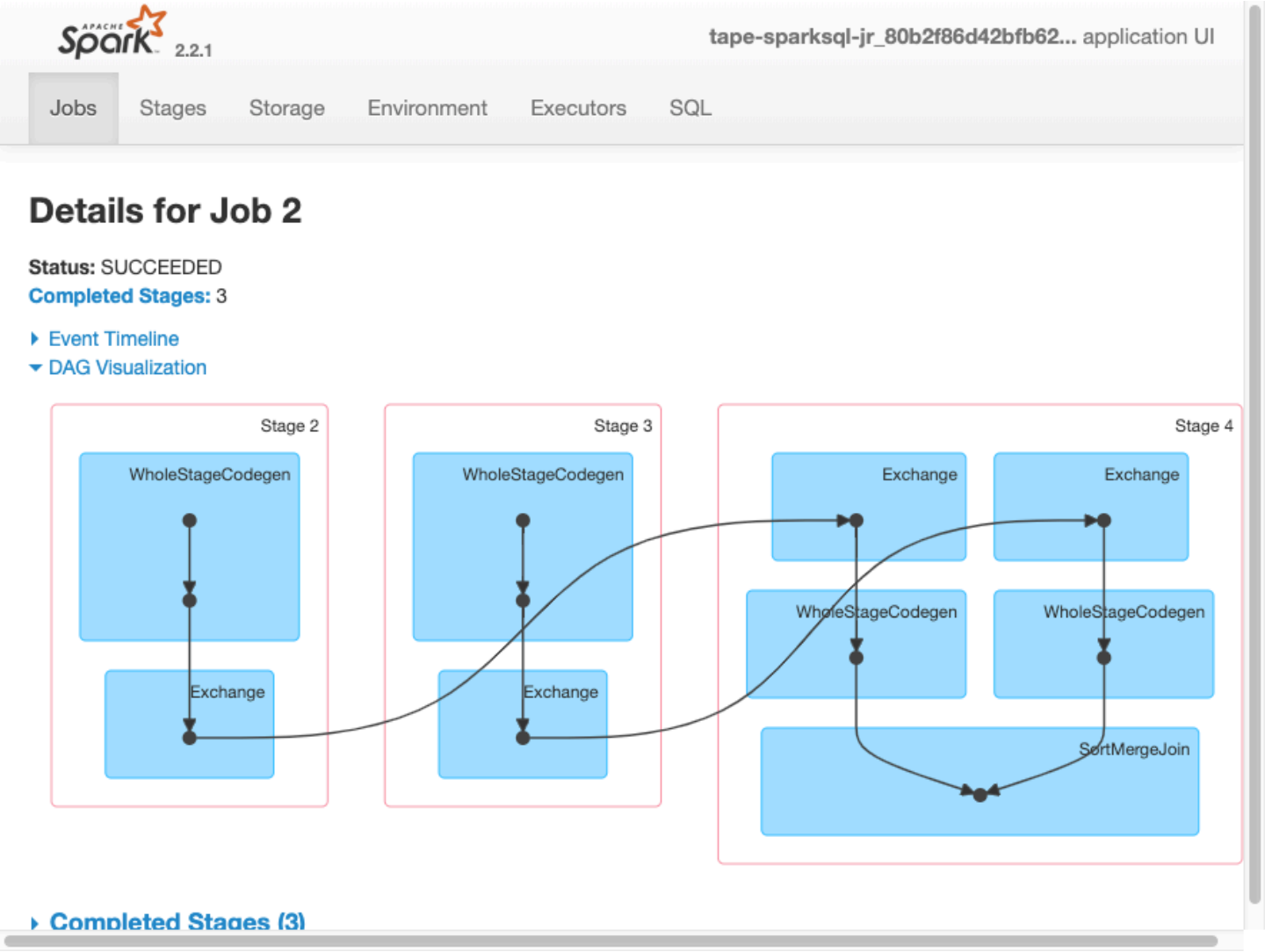
例: Apache Spark Web UI

この例では、Spark UI を使用してジョブのパフォーマンスを理解する方法を示します。 スクリーンショットは、自己管理型の Spark 履歴サーバーが提供する Spark ウェブ UI を示しています。AWS Glue コンソールの Spark UI にも同様のビューが表示されます。 Spark Web UI の使用の詳細については、Spark ドキュメントの「Web UI」を参照してください。

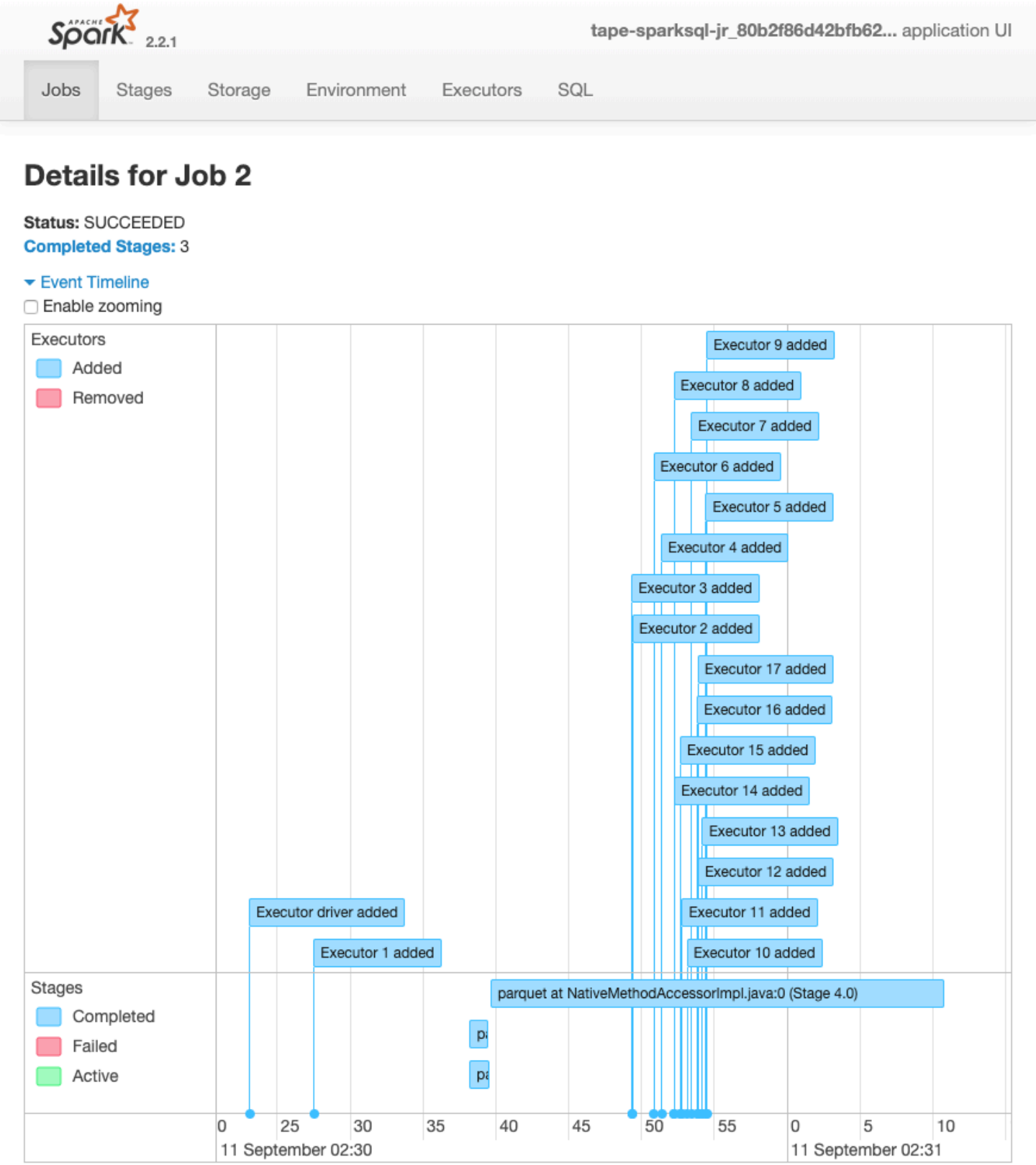
次の例に示す Spark アプリケーションでは、2 つのデータソースから読み取り、結合変換を実行し、それを Parquet 形式で Amazon S3 に書き込みます。

```
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job
from pyspark.sql.functions import count, when, expr, col, sum, isnull
from pyspark.sql.functions import countDistinct
from awsglue.dynamicframe import DynamicFrame

args = getResolvedOptions(sys.argv, ['JOB_NAME'])

sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session

job = Job(glueContext)
job.init(args['JOB_NAME'])

df_persons = spark.read.json("s3://awsglue-datasets/examples/us-legislators/all/
persons.json")
df_memberships = spark.read.json("s3://awsglue-datasets/examples/us-legislators/all/
memberships.json")

df_joined = df_persons.join(df_memberships, df_persons.id == df_memberships.person_id,
 'fullouter')
df_joined.write.parquet("s3://aws-glue-demo-sparkui/output/")

job.commit()
```

次の DAG 可視化は、この Spark ジョブのさまざまなステージを示しています。

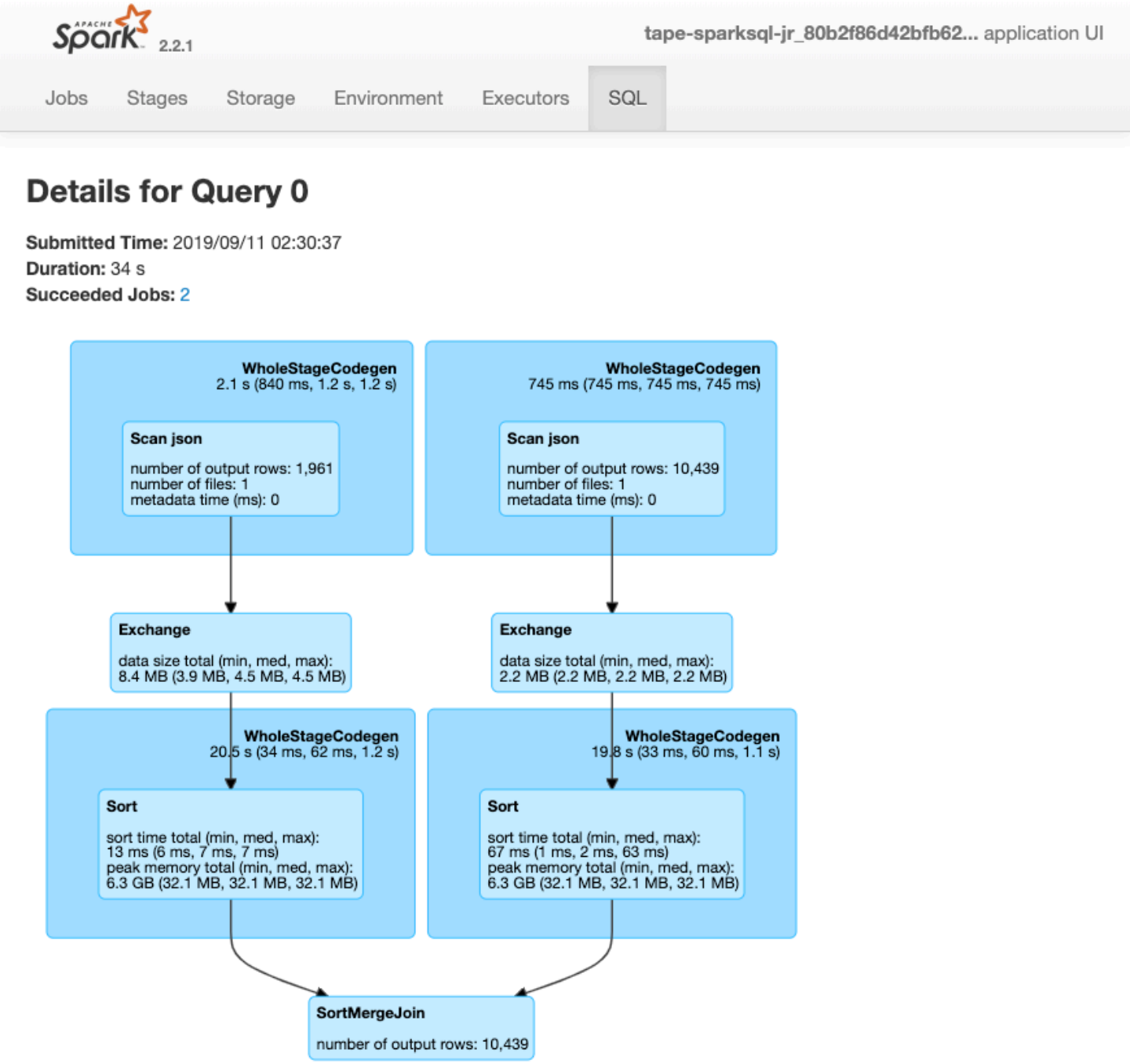


次に示すジョブのイベントタイムラインは、さまざまな Spark エグゼキューターの開始、実行、終了を示しています。


Spark 2.2.1
tape-sparksql-jr_80b2f86d42bfb62... application UI
Jobs
Stages
Storage
Environment
Executors
SQL
Details for Job 2
Status: SUCCEEDED
Completed Stages: 3
Event Timeline
Enable zooming
Executors
Added
Removed
Executor 9 added
Executor 8 added
Executor 7 added
Executor 6 added
Executor 5 added
Executor 4 added
Executor 3 added
Executor 2 added
Executor 17 added
Executor 16 added
Executor 15 added
Executor 14 added
Executor 13 added
Executor 12 added
Executor 11 added
Executor 10 added
Executor driver added
Executor 1 added
Stages
Completed
Failed
Active
parquet at NativeMethodAccessorImpl.java:0 (Stage 4.0)
0
25
30
35
40
45
50
55
0
5
10
11 September 02:30
11 September 02:31
DAG Visualization
Completed Stages (3)

次の画面は、SparkSQL クエリプランの詳細を示しています。

- 解析された論理プラン
- 分析された論理プラン
- 最適化された論理プラン
- 実行のための物理プラン


Spark 2.2.1
tape-sparksql-jr_80b2f86d42bfb62... application UI
Jobs
Stages
Storage
Environment
Executors
SQL
Details for Query 0
Submitted Time: 2019/09/11 02:30:37
Duration: 34 s
Succeeded Jobs: 2
WholeStageCodegen
2.1 s (840 ms, 1.2 s, 1.2 s)
Scan json
number of output rows: 1,961
number of files: 1
metadata time (ms): 0
WholeStageCodegen
745 ms (745 ms, 745 ms, 745 ms)
Scan json
number of output rows: 10,439
number of files: 1
metadata time (ms): 0
Exchange
data size total (min, med, max):
8.4 MB (3.9 MB, 4.5 MB, 4.5 MB)
Exchange
data size total (min, med, max):
2.2 MB (2.2 MB, 2.2 MB, 2.2 MB)
WholeStageCodegen
20.5 s (34 ms, 62 ms, 1.2 s)
Sort
sort time total (min, med, max):
13 ms (6 ms, 7 ms, 7 ms)
peak memory total (min, med, max):
6.3 GB (32.1 MB, 32.1 MB, 32.1 MB)
WholeStageCodegen
19.8 s (33 ms, 60 ms, 1.1 s)
Sort
sort time total (min, med, max):
67 ms (1 ms, 2 ms, 63 ms)
peak memory total (min, med, max):
6.3 GB (32.1 MB, 32.1 MB, 32.1 MB)
SortMergeJoin
number of output rows: 10,439
Details
== Parsed Logical Plan ==
Join FullOuter, (id#14 = person_id#50)
:-
Relation[birth_date#8,contact_details#9,death_date#10,family_name#11,gender#12,given_name#13,id#14,identifiers#15
,image#16,images#17,links#18,name#19,other_names#20,sort_name#21] json
+-
Relation[area_id#45,end_date#46,legislative_period_id#47,on_behalf_of_id#48,organization_id#49,person_id#50,role#
51,start_date#52] json
== Analyzed Logical Plan ==
birth_date: string, contact_details: array<struct<type:string,value:string>>, death_date: string, family_name:
string, gender: string, given_name: string, id: string, identifiers:
array<struct<identifier:string,scheme:string>>, image: string, images: array<struct<url:string>>, links:
array<struct<note:string,url:string>>, name: string, other_names:
array<struct<lang:string,name:string,note:string>>, sort_name: string, area_id: string, end_date: string,
legislative_period_id: string, on_behalf_of_id: string, organization_id: string, person_id: string, role: string,
start_date: string
Join FullOuter, (id#14 = person_id#50)

トピック

- AWS Glue ジョブ用の Apache Spark ウェブ UI の有効化
- Spark 履歴サーバーの起動

AWS Glue ジョブ用の Apache Spark ウェブ UI の有効化

Apache Spark ウェブ UI を使用して、AWS Glue ジョブシステムで実行されている AWS Glue ETL ジョブをモニタリングおよびデバッグできます。Spark UI は、AWS Glue コンソールまたは AWS Command Line Interface (AWS CLI) を使用して設定できます。

30 秒ごとに、AWS Glue は指定した Amazon S3 パスに Spark イベントログをバックアップします。

トピック

- Spark UI の設定 (コンソール)
- Spark UI の設定 (AWS CLI)
- Notebook を使用するセッション用の Spark UI の設定
- ローリングログを有効にする

Spark UI の設定 (コンソール)

以下のステップに従い、AWS Management Consoleを使用して Spark UI を設定します。AWS Glue ジョブを作成する際、Spark UI はデフォルトで有効になります。

ジョブを作成または編集するときに Spark UI をオンにするには

1. AWS Management Consoleにサインインし、AWS Glue コンソール (https://console.aws.amazon.com/glue/) を開きます。
2. ナビゲーションペインで ジョブを選択します。
3. [ジョブを追加] を選択するか、または既存のジョブを選択します。
4. [ジョブの詳細] で、[詳細プロパティ] を開きます。
5. [Spark UI] タブで、[Spark UI ログを Amazon S3 に書き込む] を選択します。
6. ジョブの Spark イベントログを保存するための Amazon S3 パスを指定します。ジョブでセキュリティ設定を使用する場合、暗号化は Spark UI ログファイルにも適用されることに注意してください。　詳細については、「AWS Glue によって書き込まれたデータの暗号化」を参照してください。

7. [Spark UI のログ記録とモニタリングの設定] で、次のように操作します。

   - AWS Glue コンソールで表示するログを生成している場合は、[標準] を選択します。
   - Spark 履歴サーバーで表示するログを生成している場合は、[レガシー] を選択します。
   - 両方を生成することを選択することもできます。

Spark UI の設定 (AWS CLI)

Spark UI で表示するログを生成するには、AWS Glue コンソールで AWS CLI を使用して、次のジョブパラメータを AWS Glue ジョブに渡します。詳細については、「the section called “ジョブのパラメータ”」を参照してください。

```
'--enable-spark-ui': 'true',
'--spark-event-logs-path': 's3://s3-event-log-path'
```

ログをレガシーの場所に配布するには、`--enable-spark-ui-legacy-path` パラメータを `"true"` に設定します。両方の形式でログを生成したくない場合は、`--enable-spark-ui` パラメータを削除します。

Notebook を使用するセッション用の Spark UI の設定

> **Warning**
>
> AWS Glue インタラクティブセッションは現在、コンソールでの Spark UI をサポートしていません。Spark 履歴サーバーを設定します。

AWS Glue ノートブック を使用する場合は、セッションを開始する前に SparkUI の設定をセットアップします。そのためには、次のように `%%configure` セルマジックを使用します。

```
%%configure { “--enable-spark-ui”: “true”, “--spark-event-logs-path”: “s3://path” }
```

ローリングログを有効にする

AWS Glue ジョブの SparkUI とローリングログファイルを有効にすると、いくつかの利点があります。

- ローリングログイベントファイル – ローリングログイベントファイルを有効にすると、AWS Glue はジョブ実行の各ステップごとに個別のログファイルを生成し、特定のステージまたは変換に固有の問題の特定とトラブルシューティングを容易にします。
- ログ管理の向上 – ログファイルをローリングすると、ログファイルをより効率的に管理できます。1 つの大きなログファイルを持つ代わりに、ログはジョブ実行ステージに基づいて、より小さく管理しやすいファイルに分割されます。これにより、ログのアーカイブ、分析、トラブルシューティングを簡素化できます。
- 障害耐性の向上 – AWS Glue ジョブが失敗または中断された場合、ローリングログイベントは、最後に成功したステージに関する貴重な情報を提供するため、最初から開始するのではなく、その時点からジョブを再開しやすくなります。
- コストの最適化 – ローリングログファイルを有効にすることで、ログファイルに関連するストレージコストを削減できます。1 つの大きなログファイルを保存する代わりに、より小さく、管理しやすいログファイルを保存します。これは、特に長時間実行されるジョブや複雑なジョブの場合、費用対効果が高くなります。

新しい環境では、ユーザーは以下を通じてローリングログを明示的に有効にできます。

```
'–conf': 'spark.eventLog.rolling.enabled=true'
```

または

```
'–conf': 'spark.eventLog.rolling.enabled=true –conf
spark.eventLog.rolling.maxFileSize=128m'
```

ローリングログが有効な場合、spark.eventLog.rolling.maxFileSize はロールオーバーする前のイベントログファイルの最大サイズを指定します。このオプションパラメータが指定されていない場合のデフォルト値は 128 MB です。最小は 10 MB です。

生成されたすべてのロールされたログファイルの最大合計は 2 GB です。ローリングログがサポートされていない AWS Glue ジョブの場合、SparkUI でサポートされるログイベントファイルの最大サイズは 0.5 GB です。

追加の設定を渡すことで、ストリーミングジョブのローリングログをオフにすることができます。ログファイルが非常に大きいと、管理にコストがかかる場合があることに注意してください。

ローリングログをオフにするには、次の設定を提供します。

```
'--spark-ui-event-logs-path': 'true',
```

```
'--conf': 'spark.eventLog.rolling.enabled=false'
```

Spark 履歴サーバーの起動

Spark 履歴サーバーを使用して、独自のインフラストラクチャ上で Spark ログを視覚化できます。(レガシーではなく) 標準形式で生成されたログで、AWS Glue 4.0 以降のバージョンで AWS Glue ジョブが実行された場合、AWS Glue コンソールで同じ視覚化を確認することができます。詳細については、「the section called “Spark UI を使用したモニタリング”」を参照してください。

EC2 インスタンスでサーバーをホストする AWS CloudFormation テンプレートを使用して Spark 履歴サーバーを起動するか、Docker を使用してローカルで起動できます。

トピック

- AWS CloudFormation を使用した Spark 履歴サーバーの起動と Spark UI の表示
- Docker を使用した Spark 履歴サーバーの起動と Spark UI の表示

AWS CloudFormation を使用した Spark 履歴サーバーの起動と Spark UI の表示

AWS CloudFormation テンプレートを使用して Apache Spark 履歴サーバーを起動し、Spark ウェブ UI を表示できます。これらのテンプレートは、要件に応じて変更する必要があるサンプルです。

AWS CloudFormation を使用して Spark 履歴サーバーを起動し、Spark UI を表示するには

1. 次の表に示す起動スタックボタンの 1 つを選択します。これにより、AWS CloudFormation コンソールでスタックが起動されます。

リージョン	起動する
米国東部(オハイオ)	Launch Stack
米国東部 (バージニア北部)	Launch Stack
米国西部 (北カリフォルニア)	Launch Stack
米国西部 (オレゴン)	Launch Stack

リージョン	起動する
アフリカ (ケープタウン)	Launch Stack
アジアパシフィック (香港)	Launch Stack
アジアパシフィック (ムンバイ)	Launch Stack
アジアパシフィック (大阪)	Launch Stack
アジアパシフィック (ソウル)	Launch Stack
アジアパシフィック (シンガポール)	Launch Stack
アジアパシフィック (シドニー)	Launch Stack
アジアパシフィック (東京)	Launch Stack
カナダ (中部)	Launch Stack
欧州 (フランクフルト)	Launch Stack
欧州 (アイルランド)	Launch Stack
欧州 (ロンドン)	Launch Stack
欧州 (ミラノ)	Launch Stack
ヨーロッパ (パリ)	Launch Stack
ヨーロッパ (ストックホルム)	Launch Stack

リージョン	起動する
中東 (バーレーン)	
南米 (サンパウロ)	

2. [Specify template (テンプレートの指定)] ページで、[次へ] を選択します。
3. [Specify stack details (スタック詳細の指定)] ページで、[Stack name (スタック名)] に入力します。パラメータの下に追加情報を入力します。

   a. Spark UI の設定

      以下の情報を指定します。

      - IP アドレス範囲 – Spark UI の表示に使用できる IP アドレス範囲。特定の IP アドレス範囲からのアクセスを制限する場合は、カスタム値を使用する必要があります。
      - 履歴サーバーポート – Spark UI のポート。デフォルト値を使用できます。
      - イベントログディレクトリ – AWS Glue ジョブまたは開発エンドポイントから保存された Spark イベントログの場所を選択します。イベントログのパススキームとして **s3a://** を使用する必要があります。
      - Spark パッケージの場所 – デフォルト値を使用できます。
      - キーストアパス – HTTPS の SSL/TLS キーストアパス。カスタムキーストアファイルを使用する場合は、ここで S3 パス `s3://path_to_your_keystore_file` を指定できます。このパラメータを空のままにすると、自己署名証明書ベースのキーストアが生成されて使用されます。
      - キーストアパスワード – HTTPS 用の SSL/TLS キーストアパスワードを入力します。

   b. EC2 インスタンスの設定

      以下の情報を指定します。

      - インスタンスタイプ – Spark 履歴サーバーをホストする Amazon EC2 インスタンスのタイプ。このテンプレートによってアカウントで Amazon EC2 インスタンスが起動されるため、Amazon EC2 のコストがアカウントに別個に課金されます。
      - 最新の AMI ID – Spark 履歴サーバーインスタンスの Amazon Linux 2 の AMI ID。デフォルト値を使用できます。

- VPC ID – Spark 履歴サーバーインスタンスの Virtual Private Cloud (VPC) の ID。アカウントで利用可能な VPC のいずれかを使用できます。デフォルトの VPC と デフォルトのネットワーク ACL を使用することは推奨されません。詳細については、Amazon VPC ユーザーガイド の「デフォルト VPC とデフォルトサブネット」および「VPC を作成する」を参照してください。
- サブネット ID – Spark 履歴サーバーインスタンスの ID。VPC 内の任意のサブネットを使用できます。クライアントからサブネットのネットワークにアクセスできる必要があります。インターネット経由でアクセスする場合は、ルートテーブルにインターネットゲートウェイがあるパブリックサブネットを使用する必要があります。

c. [Next] を選択します。

4. [Configure stack options] ページで、CloudFormation がスタック内のリソースを作成、変更、または削除する方法を決定するために現在のユーザー認証情報を使用する場合は、[Next] をクリックします。アクセス許可セクションで、現在のユーザー権限の代わりに使用するロールを指定し、[次へ] を選択することもできます。
5. [確認] ページで、テンプレートを確認します。

   [I acknowledge that AWS CloudFormation might create IAM resources] (AWS CloudFormation によって IAM リソースが作成される場合があることを承認します) を選択し、[Create stack] (スタックの作成) を選択します。
6. スタックが作成されるまで待ちます。
7. [Outputs (出力)] タブをクリックします。

   a. パブリックサブネットを使用している場合は、 SparkUiPublicUrl の URL をコピーします。

   b. プライベートサブネットを使用している場合は、 SparkUiPrivateUrl の URL をコピーします。
8. ウェブブラウザを開き、URL を貼り付けます。これにより、指定したポートで HTTPS を使用してサーバーにアクセスできます。ブラウザがサーバーの証明書を認識しない場合があります。この場合、保護を上書きして続行する必要があります。

Docker を使用した Spark 履歴サーバーの起動と Spark UI の表示

ローカルアクセスを希望する (Apache Spark 履歴サーバーの EC2 インスタンスを使用しない) 場合は、Docker を使用して Apache Spark 履歴サーバーを起動し、Spark UI をローカルに表示することもできます。この Dockerfile は、要件に応じて変更する必要があるサンプルです。

前提条件

ラップトップに Docker をインストールする方法については、Docker Engine community を参照してください。

Docker を使用して Spark 履歴サーバーを起動し、Spark UI をローカルに表示するには

1. GitHub からファイルをダウンロードします。

   Dockerfileとpom.xml を [AWS Glueコードサンプル]からダウンロードします。

2. アクセス AWS にユーザー認証情報を使用するか、フェデレーティッドユーザー認証情報を使用するかを決定します。

   - 現在のユーザー認証情報を使用してアクセスを AWS にするには、 AWS_ACCESS_KEY_ID と AWS_SECRET_ACCESS_KEY で使用する値を docker run コマンドで取得します。詳細については、「 IAM ユーザーガイド」の「IAM ユーザーのアクセスキーの管理」を参照してください。
   - SAML 2.0 フェデレーティッドユーザーを使用して AWS にアクセスするには、AWS_ACCESS_KEY_ID、AWS_SECRET_ACCESS_KEY、 AWS_SESSION_TOKEN で値を取得します。詳細については、「一時的なセキュリティ認証情報のリクエスト」を参照してください。

3. docker run コマンドで使用するイベントログディレクトリの場所を決定します。
4. ローカルディレクトリ内のファイルを使用して、名前 glue/sparkui とタグ latest を指定して、Docker イメージを構築します。

   ```
   $ docker build -t glue/sparkui:latest .
   ```

5. Docker コンテナを作成してスタートします。

   次のコマンドでは、ステップ 2 および 3 で事前に取得した値を使用します。

   a. ユーザー認証情報を使用して docker コンテナを作成するには、次のようなコマンドを使用します。

      ```
      docker run -itd -e SPARK_HISTORY_OPTS="$SPARK_HISTORY_OPTS -
      Dspark.history.fs.logDirectory=s3a://path_to_eventlog
       -Dspark.hadoop.fs.s3a.access.key=AWS_ACCESS_KEY_ID -
      Dspark.hadoop.fs.s3a.secret.key=AWS_SECRET_ACCESS_KEY"
       -p 18080:18080 glue/sparkui:latest "/opt/spark/bin/spark-class
       org.apache.spark.deploy.history.HistoryServer"
      ```

b. 一時的な認証情報を使用して Docker コンテナを作成するには、`org.apache.hadoop.fs.s3a.TemporaryAWSCredentialsProvider` をプロバイダとして指定し、ステップ 2 で取得した認証情報値を指定します。詳細については、「一時的な AWSCredentialsProvider でのセッション資格情報の使用のHadoop: Amazon Web Services との統合」のドキュメント を参照してください。

```
docker run -itd -e SPARK_HISTORY_OPTS="$SPARK_HISTORY_OPTS -
Dspark.history.fs.logDirectory=s3a://path_to_eventlog
 -Dspark.hadoop.fs.s3a.access.key=AWS_ACCESS_KEY_ID -
Dspark.hadoop.fs.s3a.secret.key=AWS_SECRET_ACCESS_KEY
 -Dspark.hadoop.fs.s3a.session.token=AWS_SESSION_TOKEN
 -
Dspark.hadoop.fs.s3a.aws.credentials.provider=org.apache.hadoop.fs.s3a.TemporaryAWSCred
 -p 18080:18080 glue/sparkui:latest "/opt/spark/bin/spark-class
 org.apache.spark.deploy.history.HistoryServer"
```

> **Note**
>
> これらの設定パラメータは、 Hadoop-AWS モジュールに由来します。場合によっては、ユースケースに基づいて特定の設定を追加する必要があります。たとえば、隔離されたリージョンのユーザーは、 `spark.hadoop.fs.s3a.endpoint` を設定する必要があります。

6. ブラウザで `http://localhost:18080` を開き、Spark UI をローカルに表示します。

AWS Glue ジョブ実行インサイトでのモニタリング

AWS Glue ジョブ実行インサイトは、AWS Glue ジョブのデバッグとジョブの最適化を簡素化する AWS Glue の機能です。AWS Glue は、Spark UI、および AWS Glue ジョブをモニタリングするためのCloudWatch Logs とメトリクスを提供します。この機能を使用すると、AWS Glue ジョブの実行に関する情報を取得できます:

- エラーが発生した AWS Glue ジョブスクリプトの行番号。
- ジョブの失敗の直前に Spark クエリプランで最後に実行された Spark アクション。
- 時系列ログストリームに表示される障害に関連する Spark 例外イベント。
- 根本原因の分析と、問題を解決するための推奨アクション (スクリプトのチューニングなど) 。

- 根本原因に対処する推奨アクションを含む共通の Spark イベント (Spark アクションに関連するログメッセージ) 。

これらのインサイトはすべて、AWS Glue ジョブの CloudWatch Logs 内の 2 つの新しいログストリームを使用して利用できます。

要件

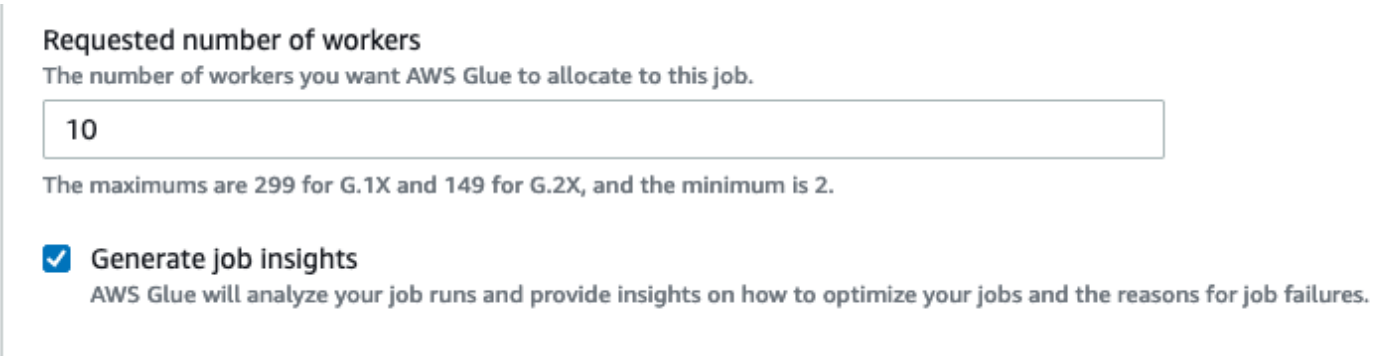
AWS Glue ジョブ実行インサイト機能は、AWS Glue バージョン 2.0、3.0、4.0 および 5.0 で利用できます。既存のジョブの移行ガイドに従って、古い AWS Glue バージョンからアップグレードすることができます。

AWS Glue ETL ジョブのジョブ実行インサイトの有効化

ジョブ実行インサイトは、AWS Glue Studio または CLI によって有効にできます。

AWS Glue Studio

AWS Glue Studio でジョブを作成する場合、[ジョブの詳細] タブでジョブ実行インサイトを有効または無効にすることができます。[ジョブインサイトを生成する] ボックスが選択されていることを確認します。



コマンドライン

CLI を使用してジョブを作成する場合は、単一の新しいジョブパラメータでジョブの実行を開始できます: `--enable-job-insights = true`。

デフォルトでは、ジョブ実行インサイトログストリームは、AWS Glue連続ログ記録 (つまり `/aws-glue/jobs/logs-v2/`)で使用されるのと同じデフォルトロググループの下に作成されます。連続ログ記録に同じ引数のセットを使用して、カスタムロググループ名、ログフィルター、およびロググループ構成を設定できます。詳細については、「AWS Glue ジョブの連続ログ記録の有効化」を参照してください。

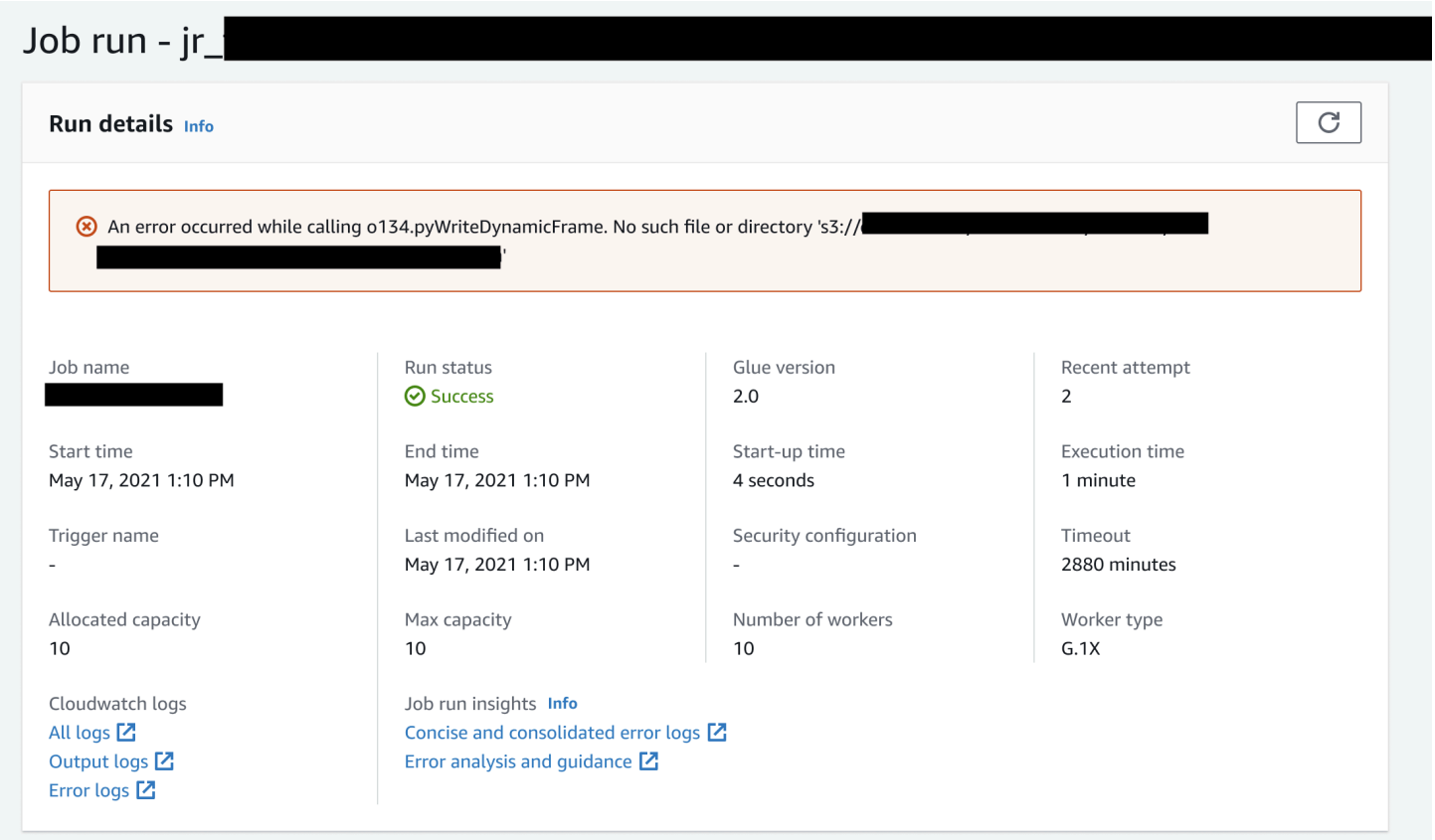
CloudWatch でのジョブ実行インサイトログストリームへのアクセス

ジョブ実行インサイト機能を有効にすると、ジョブ実行が失敗したときに 2 つのログストリームが作成されることがあります。ジョブが正常に終了すると、いずれのストリームも生成されません。

1. 例外分析ログストリーム:`<job-run-id>-job-insights-rca-driver`。このストリームは、以下を提供します:
   - エラーが発生した AWS Glue ジョブスクリプトの行番号。
   - Spark クエリプラン (DAG) で最後に実行された Spark アクション。
   - Spark ドライバおよび例外に関連するエグゼキュータからの簡潔な時系列イベント。完全なエラーメッセージ、失敗した Spark タスク、およびそのエグゼキューター ID などの詳細を調べて、必要に応じてさらに詳細な調査を行うために、特定のエグゼキューターのログストリームに集中できます。
2. ルールベースのインサイトストリーム:
   - エラーの修正方法に関する根本原因の分析とレコメンデーション (特定のジョブパラメータを使用してパフォーマンスを最適化するなど) 。
   - 関連する Spark イベントは、根本原因の分析および推奨アクションの基礎として機能します。

> **Note**
>
> 最初のストリームは、失敗したジョブ実行で例外の Spark イベントが使用できる場合にのみ存在し、2 番目のストリームは、失敗したジョブ実行に対して利用可能なインサイトがある場合にのみ存在します。例えば、ジョブが正常に終了した場合、どちらのストリームも生成されません。ジョブが失敗しても、失敗シナリオと一致するサービス定義のルールがない場合は、最初のストリームだけが生成されます。

AWS Glue Studio からジョブが作成された場合、上記のストリームへのリンクは、[ジョブ実行の詳細] タブ (ジョブ実行インサイト) の下で「簡潔で統合されたエラーログ」および「エラー分析とガイダンス」としても使用できます。



AWS Glue ジョブ実行インサイトの例

このセクションでは、ジョブ実行インサイト機能が、失敗したジョブの問題を解決するためにどのように役立つかの例を紹介します。この例では、ユーザーは AWS Glue ジョブで必要なモジュール (tensorflow) をインポートし、データに基づいて機械学習モデルを分析および構築するのを忘れていました。

```
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job
from pyspark.sql.types import *
from pyspark.sql.functions import udf,col

args = getResolvedOptions(sys.argv, ['JOB_NAME'])

sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
```

```
job = Job(glueContext)
job.init(args['JOB_NAME'], args)

data_set_1 = [1, 2, 3, 4]
data_set_2 = [5, 6, 7, 8]

scoresDf = spark.createDataFrame(data_set_1, IntegerType())

def data_multiplier_func(factor, data_vector):
    import tensorflow as tf
    with tf.compat.v1.Session() as sess:
        x1 = tf.constant(factor)
        x2 = tf.constant(data_vector)
        result = tf.multiply(x1, x2)
        return sess.run(result).tolist()

data_multiplier_udf = udf(lambda x:data_multiplier_func(x, data_set_2),
 ArrayType(IntegerType(),False))
factoredDf = scoresDf.withColumn("final_value", data_multiplier_udf(col("value")))
print(factoredDf.collect())
```

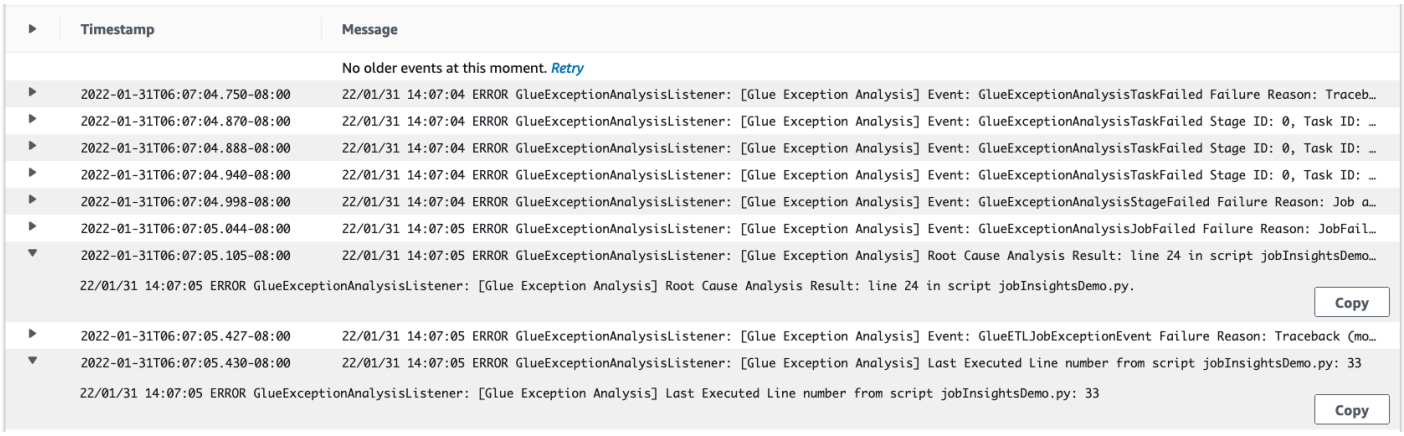
ジョブ実行インサイト機能がないと、ジョブが失敗すると、Spark によってスローされる次のメッセージのみが表示されます。

```
An error occurred while calling o111.collectToPython. Traceback (most
recent call last):
```

メッセージがあいまいなので、デバッグ体験が制限されます。この場合、この機能は 2 つの CloudWatch Logs ストリームに追加のインサイトを提供します。

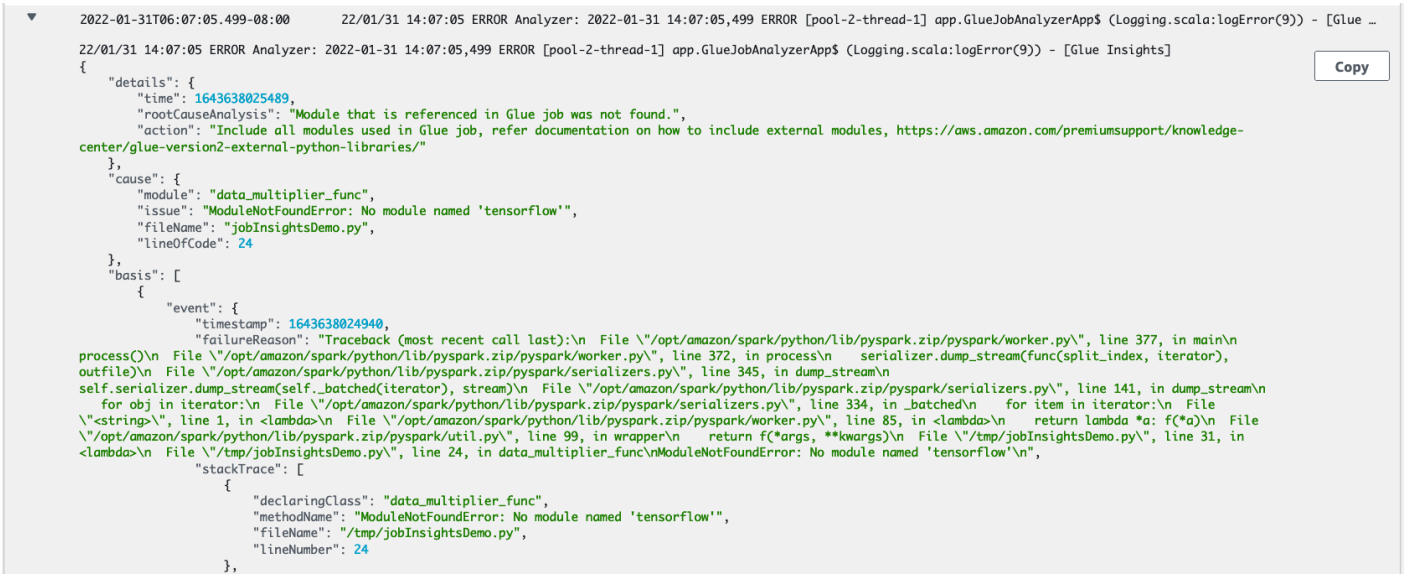
1. `job-insights-rca-driver` ログストリーム:

   - 例外イベント：このログストリームは、Spark ドライバおよびさまざまな分散ワーカーから収集された障害に関連する Spark 例外イベントを提供します。これらのイベントは、障害のあるコードが Spark タスク、エグゼキューター、および AWS Glue ワーカーに分散されたステージ全体で実行されるときに、例外の時系列の伝播を理解するのに役立ちます。
   - 行番号: このログストリームは、失敗の原因となった欠落している Python モジュールをインポートするための呼び出しを行った 21 行目を識別します。 また、スクリプトで最後に実行された行として、Spark アクション `collect()` の呼び出しである 24 行目も識別します。



2. `job-insights-rule-driver` ログストリーム:

   - 根本原因とレコメンデーション：スクリプトの障害の行番号と最後に実行された行番号に加えて、このログストリームには、根本原因の分析と、AWS Glue ドキュメントに従って、AWS Glue ジョブの追加の Python モジュールを使用するために必要なジョブパラメーターを設定するためのレコメンデーションが表示されます。
   - 基本イベント：このログストリームには、根本原因を推測してレコメンデーションを提供するために、サービス定義ルールで評価された Spark 例外イベントも表示されます。



Amazon CloudWatch によるモニタリング

Amazon CloudWatch を使用して AWS Glue をモニタリングすることで、AWS Glue から raw データを収集し、ほぼリアルタイムの読み取り可能なメトリクスに加工することができます。これらの統計は 2 週間記録されるため、履歴情報にアクセスしてウェブアプリケーションまたはサービスの動作をより的確に把握することができます。デフォルトでは、AWS Glue メトリクスデータは

CloudWatch に自動的に送信されます。詳細については、Amazon CloudWatch ユーザーガイドの「Amazon CloudWatch とは」、および「AWS Glue のメトリクス」を参照してください。

継続的なログ記録

AWS Glue は、AWS Glue ジョブのリアルタイムの連続ログ記録もサポートします。ジョブの連続ログ記録が有効になっている場合は、AWS Glue コンソールまたは CloudWatch コンソールダッシュボードでリアルタイムのログを表示できます。詳細については、「AWS Glue ジョブのログ記録」を参照してください。

オブザーバビリティメトリクス

[ジョブのオブザーバビリティメトリクス] が有効になっている場合、ジョブの実行時に追加の Amazon CloudWatch メトリクスが生成されます。AWS Glue オブザーバビリティメトリクスを使用して、AWS Glue の内部で何が起こっているかに関するインサイトを生成し、問題の優先順位付けと分析を改善できます。

トピック

- Amazon CloudWatch メトリクスを使用した AWS Glue のモニタリング
- AWS Glue ジョブプロファイルでの Amazon CloudWatch アラームの設定
- AWS Glue ジョブのログ記録
- AWS Glue オブザーバビリティメトリクスを使用したモニタリング

Amazon CloudWatch メトリクスを使用した AWS Glue のモニタリング

AWS Glue ジョブプロファイラーを使用して AWS Glue オペレーションをプロファイルおよびモニタリングできます。AWS Glue ジョブから raw データが収集され、ほぼリアルタイムの読み取り可能なメトリクスに加工されて、Amazon CloudWatch に保存されます。これらの統計は CloudWatch に保持されて集計されるため、履歴情報にアクセスしてアプリケーションの動作をより的確に把握できます。

> **Note**
>
> ジョブメトリクスを有効にし、CloudWatch カスタムメトリクスが作成されると、追加料金が発生することがあります。詳細については、「Amazon CloudWatch の料金」を参照してください。

AWS Glue メトリクスの概要

AWS Glue を操作するときに、CloudWatch にメトリクスが送信されます。これらのメトリクスは、AWS Glue コンソール (推奨される方法)、CloudWatch コンソールダッシュボード、または AWS Command Line Interface (AWS CLI) で表示できます。

AWS Glue コンソールダッシュボードを使ってメトリクスを表示するには

ジョブのメトリクスの概要または詳細なグラフを表示したり、ジョブの実行の詳細なグラフを表示したりできます。

1. AWS Management Consoleにサインインし、AWS Glue コンソール (https://console.aws.amazon.com/glue/) を開きます。
2. ナビゲーションペインで、[ジョブ実行モニタリング] を選択します。
3. [Job の実行] で [アクション] を選択して、現在実行中のジョブの停止、ジョブの表示、またはジョブのブックマークの巻き戻しを実行します。
4. ジョブを選択し、[実行の詳細を表示する] を選択すると、そのジョブ実行に関する追加情報が表示されます。

CloudWatch コンソールダッシュボードを使用してメトリクスを表示するには

メトリクスはまずサービスの名前空間ごとにグループ化され、次に各名前空間内のさまざまなディメンションの組み合わせごとにグループ化されます。

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで [Metrics (メトリクス)] を選択します。
3. [Glue] 名前空間を選択します。

AWS CLI を使ってメトリクスを表示するには

- コマンドプロンプトで、次のコマンドを使用します。

```
aws cloudwatch list-metrics --namespace Glue
```

メトリクスは 30 秒ごとに AWS Glue から CloudWatch に報告され、CloudWatch メトリクスダッシュボードは 1 分ごとにそれらを表示するように設定されています。AWS Glue メトリクスは、以

前に報告された値からデルタ値を表示します。必要に応じて、メトリクスのダッシュボードにより 30 秒の値が集計 (合計) され、直近 1 分間分の値を取得できます。

Spark ジョブの AWS Glue メトリクス動作

AWS Glue メトリクスは、スクリプトでの `GlueContext` の初期化時に有効になり、基本的には Apache Spark タスクの終了時にのみ更新されます。また、これまでに完了した Spark タスク全体の集計値を表します。

ただし、AWS Glue が CloudWatch に渡す Spark メトリクスは、一般的に報告された時点で現在の状態を表す絶対値です。それらは 30 秒ごとに AWS Glue から CloudWatch に報告され、一般的にメトリクスダッシュボードには直近 1 分間に受け取ったデータポイントの平均が表示されます。

すべての AWS Glue メトリクス名の前には、必ず次のいずれかの種類のプレフィックスが付きます。

- `glue.driver.` – 名前がこのプレフィックスで始まるメトリクスは、Spark ドライバーのすべてのエグゼキュターから集計された AWS Glue メトリクスか、Spark ドライバーに対応する Spark メトリクスを表します。
- `.executorIdglue.` – executorId は、特定の Spark エグゼキュターの番号です。ログに表示されているエグゼキュータに対応します。
- `glue.ALL.` – 名前がこのプレフィックスで始まるメトリクスは、すべての Spark エグゼキュターからの値を集計します。

AWS Glue のメトリクス

AWS Glue によって、プロファイルが作成され、次のメトリクスが 30 秒ごとに CloudWatch に送信され、AWS Glue メトリクスダッシュボードに 1 分に 1 回表示されます。

メトリクス	説明
`glue.driver.aggregate.bytesRead`	すべてのエグゼキュターで実行され完了したすべての Spark タスクによるすべてのデータソースから読み取られたバイト数。 有効なディメンション: `JobName` (AWS Glue Job の名前)、`JobRunId` (JobRun ID. または `ALL`)、`Type` (カウント)。

メトリクス	説明
	有効な統計: SUM。このメトリクスは、最後に報告された値からのデルタ値であるため、AWS Glue メトリクスダッシュボードでは、集計に SUM 統計が使用されます。 単位: バイト 次のモニタリングに使用できます。 • 読み込みバイト。 • ジョブ進行状況。 • JDBC データソース。 • ジョブブックマークの問題。 • ジョブ実行間の差異。 このメトリクスは、`glue.ALL.s3.filesystem.read_bytes` メトリクスと同様に使用できます。ただし、このメトリクスは Spark タスクの最後に更新され、S3 以外のデータソースも取得するという点が異なります。

メトリクス	説明
glue.driver.aggregate.elapsedTime	ETL の経過時間 (ミリ秒単位) (ジョブのブートストラップ時間は含まれません)。 有効なディメンション: JobName (AWS Glue Job の名前)、JobRunId (JobRun ID. または ALL)、Type (カウント)。 有効な統計: SUM。このメトリクスは、最後に報告された値からのデルタ値であるため、AWS Glue メトリクスダッシュボードでは、集計に SUM 統計が使用されます。 単位: ミリ秒 ジョブの実行に平均どのくらいの時間がかかるかを調べるために利用できます。 データの使用法をいくつか次に示します。 • ストラグラーにアラームを設定する。 • ジョブ実行間の差異を測定する。

メトリクス	説明
glue.driver.aggregate.numCompletedStages	ジョブの完了したステージの数 有効なディメンション: JobName (AWS Glue Job の名前)、JobRunId (JobRun ID. または ALL)、Type (カウント)。 有効な統計: SUM。このメトリクスは、最後に報告された値からのデルタ値であるため、AWS Glue メトリクスダッシュボードでは、集計に SUM 統計が使用されます。 単位: 個 次のモニタリングに使用できます。 • ジョブ進行状況。 • 他のメトリクスとの関連におけるジョブ実行のステージごとのタイムライン。 データの使用法をいくつか次に示します。 • ジョブの実行における要求の厳しい段階を特定する。 • ジョブ実行全体の中で関連するスパイク (要求の厳しいステージ) にアラームを設定する。

メトリクス	説明
glue.driver.aggregate.numCompletedTasks	ジョブで完了したタスクの数。 有効なディメンション: JobName (AWS Glue Job の名前)、JobRunId (JobRun ID. または ALL)、Type (カウント)。 有効な統計: SUM。このメトリクスは、最後に報告された値からのデルタ値であるため、AWS Glue メトリクスダッシュボードでは、集計に SUM 統計が使用されます。 単位: 個 次のモニタリングに使用できます。 • ジョブ進行状況。 • ステージ内の並列性。

メトリクス	説明
glue.driver.aggregate.numFailedTasks	失敗したタスクの数。 有効なディメンション: JobName (AWS Glue Job の名前)、JobRunId (JobRun ID. または ALL)、Type (カウント)。 有効な統計: SUM。このメトリクスは、最後に報告された値からのデルタ値であるため、AWS Glue メトリクスダッシュボードでは、集計に SUM 統計が使用されます。 単位: 個 次のモニタリングに使用できます。 • ジョブのタスクが失敗する原因となるデータ異常。 • ジョブのタスクが失敗する原因となるクラスター異常。 • ジョブのタスクが失敗する原因となるスクリプト異常。 データを使用して、データ、クラスター、スクリプトの異常を示唆する可能性のある障害の増加に対するアラームを設定できます。

メトリクス	説明
glue.driver.aggregate.numKilledTasks	強制終了したタスクの数。 有効なディメンション: JobName (AWS Glue Job の名前)、JobRunId (JobRun ID. または ALL)、Type (カウント)。 有効な統計: SUM。このメトリクスは、最後に報告された値からのデルタ値であるため、AWS Glue メトリクスダッシュボードでは、集計に SUM 統計が使用されます。 単位: 個 次のモニタリングに使用できます。 • タスクが強制終了される例外 (OOM) が発生するデータスキュー異常。 • タスクが強制終了される例外 (OOM) が発生するスクリプト異常。 データの使用法をいくつか次に示します。 • データ異常を示す障害が増加した場合のアラームを設定する。 • クラスター異常を示す障害が増加した場合のアラームを設定する。 • スクリプト異常を示す障害が増加した場合のアラームを設定する。

メトリクス	説明
glue.driver.aggregate.recordsRead	すべてのエグゼキューターで実行され完了したすべての Spark タスクによるすべてのデータソースから読み取られたレコードの数。 有効なディメンション: JobName (AWS Glue Job の名前)、JobRunId (JobRun ID. または ALL)、Type (カウント)。 有効な統計: SUM。このメトリクスは、最後に報告された値からのデルタ値であるため、AWS Glue メトリクスダッシュボードでは、集計に SUM 統計が使用されます。 単位: 個 次のモニタリングに使用できます。 • 読み込まれたレコード。 • ジョブ進行状況。 • JDBC データソース。 • ジョブブックマークの問題。 • ジョブ実行の日々のスキュー。 このメトリクスは、glue.ALL.s3.filesystem.read_bytes メトリクスと同様に使用できますが、このメトリクスは Spark タスクの最後に更新される点が異なります。

メトリクス	説明
glue.driver.aggregate.shuffleBytesWritten	前のレポート以降にデータをシャッフルするためにそれらのエグゼキューターにより書き込まれたバイト数 (直前の 1 分間にこの目的のために書き込まれたバイト数として AWS Glue メトリクスダッシュボードにより集計)。 有効なディメンション: JobName (AWS Glue Job の名前)、JobRunId (JobRun ID. または ALL)、Type (カウント)。 有効な統計: SUM。このメトリクスは、最後に報告された値からのデルタ値であるため、AWS Glue メトリクスダッシュボードでは、集計に SUM 統計が使用されます。 単位: バイト ジョブのデータシャッフル (大規模な結合、グループ化、再分割、合体) をモニタリングするために使用できます。 データの使用法をいくつか次に示します。 • さらに処理する前に、大きな入力ファイルを再分割または解凍します。 • ホットキーを避けるために、データをより均一に再分割します。 • 結合またはグループ化オペレーションの前にデータを事前にフィルタリングします。

メトリクス	説明
glue.driver.aggregate.shuffleLocalBytesRead	前のレポート以降にデータをシャッフルするためにそれらのエグゼキュターにより読み取られたバイト数 (直前の 1 分間にこの目的のために読み取られたバイト数として AWS Glue メトリクスダッシュボードにより集計)。 有効なディメンション: JobName (AWS Glue Job の名前)、JobRunId (JobRun ID. または ALL)、Type (カウント)。 有効な統計: SUM。このメトリクスは、最後に報告された値からのデルタ値であるため、AWS Glue メトリクスダッシュボードでは、集計に SUM 統計が使用されます。 単位: バイト ジョブのデータシャッフル (大規模な結合、グループ化、再分割、合体) をモニタリングするために使用できます。 データの使用法をいくつか次に示します。 • さらに処理する前に、大きな入力ファイルを再分割または解凍します。 • ホットキーを使用して、データをより均一に再分割します。 • 結合またはグループ化オペレーションの前にデータを事前にフィルタリングします。

メトリクス	説明
glue.driver.BlockManager.disk.diskSpaceUsed_MB	すべてのエグゼキューターで使用されたディスク領域のメガバイト数。 有効なディメンション: JobName (AWS Glue Job の名前)、JobRunId (JobRun ID. または ALL)、Type (ゲージ)。 有効な統計: Average これは Spark メトリクスで、絶対値としてレポートされます。 単位: メガバイト 次のモニタリングに使用できます。 • キャッシュされた RDD パーティションを表すブロックに使用されるディスク領域。 • 中間シャッフル出力を表すブロックに使用されるディスク領域。 • ブロードキャストを表すブロックに使用されるディスク領域。 データの使用法をいくつか次に示します。 • ディスク使用量の増加によるジョブの障害を特定する。 • スピルやシャッフルを引き起こす大きなパーティションを特定する。 • これらの問題を解決するためにプロビジョニングされた DPU 容量を増やす。

メトリクス	説明
glue.driver.ExecutorAllocationManager.executors.numberAllExecutors	アクティブに実行されているジョブエグゼキューターの数。 有効なディメンション: JobName (AWS Glue Job の名前)、JobRunId (JobRun ID. または ALL)、Type (ゲージ)。 有効な統計: Average これは Spark メトリクスで、絶対値としてレポートされます。 単位: 個 次のモニタリングに使用できます。 • ジョブアクティビティ。 • 高負荷エグゼキューター (少数のエグゼキューターのみが実行されている) • 現在のエグゼキューターレベルの並列処理。 データの使用法をいくつか次に示します。 • クラスターの使用率が低い場合に、事前に大きな入力ファイルを再分割または解凍する。 • 一部が高負荷であるために起こるステージまたはジョブ実行の遅延を特定する。 • • numberMaxNeededExecutors と比較して、より多くの DPU をプロビジョニングするためにバックログを理解する。

メトリクス	説明
glue.driver.ExecutorAllocationManager.executors.numberMaxNeededExecutors	現在の負荷を満たすために必要な (アクティブに実行中および保留中の) ジョブエグゼキューターの最大数。 有効なディメンション: JobName (AWS Glue Job の名前)、JobRunId (JobRun ID. または ALL)、Type (ゲージ)。 有効な統計: Maximum。これは Spark メトリクスで、絶対値としてレポートされます。 単位: 個 次のモニタリングに使用できます。 • ジョブアクティビティ。 • DPU の容量または強制終了/失敗したエグゼキューターのために使用できないエグゼキューターが原因で、まだスケジュールされていない現在のエグゼキューターレベルの並列処理と保留タスクのバックログ。 データの使用法をいくつか次に示します。 • スケジューリングキューの保留中/バックログを特定する。 • 一部が高負荷であるために起こるステージまたはジョブ実行の遅延を特定する。 • numberAllExecutors と比較して、より多くの DPU をプロビジョニングするためにバックログを理解する。 • プロビジョニングされた DPU 容量を増やして、保留中のエグゼキューターのバックログを修正する。

メトリクス	説明
glue.driver.jvm.heap.usage glue.executorId.jvm.heap.usage glue.ALL.jvm.heap.usage	ドライバー、executorId により識別されるエグゼキュター、またはすべてのエグゼキューターに対する、このドライバー (スケール: 0 ~ 1) の JVM ヒープにより使用されるメモリの割合。 有効なディメンション: JobName (AWS Glue Job の名前)、JobRunId (JobRun ID. または ALL)、Type (ゲージ)。 有効な統計: Average これは Spark メトリクスで、絶対値としてレポートされます。 単位: パーセント 次のモニタリングに使用できます。 • glue.driver.jvm.heap.usage 使用時のドライバーのメモリ不足状態 (OOM)。 • glue.ALL.jvm.heap.usage 使用時のエグゼキューターのメモリ不足状態 (OOM)。 データの使用法をいくつか次に示します。 • メモリを消費するエグゼキューター ID とステージを特定する。 • 高負荷のエグゼキューター ID とステージを特定する。 • ドライバーのメモリ不足状態 (OOM) を特定する。 • エグゼキューターのメモリ不足状態 (OOM) を特定し、対応するエグゼキューター ID を取得して、エグゼキューターログからスタックトレースを取得できるようにする。 • 高負荷またはメモリ不足状態 (OOM) の原因となるデータのスキューがある可能性があるファイルまたはパーティションを特定します。

メトリクス	説明
glue.driver.jvm.heap.used glue.executorId.jvm.heap.used glue.ALL.jvm.heap.used	ドライバー、executorId によって特定されたエグゼキュター、またはすべてのエグゼキューターのために JVM ヒープが使用したメモリバイト数。 有効なディメンション: JobName (AWS Glue Job の名前)、JobRunId (JobRun ID. または ALL)、Type (ゲージ)。 有効な統計: Average これは Spark メトリクスで、絶対値としてレポートされます。 単位: バイト 次のモニタリングに使用できます。 • ドライバーのメモリ不足状態 (OOM)。 • エグゼキュターのメモリ不足状態 (OOM)。 データの使用法をいくつか次に示します。 • メモリを消費するエグゼキュター ID とステージを特定する。 • 高負荷のエグゼキュター ID とステージを特定する。 • ドライバーのメモリ不足状態 (OOM) を特定する。 • エグゼキュターのメモリ不足状態 (OOM) を特定し、対応するエグゼキュター ID を取得して、エグゼキュターログからスタックトレースを取得できるようにする。 • 高負荷またはメモリ不足状態 (OOM) の原因となるデータのスキューがある可能性があるファイルまたはパーティションを特定します。

メトリクス	説明
glue.driver.s3.filesystem.read_bytes glue.executorId.s3.filesystem.read_bytes glue.ALL.s3.filesystem.read_bytes	前のレポート以降に、ドライバー、executorId によって特定されるエグゼキュター、またはすべてのエグゼキュターが Amazon S3 から読み取ったバイト数 (直前 1 分間に読み取ったバイト数として AWS Glue メトリクスダッシュボードに集計されます)。 有効なディメンション: JobName、JobRunId、および Type (ゲージ)。 有効な統計: SUM。このメトリクスは、最後に報告された値からのデルタ値であるため、AWS Glue メトリクスダッシュボードでは、集計に SUM 統計が使用されます。AWS Glue メトリクスダッシュボードのカーブの下にある領域は、2 つの異なるジョブ実行によって読み取られたバイト数を視覚的に比較するために使用できます。 単位: バイト 次のモニタリングに使用できます。 • ETL データ移動。 • ジョブ進行状況。 • ジョブのブックマークの問題 (処理されたデータ、再処理されたデータ、およびスキップされたデータ) • 読み取りと外部データソースからの取り込みとの比率の比較。 • ジョブ実行間の差異。 結果データは、次の目的で使用できます。 • DPU 容量の計画。

メトリクス	説明
	• ジョブ実行およびジョブステージのデータ読み取りの急激な増加または落ち込みに対するアラームの設定

メトリクス	説明
glue.driver.s3.filesystem.write_bytes glue.executorId.s3.filesystem.write_bytes glue.ALL.s3.filesystem.write_bytes	前のレポート以降に、ドライバー、executorId によって特定されるエグゼキュター、またはすべてのエグゼキュターが Amazon S3 に書き込んだバイト数 (直前 1 分間に書き込んだバイト数として AWS Glue メトリクスダッシュボードに集計されます)。 有効なディメンション: JobName、JobRunId、および Type (ゲージ)。 有効な統計: SUM。このメトリクスは、最後に報告された値からのデルタ値であるため、AWS Glue メトリクスダッシュボードでは、集計に SUM 統計が使用されます。AWS Glue メトリクスダッシュボードのカーブの下にある領域は、2 つの異なるジョブ実行によって書き込まれたバイト数を視覚的に比較するために使用できます。 単位: バイト 次のモニタリングに使用できます。 • ETL データ移動。 • ジョブ進行状況。 • ジョブのブックマークの問題 (処理されたデータ、再処理されたデータ、およびスキップされたデータ) • 読み取りと外部データソースからの取り込みとの比率の比較。 • ジョブ実行間の差異。 データの使用法をいくつか次に示します。 • DPU 容量の計画。

メトリクス	説明
	• ジョブ実行およびジョブステージのデータ読み取りの急激な増加または落ち込みに対するアラームの設定
glue.driver.streaming.numRecords	マイクロバッチで受信されたレコードの数。このメトリクスは AWS Glue バージョン 2.0 以降の AWS Glue ストリーミングジョブにのみ使用できます。です。 有効なディメンション: JobName (AWS Glue Job の名前)、JobRunId (JobRun ID. または ALL)、Type (カウント)。 有効な統計: Sum, Maximum, Minimum, Average, Percentile 単位: 個 次のモニタリングに使用できます。 • 読み込まれたレコード。 • ジョブ進行状況。

メトリクス	説明
glue.driver.streaming.batchProcessingTimeInMs	バッチの処理に要する時間 (ミリ秒単位)。このメトリクスは AWS Glue バージョン 2.0 以降の AWS Glue ストリーミングジョブにのみ使用できます。です。 有効なディメンション: JobName (AWS Glue Job の名前)、JobRunId (JobRun ID. または ALL)、Type (カウント)。 有効な統計: Sum, Maximum, Minimum, Average, Percentile 単位: 個 次のモニタリングに使用できます。 • ジョブ進行状況。 • スクリプトのパフォーマンス。

メトリクス	説明
glue.driver.system.cpuSystemLoad glue.executorId.system.cpuSystemLoad glue.ALL.system.cpuSystemLoad	ドライバー、executorId により識別されるエグゼキュター、またはすべてのエグゼキュターに対する使用された CPU システムロードの割合 (スケール: 0 ~ 1)。 有効なディメンション: JobName (AWS Glue Job の名前)、JobRunId (JobRun ID. または ALL)、Type (ゲージ)。 有効な統計: Average このメトリクスは絶対値としてレポートされます。 単位: パーセント 次のモニタリングに使用できます。 • ドライバー CPU 負荷。 • エグゼキュター CPU 負荷。 • ジョブ内の CPU 制約または IO 制約のエグゼキュターまたはステージの検出。 データの使用法をいくつか次に示します。 • IO メトリクス (読み取りバイト数/シャッフルバイト数、タスクの並列処理) および必要な最大エグゼキュター数のメトリクスとともに、DPU 容量を計画する。 • CPU/IO 制約比を確認する。これにより、CPU 使用率が低い分割可能なデータセットを使用して、長時間実行されるジョブに対して、再分割とプロビジョニングされた容量の増加が可能になります。

AWS Glue メトリクスのディメンション

AWS Glue メトリクスは AWS Glue 名前空間を使用し、以下のディメンションのメトリクスを提供しています。

ディメンション	説明
JobName	このディメンションで、特定の AWS Glue ジョブのすべてのジョブ実行のメトリクスがフィルタリングされます。
JobRunId	このディメンションで、JobRun ID によって特定の AWS Glue ジョブ実行のメトリクスがフィルタリングされるか、ALL となります。
Type	このディメンションで、count (集計した数) または gauge (ある時点での値) によってメトリクスがフィルタリングされます。

詳細については、『Amazon CloudWatch ユーザーガイド』を参照してください。

AWS Glue ジョブプロファイルでの Amazon CloudWatch アラームの設定

AWS Glue メトリクスは、Amazon CloudWatch でも利用できます。スケジュールされたジョブのどの AWS Glue メトリクスでもアラームを設定できます。

アラームを設定するいくつかの一般的なシナリオは次のとおりです。

- ジョブのメモリ不足 (OOM): メモリ使用率が AWS Glue ジョブのドライバーまたはエグゼキューターの通常平均を超えた場合にアラームを設定します。
- 散在したエグゼキューター: AWS Glue ジョブでエグゼキューターの数が長時間一定のしきい値を下回ったときにアラームを設定します。
- データバックログまたは再処理: CloudWatch の数式を使用して、ワークフローの個別のジョブから得られたメトリクスを比較します。その後、生成された式の値 (あるジョブにより書き込まれたバイトと後続のジョブにより読み込まれたバイトの割合など) でアラームをトリガーできます。

アラームの設定に関する詳細については、Amazon CloudWatch Events ユーザーガイドの「Create or Edit a CloudWatch Alarm」を参照してください。

CloudWatch を使用したモニタリングとデバッグのシナリオについては、「ジョブのモニタリングとデバッグ」を参照してください。

AWS Glue ジョブのログ記録

AWS Glue 5.0 では、すべてのジョブにリアルタイムのログ記録機能があります。さらに、カスタム設定オプションを指定してログ記録の動作を調整できます。これらのオプションには、Amazon CloudWatch ロググループ名、Amazon CloudWatch ログストリームプレフィックス (AWS Glue ジョブ実行 ID とドライバー/エグゼキューター ID の前に付けられる)、ログメッセージのログ変換パターンの設定が含まれます。これらの設定により、異なる有効期限ポリシーを持つカスタム Amazon CloudWatch ロググループにログを集約できます。さらに、カスタムログストリームのプレフィックスおよび変換パターンを使用することで、ログをより効果的に分析できます。このレベルのカスタマイズによって、特定の要件に応じてログ管理と分析を最適化できます。

AWS Glue 5.0 でのログ記録動作

デフォルトでは、システムログ、Spark デーモンログ、ユーザー AWS Glue ロガーログは、Amazon CloudWatch の `/aws-glue/jobs/error` ロググループに書き込まれます。一方、ユーザー stdout (標準出力) ログと stderr (標準エラー) ログは、デフォルトで `/aws-glue/jobs/output` ロググループに書き込まれます。

カスタムログ記録

次のジョブ引数を使用すると、デフォルトのロググループとログストリームプレフィックスをカスタマイズできます。

- `--custom-logGroup-prefix`: `/aws-glue/jobs/error` および `/aws-glue/jobs/output` ロググループのカスタムプレフィックスを指定できます。カスタムプレフィックスを指定すれば、ロググループ名は次の形式になります。
  - `/aws-glue/jobs/error` は *`<customer prefix>`*`/error` になります。
  - `/aws-glue/jobs/output` は *`<customer prefix>`*`/output` になります。
- `--custom-logStream-prefix`: ロググループ内のログストリーム名にカスタムプレフィックスを指定できます。カスタムプレフィックスを指定すれば、ログストリーム名は次の形式になります。
  - `jobrunid-driver` は *`<customer log stream>`*`-driver` になります。
  - `jobrunid-executorNum` は *`<customer log stream>`*`-executorNum` になります。

カスタムプレフィックスの検証ルールと制限は次のとおりです。

- ログストリーム名全体は、1～512 文字である必要があります。
- カスタムプレフィックス自体は 400 文字に制限されます。
- カスタムプレフィックスは正規表現パターン `[^:*]*` と一致する必要があります (使用できる特殊文字は '_'、'-'、'/' です)。

カスタムスクリプトロガーを使用したアプリケーション固有のメッセージのログ記録

AWS Glue ロガーを使用すると、ドライバーログストリームにリアルタイムで送信される、スクリプトのアプリケーション固有のメッセージを記録できます。

Python スクリプトの例を以下に示します。

```
from awsglue.context import GlueContext
from pyspark.context import SparkContext

sc = SparkContext()
glueContext = GlueContext(sc)
logger = glueContext.get_logger()
logger.info("info message")
logger.warn("warn message")
logger.error("error message")
```

Scala スクリプトの例を以下に示します。

```
import com.amazonaws.services.glue.log.GlueLogger

object GlueApp {
  def main(sysArgs: Array[String]) {
    val logger = new GlueLogger
    logger.info("info message")
    logger.warn("warn message")
    logger.error("error message")
  }
}
```

進行状況バーでのジョブ進行状況の表示

AWS Glue は、`JOB_RUN_ID-progress-bar` ログストリームでリアルタイムの進行状況バーを提供して AWS Glue ジョブ実行のステータスを確認します。現時点では、`glueContext` を初期化す

るジョブのみがサポートされています。glueContext を初期化せずに純粋な Spark ジョブを実行すると、AWS Glue 進行状況バーは表示されません。

進行状況バーでは、以下の進行状況が 5 秒ごとに更新されます。

```
Stage Number (Stage Name): > (numCompletedTasks + numActiveTasks) /
 totalNumOfTasksInThisStage]
```

Amazon CloudWatch ログ記録によるセキュリティ設定

Amazon CloudWatch ログのセキュリティ設定を有効にすると、AWS Glue によってセキュリティ設定名が組み込まれた特定の命名パターンを持つロググループが作成されます。

セキュリティ設定を使用したロググループの命名

デフォルトのロググループおよびカスタムロググループは次のようになります。

- デフォルトのエラーロググループ: /aws-glue/jobs/*Security-Configuration-Name*-role/*glue-job-role*/error
- デフォルトの出力ロググループ: /aws-glue/jobs/*Security-Configuration-Name*-role/*glue-job-role*/output
- カスタムエラーロググループ (AWS Glue 5.0): *custom-log-group-prefix*/*Security-Configuration-Name*-role/*glue-job-role*/error
- カスタム出力ロググループ (AWS Glue 5.0): *custom-log-group-prefix*/*Security-Configuration-Name*-role/*glue-job-role*/output

必要な IAM アクセス許可

Amazon CloudWatch Logs でセキュリティ設定を有効にする場合、logs:AssociateKmsKey 許可を IAM ロールのアクセス許可に追加する必要があります。そのアクセス許可が含まれていない場合、連続ログ記録は無効になります。

また、Amazon CloudWatch Logs 暗号化を設定するには、「Amazon Amazon CloudWatch Logs ユーザーガイド」の「Encrypt Log Data in Amazon CloudWatch Logs Using AWS Key Management Service」を参照してください。

追加情報

セキュリティ設定の作成の詳細については、「AWS Glue コンソールでのセキュリティ設定の管理」を参照してください。

トピック

- AWS Glue 4.0 以前のジョブの継続的なログ記録の有効化
- AWS Glue ジョブのログの確認

AWS Glue 4.0 以前のジョブの継続的なログ記録の有効化

> **Note**
>
> 4.0 AWS Glue 以前のバージョンでは、継続的なログ記録の機能は利用可能でした。しかし、AWS Glue 5.0 の導入により、すべてのジョブでリアルタイムのログ記録機能を利用できるようになりました。AWS Glue 5.0 のログ記録機能と設定オプションの詳細については、「AWS Glue ジョブのログ記録」を参照してください。

AWS Glue コンソールを使用するか、AWS Command Line Interface (AWS CLI) を通じて、連続ログ記録を有効にできます。

新しいジョブを作成するか、既存のジョブを編集または AWS CLI を介して有効にするときに、連続ログ記録を有効にできます。

また、Amazon CloudWatch ロググループ名、AWS Glue ジョブ実行 ID ドライバー/エグゼキューター ID の前の CloudWatch ログストリームプレフィックス、ログメッセージのログ変換パターンなどのカスタム設定オプションを指定することもできます。これらの設定は、異なる有効期限ポリシーを持つカスタム CloudWatch ロググループに集約ログを設定し、カスタムログストリームプレフィックスと変換パターンを使用してさらに分析するのに役立ちます。

トピック

- AWS Management Console を使用する場合
- カスタムスクリプトロガーを使用したアプリケーション固有のメッセージのログ記録
- 進行状況バーでのジョブ進行状況の表示
- 連続ログ記録によるセキュリティ設定。

AWS Management Console を使用する場合

次のステップに従ってコンソールを使用して、AWS Glue ジョブを作成または編集するときに連続ログ記録を有効にします。

連続ログ記録を有効にして新しい AWS Glue ジョブを作成するには

1. AWS Management Consoleにサインインし、AWS Glue コンソール (https://console.aws.amazon.com/glue/) を開きます。
2. ナビゲーションペインで [ETL ジョブ] を選択します。
3. [ビジュアル ETL] を選択します。
4. [Job 詳細] タブで、[詳細プロパティ] セクションを展開します。
5. [継続的なログ記録] で [CloudWatch でログの有効化] を選択します。

既存の AWS Glue ジョブに対して連続ログ記録を有効にするには

1. https://console.aws.amazon.com/glue/ で AWS Glue コンソール を開きます。
2. ナビゲーションペインで ジョブを選択します。
3. [Jobs (ジョブ)] リストから既存のジョブを選択します。
4. [Action (アクション)]、[Edit job (ジョブの編集)] の順に選択します。
5. [Job 詳細] タブで、[詳細プロパティ] セクションを展開します。
6. [継続的なログ記録] で [CloudWatch でログの有効化] を選択します。

AWS CLI の使用

連続ログ記録を有効にするには、 AWS Glue ジョブにジョブパラメータを渡します。他の AWS Glue ジョブパラメータに類似した次の特殊なジョブパラメータを渡します。詳細については、「AWS Glue ジョブでジョブパラメータを使用する」を参照してください。

```
'--enable-continuous-cloudwatch-log': 'true'
```

カスタム Amazon CloudWatch ロググループ名を指定できます。指定しない場合、デフォルトのロググループ名は、`/aws-glue/jobs/logs-v2` になります。

```
'--continuous-log-logGroup': 'custom_log_group_name'
```

カスタム Amazon CloudWatch ログストリームプレフィックスを指定できます。指定しない場合、デフォルトのログストリームプレフィックスはジョブ実行 ID になります。

```
'--continuous-log-logStreamPrefix': 'custom_log_stream_prefix'
```

カスタムの連続ロギング変換パターンを指定できます。指定しない場合、デフォルトの変換パターンは %d{yy/MM/dd HH:mm:ss} %p %c{1}: %m%n になります。変換パターンは、ドライバーログとエグゼキューターログにのみ適用されます。AWS Glue の進行状況バーには影響しません。

```
'--continuous-log-conversionPattern': 'custom_log_conversion_pattern'
```

カスタムスクリプトロガーを使用したアプリケーション固有のメッセージのログ記録

AWS Glue ロガーを使用すると、ドライバーログストリームにリアルタイムで送信される、スクリプトのアプリケーション固有のメッセージを記録できます。

Python スクリプトの例を以下に示します。

```
from awsglue.context import GlueContext
from pyspark.context import SparkContext

sc = SparkContext()
glueContext = GlueContext(sc)
logger = glueContext.get_logger()
logger.info("info message")
logger.warn("warn message")
logger.error("error message")
```

Scala スクリプトの例を以下に示します。

```
import com.amazonaws.services.glue.log.GlueLogger

object GlueApp {
  def main(sysArgs: Array[String]) {
    val logger = new GlueLogger
    logger.info("info message")
    logger.warn("warn message")
    logger.error("error message")
  }
}
```

進行状況バーでのジョブ進行状況の表示

AWS Glue は、JOB_RUN_ID-progress-bar ログストリームでリアルタイムの進行状況バーを提供して AWS Glue ジョブ実行のステータスを確認します。現時点では、glueContext を初期化す

るジョブのみがサポートされています。glueContext を初期化せずに純粋な Spark ジョブを実行すると、AWS Glue 進行状況バーは表示されません。

進行状況バーでは、以下の進行状況が 5 秒ごとに更新されます。

```
Stage Number (Stage Name): > (numCompletedTasks + numActiveTasks) /
 totalNumOfTasksInThisStage]
```

連続ログ記録によるセキュリティ設定。

CloudWatch ログのセキュリティ設定が有効になっている場合、AWS Glue では、連続ログに対して次のような名前のロググループが作成されます。

```
<Log-Group-Name>-<Security-Configuration-Name>
```

デフォルトのロググループおよびカスタムロググループは次のようになります。

- デフォルトの連続ロググループは /aws-glue/jobs/error-<*Security-Configuration-Name*> となります。
- カスタム連続ロググループは <*custom-log-group-name*>-<*Security-Configuration-Name*> となります。

CloudWatch Logs でセキュリティ設定を有効にする場合、logs:AssociateKmsKey を IAM ロールのアクセス許可に追加する必要があります。そのアクセス許可が含まれていない場合、連続ログ記録は無効になります。また、CloudWatch Logs 暗号化を設定するには、Amazon CloudWatch Logs ユーザーガイドの「AWS Key Management Service を使用して CloudWatch Logs のログデータを暗号化する」を参照してください。

セキュリティ設定作成の詳細については、「AWS Glue コンソールでのセキュリティ設定の管理」を参照してください。

> **Note**
>
> ログを有効にし、追加の CloudWatch ログイベントが作成されると、追加料金が発生することがあります。詳細については、「Amazon CloudWatch の料金」を参照してください。

AWS Glue ジョブのログの確認

リアルタイムのログは、AWS Glue コンソールまたは Amazon CloudWatch コンソールを使用して表示できます。

AWS Glue コンソールダッシュボードを使用してリアルタイムのログを表示するには

1. AWS Management Consoleにサインインし、AWS Glue コンソール (https://console.aws.amazon.com/glue/) を開きます。
2. ナビゲーションペインで ジョブを選択します。
3. 既存のジョブを追加または開始します。[アクション]、[ジョブの実行] の順に選択します。

   ジョブの実行を開始するときは、実行中のジョブに関する情報が含まれているページに移動します。

   - [ログ] タブには、集約された古いアプリケーションログが表示されます。
   - [ログ] タブには、`glueContext` が初期化されたジョブが実行されているときのリアルタイムの進行状況バーが表示されます。
   - [ログ] タブには、[ライバーログ] も含まれています。このログには、リアルタイムの Apache Spark ドライバーログと、ジョブが実行中に AWS Glue アプリケーションを使用して記録されたスクリプトからのアプリケーションログがキャプチャされます。
4. 古いジョブでは、リアルタイムログを [Job History] (ジョブ履歴) ビューで [Logs] (ログ) を選択して表示することもできます。このアクションを実行すると、そのジョブ実行のすべての Spark ドライバー、エグゼキュター、および進捗状況バーのログストリームを表示する CloudWatch コンソールに移動します。

CloudWatch コンソールダッシュボードを使用してリアルタイムのログを表示するには

1. CloudWatch コンソールの https://console.aws.amazon.com/cloudwatch/ を開いてください。
2. ナビゲーションペインで [ログ] を選択します。
3. /aws-glue/jobs/error/ ロググループを選択します。
4. [フィルタ] ボックスに、ジョブ実行 ID を貼り付けます。

   ドライバーログ、エグゼキュターログ、および進行状況バーを表示できます ([標準フィルタ] を使用している場合)。

AWS Glue オブザーバビリティメトリクスを使用したモニタリング

> **Note**
>
> AWS Glue オブザーバビリティメトリクスは AWS Glue 4.0 以降のバージョンで使用できます。

AWS Glue オブザーバビリティメトリクスを使用して、AWS Glue for Apache Spark ジョブの内部で何が起こっているかに関するインサイトを生成し、問題の優先順位付けと分析を改善できます。オブザーバビリティメトリクスは Amazon CloudWatch ダッシュボードを通じて視覚化され、エラーの根本原因分析の実行に役立てたり、パフォーマンスのボトルネックを診断したりするために使用できます。問題のデバッグにかかる時間を大幅に削減できるため、より迅速かつ効果的に問題を解決することに注力できます。

AWS Glue オブザーバビリティは、次の 4 つのグループに分類された Amazon CloudWatch メトリクスを提供します。

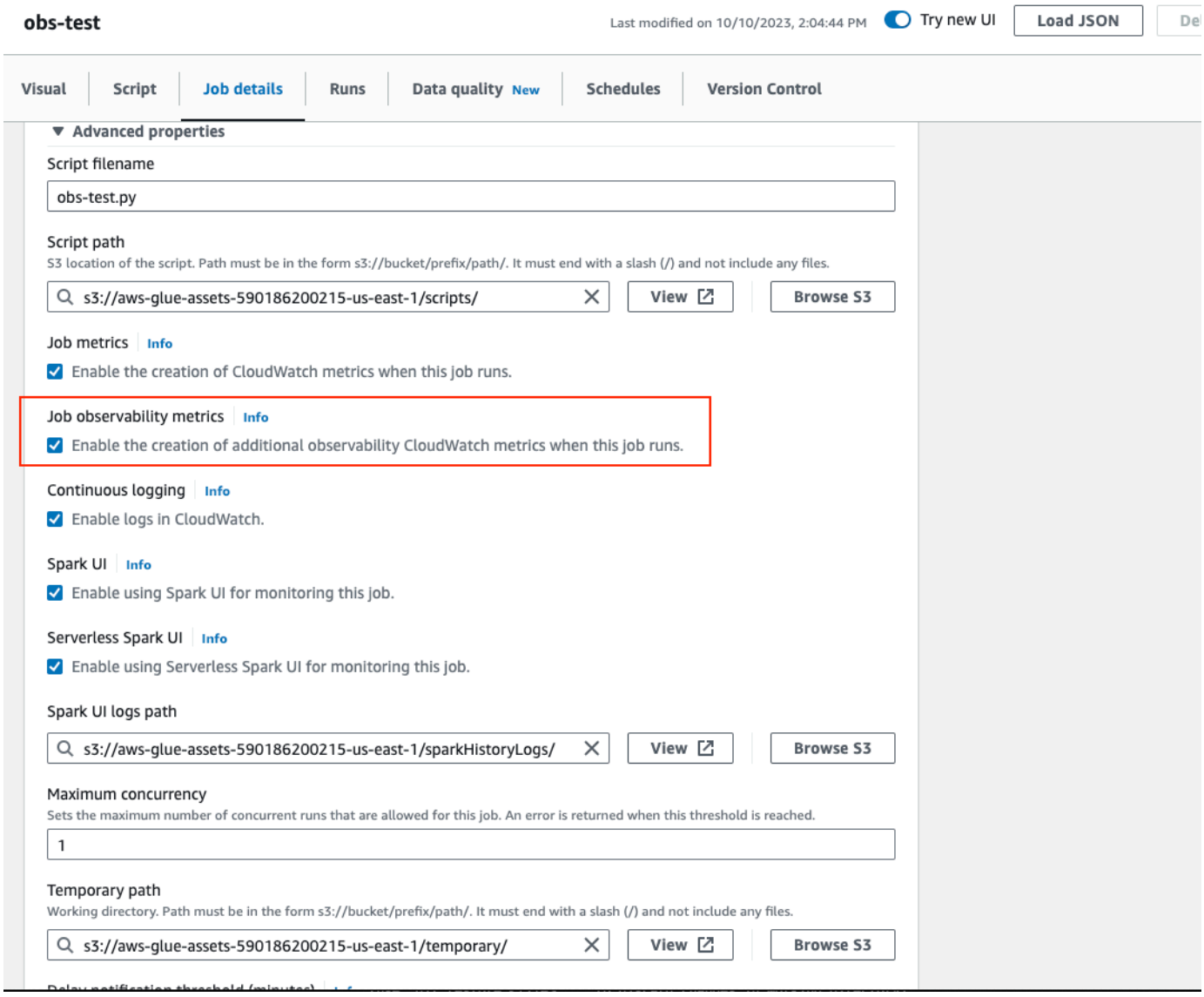
- [信頼性] (エラークラス) – 対処する特定の時間範囲における最も一般的な障害の理由を簡単に特定します。
- [パフォーマンス] (歪み) – パフォーマンスのボトルネックを特定し、チューニング手法を適用します。例えば、ジョブの歪みを理由としてパフォーマンスの低下が発生した場合、Spark Adaptive Query Execution を有効にして、歪み結合のしきい値を微調整することをお勧めします。
- [スループット] (ソース/シンクあたりのスループット) – データの読み取りと書き込みの傾向をモニタリングします。異常についての Amazon CloudWatch アラームを設定することもできます。
- [リソースの使用率] (ワーカー、メモリ、ディスクの使用率) – キャパシティの使用率が低いジョブを効率的に見つけます。これらのジョブについて AWS Glue 自動スケーリングを有効にすることをお勧めします。

AWS Glue オブザーバビリティメトリクスの開始方法

> **Note**
>
> 新しいメトリクスは、AWS Glue Studio コンソールでデフォルトで有効化されています。

AWS Glue Studio でオブザーバビリティメトリクスを設定するには:

1. AWS Glue コンソールにログインし、コンソールメニューから [ETL ジョブ] を選択します。
2. [自分のジョブ] セクションで、ジョブ名をクリックしてジョブを選択します。
3. [Job details] (ジョブの詳細) タブを選択します。
4. 一番下までスクロールして、[詳細プロパティ] を選択し、[ジョブのオブザーバビリティメトリクス] を選択します。



AWS CLI を使用して AWS Glue オブザーバビリティメトリクスを有効にするには

- 入力 JSON ファイル内の次の key-value を `--default-arguments` マップに追加します。

```
--enable-observability-metrics, true
```

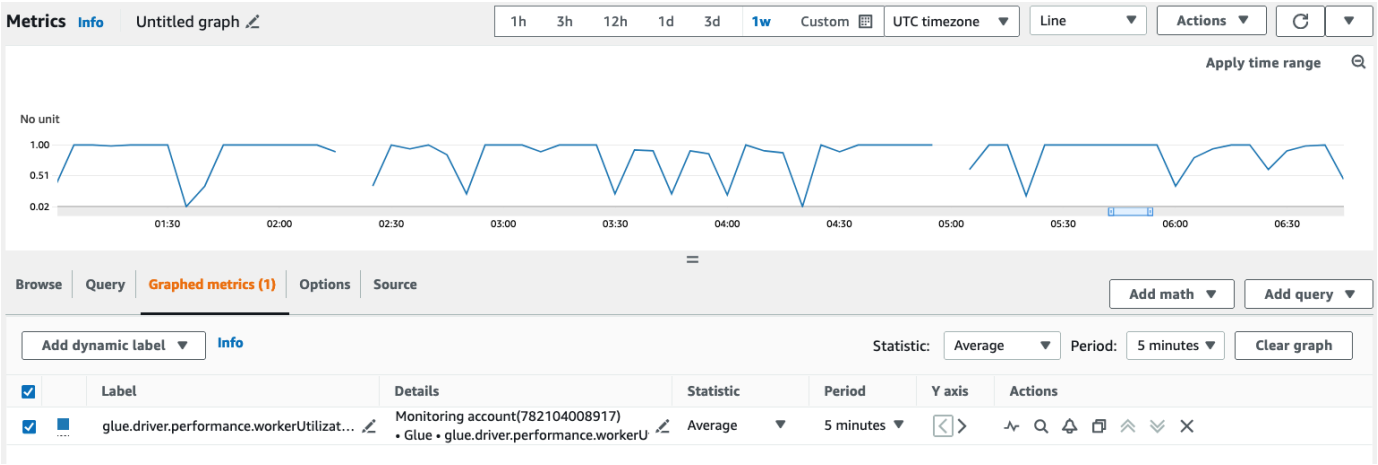
AWS Glue オブザーバビリティの使用

AWS Glue オブザーバビリティメトリクスは Amazon CloudWatch を通じて提供されるため、Amazon CloudWatch コンソール、AWS CLI、SDK、または API を使用してオブザーバビリティメトリクスデータポイントをクエリできます。AWS Glue オブザーバビリティメトリクスを使用する場合のユースケースの例については、「Using Glue Observability for monitoring resource utilization to reduce cost」を参照してください。

Amazon CloudWatch コンソールでの AWS Glue オブザーバビリティの使用

Amazon CloudWatch コンソールでメトリクスをクエリおよび視覚化するには:

1. Amazon CloudWatch コンソールを開き、[すべてのメトリクス] を選択します。
2. [カスタム名前空間] の下で、AWS Glue を選択します。
3. [ジョブのオブザーバビリティメトリクス]、[ソースごとのオブザーバビリティメトリクス]、または [シンクごとのオブザーバビリティメトリクス] を選択します。
4. 特定のメトリクス名、ジョブ名、ジョブ実行 ID を検索し、選択します。
5. [グラフ化されたメトリクス] タブで、任意の統計、期間、その他のオプションを設定します。



AWS CLI を使用してオブザーバビリティメトリクスをクエリするには:

1. メトリクス定義 JSON ファイルを作成し、`your-Glue-job-name` と `your-Glue-job-run-id` を実際のものに置き換えます。

```
$ cat multiplequeries.json
[
    {
        "Id": "avgWorkerUtil_0",
        "MetricStat": {
            "Metric": {
                "Namespace": "Glue",
                "MetricName": "glue.driver.workerUtilization",
                "Dimensions": [
                    {
                        "Name": "JobName",
                        "Value": "<your-Glue-job-name-A>"
                    },
                    {
                        "Name": "JobRunId",
                        "Value": "<your-Glue-job-run-id-A>"
                    },
                    {
                        "Name": "Type",
                        "Value": "gauge"
                    },
                    {
                        "Name": "ObservabilityGroup",
                        "Value": "resource_utilization"
                    }
                ]
            },
            "Period": 1800,
            "Stat": "Minimum",
            "Unit": "None"
        }
    },
    {
        "Id": "avgWorkerUtil_1",
        "MetricStat": {
            "Metric": {
                "Namespace": "Glue",
                "MetricName": "glue.driver.workerUtilization",
                "Dimensions": [
                    {
                        "Name": "JobName",
                        "Value": "<your-Glue-job-name-B>"
                    },
```

```
                    {
                        "Name": "JobRunId",
                        "Value": "<your-Glue-job-run-id-B>"
                    },
                    {
                        "Name": "Type",
                        "Value": "gauge"
                    },
                    {
                        "Name": "ObservabilityGroup",
                        "Value": "resource_utilization"
                    }
                ]
            },
            "Period": 1800,
            "Stat": "Minimum",
            "Unit": "None"
        }
    }
]
```

2. get-metric-data コマンドを実行します。

```
$ aws cloudwatch get-metric-data --metric-data-queries file: //multiplequeries.json
 \
     --start-time '2023-10-28T18: 20' \
     --end-time '2023-10-28T19: 10'  \
     --region us-east-1
{
    "MetricDataResults": [
        {
            "Id": "avgWorkerUtil_0",
            "Label": "<your-label-for-A>",
            "Timestamps": [
                "2023-10-28T18:20:00+00:00"
            ],
            "Values": [
                0.06718750000000001
            ],
            "StatusCode": "Complete"
        },
        {
```

```
            "Id": "avgWorkerUtil_1",
            "Label": "<your-label-for-B>",
            "Timestamps": [
                "2023-10-28T18:50:00+00:00"
            ],
            "Values": [
                0.5959183673469387
            ],
            "StatusCode": "Complete"
        }
    ],
    "Messages": []
}
```

オブザーバビリティメトリクス

AWS Glue オブザーバビリティは以下のメトリクスをプロファイリングし、30 秒ごとに Amazon CloudWatch に送信します。これらのメトリクスの一部は、AWS Glue Studio の「ジョブ実行モニタリング」ページで確認できます。

メトリクス	説明	カテゴリ
glue.driver.skewness.stage	メトリクスカテゴリ: job_performance Spark ステージの実行歪度: このメトリクスは、入力データの歪みや変換 (スキュー結合など) によって生じる可能性のある実行の歪みをキャプチャします。このメトリクスの値は [0, infinity] の範囲に入ります。ここで、0 は、ステージ内のすべてのタスクのうち、タスクの実行時間の最大値と中央値の比率が特定のステージ歪度係数未満であることを意味します。デフォルトのス	job_performance

メトリクス	説明	カテゴリ
	テージ歪度係数は `5`で、spark conf: spark.metrics.conf.driver.source.glue.jobPerformance.skewnessFactor で上書きされます ステージ歪度の値が 1 の場合、比率はステージ歪度係数の 2 倍になります。 ステージ歪度の値は、現在の歪度を反映して 30 秒ごとに更新されます。ステージ終了時の値は、最終ステージの歪度を反映しています。 有効なディメンション: JobName (AWS Glue ジョブの名前)、JobRunId (JobRun ID. または ALL)、Type (ゲージ)、ObservabilityGroup (job_performance) 有効な統計: Average、Maximum、Minimum、Percentile 単位: 数	

メトリクス	説明	カテゴリ
glue.driver.skewness.job	メトリクスカテゴリ: job_performance ジョブの歪度は、ジョブステージの歪度の加重平均です。加重平均では、実行に時間がかかるステージの重みが高くなります。これは、非常に歪んだステージが、実際には他のステージに比べて実行時間が非常に短い (したがって、その歪度はジョブパフォーマンス全体にとって重要ではなく、歪みの解消に努力する価値がない) というコーナーケースを回避するためです。 このメトリクスは各ステージの完了時に更新されるため、最後の値には実際の全体的なジョブの歪みが反映されます。 有効なディメンション: JobName (AWS Glue ジョブの名前)、JobRunId (JobRun ID. または ALL)、Type (ゲージ)、ObservabilityGroup (job_performance) 有効な統計: Average、Maximum、Minimum、Percentile 単位: 数	job_performance

メトリクス	説明	カテゴリ
glue.succeed.ALL	メトリクスカテゴリ: error 障害カテゴリの全体像を把握するための、正常なジョブ実行の総数 有効なディメンション: JobName (AWS Glue ジョブの名前)、JobRunId (JobRun ID. または ALL)、Type (カウント)、ObservabilityGroup (エラー) 有効な統計: SUM 単位: 数	エラー
glue.error.ALL	メトリクスカテゴリ: error 障害カテゴリの全体像を把握するための、ジョブ実行エラーの総数 有効なディメンション: JobName (AWS Glue ジョブの名前)、JobRunId (JobRun ID. または ALL)、Type (カウント)、ObservabilityGroup (エラー) 有効な統計: SUM 単位: 数	エラー

メトリクス	説明	カテゴリ
glue.error.[error category]	メトリクスカテゴリ: error これは実際には一連のメトリクスで、ジョブの実行が失敗した場合にのみ更新されます。エラー分類はトリアージとデバッグに役立ちます。ジョブ実行に失敗すると、失敗の原因となったエラーが分類され、対応するエラーカテゴリメトリクスが 1 に設定されます。これにより、時間の経過に伴う障害分析だけでなく、すべてのジョブエラー分析を実行して、最も一般的な障害カテゴリを特定して対処を開始できます。AWS Glue には、OUT_OF_MEMORY (ドライバとエグゼキュター)、PERMISSION、SYNTAX、THROTTLING など、28 のエラーカテゴリがあります。また、COMPILATION、LAUNCH、TIMEOUT といったエラーカテゴリもあります。 有効なディメンション: JobName (AWS Glue ジョブの名前)、JobRunId (JobRun ID. または ALL)、Type (カウント)、ObservabilityGroup (エラー) 有効な統計: SUM	エラー

メトリクス	説明	カテゴリ
	単位: 数	
glue.driver.workerUtilization	メトリクスカテゴリ: resource_utilization 割り当てられたワーカーのうち、実際に使用されているワーカーの割合。うまく機能しない場合は、自動スケーリングが役に立ちます。 有効なディメンション: JobName (AWS Glue ジョブの名前)、JobRunId (JobRun ID. または ALL)、Type (ゲージ)、ObservabilityGroup (resource_utilization) 有効な統計: Average、Maximum、Minimum、Percentile 単位: パーセント	resource_utilization

メトリクス	説明	カテゴリ
glue.driver.memory.heap.[available | used]	メトリクスカテゴリ: resource_utilization ジョブ実行中のドライバの使用可能/使用済みヒープメモリ。これは、特に時間の経過に伴うメモリ使用量の傾向を把握するのに役立ち、メモリ関連の障害をデバッグするだけでなく、潜在的な障害を回避するのにも役立ちます。 有効なディメンション: JobName (AWS Glue ジョブの名前)、JobRunId (JobRun ID. または ALL)、Type (ゲージ)、ObservabilityGroup (resource_utilization) 有効な統計: Average 単位: バイト	resource_utilization

メトリクス	説明	カテゴリ
glue.driver.memory.heap.used.percentage	メトリクスカテゴリ: resource_utilization ジョブ実行中にドライバが (%) 使用したヒープメモリ。これは、特に時間の経過に伴うメモリ使用量の傾向を把握するのに役立ち、メモリ関連の障害をデバッグするだけでなく、潜在的な障害を回避するのにも役立ちます。 有効なディメンション: JobName (AWS Glue ジョブの名前)、JobRunId (JobRun ID. または ALL)、Type (ゲージ)、ObservabilityGroup (resource_utilization) 有効な統計: Average 単位: パーセント	resource_utilization

メトリクス	説明	カテゴリ
glue.driver.memory.non-heap.[available | used]	メトリクスカテゴリ: resource_utilization ジョブ実行中のドライバの使用可能/使用済み非ヒープメモリ。これは、特に時間の経過に伴うメモリ使用量の傾向を把握するのに役立ち、メモリ関連の障害をデバッグするだけでなく、潜在的な障害を回避するのにも役立ちます。 有効なディメンション: JobName (AWS Glue ジョブの名前)、JobRunId (JobRun ID. または ALL)、Type (ゲージ)、ObservabilityGroup (resource_utilization) 有効な統計: Average 単位: バイト	resource_utilization

メトリクス	説明	カテゴリ
glue.driver.memory.non-heap.used.percentage	メトリクスカテゴリ: resource_utilization ジョブ実行中にドライバが (%) 使用した非ヒープメモリ。これは、特に時間の経過に伴うメモリ使用量の傾向を把握するのに役立ち、メモリ関連の障害をデバッグするだけでなく、潜在的な障害を回避するのにも役立ちます。 有効なディメンション: JobName (AWS Glue ジョブの名前)、JobRunId (JobRun ID. または ALL)、Type (ゲージ)、ObservabilityGroup (resource_utilization) 有効な統計: Average 単位: パーセント	resource_utilization

メトリクス	説明	カテゴリ
glue.driver.memory.total.[available | used]	メトリクスカテゴリ: resource_utilization ジョブ実行中のドライバの使用可能/使用済み合計メモリ。これは、特に時間の経過に伴うメモリ使用量の傾向を把握するのに役立ち、メモリ関連の障害をデバッグするだけでなく、潜在的な障害を回避するのにも役立ちます。 有効なディメンション: JobName (AWS Glue ジョブの名前)、JobRunId (JobRun ID. または ALL)、Type (ゲージ)、ObservabilityGroup (resource_utilization) 有効な統計: Average 単位: バイト	resource_utilization

メトリクス	説明	カテゴリ
glue.driver.memory.total.used.percentage	メトリクスカテゴリ: resource_utilization ジョブ実行中にドライバが (%) 使用した合計メモリ。これは、特に時間の経過に伴うメモリ使用量の傾向を把握するのに役立ち、メモリ関連の障害をデバッグするだけでなく、潜在的な障害を回避するのにも役立ちます。 有効なディメンション: JobName (AWS Glue ジョブの名前)、JobRunId (JobRun ID. または ALL)、Type (ゲージ)、ObservabilityGroup (resource_utilization) 有効な統計: Average 単位: パーセント	resource_utilization

メトリクス	説明	カテゴリ
glue.ALL.memory.heap.[available | used]	メトリクスカテゴリ: resource_utilization エグゼキューターの使用可能/使用済みヒープメモリ。ALL はすべてのエグゼキューターを意味します。 有効なディメンション: JobName (AWS Glue ジョブの名前)、JobRunId (JobRun ID. または ALL)、Type (ゲージ)、ObservabilityGroup (resource_utilization) 有効な統計: Average 単位: バイト	resource_utilization
glue.ALL.memory.heap.used.percentage	メトリクスカテゴリ: resource_utilization エグゼキューターの (%) 使用済みヒープメモリ。ALL はすべてのエグゼキューターを意味します。 有効なディメンション: JobName (AWS Glue ジョブの名前)、JobRunId (JobRun ID. または ALL)、Type (ゲージ)、ObservabilityGroup (resource_utilization) 有効な統計: Average 単位: パーセント	resource_utilization

メトリクス	説明	カテゴリ
glue.ALL.memory.non-heap.[available | used]	メトリクスカテゴリ: resource_utilization エグゼキューターの使用可能/使用済み非ヒープメモリ。ALL はすべてのエグゼキューターを意味します。 有効なディメンション: JobName (AWS Glue ジョブの名前)、JobRunId (JobRun ID. または ALL)、Type (ゲージ)、ObservabilityGroup (resource_utilization) 有効な統計: Average 単位: バイト	resource_utilization
glue.ALL.memory.non-heap.used.percentage	メトリクスカテゴリ: resource_utilization エグゼキューターの (%) 使用済み非ヒープメモリ。ALL はすべてのエグゼキューターを意味します。 有効なディメンション: JobName (AWS Glue ジョブの名前)、JobRunId (JobRun ID. または ALL)、Type (ゲージ)、ObservabilityGroup (resource_utilization) 有効な統計: Average 単位: パーセント	resource_utilization

メトリクス	説明	カテゴリ
glue.ALL.memory.total.[available | used]	メトリクスカテゴリ: resource_utilization エグゼキューターの使用可能/使用済み合計メモリ。ALL はすべてのエグゼキューターを意味します。 有効なディメンション: JobName (AWS Glue ジョブの名前)、JobRunId (JobRun ID. または ALL)、Type (ゲージ)、ObservabilityGroup (resource_utilization) 有効な統計: Average 単位: バイト	resource_utilization
glue.ALL.memory.total.used.percentage	メトリクスカテゴリ: resource_utilization エグゼキューターの (%) 使用済み合計メモリ。ALL はすべてのエグゼキューターを意味します。 有効なディメンション: JobName (AWS Glue ジョブの名前)、JobRunId (JobRun ID. または ALL)、Type (ゲージ)、ObservabilityGroup (resource_utilization) 有効な統計: Average 単位: パーセント	resource_utilization

メトリクス	説明	カテゴリ
glue.driver.disk.[available_GB | used_GB]	メトリクスカテゴリ: resource_utilization ジョブ実行中のドライバの使用可能/使用済みディスク容量。これは、特に時間の経過に伴うディスク使用量の傾向を把握するのに役立ち、潜在的な障害を回避できるだけでなく、ディスク容量不足に関連する障害をデバッグするのにも役立ちます。 有効なディメンション: JobName (AWS Glue ジョブの名前)、JobRunId (JobRun ID. または ALL)、Type (ゲージ)、ObservabilityGroup (resource_utilization) 有効な統計: Average 単位: ギガバイト	resource_utilization

メトリクス	説明	カテゴリ
glue.driver.disk.used.percentage]	メトリクスカテゴリ: resource_utilization ジョブ実行中のドライバの使用可能/使用済みディスク容量。これは、特に時間の経過に伴うディスク使用量の傾向を把握するのに役立ち、潜在的な障害を回避できるだけでなく、ディスク容量不足に関連する障害をデバッグするのにも役立ちます。 有効なディメンション: JobName (AWS Glue ジョブの名前)、JobRunId (JobRun ID. または ALL)、Type (ゲージ)、ObservabilityGroup (resource_utilization) 有効な統計: Average 単位: パーセント	resource_utilization

メトリクス	説明	カテゴリ
glue.ALL.disk.[available_GB | used_GB]	メトリクスカテゴリ: resource_utilization エグゼキューターの使用可能/使用済みディスク容量。ALL はすべてのエグゼキューターを意味します。 有効なディメンション: JobName (AWS Glue ジョブの名前)、JobRunId (JobRun ID. または ALL)、Type (ゲージ)、ObservabilityGroup (resource_utilization) 有効な統計: Average 単位: ギガバイト	resource_utilization
glue.ALL.disk.used.percentage	メトリクスカテゴリ: resource_utilization エグゼキューターの使用可能/使用済み/(%) 使用済みディスク容量。ALL はすべてのエグゼキューターを意味します。 有効なディメンション: JobName (AWS Glue ジョブの名前)、JobRunId (JobRun ID. または ALL)、Type (ゲージ)、ObservabilityGroup (resource_utilization) 有効な統計: Average 単位: パーセント	resource_utilization

メトリクス	説明	カテゴリ
glue.driver.bytesRead	メトリクスカテゴリ: throughput このジョブ実行で入力ソースごとに読み取られたバイト数、およびすべてのソースで読み取られたバイト数。これにより、データ量とその経時的な変化を把握でき、データの歪みなどの問題に対処しやすくなります。 有効なディメンション: JobName (AWS Glue ジョブの名前)、JobRunId (JobRun ID. または ALL)、Type (ゲージ)、ObservabilityGroup (resource_utilization)、Source (ソースデータの場所) 有効な統計: Average 単位: バイト	スループット

メトリクス	説明	カテゴリ
glue.driver.[recordsRead | filesRead]	メトリクスカテゴリ: throughput このジョブ実行で入力ソースごとに読み取られたレコード/ファイルの数、およびすべてのソースで読み取られたレコード/ファイルの数。これにより、データ量とその経時的な変化を把握でき、データの歪みなどの問題に対処しやすくなります。 有効なディメンション: JobName (AWS Glue ジョブの名前)、JobRunId (JobRun ID. または ALL)、Type (ゲージ)、ObservabilityGroup (resource_utilization)、Source (ソースデータの場所) 有効な統計: Average 単位: 数	スループット

メトリクス	説明	カテゴリ
glue.driver.partitionsRead	メトリクスカテゴリ: throughput このジョブ実行で Amazon S3 入力ソースごとに読み取られたパーティションの数、およびすべてのソースで読み取られたのパーティション数。 有効なディメンション: JobName (AWS Glue ジョブの名前)、JobRunId (JobRun ID. または ALL)、Type (ゲージ)、ObservabilityGroup (resource_utilization)、Source (ソースデータの場所) 有効な統計: Average 単位: 数	スループット

メトリクス	説明	カテゴリ
glue.driver.bytesWrittten	メトリクスカテゴリ: throughput このジョブ実行で出力シンク 1 つあたりに書き込まれたバイト数、およびすべてのシンクで書き込まれたバイト数。これにより、データ量とその経時的変化を把握でき、処理の歪みなどの問題に対処しやすくなります。 有効なディメンション: JobName (AWS Glue ジョブの名前)、JobRunId (JobRun ID. または ALL)、Type (ゲージ)、ObservabilityGroup (resource_utilization)、Sink (シンクデータの場所) 有効な統計: Average 単位: バイト	スループット

メトリクス	説明	カテゴリ
glue.driver.[recordsWritten | filesWritten]	メトリクスカテゴリ: throughput このジョブ実行で出力シンクごとに書き込まれるレコード/ファイルの数、およびすべてのシンクで書き込まれるレコード/ファイルの数。これにより、データ量とその経時的変化を把握でき、処理の歪みなどの問題に対処しやすくなります。 有効なディメンション: JobName (AWS Glue ジョブの名前)、JobRunId (JobRun ID. または ALL)、Type (ゲージ)、ObservabilityGroup (resource_utilization)、Sink (シンクデータの場所) 有効な統計: Average 単位: 数	スループット

エラーカテゴリ

エラーカテゴリ	説明
COMPILATION_ERROR	Scala コードのコンパイル中に発生するエラー。
CONNECTION_ERROR	サービス/リモート、ホスト/データベースのサービスなどへの接続中に発生するエラー。

エラーカテゴリ	説明
DISK_NO_SPACE_ERROR	ドライバ/エグゼキューターのディスクに空きがないと発生するエラー。
OUT_OF_MEMORY_ERROR	ドライバ/エグゼキューターのメモリに空きがないと発生するエラー。
IMPORT_ERROR	依存関係をインポートすると発生するエラー。
INVALID_ARGUMENT_ERROR	入力引数が無効または不正なときにエラーが発生します。
PERMISSION_ERROR	サービス、データなどに対する権限がないと発生するエラー。
RESOURCE_NOT_FOUND_ERROR	データ、位置情報などが存在しないと発生するエラー。
QUERY_ERROR	Spark SQL クエリの実行によって発生するエラー。
SYNTAX_ERROR	スクリプトに構文エラーがあると発生するエラー。
THROTTLING_ERROR	サービスの同時実行数の制限に達したり、サービスクォータの制限を超えたりすると発生するエラー。
DATA_LAKE_FRAMEWORK_ERROR	AWS Glue がネイティブでサポートするデータレイクフレームワーク (Hudi、Iceberg など) から発生するエラー。
UNSUPPORTED_OPERATION_ERROR	サポートされていない操作を行うと発生するエラー。
RESOURCES_ALREADY_EXISTS_ERROR	作成または追加するリソースがすでに存在していると発生するエラー。

エラーカテゴリ	説明
GLUE_INTERNAL_SERVICE_ERROR	AWS Glue の内部サービスに問題があると発生するエラー。
GLUE_OPERATION_TIMEOUT_ERROR	AWS Glue 操作がタイムアウトになると発生するエラー。
GLUE_VALIDATION_ERROR	AWS Glue ジョブに必要な値を検証できなかった場合に発生するエラー。
GLUE_JOB_BOOKMARK_VERSION_MISMATCH_ERROR	同じソースバケットで同じジョブを実行し、同じ/異なる送信先に同時に書き込んだ場合 (同時実行数 >1) に発生するエラー。
LAUNCH_ERROR	AWS Glue ジョブの起動段階で発生するエラー。
DYNAMODB_ERROR	Amazon DynamoDB サービスから発生する一般的なエラー。
GLUE_ERROR	AWS Glue サービスから発生する一般的なエラー。
LAKEFORMATION_ERROR	AWS Lake Formation サービスから発生する一般的なエラー。
REDSHIFT_ERROR	Amazon Redshift サービスから発生する一般的なエラー。
S3_ERROR	Amazon S3 サービスから発生する一般的なエラー。
SYSTEM_EXIT_ERROR	一般的なシステム終了エラー。
TIMEOUT_ERROR	操作がタイムアウトしてジョブが失敗すると発生する一般的なエラー。
UNCLASSIFIED_SPARK_ERROR	Spark から発生する一般的なエラー。

エラーカテゴリ	説明
UNCLASSIFIED_ERROR	デフォルトのエラーカテゴリ。

制限

> **Note**
>
> メトリクスを公開するには、`glueContext` を初期化する必要があります。

ソースディメンションの値は、ソースタイプに応じて Amazon S3 パスまたはテーブル名のいずれかになります。さらに、ソースが JDBC でクエリオプションが使用されている場合、クエリ文字列はソースディメンションに設定されます。値が 500 文字を超える場合は、500 文字以内に切り捨てられます。値には次の制限があります。

- ASCII 以外の文字は削除されます。
- ソース名に ASCII 文字が含まれていない場合は、<non-ASCII input> に変換されます。

スループットメトリクスの制約事項と考慮事項

- DataFrame と DataFrame ベースの DynamicFrame (例:JDBC、Amazon S3 の parquet からの読み取り) はサポートされていますが、RDD ベースの DynamicFrame (Amazon S3 での csv、json の読み取りなど) はサポートされていません。技術的には、Spark UI に表示されるすべての読み取りと書き込みがサポートされています。
- データソースがカタログテーブルで、形式が JSON、CSV、テキスト、または Iceberg の場合、`recordsRead` メトリクスが出力されます。
- `glue.driver.throughput.recordsWritten`、`glue.driver.throughput.bytesWritten`、`glu` メトリクスは JDBC テーブルと Iceberg テーブルでは使用できません。
- メトリクスは遅延する可能性があります。ジョブが約 1 分後に終了する場合、Amazon CloudWatch メトリクスにはスループットメトリクスがない可能性があります。

ジョブのモニタリングとデバッグ

AWS Glue ジョブに関するメトリクスを収集して AWS Glue および Amazon CloudWatch コンソールで可視化し、問題を特定して修正できます。AWS Glue ジョブのプロファイリングには、以下のステップが必要です。

1. メトリクスを有効にする:

   a. ジョブ定義で [Job metrics (ジョブメトリクス)] オプションを有効にします。AWS Glue コンソールでプロファイリングを有効にするか、ジョブに対するパラメータとして有効にできます。詳細については、「Spark ジョブのジョブプロパティの定義」または「AWS Glue ジョブでジョブパラメータを使用する」を参照してください。

   b. ジョブ定義で [AWS Glue オブザーバビリティメトリクス] オプションを有効にします。AWS Glue コンソールでオブザーバビリティを有効にするか、ジョブに対するパラメータとして有効にできます。詳細については、「AWS Glue オブザーバビリティメトリクスを使用したモニタリング」を参照してください。

2. ジョブスクリプトが `GlueContext` を初期化することを確認します。たとえば、次のスクリプトスニペットは `GlueContext` を初期化し、プロファイリングされたコードが配置されるスクリプト内の場所を示しています。この一般的な形式は、以下のデバッグシナリオで使用されます。

```
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job
import time

## @params: [JOB_NAME]
args = getResolvedOptions(sys.argv, ['JOB_NAME'])

sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
job.init(args['JOB_NAME'], args)

...
...
```

```
code-to-profile
...
...


job.commit()
```

3. ジョブを実行します。
4. メトリクスを可視化する:
   a. AWS Glue コンソールでジョブメトリクスを可視化し、ドライバまたはエグゼキューターの異常なメトリクスを識別します。
   b. オブザーバビリティメトリクスは、「ジョブ実行モニタリング」ページ、「ジョブ実行詳細」ページ、または Amazon CloudWatch で確認できます。詳細については、「AWS Glue オブザーバビリティメトリクスを使用したモニタリング」を参照してください。
5. 特定されたメトリクスを使用して根本原因を絞り込みます。
6. 必要に応じて、識別されたドライバーまたはジョブエグゼキューターのログストリームを使用して根本原因を確認します。

AWS Glue オブザーバビリティメトリクスのユースケース

- OOM 例外とジョブの異常のデバッグ
- 要求の厳しいステージとストラグラータスクのデバッグ
- 複数のジョブの進行状況のモニタリング
- DPU の容量計画のモニタリング
- AWS Glue オブザーバビリティを使用してリソースの使用状況を監視し、コストを削減します。

OOM 例外とジョブの異常のデバッグ

AWS Glue でメモリ不足 (OOM) 例外とジョブの異常をデバッグできます。以下のセクションでは、Apache Spark ドライバーや Spark エグゼキューターのメモリ不足例外をデバッグするためのシナリオについて説明します。

- ドライバー OOM 例外のデバッグ
- エグゼキューター OOM 例外のデバッグ

ドライバー OOM 例外のデバッグ

このシナリオでは、Spark ジョブで多数の小さいファイルが Amazon Simple Storage Service (Amazon S3) から読み込まれます。ファイルが Apache Parquet 形式に変換された後、Amazon S3 に書き込まれます。Spark ドライバーのメモリが不足しています。入力 Amazon S3 データの複数の Amazon S3 パーティションに 100 万を超えるファイルがあります。

プロファイルされたコードは次のとおりです。

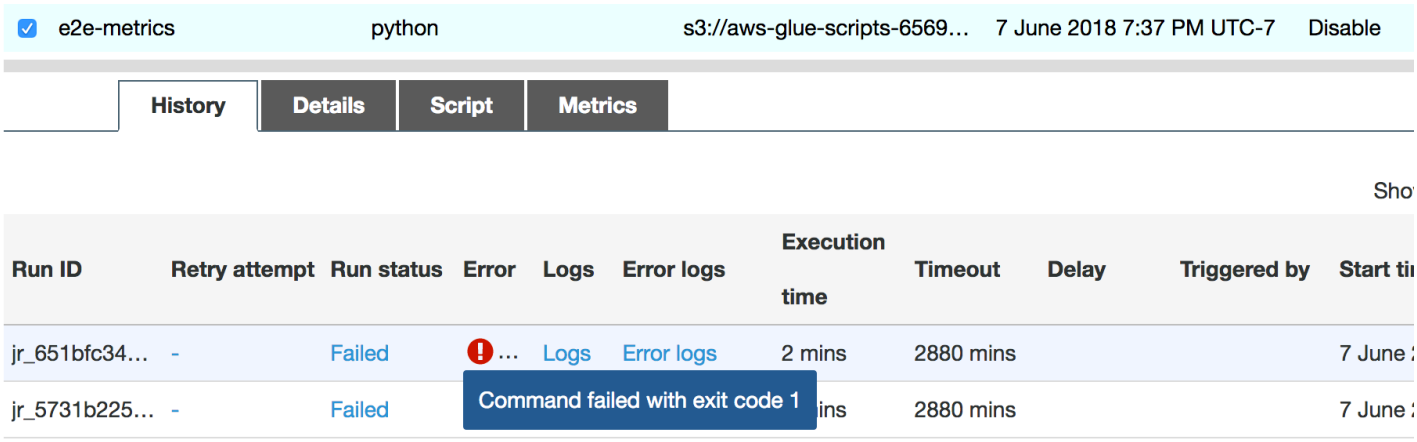
```
data = spark.read.format("json").option("inferSchema", False).load("s3://input_path")
data.write.format("parquet").save(output_path)
```

AWS Glue コンソールでプロファイルされたメトリクスを可視化する

以下のグラフでは、ドライバーとエグゼキューターのメモリ使用率がパーセントで示されています。この使用率は、直近の 1 分間に報告された値を平均した 1 つのデータポイントとしてプロットされます。ジョブのメモリプロファイルでは、ドライバーメモリが安全しきい値である使用率の 50% をすぐに超えることがわかります。一方、すべてのエグゼキューターにおける平均メモリ使用率は、まだ 4% 未満です。これは、この Spark ジョブにおけるドライバー実行に異常があることを明確に示しています。



ジョブの実行はすぐに失敗し、AWS Glue コンソールの [History] (履歴) タブにエラー「Command Failed with Exit Code 1」が表示されます。このエラー文字列は、システムエラーのためにジョブが失敗したことを意味します。この場合はドライバーのメモリ不足です。



コンソールの [History] (履歴) タブにある [Error logs] (エラーログ) リンクを選択し、CloudWatch Logs からのドライバー OOM に関する詳細情報を確認します。ジョブのエラーログで「**`Error`**」を検索し、それが本当にジョブの失敗の原因となった OOM 例外であることを確認します。

```
# java.lang.OutOfMemoryError: Java heap space
# -XX:OnOutOfMemoryError="kill -9 %p"
# Executing /bin/sh -c "kill -9 12039"...
```

ジョブの [履歴] タブで、[ログ] を選択します。ジョブの開始時における CloudWatch Logs 内のドライバー実行の以下のトレースが表示されます。Spark ドライバーは、すべてのディレクトリのすべてのファイルのリストの取得を試み、`InMemoryFileIndex` を構築してファイルごとに 1 つのタスクを起動します。この結果、Spark ドライバーは、すべてのタスクを追跡するため、大量の状態をメモリに保持する必要が生じます。また、メモリ内インデックスの多数のファイルの完全なリストを取得するため、ドライバー OOM となります。

グループ化を使用した複数のファイルの処理を修正する

でグループ化AWS Glue機能を使用して、複数のファイルの処理を修正できます。グループ化は、動的なフレームを使用しているときと、入力データセットに多数のファイル (50,000 超) があるときに自動的に有効になります。グループかにより、複数のファイルを 1 つのグループにまとめることができ、タスクは単一のファイルではなくグループ全体を処理できるようになります。その結果、Spark ドライバーがメモリに保存する状態がかなり少なくなり、追跡するタスクが減少します。

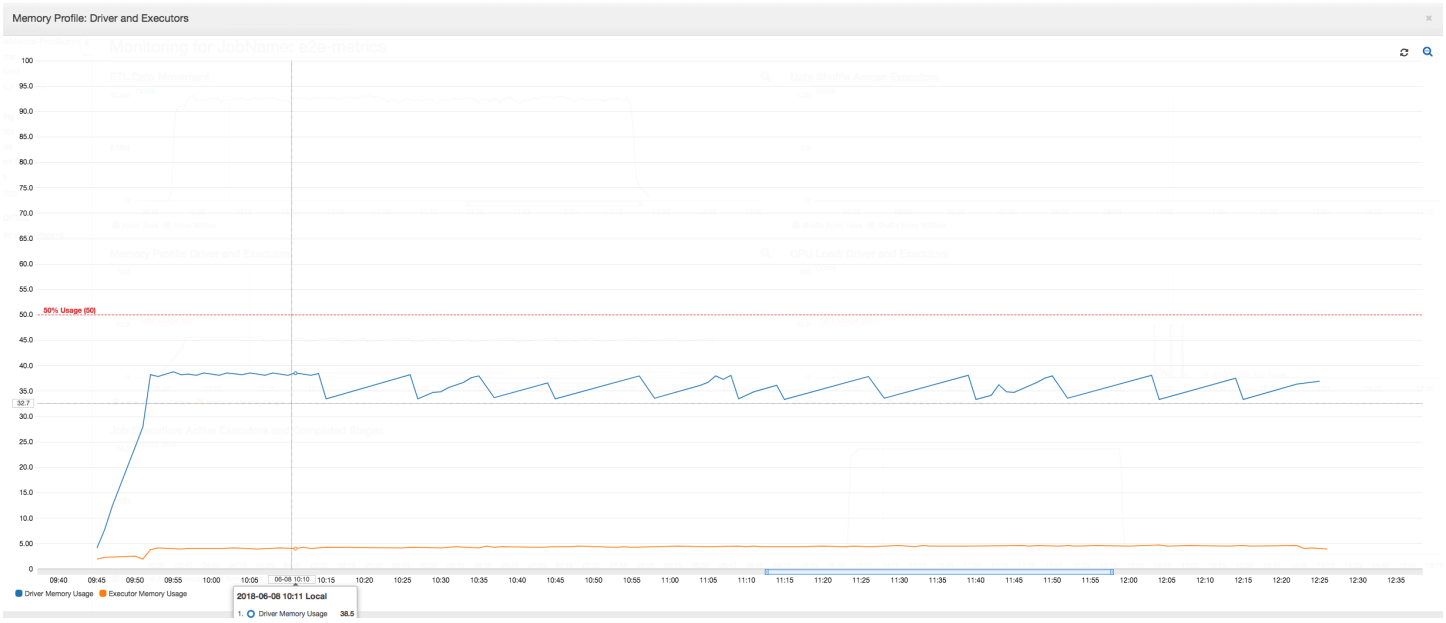
データセットのグループ化を手動で有効にする方法の詳細については、「大きなグループの入力ファイルの読み取り」を参照してください。

AWS Glue ジョブのメモリプロファイルを確認するには、グループ化を有効にして次のコードをプロファイルします。

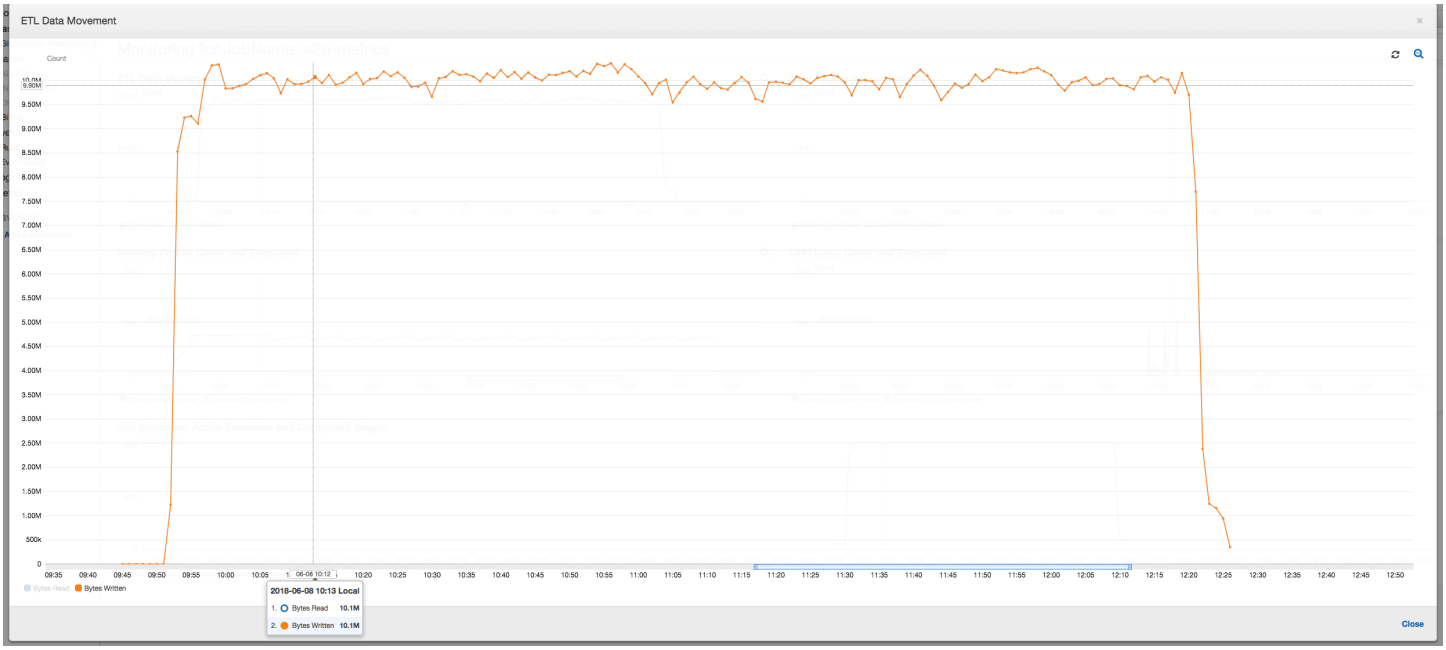
```
df = glueContext.create_dynamic_frame_from_options("s3", {'paths': ["s3://input_path"],
 "recurse":True, 'groupFiles': 'inPartition'}, format="json")
datasink = glueContext.write_dynamic_frame.from_options(frame = df, connection_type
 = "s3", connection_options = {"path": output_path}, format = "parquet",
 transformation_ctx = "datasink")
```

AWS Glue ジョブプロファイルでは、メモリプロファイルと ETL データ移動を監視できます。

ドライバーは、AWS Glue ジョブの持続期間全体にわたって、メモリ使用率のしきい値である 50% 未満で実行されます。エグゼキューターは、Amazon S3 からデータをストリーミングして処理し、Amazon S3 に書き出します。その結果、消費されるメモリは常に 5% 未満となります。



以下のデータ移動プロファイルは、ジョブが進行するにつれてすべてのエグゼキューターにより直近 1 分間に読み取りおよび書き込みされた Amazon S3 バイトの合計数を示しています。データはすべてのエグゼキューターでストリーミングされるため、どちらも同様のパターンに従っています。ジョブは、100 万ファイルすべての処理を 3 時間未満で完了します。



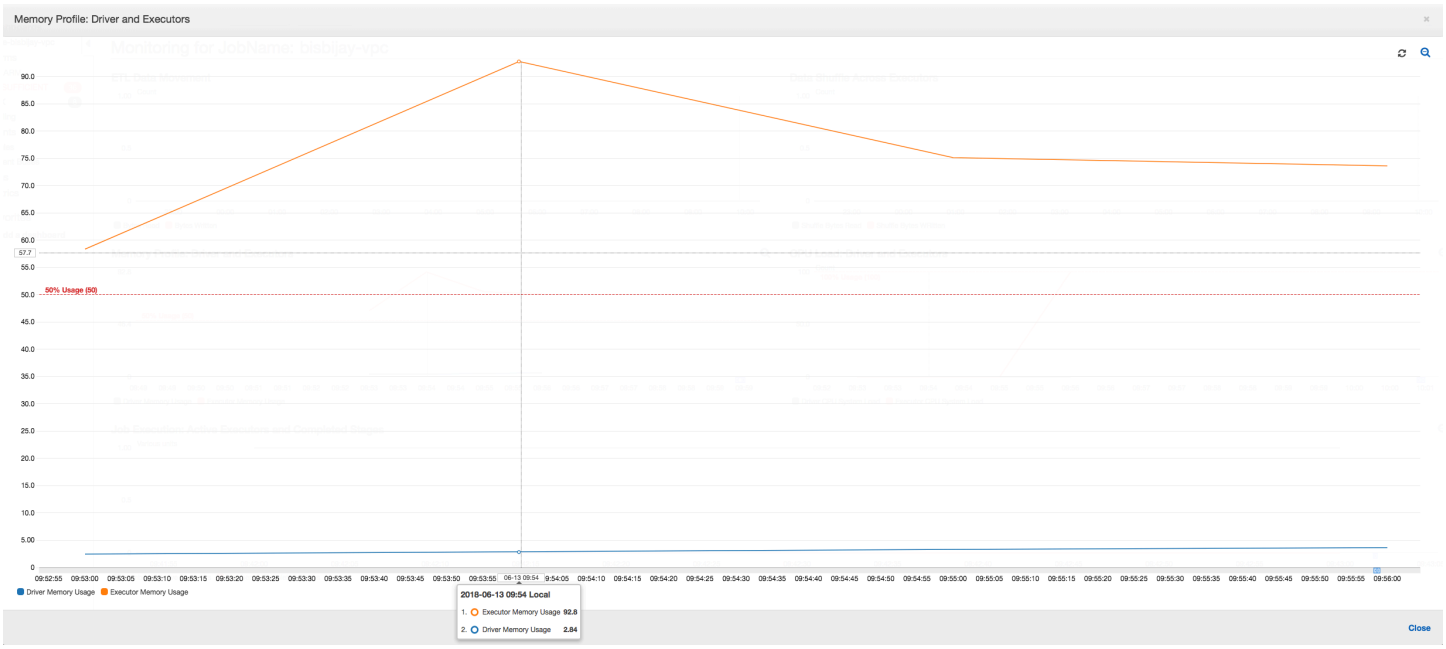
エグゼキューター OOM 例外のデバッグ

このシナリオでは、Apache Spark エグゼキューターで発生する可能性がある OOM 例外をデバッグする方法について説明します。次のコードでは、Spark MySQL リーダーを使用して、約 34 万行の大きなテーブルを Spark データフレームに読み取ります。次に、Parquet 形式で Amazon S3 に書き出します。接続プロパティを用意し、デフォルト Spark 設定を使用してテーブルを読み取ることができます。

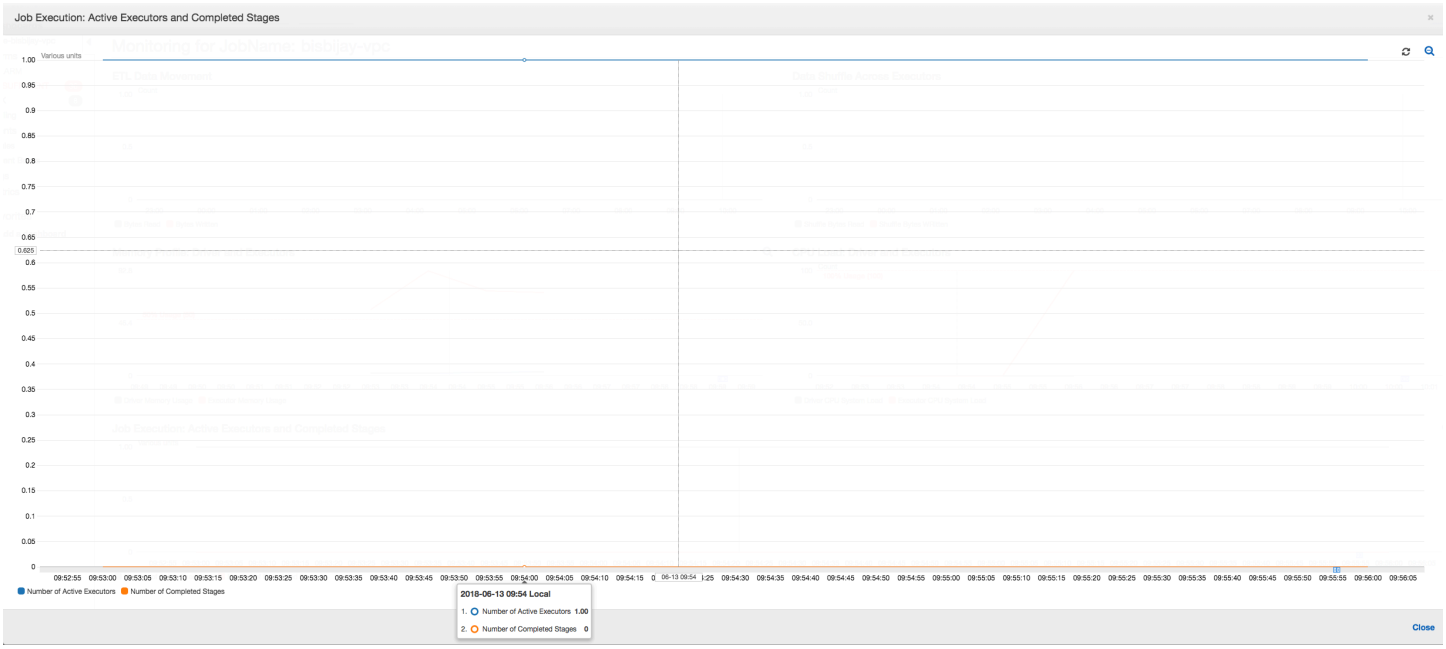
```
val connectionProperties = new Properties()
connectionProperties.put("user", user)
connectionProperties.put("password", password)
connectionProperties.put("Driver", "com.mysql.jdbc.Driver")
val sparkSession = glueContext.sparkSession
val dfSpark = sparkSession.read.jdbc(url, tableName, connectionProperties)
dfSpark.write.format("parquet").save(output_path)
```

AWS Glue コンソールでプロファイルされたメトリクスを可視化する

メモリ使用量グラフの傾きが正で 50% を超えた場合、および次のメトリクスが出力される前にジョブが失敗した場合は、メモリ枯渇が原因の候補となります。以下のグラフは、実行してから 1 分以内に、すべてのエグゼキューターにおける平均メモリ使用率がすぐに 50% を超えることを示しています。使用率は最大 92% に上昇し、エグゼキューターを実行するコンテナは Apache Hadoop YARN により停止されます。

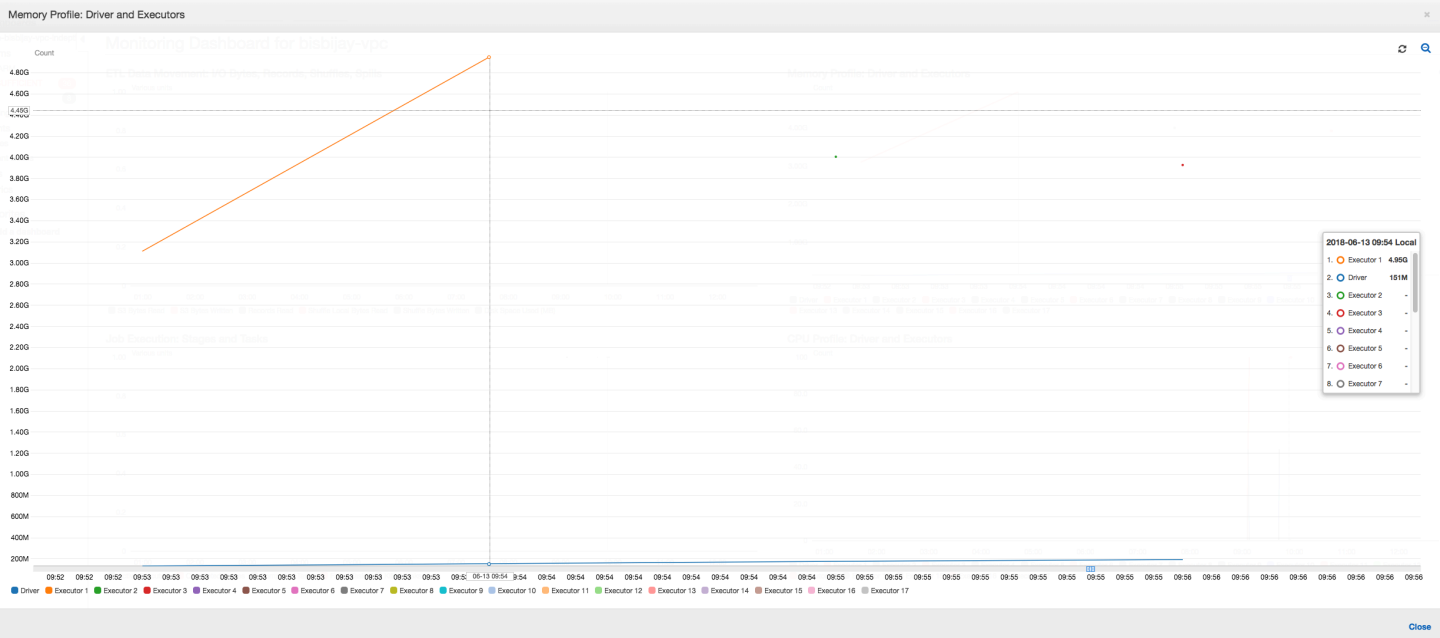


次のグラフが示すよう、ジョブが失敗するまで常に単一のエグゼキューターが実行されています。これは、新しいエグゼキューターが起動され、停止されたエグゼキューターが置き換えられるためです。JDBC データソースの読み取りは、列上のテーブルをパーティション化して複数の接続を開く必要があるため、デフォルトでは並列化されません。その結果、1 つのエグゼキューターだけがテーブル全体を順番に読み込みます。



次のグラフが示すように、Spark はジョブが失敗するまでに 4 回新しいタスクを起動しようとします。3 つのエグゼキューターのメモリプロファイルを確認できます。各エグゼキューターはすべてのメモ

リをすぐに使い尽くします。4 番目のエグゼキューターのメモリが不足し、ジョブは失敗します。その結果、そのメトリクスはすぐに報告されません。



次の図に示すように、AWS Glue コンソールのエラー文字列からは、ジョブが OOM 例外のために失敗したことを確認できます。



ジョブ出力ログ: エグゼキューター OOM 例外の結果をさらに確認するには、CloudWatch Logs を参照します。「**Error**」を検索すると、メトリクスダッシュボードに示されているように、4 つのエグゼキューターがほぼ同じ時間枠で停止されています。メモリ制限を超えると、すべて YARN により終了されます。

エグゼキューター 1

```
18/06/13 16:54:29 WARN YarnAllocator: Container killed by YARN for exceeding
 memory limits. 5.5 GB of 5.5 GB physical memory used. Consider boosting
 spark.yarn.executor.memoryOverhead.
18/06/13 16:54:29 WARN YarnSchedulerBackend$YarnSchedulerEndpoint: Container killed
 by YARN for exceeding memory limits. 5.5 GB of 5.5 GB physical memory used. Consider
 boosting spark.yarn.executor.memoryOverhead.
18/06/13 16:54:29 ERROR YarnClusterScheduler: Lost executor 1 on
 ip-10-1-2-175.ec2.internal: Container killed by YARN for exceeding
```

```
 memory limits. 5.5 GB of 5.5 GB physical memory used. Consider boosting
 spark.yarn.executor.memoryOverhead.
18/06/13 16:54:29 WARN TaskSetManager: Lost task 0.0 in stage 0.0 (TID 0,
 ip-10-1-2-175.ec2.internal, executor 1): ExecutorLostFailure (executor 1
 exited caused by one of the running tasks) Reason: Container killed by YARN for
 exceeding memory limits. 5.5 GB of 5.5 GB physical memory used. Consider boosting
 spark.yarn.executor.memoryOverhead.
```

エグゼキュター 2

```
18/06/13 16:55:35 WARN YarnAllocator: Container killed by YARN for exceeding
 memory limits. 5.8 GB of 5.5 GB physical memory used. Consider boosting
 spark.yarn.executor.memoryOverhead.
18/06/13 16:55:35 WARN YarnSchedulerBackend$YarnSchedulerEndpoint: Container killed
 by YARN for exceeding memory limits. 5.8 GB of 5.5 GB physical memory used. Consider
 boosting spark.yarn.executor.memoryOverhead.
18/06/13 16:55:35 ERROR YarnClusterScheduler: Lost executor 2 on
 ip-10-1-2-16.ec2.internal: Container killed by YARN for exceeding
 memory limits. 5.8 GB of 5.5 GB physical memory used. Consider boosting
 spark.yarn.executor.memoryOverhead.
18/06/13 16:55:35 WARN TaskSetManager: Lost task 0.1 in stage 0.0 (TID 1,
 ip-10-1-2-16.ec2.internal, executor 2): ExecutorLostFailure (executor 2 exited
 caused by one of the running tasks) Reason: Container killed by YARN for
 exceeding memory limits. 5.8 GB of 5.5 GB physical memory used. Consider boosting
 spark.yarn.executor.memoryOverhead.
```

エグゼキュター 3

```
18/06/13 16:56:37 WARN YarnAllocator: Container killed by YARN for exceeding
 memory limits. 5.8 GB of 5.5 GB physical memory used. Consider boosting
 spark.yarn.executor.memoryOverhead.
18/06/13 16:56:37 WARN YarnSchedulerBackend$YarnSchedulerEndpoint: Container killed
 by YARN for exceeding memory limits. 5.8 GB of 5.5 GB physical memory used. Consider
 boosting spark.yarn.executor.memoryOverhead.
18/06/13 16:56:37 ERROR YarnClusterScheduler: Lost executor 3 on
 ip-10-1-2-189.ec2.internal: Container killed by YARN for exceeding
 memory limits. 5.8 GB of 5.5 GB physical memory used. Consider boosting
 spark.yarn.executor.memoryOverhead.
18/06/13 16:56:37 WARN TaskSetManager: Lost task 0.2 in stage 0.0 (TID 2,
 ip-10-1-2-189.ec2.internal, executor 3): ExecutorLostFailure (executor 3
 exited caused by one of the running tasks) Reason: Container killed by YARN for
```

```
 exceeding memory limits. 5.8 GB of 5.5 GB physical memory used. Consider boosting
 spark.yarn.executor.memoryOverhead.
```

エグゼキュター 4

```
18/06/13 16:57:18 WARN YarnAllocator: Container killed by YARN for exceeding
 memory limits. 5.5 GB of 5.5 GB physical memory used. Consider boosting
 spark.yarn.executor.memoryOverhead.
18/06/13 16:57:18 WARN YarnSchedulerBackend$YarnSchedulerEndpoint: Container killed
 by YARN for exceeding memory limits. 5.5 GB of 5.5 GB physical memory used. Consider
 boosting spark.yarn.executor.memoryOverhead.
18/06/13 16:57:18 ERROR YarnClusterScheduler: Lost executor 4 on
 ip-10-1-2-96.ec2.internal: Container killed by YARN for exceeding
 memory limits. 5.5 GB of 5.5 GB physical memory used. Consider boosting
 spark.yarn.executor.memoryOverhead.
18/06/13 16:57:18 WARN TaskSetManager: Lost task 0.3 in stage 0.0 (TID 3,
 ip-10-1-2-96.ec2.internal, executor 4): ExecutorLostFailure (executor 4 exited
 caused by one of the running tasks) Reason: Container killed by YARN for
 exceeding memory limits. 5.5 GB of 5.5 GB physical memory used. Consider boosting
 spark.yarn.executor.memoryOverhead.
```

AWS Glue 動的フレームを使用してフェッチサイズ設定を修正する

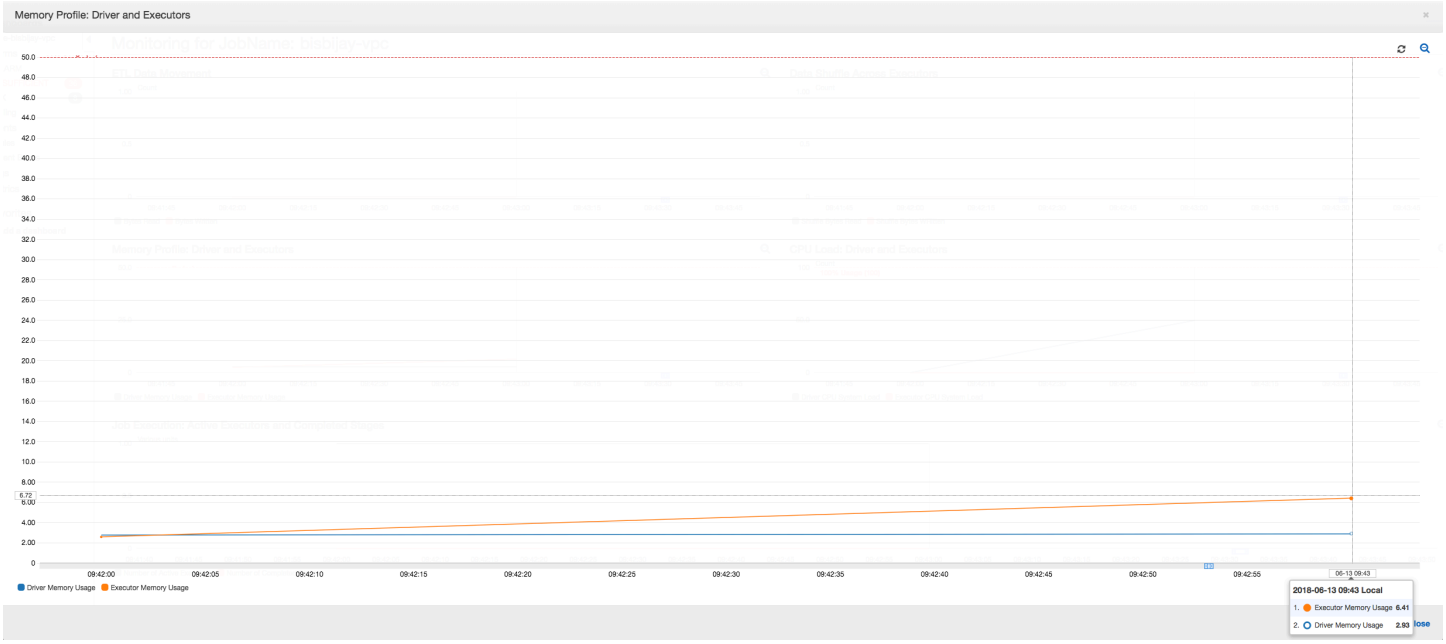
Spark JDBC フェッチサイズのデフォルトの設定が 0 であるため、エグゼキュターは JDBC テーブルの読み取り中にメモリ不足になります。つまり、Spark は行を一度に 1 つずつストリーミングしますが、Spark エグゼキュター上の JDBC ドライバーがデータベースから 3400 万行をまとめてフェッチして、それらをキャッシュしようとします。Spark を使用すると、フェッチサイズパラメータを 0 以外のデフォルト値に設定することにより、このシナリオを回避できます。

代わりに AWS Glue 動的フレームを使用することで、この問題を解決することもできます。デフォルトでは、動的フレームは 1,000 行のフェッチサイズを使用します。これは通常十分な値です。このため、エグゼキュターが合計メモリの 7% 超を使用することはありません。AWS Glue ジョブは、エグゼキュターを 1 つだけ使用して 2 分未満で完了します。AWS Glue 動的フレームを使用することが推奨されるアプローチですが、Apache Spark の `fetchsize` プロパティを使用してフェッチサイズを設定することもできます。「Spark SQL、DataFrames および Datasets ガイド」を参照してください。

```
val (url, database, tableName) = {
 ("jdbc_url", "db_name", "table_name")
 }
val source = glueContext.getSource(format, sourceJson)
```

```
val df = source.getDynamicFrame
glueContext.write_dynamic_frame.from_options(frame = df, connection_type = "s3",
 connection_options = {"path": output_path}, format = "parquet", transformation_ctx =
 "datasink")
```

プロファイルされた通常のメトリクス: 以下の図に示すように、AWS Glue 動的フレームを持つエグゼキューターメモリが安全しきい値を超えることはありません。データベースから行をストリーミングし、任意の時点で 1,000 行のみ JDBC ドライバーにキャッシュします。メモリ不足例外は発生しません。



要求の厳しいステージとストラグラータスクのデバッグ

AWS Glue ジョブプロファイリングを使用して、抽出、変換、ロード (ETL) ジョブで要求の厳しいステージとストラグラータスクを特定できます。ストラグラータスクは、AWS Glue ジョブのステージに含まれる他のタスクよりかなり時間がかかります。その結果、ステージの完了まで時間がかかり、ジョブの合計実行時間も遅延します。

小さい入力ファイルを大きい出力ファイルにまとめる

ストラグラータスクは、さまざまなタスク間の処理が均等に分散していないときや、データスキューによって特定のタスクが処理するデータが多くなった場合に発生する可能性があります。

Apache Spark の一般的なパターンである次のコードをプロファイルし、多数の小さいファイルをいくつかの大きい出力ファイルにまとめることができます。この例では、入力データセットは 32 GB の JSON Gzip 圧縮ファイルです。出力データセットには、約 190 GB の圧縮されていない JSON ファイルが含まれています。

プロファイルされたコードは次のとおりです。

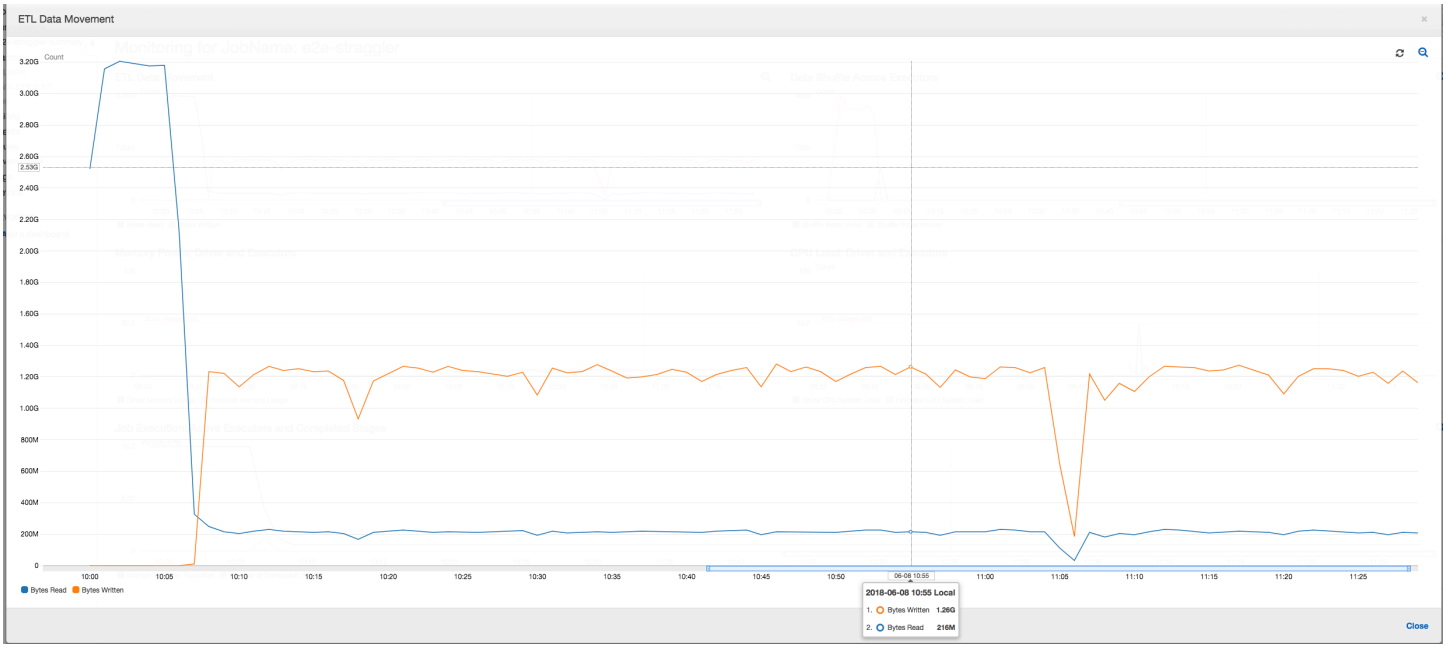
```
datasource0 = spark.read.format("json").load("s3://input_path")
df = datasource0.coalesce(1)
df.write.format("json").save(output_path)
```

AWS Glue コンソールでプロファイルされたメトリクスを可視化する

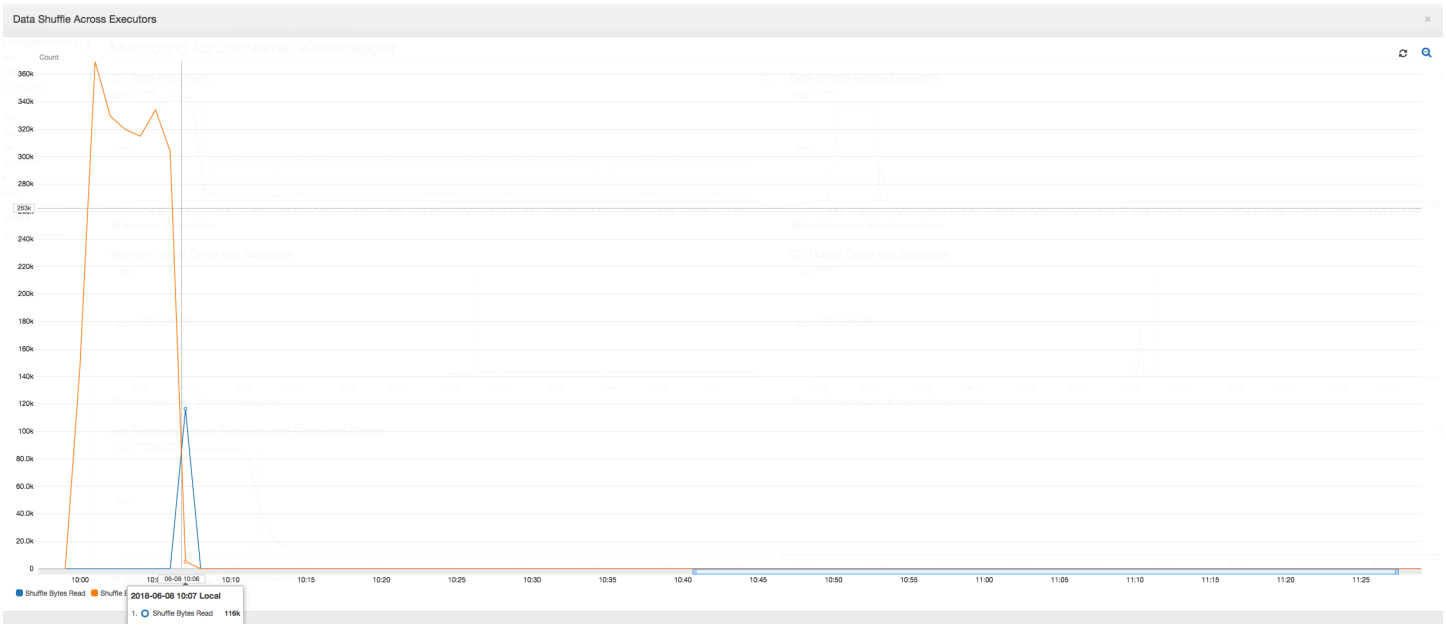
ジョブをプロファイルすると、4 つのメトリクスセットを調べることができます。

- ETL データ移動
- エグゼキュター間のデータシャッフル
- ジョブの実行
- メモリプロファイル

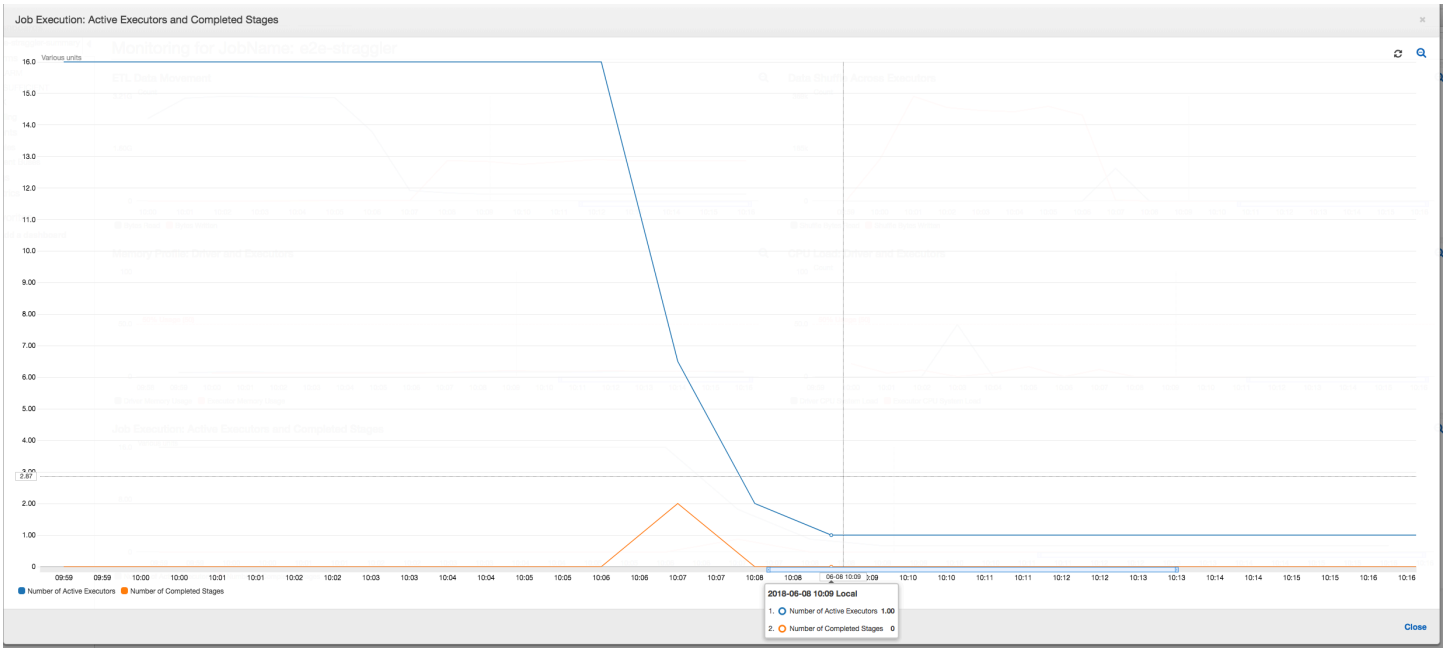
ETL データ移動: [ETL Data Movement (ETL データ移動)] プロファイルでは、バイトは、最初の 6 分間に完了する最初のステージのすべてのエグゼキュターにより比較的早く読み取られます。ただし、ジョブの合計実行時間が約 1 時間の場合、ほとんどがデータの書き込みで構成されます。



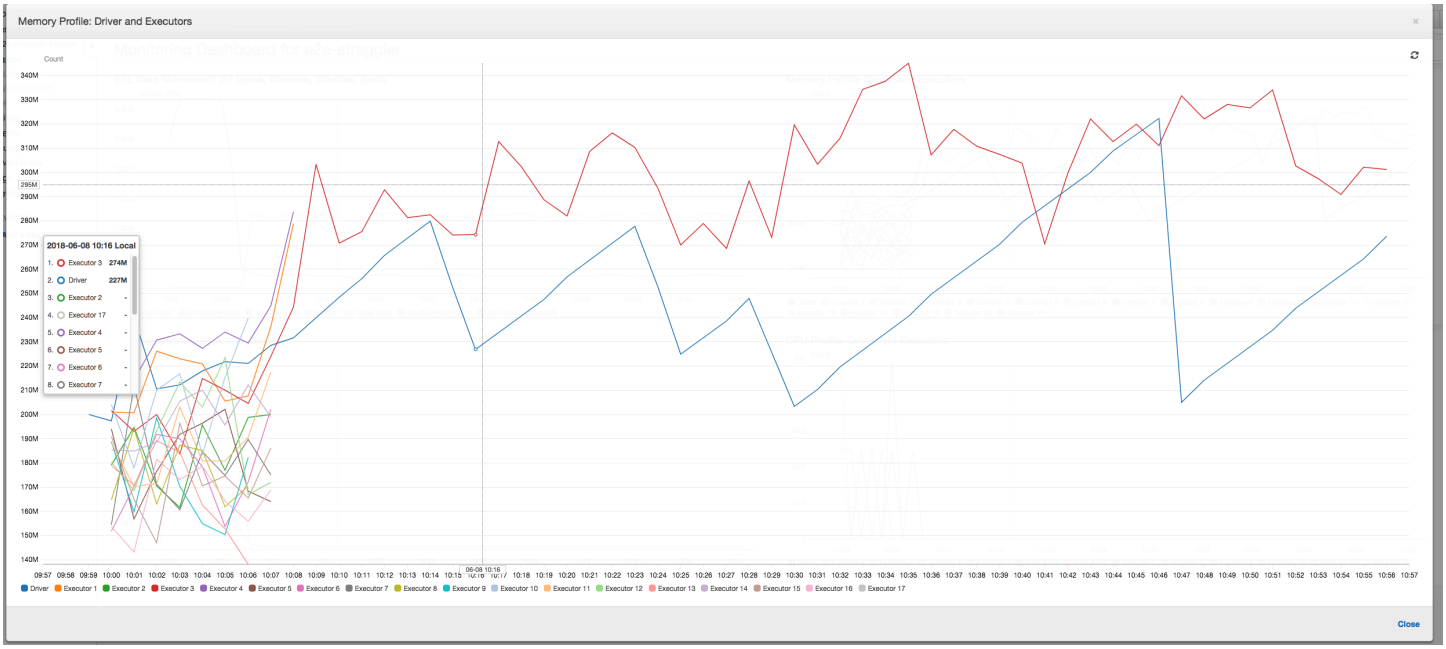
エグゼキュター間のデータシャッフル: ジョブ実行メトリクスとデータシャッフルメトリクスが示しているように、シャッフル中の読み取りおよび書き込みのバイト数も、ステージ 2 が終了する前に急増しています。すべてのエグゼキュターからのデータシャッフルの後、読み取りと書き込みはエグゼキュター 3 番のみから続行されます。



ジョブの実行: 以下のグラフに示すように、他のすべてのエグゼキューターはアイドル状態であり、最終的に時間 10:09 までに破棄されます。この時点で、エグゼキューターの合計数はわずか 1 に減ります。これは、エグゼキューター 3 番が、実行時間が最も長いストラグラータスクで構成されているため、ジョブの実行時間の大部分を占めていることを明確に示しています。



メモリプロファイル: 最初の 2 つのステージの後、エグゼキューター 3 番のみがメモリをアクティブに消費してデータを処理します。残りのエグゼキューターはアイドル状態のままであるか、最初の 2 つのステージの完了後すぐに破棄されます。



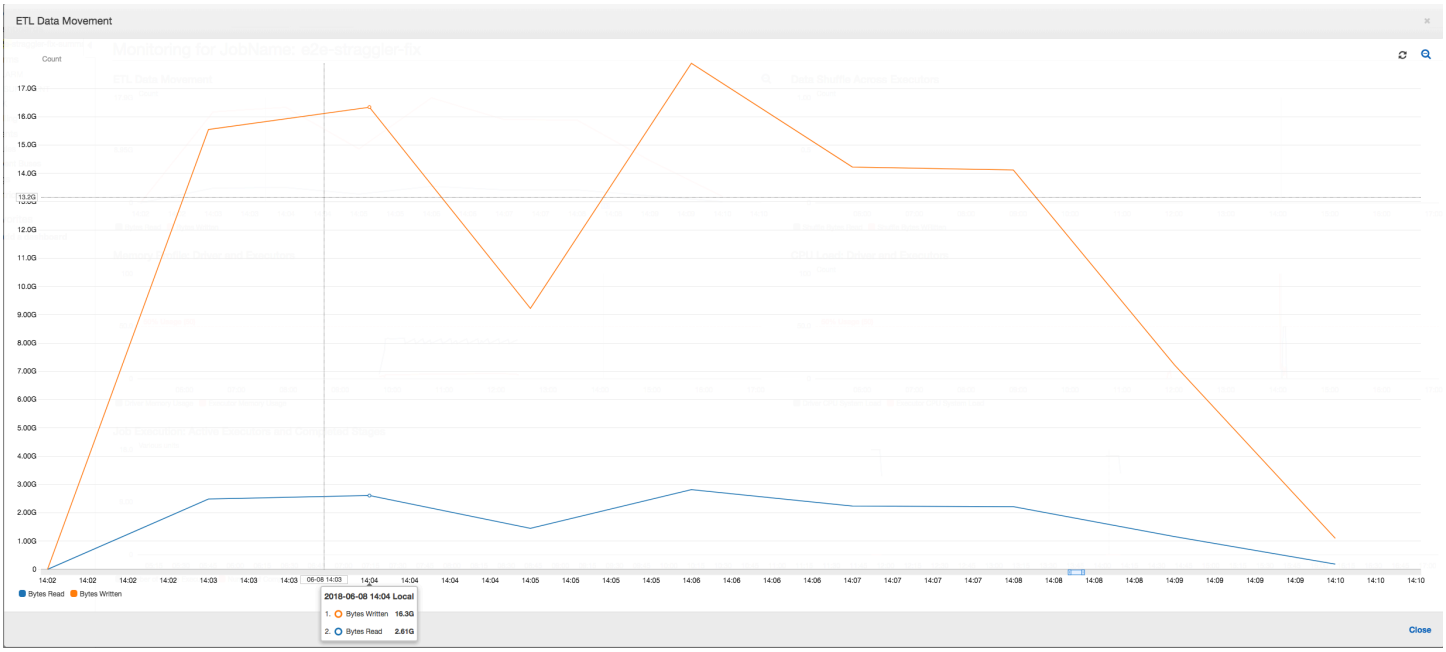
グループ化を使用してストラグラーエグゼキューターを修正する

ストラグラーエグゼキューターは、 でグループ化AWS Glueを使用して回避できます。グループ化を使用してデータをすべてのエグゼキューター間に均等に分散し、クラスター内で利用可能なすべてのエグゼキューターを使用して大きいファイルにまとめます。詳細については、 を参照してください。大きなグループの入力ファイルの読み取り

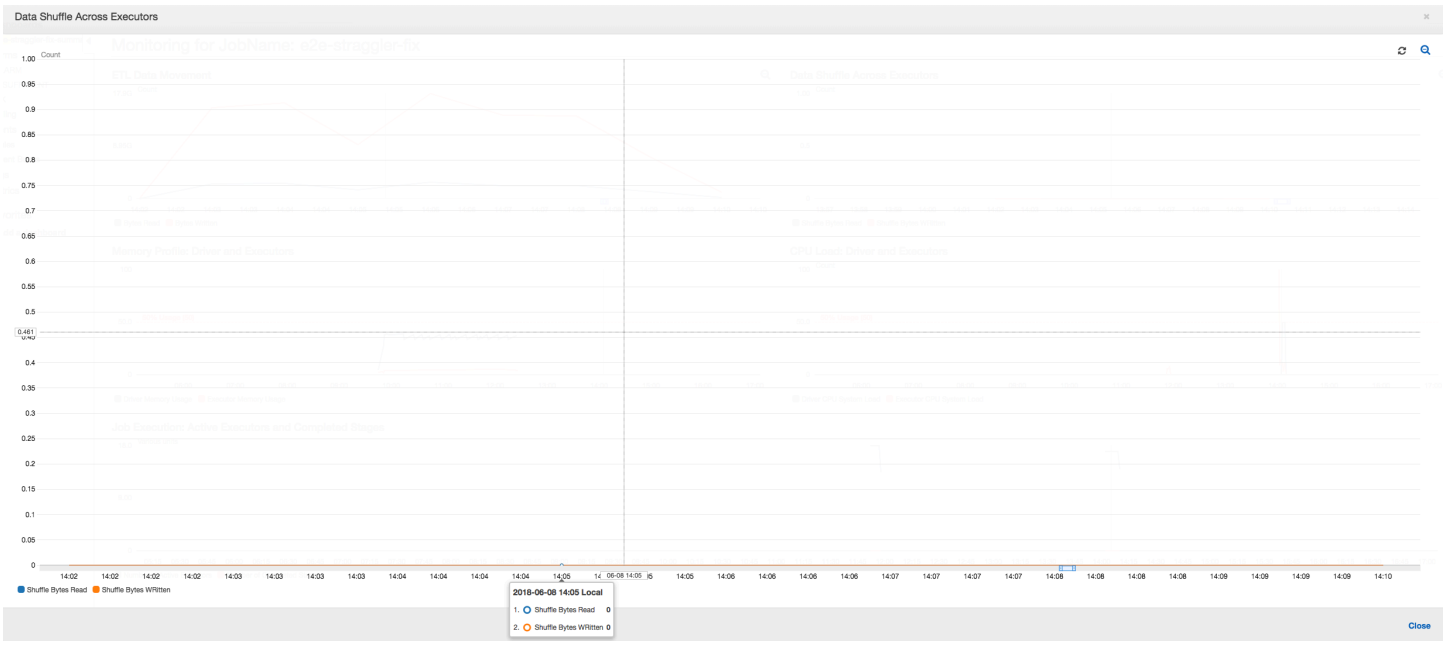
AWS Glue ジョブで ETL データ移動を確認するには、グループ化を有効にして次のコードをプロファイルします。

```
df = glueContext.create_dynamic_frame_from_options("s3", {'paths': ["s3://input_path"],
 "recurse":True, 'groupFiles': 'inPartition'}, format="json")
datasink = glueContext.write_dynamic_frame.from_options(frame = df, connection_type =
 "s3", connection_options = {"path": output_path}, format = "json", transformation_ctx
 = "datasink4")
```

ETL データ移動: データの書き込みは、ジョブの実行時間全体にわたって、データの読み取りと並行してストリーミングされるようになりました。その結果、ジョブは 8 分以内に終了します。以前よりもはるかに早いスピードです。



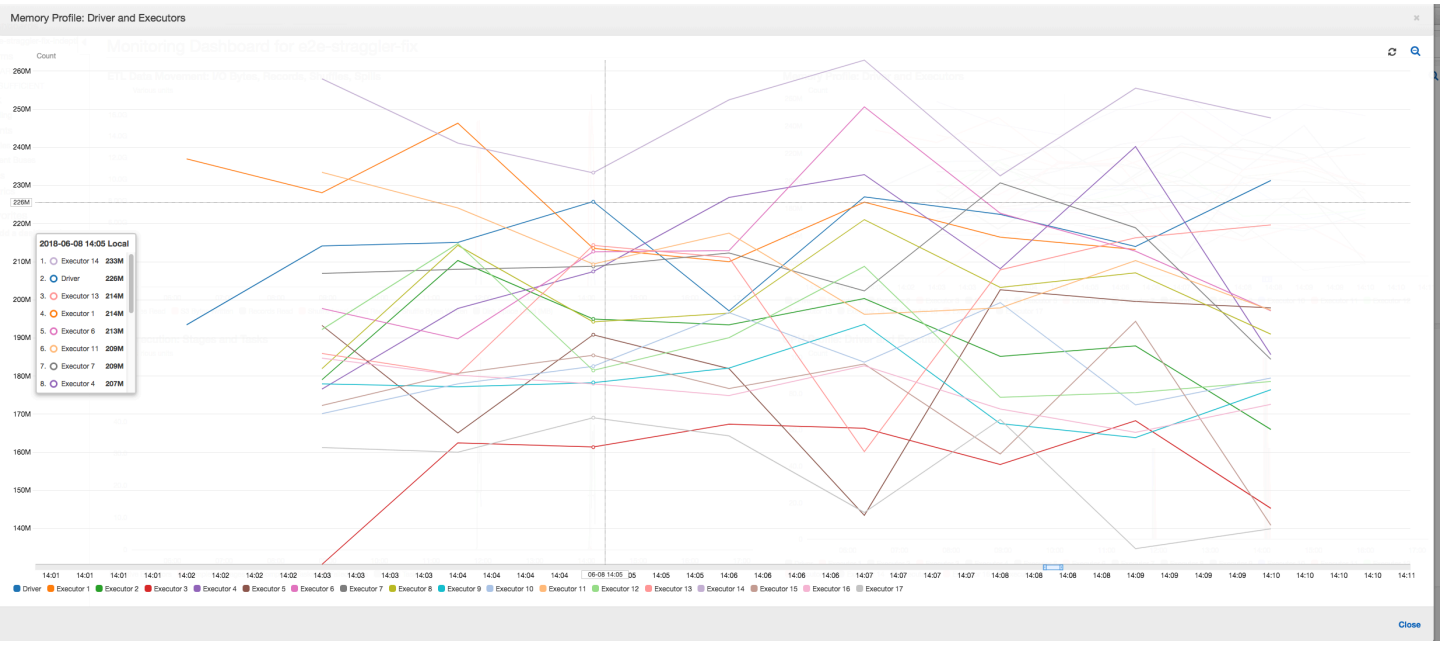
エグゼキューター間でのデータシャッフル: グループ化機能を使用して読み取り中に入力ファイルがまとめられると、データの読み取り後、コストのかかるデータシャッフルが行われなくなります。



ジョブの実行: ジョブ実行メトリクスは、実行されてデータを処理しているアクティブなエグゼキューターの合計数がかなり安定していることを示しています。ジョブに単一のストラグラーはありません。すべてのエグゼキューターがアクティブで、ジョブが完了するまでは破棄されません。エグゼキューター間でデータの中間シャッフルは行われないため、ジョブにはステージが 1 つしかありません。



メモリプロファイル: このメトリクスは、すべてのエグゼキュター間のアクティブなメモリ消費を示しています。これは、すべてのエグゼキュター間にアクティビティがあることを再確認するものです。データは並行してストリーミングおよび書き出されるため、すべてのエグゼキュターの合計メモリ使用量はほぼ均等で、すべてのエグゼキュター安全しきい値をかなり下回っています。



複数のジョブの進行状況のモニタリング

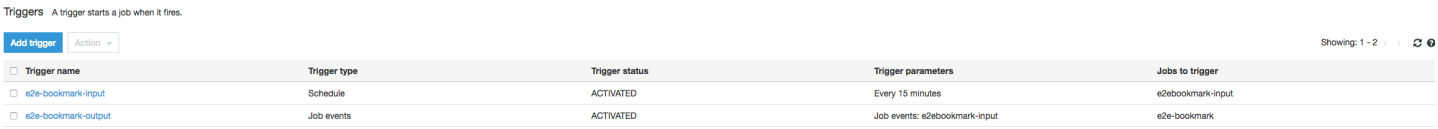
複数の AWS Glue ジョブをまとめてプロファイルし、それらの間のデータフローを監視できます。これは、一般的なワークフローパターンであり、個々のジョブの進行状況、データ処理バックログ、データの再処理、ジョブのブックマークのモニタリングが必要です。

トピック

- プロファイルされたコード
- AWS Glue コンソールでプロファイルされたメトリクスを可視化する
- ファイルの処理を修正する

プロファイルされたコード

このワークフローには、入力ジョブと出力ジョブの 2 つがあります。入力ジョブは、定期的なトリガーを使用して 30 分ごとに実行するようにスケジュールされます。出力ジョブは、入力ジョブの実行が成功するたびに実行されるようにスケジュールされます。このようなスケジュールされたジョブは、ジョブトリガーを使用して制御されます。



入力ジョブ: このジョブは Amazon Simple Storage Service (Amazon S3) の場所からデータを読み取り、ApplyMapping を使用して変換し、Amazon S3 のステージング場所に書き込みます。次のコードは、入力ジョブのプロファイルされたコードです。

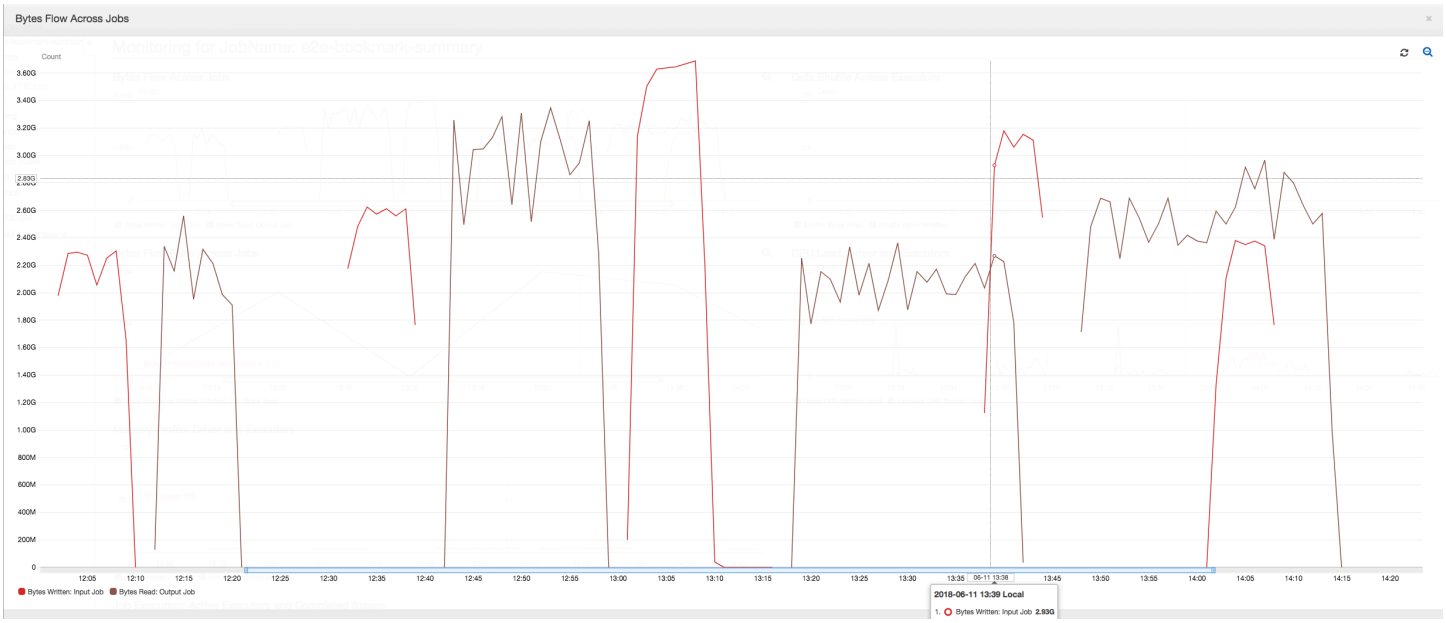
```
datasource0 = glueContext.create_dynamic_frame.from_options(connection_type="s3",
 connection_options = {"paths": ["s3://input_path"],
 "useS3ListImplementation":True,"recurse":True}, format="json")
applymapping1 = ApplyMapping.apply(frame = datasource0, mappings = [map_spec])
datasink2 = glueContext.write_dynamic_frame.from_options(frame = applymapping1,
 connection_type = "s3", connection_options = {"path": staging_path, "compression":
 "gzip"}, format = "json")
```

出力ジョブ: このジョブは、Amazon S3 内のステージング場所から入力ジョブの出力を読み取り、もう一度変換して、ターゲットの場所に書き込みます。

```
datasource0 = glueContext.create_dynamic_frame.from_options(connection_type="s3",
 connection_options = {"paths": [staging_path],
 "useS3ListImplementation":True,"recurse":True}, format="json")
applymapping1 = ApplyMapping.apply(frame = datasource0, mappings = [map_spec])
datasink2 = glueContext.write_dynamic_frame.from_options(frame = applymapping1,
 connection_type = "s3", connection_options = {"path": output_path}, format = "json")
```

AWS Glue コンソールでプロファイルされたメトリクスを可視化する

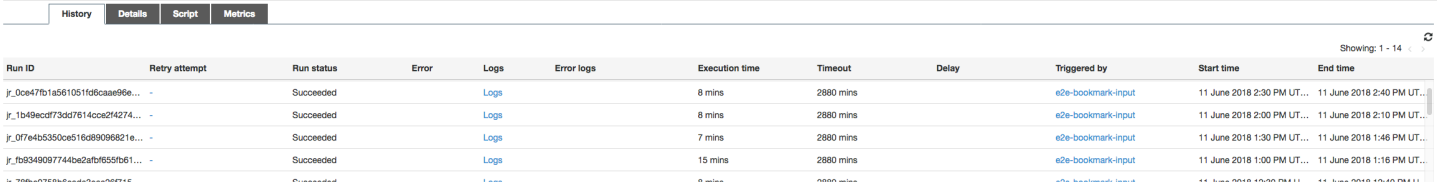
次のダッシュボードでは、入力ジョブから取得された Amazon S3 バイトの書き込みメトリクスを、出力ジョブの同じタイムラインにある Amazon S3 バイトの読み取りメトリクスに重ね合わせています。タイムラインには、入力および出力ジョブのさまざまなジョブの実行が表示されます。入力ジョブ (赤で表示) は、30 分ごとに開始されます。出力ジョブ (茶色で表示) は、入力ジョブの完了時に開始し、最大同時実行数は 1 です。



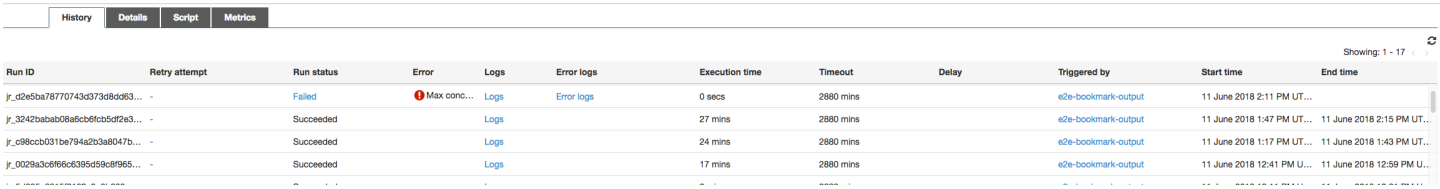
この例では、ジョブのブックマークが有効になっていません。スクリプトコードでジョブのブックマークを有効にするために使用される変換コンテキストはありません。

ジョブ履歴: [履歴] タブに示されているように、入力および出力ジョブには、午後 12 時に始まる複数の実行があります。

AWS Glue コンソールで入力ジョブは次のようになります。

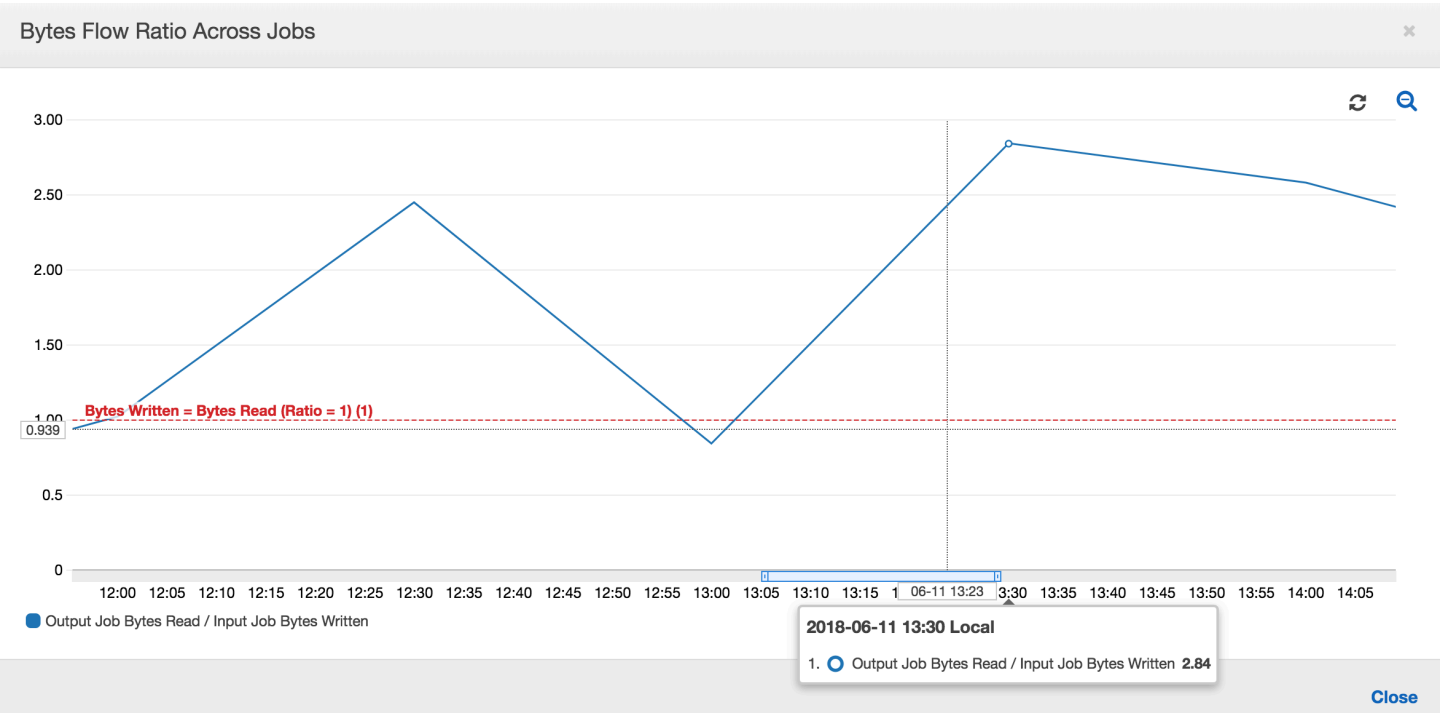


以下のイメージに、出力ジョブを示します。



最初のジョブの実行: 以下のデータバイトの読み書きグラフに示されているように、12:00 から 12:30 の間に行われた入力および出力ジョブの最初のジョブの実行は、曲線の下のほぼ同じ領域を示しています。これらの領域は、入力ジョブによって書き込まれた Amazon S3 バイトと、出力ジョブによって読み取られた Amazon S3 バイトを表しています。このデータは、書き込まれた Amazon S3 バイトの比率によっても確認されます (30 分間の累計 – 入力ジョブのジョブトリガー頻度)。午後 12 時に開始された入力ジョブの実行の比率のデータポイントも 1 です。

以下のグラフは、すべてのジョブの実行におけるデータフロー比率を示しています。



2 回目のジョブの実行: 2 回目のジョブの実行では、入力ジョブによって書き込まれるバイト数と比較して、出力ジョブにより読み取られるバイト数に明確な違いがあります。(出力ジョブの 2 つのジョブの実行にまたがる曲線の下の領域を比較するか、入力および出力ジョブの 2 回目の実行の領域を比較します)。読み取りバイトと書き込みバイトの比率は、12:30 から 13:00 までの 2 番目の 30 分スパンにおいて入力ジョブによって書き込まれるデータの約 2.5 倍のデータを出力ジョブが読み取ることを示しています。ジョブのブックマークが有効になっていなかったため、出力ジョブが入力

上部による最初のジョブの実行の出力を再処理したことがその理由です。1 より大きい比率は、出力ジョブにより処理された追加のデータバックログがあることを示しています。

3 回目のジョブの実行: 入力ジョブは、書き込まれるバイト数の観点でほぼ一定です (赤色の曲線の下にある領域を参照)。ただし、入力ジョブの 3 回目の実行には、予想されるよりも時間がかかります (赤色の曲線が長く続いていることを確認)。その結果、出力ジョブの 3 回目のジョブの実行が遅れて開始します。3 回目のジョブの実行では、13:00 から 13:30 までの残りの 30 分においてステージング場所に蓄積されたデータのごく一部のみ処理されました。バイトフローの比率は、入力ジョブの 3 回目の実行により書き込まれたデータの 0.83 のみ処理されたことを示しています (13:00 の比率を参照)。

入力ジョブと出力ジョブの重複: 入力ジョブの 4 回目のジョブの実行は、スケジュールに従って 13:30 に開始されました。これは、出力ジョブの 3 回目のジョブの実行が終了する前です。これらの 2 つのジョブの実行は一部の重複しています。ただし、出力ジョブの 3 回目の実行では、13:17 前後に開始されたとき、Amazon S3 のステージング場所にリストされたファイルのみキャプチャされます。これは、入力ジョブの最初のジョブの実行から取得されたすべてのデータ出力で構成されています。13:30 時点での実際の比率は約 2.75 です。出力ジョブの 3 回目のジョブの実行では、13:30 から 14:00 に入力ジョブの 4 回目のジョブの実行により書き込まれたデータの約 2.75 倍のデータが処理されました。

これらのイメージが示すように、出力ジョブは、入力ジョブのそれ以前のすべてのジョブの実行から取得されたステージング場所からのデータを再処理します。その結果、出力ジョブの 4 番目のジョブの実行が最も長くなり、入力ジョブの 5 回目のジョブの実行全体と重複します。

ファイルの処理を修正する

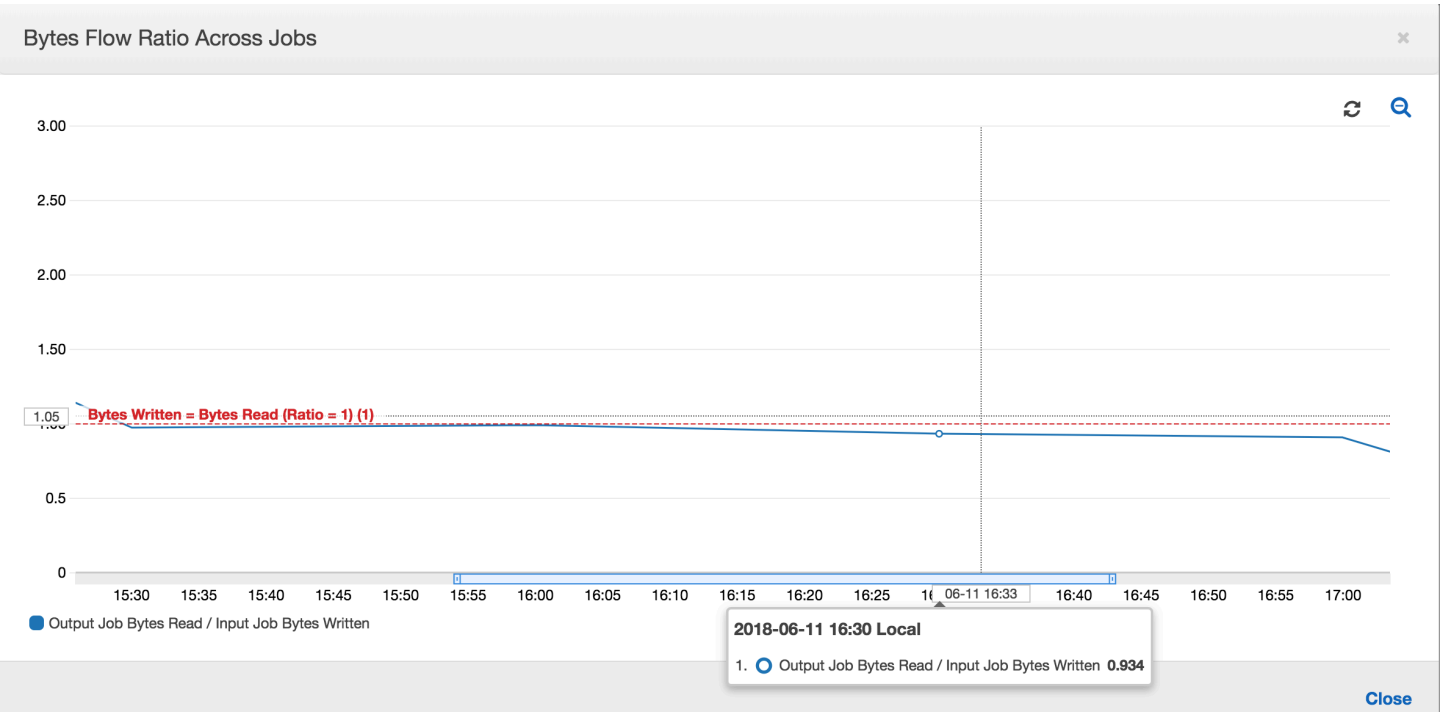
出力ジョブが、出力ジョブの以前のジョブの実行で処理されていないファイルのみ処理するようにする必要があります。これを行うには、次のように、ジョブのブックマークを有効にし、ジョブ出力に変換コンテキストを設定します。

```
datasource0 = glueContext.create_dynamic_frame.from_options(connection_type="s3",
 connection_options = {"paths": [staging_path],
 "useS3ListImplementation":True,"recurse":True}, format="json", transformation_ctx =
 "bookmark_ctx")
```

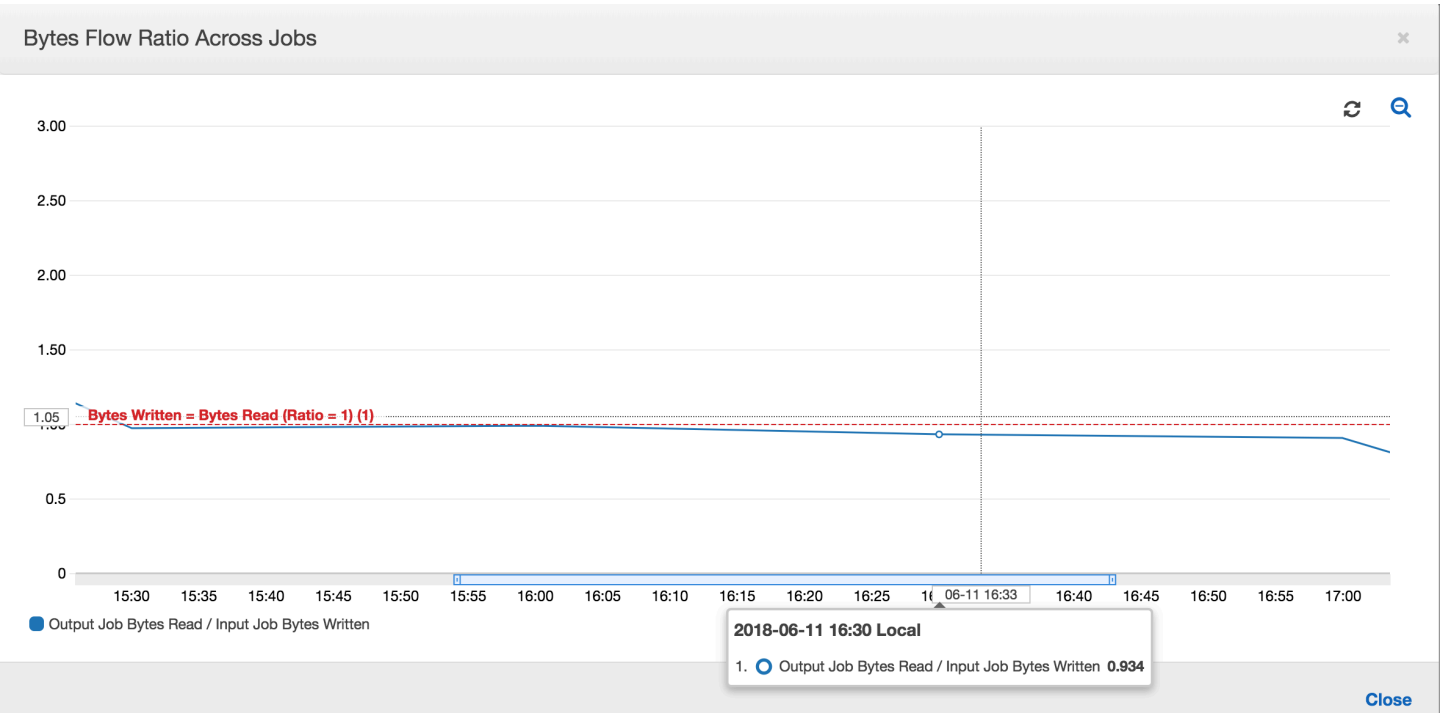
ジョブのブックマークが有効な場合、出力ジョブは、入力ジョブのそれまでのすべてのジョブの実行から取得されたステージング場所のデータを再処理しません。読み書きされたデータを示す以下のイメージでは、茶色の曲線の下にある領域はほぼ一定であり、赤色の曲線と似ています。



追加のデータが処理されないため、バイトフローの比率もほぼ 1 に維持されます。



出力ジョブのジョブの実行が開始され、次の入力ジョブの実行がデータをさらにステージング場所に置き始める前にステージング場所のファイルをキャプチャします。これを続けている限り、それまでの入力ジョブの実行からキャプチャされたファイルのみ処理され、比率はほぼ 1 に維持されます。



入力ジョブに予想以上に時間がかかるため、2 回の入力ジョブの実行から取得されたステージング場所のファイルが出力ジョブによってキャプチャされるとします。その出力ジョブの実行では、比率が 1 よりも大きくなります。ただし、出力ジョブのそれ以降のジョブの実行では、出力ジョブのそれまでのジョブの実行により既に処理されているファイルは処理されません。

DPU の容量計画のモニタリング

AWS Glue でジョブメトリクスを使用すると、AWS Glue ジョブをスケールアウトするために使用できるデータ処理単位 (DPU) の数を予測できます。

> **Note**
>
> このページは AWS Glue バージョン 0.9 および 1.0 にのみ適用できます。AWS Glue 以降のバージョンには、容量計画時に追加の考慮事項を導入するコスト削減機能が含まれています。

トピック

- プロファイルされたコード
- AWS Glue コンソールでプロファイルされたメトリクスを可視化する
- 最適な DPU 容量を決定する

プロファイルされたコード

次のスクリプトは、428 個の gzip で圧縮された JSON ファイルを含む Amazon Simple Storage Service (Amazon S3) パーティションを読み取ります。このスクリプトは、マッピングを適用してフィールド名を変更し、Apache Parquet 形式に変換して Amazon S3 に書き込みます。デフォルトに従って 10 個の DPU をプロビジョニングし、このジョブを実行します。

```
datasource0 = glueContext.create_dynamic_frame.from_options(connection_type="s3",
 connection_options = {"paths": [input_path],
 "useS3ListImplementation":True,"recurse":True}, format="json")
applymapping1 = ApplyMapping.apply(frame = datasource0, mappings = [(map_spec])
datasink2 = glueContext.write_dynamic_frame.from_options(frame = applymapping1,
 connection_type = "s3", connection_options = {"path": output_path}, format =
 "parquet")
```

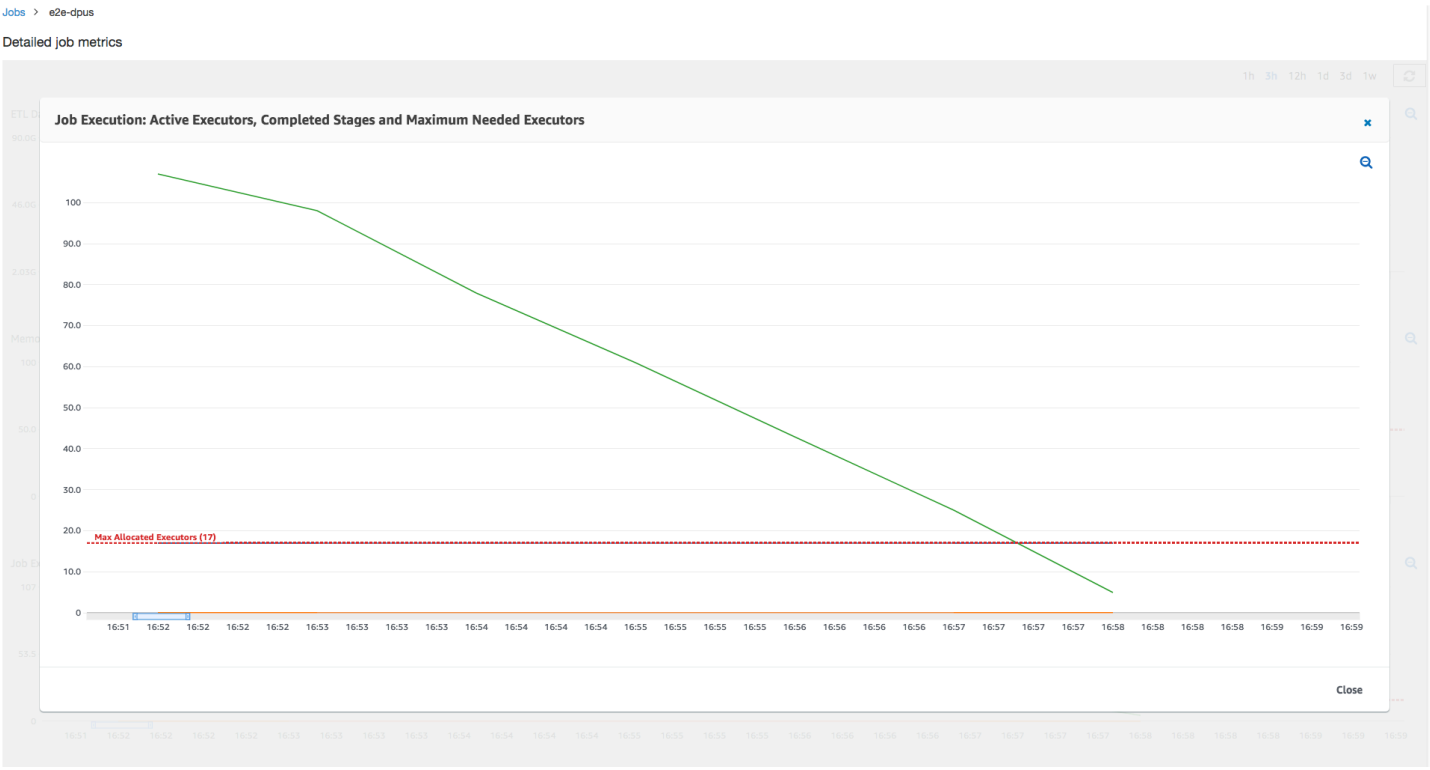
AWS Glue コンソールでプロファイルされたメトリクスを可視化する

ジョブの実行 1: このジョブの実行では、プロビジョニングが不足している DPU がクラスターにあるかどうかを調べる方法を示します。AWS Glue でのジョブ実行機能は、アクティブに実行されているエグゼキューターの合計数、完了したステージの数、必要なエグゼキューターの最大数を表示します。

必要なエグゼキューターの最大数は、実行中のタスクと保留中のタスクの合計数を加算し、エグゼキューターごとのタスクで除算することによって算出されます。この結果は、現在の負荷に対応するために必要なエグゼキューターの合計数の尺度となります。

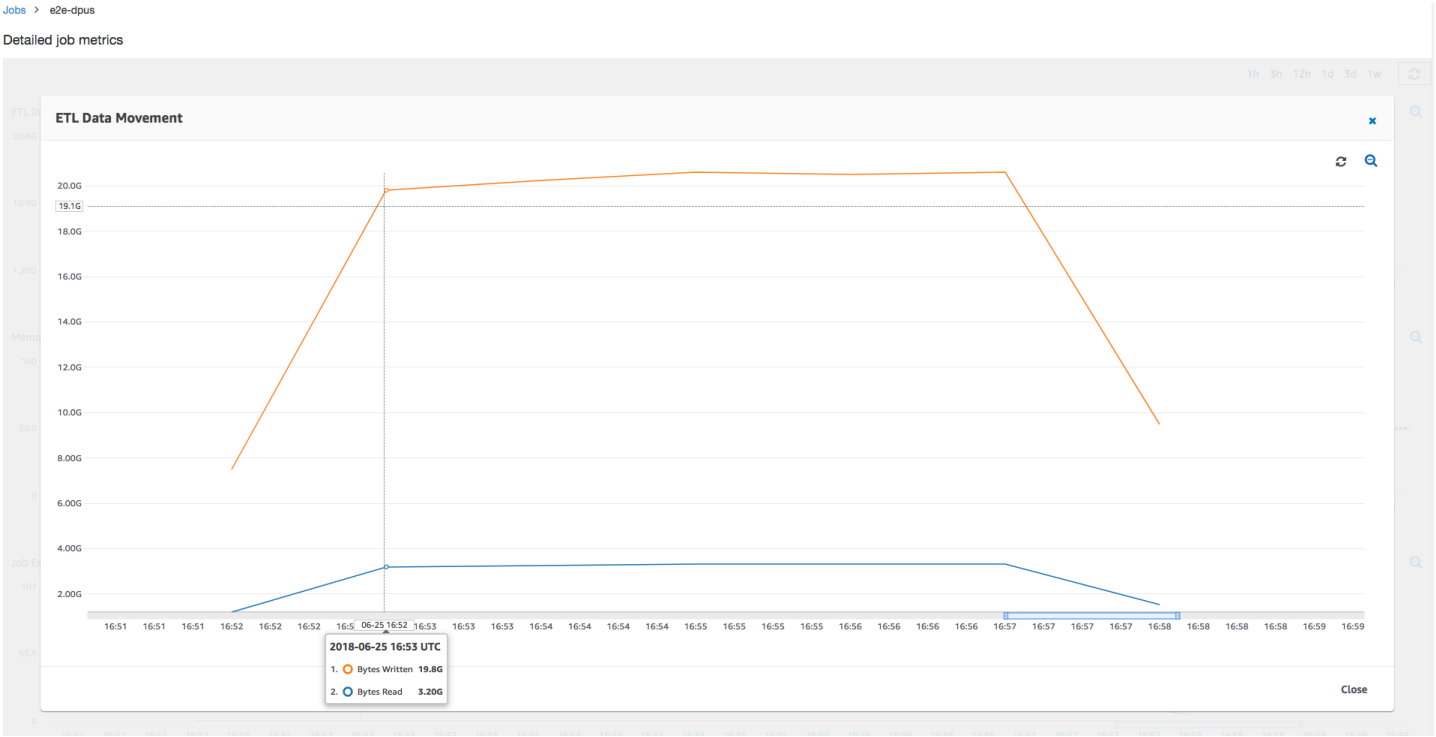
一方、アクティブに実行されているエグゼキューターの数は、アクティブな Apache Spark タスクで実行されているエグゼキューターの数を測定します。ジョブが進行するにつれて、必要なエグゼキューターの最大数は変化し、保留中のタスクキューが減るため通常はジョブの終わりに向かって減少します。

次のグラフの横方向の赤色の線は、割り当てられたエグゼキューターの最大数を示しています。これは、ジョブに割り当てる DPU の数によって異なります。この場合、ジョブの実行に対して 10 個の DPU を割り当てます。1 つの DPU が管理用に予約されています。9 個の DPU はそれぞれ 2 つのエグゼキューターを実行し、1 つのエグゼキューターは Spark ドライバー用に予約されています。Spark ドライバーはプライマリアプリケーション内で実行されます。そのため、割り当てられるエグゼキューターの最大数は、2*9 - 1 = 17 です。



グラフが示すとおり、必要なエグゼキューターの最大数はジョブの開始時に 107 から始まりますが、アクティブなエグゼキューターの数は 17 のままです。これは、10 個の DPU を持つ割り当てられるエグゼキューターの最大数と同じです。必要なエグゼキューターの最大数と割り当てられるエグゼキューターの最大数の比率 (Spark ドライバーでは両方に 1 を加算) から、プロビジョニングが不足している係数が 108/18 = 6 倍であるとわかります。6 (プロビジョニング率未満) *9 (現在の DPU 容量 -1) +1 DPU = 55 個の DPU をプロビジョニングしてジョブをスケールアウトし、最大限の並列処理で実行することで処理時間を短縮することができます。

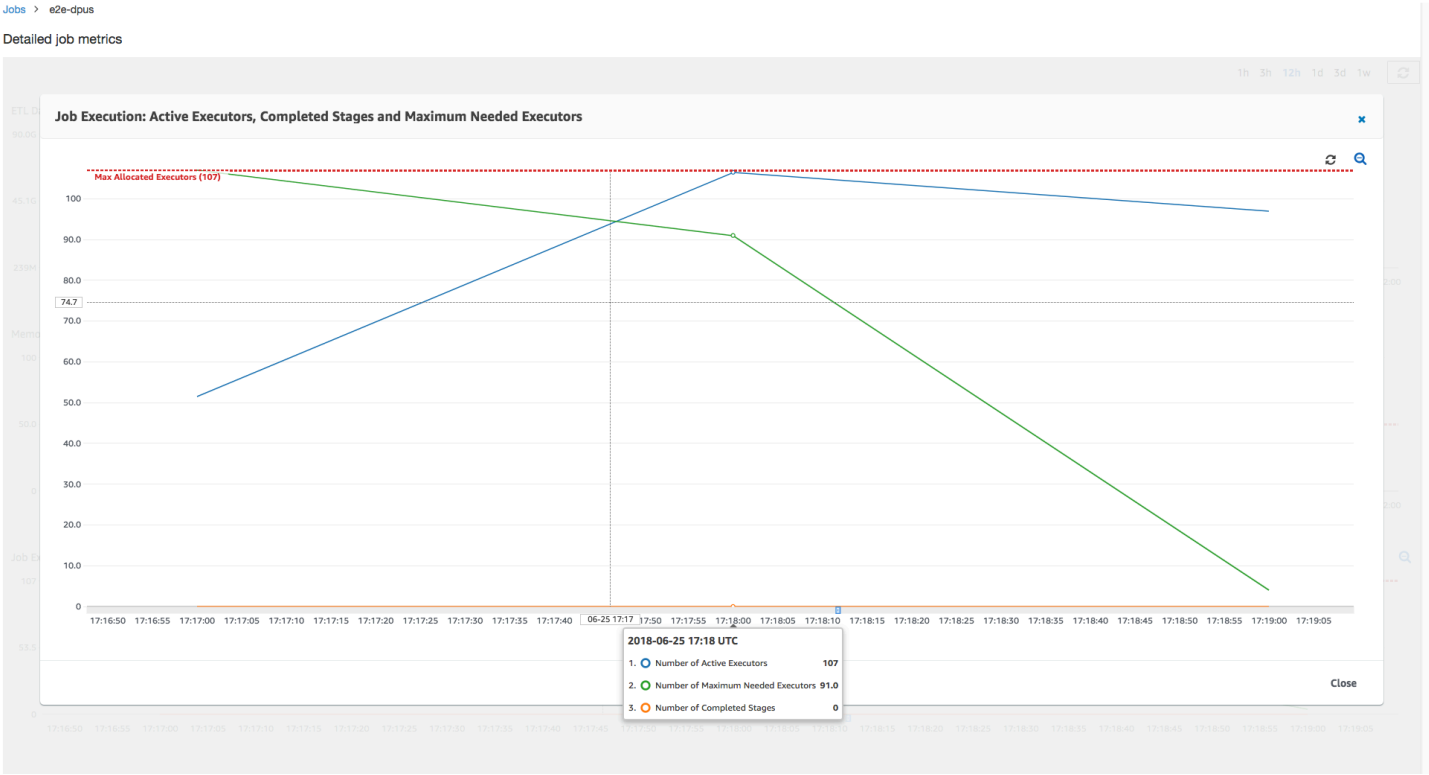
AWS Glue コンソールは、詳細なジョブメトリクスを、元の割り当てられるエグゼキューターの最大数を表す静的な線として表示します。コンソールは、メトリクスのジョブ定義からから割り当てられるエグゼキューターの最大数を計算します。対照的に、詳細なジョブ実行メトリクスについては、コンソールはジョブ実行設定から割り当てられるエグゼキューターの最大数、特にジョブ実行に割り当てられた DPU を計算します。個々のジョブ実行のメトリクスを表示するには、ジョブ実行を選択して、[実行メトリクスの表示] を選択します。



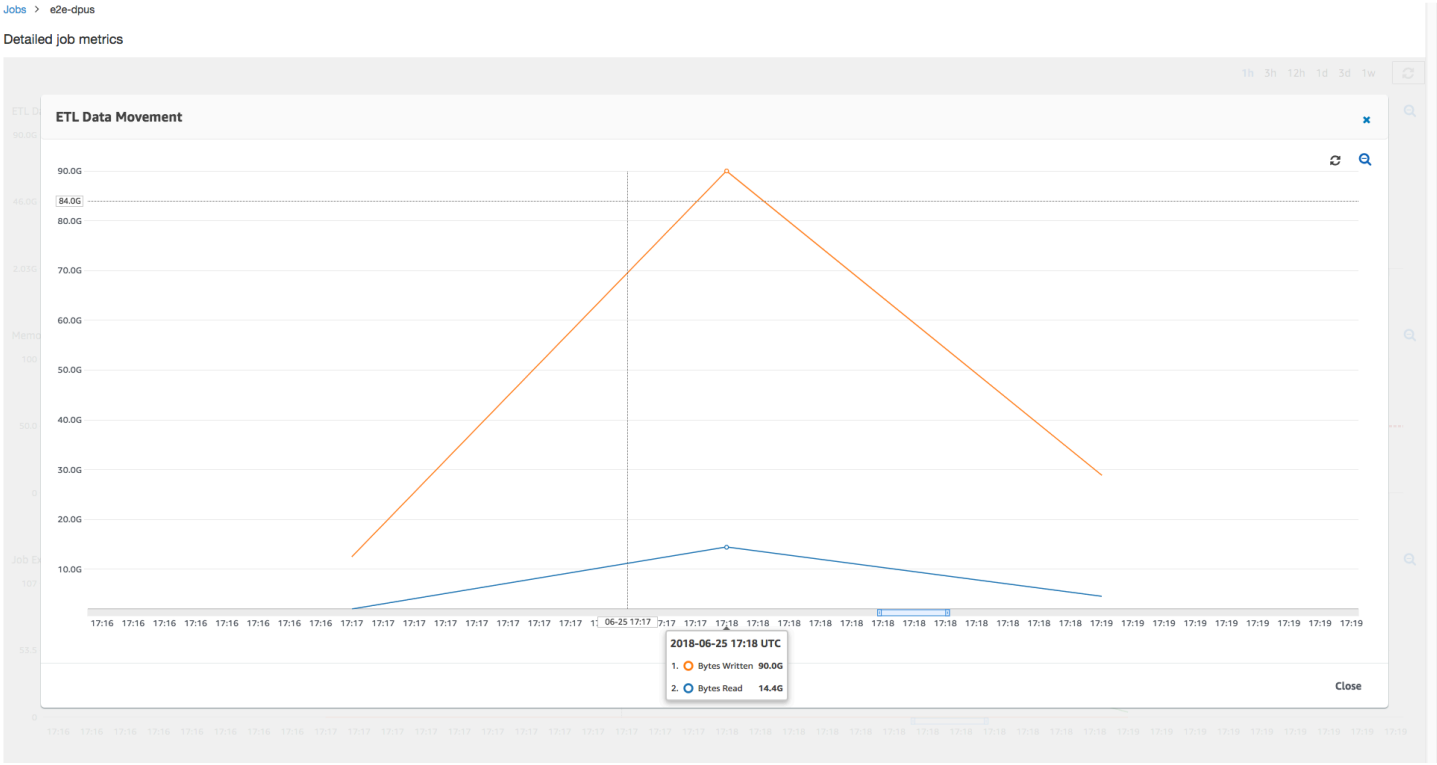
読み取りおよび書き込みされた Amazon S3 バイトを見ると、ジョブが Amazon S3 からのデータをストリーミングして並列に書き出すことに 6 分すべてを消費していることがわかります。割り当てられた DPU 上のすべてのコアは、Amazon S3 に対して読み取りと書き込みを行います。必要なエグゼキューターの最大数 107 は、入力 Amazon S3 パス内のファイル数 428 とも整合しています。各エグゼキューターは、4 つの Spark タスクを起動し、4 つの入力ファイルを処理します (gzip で圧縮された JSON)。

最適な DPU 容量を決定する

前のジョブの実行の結果に基づいて、割り当てられる DPU の合計数を 55 に増やし、ジョブがどのように実行されるかを確認できます。ジョブは 3 分未満で完了します。以前に必要な時間の半分です。この場合、実行時間の短いジョブであるため、ジョブのスケールアウトは直線的ではありません。存続期間の長いタスクまたは多数のタスクを持つジョブ (必要なエグゼキューターの最大数が大きい) は、直線的に近い DPU スケールアウトのパフォーマンス向上から恩恵を受けます。



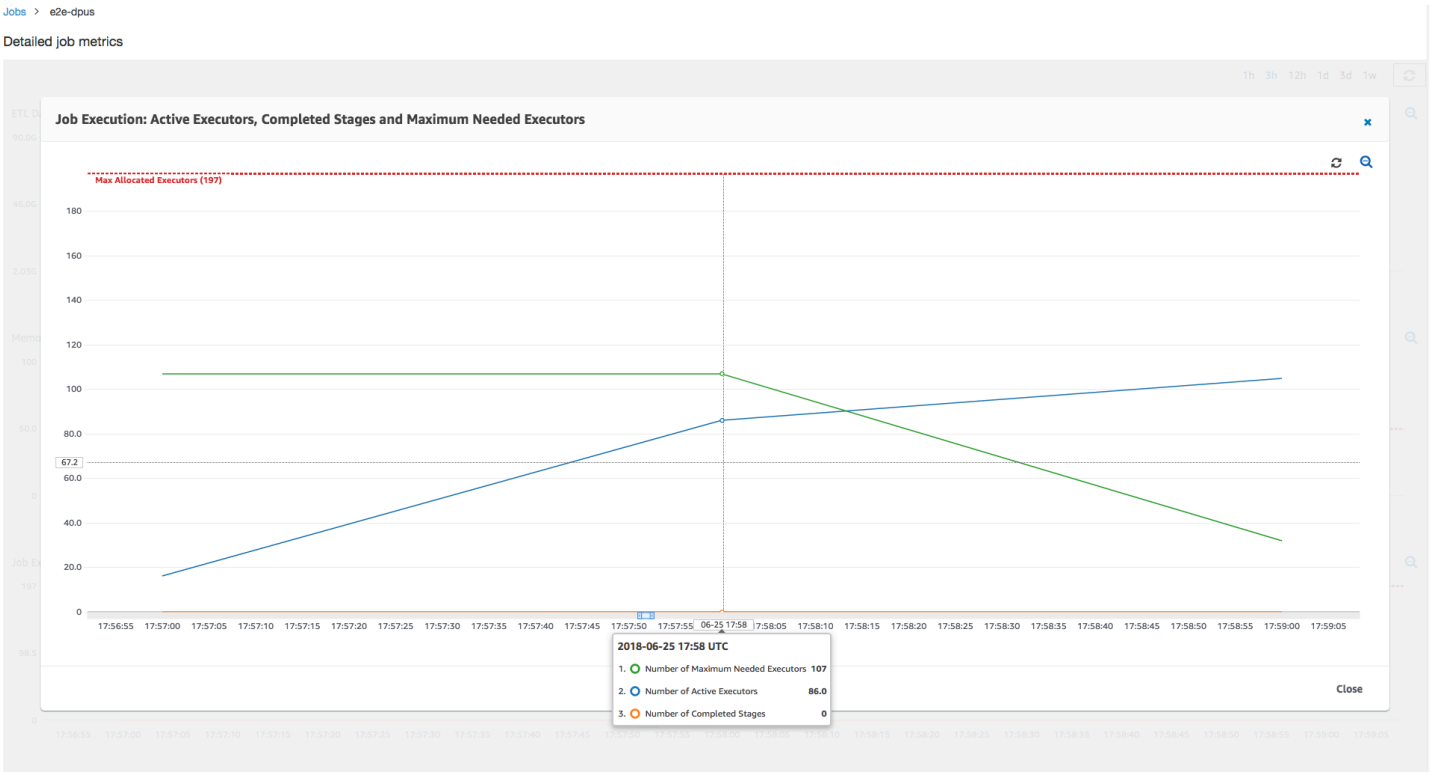
上の図に示すように、アクティブなエグゼキューターの合計数が最大割り当て数 107 個のエグゼキューターに到達します。同様に、必要なエグゼキューターの最大数が割り当てられるエグゼキューターの最大数を上回ることはありません。必要なエグゼキューターの最大数は、アクティブに実行されているタスクと保留中のタスクの数から計算されるため、アクティブなエグゼキューターの数よりも小さい可能性があります。これは、短期間部分的または完全にアイドル状態になり、まだ停止されていないエグゼキューターが存在する可能性があるためです。



このジョブの実行では、Amazon S3 から並列で読み取りと書き込みをするために 6 倍のエグゼキューターが使用されます。その結果、このジョブの実行では、読み取りと書き込みの両方に使用される Amazon S3 帯域幅が多くなり、早く完了します。

過度にプロビジョニングされた DPU を識別する

次に、100 個の DPU (99 * 2 = 198 個のエグゼキューター) でジョブをスケールアウトするとこれ以上スケールアウトしなくてよくなるかどうかを調べることができます。次のグラフが示すように、ジョブが完了するまでまだ 3 分かかります。同様に、ジョブは 107 個を超えてスケールアウトしないため (55 DPU 構成)、残りの 91 個のエグゼキューターは過剰にプロビジョニングされてまったく使用されません。必要なエグゼキューターの最大数からわかるように、これは DPU の数を増やしても必ずパフォーマンスが向上するわけではないことを示しています。



時間の差を比較する

次の表に示す 3 つのジョブが実行は、10 個の DPU、55 個の DPU、100 個の DPU のジョブの実行時間をまとめています。最初のジョブの実行をモニタリングすることで確立した推定値を使用して、ジョブの実行時間を向上させることができる DPU 容量を調べることができます。

ジョブ ID	DPU の数	実行時間
jr_c894524c8ef5048a4d9..。	10	6 分。
jr_1a466cf2575e7ffe6856..。	55	3 分。
jr_34fa1ed4c6aa9ff0a814..。	100	3 分。

AWS Glue での Apache Spark の生成 AI トラブルシューティング

Apache Spark プレビューの生成 AI トラブルシューティングは、AWS Glue 4.0、AWS Glue 5.0、および次の AWS リージョンで実行されているジョブで利用できます。

- 米国東部 (バージニア北部)

- 米国東部 (オハイオ)
- 米国西部 (オレゴン)
- 米国西部 (北カリフォルニア)
- 南米 (サンパウロ)
- カナダ (中部)
- 欧州 (アイルランド)
- 欧州 (ロンドン)
- 欧州 (パリ)
- 欧州 (ストックホルム)
- 欧州 (ミラノ)
- 欧州 (フランクフルト)
- 中東 (バーレーン)
- 中東 (アラブ首長国連邦)
- アフリカ (ケープタウン)
- アジアパシフィック (東京)
- アジアパシフィック (香港)
- アジアパシフィック (ムンバイ)
- アジアパシフィック (シンガポール)
- アジアパシフィック (ジャカルタ)
- アジアパシフィック (ソウル)
- アジアパシフィック (大阪)
- アジアパシフィック (シドニー)

プレビュー機能は変更される可能性があります。

AWS Glue の Apache Spark ジョブ用の生成 AI トラブルシューティングは、データエンジニアやサイエンティストが Spark アプリケーションの問題を簡単に診断および修正するのに役立つ新機能です。この機能は、機械学習と生成 AI テクノロジーを活用して Spark ジョブの問題を分析し、詳細な根本原因分析と、それらの問題を解決するための実用的な推奨事項を提供します。

Apache Spark の生成 AI トラブルシューティングの仕組み

失敗した Spark ジョブの場合、生成 AI のトラブルシューティングはジョブのメタデータと、ジョブのエラー署名に関連する正確なメトリクスとログを分析して根本原因分析を生成し、ジョブの障害に対処するのに役立つ特定のソリューションとベストプラクティスを推奨します。

ジョブ用の Apache Spark の生成 AI トラブルシューティングの設定

IAM 許可の設定

AWS Glue のジョブに対して Spark トラブルシューティングで使用される APIs にアクセス許可を付与するには、適切な IAM アクセス許可が必要です。次のカスタム AWS ポリシーを IAM アイデンティティ (ユーザー、ロール、グループなど) にアタッチすることで、許可を取得できます。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
"Effect": "Allow",
            "Action": [
                "glue:StartCompletion",
                "glue:GetCompletion"
            ],
            "Resource": [
                "arn:aws:glue:*:*:completion/*",
                "arn:aws:glue:*:*:job/*"
            ]
        }
    ]
}
```

> **Note**
>
> プレビュー中、Spark トラブルシューティングでは、AWS SDKを介してプログラムで使用できる API はありません。AWS Glue Studio コンソールを通じてこのエクスペリエンスを実現するための IAM ポリシーでは、`StartCompletion` と `GetCompletion` の 2 つの API が使用されます。

権限の割り当て

アクセスを提供するには、ユーザー、グループ、またはロールにアクセス許可を追加します。

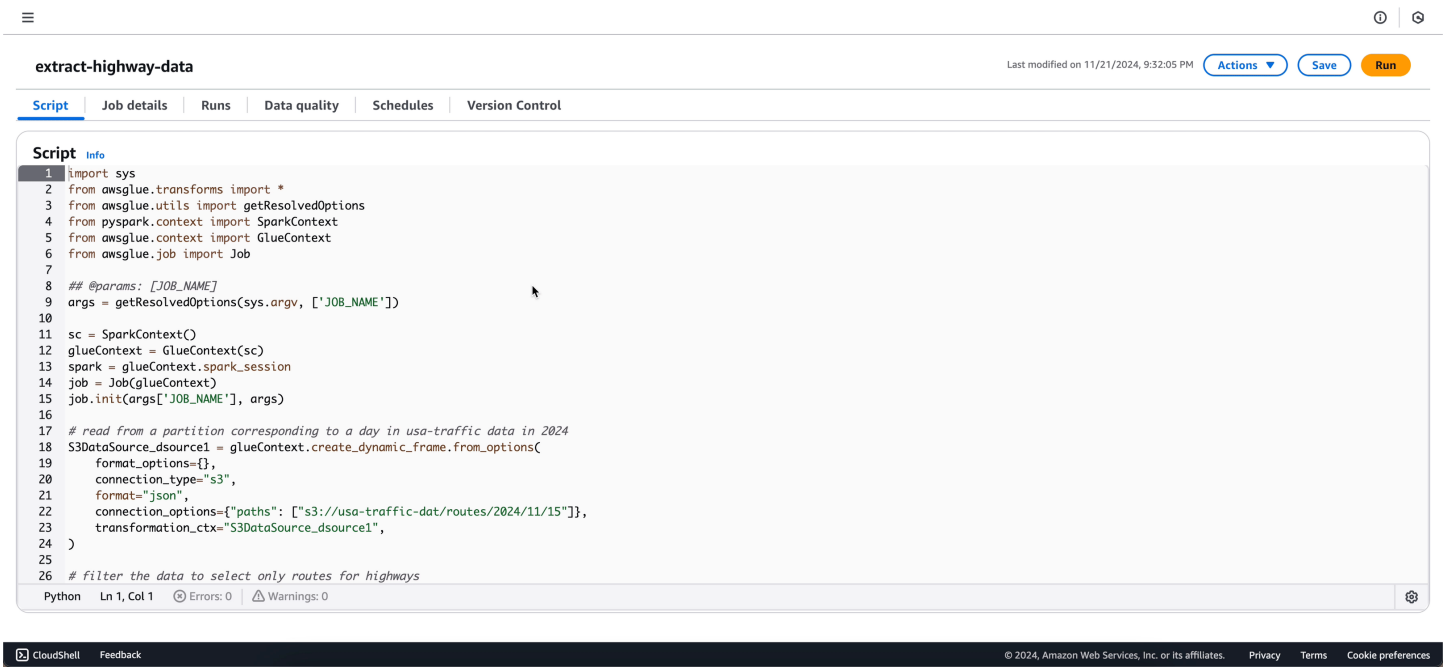
- IAM Identity Center のユーザーおよびグループでは: 許可セットを作成します。「IAM Identity Center のユーザーガイド」の「許可セットの作成」の手順に従ってください。
- アイデンティティプロバイダーを介して IAM で管理されているユーザーでは: ID フェデレーションのロールを作成します。詳細については、「IAM ユーザーガイド」の「サードパーティー ID プロバイダー (フェデレーション) 用のロールの作成」を参照してください。
- IAM ユーザーでは: 引き受けることができるロールを作成します。手順については、「IAM ユーザーガイド」の「IAM ユーザー用ロールの作成」を参照してください。

失敗したジョブ実行からのトラブルシューティング分析の実行

AWS Glue コンソールの複数のパスからトラブルシューティング機能にアクセスできます。開始方法はこちら:

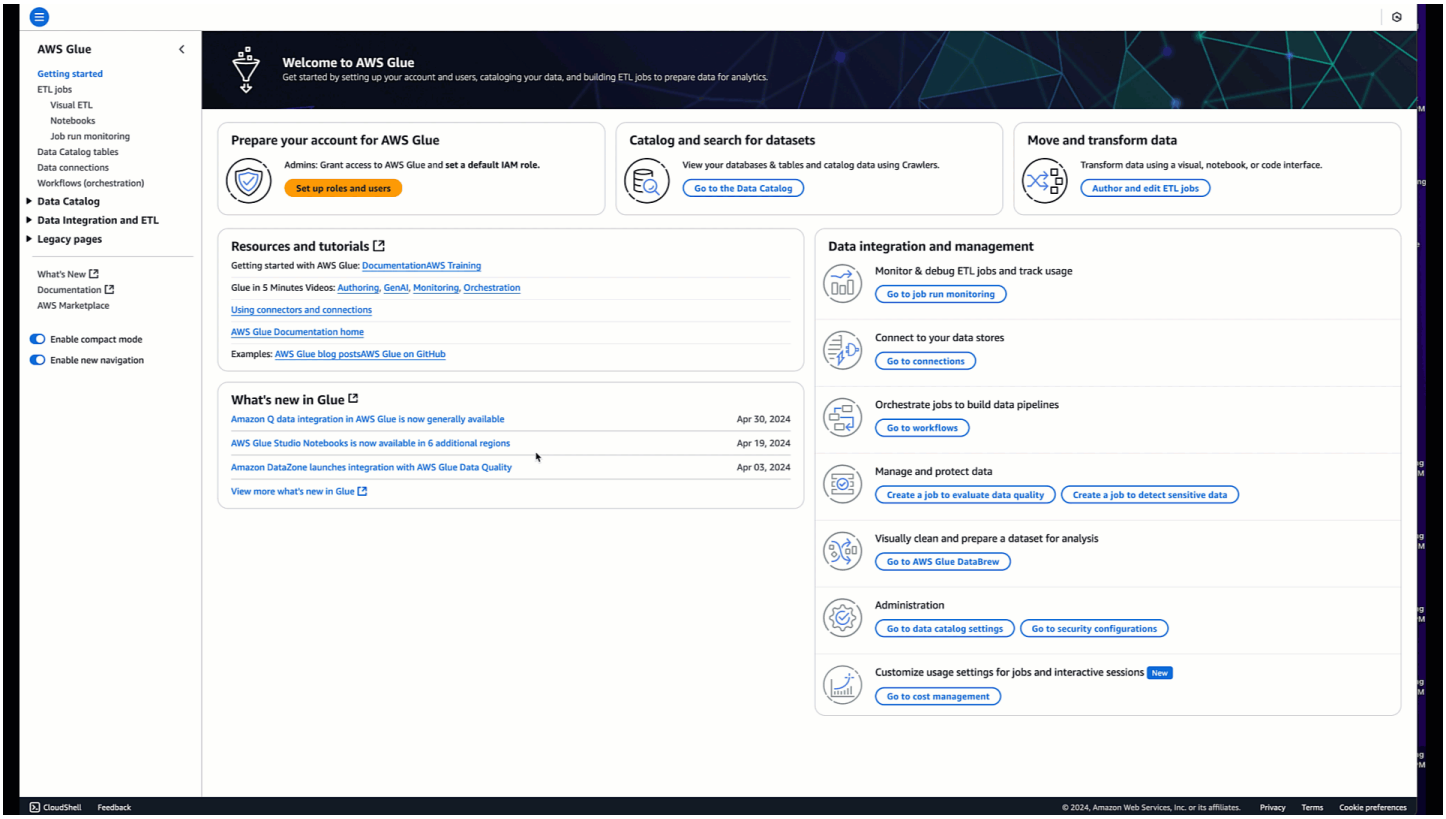
オプション 1: ジョブリストページから

1. AWS Glue コンソール (https://console.aws.amazon.com/glue/) を開きます。
2. ナビゲーションペインで、[ETL ジョブ] を選択します。
3. 失敗したジョブをジョブリストで見つけます。
4. ジョブの詳細セクションで[実行] タブを選択します。
5. 分析する失敗したジョブ実行をクリックします。
6. [AI によるトラブルシューティング] を選択して分析を開始します。
7. トラブルシューティング分析が完了したら、画面の下部にある [トラブルシューティング分析] タブで根本原因分析と推奨事項を表示できます。



オプション 2: [ジョブ実行モニタリング] ページの使用

1. [ジョブ実行のモニタリング] ページに移動します。
2. 失敗したジョブ実行を見つけます。
3. ドロップダウンメニューで [アクション] を選択します。
4. [AI によるトラブルシューティング] を選択します。



オプション 3: [ジョブ実行の詳細] ページから

1. 失敗したジョブ実行の詳細ページに移動するには、[実行] タブから失敗した実行の [詳細を表示] をクリックするか、[ジョブ実行のモニタリング] ページからジョブ実行を選択します。
2. ジョブ実行の詳細ページで、[トラブルシューティング分析] タブを見つけます。

サポートされているトラブルシューティングカテゴリ (プレビュー)

このサービスは、データエンジニアとデベロッパーが Spark アプリケーションで頻繁に遭遇する問題の 3 つの主要なカテゴリに焦点を当てています。

- リソースのセットアップとアクセスエラー: AWS Glue で Spark アプリケーションを実行する場合、リソースのセットアップとアクセスエラーは、診断するのが最も一般的でありながら難しい問題の 1 つです。これらのエラーは、Spark アプリケーションが、AWS リソースとやり取りしようとしても、アクセス許可の問題、リソースの欠落、または設定の問題が発生した場合によく発生します。
- Spark ドライバーとエグゼキューターのメモリの問題: Apache Spark ジョブのメモリ関連のエラーは、診断と解決が複雑なものである場合があります。これらのエラーは、多くの場合、データ処理

要件がドライバーノードまたはエグゼキューターノードで使用可能なメモリリソースを超えたときに発生します。

- Spark ディスク容量の問題: AWS Glue Spark ジョブのストレージ関連のエラーは、シャッフル操作、データスピル、または大規模なデータ変換の処理中に発生することがよくあります。これらのエラーは、ジョブがしばらく実行されるまで表示されない可能性があり、貴重なコンピューティング時間とリソースを浪費する可能性があります。

> **Note**
>
> 本番環境に提案された変更を実装する前に、提案された変更を徹底的に確認してください。このサービスは、パターンとベストプラクティスに基づいてレコメンデーションを提供しますが、特定のユースケースでは追加の考慮事項が必要になる場合があります。

AWS Glue でのストリーミング ETL ジョブ

連続的に実行されるストリーミング抽出/変換/ロード (ETL) ジョブを作成し、Amazon Kinesis Data Streams、Apache Kafka、Amazon Managed Streaming for Apache Kafka (Amazon MSK) などのストリーミングソースからのデータを使用できます。ジョブはデータをクレンジングして変換し、その結果を Amazon S3 データレイクまたは JDBC データストアにロードします。

さらに、Amazon Kinesis Data Streams ストリーム用のデータを生成できます。この機能は AWS Glue スクリプトを作成中のみ使用できます。詳細については、「the section called “Kinesis 接続”」を参照してください。

デフォルトでは、AWS Glue は 100 秒ウィンドウ単位でデータ処理と書き出しを行います。これにより、データを効率的に処理しつつ、想定より遅く到着したデータに対する集計を実行できます。このウィンドウサイズを変更して、タイムリーで高精度の集計にできます。AWS Glue ストリーミングジョブでは、ジョブブックマークではなくチェックポイントを使用して、読み取られたデータを追跡します。

> **Note**
>
> AWS Glue では、実行中の ETL ジョブのストリーミングに対して 1 時間ごとに課金します。

このビデオでは、ストリーミング ETL のコストに関する課題と、AWS Glue のコスト削減機能について説明しています。

ストリーミング ETL ジョブを作成するには、次の手順を実行します。

1. Apache Kafka ストリーミングソースの場合は、Kafka ソースまたは Amazon MSK クラスターへの AWS Glue 接続を作成します。
2. ストリーミングソースの Data Catalog テーブルを手動で作成します。
3. ストリーミングデータソースの ETL ジョブを作成します。ストリーミング固有のジョブプロパティを定義し、独自のスクリプトを指定するか、生成されたスクリプトを必要に応じて変更します。

詳細については、「AWS Glue でのストリーミング ETL」を参照してください。

Amazon Kinesis Data Streams にストリーミング ETL ジョブを作成する場合、AWS Glue 接続を作成する必要はありません。ただし、Kinesis Data Streams をソースとする AWS Glue ストリーミング ETL ジョブへの接続がある場合は、の場合は、Kinesis への Virtual Private Cloud (VPC) エンドポイントが必要です。詳細については、 Amazon VPC ユーザーガイド のインターフェイスエンドポイントの作成を参照してください。別のアカウントで Amazon Kinesis Data Streams ストリームを指定する場合は、クロスアカウントアクセスを許可するようにロールおよびポリシーを設定する必要があります。詳細については、「例: 別のアカウントで Kinesis Stream から読み込む」を参照してください。

AWS Glue ストリーミング ETL ジョブにより、圧縮データの自動検出が行えます。また、ストリーミングデータを透過的に解凍し、入力ソースに対して一般的な変換を実行して、出力ストアにロードすることができます。

AWS Glue は、入力形式に対して指定された、以下の圧縮タイプのための自動解凍をサポートしています。

圧縮タイプ	Avro ファイル	Avro データム	JSON	CSV	Grok
BZIP2	はい	可能	可能	可能	可能
GZIP	いいえ	あり	可能	可能	可能
SNAPPY	はい (raw Snappy)	はい (framed Snappy)	はい (フレーム付 Snappy)	はい (フレーム付 Snappy)	はい (フレーム付 Snappy)
XZ	可能	可能	可能	可能	可能

圧縮タイプ	Avro ファイル	Avro データム	JSON	CSV	Grok
ZSTD	あり	いいえ	いいえ	いいえ	いいえ
DEFLATE	はい	可能	可能	可能	可能

トピック

- Apache Kafka データストリームの AWS Glue 接続の作成
- ストリーミングソースの Data Catalog テーブルの作成
- Avro ストリーミングソースに関する注意事項と制約事項
- ストリーミングソースへの grok パターンの適用
- ストリーミング ETL ジョブのジョブプロパティの定義
- ストリーミング ETL に関する注意と制限

Apache Kafka データストリームの AWS Glue 接続の作成

Apache Kafka ストリームから読み取りを行うには、AWS Glue 接続を作成する必要があります。

Kafka ソース用の AWS Glue 接続を作成するには (コンソール)

1. https://console.aws.amazon.com/glue/ で AWS Glue コンソール を開きます。
2. ナビゲーションペインの [データカタログ] で [接続] を選択します。
3. [接続の追加] を選択し、[接続のプロパティを設定します] ページで接続名を入力します。

   > **Note**
   >
   > 接続プロパティ指定の詳細については、「AWS Glue connection properties.」(の接続プロパティ) を参照してください。

4. [接続タイプ] で、[Kafka] を選択します。
5. Kafka のブートストラップサーバーの URL として、Amazon MSK クラスターまたは Apache Kafka クラスターのブートストラップブローカーのホストとポート番号を入力します。Kafka クラスターへの初期接続を確立するには、Transport Layer Security (TLS) エンドポイントのみを使用します。Plaintext エンドポイントはサポートされていません。

Amazon MSK クラスターのホスト名とポート番号のペアのリスト例を次に示します。

```
myserver1.kafka.us-east-1.amazonaws.com:9094,myserver2.kafka.us-
east-1.amazonaws.com:9094,
myserver3.kafka.us-east-1.amazonaws.com:9094
```

ブートストラップブローカー情報の取得の詳細については、Amazon Managed Streaming for Apache Kafka デベロッパーガイドの「Getting the Bootstrap Brokers for an Amazon MSK Cluster」を参照してください。

6. Kafka データソースへのセキュアな接続が必要な場合は、[Require SSL connection] (SSL 接続が必要) を選択し、[Kafka private CA certificate location] (Kafka　のプライベート CA 証明書の場所) に、カスタム SSL 証明書への有効な Amazon S3 パスを入力します。

   自己管理型 Kafka への SSL 接続の場合、カスタム証明書は必須です。Amazon MSK ではオプションです。

   Kafka のカスタム証明書を指定する方法の詳細については、「the section called “SSL 接続プロパティ”」を参照してください。
7. AWS Glue Studio または AWS CLI を使用して、Kafka クライアントの認証方法を指定します。AWS Glue Studio にアクセスするには、左側のナビゲーションペインの [ETL] メニューから [AWS Glue] を選択します。

   Kafka クライアントの認証方法の詳細については、「クライアント認証用の AWS Glue Kafka 接続プロパティ 」を参照してください。
8. 必要に応じて説明を入力し、[Next] (次へ) をクリックします。
9. Amazon MSK クラスターの場合は、その Virtual Private Cloud (VPC)、サブネット、およびセキュリティグループを指定します。VPC 情報は、自己管理型 Kafka の場合はオプションです。
10. [Next] (次へ) をクリックして、すべての接続プロパティを確認し、[Finish] (完了) をクリックします。

AWS Glue 接続の詳細については、「データへの接続」を参照してください。

クライアント認証用の AWS Glue Kafka 接続プロパティ

SASL/GSSAPI (Kerberos) 認証

この認証方法を選択すると、Kerberos のプロパティを指定できます。

Kerberos のキータブ

キータブファイルの場所を選択します。キータブには、1 つ以上のプリンシパルの長期キーが保存されます。詳細については、「MIT Kerberos ドキュメント: キータブ」を参照してください。

Kerberos の krb5.conf ファイル

krb5.conf ファイルを選択します。これには、デフォルトの領域 (同じ KDC のシステムのグループを定義する論理ネットワークで、ドメインに類似しています) と KDC サーバーの場所が含まれます。詳細については、「MIT Kerberos ドキュメント: krb5.conf」を参照してください。

Kerberos のプリンシパルと Kerberos のサービス名

Kerberos のプリンシパルとサービス名を入力します。詳細については、「MIT Kerberos ドキュメント: Kerberos のプリンシパル」を参照してください。

SASL/SCRAM-SHA-512 認証

この認証方法を選択すると、認証情報を指定することができます。

AWS Secrets Manager

[Search] (検索) ボックスに名前または ARN を入力して、トークンを検索します。

プロバイダのユーザー名とパスワードを直接入力

[Search] (検索) ボックスに名前または ARN を入力して、トークンを検索します。

SSL クライアント認証

この認証方法を選択すると、Amazon S3 を参照することによって Kafka クライアントキーストアの場所を選択できます。オプションで、Kafka クライアントキーストアのパスワードと Kafka クライアントキーのパスワードを入力できます。

IAM 認証

この認証方法は追加の仕様を必要とせず、ストリーミングソースが MSK Kafka の場合にのみ適用されます。

SASL/PLAIN 認証

この認証方法を選択すると、認証情報を指定することができます。

ストリーミングソースの Data Catalog テーブルの作成

データスキーマなど、ソースデータストリームのプロパティを指定するデータカタログテーブルを手動でストリーミングソースに対して作成できます。このテーブルは、ストリーミング ETL ジョブのデータソースとして使用されます。

ソースデータストリーム内のデータのスキーマがわからない場合は、スキーマなしでテーブルを作成できます。次に、ストリーミング ETL ジョブを作成するときに、AWS Glue スキーマ検出機能を有効にできます。AWS Glue では、ストリーミングデータからスキーマを決定します。

テーブルを作成するには、AWS Glue コンソール、AWS Command Line Interface (AWS CLI)、または AWS Glue API を使用します。AWS Glue コンソールを使用して手動でテーブルを作成する方法については、「the section called “テーブルの作成”」を参照してください。

> **Note**
>
> AWS Lake Formation コンソールを使用してテーブルを作成することはできません。AWS Glue コンソールを使用する必要があります。

また、Avro 形式のストリーミングソースや Grok パターンを適用できるログデータについては、次の情報を考慮します。

- the section called “Avro ストリーミングソースに関する注意事項と制約事項”
- the section called “ストリーミングソースへの grok パターンの適用”

トピック

- Kinesis データソース
- Kafka データソース
- AWS Glue スキーマレジストリテーブルのソース

Kinesis データソース

テーブルを作成するときは、次のストリーミング ETL プロパティ (コンソール) を設定します。

ソースの種類

Kinesis

同じアカウント内の Kinesis ソースの場合:

リージョン

Amazon Kinesis Data Streams サービスが存在する AWS リージョン。リージョンと Kinesis ストリーム名は一緒にストリーム ARN に変換されます。

例: https://kinesis.us-east-1.amazonaws.com

Kinesis ストリーム名

Amazon Kinesis Data Streams デベロッパーガイドの「Creating a Stream」に記載されているストリーム名。

別のアカウントの Kinesis ソースについては、この例 を参照して、アカウント間アクセスを許可するようにロールとポリシーを設定します。これらの設定を行います。

ストリーム ARN

コンシューマーを登録する Kinesis データストリームの ARN。詳細については、AWS 全般のリファレンス の「Amazon リソースネーム (ARN) と AWS サービス ネームスペース」を参照してください。

引き受けたロールの ARN

引き受けるロールの Amazon リソースネーム (ARN)。

セッション名 (オプション)

引き受けたロールセッションの識別子。

異なるプリンシパルによって同じロールが引き継がれた場合にセッションを一意に識別するために、またはその他の理由で、ロールセッション名を使用します。クロスアカウントシナリオでは、ロールセッション名が表示され、ロールを所有するアカウントでログ記録できます。引き受けたロールプリンシパルの ARN では、ロールセッション名も使用されます。つまり、一時的なセキュリティ認証情報を使用する後続のクロスアカウント API リクエストでは、ロールセッション名が AWS CloudTrail ログとして外部アカウントに開示されます。

Amazon Kinesis Data Streams のストリーミング ETL プロパティを設定するには (AWS Glue API または AWS CLI)

- 同じアカウント内の Kinesis ソースのストリーミング ETL プロパティを設定するには、`CreateTable` API オペレーションまたは `create_table` CLI コマンドの `StorageDescriptor` データ構造の `streamName` および `endpointUrl` パラメータを指定します。

```
"StorageDescriptor": {
 "Parameters": {
  "typeOfData": "kinesis",
  "streamName": "sample-stream",
  "endpointUrl": "https://kinesis.us-east-1.amazonaws.com"
 }
 ...
}
```

または、streamARN を指定します。

Example

```
"StorageDescriptor": {
 "Parameters": {
  "typeOfData": "kinesis",
  "streamARN": "arn:aws:kinesis:us-east-1:123456789:stream/sample-stream"
 }
 ...
}
```

- 別のアカウントの Kinesis ソースのストリーミング ETL プロパティを設定するには、CreateTable API オペレーションまたは create_table CLI コマンドの StorageDescriptor データ構造の streamARN、awsSTSRoleARN、および awsSTSSessionName (オプション) パラメータを指定します。

```
"StorageDescriptor": {
 "Parameters": {
  "typeOfData": "kinesis",
  "streamARN": "arn:aws:kinesis:us-east-1:123456789:stream/sample-stream",
  "awsSTSRoleARN": "arn:aws:iam::123456789:role/sample-assume-role-arn",
  "awsSTSSessionName": "optional-session"
 }
 ...
}
```

Kafka データソース

テーブルを作成するときは、次のストリーミング ETL プロパティ (コンソール) を設定します。

ソースの種類

Kafka

Kafka ソースの場合:

トピック名

Kafka で指定されたトピック名。

接続

Kafka ソースを参照する AWS Glue 接続 (「the section called “Kafka データストリームの接続の作成”」を参照)。

AWS Glue スキーマレジストリテーブルのソース

AWS Glue スキーマレジストリをストリーミングジョブに使用するには、ユースケース: AWS Glue Data Catalog スキーマレジストリのテーブルを作成または更新する手順に従ってください。

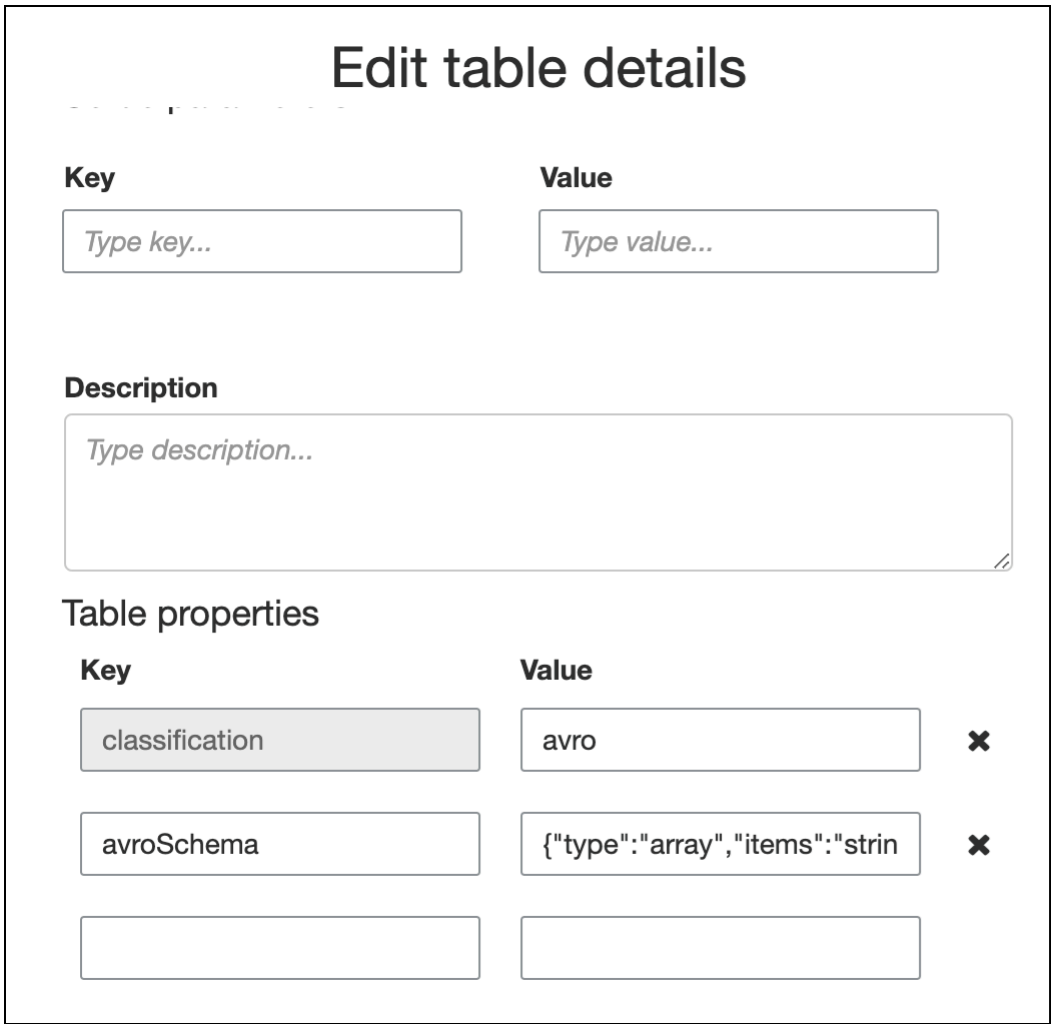
現在、AWS Glue ストリーミングは、`false` にスキーマ推論が設定されたGlue スキーマレジストリ Avro 形式のみをサポートします。

Avro ストリーミングソースに関する注意事項と制約事項

Avro 形式のストリーミングソースには、次の注意事項と制限事項が適用されます。

- スキーマの検出が有効になっている場合、Avro スキーマをペイロードに含める必要があります。無効にすると、ペイロードにはデータのみが含まれます。
- 一部の Avro データ型は、ダイナミックフレームでサポートされていません。AWS Glue コンソールの [create table] (テーブル作成) ウィザードの [Define a schema] (スキーマの定義) でスキーマを定義するときに、これらのデータ型を指定することはできません。スキーマの検出中に、Avro スキーマでサポートされていない型は、次のようにサポートされている型に変換されます。
  - `EnumType => StringType`
  - `FixedType => BinaryType`
  - `UnionType => StructType`
- コンソールの [Define a schema] (スキーマの定義) ページでテーブルスキーマを定義する場合は、スキーマの暗黙のルート要素タイプは `record` です。`record` 以外のルート要素タイプ (`array` や `map` など) が必要な場合、[Define a schema] (スキーマの定義) ページでスキーマを指定することはできません。代わりに、そのページをスキップして、スキーマをテーブルプロパティとして、または ETL スクリプト内で指定する必要があります。

- テーブルのプロパティでスキーマを指定するには、テーブル作成ウィザードを完了し、テーブルの詳細を編集し、[Table properties] (テーブルのプロパティ) に新しいキーと値のペアを追加します。`avroSchema` キーを使用して、次のスクリーンショットに示すように、値のスキーマ JSON オブジェクトを入力します。



- ETL スクリプトでスキーマを指定するには、次の Python と Scala の例に示すように、`datasource0` 代入ステートメントを修正し、`avroSchema` キーを `additional_options` 引数に追加します。

  Python

```
SCHEMA_STRING = '{"type":"array","items":"string"}'
datasource0 = glueContext.create_data_frame.from_catalog(database =
 "database", table_name = "table_name", transformation_ctx = "datasource0",
 additional_options = {"startingPosition": "TRIM_HORIZON", "inferSchema":
 "false", "avroSchema": SCHEMA_STRING})
```

Scala

```
val SCHEMA_STRING = """{"type":"array","items":"string"}"""
val datasource0 = glueContext.getCatalogSource(database = "database", tableName
 = "table_name", redshiftTmpDir = "", transformationContext = "datasource0",
 additionalOptions = JsonOptions(s"""{"startingPosition": "TRIM_HORIZON",
 "inferSchema": "false", "avroSchema":"$SCHEMA_STRING"}""")).getDataFrame()
```

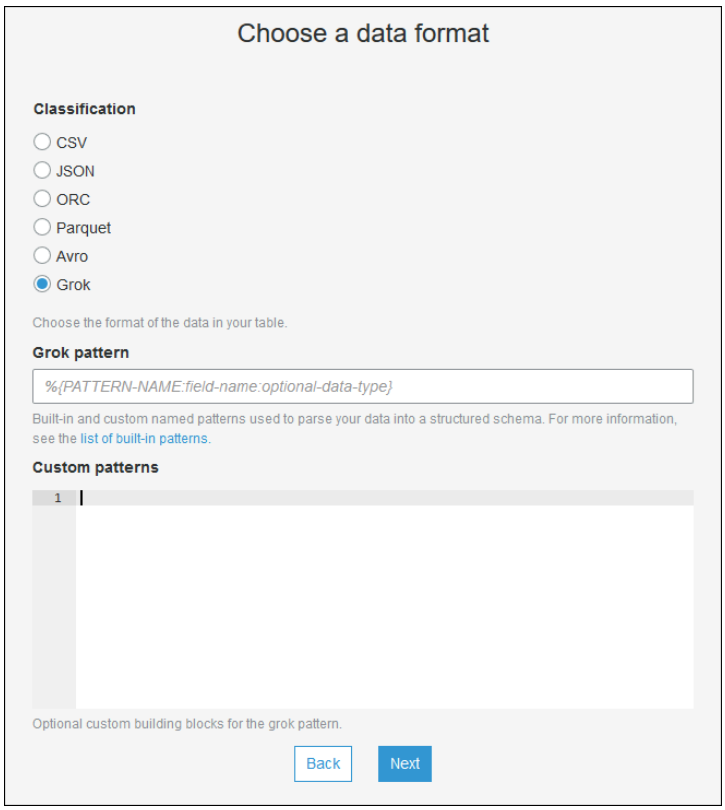
ストリーミングソースへの grok パターンの適用

ログデータソース用のストリーミング ETL ジョブを作成し、Grok パターンを使用してログを構造化データに変換できます。その後、ETL ジョブは、データを構造化データソースとして処理します。ストリーミングソースの Data Catalog テーブルを作成するときに適用する Grok パターンを指定します。

Grok パターンとカスタムパターン文字列値については、「Grok カスタム分類子の書き込み」を参照してください。

Grok パターンを Data Catalog テーブルに追加するには (コンソール)

- [create table] (テーブル作成) ウィザードを使用し、the section called “ストリーミングソースの Data Catalog テーブルの作成” に指定されたパラメータでテーブルを作成します。データ形式を Grok として指定し、[Grok pattern] (Grok パターン) フィールドに入力し、必要に応じてカスタムパターンを [Custom patterns (optional)] (カスタムパターン (オプション)) に追加します。



各カスタムパターンの後に、Enter を押します。

Grok パターンを Data Catalog テーブルに追加するには (AWS Glue API または AWS CLI)

- `GrokPattern` パラメータと、必要に応じて `CustomPatterns` パラメータを `CreateTable` API オペレーションまたは `create_table` CLI コマンドに追加します。

```
 "Parameters": {
...
    "grokPattern": "string",
    "grokCustomPatterns": "string",
...
},
```

`grokCustomPatterns` を文字列として表現し、パターン間の区切りとして「\n」を使用します。

これらのパラメータの指定の例を次に示します。

Example

```
"parameters": {
...
    "grokPattern": "%{USERNAME:username} %{DIGIT:digit:int}",
    "grokCustomPatterns": "digit \d",
...
}
```

ストリーミング ETL ジョブのジョブプロパティの定義

AWS Glue コンソールでストリーミング ETL ジョブを定義する場合は、次に示すストリーム固有のプロパティを指定します。その他のジョブプロパティの説明については、「Spark ジョブのジョブプロパティの定義」を参照してください。

IAM ロール

ジョブの実行、ストリーミングソースへのアクセス、およびターゲットデータストアへのアクセスに使用されるリソースへの権限付与に使用される AWS Identity and Access Management (IAM) ロールを指定します。

Amazon Kinesis Data Streams にアクセスする場合は、 AWS 管理ポリシー `AmazonKinesisFullAccess` をロールにアタッチするか、よりきめ細かにアクセスを許可する同様の IAM ポリシーをアタッチします。サンプルポリシーについては、「IAM を使用して Amazon Kinesis Data Streams リソースへのアクセスを制御する」を参照してください。

AWS Glue でジョブを実行するためのアクセス権限の詳細については、AWS Glue のアイデンティティとアクセスの管理 を参照してください。

Type

[Spark Streaming] (スパークストリーミング)を選択します。

AWS Glue バージョン

AWS Glue バージョンによって、ジョブで使用できる Apache Spark、および Python または Scala のバージョンが決まります。ジョブで使用可能な Python または Scala のバージョンを指定する選択肢を選択します。AWS Glueバージョン 2.0 (Python 3 をサポート) は、ストリーミング ETL ジョブのデフォルトです。

メンテナンスウィンドウ

ストリーミングジョブを再起動できるウィンドウを指定します。「the section called “メンテナンスウィンドウ”」を参照してください。

ジョブのタイムアウト

必要に応じて、長さを分単位で入力します。デフォルト値は空白です。

- ストリーミングジョブのタイムアウト値は 7 日または 10080 分未満である必要があります。
- 値が空白の場合、メンテナンスウィンドウを設定していない場合、ジョブは 7 日後に再起動されます。メンテナンスウィンドウを設定している場合、ジョブは7 日後にメンテナンスウィンドウ中に再起動されます。

データソース

「the section called “ストリーミングソースの Data Catalog テーブルの作成”」で作成したテーブルを指定します。

データターゲット

次のいずれかを行います。

- [データターゲットでテーブルを作成する] を選択し、次に示すデータターゲットプロパティを指定します。

  データストア

  Amazon S3 または JDBC を選択します。

  形式

  任意の形式を選択します。ストリーミングではすべてがサポートされています。
- [Use tables in the data catalog and update your data target] (データカタログのテーブルを使用し、データターゲットを更新する) を選択し、JDBC データストアのテーブルを選択します。

出力スキーマ定義

次のいずれかを行います。

- [Automatically detect schema of each record] (各レコードのスキーマを自動的に検出) を選択してスキーマ検出を有効にします。AWS Glue は、ストリーミングデータからスキーマを決定します。
- [Specify output schema for all records] (すべてのレコードの出力スキーマを指定する) を選択して、[Apply Mapping] (マッピングの適用) 変換を使用して出力スキーマを定義します。

スクリプト

必要に応じて、独自のスクリプトを指定するか、生成されたスクリプトを変更して、Apache Spark 構造化ストリーミングエンジンでサポートされる操作を実行します。使用可能な操作の詳細については、「Operations on streaming DataFrames/Datasets」を参照してください。

ストリーミング ETL に関する注意と制限

次の注意事項と制限事項に留意してください。

- AWS Glue ストリーミング ETL ジョブでの自動解凍は、サポートされた圧縮タイプに対してのみ使用できます。また、以下の点にも注意してください。
  - Framed Snappy とは、公式のフレーミング形式を使用する Snappy を指します。
  - Deflate がサポートされるのは、Glue バージョン 3.0 です (Glue バージョン 2.0 ではサポートされません)。
- スキーマ検出を使用する場合、ストリーミングデータの結合を実行できません。
- AWS Glue のストリーミング ETL ジョブは、Avro 形式の AWS Glue スキーマレジストリに対する Union データ型をサポートしません。
- ETL スクリプトでは、AWS Glue の組み込み変換と Apache Spark 構造化ストリーミングのネイティブな変換を使用できます。詳細については、Apache Spark ウェブサイトの「ストリーミング DataFrames/Datasets の操作」または「AWS Glue PySpark 変換リファレンス」を参照してください。
- AWS Glue ストリーミング ETL ジョブでは、チェックポイントを使用して、読み取ったデータの追跡が行われます。このため、停止して再起動されたジョブは、ストリーム内で中断した位置から開始されます。データを再処理する場合は、スクリプト内で参照されているチェックポイントフォルダを削除することができます。
- ジョブのブックマークはサポートされていません。
- ジョブで Kinesis Data Streams の拡張ファンアウト機能を使用するには、「the section called “Kinesis ストリーミングジョブでの拡張ファンアウトの使用”」を参照してください。
- AWS Glue スキーマレジストリ から作成されたData Catalog テーブルを使用する場合、新しいスキーマバージョンが使用可能になったら、新しいスキーマを反映するには、次の操作を行う必要があります。
  1. テーブルに関連するジョブを停止します。
  2. Data Catalog テーブルのスキーマを更新します。
  3. テーブルに関連付するジョブを再起動します。

AWS Lake Formation FindMatches によるレコードのマッチング

> **Note**
>
> レコードの照合は現在、次のリージョンの AWS Glue コンソールではご利用いただけません: 中東 (アラブ首長国連邦)、欧州 (スペイン)、アジアパシフィック (ジャカルタ)、欧州 (チューリッヒ)。

AWS Lake Formation には、カスタム変換を作成するための機械学習機能が用意されています。現在、FindMatches という名前の変換が 1 つあります。FindMatches 変換を使用すると、レコードに共通の一意の識別子がなく、正確に一致するフィールドがない場合でも、データセット内の重複レコードまたは一致するレコードを識別できます。そのため、コードを記述したり、機械学習の仕組みを知ったりする必要はありません。FindMatches は、次のようなさまざまな問題で役立ちます。

- 一致する顧客: 多くの顧客フィールドがデータベース間で正確に一致しない場合でも (名前のスペルや住所の違い、データの欠落や不正確ななど)、異なる顧客データベース間で顧客レコードをリンクします。
- 製品のマッチング: カタログ内の製品を、競合他社のカタログに対する製品カタログなど、他の製品ソースと照合します。エントリの構造は異なります。
- 不正検出の向上: 重複した顧客アカウントを特定し、新しく作成したアカウントが、以前に知られている不正ユーザーと一致する (またはその可能性のある) タイミングを判断します。
- マッチングに関するその他の問題: マッチアドレス、映画、パーツリストなど。一般的に、人間がデータベース行を見て、それらが一致していると判断した場合、FindMatches 変換が役に立つ可能性は非常にあります。

これらの変換は、ジョブの作成時に作成できます。作成する変換は、ラベル付けするソースデータセットからのストアスキーマとサンプルデータに基づいています (このプロセスを変換の「トレーニング」と呼びます)。ラベルを付けるレコードは、ソースデータセットに存在する必要があります。このプロセスでは、ラベル付けしたファイルを生成し、変換が学習する方法でアップロードし直します。変換を教えたら、Spark ベースの AWS Glue ジョブ (PySpark または Scala Spark) から呼び出し、互換性のあるソースデータストアを持つ他のスクリプトで使用できます。

作成した変換は AWS Glue に保存されます。AWS Glue コンソールで、作成した変換を管理できます。[データ統合とETL] のナビゲーションペインで [データ分類ツール] > [レコードのマッチング] の順に移動して、機械学習変換を編集し、トレーニングを継続できます。コンソールで変換を管理する方法の詳細については、「 機械学習変換の使用」を参照してください。

> **Note**
>
> AWS Glue version 2.0 FindMatchesジョブは、transformがデータを処理している間、`aws-glue-temp-<accountID>-<region>` 一時ファイルを保存するためにAmazon S3バケットを使用します。このデータは、手動または Amazon S3 ライフサイクルルールを設定して、実行の完了後に削除する事ができます。

機械学習変換のタイプ

機械学習変換を作成して、データを最適化できます。これらの変換は、ETL スクリプトから呼び出すことができます。データは、変換から変換へと DynamicFrame というデータ構造で渡されます。これは、Apache Spark SQL `DataFrame` を拡張したものです。`DynamicFrame` にはデータが含まれており、データを処理するためにそのスキーマを参照します。

次のタイプの機械学習変換を利用できます。

Find matches (一致の検索)

ソースデータの重複したレコードを見つけます。この機械学習変換をトレーニングするには、サンプルデータセットにラベルを付けて、どの行が一致するかを示します。機械学習変換は、ラベル付けしたサンプルデータを使用してトレーニングするほど、どの行が一致するかを学習します。変換の設定方法に応じて、出力は次のいずれかになります。

- 入力テーブルのコピーと、一致するレコードのセットを示す値が入力された `match_id` 列。`match_id` 列は任意の識別子です。同じ `match_id` を持つレコードは、互いに一致するレコードとして識別されています。`match_id` が異なるレコードが一致しません。
- 重複する行が削除された入力テーブルのコピー。複数の重複が見つかった場合、最も低いプライマリキーを持つレコードが保持されます。

インクリメンタルマッチを検索する

Find matches トランスフォームは、既存のフレームとインクリメンタルフレーム間で一致を検索し、一致グループごとに一意の ID を含む列を出力として返すように構成することもできます。

詳細については、「インクリメンタルマッチを検索する。」を参照してください。

FindMatches 変換の使用

`FindMatches` 変換を使用してソースデータの重複したレコードを見つけることができます。変換をトレーニングするために役立つラベリングファイルが生成または提供されます。

> **Note**
>
> 現在、カスタム暗号化キーを使用する FindMatches 変換は、次のリージョンではサポートされていません。
>
> - アジアパシフィック (大阪) - ap-northeast-3

FindMatches 変換の使用を開始する場合は、以下の手順に従ってください。より高度で詳細な例については、AWS ビッグデータブログの「Harmonize data using AWS Glue and AWS Lake Formation FindMatches ML to build a customer 360 view」を参照してください。

Find Matches Transform (一致の検索変換) の使用開始

FindMatches 変換の使用を開始するには、次の手順に従います。

1. 最適化するソースデータ用のテーブルを AWS Glue Data Catalog に作成します。クローラーの作成方法については、「Working with Crawlers on the AWS Glue Console」を参照してください。

   ソースデータがカンマ区切り値 (CSV) ファイルなどのテキストベースのファイルである場合は、以下の点を考慮してください。

   - 入力レコード CSV ファイルとラベリングファイルを別個のフォルダに保持します。そうしないと、AWS Glue クローラーはこれらのファイルを同じテーブルの複数の部分と見なして、データカタログに誤ったテーブルを作成する場合があります。
   - CSV ファイルに ASCII 文字のみが含まれている場合を除き、CSV ファイルには必ず BOM (バイトオーダーマーク) なしの UTF-8 エンコードを使用します。Microsoft Excel は、UTF-8 CSV ファイルの先頭に BOM を追加することがよくあります。これを削除するには、CSV ファイルをテキストエディタで開き、ファイルを BOM なしの UTF-8 として保存し直します。
2. AWS Glue コンソールでジョブを作成し、[Find matches (一致の検索)] 変換タイプを選択します。

   > **Important**
   >
   > ジョブ用に選択するデータソーステーブルの列数は 100 を超えることはできません。

3. [Generate labeling file] (ラベルファイルを生成) を選択して、ラベル付けファイルを生成するように AWS Glue に伝えます。AWS Glue は、各 labeling_set_id に対して類似のレコードをグループ化する最初のパスを実行し、グループ化を確認できるようにします。label 列で一致のラベルを付けます。

- ラベル付けファイルが既にある場合 (つまり、一致する行を示すサンプルレコードがある場合) は、そのファイルを Amazon Simple Storage Service (Amazon S3) にアップロードします。ラベリングファイルの形式の詳細については、「ラベリングファイルの形式」を参照してください。ステップ 4 に進みます。

4. ラベル付けファイルをダウンロードし、「ラベリング 」 セクションの説明に従ってファイルにラベルを付けます。
5. 修正したラベル付きファイルをアップロードします。AWS Glue は一致の検索方法について変換をトレーニングするためのタスクを実行します。

   [Machine learning transforms (機会学習変換)] リストのページで、[History (履歴)] タブを選択します。このページは、AWS Glue で以下のタスクを実行するタイミングを指定します。

   - ラベルのインポート
   - ラベルのエクスポート
   - ラベルの生成
   - 品質の推定
6. より適切な変換を作成するには、ラベル付きファイルのダウンロード、ラベル付け、アップロードを繰り返すことができます。初期の実行では、レコードのミスマッチがより多く発生しがちです。ただし、ラベリングファイルを検証して継続的にトレーニングすることで、AWS Glue は学習します。
7. 変換を評価して微調整するには、一致の検索のパフォーマンスと結果を評価します。詳細については、「AWS Glue での機械学変換の調整」を参照してください。

ラベリング

FindMatches がラベル付けファイルを生成すると、ソーステーブルからレコードが選択されます。これまでのトレーニングに基づき、FindMatches は最も学習価値が高いレコードを特定します。

ラベル付けでは、ラベリングファイル (Microsoft Excel などのスプレッドシートの使用を推奨) を編集し、一致するレコードと一致しないレコードを識別する `label` 列に識別子またはラベルを追加します。ソースデータの一致について一貫した明確な定義を持つことが重要です。FindMatches は、ユーザーが一致 (または不一致) として指定したレコードから学習します。また、ユーザーの判断を使用して重複するレコードを見つける方法を学習します。

ラベリングファイルが FindMatches で生成されると、約 100 個のレコードが生成されます。これらの 100 個のレコードは通常 10 個のラベリングセットに分割され、各ラベリングセットは FindMatches により生成される一意の `labeling_set_id` により識別されます。各ラベリング

セットは、他のラベリングセットとは独立した個別のラベリングタスクとして表示する必要があります。タスクは、各ラベリングセット内の一致レコードと非一致レコードを識別することです。

スプレッドシートでラベル付けファイルを編集するためのヒント

スプレッドシートアプリケーションでラベリングファイルを編集する場合は、以下の点を考慮してください。

- ファイルを開いたときに、列フィールドが完全に展開されていない場合があります。必要に応じて `labeling_set_id` 列と `label` 列を展開し、これらのセルの内容を表示してください。
- プライマリキー列が数値 (`long` データ型など) である場合、スプレッドシートはこれを数値として解釈し、値を変更する場合があります。このキー値は、テキストとして扱う必要があります。この問題を修正するには、プライマリキー列のすべてのセルを テキストデータとしてフォーマットします。

ラベリングファイルの形式

`FindMatches` 変換をトレーニングするために AWS Glue により生成されたラベリングファイルは、次の形式を使用します。AWS Glue 用に独自のファイルを生成する場合は、次の形式にも従う必要があります。

- これは、カンマ区切り値 (CSV) ファイルです。
- `UTF-8` でエンコードする必要があります。Microsoft Windows を使用してファイルを編集すると、ファイルは `cp1252` でエンコードされる場合があります。
- このファイルを AWS Glue に渡すには、ファイルを Amazon S3 の場所に配置する必要があります。
- 各ラベル付けタスクに使用する行は多すぎないようにします。タスクごとに許容される行数は 2～30 行ですが、10～20 行をお勧めします。50 行を超えるタスクは推奨されません。粗悪な結果やシステム障害を引き起こす可能性があります。
- 「一致」または「不一致」とラベリングされたレコードのペアで構成される、ラベリングされたデータがすでにある場合、これは問題ありません。これらのラベリングされたペアは、サイズ 2 のラベリングセットとして表すことができます。この場合、両方のレコードに一致する場合は文字「A」とラベリングし、一致しない場合は「B」とラベリングします。

> **Note**
>
> ラベリングファイルには追加の列が含まれているため、ソースデータが含まれているファイルとはスキーマが異なります。ラベリングファイルは、他のすべての変換入力 CSV

ファイルとは異なるフォルダに配置し、データカタログのテーブルの作成時に AWS Glue クローラーによって同種のファイルと見なされないようにします。そうしないと、AWS Glue クローラーで作成されたテーブルではデータが正しく表現されない場合があります。

- 最初の 2 つの列 (`labeling_set_id`、`label`) は AWS Glue の必須列です。残りの列は、処理するデータのスキーマと一致する必要があります。
- `labeling_set_id` ごとに、同じラベルを使用してすべての一致するレコードを特定します。ラベルは `label` 列に配置されている一意の文字列です。シンプルな文字 (A、B、C など) を含むラベルを使用することをお勧めします。ラベルは、大文字と小文字が区別され、 `label` 列に入力されます。
- 同じ `labeling_set_id` と同じラベルを含む行は、一致とみなされ、ラベリングされます。
- 同じ `labeling_set_id` と異なるラベルを含む行は、不一致とみなされ、ラベリングされます。
- 異なる `labeling_set_id` を含む行は、一致に関するいかなる情報も伝達しないとみなされます。

以下に示しているのは、データのラベル付けの例です。

labeling_set_id	ラベル	first_name	last_name	誕生日
ABC123	A	ジョン	Doe	04/01/1980
ABC123	B	Jane	Smith	04/03/1980
ABC123	A	Johnny	Doe	04/01/1980
ABC123	A	Jon	Doe	04/01/1980
DEF345	A	Richard	Jones	12/11/1992
DEF345	A	Rich	Jones	11/12/1992
DEF345	B	Sarah	Jones	12/11/1992
DEF345	C	Richie	Jones Jr。	05/06/2017
DEF345	B	Sarah	Jones-Walker	12/11/1992
GHI678	A	Robert	Miller	1/3/1999

labeling_set_id	ラベル	first_name	last_name	誕生日
GHI678	A	Bob	Miller	1/3/1999
XYZABC	A	William	Robinson	2/5/2001
XYZABC	B	Andrew	Robinson	2/5/1971

- 上記の例では、John/Johnny/Jon Doe を一致と認識し、これらのレコードは Jane Smith と一致しないことをシステムにトレーニングします。これとは別に、Richard と Rich Jones は同一人物であるが、これらのレコードは Sarah Jones/Jones-Walker および Jones Jr. とは一致しないとシステムをトレーニングします。
- ご覧のとおり、ラベルの範囲は `labeling_set_id` に制限されています。したがって、ラベルは `labeling_set_id` 境界を超えません。たとえば、`labeling_set_id` 1 のラベル「A」は、`labeling_set_id` 2 のラベル「A」とは何の関係もありません。
- ラベリングセット内に一致するものがほかにないレコードには、一意のラベルを割り当てます。たとえば、Jane Smith はラベリングセット ABC123 内のどのレコードとも一致しないため、そのラベリングセット内で唯一の、ラベル B を持つレコードです。
- ラベリングセット「GHI678」は、ラベリングセットが、同じラベルが付与された 2 つのレコードのみで構成されることを表示し、一致していることを示します。同様に、「XYZABC」は、異なるラベルが付与された 2 つのレコードを表示し、一致しないことを示します。
- ラベリングセットには一致が含まれていないこと（ラベリングセット内のレコードそれぞれに異なるラベルを付ける）も、ラベリングセットがすべて「同じ」こと（すべてに同じラベルを付けた）もあります。これは、ラベリングセットに、基準によって「同じ」レコードと「同じでない」レコードの例がまとめて含まれている限り、問題ありません。

> **Important**
>
> AWS Glue に渡す IAM ロールに、ラベリングファイルが格納されている Amazon S3 バケットへのアクセス権があることを確認してください。慣例として、AWS Glue ポリシーでは、名前に「aws-glue-」というプレフィックスが付いている Amazon S3 バケットまたはフォルダにアクセス許可を付与します。ラベリングファイルが別の場所にある場合、IAM ロールで、その場所に対してアクセス許可を追加します。

AWS Glue での機械学習変換の調整

AWS Glue で機械学習変換を調整することで、目的に応じてデータクレンジングジョブの結果を改善できます。変換を改善するには、変換をトレーニングします。そのためには、ラベリングセットを生成し、ラベルを追加します。これらのステップを、目的の結果が得られるまで何回も繰り返します。いくつかの機械学習パラメータを変更することで調整することもできます。

機械学習変換の詳細については、「AWS Lake Formation FindMatches によるレコードのマッチング」を参照してください。

トピック

- 機械学習の測定
- 適合率と再現率のトレードオフ
- 精度とコストのトレードオフ
- 一致信頼度スコアを使用して一致の品質を見積もります。
- 「一致の検索」変換のトレーニング

機械学習の測定

機械学習変換の調整に使用する測定を理解するには、以下の用語を理解しておく必要があります。

真陽性 (TP)

変換が正しく検出したデータ内の一致。ヒットとも呼ばれます。

真陰性 (TN)

変換が正しく拒否したデータ内の不一致。

偽陽性 (FP)

変換が誤って一致として分類したデータ内の不一致。誤警告とも呼ばれます。

偽陰性 (FN)

変換が検出しなかったデータ内の一致。 ミスとも呼ばれます。

機械学習で使用される用語の詳細については、Wikipedia の「混同行列」を参照してください。

機械学習変換を調整するには、変換の [詳細プロパティ] で以下の測定値を変更できます。

- 適合率は、変換で要請と判定されたレコードの総数 (真陽性と偽陽性) のうち、どの程度真陽性を検出するかの尺度です。詳細については、Wikipedia の「適合率と再現率」を参照してください。
- 再現率は、変換がソースデータのレコード総数から真陽性を見つける割合を測定します。詳細については、Wikipedia の「適合率と再現率」を参照してください。
- 精度は、変換が真陽性と真陰性を見つける割合を測定します。精度を高めるには、マシンリソースとコストを増やす必要があります。ただし、再現率も高くなります。詳細については、Wikipedia の「適合率と再現率」を参照してください。
- コストは、変換を実行するために消費されるコンピューティングリソースの数およびそれに伴う金額を測定します。

適合率と再現率のトレードオフ

各 FindMatches 変換には、precision-recall パラメータが含まれています。このパラメータを使用して、次のいずれかを指定します。

- 変換の結果で、2 つのレコードが誤って一致すると報告され、実際には一致していない場合は、適合率を高めます。
- 変換の結果で、2 つのレコードが実際には一致しているにもかかわらず、一致として検出されない場合は、再現率を高めます。

このトレードオフを行うには、AWS Glue コンソールまたは AWS Glue 機械学習 API オペレーションを使用できます。

適合率を優先する場合

FindMatches で実際には一致しないレコードのペアが一致と見なされるリスクに対処するには、適合率を優先します。適合率を優先するには、適合率-再現率トレードオフとしてより高い値を選択します。値を高くするほど、FindMatches 変換でレコードのペアを一致と見なす根拠がさらに必要となります。変換は調整されて、レコードのペアが一致しないと判断する傾向が強化されます。

たとえば、FindMatches を使用してビデオカタログ内の重複する商品を検出する際に、適合率-再現率のより高い値を変換に指定したとします。変換で「スターウォーズ: 新たなる希望」と「スターウォーズ: 帝国の逆襲」が誤って同じものとして検出された場合、「新たなる希望」を欲しい利用者に「帝国の逆襲」が提供される可能性があります。これは、カスタマーエクスペリエンスを悪化させることになります。

一方、変換で「スターウォーズ: 新たなる希望」と「スターウォーズ エピソード 4/新たなる希望」が同じ商品として検出されない場合、利用者は最初は混乱するとしても、最終的には同じものとして認識する可能性があります。これは誤検出ではあっても、前の例ほど深刻ではありません。

再現率を優先する場合

FindMatches 変換の結果で、実際には一致するレコードのペアが一致として検出されないリスクに対処するには、再現率を優先します。再現率を優先するには、適合率-再現率トレードオフとしてより低い値を選択します。値を低くするほど、FindMatches 変換でレコードのペアを一致と見なす根拠が少なくて済みます。変換は調整されて、レコードのペアが一致すると判断する傾向が強化されます。

たとえば、これはセキュリティを重視する組織で優先される場合があります。利用者を詐欺行為者のリストと照合する場合、利用者が詐欺行為者であるかどうかを判断することが重要です。FindMatches を使用して利用者リストと詐欺行為者リストを照合するとします。FindMatches において 2 つのリスト間で一致が検出されるたびに、該当する利用者が実際に詐欺行為者であるかどうかを人間の監査担当者が確認します。このような組織では、適合率よりも再現率を優先できます。つまり、利用者が実際に詐欺行為者リストに該当することを見逃すよりは、利用者が詐欺行為者ではないことを監査担当者が手動で確認して除外することを優先します。

適合率と再現率の両方を優先する方法

適合率と再現率の両方を改善する最適な方法は、ラベル付けするデータを増やすことです。より多くのデータをラベル付けすると、FindMatches 変換全体の精度が向上するため、適合率と再現率の両方が向上します。ただし、最も正確な変換であっても、適合率の優先、再現率の優先、または中間値の選択を試す必要があるグレー領域が常に存在します。

精度とコストのトレードオフ

各 FindMatches 変換には、accuracy-cost パラメータが含まれています。このパラメータを使用して、次のいずれかを指定できます。

- 変換で 2 つのレコードの一致が正確に報告されることをより重視する場合は、精度を強調します。
- 変換を実行するコストまたはスピードをより重視する場合は、より低いコストを強調します。

このトレードオフを行うには、AWS Glue コンソールまたは AWS Glue 機械学習 API オペレーションを使用できます。

精度を優先する場合

find matches の結果に一致が含まれないリスクに対処するには、精度を優先します。精度を優先するには、精度-コストトレードオフのより高い値を選択します。値が高いほど、FindMatches 変換でレコードを正しく一致させるために、より詳細な検索を行う時間を増やす必要があります。このパラメータは、一致しないレコードのペアを誤って一致と判断する可能性を減らすものではありません。変換は調整されて、一致の検索に費やす時間が増加されます。

コストを優先する場合

一致の検索数よりも find matches 変換を実行するコストをより重視する場合は、コストを優先します。コストを優先するには、精度-コストトレードオフのより低い値を選択します。値を低くするほど、 FindMatches 変換で実行する必要があるリソース数が減ります。一致の検索数を減らすように、変換が調整されます。より低いコストを優先したときの結果が許容できるものであれば、この設定を使用します。

精度とより低いコストの両方を優先する方法

レコードが一致するかどうかを判断するためにより多くのレコードのペアを調査するには、より多くのマシン時間が必要になります。品質を下げずにコストを削減するには、以下を実行できます。

- 一致の対象としないレコードをデータソースから除外します。
- 一致/不一致の判断に役立たないことが確実な列をデータソースから除外します。これを決める適切な方法としては、一連のレコードが「同じ」であるかどうかの判断に影響しないと思われる列を除外します。

一致信頼度スコアを使用して一致の品質を見積もります。

一致信頼度スコアは、機械学習モデルの信頼度が高い、不確か、またはありそうにない一致レコードを区別するために、findMatches によって検出された一致の品質を推定します。一致信頼度スコアは 0 から 1 の間で、スコアが高いほど類似度が高くなります。一致信頼度スコアを調べると、システムが非常に信頼できる一致のクラスター (マージすることを決定する可能性があります)、システムが不確実であるクラスター (人間がレビューしたと決定する可能性がある)、および一致のクラスターを区別できます。 システムは可能性が低いと見なします (拒否することを決定する場合があります)。

一致信頼度スコアが高いが一致がないと判断した場合、またはスコアが低いが実際に一致しているかどうかを判断する状況では、トレーニングデータを調整できます。

信頼度スコアは、すべての FindMatches 決定を確認することが不可能な、大規模な産業用データセットがある場合に特に役立ちます。

一致信頼度スコアは AWS Glue バージョン 2.0 以降。

マッチ信頼度スコアの生成

ブール値を computeMatchConfidenceScores Trueに設定するか、FindMatches または FindIncrementalMatches APIを呼び出すときに一致信頼スコアを生成できます。

AWS Glue 新規に column match_confidence_score を出力に追加します。

スコアリングの例に一致

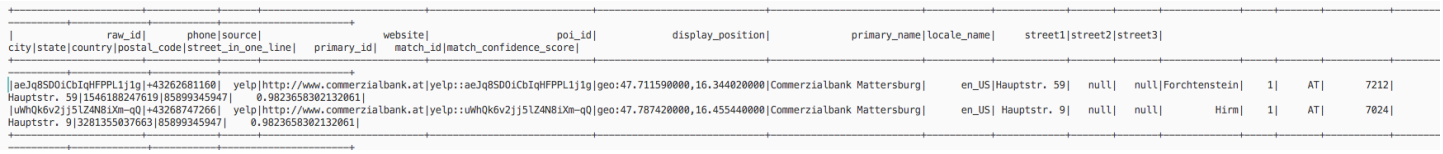
たとえば、次のマッチングレコードを考えてみましょう。

スコア >= 0.9

一致したレコードのサマリー:

```
  primary_id  |   match_id  | match_confidence_score

3281355037663    85899345947   0.9823658302132061
1546188247619    85899345947   0.9823658302132061
```

詳細:



```
+-------------+-----------+------+--------------------+--------------------+--------------------+--------------+-----------+-------+-------+-------+-------+-------+-------+-------+-------+
|       raw_id|      phone|source|             website|              poi_id|    display_position|  primary_name|locale_name|street1|street2|street3|   city|  state|country|postal_code|street_in_one_line|
+-------------+-----------+------+--------------------+--------------------+--------------------+--------------+-----------+-------+-------+-------+-------+-------+-------+-------+-------+
|aeJq8SDOiCbIqHFPPL1j1g|+43262681160|  yelp|http://www.commerzialbank.at|yelp::aeJq8SDOiCbIqHFPPL1j1g|geo:47.711590000,16.344020000|Commerzialbank Mattersburg|      en_US|Hauptstr. 59|   null|   null|Forchtenstein|    1|     AT|      7212|
Hauptstr. 59|1546188247619|85899345947|    0.9823658302132061|
|uWhQk6v2jj5lZ4N8iXm-qQ|+43268747266|  yelp|http://www.commerzialbank.at|yelp::uWhQk6v2jj5lZ4N8iXm-qQ|geo:47.787420000,16.455440000|Commerzialbank Mattersburg|      en_US| Hauptstr. 9|   null|   null|         Hirm|    1|     AT|      7024|
Hauptstr. 9|3281355037663|85899345947|    0.9823658302132061|
+-------------+-----------+------+--------------------+--------------------+--------------------+--------------+-----------+-------+-------+-------+-------+-------+-------+-------+-------+
```

この例から、2 つのレコードが非常によく似ていて、display_position、primary_name、およびstreet name を共有していることがわかります。

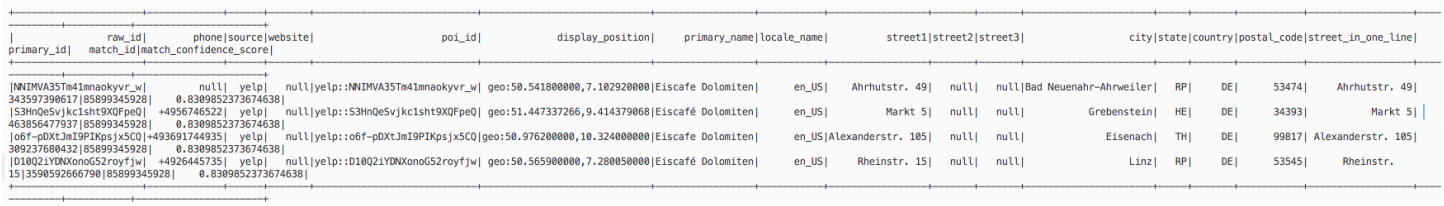
スコア >= 0.8、得点 < 0.9

一致したレコードのサマリー:

```
  primary_id  |   match_id  | match_confidence_score

309237680432     85899345928   0.8309852373674638
3590592666790    85899345928   0.8309852373674638
343597390617     85899345928   0.8309852373674638
249108124906     85899345928   0.8309852373674638
```

```
463856477937      85899345928    0.8309852373674638
```

詳細:



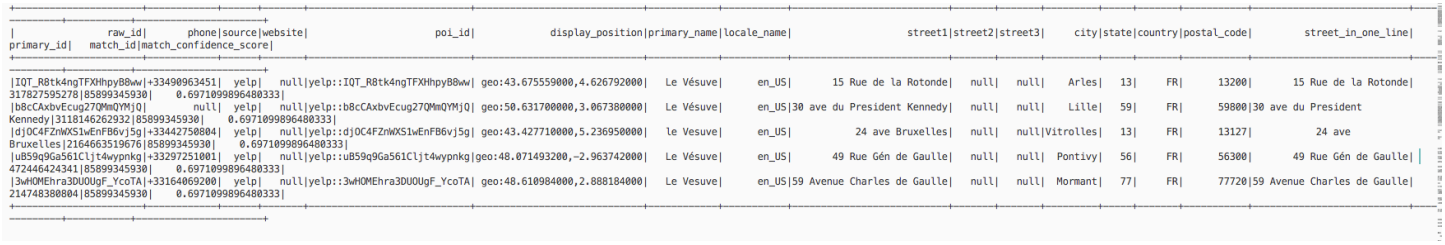
この例から、これらのレコードが同じ primary_name および country を共有していることがわかります。

スコア >= 0.6、得点 < 0.7

一致したレコードのサマリー:

```
  primary_id   |   match_id   | match_confidence_score

2164663519676    85899345930   0.6971099896480333
 317827595278    85899345930   0.6971099896480333
 472446424341    85899345930   0.6971099896480333
3118146262932    85899345930   0.6971099896480333
 214748380804    85899345930   0.6971099896480333
```

詳細:



この例から、これらのレコードが同じもの primary_name を共有しているだけであることがわかります 。

詳細については、以下を参照してください。

- ステップ 5: 機械学習変換でジョブを追加して実行する
- PySpark: FindMatches クラス

- PySpark: FindIncrementalMatches クラス
- Scala: FindMatches クラス
- Scala: FindIncrementalMatches クラス

「一致の検索」変換のトレーニング

一致と不一致を見分けられるように各 FindMatches 変換をトレーニングする必要があります。変換をトレーニングするには、ファイルにラベルを追加して選択結果を AWS Glue にアップロードします。

このラベル付けは、AWS Glue コンソールまたは AWS Glue 機械学習 API オペレーションを使用して調整できます。

ラベルを追加する回数は? 必要なラベルの数は?

これらの質問に対する回答は、主にユーザーが決定します。FindMatches で必要な精度レベルを達成できるかどうか、さらにラベル付けの作業を追加するメリットがあるかどうかは、ユーザーが評価する必要があります。これを決定する最善の方法は、AWS Glue コンソールで [Estimate quality] (品質の推定) を選択して、「適合率」、「再現率」、「適合率再現率曲線の下の面積」のメトリクスを調べることです。さらに多くのタスクセットにラベル付けをしたら、これらのメトリクスを再実行して結果が改善されたかどうかを確認します。いくつかのタスクセットをラベル付けをした後で、注目しているメトリクスに改善が見られなかった場合、変換の品質はそれ以上向上しない状態に達している可能性があります。

真陽性と真陰性の両方のラベルが必要な理由は?

FindMatches 変換は、ユーザーが何を一致と見なしているかを学習するために真陽性と真陰性の両方のサンプルを必要とします。FindMatches で生成されるトレーニングデータ ([I do not have labels (ラベルがありません)] オプションなどを使用した場合) にラベル付けを行う場合、FindMatches はユーザーに代わって「ラベルセット ID」を生成しようとします。各タスク内で、一部のレコードに同じ「ラベル」を付け、他のレコードに異なる「ラベル」を付けます。つまり、通常、タスクがすべて同じものであったり、すべて異なるものであったりすることはありません (ただし、特定のタスク内のレコードがすべて「同じ」ものであったり、すべて「異なる」ものであったりすることは問題ありません)。

[Upload labels from S3] (S3 からラベルをアップロードする) オプションを使用して FindMatches 変換をトレーニングする場合は、一致レコードと不一致レコードの両方を含めるようにしてください。1 つのタイプのみを使用することもできます。これらのラベルはより正確な FindMatches 変換

の構築に役立ちますが、[Generate labeling file (ラベリングファイルの生成)] オプションを使用して生成する一部のレコードには依然としてラベル付けを行う必要があります。

どうしたら、ユーザーがトレーニングしたとおりのマッチングを変換に実行させることができるか?

FindMatches 変換は、ユーザーが指定したラベルから学習するため、指定したラベルを優先しないレコードのペアが生成される場合があります。ユーザーのラベルを優先するように FindMatches 変換を強制するには、 [FindMatchesParameter] の [EnforceProvidedLabels] を選択します。

ML 変換が真の一致ではない項目を一致として識別した場合、どのように対策できますか?

以下の対策を実行できます。

- precisionRecallTradeoff の値をより高くします。これにより、検出される一致の数が減りますが、十分に高い値に達すると、大きなクラスターの分割も行われます。
- 不正確な結果に対応する出力行を取得し、これらの行をラベリングセットとして再フォーマットします (match_id 列を削除し、labeling_set_id 列および label 列を追加します)。必要に応じて、複数のラベリングセットに分割 (再分割) し、ラベラーが各ラベリングセットに留意しながらラベルの割り当てができるようにします。次に、一致するセットに正しくラベル付けを行い、ラベルファイルをアップロードして既存のラベルに付加します。これにより、パターンを理解するためにどのようなデータが必要であるかについてトランスフォーマーが十分にトレーニングされます。
- (高度) 最後に、そのデータを確認し、システムが気付いていないパターンを検出できるかどうかを判断します。標準の AWS Glue 関数を使用してそのデータを事前処理し、データを正規化します。データの種類が十分に異なることが判明している場合は、データを種類別に分離し、アルゴリズムに学習させるデータを明確にします。または、複数の列のデータが関連していることが判明している場合は、これらの列を結合します。

機械学習変換の使用

AWS Glue に使用して、データの最適化に使用できるカスタム機械学習変換を作成できます。これらの変換は、AWS Glue コンソールでのジョブの作成時に使用できます。

機械学習変換の作成方法の詳細については、「AWS Lake Formation FindMatches によるレコードのマッチング」を参照してください。

トピック

- 変換のプロパティ
- 機械学習変換の追加と編集

- 変換の詳細の確認
- ラベルを使って変換を教える

変換のプロパティ

既存の機械学習変換を表示するには、AWS Management Console にサインインして、AWS Glue コンソール (https://console.aws.amazon.com/glue/) を開きます。[データ統合と ETL] のナビゲーションペインで、[データ分類ツール] > [レコードのマッチング] の順に選択します。

各変換のプロパティ:

変換名

変換の作成時に付けた一意の変換名。

ID

変換の一意の識別子。

[Label count] (ラベル数)

変換のトレーニングに役立てるために指定されたラベリングファイル内のラベルの数。

ステータス

変換が [Ready] (使用可能) であるか、[Needs training] (トレーニングが必要) であるかを示します。ジョブで機械学習変換を正常に実行するには、ジョブのステータスが [Ready] (使用可能) であることが必要です。

作成

変換の作成日。

変更済み

変換が最後に更新された日付。

説明

変換の説明 (説明が入力された場合)。

AWS Glue バージョン

使用する AWS Glue のバージョン。

実行 ID

変換の作成時に付けた一意の変換名。

タスクタイプ

機械学習変換のタイプ (一致するレコードの検索など)。

ステータス

タスク実行のステータスを示します。次のようなステータスがあります。

- スタート
- Running (実行中)
- 停止中
- 停止
- 成功
- Failed (失敗)
- タイムアウト

エラー

ステータスが [失敗] の場合、失敗の理由を説明するエラーメッセージが表示されます。

機械学習変換の追加と編集

AWS Glue コンソールで変換の表示、削除、セットアップ、トレーニング、および調整を行うことができます。リストの変換の横にあるチェックボックスをオンにし、 [アクション] を選択して、実行するアクションを選択します。

新しい ML 変換の作成

新しい機械学習変換を追加するには、[変換の作成] を選択します。次に、[ジョブの追加] ウィザードの手順に従います。詳細については、「AWS Lake Formation FindMatches によるレコードのマッチング」を参照してください。

ステップ 1. 変換のプロパティを設定する。

1. 名前と説明 (オプション) を入力します。
2. オプションで、セキュリティ設定を設定します。「機械学習変換でのデータ暗号化の使用」を参照してください。
3. オプションで、タスク実行の設定を設定します。タスク実行の設定では、タスクの実行方法をカスタマイズできます。ワーカータイプ、ワーカーの数、タスクのタイムアウト (分単位)、再試行回数、AWS Glue バージョンを選択します。

4. 必要に応じて、タグを設定します。タグとは、AWS リソースに割り当てることができるラベルです。各タグは、キー、および値 (オプション) で構成されます。タグを使用して、リソースを検索およびフィルタリングしたり、AWS のコストを追跡できます。

ステップ 2. テーブルとプライマリキーを選択する。

1. AWS Glue カタログデータベースおよびテーブルを選択します。
2. 選択したテーブルからプライマリキーを選択します。プライマリキー列には通常、データソース内のすべてのレコードの固有の識別子が含まれます。

ステップ 3. 調整オプションを選択する。

1. [再現率と適合率] では、再現率または適合率を優先するように変換を調整する調整値を選択します。デフォルトでは [バランス] が選択されていますが、再現率または適合率を優先するように選択するか、[カスタム] を選択して 0.0 ~ 1.0 (両端を含む) の値を入力できます。
2. [コスト削減と正確度] では、コスト削減または正確度を優先するように調整値を選択するか、[カスタム] を選択して 0.0 ~ 1.0 (両端を含む) の値を入力できます。
3. 使用するラベルと出力を一致させることで ML 変換を教える場合は、[一致のエンフォースメント] で、[出力をラベルに一致させる] を選択します。

ステップ 4. 確認して作成します。

1. ステップ 1 ~ 3 のオプションを確認します。
2. 変更が必要な手順については、[編集] を選択します。[変換の作成] を選択して、変換の作成ウィザードを完了します。

機械学習変換でのデータ暗号化の使用

機械学習変換を AWS Glue に追加する際に、データソースまたはデーターゲットに関連付けられたセキュリティ設定をオプションで指定できます。データの格納に使用される Amazon S3 バケットがセキュリティ設定で暗号化されている場合は、変換の作成時に同じセキュリティ設定を指定します。

AWS KMS (SSE-KMS) でサーバー側の暗号化を使用して、モデルとラベルを暗号化し、許可されていない人に閲覧されるのを防ぐという選択もできます。このオプションを選択した場合は、AWS KMS key の名前を選択するように求められます。[Enter a key ARN] (キーの ARN を入力) を選択す

ることもできます。KMS キーの ARN を入力することを選択した場合、KMS キー ARN を入力できる 2 番目のフィールドが表示されます。

> **Note**
>
> 現在、次のリージョンではカスタム暗号化キーを使用する ML 変換がサポートされていません。
>
> - アジアパシフィック (大阪) - `ap-northeast-3`

変換の詳細の確認

変換のプロパティを確認する

[変換プロパティ] ページには、変換の属性が含まれます。変換定義に関する詳細として、以下の内容が表示されます。

- [変換名] は変換の名前を示します。
- [タイプ] は変換のタイプを一覧表示します。
- [ステータス] は、変換がスクリプトまたはジョブで使用可能かどうかを示します。
- [Force output to match labels (出力をラベルに一致させる)] は、ユーザーから提供されたラベルに出力を一致させるかどうかを示します。
- [Spark version] (Spark バージョン) は、変換を追加したときに [Task run properties] (タスク実行のプロパティ) で選択した AWS Glue のバージョンに関連しています。ほとんどのお客様に AWS Glue 1.0 および Spark 2.4 が推奨されます。詳細については、「AWS Glue Versions」を参照してください。

[履歴]、[品質を推定する]、[タグ] タブ

変換の詳細には、変換の作成時に定義した情報が含まれます。変換の詳細を確認するには、機械学習変換のリストで変換を選択し、以下のタブの情報を確認します。

- 履歴
- 品質の推定
- タグ

履歴

[履歴] タブには、変換のタスク実行の履歴が表示されます。変換をトレーニングするには、複数のタイプのタスクを実行します。タスクごとに、実行メトリクスとして以下が含まれます。

- [実行 ID] は、このタスクの実行ごとに AWS Glue によって作成される識別子です。
- [Task type (タスクタイプ)] は、タスク実行のタイプを示します。
- [実行ステータス] は、各タスクの成功した実行を一覧表示します。最新の実行が一番上に表示されます。
- [Error (エラー)] には、実行が正常に行われなかった場合のエラーメッセージの詳細が表示されます。
- [開始時刻] は、タスクの開始日時 (現地時間) を示します。
- [終了時刻] には、タスクの終了日時 (現地時間) が表示されます。
- [ログ] は、このジョブ実行の `stdout` に書き込まれたログにリンクされています。

  [Logs] (ログ) のリンク先は Amazon CloudWatch Logs です。このリンク先では、AWS Glue Data Catalog で作成されたテーブルと、発生したエラーに関する詳細を確認できます。ログの保持期間は、CloudWatch コンソールで管理できます。デフォルトのログ保持期間は `Never Expire` です。保持期間を変更する方法の詳細については、Amazon CloudWatch Logs ユーザーガイドの「CloudWatch Logs でログデータ保管期間を変更する」を参照してください。
- [ラベルファイル] には、生成されたラベリングファイル用の Amazon S3 へのリンクが示されます。

品質の推定

[Estimate Quality (品質の推定)] タブは、変換の品質を測定するために使用するメトリクスを示します。推定値は、ラベル付きデータのサブセットを使用して、指定したラベルに変換一致予測を比較することによって計算されます。これらの推定値はおおよその値です。このタブから [Estimate quality (品質の推定)] タスク実行を呼び出すことができます。

Estimate quality タブには、次のプロパティを含む最後の Estimate quality の実行のメトリクスが表示されます。

- Precision-Recall 曲線の下にある領域は、変換の全体的な品質の上限を推定する単一の数値です。これは適合率-再現率パラメータ用に行った選択とは関係ありません。値が高いほど、適合率と再現率のトレードオフが適切であることを示します。
- 適合率は、変換が一致を推定して、その推定が正確である度合いを見積もります。

- 再現率の上限は、実際の一致件数に対して、変換が推定した一致件数の割合の見積もりです。
- [F1] は、変換の正確性 (1～0) を示します。1 は正確性が最適であることを表します。詳細については、Wikipedia の「F1 スコア」を参照してください。
- [Column importance] (列の重要度) テーブルには、各列の列名と重要度スコアが表示されます。列の重要度は、レコード内のどの列がマッチングを行うために最も使用されているかを識別することで、列がモデルにどの程度寄与しているかを理解するのに役立ちます。列の重要度を上げたり下げたりするために、このデータをラベルセットに追加したりラベルセットを変更したりするように促されることがあります。

  [Importance] (重要度) 列には、各列の数値スコアが 1.0 以下の 10 進数で示されます。

品質の推定と真の品質の比較については、「 品質の推定とエンドツーエンド (真の) 品質の比較」を参照してください。

変換の調整の詳細については、「AWS Glue での機械学変換の調整」を参照してください。

品質の推定とエンドツーエンド (真の) 品質の比較

AWS Glue は、変換の品質を推定するために、内部の機械学習モデルに対してレコードのペアをいくつか提示します。これらのレコードのペアは、一致のラベルを指定したレコードですが、モデルは初見のものです。これらの品質の推定は、機械学習モデルの品質の関数です (モデルは、ユーザーが変換を「トレーニング」するためにラベル付けするレコードの数から影響を受けます)。エンドツーエンド、または真の再現率 (`ML transform` 変換によって自動的に計算されません) は、機械学習モデルに対してさまざまな一致を提案する `ML transform` フィルタリングメカニズムの影響を受けます。

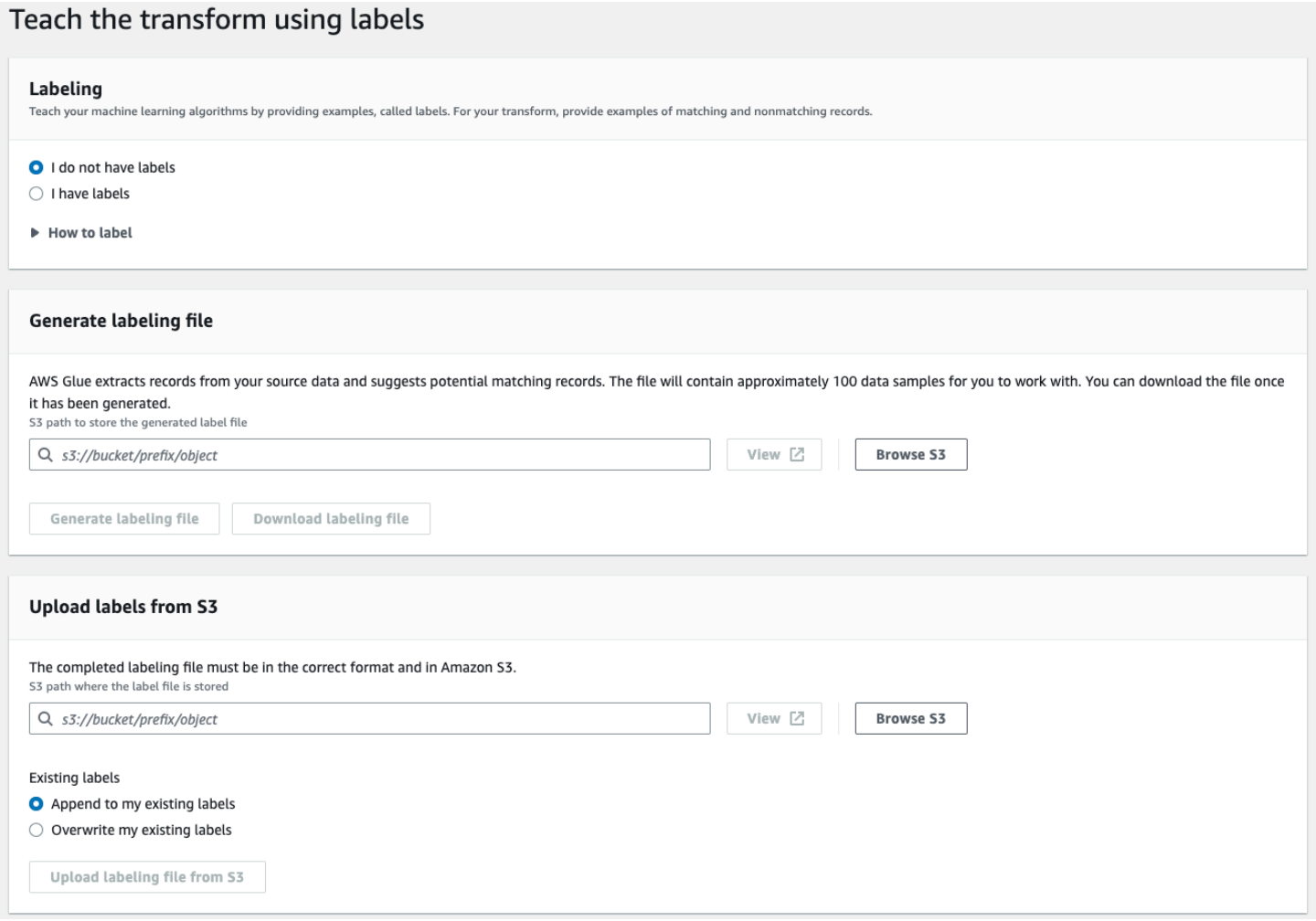
このフィルタリング方法を調整するには、主に [低コスト - 正確性] の調整値を指定します。[正確性] を優先するようにこの調整値を近づけるほど、システムは一致する可能性のあるレコードのペアをより深く、広く検索します。機械学習モデルに取り込まれるレコードのペアが増えると、`ML transform` のエンドツーエンド、または真の再現率がより推定の再現率メトリクスに近づきます。その結果、マッチングのコスト/精度のトレードオフの変更によるマッチングのエンドツーエンドの品質の変更は、通常、品質の見積りには反映されません。

タグ

タグとは、AWS リソースに割り当てることができるラベルです。各タグは、キー、および値 (オプション) で構成されます。タグを使用して、リソースを検索およびフィルタリングしたり、AWS のコストを追跡できます。

ラベルを使って変換を教える

ML 変換の詳細ページから [変換を教える] を選択すると、ラベル (例) を使用して ML 変換を教えることができます。例 (ラベルと呼ばれます) を提供して機械学習のアルゴリズムを教える場合、使用する既存のラベルを選択したり、ラベリングファイルを作成できます。



- [ラベリング] — ラベルがある場合は、[ラベルがあります] を選択します。ラベルがない場合でも、ラベリングファイルを生成する次の手順に進むことができます。
- [ラベリングファイルを生成する] — AWS Glue により、ソースデータからレコードが抽出され、一致する可能性のあるレコードが提案されます。生成されたラベルファイルを保存する Amazon S3 バケットを選択します。[ラベルリングファイルを生成する] を選択してプロセスを開始します。完了したら、[ラベリングファイルをダウンロードする] を選択します。ダウンロードしたファイルには、ラベルを入力できるラベル用の列があります。
- [Amazon S3 からラベルをアップロードする] — ラベルファイルが保存されている Amazon S3 バケットから、完成したラベリングファイルを選択します。その後、既存のラベルにラベルを追加す

るか、既存のラベルを上書きするかを選択します。[Amazon S3 からラベルリングファイルをアップロードする] を選択します。

チュートリアル: AWS Glue で機械学習変換を作成する

このチュートリアルでは、AWS Glue を使用して機械学習 (ML) 変換を作成および管理するためのアクションについてガイドします。このチュートリアルを使用するには、 AWS Glue コンソールを使用してクローラとジョブを追加し、スクリプトを編集する方法をよく理解している必要があります。Amazon Simple Storage Service (Amazon S3) で、ファイルを検索してダウンロードする方法についてもよく理解している必要があります。

次の例では、一致するレコードを見つけるための `FindMatches` 変換を作成します。また、一致する/一致しないレコードを識別する方法を、この変換にトレーニングして AWS Glue ジョブで使用します。AWS Glue ジョブは、`match_id` という別の列を追加して新しい Amazon S3 ファイルを作成します。

このチュートリアルで使用されているソースデータは、`dblp_acm_records.csv` という名前のファイルです。このファイルは、元の DBLP ACM データセットから入手できる学術刊行物 (DBLP および ACM) の修正バージョンです。`dblp_acm_records.csv` ファイルは、BOM (バイトオーダーマーク) なしの UTF-8 形式でエンコードされたカンマ区切り値 (CSV) ファイルです。

2 番目のファイル `dblp_acm_labels.csv` はサンプルのラベリングファイルです。このファイルには、チュートリアルの一環として変換をトレーニングするために使用する、一致する/一致しないレコードの両方が含まれています。

トピック

- ステップ 1: ソースデータをクロールする
- ステップ 2: 機械学習変換を追加する
- ステップ 3: 機械学習変換をトレーニングする
- ステップ 4: 機械学習変換の品質を推定する
- ステップ 5: 機械学習変換でジョブを追加して実行する
- ステップ 6: Amazon S3 の出力データを確認する

ステップ 1: ソースデータをクロールする

まず、ソースである Amazon S3 CSV ファイルをクロールして、対応するメタデータテーブルを Data Catalog に作成します。

> **Important**
>
> CSV ファイルのみのテーブルを作成するようクローラに指示するために、CSV ソースデータを他のファイルとは異なる Amazon S3 フォルダに保存します。

1. AWS Management Consoleにサインインし、AWS Glue コンソール (https://console.aws.amazon.com/glue/) を開きます。
2. ナビゲーションペインで、[クローラ]、[クローラの追加] の順に選択します。
3. ウィザードに従って `demo-crawl-dblp-acm` というクローラを作成して実行し、出力をデータベース `demo-db-dblp-acm` に保存します。ウィザードの実行時に、データベース`demo-db-dblp-acm` がまだ存在していない場合は、これを作成します。現在の AWS リージョンでのサンプルデータへの Amazon S3 インクルードパスを選択します。例えば、`us-east-1` の場合、ソースファイルへの Amazon S3 インクルードパスは `s3://ml-transforms-public-datasets-us-east-1/dblp-acm/records/dblp_acm_records.csv` です。

   完了すると、クローラによってテーブル `dblp_acm_records_csv` が作成され、テーブルの列として、ID、タイトル、作成者、場所、年、およびソースが設定されます。

ステップ 2: 機械学習変換を追加する

次に、`demo-crawl-dblp-acm` というクローラで作成されたデータソーステーブルのスキーマに基づく機械学習変換を追加します。

1. AWS Glue コンソールにある [データ統合と ETL] のナビゲーションペインで、[データ分類ツール] > [レコードのマッチング] の順に選択し、[変換の追加] を選択します。ウィザードに従って、以下のプロパティを持つ `Find matches` 変換を作成します。

   a. [変換の名前] に、「**`demo-xform-dblp-acm`**」と入力します。これは、ソースデータの一致を見つけるために使用する変換の名前です。

   b. [IAM role] (IAM ロール) で、Amazon S3 ソースデータ、ラベリングファイル、および AWS Glue API オペレーションへのアクセス許可を持つ IAM ロールを選択します。詳細については、AWS Glue デベロッパーガイドの「Create an IAM Role for AWS Glue」を参照してください。

   c. [Data source] (データソース) に、demo-db-dblp-acm というデータベースの dblp_acm_records_csv という名前のテーブルを選択します。

d. [Primary key (プライマリキー)] で、テーブルのプライマリキー列である [ID] を選択します。

2. ウィザードで、[完了] を選択し、[ML 変換] リストに戻ります。

ステップ 3: 機械学習変換をトレーニングする

次に、チュートリアルのサンプルのラベリングファイルを使用して、機械学習変換をトレーニングします。

機械学習変換は、そのステータスが [Ready for use (使用可能)] になるまで、ETL (抽出、変換、ロード) ジョブで使用できません。変換を使用可能にするには、一致するレコードと一致しないレコードのサンプルを提供し、レコードの一致/不一致を見分ける方法について変換をトレーニングする必要があります。変換をトレーニングするには、ラベルファイルを生成して、ラベルを追加し、ラベルファイルをアップロードします。このチュートリアルでは、`dblp_acm_labels.csv` というサンプルラベリングファイルを使用できます。ラベリングプロセスの詳細については、「ラベリング」を参照してください。

1. AWS Glue コンソールのナビゲーションペインで、[レコードのマッチング] を選択します。
2. `demo-xform-dblp-acm` 変換を選択し、次に [アクション]、[Teach (トレーニング)] の順に選択します。ウィザードに従って、`Find matches` 変換をトレーニングします。
3. 変換のプロパティページで、[I have labels (ラベルがあります)] を選択します。現在の AWS リージョンでサンプルラベリングファイルへの Amazon S3 パスを選択します。例えば、`us-east-1` の場合、指定したラベリングファイルを Amazon S3 パス `s3://ml-transforms-public-datasets-us-east-1/dblp-acm/labels/dblp_acm_labels.csv` からアップロードします。この際に、既存のラベルを上書きするオプションを使用します。ラベリングファイルは、AWS Glue コンソールと同じリージョンの Amazon S3 に配置する必要があります。

   ラベリングファイルをアップロードすると、データソースの処理方法について変換をトレーニングするために使用するラベルを追加または上書きするためのタスクが AWS Glue で開始されます。
4. ウィザードの最後のページで、[完了] を選択し、[ML 変換] リストに戻ります。

ステップ 4: 機械学習変換の品質を推定する

次に、機械学習変換の品質を推定できます。品質は、どれだけのラベル付けを実施したかに応じて異なります。品質の推定の詳細については、「品質の推定」を参照してください。

1. AWS Glue コンソールにある [データ統合と ETL] のナビゲーションペインで、[データ分類ツール] > [レコードのマッチング] の順に選択します。
2. demo-xform-dblp-acm 変換を選択し、[Estimate quality (品質の推定)] タブをクリックします。このタブには、変換の現在の品質の推定 (使用可能な場合) が表示されます。
3. [Estimate quality (品質の推定)] を選択して、変換の品質を推定するためのタスクを開始します。品質の推定の精度はソースデータのラベル付けに基づきます。
4. [履歴] タブに移動します。このペインには、変換のタスク実行が一覧表示されます。これには、品質の推定タスクも含まれます。実行の詳細を参照するには、[ログ] を選択します。実行が完了したときの実行ステータスが [成功] になっていることを確認します。

ステップ 5: 機械学習変換でジョブを追加して実行する

このステップでは、機械学習変換を使用して AWS Glue にジョブを追加して実行します。変換 demo-xform-dblp-acm は、[Ready for use (使用可能)] である場合に、ETL ジョブで使用できます。

1. AWS Glue コンソールのナビゲーションペインで、[ジョブ] を選択します。
2. [Add job (ジョブの追加)] を選択し、ウィザードの手順に従い、生成されたスクリプトを使用して ETL Spark ジョブを作成します。変換のプロパティ値として以下を選択します。
   a. [名前] で、このチュートリアルのサンプルジョブである demo-etl-dblp-acm を選択します。
   b. [IAM role] (IAM ロール) に、Amazon S3 ソースデータ、ラベリングファイル、および AWS Glue API オペレーションへのアクセス許可を持つ IAM ロールを選択します。詳細については、AWS Glue デベロッパーガイドの「Create an IAM Role for AWS Glue」を参照してください。
   c. [ETL 言語] で、[Scala] を選択します。これは ETL スクリプトのプログラミング言語です。
   d. [スクリプトファイル名] で、[demo-etl-dblp-acm] を選択します。これは Scala スクリプトのファイル名です (ジョブ名と同じです)。
   e. [データソース] で、[dblp_acm_records_csv] を選択します。選択したデータソースは、機械学習変換のデータソーススキーマと一致する必要があります。
   f. [Transform type (変換タイプ)] で、[Find matching records (一致するレコードの検索)] を選択し、機械学習変換を使用してジョブを作成します。
   g. [Remove duplicate records (重複するレコードの削除)] をオフにします。重複するレコードを削除しない理由は、書き込まれる出力レコードに別の match_id フィールドが追加されるためです。

h. [変換] で、ジョブによって使用される機械学習変換である demo-xform-dblp-acm を選択します。

i. [データターゲットでテーブルを作成する] で、次のプロパティを使用してテーブルを作成することを選択します。

- [Data store type] (データストアのタイプ) – **Amazon S3**
- [Format] (形式) – **CSV**
- [Compression type] (圧縮タイプ) – **None**
- [Target path] (ターゲットパス) – ジョブの出力が書き込まれる Amazon S3 パス (現在のコンソールの AWS リージョン)

3. [ジョブを保存してスクリプトを編集する] を選択して、スクリプトエディタページを表示します。

4. スクリプトを編集し、ターゲットパスへのジョブ出力を単一のパーティションファイルに書き込ませるためのステートメントを追加します。このステートメントは、FindMatches 変換を実行するステートメントの直後に追加します。このステートメントは次のようになります。

```
val single_partition = findmatches1.repartition(1)
```

出力を.writeDynamicFrame(single_partion) として書き込むには、.writeDynamicFrame(findmatches1) ステートメントを変更する必要があります。

5. スクリプトを編集したら、[保存] を選択します。変更後のスクリプトは次のコードのようになりますが、使用環境向けにカスタマイズされます。

```
import com.amazonaws.services.glue.GlueContext
import com.amazonaws.services.glue.errors.CallSite
import com.amazonaws.services.glue.ml.FindMatches
import com.amazonaws.services.glue.util.GlueArgParser
import com.amazonaws.services.glue.util.Job
import com.amazonaws.services.glue.util.JsonOptions
import org.apache.spark.SparkContext
import scala.collection.JavaConverters._

object GlueApp {
  def main(sysArgs: Array[String]) {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)
    // @params: [JOB_NAME]
```

```
    val args = GlueArgParser.getResolvedOptions(sysArgs, Seq("JOB_NAME").toArray)
    Job.init(args("JOB_NAME"), glueContext, args.asJava)
    // @type: DataSource
    // @args: [database = "demo-db-dblp-acm", table_name = "dblp_acm_records_csv",
 transformation_ctx = "datasource0"]
    // @return: datasource0
    // @inputs: []
    val datasource0 = glueContext.getCatalogSource(database = "demo-db-dblp-acm",
 tableName = "dblp_acm_records_csv", redshiftTmpDir = "", transformationContext =
 "datasource0").getDynamicFrame()
    // @type: FindMatches
    // @args: [transformId = "tfm-123456789012", emitFusion = false,
 survivorComparisonField = "<primary_id>", transformation_ctx = "findmatches1"]
    // @return: findmatches1
    // @inputs: [frame = datasource0]
    val findmatches1 = FindMatches.apply(frame = datasource0, transformId
 = "tfm-123456789012", transformationContext = "findmatches1",
 computeMatchConfidenceScores = true)


    // Repartition the previous DynamicFrame into a single partition.
    val single_partition = findmatches1.repartition(1)


    // @type: DataSink
    // @args: [connection_type = "s3", connection_options = {"path": "s3://aws-
glue-ml-transforms-data/sal"}, format = "csv", transformation_ctx = "datasink2"]
    // @return: datasink2
    // @inputs: [frame = findmatches1]
    val datasink2 = glueContext.getSinkWithFormat(connectionType =
 "s3", options = JsonOptions("""{"path": "s3://aws-glue-ml-transforms-
data/sal"}"""), transformationContext = "datasink2", format =
 "csv").writeDynamicFrame(single_partition)
    Job.commit()
  }
}
```

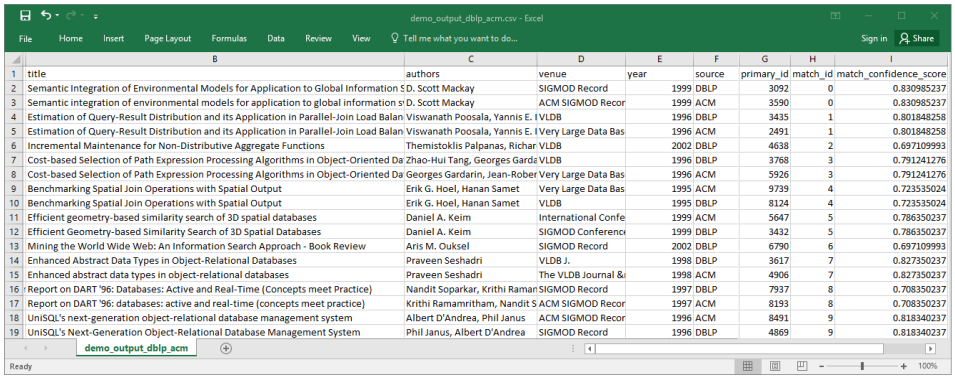
6. [ジョブの実行] を選択してジョブの実行を開始します。ジョブリストでジョブのステータスを確認します。ジョブが完了すると、[ML 変換] の [履歴] タブに、[ETL ジョブ] タイプの新しい [実行 ID] 行が追加されます。
7. [ジョブ]、[履歴] タブの順に移動します。このペインには、ジョブの実行が一覧表示されます。実行の詳細を参照するには、[ログ] を選択します。実行が完了したときの実行ステータスが [成功] になっていることを確認します。

ステップ 6: Amazon S3 の出力データを確認する

このステップでは、ジョブの追加時に選択した Amazon S3 バケットのジョブ実行の出力を確認します。出力ファイルをローカルマシンにダウンロードして、一致するレコードが識別されていることを確認します。

1. https://console.aws.amazon.com/s3/ で Amazon S3 コンソールを開きます。
2. ジョブ `demo-etl-dblp-acm` のターゲット出力ファイルをダウンロードします。このファイルをスプレッドシートアプリケーションで開きます (ファイルを正常に開くには、必要に応じてファイル拡張子 `.csv` を追加します)。

   次の図は、Microsoft Excel で開いた出力の抜粋を示しています。



	title	authors	venue	year	source	primary_id	match_id	match_confidence_score
2	Semantic Integration of Environmental Models for Application to Global Information S	D. Scott Mackay	SIGMOD Record	1999	DBLP	3092	0	0.830985237
3	Semantic integration of environmental models for application to global information s	D. Scott Mackay	ACM SIGMOD Recor	1999	ACM	3590	0	0.830985237
4	Estimation of Query-Result Distribution and its Application in Parallel-Join Load Balan	Viswanath Poosala, Yannis E. I	VLDB	1996	DBLP	3435	1	0.801848258
5	Estimation of Query-Result Distribution and its Application in Parallel-Join Load Balan	Viswanath Poosala, Yannis E. I	Very Large Data Bas	1996	ACM	2491	1	0.801848258
6	Incremental Maintenance for Non-Distributive Aggregate Functions	Themistoklis Palpanas, Richar	VLDB	2002	DBLP	4638	2	0.697109993
7	Cost-based Selection of Path Expression Processing Algorithms in Object-Oriented Da	Zhao-Hui Tang, Georges Garda	VLDB	1996	DBLP	3768	3	0.791241276
8	Cost-based Selection of Path Expression Processing Algorithms in Object-Oriented Da	Georges Gardarin, Jean-Rober	Very Large Data Bas	1996	ACM	5926	3	0.791241276
9	Benchmarking Spatial Join Operations with Spatial Output	Erik G. Hoel, Hanan Samet	Very Large Data Bas	1995	ACM	9739	4	0.723535024
10	Benchmarking Spatial Join Operations with Spatial Output	Erik G. Hoel, Hanan Samet	VLDB	1995	DBLP	8124	4	0.723535024
11	Efficient geometry-based similarity search of 3D spatial databases	Daniel A. Keim	International Confe	1999	ACM	5647	5	0.786350237
12	Efficient Geometry-based Similarity Search of 3D Spatial Databases	Daniel A. Keim	SIGMOD Conferenc	1999	DBLP	3432	5	0.786350237
13	Mining the World Wide Web: An Information Search Approach - Book Review	Aris M. Ouksel	SIGMOD Record	2002	DBLP	6790	6	0.697109993
14	Enhanced Abstract Data Types in Object-Relational Databases	Praveen Seshadri	VLDB J.	1998	DBLP	3617	7	0.827350237
15	Enhanced abstract data types in object-relational databases	Praveen Seshadri	The VLDB Journal &	1998	ACM	4906	7	0.827350237
16	Report on DART '96: Databases: Active and Real-Time (Concepts meet Practice)	Nandit Soparkar, Krithi Raman	SIGMOD Record	1997	DBLP	7937	8	0.708350237
17	Report on DART '96: databases: active and real-time (concepts meet practice)	Krithi Ramamritham, Nandit S	ACM SIGMOD Recor	1997	ACM	8193	8	0.708350237
18	UniSQL's next-generation object-relational database management system	Albert D'Andrea, Phil Janus	ACM SIGMOD Recor	1996	ACM	8491	9	0.818340237
19	UniSQL's Next-Generation Object-Relational Database Management System	Phil Janus, Albert D'Andrea	SIGMOD Record	1996	DBLP	4869	9	0.818340237

   データソースおよびターゲットファイルの両方に 4,911 個のレコードがあります。ただし、`Find matches` 変換には、出力内の一致するレコードを識別するために `match_id` という別の列が追加されています。同じ `match_id` の行は、一致するレコードと見なされます。`match_confidence_score` は 0 ~ 1 の間の数値で、`Find matches` によって検出された次の値で一致の品質を推定します。
3. 出力ファイルを `match_id` で並べ替えると、どのレコードが一致しているか簡単にわかります。他の列の値と比較して、`Find matches` 変換の結果が適切であるかどうかを判断します。適切でない場合は、さらにラベルを追加することで、引き続き変換をトレーニングします。

   ファイルをソートするために別のフィールド (`title` など) を使用して、同じようなタイトルのレコードが同じ `match_id` を持っているかどうかを確認することもできます。

インクリメンタルマッチを検索する。

FindMatches 変換を使用すると、レコードに共通の一意の識別子がなく、正確に一致するフィールドがない場合でも、データセット内の重複レコードまたは一致するレコードを識別できます。検索マッチ変換の最初のリリースでは、単一のデータセット内で一致するレコードが識別されました。データ

セットに新しいデータを追加する場合、既存のクリーンなデータセットとマージし、マージされた完全なデータセットに対してマッチングを再実行する必要があります。

インクリメンタルマッチング機能を使用すると、既存の一致したデータセットに対するインクリメンタルレコードとの照合が簡単になります。既存の顧客データセットと見込み客のデータを照合するとします。差分一致機能を使用すると、結果を単一のデータベースまたはテーブルにマージすることで、数十万の新規プロスペクトを既存のプロスペクトと顧客の既存のデータベースと照合できる柔軟性が得られます。新規データセットと既存のデータセット間のみ照合を行うことで、差分一致の検索最適化によって計算時間が短縮され、コストも削減されます。

インクリメンタルマッチングの使用法は、チュートリアル: AWS Glue で機械学習変換を作成する で説明されているように、「一致を検索」と似ています。このトピックでは、インクリメンタルマッチングの相違点のみについて説明します。

詳細については、インクリメンタルデータマッチングのブログ記事を参照してください。

インクリメンタルマッチングジョブの実行

次の手順では、次の様に仮定します。

- 既存のデータセット内をクロールして first_records テーブルに到達しました。first_records データセットは、一致したデータセット、または一致したジョブの出力であるはずです。
- AWS Glue バージョン 2.0 を使用して、一致検索変換の作成とトレーニングが完了しました。このバージョンが唯一インクリメンタルマッチを AWS Glue サポートしています。
- ETL 言語は Scala です。Python もサポートされていることに注意してください。
- 既に生成されたモデルは `demo-xform` と呼ばれます。

1. インクリメンタルデータセットを、テーブル second records にクロールする。
2. AWS Glue コンソールのナビゲーションペインで、[ジョブ] を選択します。
3. [Add job (ジョブの追加)] を選択し、ウィザードの手順に従い、生成されたスクリプトを使用して ETL Spark ジョブを作成します。変換のプロパティ値として以下を選択します。

    a. 名前で、デモetlを使用する場合。

    b. IAM ロール に、Amazon S3 ソースデータ、ラベリングファイル、および AWS GlueAPI オペレーションへのアクセス許可を持つ IAM ロールを選択します。

    c. [ETL 言語] で、[Scala] を選択します。

    d. [スクリプトファイル名] で、[demo-etl] を選択します。これは Scala スクリプトのファイル名です。

e. データソースはfirst recordsを選択します。選択したデータソースは、機械学習変換のデータソーススキーマと一致する必要があります。

f. [Transform type (変換タイプ)] で、[Find matching records (一致するレコードの検索)] を選択し、機械学習変換を使用してジョブを作成します。

g. インクリメンタルマッチングのオプションを選択し、データ・ソースにsecond_recordsという名前のテーブルを選択します。

h. [トランスフォーム] で、ジョブによって使用される機械学習変換である demo-xform-dblp-acm を選択します。

i. データターゲットにテーブルを作成するか、データカタログのテーブルを使用してデータターゲットを更新するを選択します。

4. [ジョブを保存してスクリプトを編集する] を選択して、スクリプトエディタページを表示します。

5. [ジョブの実行] を選択してジョブの実行を開始します。

ビジュアルジョブでの FindMatches の使用

AWS Glue Studio で FindMatches 変換を使用するには、FindMatches API を呼び出す カスタム変換ノードを使用します。カスタム変換の使用に関する詳細については、「カスタム変換を作成する」を参照してください。

> **Note**
>
> 現在、FindMatches API は、`Glue 2.0` でのみ動作します。FindMatches API を呼び出すカスタム変換を使用してジョブを実行するには、AWS Glue のバージョンが [ジョブの詳細] タブの `Glue 2.0` に表示されていることを確認してください。AWS Glue のバージョンが `Glue 2.0` でない場合、ジョブは実行時に失敗し、「'awsglueml.transforms' から 'FindMatches' という名前の変換をインポートできません」というエラーメッセージが表示されます。

前提条件

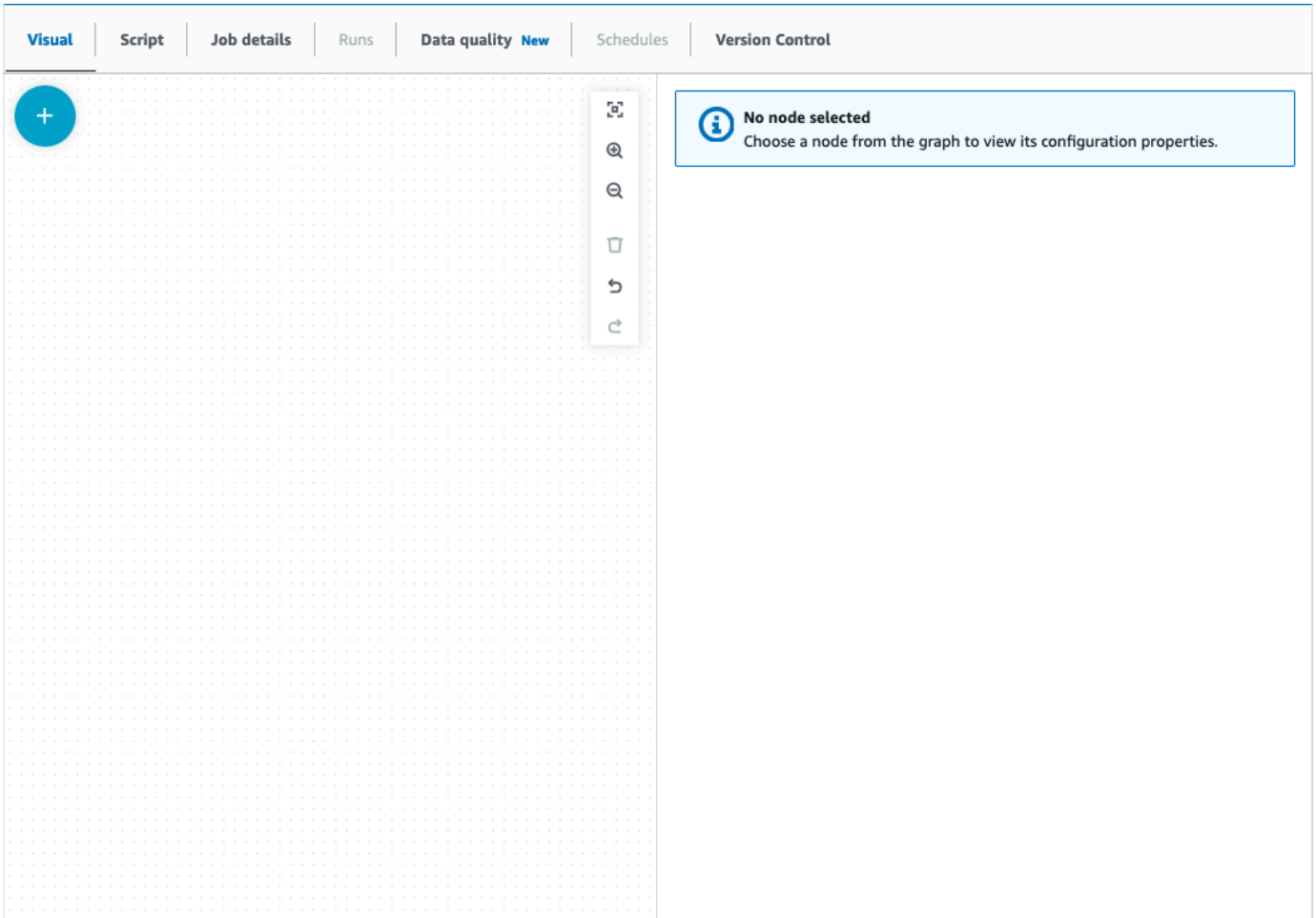
- 一致の検索 変換を使用するには、AWS Glue Studio で [https://console.aws.amazon.com/gluestudio/] コンソールを開きます。

- 機械学習変換を作成します。作成すると、transformId が生成されます。この ID は以下の手順で必要になります。機械学習変換の作成方法の詳細については、「機械学習変換の追加と編集」を参照してください。

FindMatches 変換の追加

FindMatches 変換を追加するには

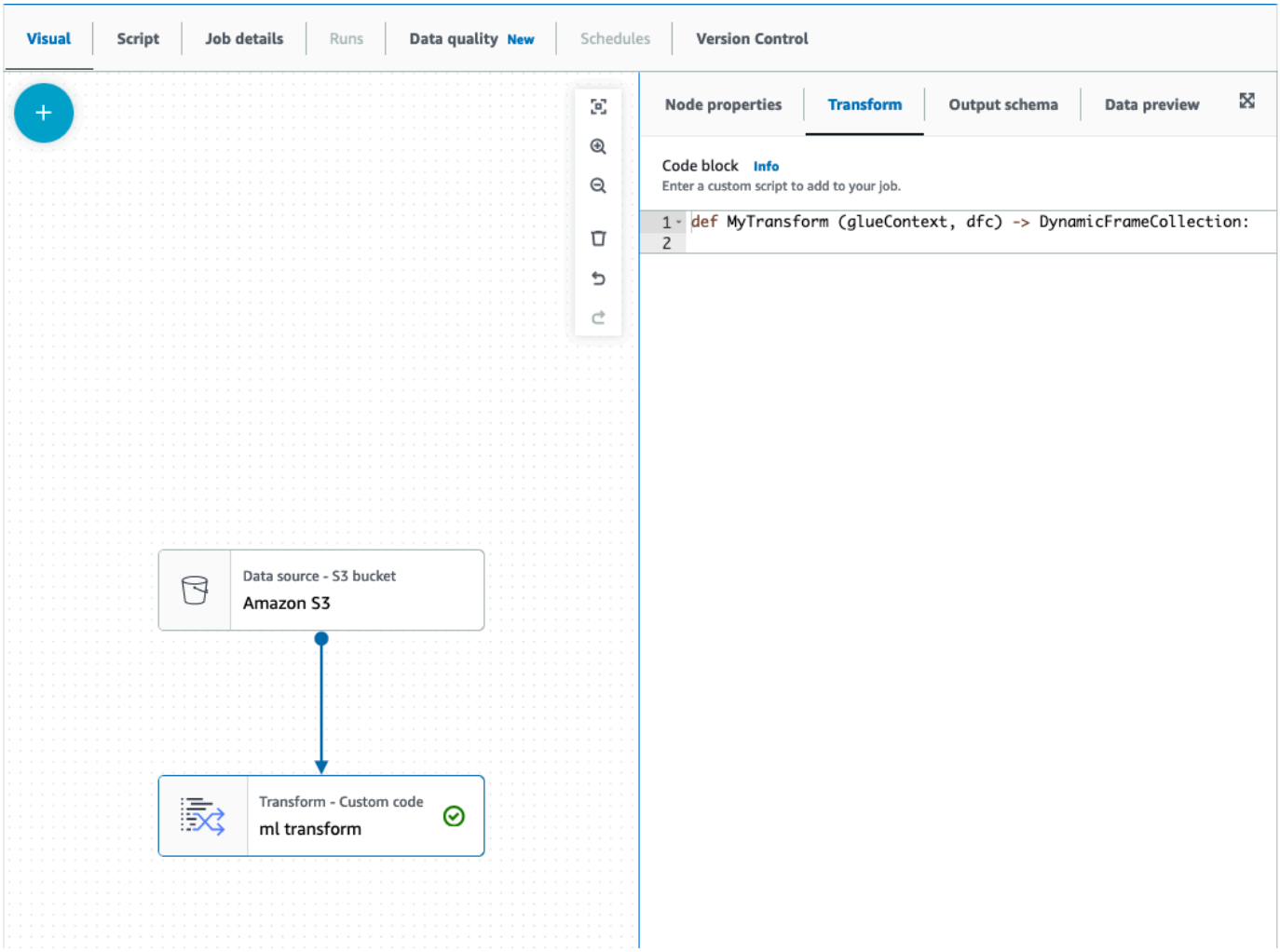
1. AWS Glue Studio ジョブエディターで、ビジュアルジョブグラフの左上隅にある十字記号をクリックして [リソース] パネルを開き、[データ] タブを選択してデータソースを選択します。これは、一致を確認するデータ ソースです。



2. データソースノードを選択し、ビジュアルジョブグラフの左上隅にある十字記号をクリックして [リソース] パネルを開き、「custom transform」を検索します。カスタム変換ノードを選択してグラフに追加します。カスタム変換はデータソースノードにリンクされています。リンクされていない場合は、カスタム変換ノードをクリックして[ノードプロパティ] タブを選択し、[ノードの親] でデータソースを選択します。

3. ビジュアルグラフの[カスタム変換]ノードをクリックし、[ノードプロパティ]タブを選択し、カスタム変換に名前を付けます。ビジュアルグラフに表示される変換名は、わかりやすい名前に変更することをお勧めします。

4. コードブロックを編集するには、[変換] タブを選択します。ここで、FindMatches API を呼び出すコードを追加できます。



コードブロックには、すぐに始めることができるようにコードが予め入力されています。予め入力されているコードを下記のテンプレートで上書きします。テンプレートには、入力できる transformId のプレースホルダーが表示されています。

```
def MyTransform (glueContext, dfc) -> DynamicFrameCollection:
    dynf = dfc.select(list(dfc.keys())[0])
    from awsglueml.transforms import FindMatches
    findmatches = FindMatches.apply(frame = dynf, transformId = "<your id>")
    return(DynamicFrameCollection({"FindMatches": findmatches}, glueContext))
```

5. ビジュアルグラフの[カスタム変換]ノードをクリックし、ビジュアルジョブグラフの左上隅にある十字記号をクリックして[リソース] パネルを開き、「Select From Collection」を検索します。コレクションには DynamicFrame が 1 つしかないため、デフォルトの選択を変更する必要はありません。
6. 続けて変換を追加したり、結果を保存したりすることができます。結果には、[一致の検索] 列が追加されました。これらの新しい列を下流の変換で参照する場合は、それらを変換出力スキーマに追加する必要があります。それを行う最も簡単な方法は、[データプレビュー] タブを選択し、[スキーマ] タブで [データプレビュースキーマの使用] を選択することです。
7. FindMatches をカスタマイズするには、「apply」メソッドに渡すパラメーターを追加できます。FindMatches クラスを参照してください。

「FindMatches」段階的な変換の追加

インクリメンタルマッチングの場合、プロセスは FindMatches 変換の追加と同じですが、次の点が異なります。

- カスタム変換の親ノードの代わりに、2 つの親ノードが必要です。
- 最初の親ノードはデータセットでなければなりません。
- 2 番目の親ノードはインクリメンタルデータセットでなければなりません。

  テンプレートコードブロックで `transformId` を`transformId` に置き換えてください。

```
def MyTransform (glueContext, dfc) -> DynamicFrameCollection:
    dfs = list(dfc.values())
    dynf = dfs[0]
    inc_dynf = dfs[1]
    from awsglueml.transforms import FindIncrementalMatches
    findmatches = FindIncrementalMatches.apply(existingFrame = dynf, incrementalFrame
 = inc_dynf,
                                    transformId = "<your id>")
    return(DynamicFrameCollection({"FindMatches": findmatches}, glueContext))
```

- オプションのパラメータについては、「FindIncrementalMatches クラス」を参照してください。

Apache Spark プログラムを AWS Glue に移行する

Apache Spark は、大規模なデータセットで実行される分散コンピューティングワークロード向けのオープンソースプラットフォームです。AWS Glue は、Spark の機能を活用して ETL に最適化されたエクスペリエンスを提供します。Spark プログラムは、AWS Glue に移行することで機能を最大限に活用できます。AWS Glue は、Amazon EMR の Apache Spark と同等のパフォーマンス向上を実現します。

Spark コードを実行する

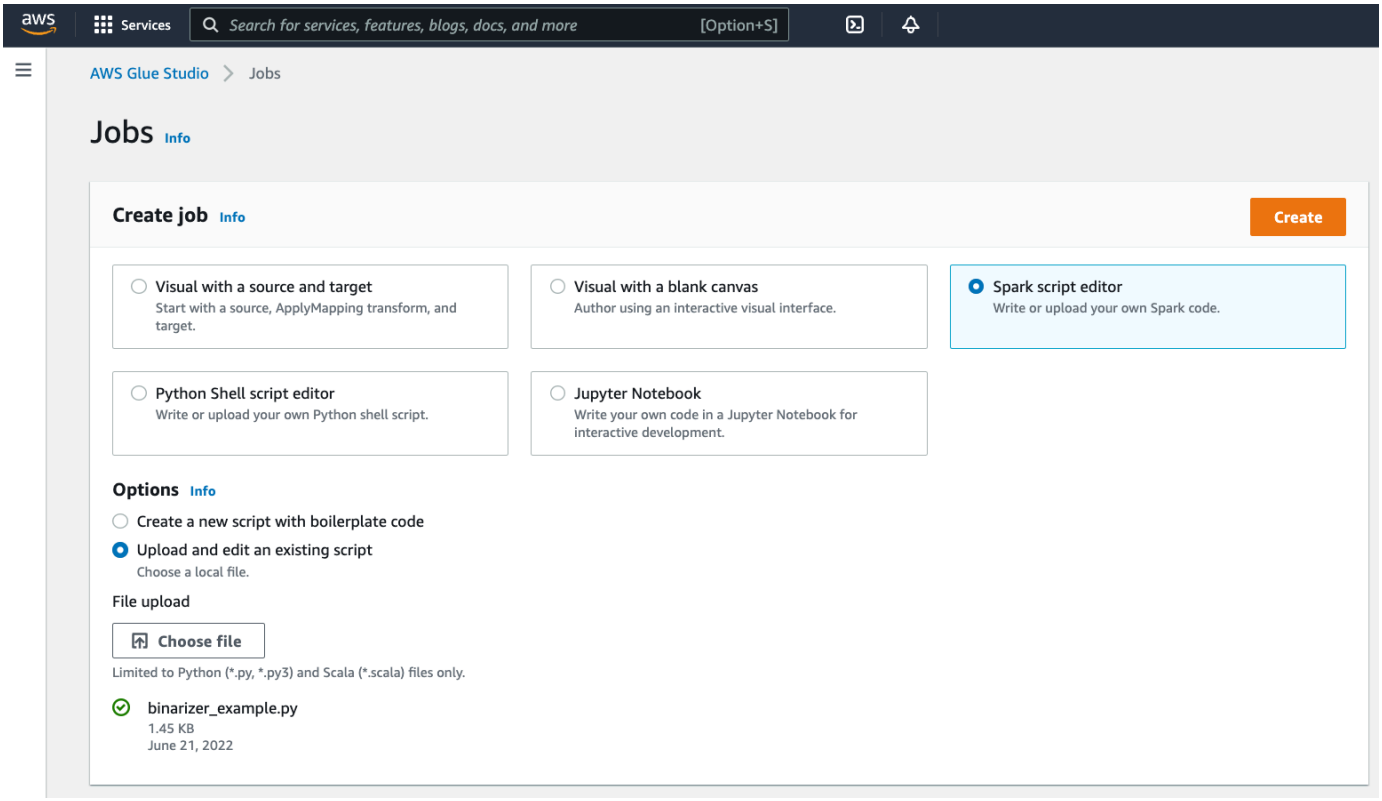
ネイティブ Spark コードは、AWS Glue 環境で追加設定なしですぐに使用できます。スクリプトは多くの場合、インタラクティブセッションに適したワークフローであるコードを繰り返し変更することによって開発されます。ただし、既存のコードの方が AWS Glue ジョブでの実行に適しています。これにより、スクリプトを実行するたびにログとメトリクスをスケジュールし、一貫して取得できます。コンソールから既存のスクリプトをアップロードして編集できます。

1. スクリプトのソースを取得します。この例では、Apache Spark リポジトリのサンプルスクリプトを使用します。バイナライザーの例
2. AWS Glue コンソールで、左側のナビゲーションペインを展開し、[ETL] > [Jobs] (ジョブ) を選択します。

   [Create job] (ジョブの作成) パネルで、[Spark script editor] (Spark スクリプトエディタ) を選択します。[Options] (オプション) セクションが表示されます。[Options] (オプション) で [Upload and edit an existing script] (既存のスクリプトのアップロードと編集) を選択します。

   [File upload] (ファイルのアップロード) セクションが表示されます。[File upload] (ファイルのアップロード) で [Choose file] (ファイルを選択) をクリックします。システムファイル選択ダイアログが表示されます。`binarizer_example.py` を保存した場所に移動してこの場所を選択し、選択を確定します。

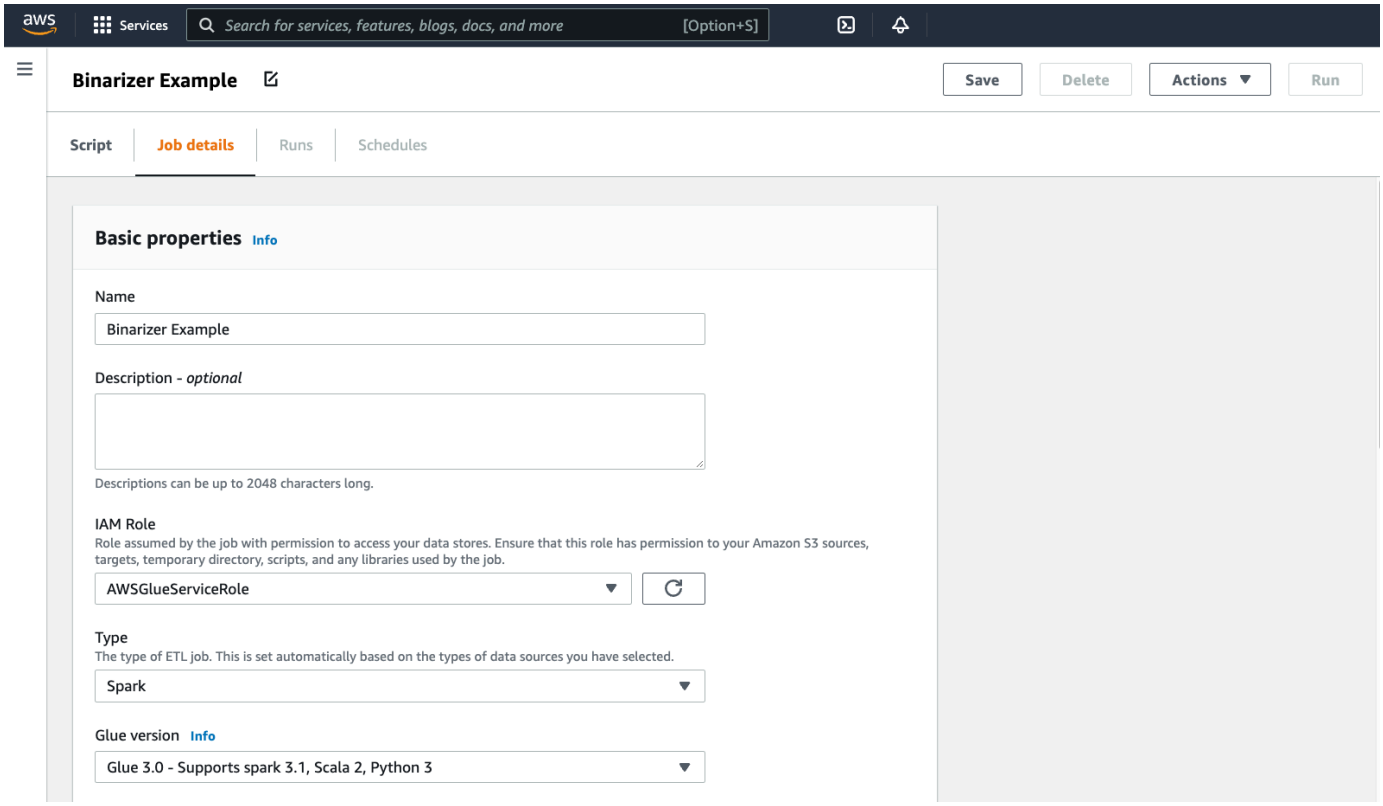
   [Create job] (ジョブの作成) パネルのヘッダーに [Create] (作成) ボタンが表示されます。このボタンをクリックします。



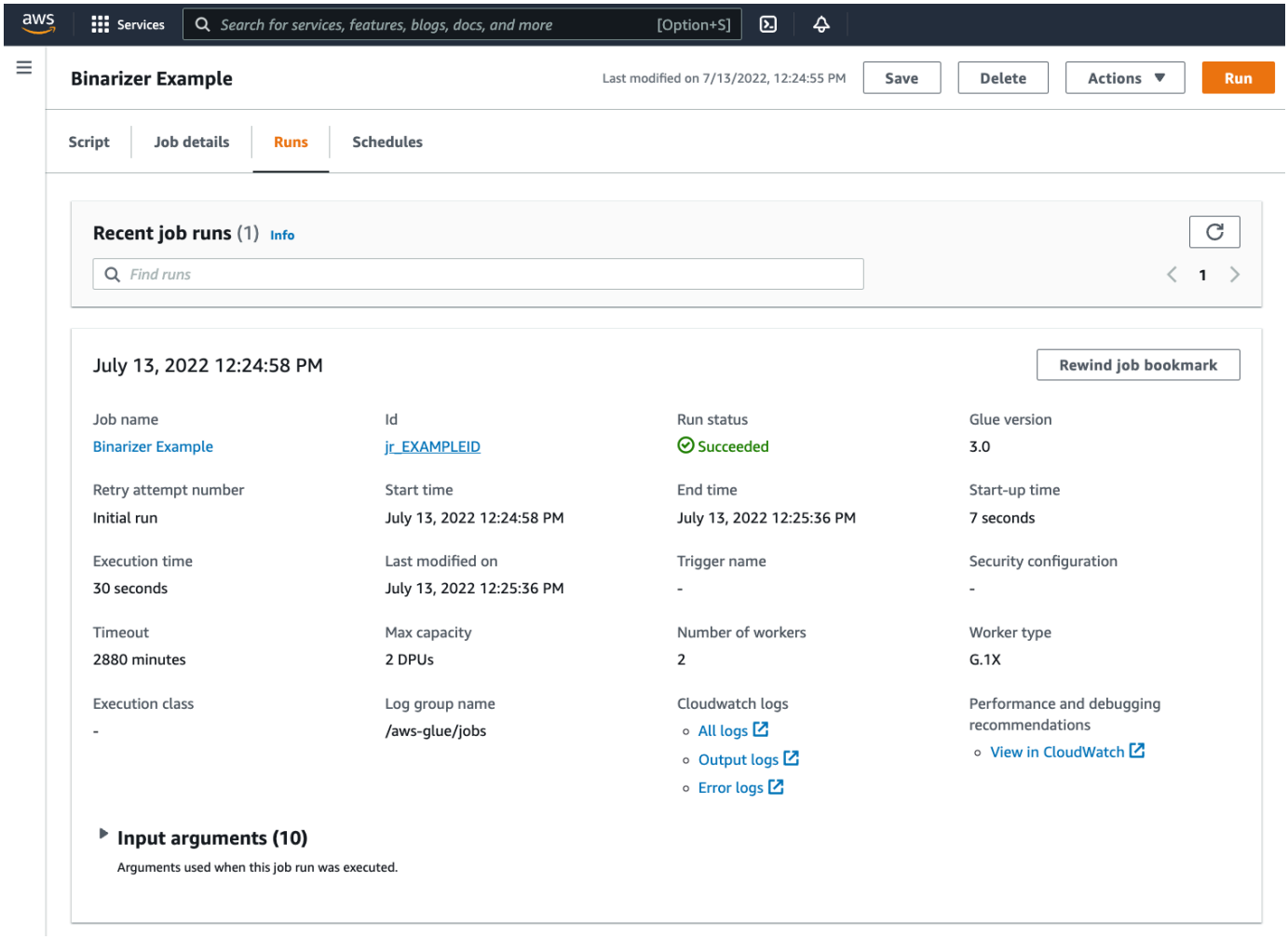
3. ブラウザがスクリプトエディタに移動します。ヘッダーで、[Job details] (ジョブの詳細) タブをクリックします。名前と IAM ロールを設定します。AWS Glue IAM ロールに関するガイダンスについては、「the section called “IAM アクセス許可のセットアップ”」を参照してください。

   必要に応じて、[Requested number of workers] (要求されたワーカー数) を 2 に、[Number of retries] (再試行回数) を 1 に設定します。これらのオプションは本番ジョブを実行する場合に役立ちますが、無効にすると、機能をテストする際のエクスペリエンスが合理化されます。

   タイトルバーで、[Save] (保存) をクリックし、続いて [Run] (実行) をクリックします。

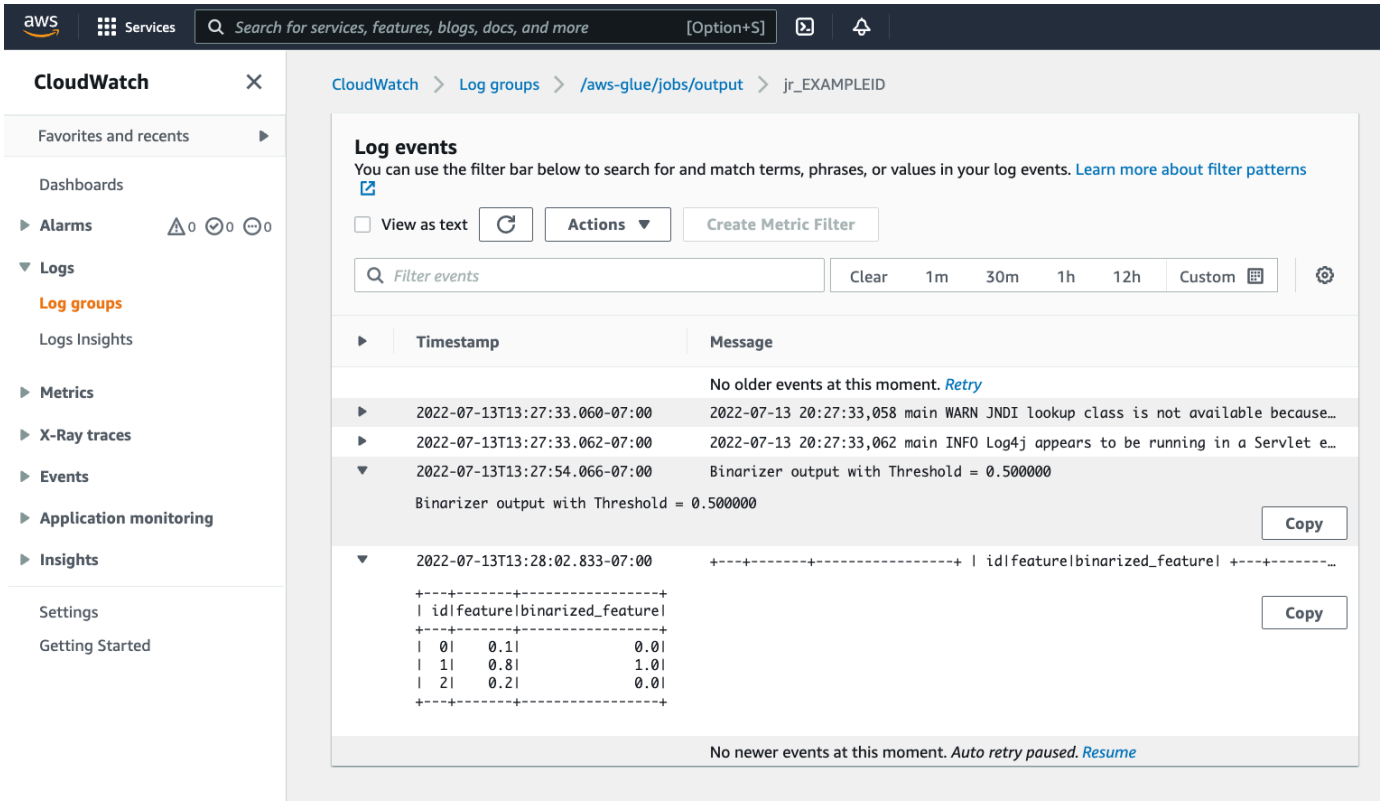


4. [Runs] (実行) タブに移動します。ジョブの実行に対応するパネルが表示されます。数分間待機した後、ページが自動的に更新され、[Run status] (実行ステータス) の下に[Succeeded] (成功) が表示されます。



5. 出力を調べて、Spark スクリプトが意図したとおりに実行されたことを確認する必要があります。この Apache Spark サンプルスクリプトは、出力ストリームに文字列を書き込む必要があります。これは、実行が成功したジョブのパネル内の [Cloudwatch Logs] の下にある [Output logs] (出力ログ) に移動して確認できます。ジョブ実行 ID をメモします。これは、`jr_`で始まる [ID] ラベルの下に生成されます。

   これにより、CloudWatch コンソールが開き、デフォルトの AWS Glue ロググループ`/aws-glue/jobs/output`のコンテンツを視覚化するように設定されます。これで、ジョブ実行 ID のログストリームのコンテンツに絞り込まれます。各ワーカーはログストリームを生成し、[Log streams] (ログストリーム) の下の行として表示されます。1 人のワーカーが要求されたコードを実行しているはずです。正しいワーカーを特定するには、すべてのログストリームを開く必要があります。適切なワーカーが見つかると、次の画像に示すように、スクリプトの出力が表示されます。



Spark プログラムの移行に必要な一般的な手順

Spark バージョンサポートの評価

AWS Glue リリースバージョンは、AWS Glue ジョブで使用できる Apache Spark と Python のバージョンを決定します。AWS Glue バージョンとそのサポート対象については、「the section called “AWS Glue バージョン”」をご覧ください。特定の AWS Glue 機能にアクセスするには、Spark プログラムを新しいバージョンの Spark と互換性があるように更新する必要がある場合があります。

サードパーティライブラリを含める

既存の Spark プログラムの多くは、プライベートアーティファクトとパブリックアーティファクトの両方に依存しています。AWS Glue は、Scala ジョブの JAR スタイルの依存関係と、Python ジョブのホイールとソース Pure-Python 依存関係をサポートします。

Python-Python の依存関係については、「the section called “Python ライブラリ”」を参照してください。

一般的な Python の依存関係は、一般的に要求される Pandas ライブラリなどを含めて AWS Glue 環境で提供されています。依存関係は AWS Glue バージョン 2.0 以降に含まれています。提供されるモジュールの詳細については、「the section called “AWS Glue で提供済みの Python モジュール”」

を参照してください。デフォルトで含まれている依存関係の別のバージョンをジョブに提供する必要がある場合は、`--additional-python-modules` を使用します。ジョブ引数の詳細については、「the section called “ジョブのパラメータ”」を参照してください。

追加の Python 依存関係は、`--extra-py-files` ジョブ引数で指定できます。Spark プログラムからジョブを移行する場合、このパラメータは良いオプションです。PySpark の `--py-files` フラグと機能的に同等で、同じ制限が適用されるからです。`--extra-py-files` パラメータの詳細については、「the section called “PySpark のネイティブ機能を使用して Python ファイルを含める”」を参照してください。

新しいジョブについては、`--additional-python-modules` 引数を使用してPython の依存関係を管理できます。この引数を使用すると、より徹底した依存関係管理が可能になります。このパラメータは、Amazon Linux 2 と互換性のあるネイティブコードバインディングを含むホイールスタイルの依存関係をサポートします。

Scala

追加の Scala 依存関係は、`--extra-jars` ジョブ引数で指定できます。依存関係は Amazon S3 でホストされている必要があり、引数の値は、スペースを含まない Amazon S3 パスのカンマ区切りリストである必要があります。依存関係をホストして設定する前に再バンドルすると、設定を管理しやすくなる場合があります。AWS GlueJAR 依存関係には Java バイトコードが含まれており、任意の JVM 言語から生成できます。Java などの他の JVM 言語を使用して、カスタムの依存関係を記述できます。

データソース認証情報の管理

既存の Spark プログラムには、データソースからデータを取得するための複雑な構成またはカスタム構成が付属している場合があります。一般的なデータソース認証フローは、AWS Glue 接続でサポートされます。AWS Glue 接続の詳細については、「データへの接続」を参照してください。

AWS Glue 接続は、主に次の 2 つの方法 (ライブラリへのメソッド呼び出しと、AWS コンソールで [Additional network connection] (追加のネットワーク接続) を設定する) を通じて、ジョブをさまざまなタイプのデータストアに接続しやすくします。また、接続から情報を取得するため、ジョブ内からの AWS SDK を呼び出すこともできます。

メソッドの呼び出し – AWS Glue 接続は、AWS Glue データカタログと密接に統合されています。これは、データセットに関する情報と、この情報が反映された AWS Glue 接続の操作に使用できるメソッドの管理に利用できます。JDBC 接続で再利用したい既存の認証設定がある場合は、`GlueContext` の `extract_jdbc_conf` メソッドを介して AWS Glue 接続の設定にアクセスできます。詳細については、「the section called “extract_jdbc_conf”」を参照してください。

コンソール設定 – AWS Glue ジョブは、関連する AWS Glue 接続を使用して Amazon VPC サブネットへの接続を設定します。セキュリティ資料を直接管理している場合は、AWSコンソールで、NETWORK タイプ [Additional network connection] (追加のネットワーク接続) を指定してルーティングを設定します。AWS Glue 接続 API の詳細については、「the section called “接続”」を参照してください

Spark プログラムにカスタム認証フローまたは一般的でない認証フローがある場合、セキュリティ資料をハンズオンで管理する必要があるかもしれません。AWS Glue 接続が適切でない場合、セキュリティ資料をシークレットマネージャーで安全にホストし、ジョブで提供される boto3 または AWS SDK を介してアクセスします。

Apache Spark の設定

複雑な移行では、ワークロードに対応するために Spark 設定が変更されることがよくあります。Apache Spark の最新バージョンでは、ランタイム設定を SparkSession で設定できます。AWS Glue3.0 以降のジョブでは SparkSession が提供され、ランタイム設定の変更ができます。Apache Spark 設定。Spark のチューニングは複雑で、AWS Glue は、すべての Spark 設定に対するサポートを保証するものではありません。移行に相当する Spark レベルの設定が必要な場合は、サポートにお問い合わせください。

カスタム設定のセット

移行された Spark プログラムは、カスタム設定を選択するように設計されている場合があります。AWS Glue はジョブ引数を使用して、ジョブとジョブの実行レベルで構成を設定できます。ジョブ引数の詳細については、「the section called “ジョブのパラメータ”」を参照してください。ライブラリを通じて、ジョブのコンテキスト内でジョブの引数にアクセスできます。AWS Glue は、ジョブに設定された引数とジョブ実行時に設定された引数の間で一貫したビューを提供するユーティリティ関数を提供します。Python の「the section called “getResolvedOptions”」、Scala の「the section called “GlueArgParser”」を参照してください。

Java コードの移行

「the section called “サードパーティライブラリ”」で説明されているように、Java や Scala などの JVM 言語で生成されたクラスを依存関係に含めることができます。依存関係には main メソッドが含まれます。AWS Glue Scala ジョブのエントリポイントとして、依存関係内に main メソッドを使用できます。これにより、Java に main メソッドを記述するか、独自のライブラリ標準にパッケージ化された main メソッドを再使用できます。

依存関係から main メソッドを使用するには、次のいずれかの方法を実行します: デフォルトの GlueApp オブジェクトを提供する編集ペインの内容をクリアします。依存関係内のクラスの完全修

飾名を、キー --class を使用したジョブ引数として指定します。これで、ジョブ実行をトリガーできるようになります。

AWS Glue が main メソッドに引き渡す引数の順序や構造は設定できません。既存のコードで、AWS Glue の設定を読み取る必要がある場合、以前のコードとの非互換性を引き起こす可能性があります。getResolvedOptions を使用していると、このメソッドを呼び出すのに適した条件もありません。AWS Glue で生成されたメインメソッドから依存関係を直接呼び出すことをおすすめします。この例については、以下の AWS Glue ETL スクリプトをご覧ください。

```
import com.amazonaws.services.glue.util.GlueArgParser

object GlueApp {
  def main(sysArgs: Array[String]) {
    val args = GlueArgParser.getResolvedOptions(sysArgs, Seq("JOB_NAME").toArray)

    // Invoke static method from JAR. Pass some sample arguments as a String[], one
 defined inline and one taken from the job arguments, using getResolvedOptions
    com.mycompany.myproject.MyClass.myStaticPublicMethod(Array("string parameter1",
 args("JOB_NAME")))

    // Alternatively, invoke a non-static public method.
    (new com.mycompany.myproject.MyClass).someMethod()
  }
}
```

AWS Glue での Ray ジョブの使用

このセクションでは、AWS Glue for Ray のジョブを使用する方法について説明します。AWS Glue for Ray のスクリプトを記述する方法の詳細については、「the section called “AWS Glue for Ray”」セクションを参照してください。

トピック

- AWS Glue for Ray の使用を開始する
- サポートされている Ray のランタイム環境
- Ray ジョブのワーカーの考慮
- Ray ジョブでジョブパラメータを使用する
- メトリクスによる Ray ジョブのモニタリング

AWS Glue for Ray の使用を開始する

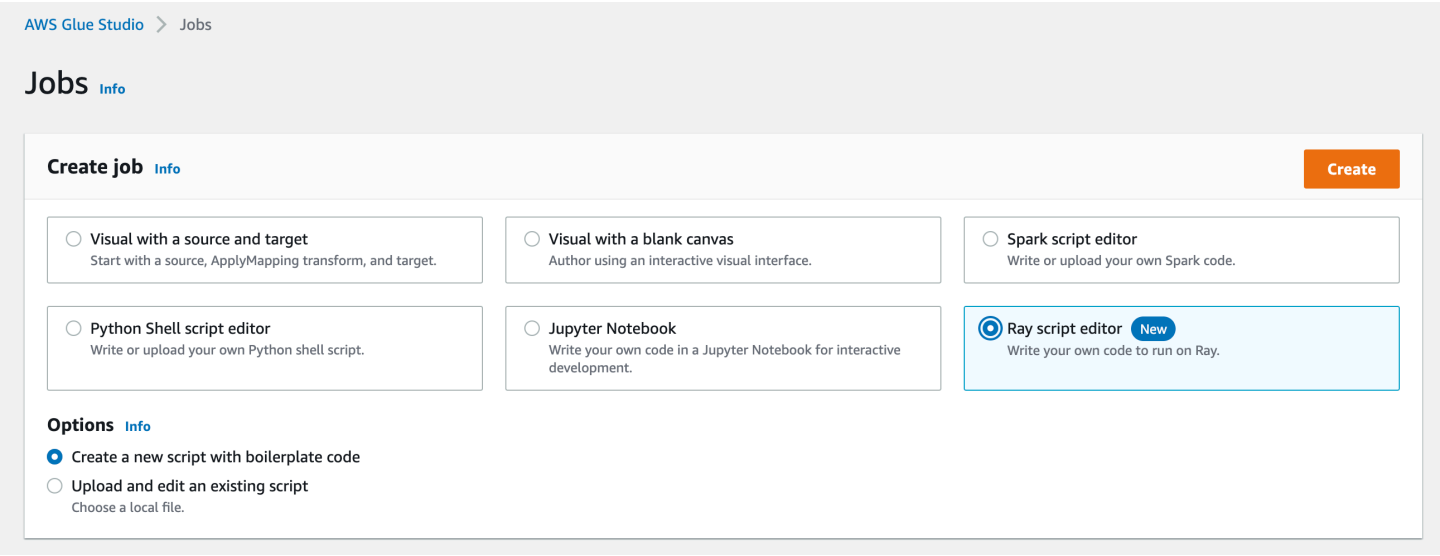
AWS Glue for Ray を使用するには、AWS Glue for Spark で使用しているものと同じ、AWS Glue ジョブとインタラクティブセッションを使用します。AWS Glue ジョブは、同じスクリプトを一定の頻度で繰り返し実行するように設計されていますが、インタラクティブセッションは、プロビジョニングされた同じリソースに対して、コードのスニペットを順番に実行するように設計されています。

AWS Glue ETL と Ray は下部構造が異なるため、スクリプトではさまざまなツール、機能、設定にアクセスすることができます。AWS Glue が管理する新しい計算フレームワークとして、Ray は異なるアーキテクチャを持ちます。また、その機能を説明するボキャブラリーも異なります。詳細については、Ray のドキュメントの「Architecture Whitepapers」(アーキテクチャホワイトペーパー) を参照してください。

> **Note**
>
> AWS Glue for Ray は、米国東部 (バージニア北部)、米国東部 (オハイオ)、米国西部 (オレゴン)、アジアパシフィック (東京)、欧州 (アイルランド) でご利用いただけます。

AWS Glue Studio コンソールでの Ray ジョブ

AWS Glue Studio コンソールの [ジョブ] ページでは、AWS Glue Studio でジョブを作成する際に、新たに [Ray script editor] を選択できます。コンソールに Ray ジョブを作成するときはこちらを選択します。ジョブとその使用方法についての詳細は、「AWS Glue Studio でビジュアル ETL ジョブを作成する」を参照してください。



AWS CLI および SDK での Ray ジョブ

AWS CLI での Ray ジョブは、他のジョブと同じ SDK アクションとパラメータを使用します。AWS Glue for Ray では、特定のパラメータに新しい値が導入されています。ジョブの API の詳細については、「the section called “ジョブ”」を参照してください。

サポートされている Ray のランタイム環境

Spark ジョブでは、`GlueVersion` によって、AWS Glue for Spark ジョブで利用できる Apache Spark と Python のバージョンが決まります。Python のバージョンは、Spark タイプのジョブでサポートされているバージョンを示します。これは、Ray のランタイム環境の設定方法とは異なります。

Ray ジョブの場合、`GlueVersion` を `4.0` 以降に設定する必要があります。ただし、Ray ジョブで使用できる Ray、Python、その他のライブラリバージョンは、ジョブ定義の `Runtime` フィールドにより決まります。

`Ray2.4` ランタイム環境は、リリースから最低 6 か月間は使用可能です。Ray は進化のスピードが速いため、今後のランタイム環境のリリースを通じてその更新や改良を統合していけます。

有効な値: `Ray2.4`

ランタイム値	Ray と Python のバージョン
`Ray2.4` (AWS Glue 4.0 以降)	Ray 2.4.0 Python 3.9

追加情報

- AWS Glue on Ray のリリースに付属のリリースノートについては、「the section called “AWS Glue バージョン”」を参照してください。
- ランタイム環境で提供される Python ライブラリについては、「the section called “Ray ジョブで提供されるモジュール”」を参照してください。

Ray ジョブのワーカーの考慮

AWS Glue は、Ray ジョブを Graviton ベースの新しい EC2 ワーカータイプで実行します。このタイプは Ray ジョブでのみで利用できるものです。これらのワーカーを、Ray の設計の根拠であるワークロードに合わせて適切にプロビジョニングするため、コンピューティングリソースとメモリリソースの比率がほとんどのワーカーとは異なっています。これらのリソースを考慮して、標準データ処理ユニット (DPU) ではなく、メモリ最適化データ処理ユニット (M-DPU) が使用されています。

- 1 基の M-DPU は 4 基の vCPU と 32 GB のメモリに相当します。
- 1 基の DPU は 4 基の vCPU と 16 GB のメモリに相当します。DPU は、AWS Glue with Spark のジョブと対応するワーカーのリソースを考慮して、使用されています。

Ray ジョブでは現在、`Z.2X` という 1 つのワーカータイプしか使用できません。`Z.2X` ワーカーは 2 つの M-DPU (8 基の vCPU、64 GB のメモリ) にマップされ、128 GB のディスク容量を有しています。`Z.2X` マシンには 8 つの Ray ワーカーがあります (vCPU につき 1 つ)。

1 つのアカウントで同時に使用できる M-DPU の数は、サービスクォータの対象となります。AWS Glue のアカウント制限についての詳細は、「AWS Glue エンドポイントとクォータ」を参照してください。

Ray ジョブで使用できるワーカーノードの数は、ジョブ定義で `--number-of-workers` (`NumberOfWorkers`) のように指定します。ジョブ API の Ray 値の詳細については、「the section called “ジョブ”」を参照してください。

また、Ray ジョブが割り当てるワーカーの最小数は、`--min-workers` ジョブパラメータを使用して指定できます。ジョブパラメータについては、「the section called “リファレンス”」を参照してください。

Ray ジョブでジョブパラメータを使用する

AWS Glue Ray ジョブの引数は、AWS Glue for Spark ジョブの引数を設定するときと同じ方法で設定します。AWS Glue API の詳細については、「the section called “ジョブ”」を参照してください。AWS Glue Ray ジョブはさまざまな引数を用いて設定できます。これらはこちらのリファレンスに記載されています。また、独自の引数を指定することもできます。

ジョブは、Job Parameters (ジョブパラメータ) 見出しの Job details (ジョブの詳細) タブからコンソールで設定できます。また、ジョブに `DefaultArguments`、またはジョブ実行に `Arguments` を設定することで、AWS CLI を使用してジョブを設定することもできます。デフォルトの引数とジョブパラメータは、複数回実行してもジョブに対して有効です。

以下は、特殊なパラメータをセットするために --arguments を使用する、ジョブ実行の構文例です。

```
$ aws glue start-job-run --job-name "CSV to CSV" --arguments='--scriptLocation="s3://
my_glue/libraries/test_lib.py",--test-environment="true"'
```

引数を設定すると、環境変数を使用して Ray ジョブ内からジョブパラメータにアクセスできます。そうすることで、ジョブを実行するたびにジョブの設定を行うことができます。環境変数の名前は、先頭の -- を除いたジョブ引数の名前になります。

例えば、上記の例では、変数の名前は scriptLocation および test-environment になります。次に、標準ライブラリで利用できる方法を使用して引数を取得します (test_environment = os.environ.get('test-environment'))。Python で環境変数にアクセスする方法の詳細については、Python のドキュメントの「os module」を参照してください。

Ray ジョブがログを生成する方法を設定する

デフォルトでは、Ray ジョブは CloudWatch と Amazon S3 に送信されるログとメトリクスを生成します。--logging_configuration パラメータを使用してログの生成方法を変更できます。現在のところ、これを使用して Ray ジョブがさまざまなタイプのログを生成しないようにすることができます。このパラメータは、変更したいログや動作に対応するキーを持つ JSON オブジェクトを取ります。次のキーがサポートされています。

- CLOUDWATCH_METRICS — ジョブの状態を視覚化するために使用できる CloudWatch メトリクスシリーズを設定します。メトリクスの詳細については、「the section called “Ray ジョブメトリクス”」を参照してください。
- CLOUDWATCH_LOGS — ジョブの実行ステータスに関する Ray アプリケーションレベルの詳細を提供する CloudWatch ログを設定します。ログの詳細については、the section called “Ray エラーをトラブルシューティングする” を参照してください。
- S3 — AWS Glue が Amazon S3 に何を書き込むかを設定します。主として CloudWatch ログと同様の情報ですが、ログストリームではなくファイルとして書き込まれます。

Ray ログ記録の動作を無効にするには、値 {\"IS_ENABLED\": \"False\"} を指定します。例えば、CloudWatch メトリクスと CloudWatch ログを無効にするには、以下の設定を指定します。

```
"--logging_configuration": "{\"CLOUDWATCH_METRICS\": {\"IS_ENABLED\": \"False\"},
 \"CLOUDWATCH_LOGS\": {\"IS_ENABLED\": \"False\"}}"
```

リファレンス

Ray ジョブは、Ray ジョブおよびジョブ実行の、スクリプト環境のセットアップに使用できる、以下の引数名を認識します。

- `--logging_configuration` — Ray ジョブによって作成されるさまざまなログの生成を停止するために使用されます。これらのログは、すべての Ray ジョブでデフォルトで生成されます。形式: 文字列がエスケープされた JSON オブジェクト。詳細については、「the section called “Ray ジョブがログを生成する方法を設定する”」を参照してください。
- `--min-workers` — Ray ジョブに割り当てられるワーカーノードの最小数。ワーカーノードは、複数のレプリカを実行できます。レプリカは仮想 CPU ごとに 1 つ割り当てられます。形式: 整数。最小値: 0。最大値: ジョブ定義で `--number-of-workers` (`NumberOfWorkers`) に指定できる値。ワーカーノードの考慮に関する詳細は、「the section called “Ray ジョブのワーカーの考慮”」を参照してください。
- `--object_spilling_config` — AWS Glue for Ray は、Ray のオブジェクトストアで利用できるスペースを拡張する方法として、Amazon S3 の使用をサポートしています。この動作を有効にするには、このパラメータを使用して Ray に object spilling の JSON 設定オブジェクトを提供します。Ray の object spilling 設定の詳細については、Ray のドキュメントの「Object Spilling」を参照してください。形式: JSON オブジェクト。

  AWS Glue for Ray では、一度にディスクへのスピリングまたは Amazon S3 へのスピリングのいずれかのみがサポートされています。この制限内であれば、スピリングに複数の場所を指定できます。Amazon S3 にスピリングする場合は、IAM 権限をこのバケットのジョブに追加することも必要になります。

  CLI の構成として JSON オブジェクトを提供する場合は、JSON オブジェクトの文字列をエスケープした文字列として提供する必要があります。例えば、1 つの Amazon S3 パスにスピリングする文字列の値は、`"{\"type\": \"smart_open\", \"params\": {\"uri\": \"s3path\"}}"` のようになります。AWS Glue Studio で、このパラメータを特別な形式のない JSON オブジェクトとして指定します。
- `--object_store_memory_head` — Ray のヘッドノードの Plasma オブジェクトストアに割り当てられるメモリ。このインスタンスは、クラスター管理のサービスおよびワーカーレプリカを実行します。この値は、ウォームスタート後のインスタンスの空きメモリの割合を表します。このパラメータは、メモリを大量に消費するワークロードの調整に使用します。ほとんどのユースケースにはデフォルト値を使用できます。形式: 正の整数。最小値: 1。最大値: 100。

Plasma の詳細については、Ray のドキュメントで「The Plasma In-Memory Object Store」(Plasma インメモリオブジェクトストア) を参照してください。

- `--object_store_memory_worker` — Ray のワーカーノードの Plasma オブジェクトストアに割り当てられるメモリ。これらのインスタンスはワーカーレプリカのみを実行します。この値は、ウォームスタート後のインスタンスの空きメモリの割合を表します。このパラメータは、メモリを大量に消費するワークロードを調整するために使用されます。ほとんどのユースケースにはデフォルト値を使用できます。形式: 正の整数。最小値: 1。最大値: 100。

  Plasma の詳細については、Ray のドキュメントで「The Plasma In-Memory Object Store」(Plasma インメモリオブジェクトストア) を参照してください。

- `--pip-install` — インストールされる Python パッケージのセット。この引数を使用して PyPI からパッケージをインストールできます。形式: カンマ区切りリスト。

  PyPI パッケージのエントリは、ターゲットパッケージの PyPI 名とバージョンを含む `package==version` の形式です。エントリは、Python のバージョンマッチングを使用してパッケージとバージョンをマッチさせるため、シングルイコール = ではなく、== のようにします。バージョンマッチングの演算子は他にもあります。詳細については、Python ウェブサイトの「PEP 440」を参照してください。`--s3-py-modules` を使用してカスタムのモジュールを提供することもできます。

- `--s3-py-modules` — Python モジュールのディストリビューションをホストする Amazon S3 パスのセットです。形式: カンマで区切られたリスト。

  これを使用すれば、独自のモジュールを Ray ジョブに配布できます。`--pip-install` を使用して PyPI からモジュールを提供することもできます。AWS Glue ETL とは異なり、カスタムのモジュールは pip では設定されず、ディストリビューション用に Ray に渡されます。詳細については、「the section called “Ray ジョブ用の Python モジュールを追加する”」を参照してください。

- `--working-dir` – Amazon S3 でホストされている .zip ファイルへのパスです。このパスには、Ray ジョブを実行しているすべてのノードに配布されるファイルが格納されています。形式: 文字列。詳細については、「the section called “Ray ジョブへのファイルの提供”」を参照してください。

メトリクスによる Ray ジョブのモニタリング

AWS Glue Studio と Amazon CloudWatch を使用して、Ray ジョブのモニタリングができます。CloudWatch は、分析が可能にする Ray を使用した AWS Glue から未加工のメトリクスを収集

して処理します。これらのメトリクスは AWS Glue Studio コンソールで確認できます。つまり、ジョブの実行をモニタリングできます。

AWS Glue のモニタリング方法についての概要は、「the section called “CloudWatch メトリクスの使用”」を参照してください。AWS Glue で公開されている CloudWatch メトリクスの使用方法についての概要は、「the section called “CloudWatch によるモニタリング”」を参照してください。

AWS Glue コンソールで Ray ジョブをモニタリングする

ジョブ実行の詳細ページにある [Run details] セクションには、利用可能なジョブメトリクスを視覚化した、事前構築済みの集計グラフを表示できます。AWS Glue Studio は、ジョブが実行されると、その都度ジョブメトリクスを CloudWatch に送信します。これらを使用することで、クラスターとタスクのプロファイルを作成したり、各ノードに関する詳細情報にアクセスしたりできます。

利用可能なメトリクスグラフの詳細については、「the section called “Ray ジョブ実行の Amazon CloudWatch メトリクスを表示”」を参照してください。

CloudWatch の Ray ジョブメトリクスの概要

CloudWatch で詳細モニタリングが有効になっている場合、Ray のメトリクスが公開されます。メトリクスは、CloudWatch の `Glue/Ray` 名前空間に公開されます。

- インスタンスメトリクス

  ジョブに割り当てられたインスタンスの CPU、メモリ、およびディスクの使用率に関するメトリクスを公開します。これらのメトリクスは、`ExecutorId`、`ExecutorType` および `host` などの特徴で識別されます。これらのメトリクスは、標準 Linux CloudWatch エージェントのメトリクスのサブセットです。CloudWatch のドキュメントで、メトリクスの名前と特徴に関する情報を確認できます。詳細については、「CloudWatch エージェントにより収集されるメトリクス」を参照してください。
- Ray クラスターメトリクス

  スクリプトを実行する Ray プロセスからこの名前空間にメトリクスを転送し、最も重要なものを提供します。使用可能なメトリクスは Ray のバージョンによって異なる場合があります。ジョブで実行されている Ray のバージョンについての詳細は、「the section called “AWS Glue バージョン”」を参照してください。

  Ray はメトリクスをインスタンスレベルで収集します。また、タスクとクラスターのメトリクスも提供します。Ray の基盤となるメトリクス戦略の詳細については、Ray のドキュメントの「Metrics」を参照してください。

> **Note**
>
> `Glue/Job Metrics/` 名前空間に Ray のメトリクスが公開されることはありません。この名前空間は、AWS Glue ETL ジョブにのみ使用されます。

AWS Glue での Python シェルジョブに関するジョブプロパティの設定

Python シェルジョブを使用して、AWS Glue でシェルとして Python スクリプトを実行できます。Python シェルジョブを使用すると、Python 3.6 または Python 3.9 と互換性のあるスクリプトを実行することができます。

> **Note**
>
> Pyshell v3.6 のサポートは 2026 年 3 月 1 日に終了します。ワークロードを移行するには、「Migrate from AWS Glue Python shell jobs」を参照してください。Python シェル v3.9 を続行する場合は、「Python シェル 3.6 から Python シェル 3.9 への移行」を参照してください。

トピック

- 制限
- Python シェルジョブのジョブプロパティの定義
- サポートされている Python シェルジョブのライブラリ
- 独自の Python ライブラリの提供
- AWS Glue の Python シェルジョブで AWS CloudFormation を使用する
- Python シェル 3.6 から Python シェル 3.9 への移行
- AWS Glue Python シェルジョブからの移行

制限

Python シェルジョブには、次の制限があることに注意してください。

- Python シェルを使用してジョブのブックマークを使用することはできません。

- Python 3.9 以降では、.egg ファイルとして Python ライブラリをパッケージ化することはできません。代わりに .whl を使用してください。
- S3 データの一時的なコピーには制限があるため、--extra-files オプションは使用できません。

Python シェルジョブのジョブプロパティの定義

このセクションでは、AWS Glue Studio でのジョブプロパティの定義、または AWS CLI の使用について説明します。

AWS Glue Studio

AWS Glue Studio で Python シェルジョブを定義する場合、次のいくつかのプロパティを指定します。

IAM ロール

ジョブの実行とデータストアへのアクセスに使用されるリソースの認可に使用する AWS Identity and Access Management IAM ロールを指定します。AWS Glue でジョブを実行するためのアクセス権限の詳細については、AWS Glue のアイデンティティとアクセスの管理 を参照してください。

タイプ

[Python シェル] を選択して、pythonshell という名前のジョブコマンドを使用して Python スクリプトを実行します。

Python バージョン

Python のバージョンを選択します。デフォルトは Python 3.9 です。有効なバージョンは Python 3.6 および Python 3.9 です。

共通分析ライブラリを読み込む (推奨)

Python 3.9 用の共通ライブラリを Python シェルに含めるには、このオプションを選択します。

ライブラリがカスタムであったり、インストール済みのライブラリと競合する場合は、共通ライブラリをインストールしないことを選択することができます。しかしながら、共通ライブラリ以外のライブラリを追加でインストールすることは可能です。

このオプションを選択する場合、library-set オプションが analytics に設定されます。このオプションの選択を外す場合、library-set オプションが none に設定されます。

スクリプトのファイル名とスクリプトパス

スクリプトのコードでジョブの手続きロジックを定義します。Amazon Simple Storage Service (Amazon S3) でスクリプト名と場所を指定します。パスのスクリプトディレクトリと同じ名前のファイルが存在していないことを確認します。スクリプトの使用の詳細については、「AWS Glue プログラミングガイド」を参照してください。

スクリプト

スクリプトのコードでジョブの手続きロジックを定義します。Python 3.6 または Python 3.9 でスクリプトをコーディングできます。スクリプトは AWS Glue Studio で編集できます。

データ処理単位

このジョブの実行に割り当てられる AWS Glue データ処理ユニット (DPU) の最大数。DPU は処理能力を相対的に測定するもので、4 個の vCPU のコンピューティング性能と 16 GB のメモリで構成されています。詳細については、「AWS Glue 料金表」を参照してください。

値を 0.0625 または 1 に設定できます。デフォルトは 0.0625 です。いずれの場合も、インスタンスのローカルディスクは 20 GB になります。

CLI

また、以下の例のように、AWS CLI を使用して Python シェルジョブを作成することもできます

```
aws glue create-job --name python-job-cli --role Glue_DefaultRole
    --command '{"Name" :  "pythonshell", "PythonVersion": "3.9", "ScriptLocation" :
"s3://amzn-s3-demo-bucket/scriptname.py"}'
    --max-capacity 0.0625
```

> **Note**
>
> `--glue-version` パラメータは AWS Glue シェルジョブに適用されないため、AWS Glue のバージョンを指定する必要はありません。指定されたバージョンは無視されます。

AWS CLI で作成したジョブはデフォルトで Python 3 になります。有効な Python バージョンは 3 (3.6 に対応) と 3.9 です。Python 3.6 を指定するには、`--command` パラメータにこのタプルを追加します: `"PythonVersion":"3"`

Python 3.9 を指定するには、`--command` パラメータにこのタプルを追加します: `"PythonVersion":"3.9"`

Python シェルジョブによって使用される最大容量を設定するには、`--max-capacity` パラメータを指定します。Python シェルジョブに `--allocated-capacity` パラメータは使用できません。

サポートされている Python シェルジョブのライブラリ

Python 3.9 を使用する Python シェルでは、必要に応じて事前にパッケージ済みのライブラリセットを使用するようにライブラリセットを選択できます。ライブラリセットを選択するには、`library-set` オプションを使用します。有効な値は、`analytics` および `none` です。

Python シェルジョブを実行する環境は、次のライブラリをサポートしています。

Python バージョン	Python 3.6	Python 3.9	
ライブラリセット	該当なし	分析	なし
avro		1.11.0	
awscli	116.242	1.23.5	1.23.5
awswrangler		2.15.1	
botocore	1.12.232	1.24.21	1.23.5
boto3	1.9.203	1.21.21	
Elasticsearch		8.2.0	
numpy	1.16.2	1.22.3	
pandas	0.24.2	1.4.2	
psycopg2		2.9.3	
pyathena		2.5.3	
PyGreSQL	5.0.6		
PyMySQL		1.0.2	

Python バージョン	Python 3.6	Python 3.9	
pyodbc		4.0.32	
pyorc		0.6.0	
redshift-connector		2.0.907	
リクエスト	2.22.0	2.27.1	
scikit-learn	0.20.3	1.0.2	
scipy	1.2.1	1.8.0	
SQLAlchemy		1.4.36	
s3fs		2022.3.0	

科学計算用の Python シェルジョブで、NumPy ライブラリを使用できます。詳細については、「NumPy」を参照してください。以下の例は、Python シェルジョブで使用できる NumPy スクリプトを示しています。このスクリプトは、「Hello world」といくつかの数値計算の結果を出力します。

```
import numpy as np
print("Hello world")

a = np.array([20,30,40,50])
print(a)

b = np.arange( 4 )

print(b)

c = a-b

print(c)

d = b**2

print(d)
```

独自の Python ライブラリの提供

PIP を使用する

Python 3.9 を使用する Python シェルでは、ジョブレベルで追加の Python モジュールや異なるバージョンを指定することもできます。`--additional-python-modules` オプションでコンマ区切りの Python モジュールのリストを指定することで、新しいモジュールを追加したり、既存のモジュールのバージョンを変更したりできます。Python シェルジョブを使用する場合、Amazon S3 でホストされているカスタム Python モジュールにこのパラメータを指定することはできません。

例えば、`scikit-learn` モジュールを更新したり新しく追加したりするには、次のキー/値を使用します。`"--additional-python-modules", "scikit-learn==0.21.3"`

AWS Glue は Python パッケージインストーラ (pip3) を使用して追加のモジュールをインストールします。`--additional-python-modules` 値の中で、追加の pip3 オプションを渡すことができます。例えば、`"scikit-learn==0.21.3 -i https://pypi.python.org/simple/"`。pip3 からの非互換性や制限は、すべて適用されます。

> **Note**
>
> 将来的に互換性が失われないように、Python3.9用にビルドされたライブラリを使用することをお勧めします。

Egg または Whl ファイルを使用する

1 つ以上の Python ライブラリが `.egg` または `.whl` ファイルとして既にパッケージ化されている場合があります。その場合は、次の例のように、「`--extra-py-files`」フラグの下で AWS Command Line Interface (AWS CLI) を使用して、それらをジョブに指定できます。

```
aws glue create-job --name python-redshift-test-cli --role role --command '{"Name" :
 "pythonshell", "ScriptLocation" : "s3://MyBucket/python/library/redshift_test.py"}'
    --connections Connections=connection-name --default-arguments '{"--extra-py-
files" : ["s3://amzn-s3-demo-bucket/EGG-FILE", "s3://amzn-s3-demo-bucket/WHEEL-FILE"]}'
```

Python ライブラリから `.egg` ファイルまたは `.whl` ファイルを作成する方法がわからない場合は、次のステップを使用してください。この例は、macOS、Linux、および Windows Subsystem for Linux (WSL) で適用できます。

Python の .egg ファイルまたは .whl ファイルを作成するには

1. Virtual Private Cloud (VPC) で Amazon Redshift クラスターを作成し、テーブルにデータを追加します。
2. クラスターを作成した VPC と SecurityGroup とサブネットを組み合わせて AWS Glue 接続を作成します。接続に成功するかどうかをテストします。
3. `redshift_example` という名前のディレクトリを作成して、`setup.py` という名前のファイルを作成します。次のコードを `setup.py` に貼り付けます。

```
from setuptools import setup

setup(
    name="redshift_module",
    version="0.1",
    packages=['redshift_module']
)
```

4. `redshift_example` ディレクトリに `redshift_module` ディレクトリを作成します。`redshift_module` ディレクトリに、ファイル `__init__.py` と `pygresql_redshift_common.py` を作成します。
5. `__init__.py` ファイルは空のままにします。`pygresql_redshift_common.py` 内に次のコードを貼り付けます。*port*、*db_name*、*user*、および *password_for_user* を、Amazon Redshift クラスター固有の詳細に置き換えます。*table_name* を Amazon Redshift のテーブルの名前と置き換えます。

```
import pg


def get_connection(host):
    rs_conn_string = "host=%s port=%s dbname=%s user=%s password=%s" % (
        host, port, db_name, user, password_for_user)

    rs_conn = pg.connect(dbname=rs_conn_string)
    rs_conn.query("set statement_timeout = 1200000")
    return rs_conn


def query(con):
    statement = "Select * from table_name;"
    res = con.query(statement)
```

```
    return res
```

6. redshift_example ディレクトリに変更します (現在のディレクトリではない場合)。
7. 次のいずれかを行います :

   - .egg ファイルを作成するには、次のコマンドを実行します。

     ```
     python setup.py bdist_egg
     ```

   - .whl ファイルを作成するには、次のコマンドを実行します。

     ```
     python setup.py bdist_wheel
     ```

8. 前述のコマンドに必要な依存関係をインストールします。
9. このコマンドでは、dist ディレクトリにファイルを作成します。

   - egg ファイルを作成した場合は、redshift_module-0.1-py2.7.egg という名前になります。
   - wheel ファイルを作成した場合は、redshift_module-0.1-py2.7-none-any.whl という名前になります。

   このファイルを Amazon S3 にアップロードします。

   この例では、アップロードされたファイルパスは *s3://amzn-s3-demo-bucket/EGG-FILE* または *s3://amzn-s3-demo-bucket/WHEEL-FILE* です。
10. AWS Glue ジョブのスクリプトとして使用する Python ファイルを作成し、このファイルに次のコードを追加します。

```
from redshift_module import pygresql_redshift_common as rs_common

con1 = rs_common.get_connection(redshift_endpoint)
res = rs_common.query(con1)

print "Rows in the table cities are: "

print res
```

11. 前述のファイルを Amazon S3 にアップロードします。この例では、アップロードされたファイルパスは *s3://amzn-s3-demo-bucket/scriptname.py* です。
12. このスクリプトを使用して、Python シェルジョブを作成します。AWS Glue コンソールの [Job properties] (ジョブプロパティ) ページの [Python library path] (Python ライブラリパス) ボックスで `.egg/.whl` ファイルへのパスを指定します。複数の `.egg/.whl` ファイルおよび Python ファイルがある場合は、このボックスにカンマ区切りリストとして指定します。

    `.egg` ファイルを変更または名前変更する場合、ファイル名は「python setup.py bdist_egg」コマンドによって生成されたデフォルト名を使用するか、Python モジュールの命名規則に従う必要があります。詳細については、「Style Guide for Python Code」を参照してください。

    次の例のように、AWS CLI でコマンドを使用してジョブを作成します。

    ```
    aws glue create-job --name python-redshift-test-cli --role Role --command
     '{"Name" :  "pythonshell", "ScriptLocation" : "s3://amzn-s3-demo-bucket/
    scriptname.py"}'
         --connections Connections="connection-name" --default-arguments '{"--extra-
    py-files" : ["s3://amzn-s3-demo-bucket/EGG-FILE", "s3://amzn-s3-demo-bucket/WHEEL-
    FILE"]}'
    ```

    ジョブが実行されると、スクリプトは、Amazon Redshift クラスターの [*table_name*] テーブルに作成された行をプリントします。

AWS Glue の Python シェルジョブで AWS CloudFormation を使用する

AWS Glue の Python シェルジョブで AWS CloudFormation を使用できます。以下に例を示します。

```
AWSTemplateFormatVersion: 2010-09-09
Resources:
  Python39Job:
    Type: 'AWS::Glue::Job'
    Properties:
      Command:
        Name: pythonshell
        PythonVersion: '3.9'
        ScriptLocation: 's3://bucket/location'
      MaxRetries: 0
      Name: python-39-job
      Role: RoleName
```

Python シェルジョブ用の Amazon CloudWatch Logs グループは、`/aws-glue/python-jobs/output` です。エラーについては、「ロググループ `/aws-glue/python-jobs/error`」を参照してください。

Python シェル 3.6 から Python シェル 3.9 への移行

Python シェルのジョブを最新の AWS Glue バージョンに移行するには:

1. AWS Glue コンソール (https://console.aws.amazon.com/glue/) で、既存の Python シェルのジョブを選択します。
2. [ジョブ] の詳細タブで、Python バージョンを `Python 3.9` に設定し、[保存] を選択します。
3. ジョブスクリプトが Python 3.9 と互換性があり、正常に実行されることを確認してください。

AWS Glue Python シェルジョブからの移行

AWS は、2018 年に AWS Glue Pythonシェルジョブをリリースしました。AWS は、小規模から中規模の ETL ジョブ用の Python スクリプトを簡単に実行し、SQL クエリをトリガーする方法をお客様に提供するために、2018 年に AWS Glue Python シェルジョブをリリースしました。ただし、PythonShell で現在実行されているワークロードに対応するための、より最新で柔軟なオプションが登場しました。このトピックでは、利用可能な新しい機能を活用するために、ワークロードを AWS Glue Python シェルジョブからこれらの代替オプションのいずれかに移行する方法について説明します。

このトピックでは、AWS Glue Python シェルジョブから代替オプションに移行する方法について説明します。

ワークロードの AWS Glue Spark ジョブへの移行

AWS Glue Spark ジョブと PySpark ジョブを使用すると、ワークロードを分散して実行できます。AWS Glue Python Shell ジョブと AWS Glue Spark ジョブはどちらも同じプラットフォームで実行されるため、移行が容易で、AWS Glueワークフロー、AWS Glueトリガー、AWS Glueの Amazon EventBridge 統合、\ PIP ベースのパッケージインストールなど、Python Shell ジョブで使用している既存の AWS Glue 機能を引き続き使用できます。

ただし、AWS Glue Spark ジョブは Spark ワークロードを実行するように設計されており、ワーカーの最小数は 2 です。スクリプトを変更せずに Python Shell ジョブから移行する場合、実際に使用さ

れるワーカーは 1 つだけで、他のワーカーはアイドル状態のままになります。これによりコストが増加します。

効率化するには、Spark の機能を活用し、ワークロードを複数のワーカーに分散するように Python ジョブ スクリプトを書き直します。Python スクリプトが Pandas ベースの場合、Spark 上の新しい Pandas API を使用して簡単に移行できます。詳細については、「AWS Big Data Blog: AWS Glue 4.0 for Apache Spark」をご覧ください。

AWS Lambda へのワークロードの移行

AWS Lambda はサーバーのプロビジョニングや管理をする必要がなく、コードを実行できるサーバーレスコンピューティングサービスです。AWS Lambda は起動時間を短縮し、コンピューティング容量の柔軟なオプションを提供するため、これらの利点を活用できます。追加の Python ライブラリを管理するために、 AWS Glue Python シェルジョブは PIP ベースのインストールを使用します。ただし、AWS Lambda では、zip アーカイブ、コンテナイメージ、または Lambda レイヤーのいずれかのオプションを選択する必要があります。

一方、 AWS Lambda の最大タイムアウトは 900 秒 (15 分) です。既存の AWS Glue Python Shell ジョブワークロードのジョブ期間がそれ以上の場合、またはワークロードのパターンが急激でジョブ期間が長くなる可能性がある場合は、AWS Lambda の代わりに他のオプションを検討することをお勧めします。

Amazon ECS/Fargate へのワークロードの移行

Amazon Elastic Container Service (Amazon ECS) は、コンテナ化されたアプリケーションのデプロイ、管理、スケーリングを簡素化するフルマネージドサービスです。 AWS Fargate は、Amazon ECS および Amazon Elastic Kubernetes Service (Amazon EKS) で実行されるコンテナ化されたワークロード用のサーバーレスコンピューティングエンジンです。Amazon ECS と Fargate には最大タイムアウトがないため、長時間実行されるジョブに適しています。コンテナイメージを完全に制御できるため、Python スクリプトと追加の Python ライブラリをコンテナに取り込み、使用できます。ただし、このアプローチを使用するには、Python スクリプトをコンテナ化する必要があります。

Apache Airflow Python Operator の Amazon Managed Workflows へのワークロードの移行

Amazon Managed Workflows for Apache Airflow (MWAA) は、Apache Airflow 向けのマネージドオーケストレーションサービスです。これにより、クラウド内のエンドツーエンドのデータパイプラインを大規模にセットアップして運用することが容易になります。MWAA 環境がすでにある場合は、AWS Glue Python シェルジョブの代わりに Python 演算子を使用するのが簡単です。Python 演

算子は、Airflow ワークフロー内で Python コードを実行する演算子です。ただし、既存の MWAA 環境がない場合は、他のオプションを検討することをお勧めします。

Amazon SageMaker AI AI トレーニングジョブへのワークロードの移行

Amazon SageMaker AI Training は、Amazon SageMaker AI が提供するフルマネージド機械学習 (ML) サービスであり、さまざまな ML モデルを大規模かつ効率的にトレーニングするのに役立ちます。Amazon SageMaker AI AIジョブの中核は、ML ワークロードのコンテナ化と AWS コンピューティングリソースを管理する機能です。最大タイムアウトがないサーバーレス環境を優先する場合、Amazon SageMaker AI AI トレーニングジョブが適している可能性があります。ただし、スタートアップレイテンシーは AWS Glue Python シェルジョブよりも長くなる傾向があります。レイテンシーの影響を受けやすいジョブの場合は、他のオプションを検討することをお勧めします。

AWS Glue のモニタリング

モニタリングは、AWS Glue およびその他の AWS ソリューションの信頼性、可用性、およびパフォーマンスを維持する上で重要な部分です。AWS には、AWS Glue を監視したり、問題が発生したときに報告したり、必要に応じて自動的にアクションを実行するために使用する監視ツールが用意されています。

以下の自動化されたモニタリングツールを使用して、AWS Glue をモニタリングし、問題が発生したときにレポートできます。

- Amazon CloudWatch Events は、AWS リソースの変更を示すシステムイベントをほぼリアルタイムのストリームとして提供します。CloudWatch Events によって、自動イベント駆動型コンピューティングが有効になります。特定のイベントを監視し、これらのイベントが発生したときに他の AWS サービスで自動アクションをトリガーするルールを記述できます。詳細については、「Amazon CloudWatch Events ユーザーガイド」を参照してください。
- Amazon CloudWatch Logs によって Amazon EC2 インスタンス、AWS CloudTrail、などからのログファイルのモニタリング、保存、アクセスができます。CloudWatch Logs は、ログファイル内の情報をモニタリングし、特定のしきい値が満たされたときに通知します。高い耐久性を備えたストレージにログデータをアーカイブすることも可能です。詳細については、「Amazon CloudWatch Logs ユーザーガイド」を参照してください。
- AWS CloudTrail は、AWS アカウントにより、またはそのアカウントに代わって行われた API よルや関連イベントを取得し、指定した Amazon S3 バケットにログファイルを配信します。AWS を呼び出すユーザーとアカウント、呼び出しの送信元 IP アドレス、および呼び出しが発生した時刻を特定できます。詳細については、AWS CloudTrail ユーザーガイドを参照してください。

さらに、AWS Glue コンソールで以下のインサイトにアクセスし、ジョブのデバッグやプロファイリングに役立てることができます。

- Spark ジョブ – 選択した CloudWatch メトリクスシリーズの視覚化を確認できます。また、新しいジョブは Spark UI にアクセスできます。詳細については、「the section called “スパークジョブのモニタリング”」を参照してください。
- Ray ジョブ – 選択した CloudWatch メトリクスシリーズの視覚化を確認できます。詳細については、「the section called “Ray ジョブメトリクス”」を参照してください。

トピック

- AWSAWS Glue のタグ
- EventBridge を使用した AWS Glue の自動化
- AWS Glue リソースのモニタリング
- AWS Glue による AWS CloudTrail API コールのログ記録

AWSAWS Glue のタグ

AWS Glue リソースを管理しやすくするために、オプションでいくつかの AWS Glue リソースタイプに独自のタグを割り当てることができます。タグとは、AWS リソースに割り当てるラベルです。タグはそれぞれ、1 つのキーとオプションの 1 つの値で設定されており、どちらもお客様側が定義します。AWS Glue のタグを使用して、自分のリソースを整理し識別することができます。タグを使用してコスト計算レポートを作成し、リソースへのアクセスを制限することができます。AWS Identity and Access Management を使用している場合は、AWS アカウント内のどのユーザーがタグを作成、編集、削除する許可を持つかをコントロールできます。タグ関連の API を呼び出すためのアクセス許可に加えて、接続でタグ付け API を呼び出すための `glue:GetConnection` アクセス許可と、データベースでタグ付け API を呼び出すための `glue:GetDatabase` アクセス許可も必要です。詳細については、「ABAC AWS Glue を使用」を参照してください。

AWS Glue では、以下のリソースにタグを付けることができます。

- Connection
- データベース
- Crawler
- インタラクティブセッション
- 開発エンドポイント

- ジョブ
- Trigger トリガー)
- ワークフロー
- ブループリント
- 機械学習変換
- データ品質ルールセット
- ストリームスキーマ
- ストリームスキーマレジストリ

> **Note**
>
> ベストプラクティスとして、これらの AWS Glue リソースのタグ付けを可能にするには、常にポリシーに `glue:TagResource` アクションを含めます。

AWS Glue でタグを使用するときは、以下について検討します。

- エンティティ 1 つにつき、最大 50 までのタグがサポートされています。
- AWS Glue では、タグをキーと値のペアのリスト (`{"string": "string" ...}` 形式) で指定します。
- オブジェクトにタグを作成する場合、タグキーは必須で、タグ値はオプションです。
- タグキーとタグ値の大文字と小文字は区別されます。
- タグキーとタグ値にプレフィックス awsを含めることはできません。そのようなタグに対するオペレーションは許可されていません。
- タグキーの最大長は UTF-8 で 128 Unicode 文字です。タグキーを空または null にすることはできません。
- タグ値の最大長は UTF-8 で 256 Unicode 文字です。タグ値は空または null にすることができます。

AWS Glue 接続のタグ付けのサポート

リソースタグに基づき、`CreateConnection`、`UpdateConnection`、`GetConnection`、`DeleteConnection` のアクションの許可を制限することができます。これにより、データカタログから JDBC 接続情報を取得

する必要がある JDBC データソースを持つ AWS Glue ジョブに対して、最小特権のアクセス制御を実装することができます。

使用例

タグ ["connection-category", "dev-test"] との AWS Glue 接続を作成する。

IAM ポリシーの `GetConnection` アクションのタグ条件を指定します。

```
{
   "Effect": "Allow",
   "Action": [
     "glue:GetConnection"
     ],
   "Resource": "*",
   "Condition": {
     "ForAnyValue:StringEquals": {
       "aws:ResourceTag/tagKey": "dev-test"
     }
   }
 }
```

例

次の例では、タグが割り当てられたジョブを作成します。

AWS CLI

```
aws glue create-job --name job-test-tags --role MyJobRole --command
 Name=glueetl,ScriptLocation=S3://aws-glue-scripts//prod-job1
--tags key1=value1,key2=value2
```

AWS CloudFormation JSON

```
{
  "Description": "AWS Glue Job Test Tags",
  "Resources": {
    "MyJobRole": {
      "Type": "AWS::IAM::Role",
      "Properties": {
        "AssumeRolePolicyDocument": {
          "Version": "2012-10-17",
          "Statement": [
```

```
            {
              "Effect": "Allow",
              "Principal": {
                "Service": [
                  "glue.amazonaws.com"
                ]
              },
              "Action": [
                "sts:AssumeRole"
              ]
            }
          ]
        },
        "Path": "/",
        "Policies": [
          {
            "PolicyName": "root",
            "PolicyDocument": {
              "Version": "2012-10-17",
              "Statement": [
                {
                  "Effect": "Allow",
                  "Action": "*",
                  "Resource": "*"
                }
              ]
            }
          }
        ]
      }
    },
    "MyJob": {
      "Type": "AWS::Glue::Job",
      "Properties": {
        "Command": {
          "Name": "glueetl",
          "ScriptLocation": "s3://aws-glue-scripts//prod-job1"
        },
        "DefaultArguments": {
          "--job-bookmark-option": "job-bookmark-enable"
        },
        "ExecutionProperty": {
          "MaxConcurrentRuns": 2
        },
```

```
        "MaxRetries": 0,
        "Name": "cf-job1",
        "Role": {
           "Ref": "MyJobRole",
           "Tags": {
                "key1": "value1",
                "key2": "value2"
           }
        }
      }
    }
  }
}
```

AWS CloudFormation YAML

```
Description: AWS Glue Job Test Tags
Resources:
  MyJobRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: Allow
            Principal:
              Service:
                - glue.amazonaws.com
            Action:
              - sts:AssumeRole
      Path: "/"
      Policies:
        - PolicyName: root
          PolicyDocument:
            Version: '2012-10-17'
            Statement:
              - Effect: Allow
                Action: "*"
                Resource: "*"
  MyJob:
    Type: AWS::Glue::Job
    Properties:
      Command:
```

```
        Name: glueetl
        ScriptLocation: s3://aws-glue-scripts//prod-job1
      DefaultArguments:
        "--job-bookmark-option": job-bookmark-enable
      ExecutionProperty:
        MaxConcurrentRuns: 2
      MaxRetries: 0
      Name: cf-job1
      Role:
        Ref: MyJobRole
        Tags:
           key1: value1
           key2: value2
```

詳細については、「AWSタグ付け戦略」を参照してください。

タグを使用してアクセスをコントロールする方法については、「ABAC AWS Glue を使用」を参照してください。

EventBridge を使用した AWS Glue の自動化

Amazon EventBridge を使用して AWS のサービスを自動化することで、アプリケーションの可用性の問題やリソースの変更などのシステムイベントに自動的に対応できます。AWS のサービスからのイベントは、ほぼリアルタイムに EventBridge に提供されます。簡単なルールを記述して、注目するイベントと、イベントがルールに一致した場合に自動的に実行するアクションを指定できます。自動的にトリガーできるオペレーションには、以下が含まれます。

- AWS Lambda 関数の呼び出し
- Amazon EC2 Run Command の呼び出し
- Amazon Kinesis Data Streams へのイベントの中継
- AWS Step Functions ステートマシンのアクティブ化
- Amazon SNS トピックまたは Amazon SQS キューの通知

AWS Glue で EventBridge を使用する例をいくつか以下に示します。

- ETL ジョブが成功したときの Lambda 関数のアクティブ化
- ETL ジョブが失敗したときの Amazon SNS トピックへの通知

以下の EventBridge は、AWS Glue によって生成されます。

- "detail-type":"Glue Job State Change" のイベントが SUCCEEDED、FAILED、TIMEOUT、STOPPED に対して生成されます。
- ジョブ遅延通知のしきい値を超えると、"detail-type":"Glue Job Run Status" のイベントが RUNNING、STARTING、STOPPING ジョブの実行に対して生成されます。これらのイベントを受信するには、ジョブ遅延通知のしきい値プロパティを設定する必要があります。

  ジョブ遅延通知のしきい値を超えると、ジョブの実行ステータスごとに 1 つだけイベントが生成されます。
- "detail-type":"Glue Crawler State Change" のイベントが Started、Succeeded、Failed に対して生成されます。
- “detail_type”:“Glue Scheduled Crawler Invocation Failure” のイベントは、スケジュールされたクローラが起動に失敗すると生成されます。通知の詳細情報。
  - customerId はお客様のアカウント ID です。
  - crawlerName には、起動に失敗したクローラの名前が含まれています。
  - errorMessage には、呼び出し失敗の例外メッセージが含まれています。
- “detail_type”:“Glue Auto Statistics Invocation Failure“ のイベントは、自動管理列統計タスクの実行開始に失敗した場合に生成されます。通知の詳細情報。
  - catalogId には、カタログに関連付けられた ID が含まれています。
  - databaseName には影響を受けるデータベースの名前が含まれます。
  - tableName には影響を受けるテーブルの名前が含まれます。
  - errorMessage には、呼び出し失敗の例外メッセージが含まれています。
- “detail_type”:“Glue Scheduled Statistics Invocation Failure” のイベントは、(cron) スケジュールされた列統計タスクの実行開始に失敗した場合に生成されます。通知の詳細情報。
  - catalogId には、カタログに関連付けられた ID が含まれています。
  - databaseName には影響を受けるデータベースの名前が含まれます。
  - tableName には影響を受けるテーブルの名前が含まれます。
  - errorMessage には、呼び出し失敗の例外メッセージが含まれています。
- “detail_type”:“Glue Statistics Task Started” のイベントは、列統計タスクの実行が開始すると生成されます。
- “detail_type”:“Glue Statistics Task Succeeded” のイベントは、列統計タスクの実行が成功したときに生成されます。

- “detail_type”:“Glue Statistics Task Failed” のイベントは、列統計タスクの実行が失敗したときに生成されます。
- "detail-type":"Glue Data Catalog Database State Change" のイベントが、CreateDatabase、DeleteDatabase、 CreateTable、DeleteTable、および BatchDeleteTable に対して生成されます。たとえば、テーブルが作成または削除されると、EventBridge に通知が送信されます。通知イベントの順序や有無に依存するプログラムは作成できないことに注意してください。通知イベントは順番が違っていたり不足している可能性があります。イベントはベストエフォートベースで発生します。通知の詳細情報。
  - typeOfChange には、API オペレーションの名前が含まれます。
  - databaseName には影響を受けるデータベースの名前が含まれます。
  - changedTables には、通知ごとに最大 100 の影響を受けるテーブルの名前が含まれます。テーブル名が長いと、複数の通知が作成されることがあります。
- "detail-type":"Glue Data Catalog Table State Change" のイベントが、UpdateTable、CreatePartition、BatchCreatePartition、UpdatePartition、DeletePa および BatchDeletePartition に対して生成されます。たとえば、テーブルまたはパーティションが更新されると、EventBridge に通知が送信されます。通知イベントの順序や有無に依存するプログラムは作成できないことに注意してください。通知イベントは順番が違っていたり不足している可能性があります。イベントはベストエフォートベースで発生します。通知の詳細情報。
  - typeOfChange には、API オペレーションの名前が含まれます。
  - databaseName には影響を受けるリソースを含むデータベースの名前が含まれます。
  - tableName には影響を受けるテーブルの名前が含まれます。
  - changedPartitions は 1 つの通知で影響を受けるパーティションを最大 100 まで指定します。パーティション名が長いと、複数の通知が作成されることがあります。

    たとえば、Year と Month の 2 つのパーティションキーがある場合、"2018,01", "2018,02" は、パーティション "Year=2018" and "Month=01"、およびパーティション "Year=2018" and "Month=02" を変更します。

```
{
    "version":"0",
    "id":"abcdef00-1234-5678-9abc-def012345678",
    "detail-type":"Glue Data Catalog Table State Change",
    "source":"aws.glue",
    "account":"123456789012",
    "time":"2017-09-07T18:57:21Z",
    "region":"us-west-2",
```

```
    "resources":["arn:aws:glue:us-west-2:123456789012:database/default/foo"],
    "detail":{
        "changedPartitions": [
            "2018,01",
            "2018,02"
        ],
        "databaseName": "default",
        "tableName": "foo",
        "typeOfChange": "BatchCreatePartition"
        }
}
```

詳細については、「Amazon CloudWatch Events ユーザーガイド」を参照してください。AWS Glue 固有のイベントについては、「AWS Glue イベント」を参照してください。

AWS Glue リソースのモニタリング

AWS Glue には、想定外の過剰なプロビジョニングや請求額を増やすことを目的とした悪意のある行為からお客様を保護するためのサービス制限があります。制限はサービスの保護にもなります。AWS サービスクォータコンソールにログインすると、お客様は現在のリソース制限を確認し、(必要に応じて) 増加をリクエストできます。

AWS Glue では、サービスのリソース使用率を Amazon CloudWatch 内のパーセンテージで表示し、使用状況をモニタリングするように CloudWatch アラームを設定できます。Amazon CloudWatch は、Amazon インフラストラクチャで実行されている AWS リソースと顧客アプリケーションをモニタリングします。メトリクスは無料で使用できます。次のメトリクスがサポートされています。

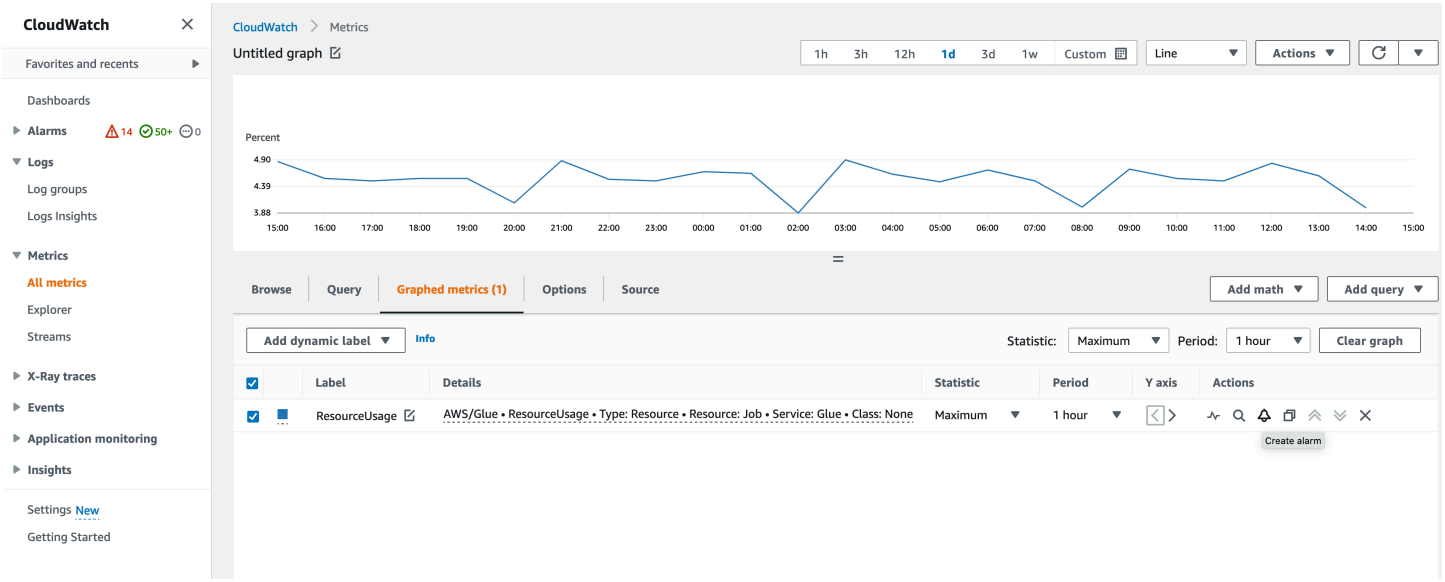
- アカウントあたりのワークフローの数
- アカウントあたりのトリガー数
- アカウントあたりのジョブ数
- アカウントあたりの同時ジョブの実行数
- アカウントあたりのブループリントの数
- アカウントあたりのインタラクティブセッションの数

リソースメトリクスの設定と使用

この機能を使用するには、Amazon CloudWatch コンソールに移動してメトリクスを表示し、アラームを設定します。メトリクスは AWS/Glue 名前空間の下にあり、実際のリソース使用量をリソース

クォータで割ったパーセンテージです。CloudWatch メトリクスはアカウントに配信され、費用はかかりません。例えば、10 件のワークフローを作成していて、サービスクォータで最大 200 件のワークフローを作成できる場合、使用率は 10/200 = 5% となり、グラフでは 5 のデータポイントがパーセンテージとして表示されます。具体的には:

```
Namespace: AWS/Glue
Metric name: ResourceUsage
Type: Resource
Resource: Workflow (or Trigger, Job, JobRun, Blueprint, InteractiveSession)
Service: Glue
Class: None
```



CloudWatch コンソールでクラスターメトリクスにアラームを作成するには:

1. メトリクスを見つけたら、[グラフ化したメトリクス] に移動します。
2. [アクション] の [アラームを作成] をクリックします。
3. 必要に応じてアラームを設定します。

リソース使用量が変化 (増加や減少) するたびにメトリクスを出力します。ただし、リソースの使用量が変わらない場合でも、メトリクスは 1 時間ごとに出力されるため、CloudWatch グラフは連続的になります。データポイントの欠落を防ぐため、1 時間未満の期間を設定することはお勧めしません。

また、以下の例のように、AWS CloudFormation を使用してアラームを設定することもできます。この例では、ワークフローリソースの使用率が 80% に達すると、アラームがトリガーされて既存

の SNS トピックにメッセージが送信されます。サブスクライブすると通知を受け取ることができます。

```
{
 "Type": "AWS::CloudWatch::Alarm",
 "Properties": {
  "AlarmName": "WorkflowUsageAlarm",
  "ActionsEnabled": true,
  "OKActions": [],
  "AlarmActions": [
   "arn:aws:sns:af-south-1:085425700061:Default_CloudWatch_Alarms_Topic"
  ],
  "InsufficientDataActions": [],
  "MetricName": "ResourceUsage",
  "Namespace": "AWS/Glue",
  "Statistic": "Maximum",
  "Dimensions": [{
    "Name": "Type",
    "Value": "Resource"
   },
   {
    "Name": "Resource",
    "Value": "Workflow"
   },
   {
    "Name": "Service",
    "Value": "Glue"
   },
   {
    "Name": "Class",
    "Value": "None"
   }
  ],
  "Period": 3600,
  "EvaluationPeriods": 1,
  "DatapointsToAlarm": 1,
  "Threshold": 80,
  "ComparisonOperator": "GreaterThanThreshold",
  "TreatMissingData": "notBreaching"
 }
}
```

AWS Glue による AWS CloudTrail API コールのログ記録

AWS Glue は AWS CloudTrail という、AWS Glue の ユーザー、ロール、または AWS のサービスが実行したアクションを記録するサービスと統合しています。CloudTrail は、AWS Glue のすべての API コールをイベントとしてキャプチャします。キャプチャされたコールには、AWS Glue コンソールのコールと、AWS Glue API オペレーションへのコードのコールが含まれます。証跡を作成する場合は、AWS Glue のイベントなど、Amazon S3 バケットへの CloudTrail イベントの継続的な配信を有効にすることができます 証跡を設定しない場合でも、「CloudTrail」 コンソールの 「イベント履歴」 で最新のイベントを表示できます。CloudTrail が収集した情報を使用して、AWS Glue に対して行われた要求、要求が行われた IP アドレス、要求を行った人、要求が行われた日時、および追加の詳細を判別できます。

CloudTrail の詳細については、「AWS CloudTrail ユーザーガイド」を参照してください。

CloudTrail の AWS Glue 情報

CloudTrailは、アカウントを作成するとAWSアカウントで有効になります。AWS Glue でアクティビティが発生すると、そのアクティビティは Event history イベント履歴で AWS のその他のサービスのイベントと共に CloudTrail イベントに記録されます。最近のイベントは、AWSアカウントで表示、検索、ダウンロードできます。詳細については、「CloudTrailイベント履歴でのイベントの表示」を参照してください。

AWSのイベントなど、AWS Glueアカウントのイベントの継続的なレコードについては、追跡を作成します。証跡により、CloudTrail はログファイルを Amazon S3 バケットに配信できます。デフォルトでは、コンソールで証跡を作成すると、すべての AWS リージョンに証跡が適用されます。証跡は、AWSパーティションのすべてのリージョンからのイベントをログに記録し、指定した Amazon S3 バケットにログファイルを配信します。さらに、CloudTrail ログで収集したイベントデータをより詳細に分析し、それに基づいて対応するため、他の AWS サービスを構成できます。詳細については、次を参照してください:

- AWS アカウントの証跡の作成
- 「CloudTrail がサポートされているサービスと統合」
- CloudTrail 用 Amazon SNS 通知の構成
- 「複数のリージョンからCloudTrailログファイルを受け取る」および「複数のアカウントからCloudTrailログファイルを受け取る」

すべての AWS Glue アクションは CloudTrail によってログに記録され、AWS Glue API に記録されます。例えば、CreateDatabase、CreateTable、CreateScript の各アクションを呼び出すと、CloudTrail ログファイルにエントリが生成されます。

各イベントまたはログエントリには、誰がリクエストを生成したかという情報が含まれます。同一性情報は次の判断に役立ちます。

- リクエストが、ルートと IAM ユーザー認証情報のどちらを使用して送信されたか。
- リクエストがロールまたはフェデレーションユーザーの一時的なセキュリティ認証情報を使用して行われたかどうか。
- リクエストが別の AWS サービスによって行われたかどうか。

詳細については、「CloudTrail userIdentity Element」(CloudTrail userIdentity 要素) を参照してください。

ただし、CloudTrail は呼び出しに関するすべての情報は記録しません。たとえば、接続リクエストで使用される ConnectionProperties などの機密情報は記録せず、次の API によって返される応答の代わりに null を記録します。

```
BatchGetPartition       GetCrawlers          GetJobs          GetTable
CreateScript            GetCrawlerMetrics    GetJobRun        GetTables
GetCatalogImportStatus  GetDatabase          GetJobRuns       GetTableVersions
GetClassifier           GetDatabases         GetMapping       GetTrigger
GetClassifiers          GetDataflowGraph     GetObjects       GetTriggers
GetConnection           GetDevEndpoint       GetPartition     GetUserDefinedFunction
GetConnections          GetDevEndpoints      GetPartitions    GetUserDefinedFunctions
GetCrawler              GetJob               GetPlan
```

AWS Glue ログファイルエントリの理解

追跡は、指定したAmazon S3バケットにイベントをログファイルとして配信するように設定できます。CloudTrail のログファイルは、単一か複数のログエントリを含みます。イベントは任意ソースからの単一リクエストを表し、リクエストされたアクション、アクションの日時、リクエストパラメータなどの情報を含みます。CloudTrail ログファイルは、パブリック API コールの順序付けられたスタックトレースではないため、特定の順序では表示されません。

以下の例は、DeleteCrawler アクションを示す CloudTrail ログエントリです。

```
{
```

```
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AKIAIOSFODNN7EXAMPLE",
    "arn": "arn:aws:iam::123456789012:user/johndoe",
    "accountId": "123456789012",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "userName": "johndoe"
  },
  "eventTime": "2017-10-11T22:29:49Z",
  "eventSource": "glue.amazonaws.com",
  "eventName": "DeleteCrawler",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "72.21.198.64",
  "userAgent": "aws-cli/1.11.148 Python/3.6.1 Darwin/16.7.0 botocore/1.7.6",
  "requestParameters": {
    "name": "tes-alpha"
  },
  "responseElements": null,
  "requestID": "b16f4050-aed3-11e7-b0b3-75564a46954f",
  "eventID": "e73dd117-cfd1-47d1-9e2f-d1271cad838c",
  "eventType": "AwsApiCall",
  "recipientAccountId": "123456789012"
}
```


この例では、CreateConnection アクションを表す CloudTrail ログエントリを示しています。


```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AKIAIOSFODNN7EXAMPLE",
    "arn": "arn:aws:iam::123456789012:user/johndoe",
    "accountId": "123456789012",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "userName": "johndoe"
  },
  "eventTime": "2017-10-13T00:19:19Z",
  "eventSource": "glue.amazonaws.com",
  "eventName": "CreateConnection",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "72.21.198.66",
  "userAgent": "aws-cli/1.11.148 Python/3.6.1 Darwin/16.7.0 botocore/1.7.6",
```

```
  "requestParameters": {
    "connectionInput": {
      "name": "test-connection-alpha",
      "connectionType": "JDBC",
      "physicalConnectionRequirements": {
        "subnetId": "subnet-323232",
        "availabilityZone": "us-east-1a",
        "securityGroupIdList": [
          "sg-12121212"
        ]
      }
    }
  },
  "responseElements": null,
  "requestID": "27136ebc-afac-11e7-a7d6-ab217e5c3f19",
  "eventID": "e8b3baeb-c511-4597-880f-c16210c60a4a",
  "eventType": "AwsApiCall",
  "recipientAccountId": "123456789012"
}
```

AWS Glue ジョブ実行ステータス

AWS Glue 抽出、変換、ロード (ETL) ジョブのステータスは、実行中または停止後に表示できます。ステータスは AWS Glue コンソール、AWS Command Line Interface (AWS CLI)、または AWS Glue API の GetJobRun アクションで表示できます。

ジョブ実行ステータスに
は、STARTING、RUNNING、STOPPING、STOPPED、SUCCEEDED、FAILED、ERROR、WAITING、TIMEOUT
があります。

次の表に、ジョブの異常終了を示すステータスを一覧表示します。

ジョブ実行ステータス	説明
FAILED	ジョブが許容される最大同時実行数を超えているか、不明な終了コードで終了しました。
ERROR	ワークフロー、スケジュールトリガー、またはイベントトリガーが削除されたジョブを実行しようとしました。

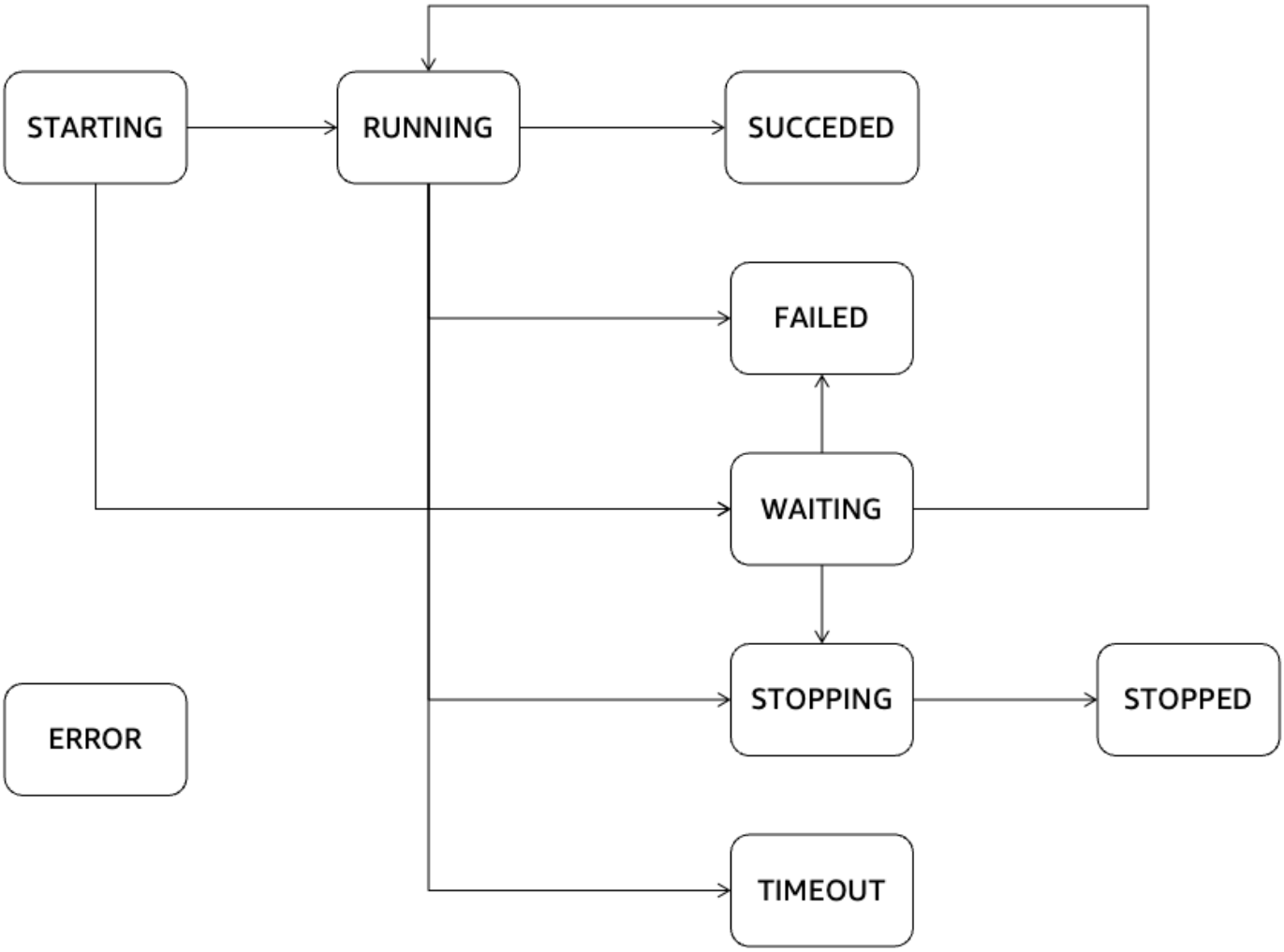
ジョブ実行ステータス	説明
TIMEOUT	ジョブの実行時間が、指定されたタイムアウト値を超えました。
WAITING	リソースを待機しているジョブ実行。

WAITING ステータスは、ジョブの実行がリソース待ちであることを示しています。次の表には、さまざまなクラスのジョブの待機動作が記載されています。

ジョブタイプ	Behavior
Spark ジョブ (標準)	「AWS Glue で Spark ジョブに関するジョブプロパティの構成」で説明されているように、AWS Glue ジョブ実行のキューイングをオプトインすると、ジョブが WAITING 状態になることがあります。ジョブ実行は、アカウントの Service Quotas またはリージョンのキャパシティ制限により [待機中] 状態になることがあり、次のいずれかのエラーケースにつながる可能性があります。 • アカウントあたりの同時実行ジョブの最大実行数を超過しました • ジョブごとの最大同時ジョブ実行数を超過した (アカウントレベルのサービスクォータと、MaxConcurrentRuns でジョブについて指定した制限を含みます) • 最大同時コンピューティング (DPU 使用量) を超過した • リソースを利用できません。 ジョブは、Service Quotas に関係なく、ジョブ実行を開始するのに十分なリソースがない場合に WAITING 状態になることもあります。

ジョブタイプ	Behavior
Spark ジョブ (Flex)	サービスが実行を開始するのに十分なリソースを取得できない場合、新しいジョブの実行は WAITING 状態になり、実行の開始が遅れます。実行は最大 20 分間にわたって WAITING 状態になります (タイムアウト時間はサービスが制御)。15 分経過すると、サービスは強制起動を試み、使用可能な容量に応じて実行が開始するか失敗し、適切なエラーメッセージが表示されます。
Python シェルジョブ	Spark を使用する標準ジョブと同じ動作。

次の状態図は、AWS Glue ジョブのライフサイクルを通じて予想される状態遷移の概要を示しています。この情報はすべてのジョブタイプに適用されます。


STARTING
RUNNING
SUCCEDED
FAILED
WAITING
STOPPING
STOPPED
ERROR
TIMEOUT

AWS Glue Streaming

AWS Glue のコンポーネントである AWS Glue Streaming を使用すると、ストリーミングデータをほぼリアルタイムで効率的に処理できるようになり、データインジェスト、処理、機械学習などの重要なタスクを実行できるようになります。AWS Glue Streaming は、Apache Spark Streaming フレームワークを使用して、ストリーミングデータを大規模に処理できるサーバーレスサービスを提供します。AWS Glue は、サーバーレスインフラストラクチャ、自動スケーリング、ビジュアルジョブ開発、ストリーミングジョブの瞬時オンノートブック、その他のパフォーマンスの向上など、Apache Spark 上でさまざまな最適化を提供します。

ストリーミングのユースケース

AWS Glue Streaming の一般的なユースケースには以下が含まれます。

ほぼリアルタイムのデータ処理: AWS Glue Streaming により、組織はストリーミングデータをほぼリアルタイムで処理できるため、最新の情報に基づいて洞察を導き出し、タイムリーな意思決定を行うことができます。

不正検出: AWS Glue Streaming はストリーミングデータのリアルタイム分析に利用できるため、クレジットカード詐欺、ネットワーク侵入、オンライン詐欺などの不正行為の検出に役立ちます。受信データを継続的に処理して分析することで、疑わしいパターンや異常を迅速に特定できます。

ソーシャルメディア分析: AWS Glue Streaming では、ツイート、投稿、コメントなどのソーシャルメディアデータをリアルタイムで処理できるため、組織はトレンドの監視、感情分析、ブランドの評判の管理をリアルタイムで行うことができます。

モノのインターネット (IoT) 分析: AWS Glue Streaming は、IoT デバイス、センサー、接続された機械によって生成される高速データストリームの処理と分析に適しています。これにより、リアルタイムの監視、異常検知、予知保全、およびその他の IoT 分析のユースケースが可能になります。

クリックストリーム分析: AWS Glue Streaming では、ウェブサイトやモバイルアプリケーションからのリアルタイムのクリックストリームデータを処理して分析できます。これにより、企業はユーザーの行動に関する洞察を得たり、ユーザー体験をパーソナライズしたり、リアルタイムのクリックストリームデータに基づいてマーケティングキャンペーンを最適化したりすることができます。

ログの監視と分析: AWS Glue Streaming では、サーバー、アプリケーション、ネットワークデバイスからのログデータをリアルタイムで継続的に処理および分析できます。これは、異常の検出、問題のトラブルシューティング、システムの状態とパフォーマンスの監視に役立ちます。

レコメンデーションシステム: AWS Glue Streaming では、ユーザーアクティビティデータをリアルタイムで処理し、レコメンデーションモデルを動的に更新できます。これにより、ユーザーの行動や好みに基づいた、パーソナライズされたリアルタイムのレコメンデーションが可能になります。

これらは、AWS Glue Streaming を適用できるさまざまなユースケースの例です。AWS エコシステムやマネージドサービスとの統合により、クラウドでのリアルタイムのストリーム処理や分析を便利に行えるようになります。

AWS Glue Streaming を使用することの利点は何ですか。

AWS Glue Streaming を使用する利点は以下のとおりです。

- サーバーレス: AWS Glue Streaming はサーバーレスなので、インフラストラクチャを管理する必要がありません。これにより、運用上のオーバーヘッドが軽減され、ユーザーは、インフラストラクチャ管理ではなくデータ処理および分析タスクに集中できます。
- 自動スケーリング: AWS Glue Streaming には自動スケーリング機能があり、ワークロードに基づいて処理能力を動的に調整します。データ量の変動に合わせて自動的にスケールアウトまたはスケールインを行い、最適なパフォーマンスとリソース使用率を確保します。
- ビジュアル開発: ストリーミングジョブの開発は複雑な場合があります。AWS GlueStreaming は、ビジュアルオーサリングツールである AWS Glue Studio を提供することで、この課題を解決します。AWS GlueStudio を使用すると、ストリーミングワークフローの作成プロセスが簡略化され、開発者がストリーミングアプリケーションを視覚的に設計および管理できるようになるため、習得時間が短縮され、生産性が向上します。
- 費用対効果: AWS Glue Streaming はサーバーレスサービスであり、インフラストラクチャのプロビジョニングや保守が不要になるため、コスト効率が向上します。ユーザーへの請求は、ストリーミングジョブの実行中に消費されたリソースに基づいて行われるため、実際の使用量に基づくコストの最適化とスケーリングが可能になります。
- 複雑なワークロードの処理: AWS Glue Streaming は複雑なストリーミングワークロードを処理するように設計されています。大量のリアルタイムデータの処理と分析、高度な変換のサポート、他の AWS サービスとの統合が可能なため、高度なストリーミングデータパイプラインおよび分析ワークフローが可能になります。
- ロックインなし: AWS Glue Streaming には柔軟性があり、ベンダーロックインを回避できます。ユーザーは AWS Glue Streaming を広範な AWS エコシステムの一部として活用し、他の AWS サービスとシームレスに統合できます。これにより、特定のテクノロジーやプラットフォームに縛られることなく、既存のデータソース、アプリケーション、サービスと簡単に統合できます。

AWS Glue Streaming はどのようなときに使うべきですか。

ストリーミングのユースケースに関して言えば、多くの選択肢があります。AWS Glue Streaming は以下のシナリオで推奨されます。

1. 既に AWS Glue または Spark をバッチ処理に使用している場合は、AWS Glue Streaming が理想的な選択肢です。新しい言語やフレームワークを学習しなくても、ストリーミングジョブの構築にシームレスに移行できます。AWS Glue Streaming を利用すると、既存の知識とインフラストラクチャを活用してジョブ開発プロセスを簡素化し、データ処理機能をリアルタイムのストリーミングシナリオに簡単に拡張できます。
2. バッチ、ストリーミング、およびイベント駆動型ワークロードを処理する統合サービスまたは製品が必要な場合は、AWS Glue Streaming が最適なソリューションです。AWS Glue Streaming を利用すれば、データ処理のニーズを 1 つのフレームワークに統合できるため、複数のシステムを管理する複雑さがなくなります。これにより、さまざまなワークロードタイプ間での一貫性と互換性を確保しながら、多様なデータワークフローを効率的に開発および保守できます。
3. AWS Glue Streaming は、ストリーミングデータ量が非常に多く、ストリームやリレーショナルデータベース間の結合などの複雑な変換を伴うシナリオに適しています。大量のデータストリームを効率的に処理して分析できるため、要求の厳しいワークロードにも簡単に取り組むことができます。高速なデータインジェストでも、複雑なデータ操作でも、AWS Glue Streaming のスケーラビリティと高度な処理機能により、最適なパフォーマンスと正確な結果が得られます。
4. ストリーミングジョブの構築に視覚的なアプローチを希望する場合は、AWS Glue が提供する、ストリーミングアプリケーションを視覚的に設計および管理できる AWS Glue Studio によって、開発プロセスを簡素化することができます。この直感的なインターフェイスにより、開発者はビジュアルインターフェイスを使用してストリーミングワークフローを作成、設定、監視できるため、習得時間が短縮され、生産性が向上します。
5. AWS Glue Streaming は、10 秒を超える厳しい SLA (サービスレベル契約) が適用されるほぼリアルタイムのユースケースに最適です。
6. Apache Iceberg、Apache Hudi、または Delta Lake を使用してトランザクションデータレイクを構築する場合、AWS Glue Streaming は、これらのオープンテーブル形式をネイティブにサポートします。このシームレスな統合により、これらのトランザクションデータレイクからストリーミングデータを直接処理できるようになり、データ整合性、完全性、互換性が確保されます。
7. さまざまなデータターゲットのストリーミングデータを取り込む必要がある場合: AWS Glue Streaming は、Amazon Redshift、Amazon RDS、Amazon Aurora、Oracle、SQL Server などのさまざまなデータターゲットにネイティブターゲットを提供します。

サポートされているデータソース

AWS Glue Streaming では、次のデータソースがサポートされています。

- Amazon Kinesis
- Amazon MSK (Managed Streaming for Apache Kafka)
- セルフマネージド Apache Kafka

サポートされるデータターゲット

AWS Glue Streaming では、次のようなさまざまなデータターゲットがサポートされています。

- AWS Glue データカタログがサポートするデータターゲット
- Amazon S3
- Amazon Redshift
- MySQL
- PostgreSQL
- Oracle
- Microsoft SQL Server
- Snowflake
- JDBC を使用して接続できるすべてのデータベース
- Apache Iceberg、Delta、および Apache Hudi
- AWS Glue Marketplace コネクタ

チュートリアル: AWS Glue Studio を使用して最初のストリーミングワークロードを構築する

このチュートリアルでは、AWS Glue Studio を使用してストリーミングジョブを作成する方法について説明します。AWS GlueStudio は、AWS Glue ジョブを作成するためのビジュアルインターフェイスです。

Amazon Kinesis Data Streams、Apache Kafka、および Amazon Managed Streaming for Apache Kafka (Amazon MSK) で、ストーミングソースからのデータを消費し、連続的に実行するストリーミング抽出、変換、ロード (ETL) ジョブを作成できます。

前提条件

このチュートリアルを実行するには、AWS Glue、Amazon Kinesis、Amazon S3、Amazon Athena、AWS CloudFormation、AWS Lambda、Amazon Cognito を使用する AWS コンソール権限を持つユーザーが必要です。

Amazon Kinesis からストリーミングデータを消費する

トピック

- Kinesis Data Generator でモックデータを生成する
- AWS Glue Studio で AWS Glue Streaming ジョブを作成する
- 変換を実行し、変換した結果を Amazon S3 に保存する

Kinesis Data Generator でモックデータを生成する

Kinesis Data Generator (KDG) を使用して JSON 形式のサンプルデータを合成的に生成できます。完全な手順と詳細については、ツールのドキュメントを参照してください。

1. 開始するには、 

をクリックして、ご使用の AWS 環境で AWS CloudFormation テンプレートを実行します。

> **Note**
>
> Kinesis Data Generator の Amazon Cognito ユーザーなどのリソースが AWS アカウントに既に存在しているために、CloudFormation テンプレートの実行が失敗する場合があります。これは、別のチュートリアルやブログで既に設定していたことが原因である可能性があります。これに対処するには、新しい AWS アカウントでテンプレートの実行をやり直すか、別の AWS リージョンを試してみてください。これらの方法により、既存のリソースと競合することなくチュートリアルを実行できるようになります。

このテンプレートにより、Kinesis データストリームと Kinesis Data Generator アカウントがプロビジョニングされます。また、データを保持する Amazon S3 バケットと、このチュートリアルに必要な権限を持つ Glue サービスロールが作成されます。

2. KDG が認証に使用する [ユーザー名] と [パスワード] を入力します。後で使用するために、ユーザー名とパスワードをメモしておきます。
3. 最後のステップまで [次へ] を選択していきます。IAM リソースの作成を承認します。パスワードが最小要件を満たしていないなど、画面上部にエラーがないか確認し、テンプレートをデプロイします。
4. スタックの [出力] タブに移動します。テンプレートがデプロイされると、生成されたプロパティ KinesisDataGeneratorUrl が表示されます。その URL をクリックします。
5. メモしておいた [ユーザー名] と [パスワード] を入力します。
6. 使用しているリージョンを選択し、Kinesis ストリーム `GlueStreamTest-{AWS::AccountId}` を選択します。
7. 以下のテンプレートを入力します。

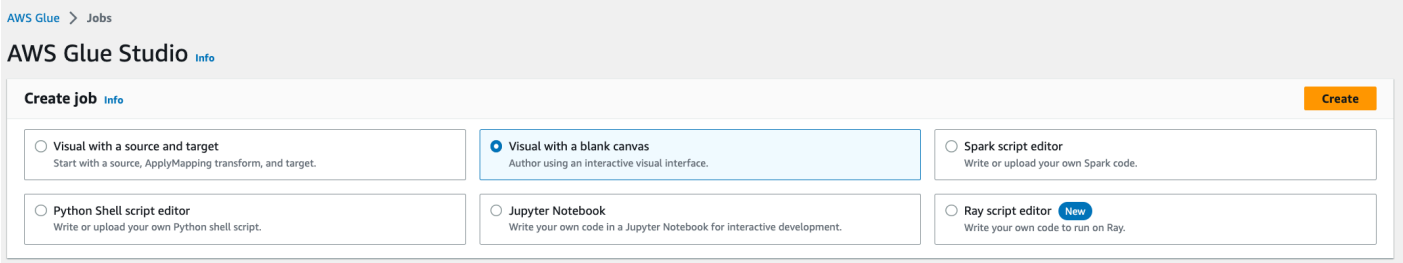
```
{
    "ventilatorid": {{random.number(100)}},
    "eventtime": "{{date.now("YYYY-MM-DD HH:mm:ss")}}",
    "serialnumber": "{{random.uuid}}",
    "pressurecontrol": {{random.number(
        {
            "min":5,
            "max":30
        }
    )}},
    "o2stats": {{random.number(
        {
            "min":92,
            "max":98
        }
    )}},
    "minutevolume": {{random.number(
        {
            "min":5,
            "max":8
        }
    )}},
    "manufacturer": "{{random.arrayElement(
        ["3M", "GE","Vyaire", "Getinge"]
    )}}"
}
```

[テストテンプレート] を使用してモックデータを表示し、[データを送信する] を使用してモックデータを Kinesis に取り込むことができるようになりました。

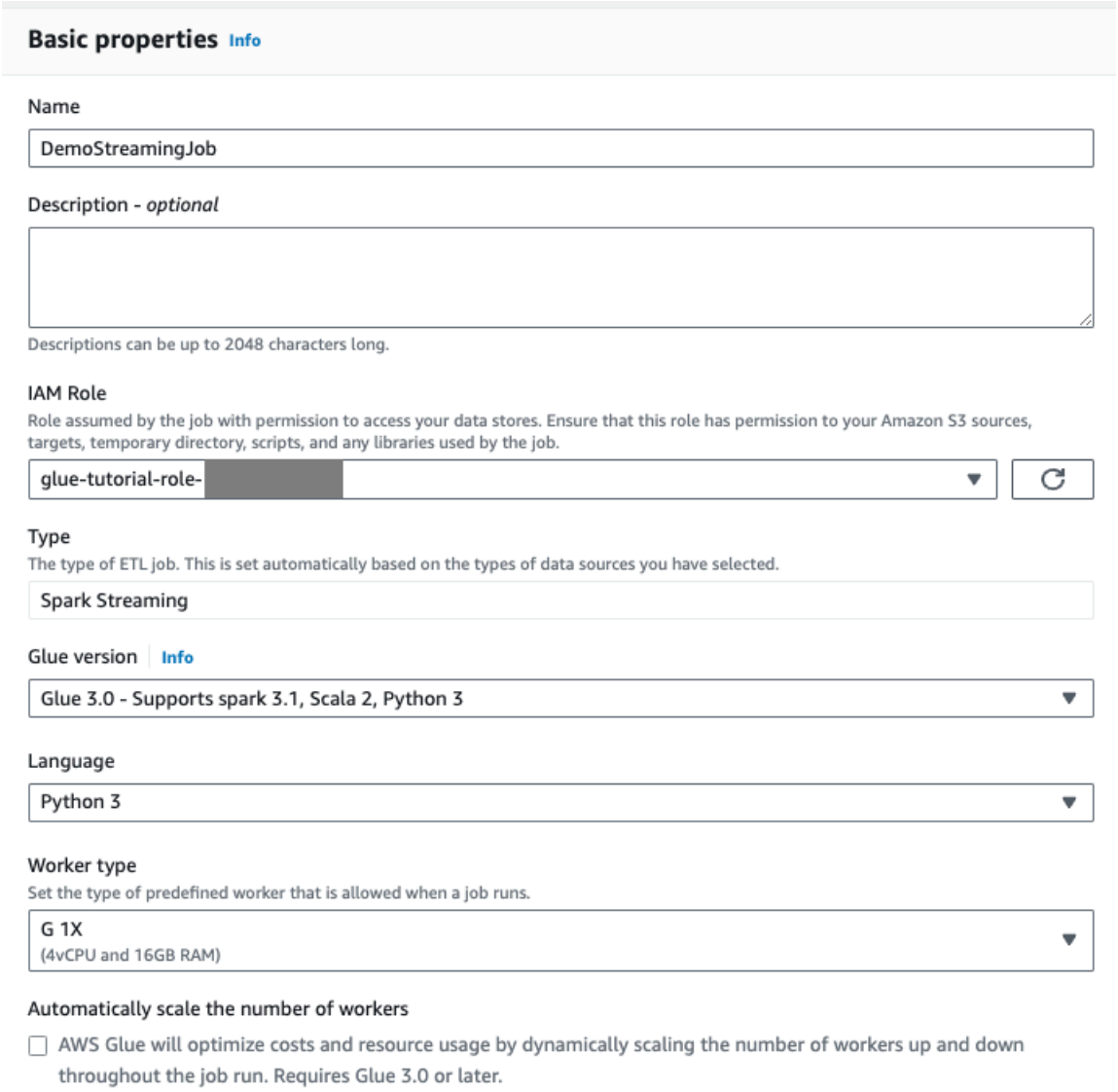
8. [データを送信する] をクリックして、5 ~ 10K のレコードを Kinesis に生成します。

AWS Glue Studio で AWS Glue Streaming ジョブを作成する

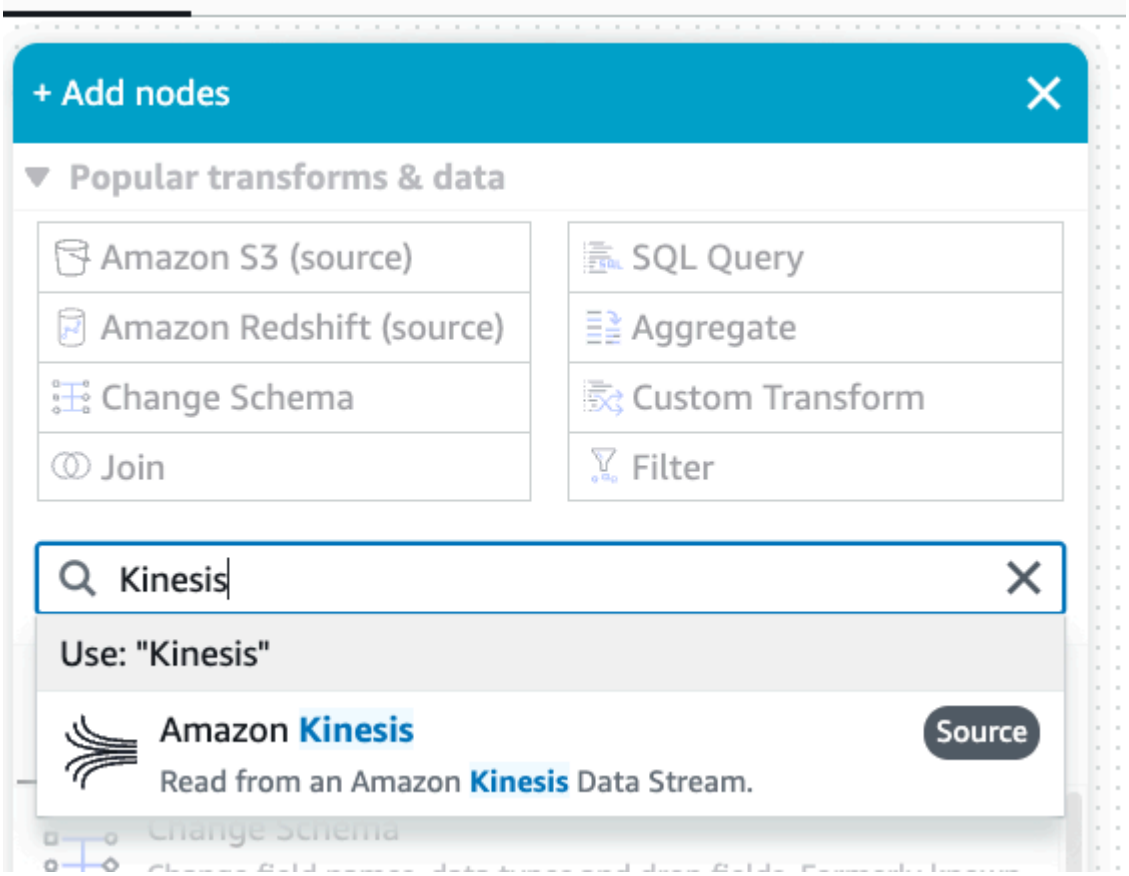
1. 同じリージョンのコンソールで AWS Glue に移動します。
2. 左側のナビゲーションバーの [データ統合と ETL] にある [ETL ジョブ] を選択します。
3. [空白のキャンバスのビジュアル] を使用して AWS Glue ジョブを作成します。



4. [ジョブの詳細] タブに移動します。
5. AWS Glue のジョブ名には `DemoStreamingJob` を入力します。
6. [IAM ロール] では、CloudFormation テンプレートによってプロビジョニングされたロール `glue-tutorial-role-${AWS::AccountId}` を選択します。
7. [Glue バージョン] では、Glue 3.0 を選択します。その他のオプションはすべて、デフォルト設定のままにします。



8. [ビジュアル] タブに移動します。
9. プラスアイコンをクリックします。検索バーに「Kinesis」と入力します。[Amazon Kinesis] データソースを選択します。



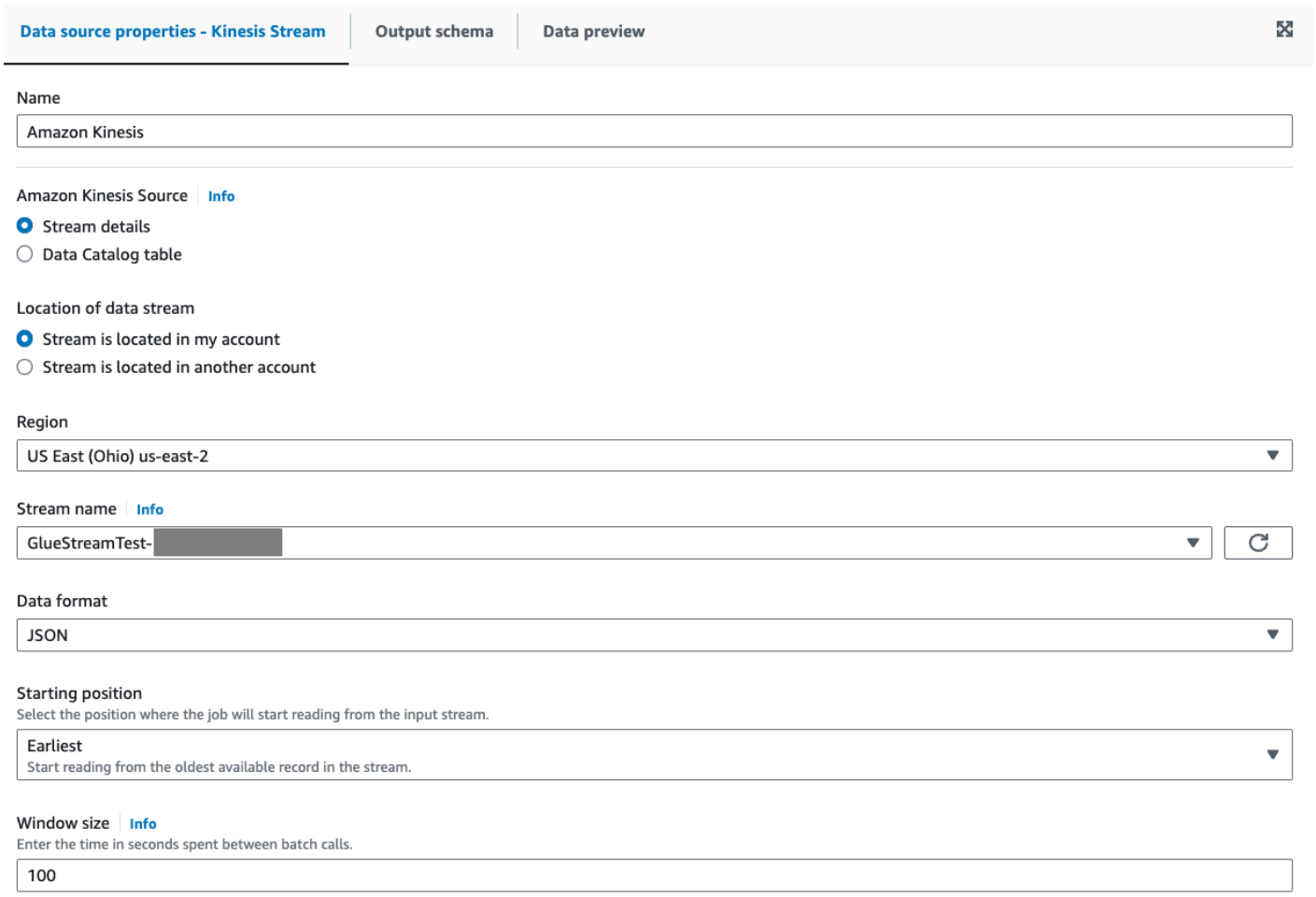
10[データソースのプロパティ - Kinesis ストリーム] タブの [Amazon Kinesis ソース] で [ストリームの詳細] を選択します。

11[データストリームの場所] で [自分のアカウントにストリームを配置する] を選択します。

12使用しているリージョンを選択します。

13`GlueStreamTest-{AWS::AccountId}` ストリームを選択します。

14他の設定はすべて、デフォルトのままにします。

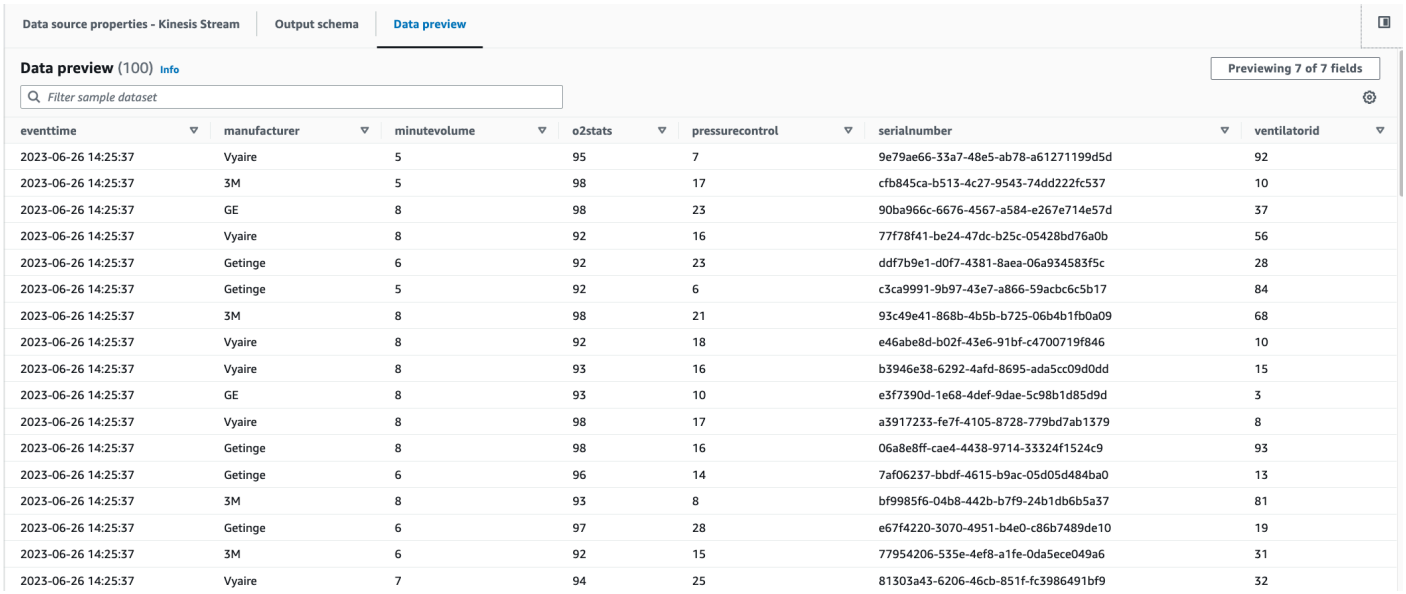


15[データのプレビュー] タブに移動します。

16[データプレビューセッションを開始] をクリックすると、KDG が生成したモックデータがプレビューされます。AWS Glue Streaming ジョブのために前もって作成した Glue サービスロールを選択します。

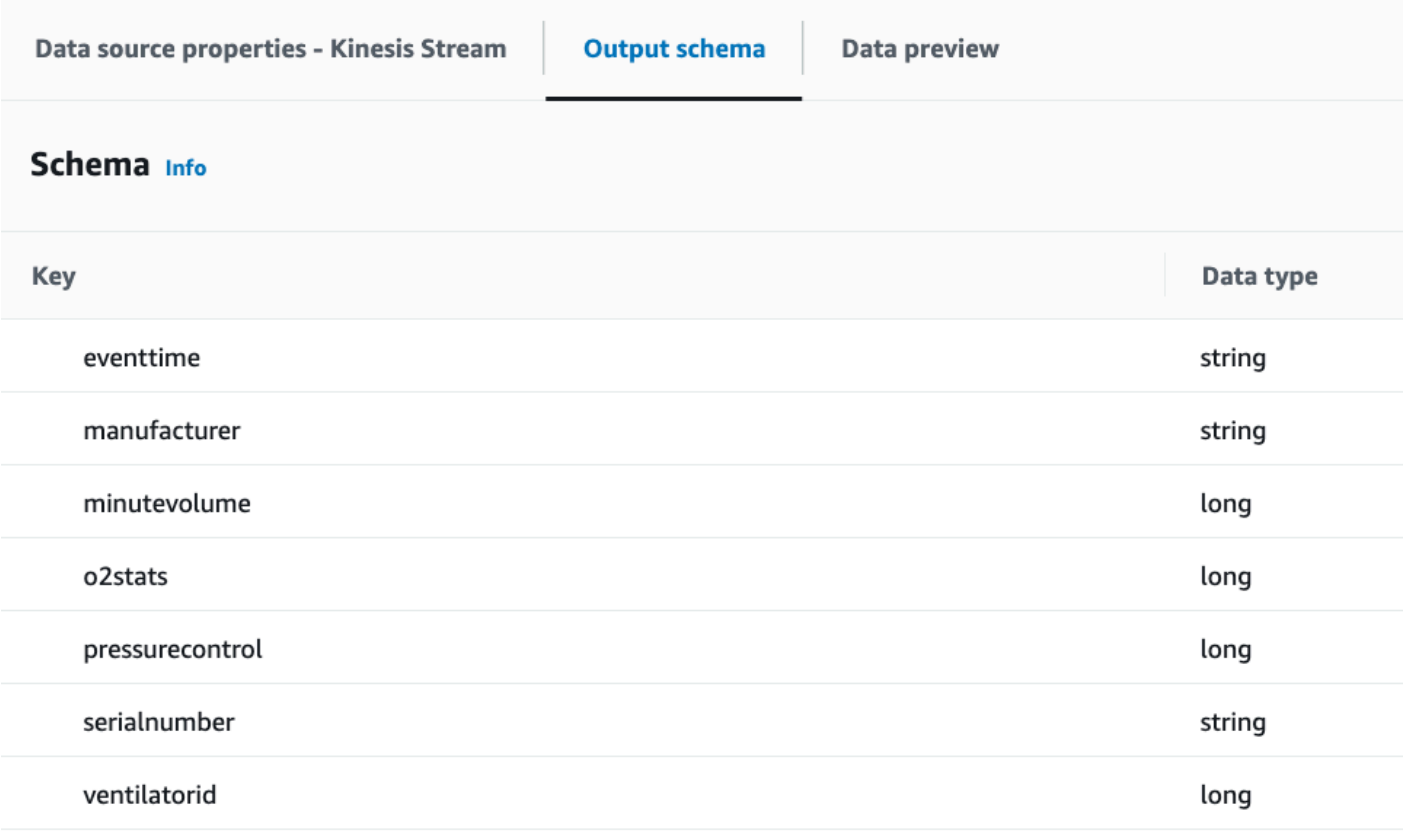
プレビューデータが表示されるまで 30～60 秒かかります。[表示するデータがありません] と表示された場合は、歯車アイコンをクリックして、[サンプリングする行数] を `100` に変更します。

サンプルデータは以下のように表示されます。



eventtime	manufacturer	minutevolume	o2stats	pressurecontrol	serialnumber	ventilatorid
2023-06-26 14:25:37	Vyaire	5	95	7	9e79ae66-33a7-48e5-ab78-a61271199d5d	92
2023-06-26 14:25:37	3M	5	98	17	cfb845ca-b513-4c27-9543-74dd222fc537	10
2023-06-26 14:25:37	GE	8	98	23	90ba966c-6676-4567-a584-e267e714e57d	37
2023-06-26 14:25:37	Vyaire	8	92	16	77f78f41-be24-47dc-b25c-05428bd76a0b	56
2023-06-26 14:25:37	Getinge	6	92	23	ddf7b9e1-d0f7-4381-8aea-06a934583f5c	28
2023-06-26 14:25:37	Getinge	5	92	6	c3ca9991-9b97-43e7-a866-59acbc6c5b17	84
2023-06-26 14:25:37	3M	8	98	21	93c49e41-868b-4b5b-b725-06b4b1fb0a09	68
2023-06-26 14:25:37	Vyaire	8	92	18	e46abe8d-b02f-43e6-91bf-c4700719f846	10
2023-06-26 14:25:37	Vyaire	8	93	16	b3946e38-6292-4afd-8695-ada5cc09d0dd	15
2023-06-26 14:25:37	GE	8	93	10	e3f7390d-1e68-4def-9dae-5c98b1d85d9d	3
2023-06-26 14:25:37	Vyaire	8	98	17	a3917233-fe7f-4105-8728-779bd7ab1379	8
2023-06-26 14:25:37	Getinge	8	98	16	06a8e8ff-cae4-4438-9714-33324f1524c9	93
2023-06-26 14:25:37	Getinge	6	96	14	7af06237-bbdf-4615-b9ac-05d05d484ba0	13
2023-06-26 14:25:37	3M	8	93	8	bf9985f6-04b8-442b-b7f9-24b1db6b5a37	81
2023-06-26 14:25:37	Getinge	6	97	28	e67f4220-3070-4951-b4e0-c86b7489de10	19
2023-06-26 14:25:37	3M	6	92	15	77954206-535e-4ef8-a1fe-0da5ece049a6	31
2023-06-26 14:25:37	Vyaire	7	94	25	81303a43-6206-46cb-851f-fc3986491bf9	32

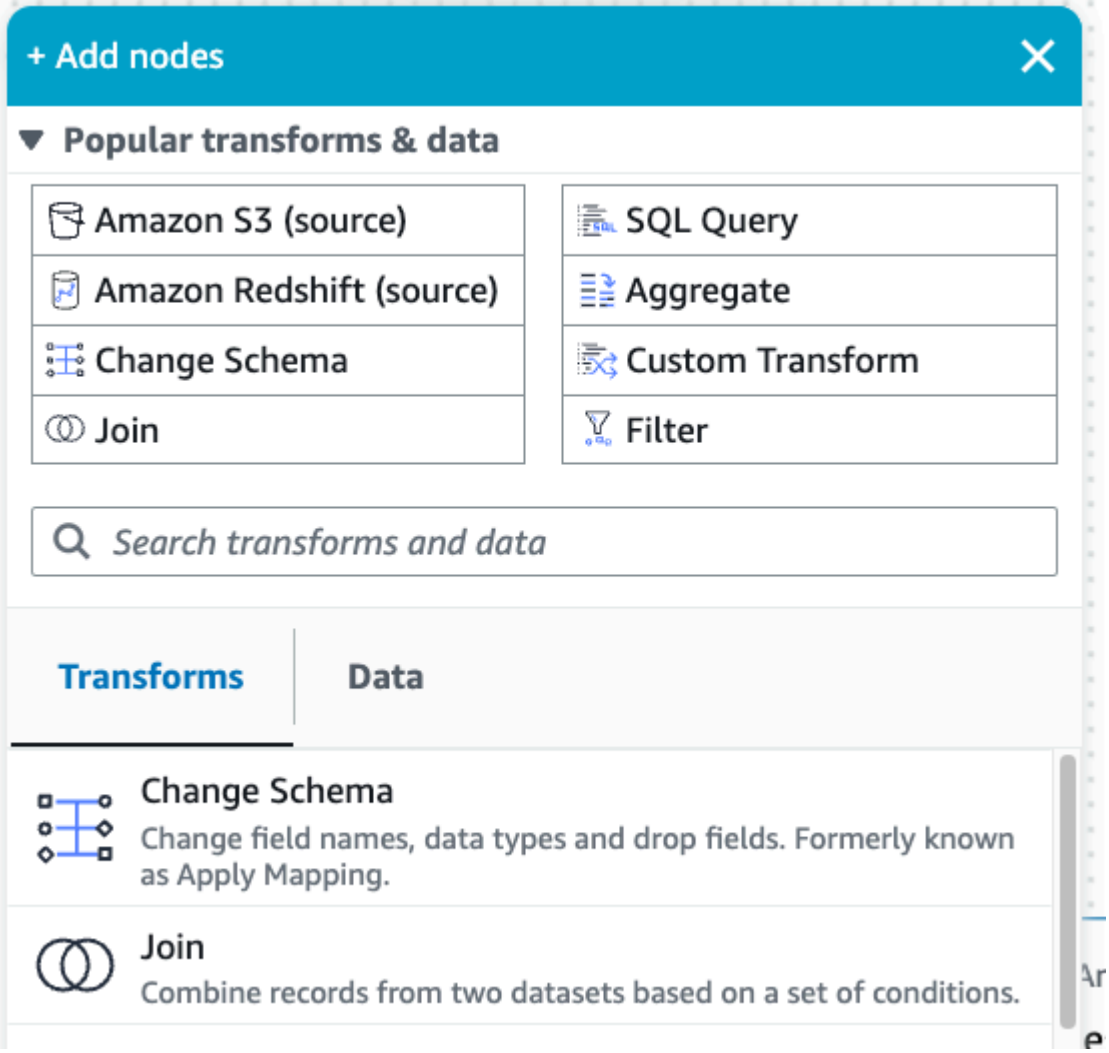
推測されたスキーマは [出力スキーマ] タブでも確認できます。



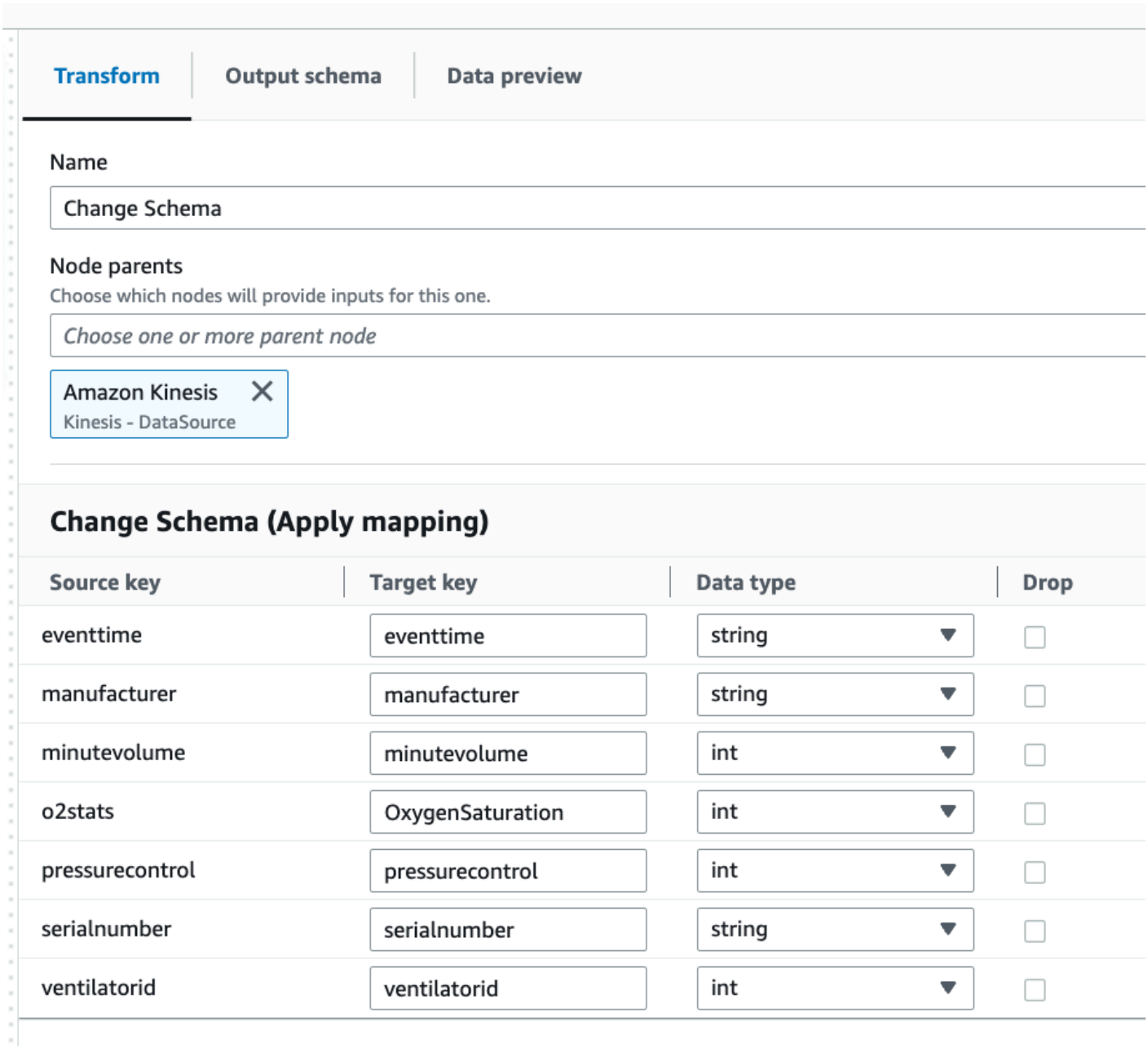
Key	Data type
eventtime	string
manufacturer	string
minutevolume	long
o2stats	long
pressurecontrol	long
serialnumber	string
ventilatorid	long

変換を実行し、変換した結果を Amazon S3 に保存する

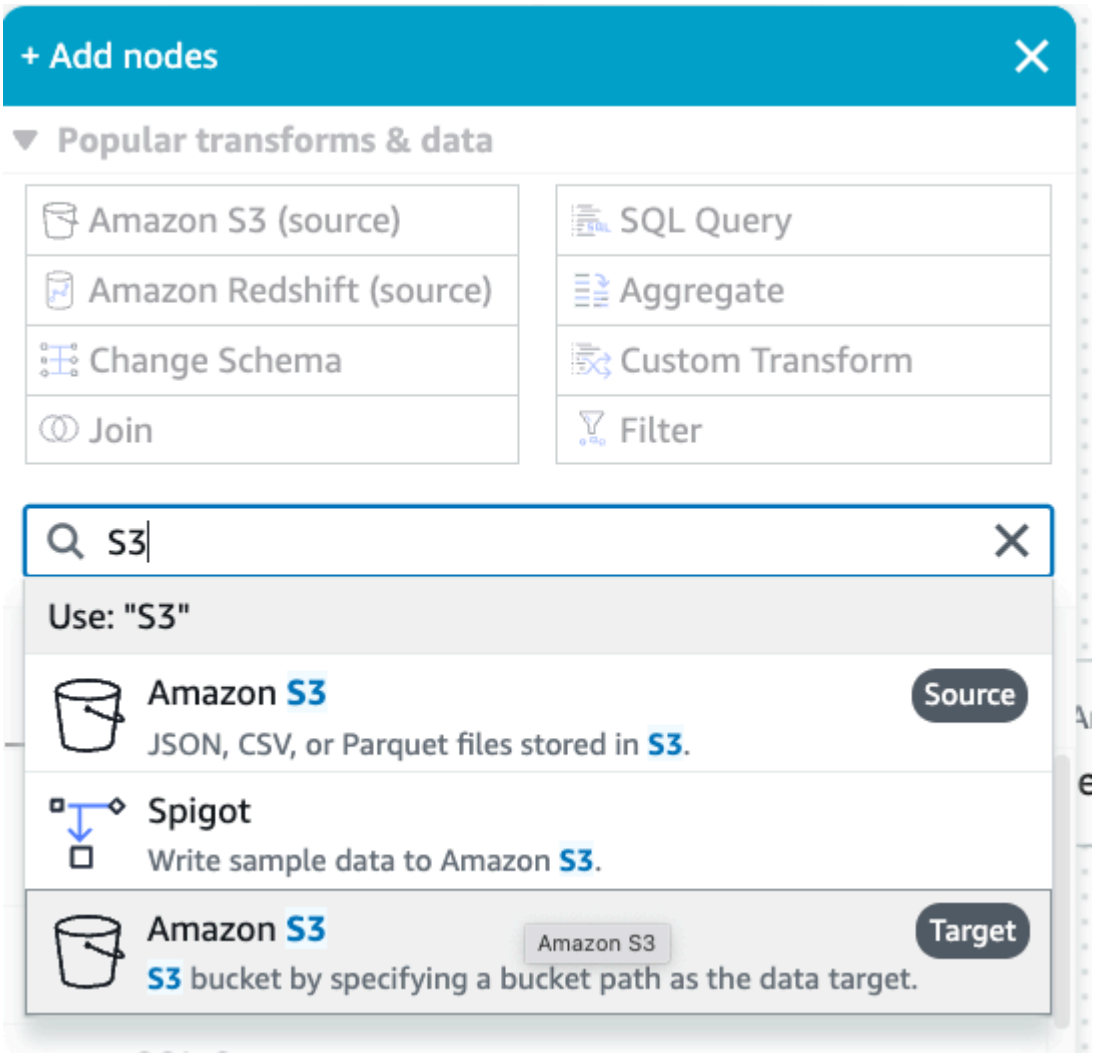
1. ソースノードを選択した状態で、左上のプラスアイコンをクリックして [変換] ステップを追加します。
2. [スキーマ変更] ステップを選択します。



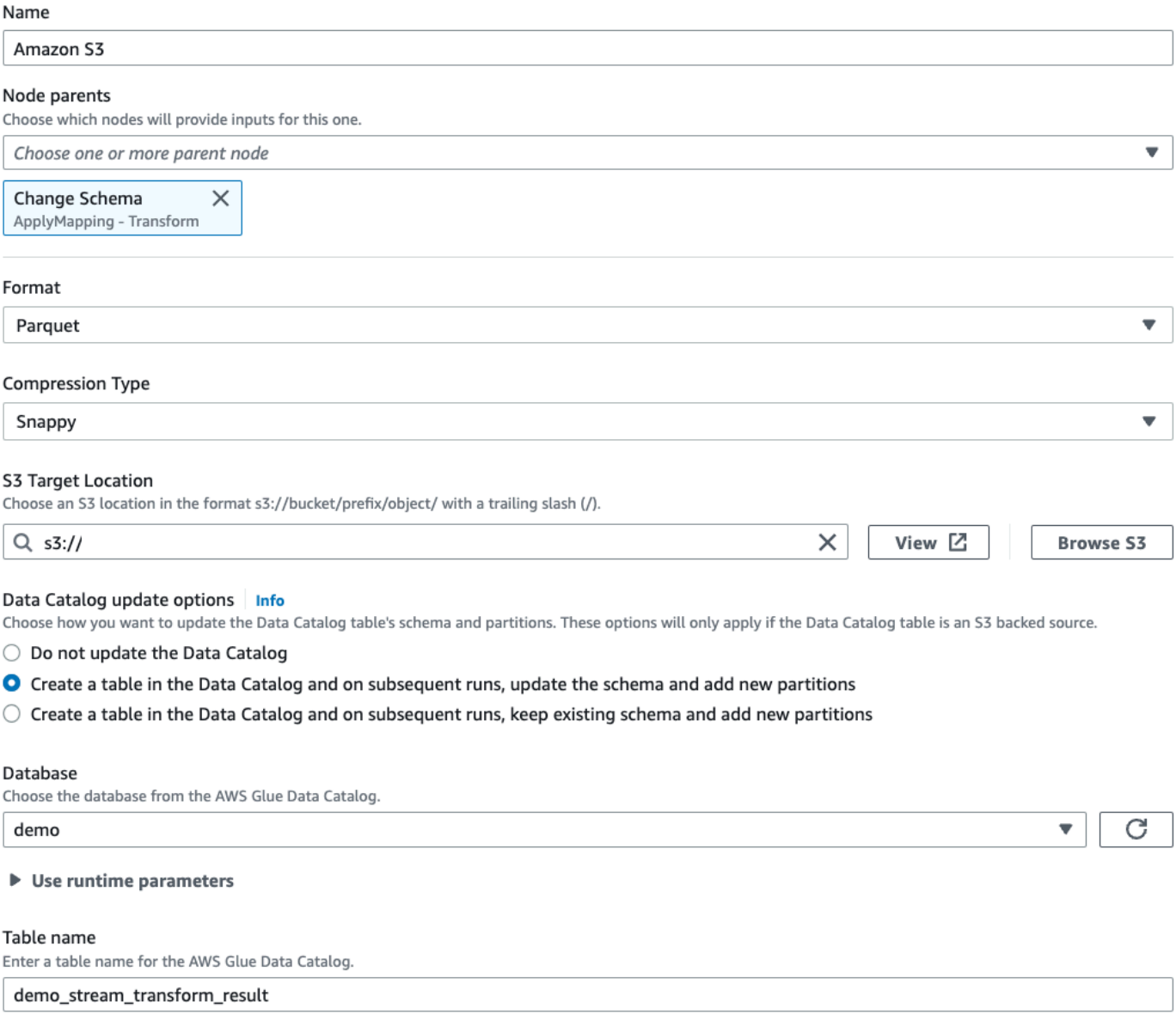
3. このステップでは、フィールドの名前を変更し、フィールドのデータ型を変換することができます。`o2stats` 列の名前を `OxygenSaturation` に変更し、すべての `long` データ型を `int` に変換します。



4. プラスアイコンをクリックして [Amazon S3] ターゲットを追加します。検索ボックスに「S3」と入力し、[Amazon S3 - ターゲット] 変換ステップを選択します。



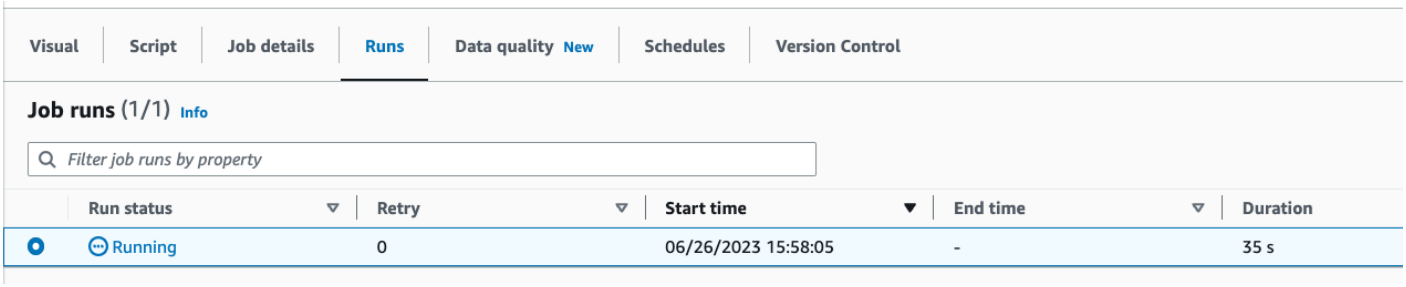
5. ターゲットファイル形式として [Parquet] を選択します。
6. 圧縮タイプとして [Snappy] を選択します。
7. CloudFormation テンプレート `streaming-tutorial-s3-target-{AWS::AccountId}` によって作成された [S3 ターゲットの場所] を入力します。
8. [データカタログにテーブルを作成し、それ以降の実行時にスキーマを更新して新しいパーティションを追加する] を選択します。
9. Amazon S3 ターゲットテーブルのスキーマを保存するターゲットの [データベース] と [テーブル名] を入力します。



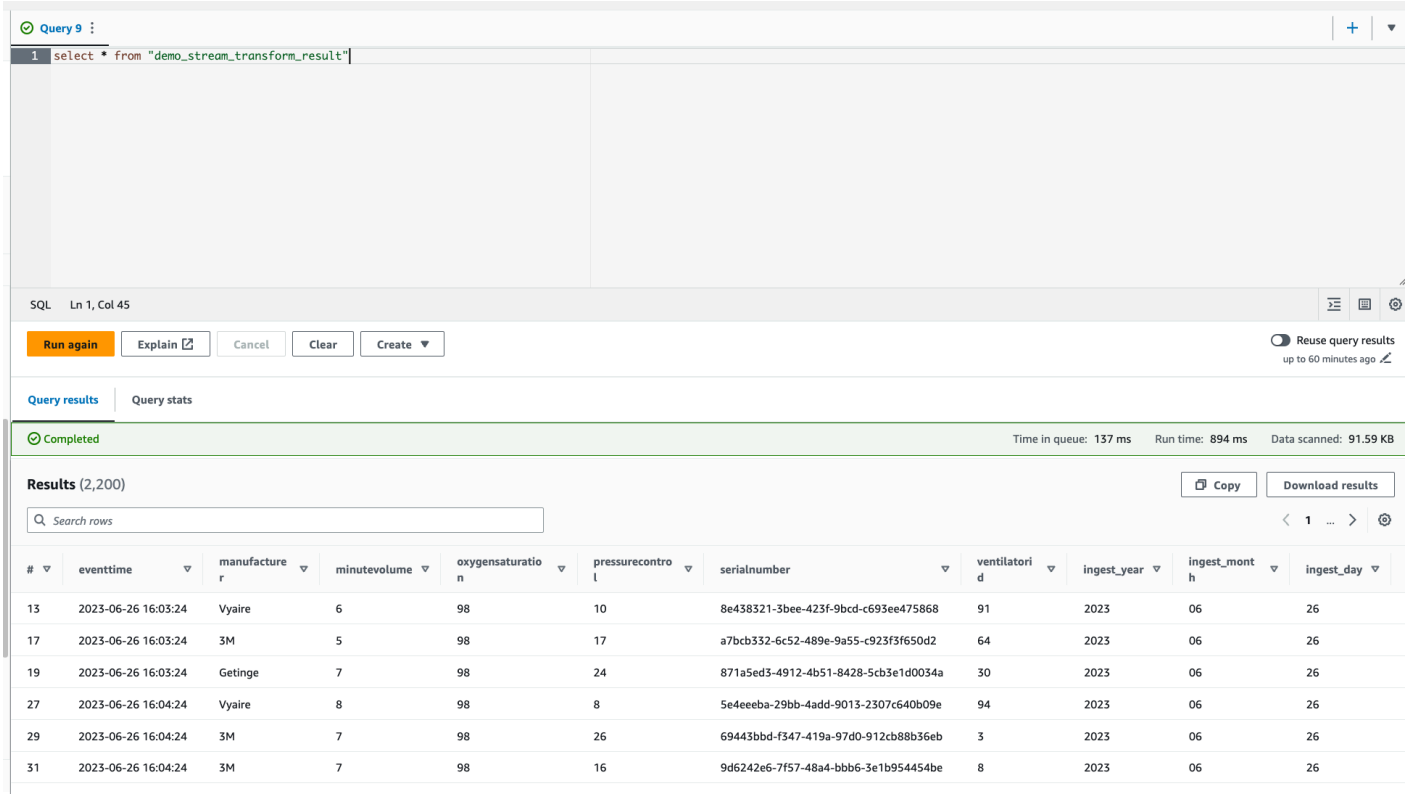
10[スクリプト] タブをクリックすると、生成されたコードが表示されます。

11右上の [保存] をクリックして ETL コードを保存し、[実行] をクリックして AWS Glue Streaming ジョブを開始します。

[実行ステータス] は [実行] タブで確認できます。ジョブを 3 ~ 5 分間実行してから、ジョブを停止します。



12Amazon Athena で作成された新しいテーブルを確認します。



チュートリアル: AWS Glue Studio ノートブックを使用して最初のストリーミングワークロードを構築する

このチュートリアルでは、AWS Glue Studio ノートブックを活用して ETL ジョブをインタラクティブに構築および改良し、ほぼリアルタイムのデータ処理を行う方法について説明します。このガイドでは、AWS Glue が初めての方も、スキルセットの向上を目指している方も、AWS Glue インタラクティブセッションノートブックの可能性を最大限に引き出すことができるよう、順を追って説明していきます。

AWS Glue Streaming では、Amazon Kinesis Data Streams、Apache Kafka、Amazon Managed Streaming for Apache Kafka (Amazon MSK) などのストーミングソースからのデータを消費し、連続的に実行するストリーミング抽出、変換、ロード (ETL) ジョブを作成できます。

前提条件

このチュートリアルを実行するには、AWS Glue、Amazon Kinesis、Amazon S3、Amazon Athena、AWS CloudFormation、AWS Lambda、Amazon Cognito を使用する AWS コンソール権限を持つユーザーが必要です。

Amazon Kinesis からストリーミングデータを消費する

トピック

- Kinesis Data Generator でモックデータを生成する
- AWS Glue Studio で AWS Glue Streaming ジョブを作成する
- クリーンアップ
- 結論

Kinesis Data Generator でモックデータを生成する

> **Note**
>
> 前の「チュートリアル: AWS Glue Studio を使用して最初のストリーミングワークロードを構築する」を既に完了している場合は、Kinesis Data Generator がアカウントに既にインストールされているため、以下のステップ 1～8 をスキップしてセクション「AWS Glue Studio で AWS Glue Streaming ジョブを作成する」に進んでください。

Kinesis Data Generator (KDG) を使用して JSON 形式のサンプルデータを合成的に生成できます。完全な手順と詳細については、ツールのドキュメントを参照してください。

1. 開始するには、Launch Stack をクリックして、ご使用の AWS 環境で AWS CloudFormation テンプレートを実行します。

> **Note**
>
> Kinesis Data Generator の Amazon Cognito ユーザーなどのリソースが AWS アカウントに既に存在しているために、CloudFormation テンプレートの実行が失敗する場合があります。これは、別のチュートリアルやブログで既に設定していたことが原因である可能性があります。これに対処するには、新しい AWS アカウントでテンプレートの実行をやり直すか、別の AWS リージョンを試してみてください。これらの方法により、既存のリソースと競合することなくチュートリアルを実行できるようになります。

このテンプレートにより、Kinesis データストリームと Kinesis Data Generator アカウントがプロビジョニングされます。

2. KDG が認証に使用する [ユーザー名] と [パスワード] を入力します。後で使用するために、ユーザー名とパスワードをメモしておきます。
3. 最後のステップまで [次へ] を選択していきます。IAM リソースの作成を承認します。パスワードが最小要件を満たしていないなど、画面上部にエラーがないか確認し、テンプレートをデプロイします。
4. スタックの [出力] タブに移動します。テンプレートがデプロイされると、生成されたプロパティ KinesisDataGeneratorUrl が表示されます。その URL をクリックします。
5. メモしておいた [ユーザー名] と [パスワード] を入力します。
6. 使用しているリージョンを選択し、Kinesis ストリーム `GlueStreamTest-{AWS::AccountId}` を選択します。
7. 以下のテンプレートを入力します。

```
{
    "ventilatorid": {{random.number(100)}},
    "eventtime": "{{date.now("YYYY-MM-DD HH:mm:ss")}}",
    "serialnumber": "{{random.uuid}}",
    "pressurecontrol": {{random.number(
        {
            "min":5,
            "max":30
        }
    )}},
    "o2stats": {{random.number(
        {
            "min":92,
```

```
            "max":98
        }
    )}},
    "minutevolume": {{random.number(
        {
            "min":5,
            "max":8
        }
    )}},
    "manufacturer": "{{random.arrayElement(
        ["3M", "GE","Vyaire", "Getinge"]
    )}}"
}
```

[テストテンプレート] を使用してモックデータを表示し、[データを送信する] を使用してモックデータを Kinesis に取り込むことができるようになりました。

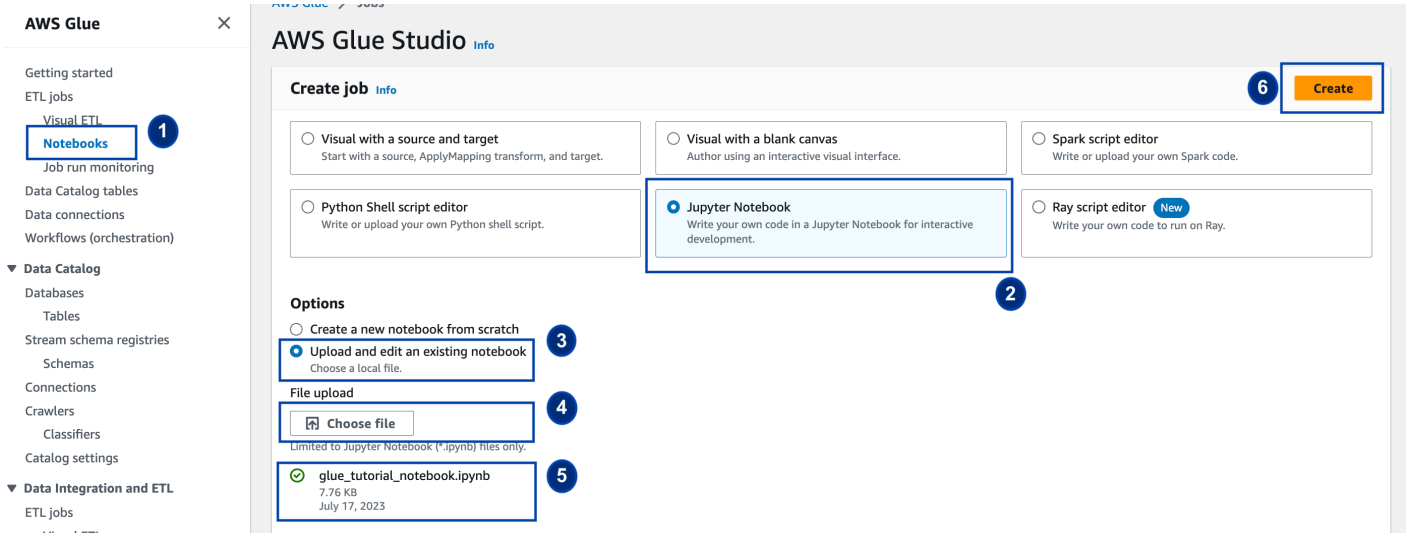
8. [データを送信する] をクリックして、5 ~ 10K のレコードを Kinesis に生成します。

AWS Glue Studio で AWS Glue Streaming ジョブを作成する

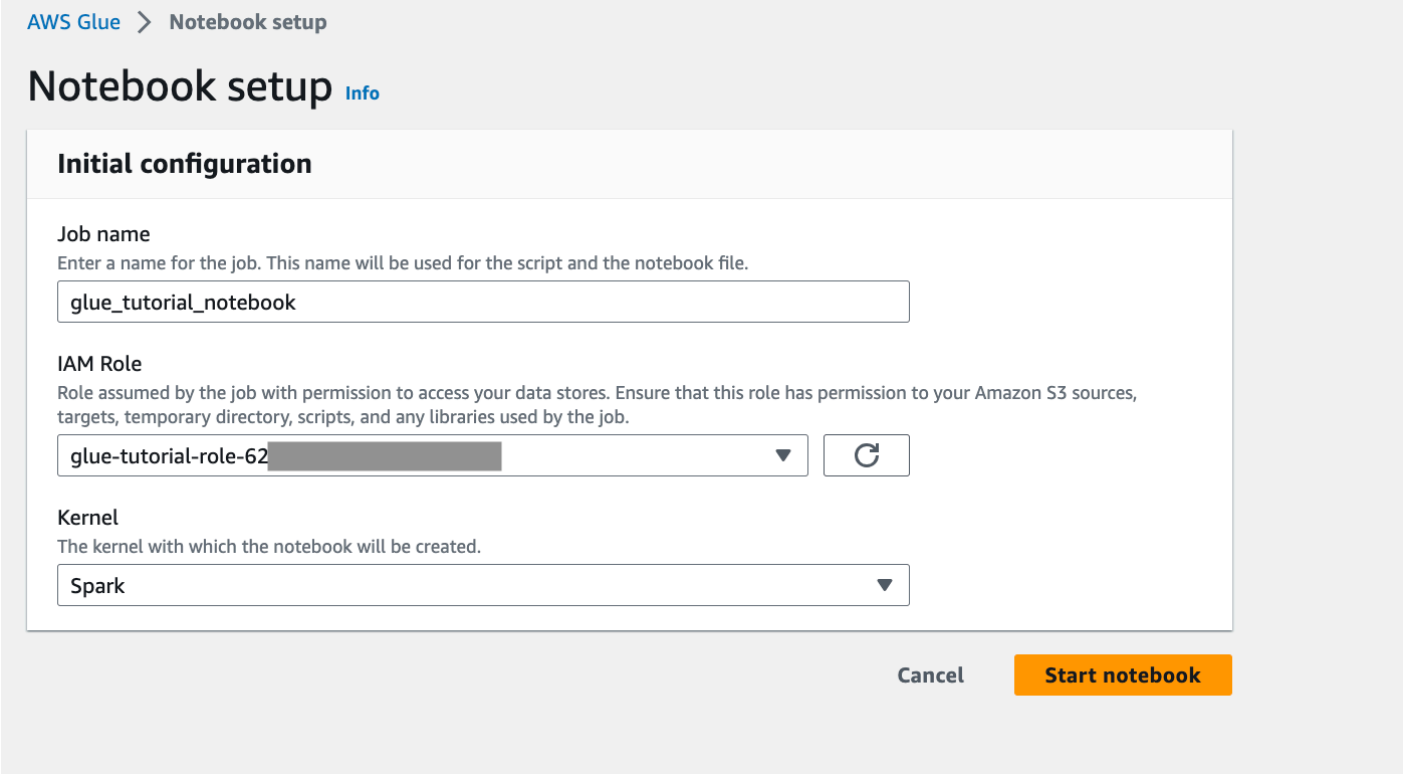
AWS Glue Studio は、データ統合パイプラインの設計、オーケストレーション、監視のプロセスを簡素化するビジュアルインターフェイスです。これにより、ユーザーは大量のコードを記述しなくてもデータ変換パイプラインを構築できます。ビジュアルジョブオーサリングエクスペリエンスとは別に、AWS Glue Studio には、このチュートリアルで使用する AWS Glue インタラクティブセッションによって支援された Jupyter Notebook も含まれています。

AWS Glue Streaming インタラクティブセッションジョブを設定する

1. 提供されているノートブックファイルをダウンロードして、ローカルディレクトリに保存します。
2. AWS Glue コンソールを開き、左側のペインで [ノートブック] > [Jupyter Notebook] > [既存のノートブックをアップロードして編集する] の順にクリックします。前のステップのノートブックをアップロードし、[作成] をクリックします。



3. ジョブに対して名前とロールを指定し、デフォルトの Spark カーネルを選択します。次に [ノートブックの開始] をクリックします。[IAM ロール] では、CloudFormation テンプレートによってプロビジョニングされたロールを選択します。これは CloudFormation の [出力] タブで確認できます。



ノートブックには、このチュートリアルを続けるために必要なすべての手順が記載されています。ノートブックに記載されている手順を実行することも、このチュートリアルに従ってジョブの開発を続けることもできます。

ノートブックのセルを実行する

1. (オプション) 最初のコードセルの %help は、使用可能なすべてのノートブックマジックをリストします。ここではこのセルはスキップしても構いませんが、自由に試してみてください。
2. 次のコードブロック %streaming から始めます。このマジックにより、ジョブタイプがストリーミングに設定され、AWS Glue Streaming ETL ジョブを開発、デバッグ、デプロイできるようになります。
3. 次のセルを実行して AWS Glue インタラクティブセッションを作成します。出力セルには、セッションの作成を確認するメッセージが表示されます。

Run this cell to set up and start your interactive session.

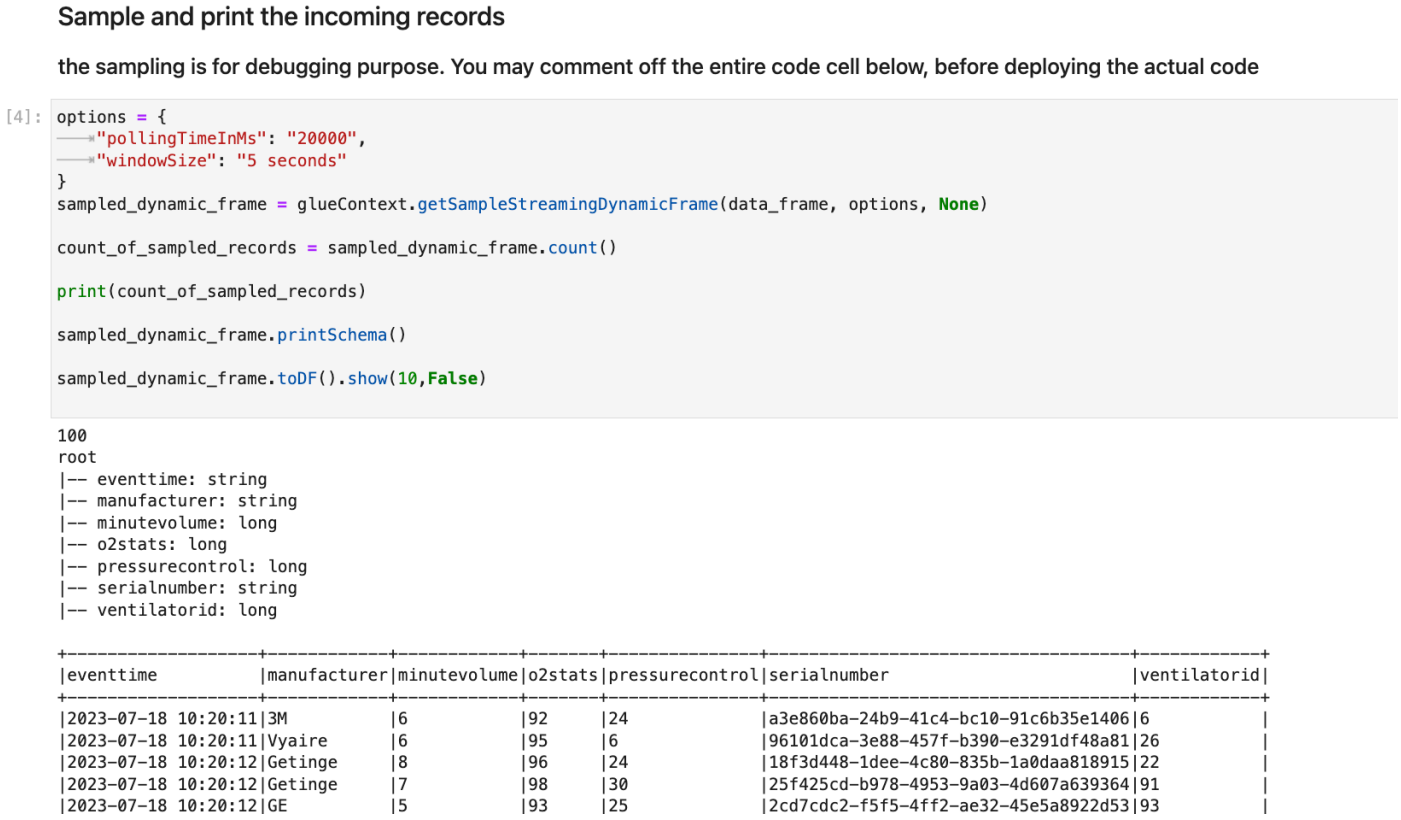
```
[1]: %glue_version 3.0

import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job
from awsglue import DynamicFrame
from datetime import datetime
from pyspark.sql.types import StructType, StructField, StringType, LongType
from pyspark.sql.functions import lit,col,from_json
import boto3

sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)

Setting Glue version to: 3.0
Authenticating with environment variables and user-defined glue_role_arn: arn:aws:iam::62          0:role/glue-tutorial-role
Trying to create a Glue session for the kernel.
Worker Type: G.1X
Number of Workers: 5
Session ID:                                    af
Job Type: gluestreaming
Applying the following default arguments:
--glue_kernel_version 0.37.3
--enable-glue-datacatalog true
Waiting for session 4                                  to get into ready status...
Session 48                                    has been created.
```

4. 次のセルでは変数を定義します。値をジョブに適した値に置き換え、セルを実行します。例:

```
output_database_name="default"
output_table_name="test_stream_001"

account_id = boto3.client("sts").get_caller_identity()["Account"]
region_name=boto3.client('s3').meta.region_name
stream_arn_name = "arn:aws:kinesis:{}:{}:stream/GlueStreamTest-{}".format(region_name,account_id,account_id)
s3_bucket_name = "streaming-tutorial-s3-target-{}".format(account_id)

output_location = "s3://{}/streaming_output/".format(s3_bucket_name)
checkpoint_location = "s3://{}/checkpoint_location/".format(s3_bucket_name)
```

5. データは既に Kinesis Data Streams にストリーミングされているため、次のセルではストリームからの結果を消費します。次のセルを実行します。print ステートメントがないため、このセルからの期待される出力はありません。

6. 次のセルでは、サンプルセットを取得して受信ストリームを調べ、そのスキーマと実際のデータを出力します。例:



7. 次に、実際のデータ変換ロジックを定義します。セルは、マイクロバッチごとにトリガーされる `processBatch` メソッドで構成されます。セルを実行します。概説すると、受信ストリームに対して次のことを行います。

   a. 入力列のサブセットを選択します。

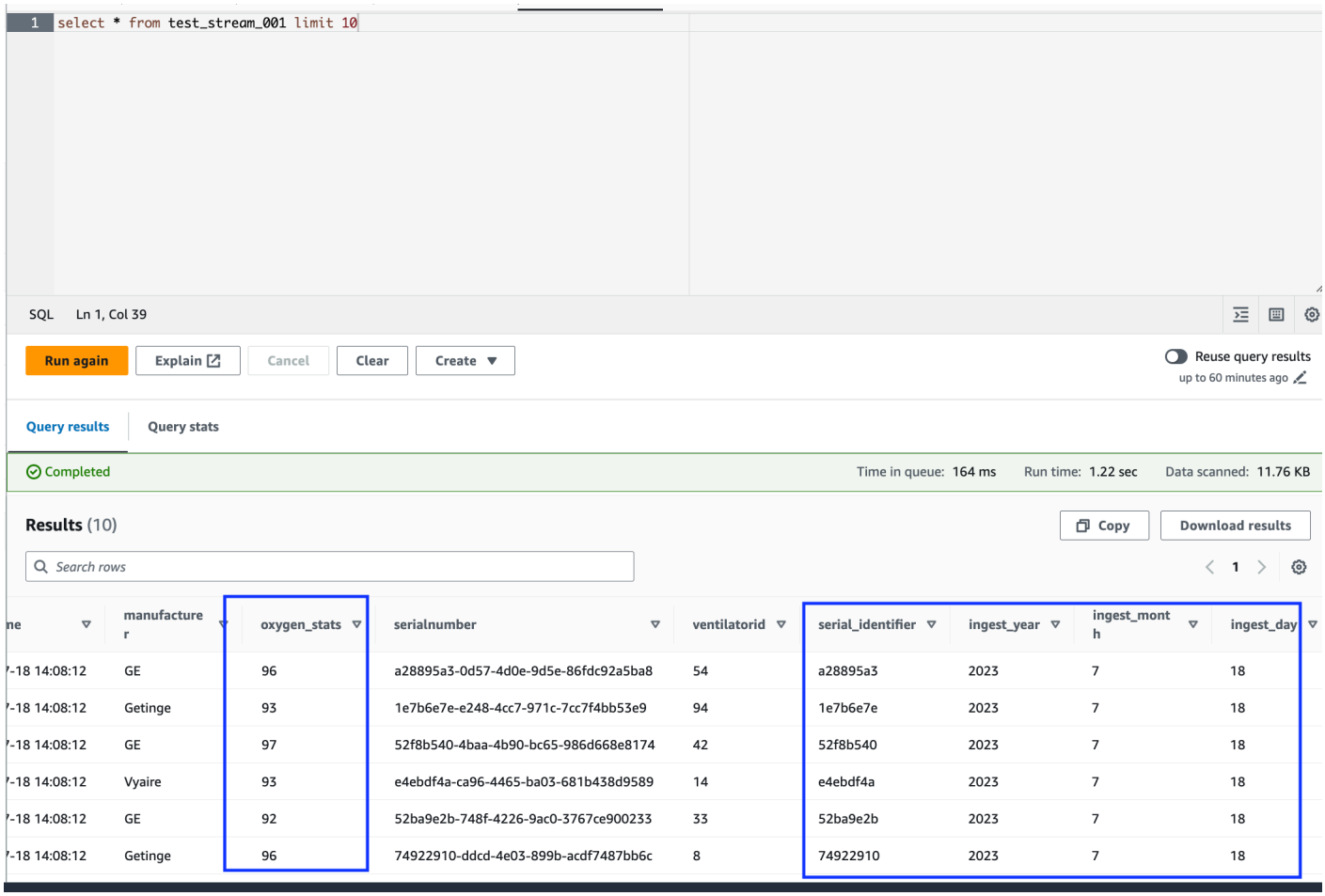
   b. 列の名前を変更します (o2stats を oxygen_stats に変更します)。

   c. 新しい列 (serial_identifier、ingest_year、ingest_month、ingest_day) を取得します。

   d. 結果を Amazon S3 バケットに保存し、パーティション化された AWS Glue カタログテーブルも作成します。

8. 最後のセルでは、10 秒ごとにプロセスバッチをトリガーします。セルを実行し、Amazon S3 バケットと AWS Glue カタログテーブルに入力されるまで約 30 秒待ちます。

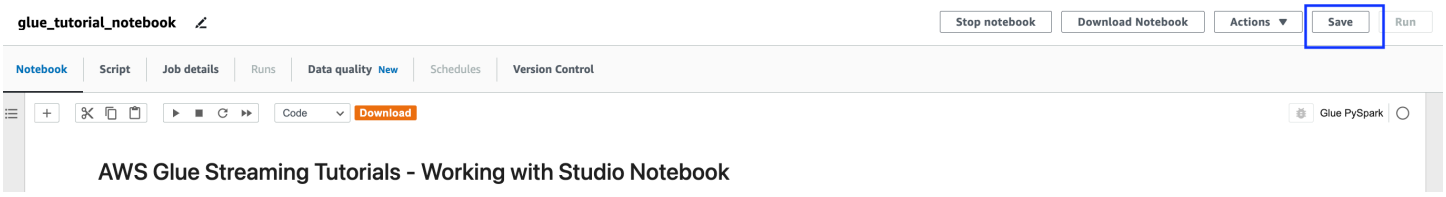
9. 最後に、Amazon Athena クエリエディタを使用して、保存されたデータを参照します。名前が変更された列と新しいパーティションが表示されます。



ノートブックには、このチュートリアルを続けるために必要なすべての手順が記載されています。ノートブックに記載されている手順を実行することも、このチュートリアルに従ってジョブの開発を続けることもできます。

AWS Glue ジョブを保存して実行する

インタラクティブセッションノートブックを使用したアプリケーションの開発とテストが完了したら、ノートブックインターフェイスの上部にある [保存] をクリックします。保存後は、アプリケーションをジョブとして実行することもできます。



クリーンアップ

アカウントに追加料金が発生しないように、手順の一部として開始したストリーミングジョブを停止してください。これを行うには、ノートブックを停止します。これでセッションが終了します。Amazon S3 バケットを空にして、前にプロビジョニングした AWS CloudFormation スタックを削除します。

結論

このチュートリアルでは、AWS Glue Studio ノートブックを使用して次のことを行う方法を説明しました。

- ノートブックを使用してストリーミング ETL ジョブを作成する
- 受信データストリームをプレビューする
- AWS Glue ジョブを公開することなくコーディングして問題を修正する
- エンドツーエンド作業コードを確認し、デバッグを除去して、ノートブックからステートメントやセルを出力する
- コードを AWS Glue ジョブとして公開する

このチュートリアルの目的は、AWS Glue Streaming とインタラクティブセッションを使用する実践的な使用体験を提供することです。AWS Glue Streaming の個々のユースケースの参考としてご利用いただくことをお勧めします。詳細については、「AWS Glue のインタラクティブセッションの開始方法」を参照してください。

AWS Glue Streaming の概念

以下のセクションでは、AWS Glue Streaming の概念について説明します。

トピック

- AWS Glue Streaming ジョブの構造

AWS Glue Streaming ジョブの構造

AWS Glue Streaming ジョブは Spark ストリーミングパラダイムで動作し、Spark フレームワークの構造化されたストリーミングを活用します。ストリーミングジョブは、特定の時間間隔で常にスト

リーミングデータソースをポーリングし、レコードをマイクロバッチとして取得します。以下のセクションでは、AWS Glue Streaming ジョブのさまざまな部分について説明します。



forEachBatch

`forEachBatch` メソッドは、AWS Glue Streaming ジョブ実行のエントリポイントです。AWS Glue Streaming ジョブは、`forEachBatch` メソッドを使用して、イテレーターのように機能するデータをポーリングします。このデータはストリーミングジョブのライフサイクル中ずっとアクティブなままです。新しいデータがないかストリーミングソースを定期的にポーリングして、最新のデータをマイクロバッチで処理します。

```
glueContext.forEachBatch(
    frame=dataFrame_AmazonKinesis_node1696872487972,
    batch_function=processBatch,
    options={
        "windowSize": "100 seconds",
        "checkpointLocation": args["TempDir"] + "/" + args["JOB_NAME"] + "/
checkpoint/",
    },
)
```

forEachBatch の frame プロパティを設定してストリーミングソースを指定します。この例では、ジョブの作成中に空白のキャンバスに作成したソースノードに、ジョブのデフォルトの DataFrame が入力されます。batch_function プロパティを、マイクロバッチ操作ごとに呼び出す function として設定します。受信データのバッチ変換を処理する関数を定義する必要があります。

ソース

processBatch 関数の最初のステップで、プログラムは、forEachBatch のフレームプロパティとして定義した DataFrame のレコード数を検証します。プログラムは、空でない DataFrame に取り込みタイムスタンプを追加します。data_frame.count()>0 句は、最新のマイクロバッチが空ではなく、さらに処理できる状態にあるかどうかを判断します。

```
def processBatch(data_frame, batchId):
    if data_frame.count() >0:
        AmazonKinesis_node1696872487972 = DynamicFrame.fromDF(
            glueContext.add_ingestion_time_columns(data_frame, "hour"),
            glueContext,
            "from_data_frame",
        )
```

マッピング

プログラムの次のセクションでは、マッピングを適用します。Spark DataFrame で Mapping.apply メソッドを使用すると、データ要素に関する変換ルールを定義できます。通常、名前を変更したり、データ型を変更したり、ソースデータ列でカスタム関数を適用してターゲット列にマッピングしたりすることができます。

```
    #Script generated for node ChangeSchema
    ChangeSchema_node16986872679326 = ApplyMapping.apply(
        frame =  AmazonKinesis_node1696872487972,
        mappings = [
            ("eventtime", "string", "eventtime", "string"),
            ("manufacturer", "string", "manufacturer", "string"),
            ("minutevolume", "long", "minutevolume", "int"),
            ("o2stats", "long", "OxygenSaturation", "int"),
```

```
            ("pressurecontrol", "long", "pressurecontrol", "int"),
            ("serialnumber", "string", "serialnumber", "string"),
            ("ventilatorid", "long", "ventilatorid", "long"),
            ("ingest_year", "string", "ingest_year", "string"),
            ("ingest_month", "string", "ingest_month", "string"),
            ("ingest_day", "string", "ingest_day", "string"),
            ("ingest_hour", "string", "ingest_hour", "string"),
        ],
        transformation_ctx="ChangeSchema_node16986872679326",
    )
        )
```

シンク

このセクションでは、ストリーミングソースから受信したデータセットがターゲットの場所に保存されます。この例では、Amazon S3 の場所にデータを書き込みます。AmazonS3_node_path プロパティの詳細は、キャンバスからのジョブ作成時に使用した設定に基づいて事前入力されます。ユースケースに基づいて updateBehavior を設定し、データカタログテーブルを更新しないか、それ以降の実行時にデータカタログを作成してデータカタログスキーマを更新するか、カタログテーブルを作成してそれ以降の実行時にスキーマ定義を更新しないかを決定できます。

partitionKeys プロパティは、ストレージパーティションオプションを定義します。デフォルトの動作では、ソースセクションで使用できるようになった ingestion_time_columns ごとにデータをパーティション化します。compression プロパティでは、ターゲットへの書き込み時に適用される圧縮アルゴリズムを設定できます。圧縮方式として Snappy、LZO、または GZIP を設定するオプションがあります。enableUpdateCatalog プロパティは、AWS Glue カタログテーブルを更新する必要があるかどうかを制御します。このプロパティで使用できるオプションは True または False です。

```
    #Script generated for node Amazon S3
    AmazonS3_node1696872743449 = glueContext.getSink(
        path =  AmazonS3_node1696872743449_path,
        connection_type = "s3",
        updateBehavior = "UPDATE_IN_DATABASE",
        partitionKeys = ["ingest_year", "ingest_month", "ingest_day", "ingest_hour"],
        compression = "snappy",
        enableUpdateCatalog = True,
        transformation_ctx = "AmazonS3_node1696872743449",
    )
```

AWS Glue カタログシンク

ジョブのこのセクションでは、AWS Glue カタログテーブルの更新動作を制御します。設計中の AWS Glue ジョブに関連する AWS Glue カタログデータベース名とテーブル名ごとに `catalogDatabase` プロパティと `catalogTableName` プロパティを設定します。`setFormat` プロパティを使用してターゲットデータのファイル形式を定義できます。この例では、データを parquet 形式で保存します。

このチュートリアルを参照して AWS Glue Streaming ジョブを設定して実行すると、Amazon Kinesis Data Streams で生成されたストリーミングデータは、Snappy 圧縮方式、parquet 形式で、Amazon S3 の場所に保存されます。ストリーミングジョブが正常に実行されると、Amazon Athena でデータをクエリできるようになります。

```
AmazonS3_node1696872743449 = setCatalogInfo(
    catalogDatabase =  "demo", catalogTableName = "demo_stream_transform_result"
)
AmazonS3_node1696872743449.setFormat("glueparquet")
AmazonS3_node1696872743449.writeFormat("ChangeSchema_node16986872679326")
)
```

AWS Glue ストリーミング接続

以下のセクションでは、AWS Glue Streaming の接続の使用方法についての情報を提供します。

トピック

- Kafka 接続の操作
- Kinesis 接続の操作

Kafka 接続の操作

Kafka 接続を使用すると、Data Catalog テーブルに格納されている情報を使用するか、データストリームに直接アクセスするための情報を指定することにより、Kafka データストリームへ読み込む

または書き込むことができます。接続は、Kafka クラスターまたは Amazon Managed Streaming for Apache Kafka クラスターをサポートします。Kafka から Spark DataFrame に情報を読み込み、AWS Glue DynamicFrame に変換できます。DynamicFrames を JSON 形式で Kafka に書き込むことができます。データストリームに直接アクセスする場合は、これらのオプションを使用して、データストリームへのアクセス方法に関する情報を提供します。

`getCatalogSource` または `create_data_frame_from_catalog` を使用して Kafka ストリーミングソースからレコードを消費するか、`getCatalogSink` または `write_dynamic_frame_from_catalog` を使用して Kafka にレコードを書き込み、ジョブに Data Catalog データベースおよびテーブル名の情報があり、それを使用して Kafka ストリーミングソースから読み込むためのいくつかの基本パラメータを取得できる場合。`getSource`、`getCatalogSink`、`getSourceWithFormat`、`getSinkWithFormat`、`createDataF` を使用する場合、ここで説明する接続オプションを使用し、これらの基本パラメータを指定する必要があります。

`GlueContext` クラスで指定されたメソッドの次の引数を使用し、Kafka の接続オプションを指定できます。

- Scala
  - `connectionOptions`: `getSource`、`createDataFrameFromOptions`、`getSink` で使用
  - `additionalOptions`: `getCatalogSource`、`getCatalogSink` で使用
  - `options`: `getSourceWithFormat`、`getSinkWithFormat` で使用
- Python
  - `connection_options`: `create_data_frame_from_options`、`write_dynamic_frame_from_options` で使用
  - `additional_options`: `create_data_frame_from_catalog`、`write_dynamic_frame_from_catalog` で使用
  - `options`: `getSource`、`getSink` で使用

ストリーミング ETL ジョブに関する注意事項と制限事項については、「the section called “ストリーミング ETL に関する注意と制限”」を参照してください。

Kafka の設定

インターネット経由で利用可能な Kafka ストリームに接続するための AWS の前提条件はありません。

AWS Glue Kafka 接続を作成して、接続認証情報を管理できます。詳細については、「the section called “Kafka データストリームの接続の作成”」を参照してください。AWS Glue ジョブ設定で、*connectionName* を追加ネットワーク接続として指定し、メソッド呼び出しで connectionName パラメータに *connectionName* を指定します。

場合によっては、追加の前提条件を設定する必要があります。

- IAM 認証で Amazon Managed Streaming for Apache Kafka を使用する場合は、適切な IAM 設定が必要になります。
- Amazon VPC で Amazon Managed Streaming for Apache Kafka を使用する場合は、適切な Amazon VPC 設定が必要になります。Amazon VPC 接続情報を提供する AWS Glue 接続を作成する必要があります。AWS Glue 接続を追加ネットワーク接続として含めるには、ジョブ設定が必要です。

ストリーミング ETL ジョブの前提条件の詳細については、「the section called “ストリーミング ETL ジョブ”」を参照してください。

例: Kafka ストリームからの読み込み

the section called “forEachBatch” と組み合わせて使用します。

Kafka ストリーミングソースの例 :

```
kafka_options =
    { "connectionName": "ConfluentKafka",
      "topicName": "kafka-auth-topic",
      "startingOffsets": "earliest",
      "inferSchema": "true",
      "classification": "json"
    }
data_frame_datasource0 =
 glueContext.create_data_frame.from_options(connection_type="kafka",
 connection_options=kafka_options)
```

例: Kafka ストリームへの書き込み

Kafka への書き込み例:

getSink メソッドを使った例:

```
data_frame_datasource0 =
glueContext.getSink(
 connectionType="kafka",
 connectionOptions={
  JsonOptions("""{
   "connectionName": "ConfluentKafka",
   "classification": "json",
   "topic": "kafka-auth-topic",
   "typeOfData": "kafka"}
  """)},
 transformationContext="dataframe_ApacheKafka_node1711729173428")
 .getDataFrame()
```

write_dynamic_frame.from_options メソッドを使った例:

```
kafka_options =
    { "connectionName": "ConfluentKafka",
      "topicName": "kafka-auth-topic",
      "classification": "json"
    }
data_frame_datasource0 =
 glueContext.write_dynamic_frame.from_options(connection_type="kafka",
 connection_options=kafka_options)
```

Kafka 接続オプションのリファレンス

読み込むとき、"connectionType": "kafka" に次の接続オプションを使用します。

- "bootstrap.servers" (必須) ブートストラップサーバーの URL のリスト (例: b-1.vpc-test-2.o4q88o.c6.kafka.us-east-1.amazonaws.com:9094)。このオプションは API 呼び出しで指定するか、データカタログ内のテーブルメタデータで定義する必要があります。
- "security.protocol" (必須) ブローカーと通信するために使用されるプロトコル。使用できる値は、"SSL" または "PLAINTEXT" です。
- "topicName" (必須) サブスクライブするトピックのカンマ区切りリスト。"topicName"、"assign"、または "subscribePattern" の中から、いずれか 1 つのみを指定する必要があります。
- "assign": (必須) 消費する特定の TopicPartitions を指定する JSON 文字列。"topicName"、"assign"、または "subscribePattern" の中から、いずれか 1 つのみを指定する必要があります。

例: '{"topicA":[0,1],"topicB":[2,4]}'

- "subscribePattern": (必須) サブスクライブする先のトピックリストを識別する Java の正規表現文字列。"topicName"、"assign"、または "subscribePattern" の中から、いずれか 1 つのみを指定する必要があります。

  例: 'topic.*'

- "classification" (必須) レコード内のデータで使用されるファイル形式。データカタログを通じて提供されていない限り、必須です。
- "delimiter" (オプション) classification が CSV の場合に使用される値の区切り文字。デフォルトは「,」です。
- "startingOffsets": (オプション) Kafka トピック内で、データの読み取りを開始する位置 使用できる値は、"earliest" または "latest" です。デフォルト値は "latest" です。
- "startingTimestamp": (オプション、AWS Glue バージョン 4.0 以降でのみサポート) Kafka トピック内で、データの読み込みを開始するレコードのタイムスタンプ。指定できる値は UTC 形式 (yyyy-mm-ddTHH:MM:SSZ のパターン) のタイムスタンプ文字列です (Z は UTC タイムゾーンのオフセットを +/- で表します。例: 「2023-04-04T08:00:00-04:00」)。

  注: AWS Glue ストリーミングスクリプトの接続オプションリストには、「startingOffsets」または「startingTimestamp」のいずれかしか表示できません。これらのプロパティの両方を含めると、ジョブが失敗します。

- "endingOffsets": (オプション) バッチクエリの終了位置。設定が可能な値は、"latest" または、各 TopicPartition の終了オフセットを指定する JSON 文字列のいずれかです。

  JSON 文字列の場合、{"topicA":{"0":23,"1":-1},"topicB":{"0":-1}} の形式を使用します。オフセットとして値 -1 を設定する場合、"latest" の意味になります。

- "pollTimeoutMs": (オプション) Spark ジョブエグゼキュータで、Kafka からデータをポーリングする際のタイムアウト値 (ミリ秒単位)。デフォルト値は 600000 です。
- "numRetries": (オプション) Kafka オフセットのフェッチが失敗したと判断される前の再試行回数。デフォルト値は 3 です。
- "retryIntervalMs": (オプション) Kafka オフセットのフェッチを開始するまでの待機時間 (ミリ秒)。デフォルト値は 10 です。
- "maxOffsetsPerTrigger": (オプション) 処理されるオフセットの最大数を、トリガー間隔ごとのレート上限で指定する値。指定されたオフセットの合計数は、異なるボリュームの topicPartitions 間で均等に分割されます。デフォルト値は「null」です。この場合、コンシューマーは既知の最新のオフセットまで、すべてのオフセットを読み取ります。

- "minPartitions": (オプション) Kafka から読み取ることを想定する、最小のパーティション数。デフォルト値は「null」です。これは、Spark パーティションの数が Kafka パーティションの数に等しいことを意味します。
- "includeHeaders": (オプション) Kafka ヘッダーを含めるかどうかを決定します。このオプションが「true」に設定されている場合、データ出力には、「glue_streaming_kafka_headers」という名前で Array[Struct(key: String, value: String)] 型の列が追加されます。デフォルト値は『false』です。このオプションは、AWS Glue バージョン 3.0 以降でのみ使用可能です。
- "schema": (inferSchema に false を設定した場合は必須) ペイロードの処理に使用するスキーマ。分類が avro である場合、提供されるスキーマには Avro スキーマ形式を使用する必要があります。分類が avro 以外の場合、提供されるスキーマは DDL スキーマ形式である必要があります。

  以下に、スキーマの例を示します。

  Example in DDL schema format

  ```
  'column1' INT, 'column2' STRING , 'column3' FLOAT
  ```

  Example in Avro schema format

  ```
  {
  "type":"array",
  "items":
  {
  "type":"record",
  "name":"test",
  "fields":
  [
    {
      "name":"_id",
      "type":"string"
    },
    {
      "name":"index",
      "type":
      [
        "int",
        "string",
        "float"
      ]
    }
  ]
  ```

```
}
}
```

- "inferSchema": (オプション) デフォルト値は「false」です。「true」に設定すると、実行時に、スキーマが foreachbatch 内のペイロードから検出されます。
- "avroSchema": (非推奨) Avro 形式を使用する場合に、Avro データのスキーマを指定するために使用されるパラメータです。このパラメータは非推奨となりました。schema パラメータを使用します。
- "addRecordTimestamp": (オプション) このオプションを「true」に設定すると、トピックが対応するレコードを受信した時刻を表示する「__src_timestamp」という列が、データ出力に追加で表示されます。デフォルト値は、「false」です。このオプションは AWS Glue のバージョン 4.0 以降でサポートされています。
- "emitConsumerLagMetrics": (オプション) このオプションを「true」に設定すると、バッチごとに、トピックが受信した最も古いレコードと、それが AWS Glue で CloudWatch に到着した時間との、間隔のメトリクスが出力されます。このメトリクスの名前は「glue.driver.streaming.maxConsumerLagInMs」です。デフォルト値は、「false」です。このオプションは AWS Glue バージョン 4.0 以降でサポートされています。

書き込むとき、"connectionType": "kafka" に次の接続オプションを使用します。

- "connectionName" (必須) Kafka クラスターへの接続に使用される AWS Glue 接続の名前 (Kafka ソースと同様)。
- "topic" (必須) トピック列が存在する場合、トピック設定オプションが設定されていない限り、指定された行を Kafka に書き込む際にトピック列の値がトピックとして使用されます。つまり、topic 設定オプションはトピック列をオーバーライドします。
- "partition" (オプション) 有効なパーティション番号が指定された場合、レコードの送信時にその partition が使用されます。

  パーティションが指定されずに key が存在する場合、キーのハッシュを使用してパーティションが選択されます。

  key と partition のどちらも存在しない場合、そのパーティションに少なくとも batch.size バイトが生成された際に変更内容のスティッキーパーティション分割に基づいてパーティションが選択されます。
- "key" (オプション) partition が null の場合、パーティション分割に使用されます。

- "classification" (オプション) レコードのデータに使用されるファイル形式。JSON、CSV、Avro のみをサポートしています。

  Avro 形式を使用すると、シリアル化するためにカスタムの AvroSchema を提供できますが、シリアル化解除する場合にもソースに提供する必要があることに注意してください。それ以外の場合、デフォルトで Apache AvroSchema を使用してシリアル化します。

さらに、Kafka プロデューサーの設定パラメーターを更新することにより、必要に応じて Kafka シンクを微調整できます。接続オプションには許可リストがないことに注意してください。すべてのキー値のペアはそのままシンクに保持されます。

ただし、有効にならないオプションの小さな拒否リストがあります。詳細については、「Apache 固有の設定」を参照してください。

Kinesis 接続の操作

Kinesis 接続を使用すると、データカタログ テーブルに保存されている情報を使用して、またはデータ ストリームに直接アクセスするための情報を提供することで、Amazon Kinesis データ ストリームの読み取りと書き込みを行うことができます。Kinesis から Spark DataFrame に情報を読み取り、それを AWS Glue DynamicFrame に変換できます。DynamicFrames は JSON 形式で Kinesis に書き込むことができます。データストリームに直接アクセスする場合は、これらのオプションを使用して、データストリームへのアクセス方法に関する情報を提供します。

`getCatalogSource` または `create_data_frame_from_catalog` を使用して Kinesis ストリーミングソースからレコードを消費する場合、ジョブはデータカタログデータベースとテーブル名の情報を持っており、それを使用して Kinesis ストリーミングソースから読み込むためのいくつかの基本パラメータを取得することができます。`getSource`、`getSourceWithFormat`、`createDataFrameFromOptions`、または `create_data_frame_from_options` を使用している場合、ここで説明する接続オプションを使用して、これらの基本パラメータを指定する必要があります。

Kinesis の接続オプションは、`GlueContext` クラス内の指定されたメソッドに対して以下の引数で指定することができます。

- Scala
  - `connectionOptions`: `getSource`、`createDataFrameFromOptions`、`getSink` で使用
  - `additionalOptions`: `getCatalogSource`、`getCatalogSink` で使用
  - `options`: `getSourceWithFormat`、`getSinkWithFormat` で使用

- Python
  - `connection_options`: `create_data_frame_from_options`、`write_dynamic_frame_from_options` で使用
  - `additional_options`: `create_data_frame_from_catalog`、`write_dynamic_frame_from_catalog` で使用
  - `options`: `getSource`、`getSink` で使用

ストリーミング ETL ジョブに関する注意事項と制限事項については、「the section called “ストリーミング ETL に関する注意と制限”」を参照してください。

Kinesis の設定

AWS Glue Spark ジョブで Kinesis データストリームに接続するには、いくつかの前提条件が必要です。

- 読み取る場合、AWS Glue ジョブには、Kinesis データストリームへの読み取りアクセスレベルの IAM 権限が必要です。
- 書き込みの場合、AWS Glue ジョブには、Kinesis データストリームへの書き込みアクセスレベルの IAM 権限が必要です。

場合によっては、追加の前提条件を設定する必要があります。

- AWS Glue ジョブが (通常は他のデータセットに接続するための) 追加のネットワーク接続で設定されていて、その接続のいずれかが Amazon VPC ネットワークオプションを提供している場合、ジョブは Amazon VPC 経由で通信するように指示されます。この場合、Amazon VPC を介して通信するように Kinesis データストリームを設定する必要もあります。そのためには、Amazon VPC と Kinesis データストリームの間にインターフェイス VPC エンドポイントを作成します。詳細については、「インターフェイス VPC エンドポイントと Amazon Kinesis Data Streams の使用」を参照してください。
- 別のアカウントで Amazon Kinesis Data Streams を指定する場合は、クロスアカウントアクセスを許可するようにロールとポリシーを設定する必要があります。詳細については、「例: 別のアカウントで Kinesis Stream から読み込む」を参照してください。

ストリーミング ETL ジョブの前提条件の詳細については、「the section called “ストリーミング ETL ジョブ”」を参照してください。

Kinesis からの読み込み

例: Kinesis ストリームからの読み込み

the section called “forEachBatch” と組み合わせて使用します。

Amazon Kinesis ストリーミングソースの例:

```
kinesis_options =
   { "streamARN": "arn:aws:kinesis:us-east-2:777788889999:stream/fromOptionsStream",
     "startingPosition": "TRIM_HORIZON",
     "inferSchema": "true",
     "classification": "json"
   }
data_frame_datasource0 =
 glueContext.create_data_frame.from_options(connection_type="kinesis",
 connection_options=kinesis_options)
```

Kinesis への書き込み

例: Kinesis ストリームへの書き込み

the section called “forEachBatch” と組み合わせて使用します。DynamicFrame は JSON 形式でストリームに書き込まれます。何度か再試行してもジョブの書き込みが実行できないと、ジョブは失敗します。デフォルトでは、各 DynamicFrame レコードは Kinesis ストリームに個別に送信されます。この動作は、aggregationEnabled と関連するパラメータを使用して設定できます。

ストリーミングジョブから Amazon Kinesis への書き込みの例:

Python

```
glueContext.write_dynamic_frame.from_options(
    frame=frameToWrite
    connection_type="kinesis",
    connection_options={
        "partitionKey": "part1",
        "streamARN": "arn:aws:kinesis:us-east-1:111122223333:stream/streamName",
    }
)
```

Scala

```
glueContext.getSinkWithFormat(
```

```
                connectionType="kinesis",
                options=JsonOptions("""{
                    "streamARN": "arn:aws:kinesis:us-
east-1:111122223333:stream/streamName",
                    "partitionKey": "part1"
                }"""),
            )
            .writeDynamicFrame(frameToWrite)
```

Kinesis 接続パラメータ

Amazon Kinesis Data Streams への接続を指定します。

Kinesis ストリーミングデータソースには、次の接続オプションを使用します。

- "streamARN" (必須) 読み取り/書き込みに使用されます。Kinesis データストリームの ARN。
- "classification" (読み取りに必須) 読み取りで使用。レコード内のデータで使用されるファイル形式。データカタログを通じて提供されていない限り、必須です。
- "streamName" - (オプション) 読み取りで使用。読み取り対象/読み取り元の Kinesis データストリームの名前。endpointUrl で使用。
- "endpointUrl" - (オプション) 読み取りで使用。デフォルト: "https://kinesis.us-east-1.amazonaws.com"。Kinesis エンドポイントの AWS エンドポイント。特別なリージョンに接続する場合を除き、これを変更する必要はありません。
- "partitionKey" - (オプション) 書き込みに使用。レコードを作成する際に使用される Kinesis パーティションキー。
- "delimiter" (オプション) 読み取りに使用。classification が CSV の場合に使用される値の区切り文字。デフォルトは「,」です。
- "startingPosition": (オプション) 読み込みに使用。Kinesis データストリーム内の、データの読み取り開始位置。指定できる値は "latest"、"trim_horizon"、"earliest"、または UTC 形式のタイムスタンプ文字列であり、この文字列のパターンは yyyy-mm-ddTHH:MM:SSZ です (Z は UTC タイムゾーンのオフセットを +/- で表します。例:「2023-04-04T08:00:00-04:00」)。デフォルト値は "latest" です。注意: "startingPosition" の UTC 形式のタイムスタンプ文字列は AWS Glue バージョン 4.0 以降のみでサポートされます。
- "failOnDataLoss": (オプション) アクティブなシャードがないか、有効期限が切れている場合、ジョブは失敗します。デフォルト値は "false" です。

- "awsSTSRoleARN": (オプション) 読み取り/書き込みに使用。AWS Security Token Service (AWS STS) を使用して担うロールの Amazon リソースネーム (ARN)。このロールには、Kinesis データストリームのレコードの説明操作または読み取り操作の権限が必要です。このパラメーターは、別のアカウントのデータストリームにアクセスするときに使用する必要があります。"awsSTSSessionName" と組み合わせて使用します。
- "awsSTSSessionName": (オプション) 読み取り/書き込みに使用。AWS STS を使って、ロールを担うセッションの識別子。このパラメータは、別のアカウントのデータストリームにアクセスするときに使用する必要があります。"awsSTSRoleARN" と組み合わせて使用します。
- "awsSTSEndpoint": (オプション) 引き受けたロールで Kinesis に接続するときに使用する AWS STS エンドポイント。これにより、デフォルトのグローバルエンドポイントでは不可能な VPC 内のリージョナル AWS STS エンドポイントを使用できます。
- "maxFetchTimeInMs": (オプション) 読み込みに使用。ジョブエグゼキューターが Kinesis データストリームから現在のバッチのレコードを読み取るために費やした最大時間は、ミリ秒 (ms) 単位で指定されます。この時間内に複数の GetRecords API コールを行うことができます。デフォルト値は 1000 です。
- "maxFetchRecordsPerShard": (オプション) 読み込みに使用。1 マイクロバッチ当たりに Kinesis データストリームでシャードごとにフェッチするレコードの最大数。メモ: ストリーミングジョブが既に Kinesis (同じ get-records 呼び出しで) から余分なレコードを読み取っている場合、クライアントはこの制限を超えることができます。maxFetchRecordsPerShard が厳密である必要がある場合、maxRecordPerRead の倍数にする必要があります。デフォルト値は 100000 です。
- "maxRecordPerRead": (オプション) 読み込みに使用。getRecords オペレーションごとに、Kinesis データストリームからフェッチするレコードの最大数。デフォルト値は 10000 です。
- "addIdleTimeBetweenReads": (オプション) 読み込みに使用。2 つの連続する getRecords オペレーション間の遅延時間を追加します。デフォルト値は "False" です。このオプションは、Glue バージョン 2.0 以降でのみ設定可能です。
- "idleTimeBetweenReadsInMs": (オプション) 読み込みに使用。2 つの連続する getRecords オペレーション間での、最短の遅延時間 (ミリ秒単位で指定)。デフォルト値は 1000 です。このオプションは、Glue バージョン 2.0 以降でのみ設定可能です。
- "describeShardInterval": (オプション) 読み込みに使用。スクリプトが呼び出す 2 つの ListShards API 間での、再シャーディングを考慮すべき最小時間。詳細については、Amazon Kinesis Data Streams デベロッパーガイドの「リシャーディングのための戦略」を参照してください。デフォルト値は 1s です。

- "numRetries": (オプション) 読み込みに使用。Kinesis Data Streams API リクエストを再試行する最大の回数。デフォルト値は 3 です。
- "retryIntervalMs": (オプション) 読み込みに使用。Kinesis Data Streams API 呼び出しを再試行するまでのクールオフ期間 (ミリ秒単位で指定)。デフォルト値は 1000 です。
- "maxRetryIntervalMs": (オプション) 読み込みに使用。再試行で 2 つの Kinesis Data Streams API を呼び出す間の最大クールオフ期間 (ミリ秒単位で指定)。デフォルト値は 10000 です。
- "avoidEmptyBatches": (オプション) 読み込みに使用。バッチ処理を開始する前に、Kinesis データストリームで未読のデータをチェックすることで、空のマイクロバッチジョブを作成しないようにします。デフォルト値は "False" です。
- "schema": (inferSchema に false を設定した場合は必須) 読み取りに使用。ペイロードの処理に使用するスキーマ。分類が avro である場合、提供されるスキーマには Avro スキーマ形式を使用する必要があります。分類が avro 以外の場合、提供されるスキーマは DDL スキーマ形式である必要があります。

  以下に、スキーマの例を示します。

  Example in DDL schema format

  ```
  `column1` INT, `column2` STRING , `column3` FLOAT
  ```

  Example in Avro schema format

  ```
  {
    "type":"array",
    "items":
    {
      "type":"record",
      "name":"test",
      "fields":
      [
        {
          "name":"_id",
          "type":"string"
        },
        {
          "name":"index",
          "type":
          [
            "int",
            "string",
  ```

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

- "inferSchema": (オプション) 読み込みに使用。デフォルト値は、「false」です。「true」に設定すると、実行時に、スキーマが foreachbatch 内のペイロードから検出されます。
- "avroSchema": (非推奨) 読み取りに使用。Avro 形式を使用する場合に、Avro データのスキーマを指定するために使用されるパラメータです。このパラメータは非推奨となりました。schema パラメータを使用します。
- "addRecordTimestamp": (オプション) 読み込みに使用。このオプションが「true」に設定されている場合、データ出力には、対応するレコードがストリームによって受信された時刻を表示する「__src_timestamp」という名前が付けられた追加の列が含まれます。デフォルト値は、「false」です。このオプションは AWS Glue バージョン 4.0 以降でサポートされています。
- "emitConsumerLagMetrics": (オプション) 読み込みに使用。このオプションを「true」に設定すると、バッチごとに、ストリームが受信した最も古いレコードと、それが AWS Glue で CloudWatch に到着した時間との間隔のメトリクスが出力されます。メトリクスの名前は「glue.driver.streaming.maxConsumerLagInMs」です。デフォルト値は、「false」です。このオプションは AWS Glue バージョン 4.0 以降でサポートされています。
- "fanoutConsumerARN": (オプション) 読み込みに使用。streamARN で指定されたストリームの Kinesis ストリームコンシューマの ARN。Kinesis 接続の拡張ファンアウトモードを有効にするために使用されます。拡張ファンアウトが使用された Kinesis ストリームの使用に関する詳細については、「the section called “Kinesis ストリーミングジョブでの拡張ファンアウトの使用”」を参照してください。
- "recordMaxBufferedTime" - (オプション) 書き込みに使用。デフォルト: 1000 (ミリ秒)。レコードが書き込まれるのを待っている間にバッファリングされる最大時間。
- "aggregationEnabled" - (オプション) 書き込みに使用。デフォルト: true。Kinesis に送信する前にレコードを集約するかどうかを指定します。
- "aggregationMaxSize" - (オプション) 書き込みに使用。デフォルト: 51200 (バイト) レコードがこの制限よりも大きい場合、そのレコードはアグリゲータをバイパスします。注: Kinesis では、レコードサイズに 50 KB の制限が適用されます。これを 50 KB を超えて設定すると、サイズ超過のレコードは Kinesis によって拒否されます。
- "aggregationMaxCount" - (オプション) 書き込みに使用。デフォルト: 4294967295。集計されたレコードにパックされる項目の最大数。

- `"producerRateLimit"` - (オプション) 書き込みに使用。デフォルト: 150 (%)。1 つのプロデューサー (ジョブなど) からの送信されるシャード単位のスループットをバックエンド制限のパーセンテージとして制限できます。
- `"collectionMaxCount"` - (オプション) 書き込みに使用。デフォルト: 500。PutRecords レコードにパックされる項目の最大数。
- `"collectionMaxSize"` - (オプション) 書き込みに使用。デフォルト: 5242880 (バイト)。PutRecords リクエストで送信するデータの最大量。

AWS Glue Streaming 自動スケーリング

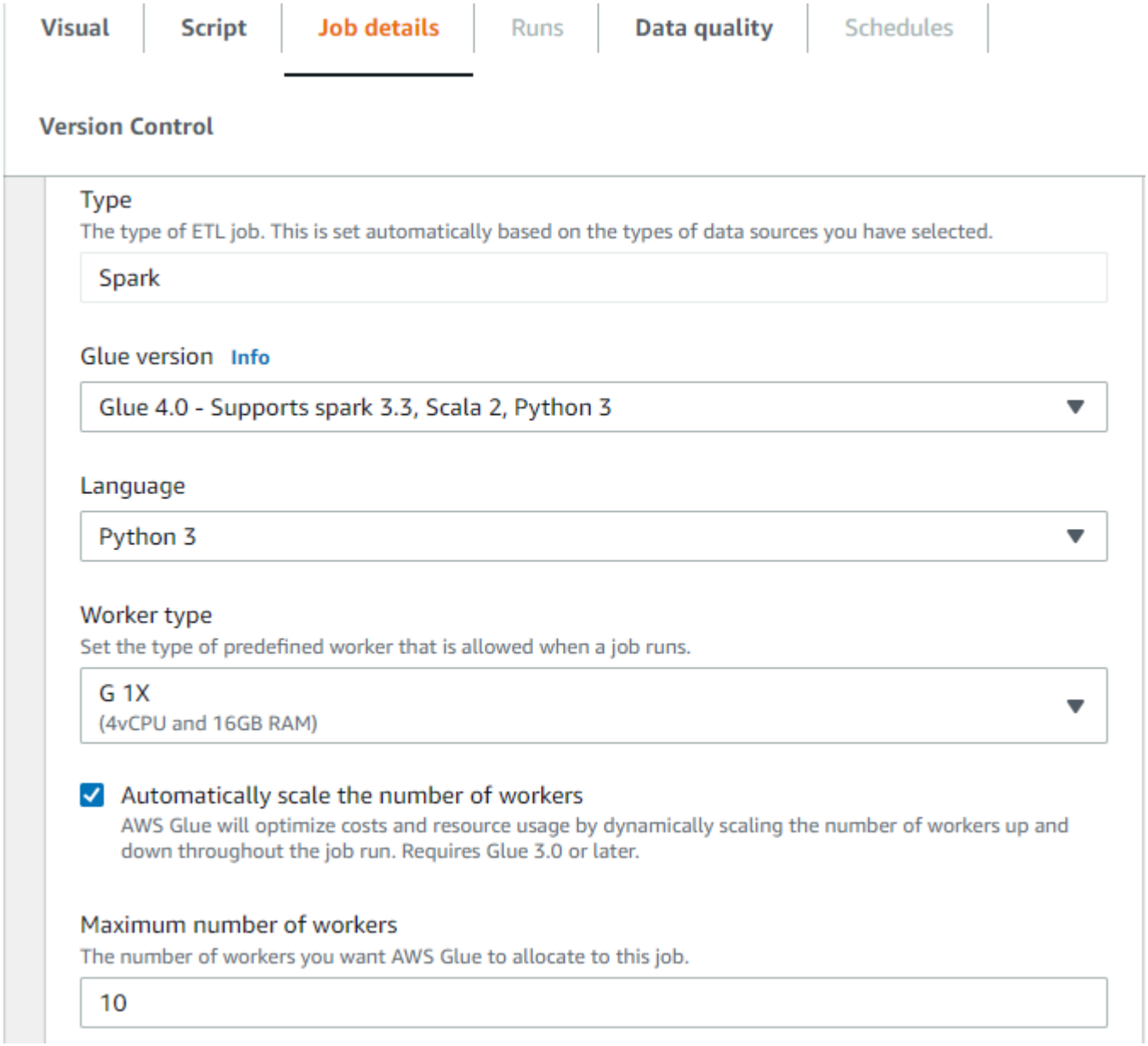
AWS Glue ストリーミング ETL ジョブは、ストリーミングソースからのデータを継続的に消費し、転送中のデータをクリーンアップおよび変換して、分析できるようにします。ジョブ実行の各ステージをモニタリングすることで、AWS Glue 自動スケーリングは、ワーカーがアイドル状態のときにワーカーをオフにしたり、追加の並列処理が可能な場合はワーカーを追加したりできます。

以下のセクションでは、AWS Glue Streaming 自動スケーリングについて説明します。

AWS Glue Studio の Auto-Scaling を有効にする

AWS Glue Studio の [Job details] (ジョブ詳細) タブで、タイプに [Spark] または [Spark Streaming] を選択し、[Glue version] (Glue バージョン) には [**`Glue 3.0`**] もしくは [**`Glue 4.0`**] を選択します。次に、[Worker type] (ワーカータイプ) の下にチェックボックスが表示されます。

- [Automatically scale the number of workers] (ワーカー数を自動的にスケーリングする) を選択します。
- ワーカーの最大数をセットして、ジョブ実行に投入できるワーカーの最大数を定義します。



AWS CLI または SDK を使用した Auto Scaling の有効化

Auto Scaling を有効にするには、ジョブ実行の AWS CLI から、次の設定で `start-job-run` を実行します。

```
{
    "JobName": "<your job name>",
    "Arguments": {
        "--enable-auto-scaling": "true"
    },
    "WorkerType": "G.2X", // G.1X and G.2X are allowed for Auto Scaling Jobs
```

```
    "NumberOfWorkers": 20, // represents Maximum number of workers
    ...other job run configurations...
}
```

ETL ジョブ実行が終了したら、get-job-run を呼び出して、実行されたジョブの実際のリソース使用量を DPU 秒単位で確認することもできます。注: 新しいフィールド [DPUSeconds] は、Auto Scaling が有効になっている AWS Glue 3.0 以降のパッチジョブに対してのみ表示されます。このフィールドは、ストリーミングジョブではサポートされません。

```
$ aws glue get-job-run --job-name your-job-name --run-id jr_xx --endpoint https://
glue.us-east-1.amazonaws.com --region us-east-1
{
    "JobRun": {
        ...
        "GlueVersion": "3.0",
        "DPUSeconds": 386.0
    }
}
```

同じ設定で AWS Glue SDK を使用して、Auto Scaling でジョブ実行を設定することもできます。

仕組み

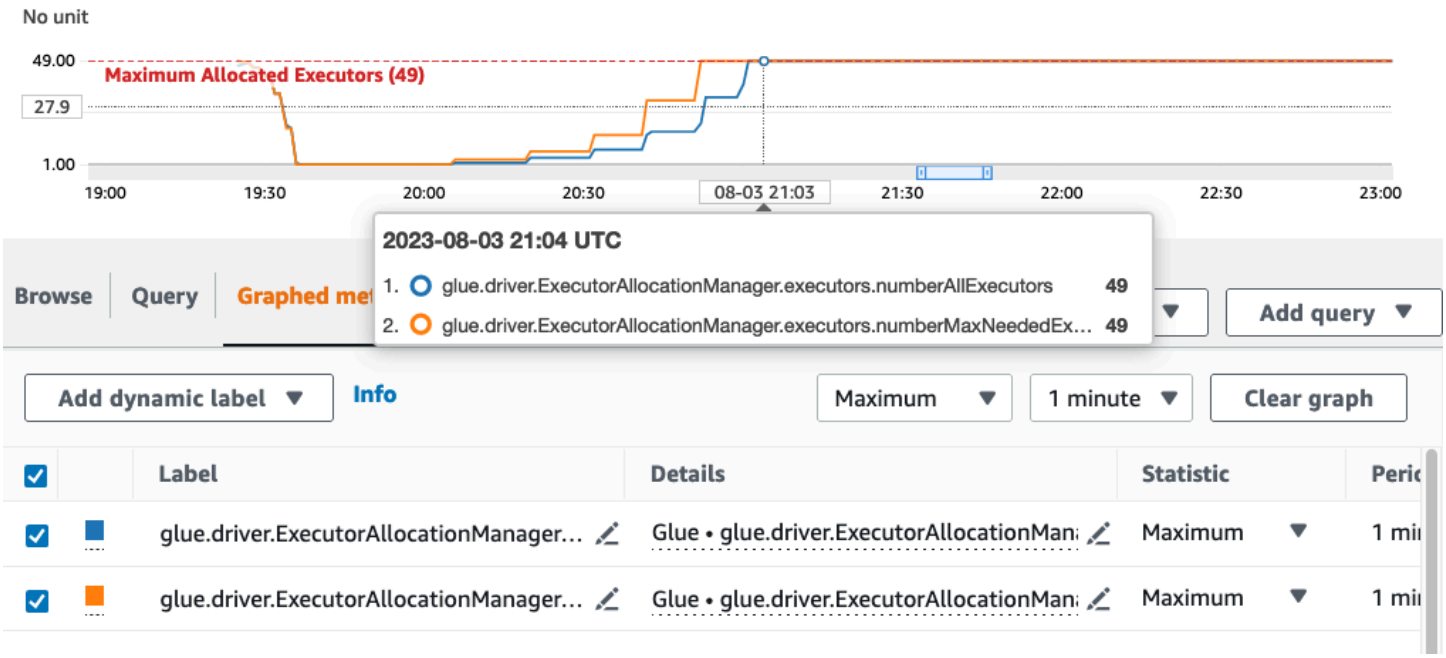
マイクロバッチ全体のスケーリング

次の例を使用して、自動スケーリングの仕組みを説明します。

- 50 DPU から始まる AWS Glue ジョブがあります。
- 自動スケーリングが有効になっています。

この例では、AWS Glue は、いくつかのマイクロバッチの「batchProcessingTimeInMs」メトリクスを確認し、設定されたウィンドウサイズ内でジョブが完了するかどうかを判断します。ジョブがすぐに完了する場合は、その完了するまでの時間によっては AWS Glue がスケールダウンすることもあります。「numberAllExecutors」でプロットされたこのメトリクスを Amazon CloudWatch でモニタリングして、自動スケーリングがどのように機能するかを確認できます。

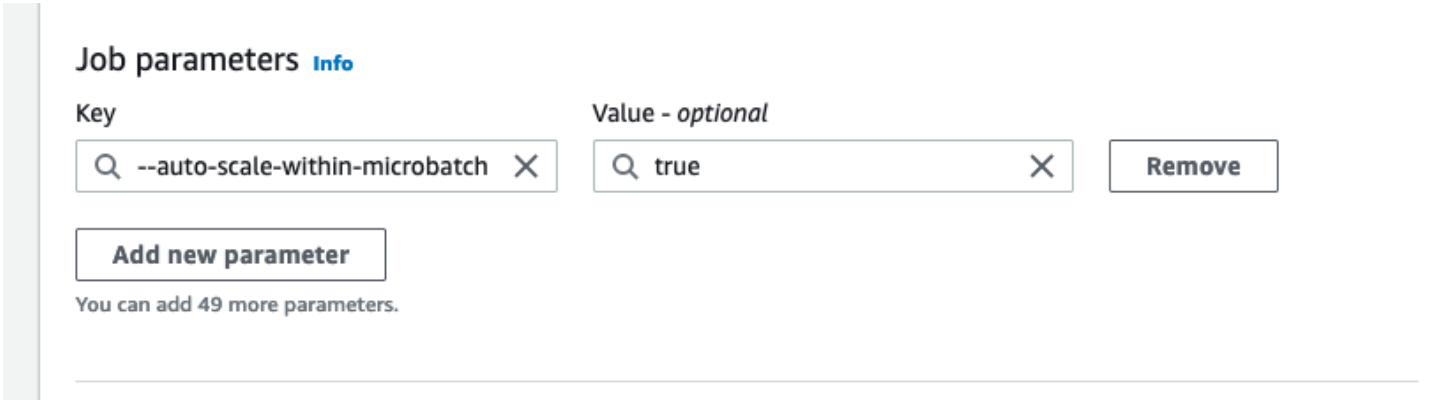
エグゼキューターの数は、マイクロバッチが完了するたびに指数関数的にスケールアップまたはスケールダウンします。Amazon CloudWatch モニタリングログからわかるように、AWS Glue は、必要なエグゼキューターの数 (オレンジ色の線) を確認し、それに合わせてエグゼキューター (青い線) を自動的にスケーリングしています。



AWS Glue がエグゼキューターの数をスケールダウンし、データ量が増加してマイクロバッチ処理時間が長くなることに気付くと、AWS Glue が指定された上限である 50 DPU にスケールアップします。

マイクロバッチ内でのスケーリング

上記の例では、システムが、完了したいくつかのマイクロバッチをモニタリングして、スケールアップするかスケールダウンするかを決定しています。ウィンドウが長くなると、いくつかのマイクロバッチを待つよりも、自動スケーリングがマイクロバッチ内で速く応答する必要になります。このような場合は、追加の設定 `--auto-scaIe-within-microbatch` を `true` に指定して使用できます。以下に示すように、これを AWS Glue Studio で AWS Glue ジョブプロパティに追加できます。



AWS Glue ストリーミングのメンテナンスウィンドウ

AWS Glue は定期的にメンテナンスアクティビティを実行します。AWS Glue は、これらのメンテナンスウィンドウ中にストリーミングジョブを再起動する必要があります。メンテナンスウィンドウを指定することで、ジョブを再起動するタイミングを制御できます。このセクションでは、メンテナンスウィンドウを設定する場所と、考慮すべき特定の動作について説明します。

トピック

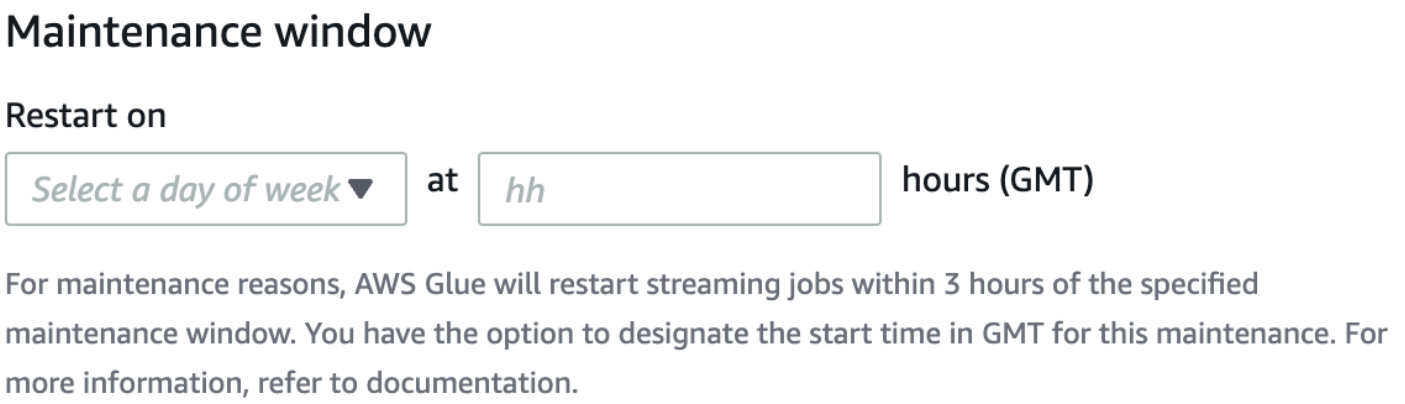
- メンテナンスウィンドウの設定
- メンテナンスウィンドウの動作
- ジョブのモニタリング
- データ損失処理

メンテナンスウィンドウの設定

AWS Glue Studio または API を使用してメンテナンスウィンドウを設定できます。

AWS Glue Studio でのメンテナンスウィンドウの設定

メンテナンスウィンドウは、AWS Glue ストリーミングジョブの [ジョブ詳細] ページで指定できます。GMT で日時を指定できます。AWS Glue は指定された時間枠内にジョブを再起動します。



API でのメンテナンスウィンドウの設定

または、Create Job API でメンテナンスウィンドウを設定することもできます。API を使用してメンテナンスウィンドウを設定する例を次に示します。

```
aws glue create-job –name jobName –role roleArnForTheJob –command
 Name=gluestreaming,ScriptLocation=s3-path-to-the-script --maintenance-window="Sun:10"
```

以下にコマンドの例を示します。

```
aws glue create-job –name testMaintenance –role arn:aws:iam::012345678901:role/
Glue_DefaultRole –command Name=gluestreaming,ScriptLocation=s3://glue-example-test/
example.py –maintenance-window="Sun:10
```

メンテナンスウィンドウの動作

AWS Glue は、ジョブを再起動するタイミングを決定するために一連のステップを実行します。

1. 新しいストリーミングジョブが開始されると、AWS Glue はまず、ジョブの実行に関連付けられたタイムアウトがあるかどうかを確認します。タイムアウトを使用すると、ジョブの終了時間を設定できます。タイムアウトが 7 日未満の場合、ジョブは再起動されません。
2. タイムアウトが 7 日を超える場合、AWS Glue はメンテナンスウィンドウがジョブに設定されているかどうかを確認します。その場合、そのウィンドウがピックアップされ、そのウィンドウがジョブ実行に割り当てられます。AWS Glue は、指定されたメンテナンスウィンドウから 3 時間以内にジョブを再起動します。たとえば、月曜日の午前 10 時 (GMT) にメンテナンスウィンドウを設定すると、ジョブは午前 10 時 (GMT) から午後 1 時 (GMT) までの間に再起動されます。
3. メンテナンスウィンドウが設定されていない場合、AWS Glue は再起動時間をジョブ実行開始時刻の 7 日前に自動的に設定します。例えば、ジョブを 7/1/2024 午前 12 時に開始し、メンテナンスウィンドウを指定しなかった場合、ジョブは 7/8/2024 午前 12 時に再起動するように設定されます。

   > **Note**
   >
   > ストリーミングジョブを既に実行している場合、この変更は 2024 年 7 月 1 日以降に影響します。メンテナンスウィンドウを設定する時間は 6 月 30 日までです。7 月 1 日以降、開始したストリーミングジョブは、このドキュメントに従って再起動されます。追加のサポートが必要な場合は、AWS サポートにお問い合わせください。

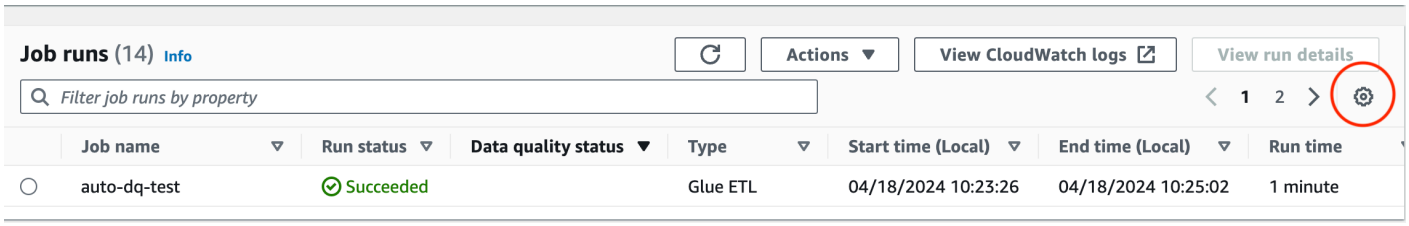
4. 特に進行中のマイクロバッチが処理されない場合、AWS Glue はジョブを再起動できないことがあります。このような場合、ジョブは中断されません。このような場合、AWS Glue は 14 日後にジョブを再起動します。この場合、メンテナンスウィンドウは受け入れられません。

ジョブのモニタリング

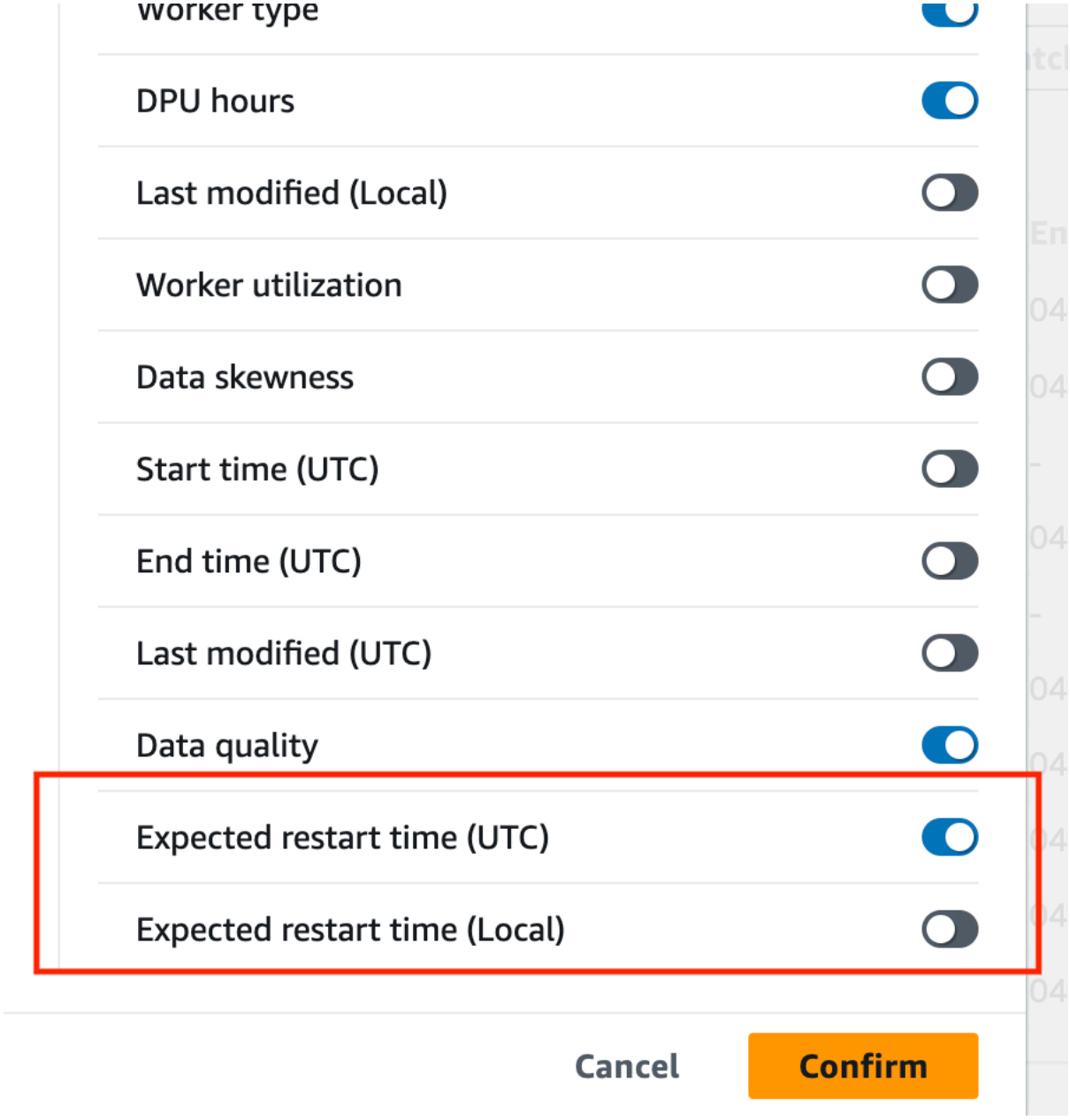
AWS Glue Studio の [モニタリング] ページでジョブをモニタリングできます。

ストリーミングジョブの次回の再起動予定時間を確認するには、[モニタリング] ページのジョブ実行テーブルに 列を表示します。

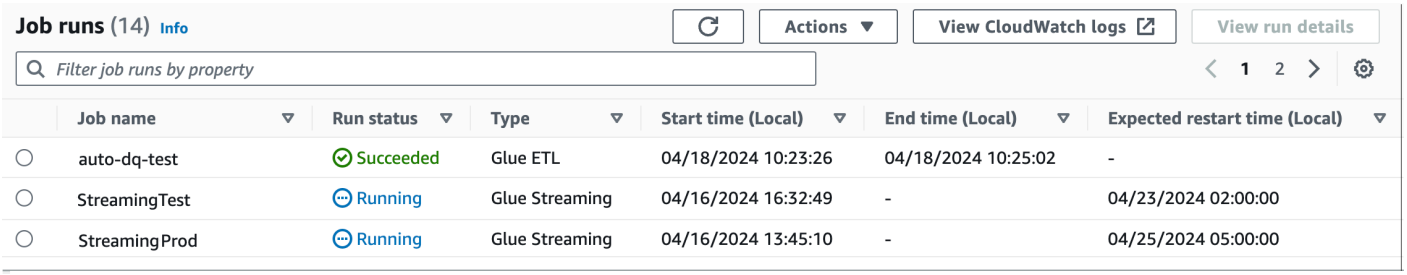
1. テーブルの右上にある歯車アイコンをクリックします。



2. 下にスクロールして、[予想される再起動時間] の列をオンにします。UTC とローカル時間の両方のオプションを使用できます。



3. その後、テーブル内の列を表示できます。



元のジョブには「EXPIRED」ステータスがあり、新しいジョブインスタンスには「RUNNING」ステータスがあります。再起動された新しいジョブ実行には、最初のジョブ実行 ID と再起動数を表すプレフィックス「restart_」の連結としてジョブ実行 ID が付けられます。例えば、最初のジョブ実行 ID が jr_1234 の場合、再起動されたジョブ実行には、最初の再起動の ID jr1234_restart_1 が含まれます。2 回目の再起動は 2 回目の再起動などの jr1234_restart_2 になります。

再起動により、再試行に影響はありません。自動再試行が原因で実行が失敗し、新しい実行が開始された場合、再起動のカウンターは 1 から再び開始されます。例えば、jr_1234_attempt_3_restart_5 で実行が失敗した場合、自動再試行は ID:jr_id1_attempt_4 で新しい実行を開始し、この試行が 7 日後に再開されると、新しい実行 ID は jr_id1_attempt_4_restart_1 になります。

データ損失処理

メンテナンスの再起動中、AWS Glue ストリーミングは、前のジョブ実行と再起動されたジョブ実行の間のデータの整合性と一貫性を確保するプロセスに従います。AWS Glue は、ジョブの再起動間のデータの整合性と一貫性を保証するものではありません。ストリーミングジョブ内で重複したデータを処理するには、アーキテクチャに関する考慮事項をお勧めします。

1. メンテナンスの再起動条件の検出: AWS Glue ストリーミングは、7 日後にメンテナンスウィンドウに達した場合や 14 日後にハード再起動が必要な場合など、メンテナンスの再起動をトリガーすべきタイミングを示す条件をモニタリングします。
2. 正常な終了の呼び出し: メンテナンスの再起動条件が満たされると、AWS Glue ストリーミングは現在実行中のジョブの正常な終了プロセスを開始します。このプロセスには、以下のステップが含まれます。
   a. 新しいデータの取り込みの停止: ストリーミングジョブは、入力ソース (Kafka トピック、Kinesis ストリーム、ファイルなど) からの新しいデータの消費を停止します。
   b. 保留中のデータの処理: ジョブは、内部バッファまたはキューに既に存在するデータを引き続き処理します。
   c. オフセットとチェックポイントのコミット: ジョブは、最新のオフセットまたはチェックポイントを外部システム (Kafka、Kinesis、Amazon S3 など) にコミットし、再起動されたジョブが前のジョブが中断した場所から取得できるようにします。
3. ジョブの再起動: 正常な終了プロセスが完了すると、AWS Glue ストリーミングは保持された状態とチェックポイントを使用してジョブを再起動します。再起動されたジョブは、最後にコミットされたオフセットまたはチェックポイントから処理をピックアップし、データが失われたり重複したりしないようにします。

4. データ処理の再開: 再起動されたジョブは、前のジョブが中断された時点からデータ処理を再開します。最後にコミットされたオフセットまたはチェックポイントから、入力ソースから新しいデータを取り込み続け、定義された ETL ロジックに従ってデータを処理します。

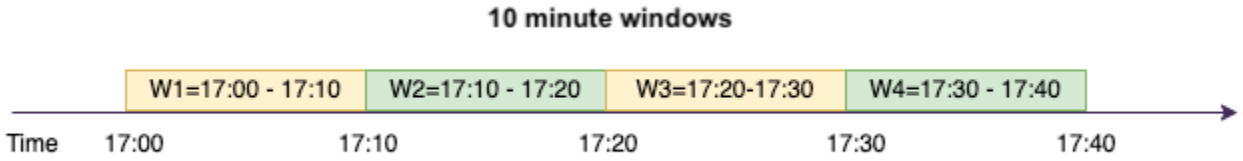
高度な AWS Glue のストリーミングの概念

現代のデータ駆動型のアプリケーションでは、データの重要性は時間の経過と共に低下し、予測データから事後的データへとその価値が移行します。そのため、お客様はデータをリアルタイムで処理して、より迅速な意思決定を行いたいと考えています。IoT センサーなどからのリアルタイムのデータフィードを処理する場合、取り込み中のネットワーク遅延やその他のソース関連の障害により、データの到着が順序付けされていないことや、処理に遅延が発生することがあります。AWS Glue プラットフォームの一部である AWS Glue Streaming は、これらの機能を基盤として、Apache Spark 構造化ストリーミングを利用したスケーラブルでサーバーレスなストリーミング ETL を提供し、ユーザーがリアルタイムでデータを処理できるようにします。

このトピックでは、高度なストリーミングの概念と AWS Glue Streaming の機能について説明します。

ストリームを処理する際の時間に関する考慮事項

ストリームを処理する際の時間には、4 つの概念があります。



- [イベント時間] – イベントが発生した時間。ほとんどの場合、このフィールドはソースでイベントデータ自体に埋め込まれています。
- [イベントタイムウィンドウ] – 2 つのイベント時間の間の時間枠。上の図に示されているように、W1 は 17:00 から 17:10 までのイベントタイムウィンドウです。各イベントタイムウィンドウは、複数のイベントをグループ化したものです。
- [トリガー時間] – トリガー時間により、データの処理と結果の更新が行われる頻度が制御されます。これは、マイクロバッチ処理が開始された時間です。
- [取り込み時間] – ストリームデータがストリーミングサービスに取り込まれた時間。イベント時間がイベント自体に組み込まれていない場合、この時間をウィンドウ処理に使用できる場合があります。

ウィンドウ処理

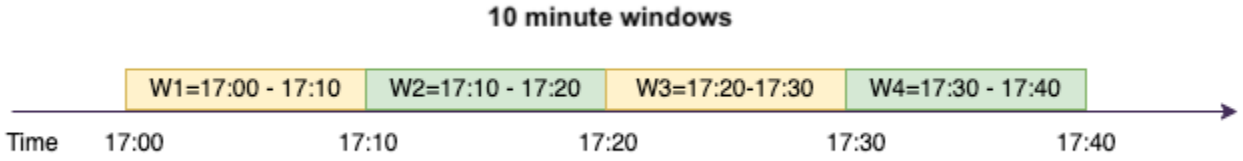
ウィンドウ処理は、複数のイベントをイベントタイムウィンドウごとにグループ化して集計する手法です。以下の例で、ウィンドウ処理の利点と、どのような場合に使用するかを説明します。

ビジネスユースケースに応じて、Spark では 3 種類のタイムウィンドウがサポートされます。

- タンブリングウィンドウ - 集計する、一連の重なり合わない固定サイズのイベントタイムウィンドウ。
- スライディングウィンドウ - 「固定サイズ」であるという点ではタンブリングウィンドウに似ていますが、スライドの継続時間がウィンドウ自体の継続時間よりも短い限り、ウィンドウが重なり合ったりスライドしたりできます。
- セッションウィンドウ - 入力データイベントで開始され、一定の間隔または非アクティブ時間内に入力が受信されている限り、長くなり続けます。セッションウィンドウは、入力に応じて、ウィンドウの長さが静的サイズまたは動的サイズになります。

タンブリングウィンドウ

タンブリングウィンドウは、集計する、一連の重なり合わない固定サイズのイベントタイムウィンドウです。実際の例を使って理解してみましょう。



ABC Auto 社は、スポーツカーの新しいブランドのマーケティングキャンペーンを行いたいと考えています。彼らは、スポーツカーのファンが最も多い都市を選びたいと考えています。この目標を達成するために、ウェブサイトで車を紹介する 15 秒間の短い広告を掲載しています。すべての「クリック」とそれに対応する「都市」が記録され、Amazon Kinesis Data Streams にストリーミングされます。10 分のウィンドウでクリック数をカウントし、都市別にグループ化して、需要が最も多い都市を確認したいと考えています。この集計の出力を次に示します。

window_start_time	window_end_time	city	total_clicks
2023-07-10 17:00:00	2023-07-10 17:10:00	ダラス	75

window_start_time	window_end_time	city	total_clicks
2023-07-10 17:00:00	2023-07-10 17:10:00	シカゴ	10
2023-07-10 17:20:00	2023-07-10 17:30:00	ダラス	20
2023-07-10 17:20:00	2023-07-10 17:30:00	シカゴ	50

上で説明したように、これらのイベントタイムウィンドウは、トリガー時間の間隔とは異なります。例えば、トリガー時間が 1 分ごとであっても、出力結果では 10 分の重なり合わない集計ウィンドウのみが表示されます。最適化するには、トリガー間隔をイベントタイムウィンドウに合わせることをお勧めします。

上の表で、17:00 ~ 17:10 のウィンドウでダラスでは 75 回のクリックがあり、シカゴでは 10 回のクリックがありました。また、どの都市についても 17:10 ~ 17:20 のウィンドウのデータがないため、このウィンドウは省略されています。

これで、ダウンストリーム分析アプリケーションでこのデータをさらに分析して、マーケティングキャンペーンを実施するのに最も適した都市を特定できます。

AWS Glue でのタンブリングウィンドウの使用

1. Amazon Kinesis Data Streams DataFrame を作成し、そこから読み取ります。例：

```
parsed_df = kinesis_raw_df \
                .selectExpr('CAST(data AS STRING)') \
                .select(from_json("data", ticker_schema).alias("data")) \
                .select('data.event_time','data.ticker','data.trade','data.volume',
 'data.price')
```

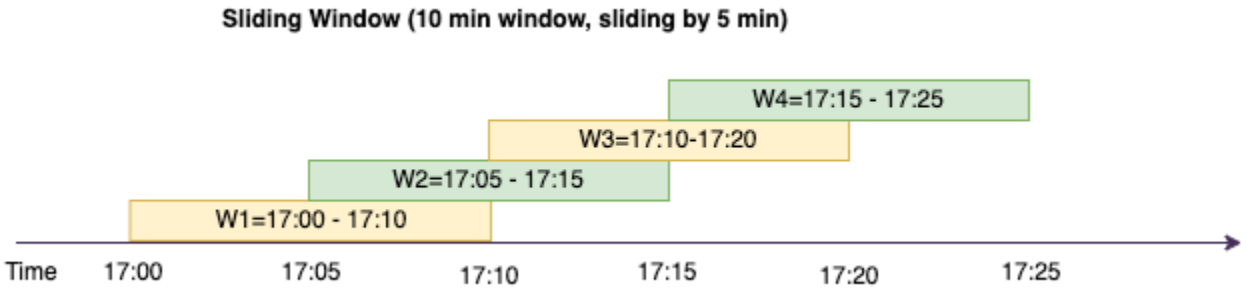
2. タンブリングウィンドウでデータを処理します。以下の例では、入力フィールド「event_time」に基づいて 10 分のタンブリングウィンドウでデータがグループ化され、その出力が Amazon S3 データレイクに書き込まれます。

```
grouped_df = parsed_df \
                .groupBy(window("event_time", "10 minutes"), "city") \
                .agg(sum("clicks").alias("total_clicks"))

summary_df = grouped_df \
```

```
                .withColumn("window_start_time", col("window.start")) \
                .withColumn("window_end_time", col("window.end")) \
                .withColumn("year", year("window_start_time")) \
                .withColumn("month", month("window_start_time")) \
                .withColumn("day", dayofmonth("window_start_time")) \
                .withColumn("hour", hour("window_start_time")) \
                .withColumn("minute", minute("window_start_time")) \
                .drop("window")

write_result = summary_df \
                .writeStream \
                .format("parquet") \
                .trigger(processingTime="10 seconds") \
                .option("checkpointLocation", "s3a://bucket-stock-stream/stock-
stream-catalog-job/checkpoint/") \
                .option("path", "s3a://bucket-stock-stream/stock-stream-catalog-
job/summary_output/") \
                .partitionBy("year", "month", "day") \
                .start()
```

スライディングウィンドウ

スライディングウィンドウは「固定サイズ」であるという点ではタンブリングウィンドウに似ていますが、スライドの継続時間がウィンドウ自体の継続時間よりも短い限り、ウィンドウが重なり合ったりスライドしたりできます。スライディングの性質上、1 つの入力を複数のウィンドウにバインドできます。



理解を深めるために、クレジットカードの不正使用の可能性を検知したい銀行の例を考えてみましょう。ストリーミングアプリケーションでは、クレジットカードの取引の流れを継続的にモニタリングできます。これらの取引が 10 分間のウィンドウに集計され、5 分ごとにウィンドウが先にスライドされて、最も古い 5 分間のデータが削除され、最新の 5 分間の新しいデータが追加されます。各

ウィンドウ内で、取引が国ごとにグループ化され、疑わしいパターンがないかチェックできます。例えば、米国での取引の直後にオーストラリアで発生した取引などです。わかりやすくするために、取引総額が 100 USD を超える場合は、その取引を不正使用として分類してみましょう。このようなパターンが検出された場合は、不正使用の可能性があることが示され、カードが凍結される可能性があります。

クレジットカード処理システムは、国と共に各カード ID の取引イベントのストリームを Kinesis に送信しています。AWS Glue ジョブが分析を実行し、以下の集計された出力を生成します。

window_start_time	window_end_time	card_last_four	country	total_amount
2023-07-10 17:00:00	2023-07-10 17:10:00	6544	米国	85
2023-07-10 17:00:00	2023-07-10 17:10:00	6544	オーストラリア	10
2023-07-10 17:05:45	2023-07-10 17:15:45	6544	米国	50
2023-07-10 17:10:45	2023-07-10 17:20:45	6544	米国	50
2023-07-10 17:10:45	2023-07-10 17:20:45	6544	オーストラリア	150

上記の集計に基づき、取引金額で合計された、5 分ごとにスライドする 10 分のウィンドウが表示されます。外れ値 (オーストラリアでの 150 USD の取引) がある 17:10 ~ 17:20 のウィンドウで異常が検出されます。AWS Glue は、この異常を検出し、boto3 を使用して問題のキーを含むアラームイベントを SNS トピックにプッシュすることができます。さらに、Lambda 関数はこのトピックにサブスクライブしてアクションを実行できます。

スライディングウィンドウでのデータの処理

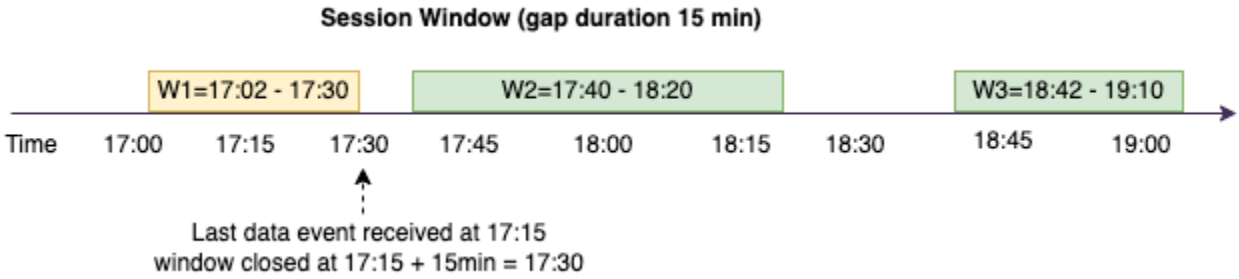
以下のように、group-by 句とウィンドウ関数を使用してスライディングウィンドウを実装します。

```
grouped_df = parsed_df \
```

```
            .groupBy(window(col("event_time"), "10 minute", "5 min"), "country",
"card_last_four") \
            .agg(sum("tx_amount").alias("total_amount"))
```

セッションウィンドウ

固定サイズである上記の 2 つのウィンドウとは異なり、セッションウィンドウは、入力に応じて、ウィンドウの長さが静的サイズまたは動的サイズになります。セッションウィンドウは入力データイベントで開始され、一定の間隔または非アクティブ時間内に入力が受信されている限り、長くなり続けます。



例を見てみましょう。ABC hotel 社は、1 週間のうちで最も混雑する時間帯を調べて、宿泊客により良い取引を提供したいと考えています。宿泊客がチェックインするとすぐにセッションウィンドウが開始され、Spark はそのイベントタイムウィンドウの間、集計する状態を維持します。宿泊客がチェックインするたびに、イベントが生成されて Amazon Kinesis Data Streams に送信されます。このホテルは、15 分間チェックインがない場合はイベントタイムウィンドウを終了できると判断しました。新しいチェックインがあると、次のイベントタイムウィンドウが再度開始されます。出力は次のとおりです。

window_start_time	window_end_time	city	total_checkins
2023-07-10 17:02:00	2023-07-10 17:30:00	ダラス	50
2023-07-10 17:02:00	2023-07-10 17:30:00	シカゴ	25
2023-07-10 17:40:00	2023-07-10 18:20:00	ダラス	75
2023-07-10 18:50:45	2023-07-10 19:15:45	ダラス	20

最初のチェックインは、event_time=17:02 に行われました。集計イベントタイムウィンドウは 17:02 に開始されます。この集計は、15 分以内にイベントが受信される限り継続されます。上の例では、17:15 に最後のイベントが受信され、その後の 15 分間はイベントがありませんでした。その結果、Spark はこのイベントタイムウィンドウを 17:15+15 分 = 17:30 に終了し、17:02 ~ 17:30 として設定しました。新しいチェックインデータイベントが受信された 17:47 に、新しいイベントタイムウィンドウが開始されました。

セッションウィンドウでのデータの処理

group-by 句とウィンドウ関数を使用してスライディングウィンドウを実装します。

```
grouped_df = parsed_df \
            .groupBy(session_window(col("event_time"), "10 minute"), "city") \
            .agg(count("check_in").alias("total_checkins"))
```

出力モード

出力モードとは、無制限テーブルからの結果が外部シンクに書き込まれるときのモードです。3 つのモードを使用できます。次の例では、各マイクロバッチでデータの行がストリーミングおよび処理されるときに、単語の出現回数をカウントしています。

- 完全モード – 現在のイベントタイムウィンドウで単語数が更新されていなくても、マイクロバッチ処理のたびに結果テーブル全体がシンクに書き込まれます。
- 付加モード – デフォルトのモードで、最後のトリガー以降に結果テーブルに追加された新しい単語や行のみがシンクに書き込まれます。このモードは、map、flatMap、filter などのクエリのステートレスストリーミングに適しています。
- 更新モード – 最後のトリガー以降に更新または追加された結果テーブルの単語や行のみがシンクに書き込まれます。

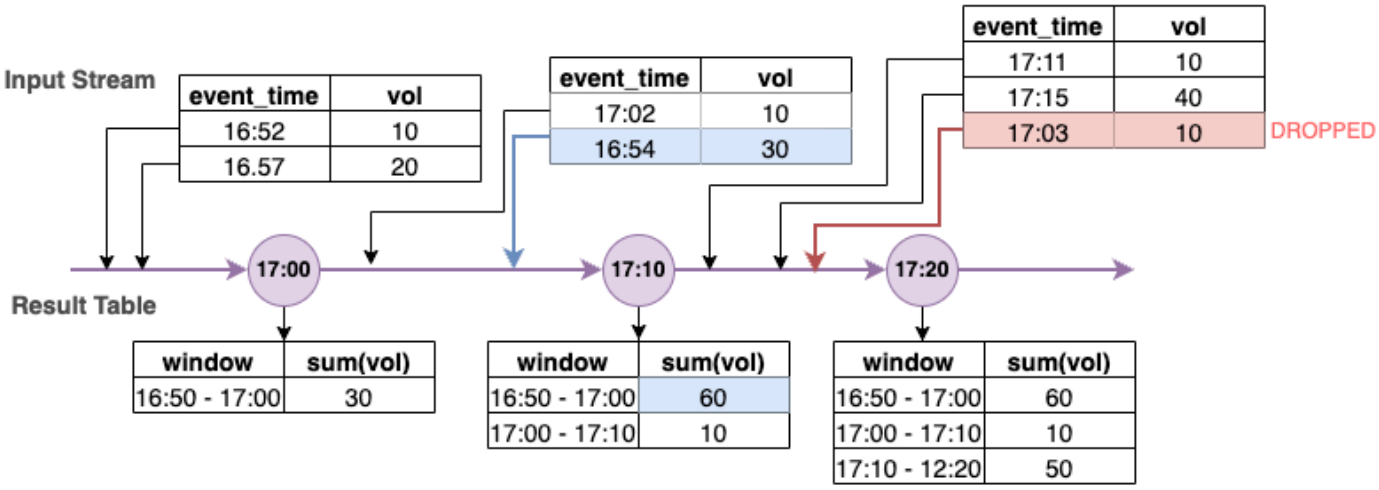
> **Note**
>
> セッションウィンドウでは「更新」の出力モードはサポートされていません。

遅延データとウォーターマークの処理

リアルタイムデータを扱う場合、ネットワーク遅延やアップストリームの障害によりデータの到着が遅れる可能性があるため、失われたイベントタイムウィンドウに対して再び集計を実行するメカニ

ズムが必要です。ただし、そのためにはステートを維持する必要があります。同時に、ステートのサイズを制限するために、古いデータをクリーンアップする必要があります。Spark バージョン 2.1 では、ステートを維持し、遅延データのしきい値をユーザーが指定することができる、ウォーターマークと呼ばれる機能のサポートが追加されました。

前述の株価ティッカーの例について、遅延データの許容しきい値が 10 分以内であると考えてみましょう。単純化のために、タンブリングウィンドウ、ティッカーは AMZ、取引を BUY と仮定します。



上の図では、10 分間のタンブリングウィンドウで合計株数を計算しています。17:00、17:10、および 17:20 にトリガーされています。タイムラインの矢印の上には入力データストリームがあり、下には無制限の結果のテーブルがあります。

最初の 10 分間のタンブリングウィンドウでは、event_time に基づいて集計され、total_volume は 30 と計算されました。2 番目のイベントタイムウィンドウで、Spark は event_time=17:02 の最初のデータイベントを取得しました。これは Spark でこれまでに確認された event_time の最大値であるため、ウォーターマークのしきい値がこの 10 分前に設定されます (つまり、watermark_event_time=16:52)。event_time が 16:52 以降のデータイベントは時間が制限された集計で考慮され、それより前のデータイベントはドロップされます。これにより、Spark はさらに 10 分間中間状態を維持して、遅延データを受け入れることができます。実時間で 17:08 ごろに Spark が event_time=16:54 のイベントを受信し、これはしきい値内でした。そこで、Spark は「16:50 ~ 17:00」のイベントタイムウィンドウを再計算し、合計株数が 30 から 60 に更新されました。

しかし、トリガー時間 17:20 で、Spark が event_time=17:15 のイベントを受信したときに watermark_event_time が 17:05 に設定されます。そのため、event_time=17:03 の遅延データイベントは「遅すぎる」と見なされ、無視されました。

```
Watermark Boundary = Max(Event Time) - Watermark Threshold
```

AWS Glue でのウォーターマークの使用

Spark は、ウォーターマークの境界が越えられるまで、外部シンクへのデータの出力または書き込みを行いません。AWS Glue でウォーターマークを実装するには、以下の例を参照してください。

```
grouped_df = parsed_df \
                .withWatermark("event_time", "10 minutes") \
                .groupBy(window("event_time", "5 minutes"), "ticker") \
                .agg(sum("volume").alias("total_volume"))
```

AWS Glue Streaming ジョブのモニタリングについて

ストリーミングジョブのモニタリングは、ETL パイプラインを構築するうえで極めて重要です。Spark UI を使用する以外に、Amazon CloudWatch を使用してメトリクスをモニタリングすることもできます。以下は、AWS Glue フレームワークによって出力されるストリーミングメトリクスのリストです。すべての AWS Glue メトリクスが含まれる完全なリストについては、「Amazon CloudWatch メトリクスを使用した AWS Glue のモニタリング」を参照してください。

AWS Glue では、構造化されたストリーミングフレームワークを使用して入力イベントを処理します。Spark API をコード内で直接使用することも、これらのメトリクスを発行する `GlueContext` で提供された `ForEachBatch` を利用することもできます。これらのメトリクスを理解するには、まず `windowSize` を理解する必要があります。

windowSize: `windowSize` は、指定するマイクロバッチの間隔です。ウィンドウサイズを 60 秒に指定すると、AWS Glue Streaming ジョブは 60 秒 (それまでに前のバッチが完了していない場合はそれ以上) 待ってから、ストリーミングソースからバッチのデータを読み取り、`ForEachBatch` で提供される変換を適用します。これはトリガー間隔とも呼ばれます。

メトリクスを詳しく見直して、ヘルスとパフォーマンスの特徴を理解しましょう。

> **Note**
>
> メトリクスは 30 秒ごとに出力されます。`windowSize` が 30 秒未満の場合、報告されるメトリクスは集計されたものとなります。例えば、`windowSize` が 10 秒で、マイクロバッチ

あたり 20 件のレコードを安定して処理しているとします。このシナリオでは、numRecords の出力メトリクス値は 60 になります。
メトリクスは、使用できるデータがない場合は出力されません。また、コンシューマーラグメトリクスの場合は、そのメトリクスを取得する機能を有効にする必要があります。

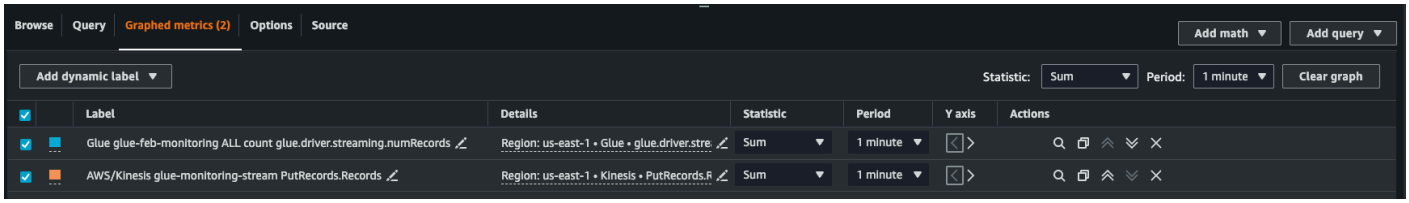
AWS Glue ストリーミングメトリクスの視覚化

ビジュアルメトリクスをプロットするには:

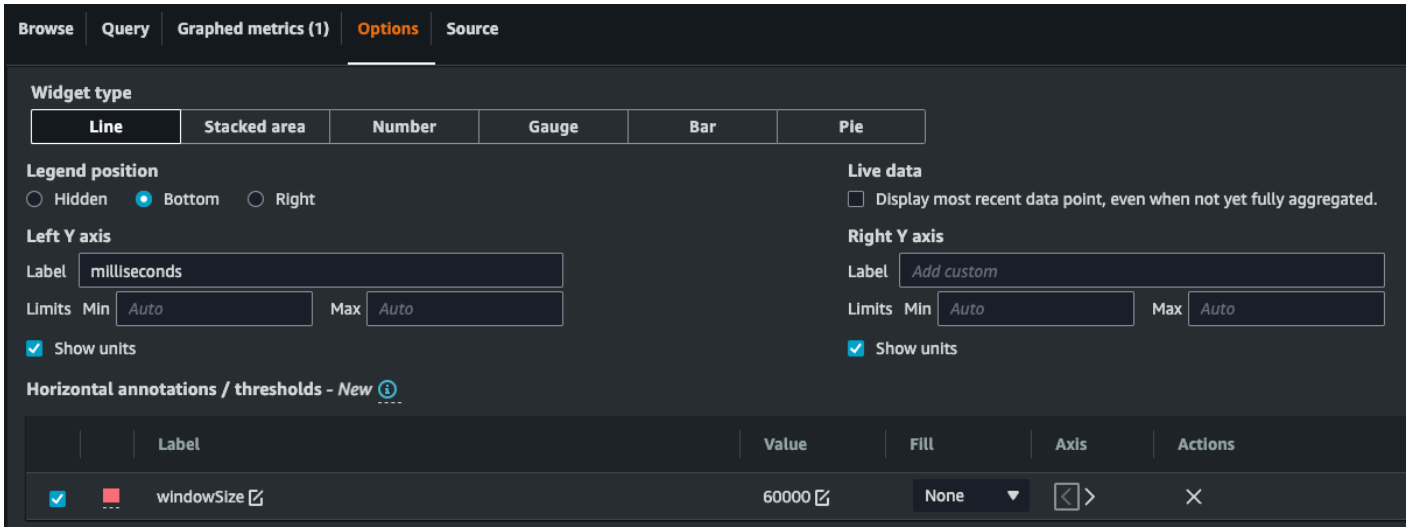
1. Amazon CloudWatch コンソールの [メトリクス] に移動し、[参照] タブを選択します。次に、「カスタム名前空間」の [Glue] を選択します。



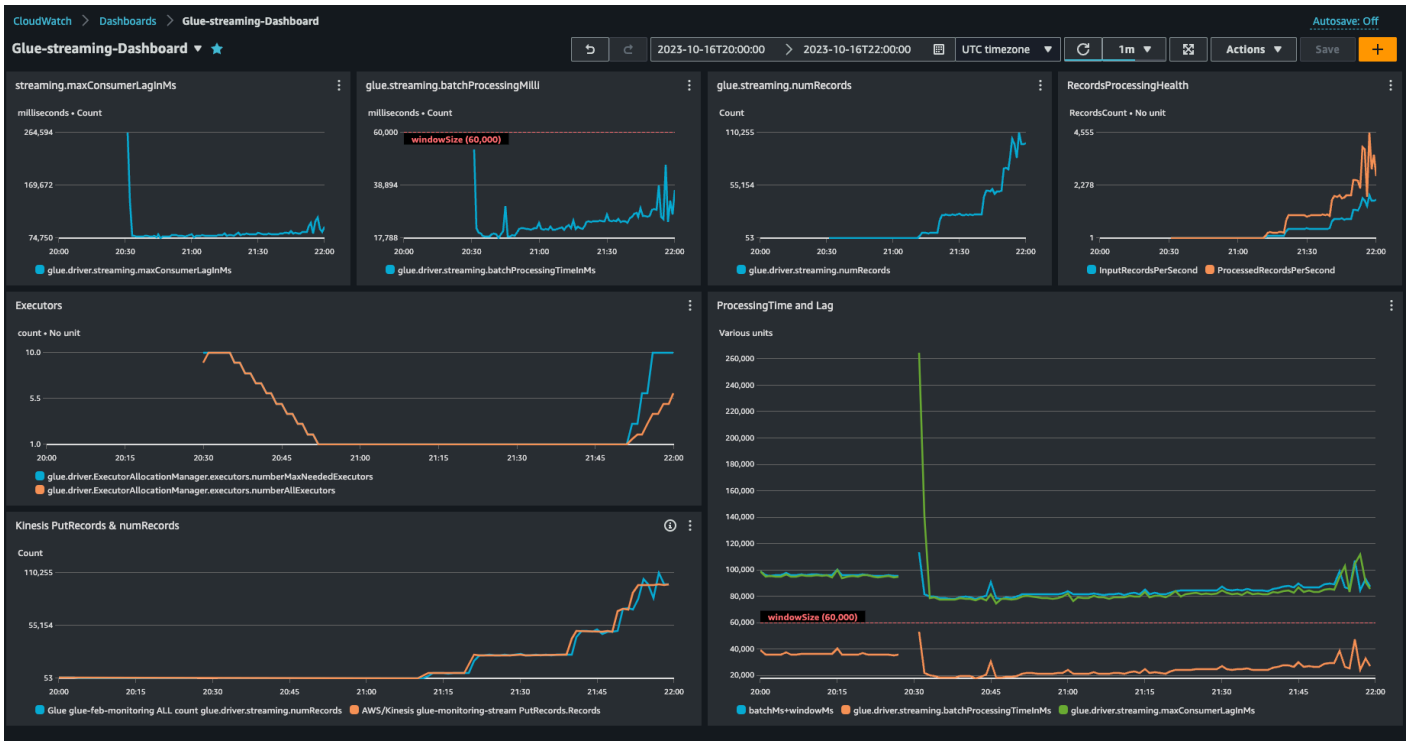
2. [ジョブメトリクス] を選択すると、すべてのジョブのメトリクスが表示されます。
3. JobName=glue-feb-monitoring に基づいてメトリクスをフィルタリングし、次に JobRunId=ALL に基づいてフィルタリングします。下図のように「+」記号をクリックすると、検索フィルターに追加できます。
4. 対象のメトリクスのチェックボックスを選択します。以下の図では、`numberAllExecutors` と `numberMaxNeededExecutors` を選択しています。



5. これらのメトリクスを選択したら、[グラフ化したメトリクス] タブに移動して統計を適用できます。
6. メトリクスは 1 分ごとに出力されるため、`batchProcessingTimeInMs` および `maxConsumerLagInMs` には 1 分より長い「average」を適用できます。`numRecords` には、1 分ごとの「sum」を適用できます。
7. [オプション] タブでは、水平方向の `windowSize` の注釈をグラフに追加できます。



8. メトリクスを選択したら、ダッシュボードを作成して追加します。ダッシュボードの例を以下に示します。

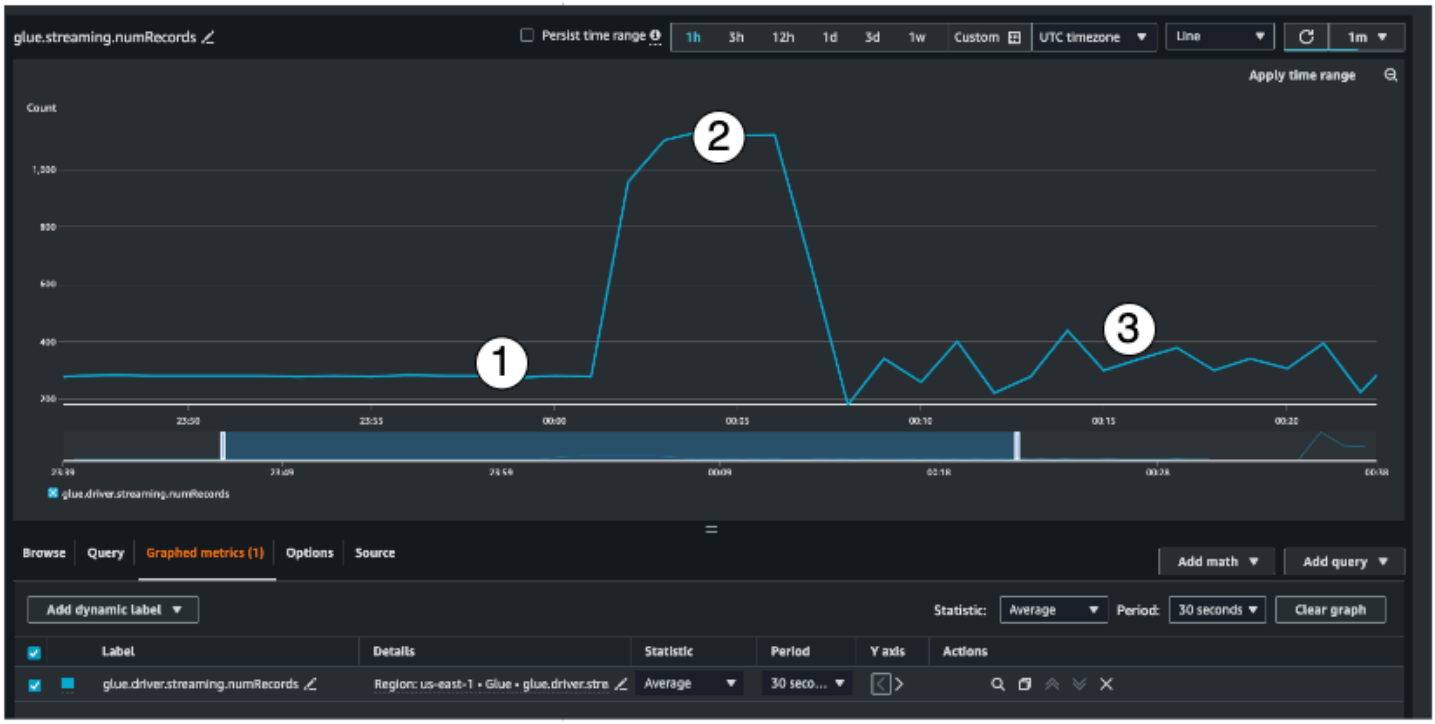


AWS Glue ストリーミングメトリックの使用

このセクションでは、各メトリクスと、それらが相互にどう関連しているかについて説明します。

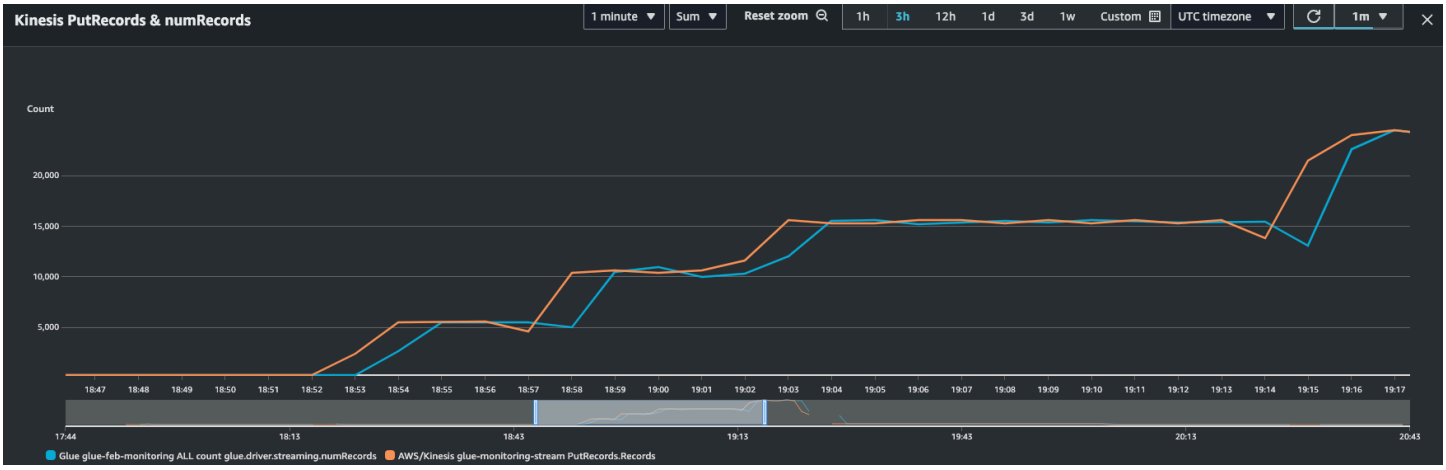
レコード数 (メトリクス: streaming.numRecords)

このメトリクスは、処理中のレコードの数を示します。



このストリーミングメトリクスにより、処理中のレコードの数をウィンドウ内で可視化できます。処理中のレコードの数だけでなく、入力トラフィックの動作を把握するのにも役立ちます。

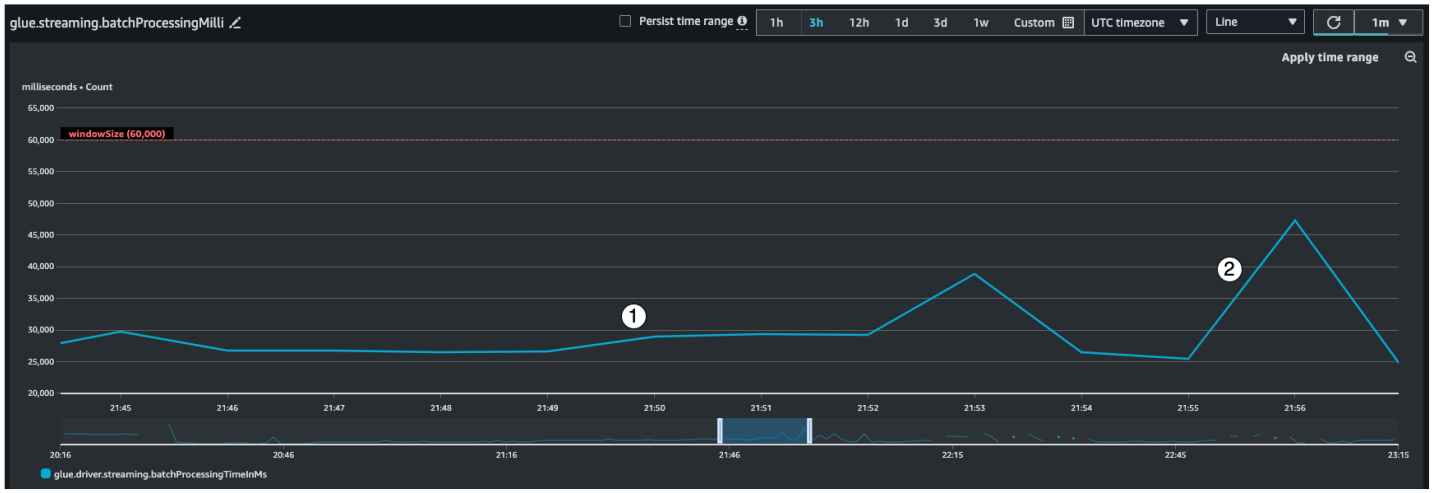
- インジケータ #1 は、トラフィックが安定しており、バーストが発生していない例を示しています。一般に、これは定期的にデータを収集してストリーミングソースに送信する IoT センサーのようなアプリケーションです。
- インジケータ #2 は、他の負荷が安定しているのに、トラフィックで突然バーストが発生した例を示しています。これは、ブラックフライデーのようなマーケティングイベントがあり、クリック数が急増した場合に、クリックストリームアプリケーションで発生することがあります。
- インジケータ #3 は、予測不可能なトラフィックの例を示しています。予想できないトラフィックは問題があることを指しません。これは入力データの性質にすぎません。IoT センサーの例に話を戻すと、何百ものセンサーが気象変化イベントをストリーミングソースに送信していると考えることができます。気象の変化は予測できないため、データも予測できません。トラフィックパターンを理解することは、エグゼキューターのサイズを決定するうえで重要です。入力が非常に多い場合は、自動スケーリングの使用を検討してください (これについては後で詳しく説明します)。



このメトリクスを Kinesis PutRecords メトリクスと組み合わせて、取り込まれるイベントの数と読み取られるレコードの数がほぼ同じになるようにすることができます。これは特に、ラグについて把握したい場合に便利です。取り込み率が高くなると、AWS Glue によって読み取られる numRecords も増加します。

バッチ処理時間 (メトリクス: streaming.batchProcessingTimeInMs)

バッチ処理時間メトリクスは、クラスターがプロビジョニング不足かプロビジョニング過剰かを判断するのに役立ちます。

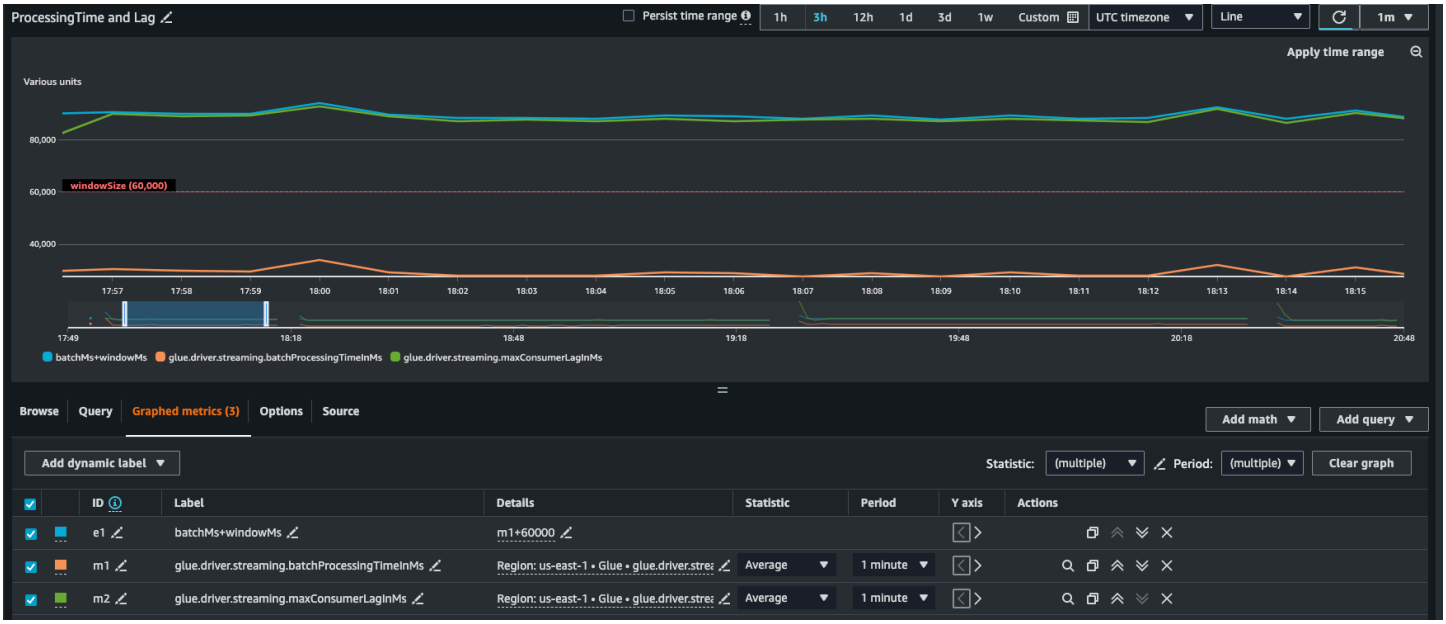


このメトリクスは、レコードの各マイクロバッチの処理にかかったミリ秒数を示します。ここでの主な目的は、この時間をモニタリングして windowSize の間隔を下回るようにすることです。次のウィンドウ間隔で回復するのであれば、batchProcessingTimeInMs が一時的にオーバーしても問題ありません。インジケータ #1 は、ジョブの処理にかかった時間がほぼ安定していることを示しています。ただし、入力レコードの数が増えている場合は、インジケータ #2 で示されているように、ジョブの処理にかかる時間も長くなります。numRecords が増えていない

のに処理時間が長くなっている場合は、エグゼキューターのジョブ処理を詳しく調べる必要があります。batchProcessingTimeInMs が 10 分より長く 120% を超えることがないように、しきい値とアラームを設定することをお勧めします。アラームの設定の詳細については、「Amazon CloudWatch でのアラームの使用」を参照してください。

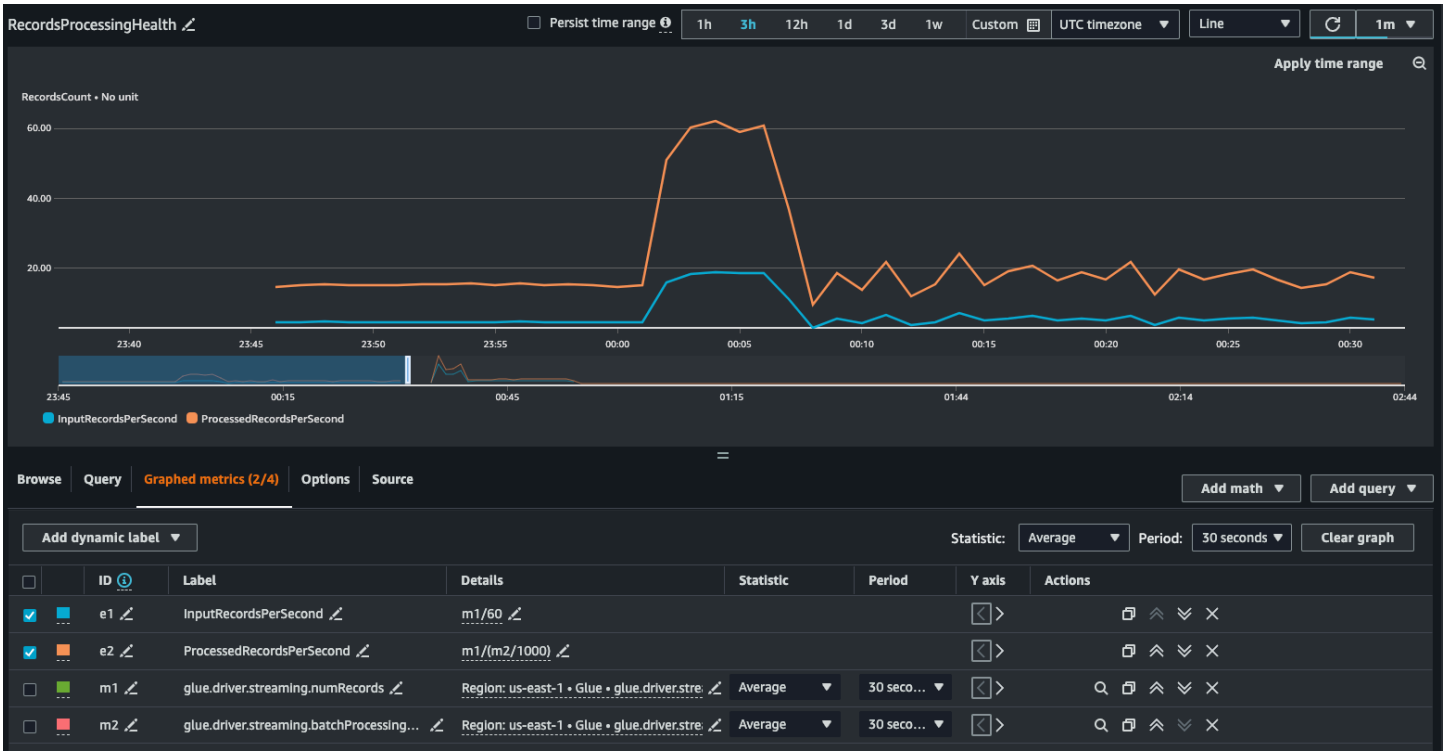
コンシューマーラグ (メトリクス: streaming.maxConsumerLagInMs)

コンシューマーラグメトリクスは、イベントの処理にラグがあるかどうかを把握するのに役立ちます。適切な windowSize を設定していても、ラグが大きすぎると、使用している処理 SLA が満たされない可能性があります。このメトリクスは、emitConsumerLagMetrics 接続オプションを使用して明示的に有効にする必要があります。詳細については、「KinesisStreamingSourceOptions」を参照してください。



派生メトリクス

より深い洞察を得るために、派生メトリクスを作成して、Amazon CloudWatch のストリーミングジョブについて理解を深めることができます。



派生メトリクスが含まれるグラフを作成して、DPU をさらに使用する必要があるかどうかを判断できます。自動スケーリングはこれを自動的に行うのに役立ちますが、派生メトリクスを使用することで、自動スケーリングが効果的に機能しているかどうかを判断できます。

- InputRecordsPerSecond は、入力レコードを取得する速度を示します。これは次のように算出されます: 入力レコードの数 (glue.driver.streaming.numRecords)/ WindowSize。
- ProcessingRecordsPerSecond は、レコードを処理する速度を示します。これは次のように算出されます: 入力レコードの数 (glue.driver.streaming.numRecords)/ batchProcessingTimeInMs。

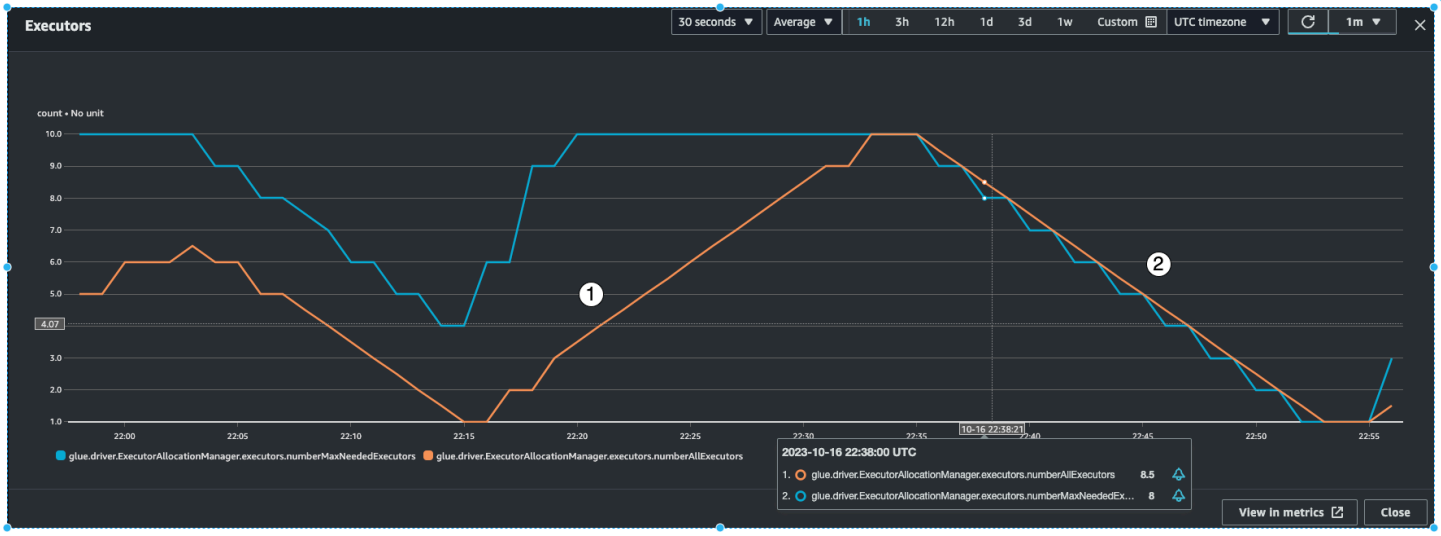
入力速度が処理速度より高いときは、ジョブを処理するための容量を増やすか、並列処理を増やす必要がある場合があります。

自動スケーリングメトリクス

入力トラフィックが急増しているときは、自動スケーリングを有効にすることを検討し、最大ワーカー数を指定する必要があります。そのために、`numberAllExecutors` と `numberMaxNeededExecutors` という 2 つのメトリクスが用意されています。

- numberAllExecutors は、アクティブに実行されているジョブエグゼキューターの数です。
- numberMaxNeededExecutors は、現在の負荷を満たすために必要な (アクティブに実行中および保留中の) ジョブエグゼキューターの最大数です。

この 2 つのメトリクスは、自動スケーリングが正しく機能しているかどうかを判断するのに役立ちます。



AWS Glue は、数マイクロバッチにわたって `batchProcessingTimeInMs` メトリクスをモニタリングし、次の 2 つのいずれかを実行します。`batchProcessingTimeInMs` が `windowSize` に近い場合はエグゼキューターをスケールアウトし、`batchProcessingTimeInMs` が `windowSize` より比較的低い場合はエグゼキューターをスケールインします。また、エグゼキューターをステップスケールするアルゴリズムも使用します。

- インジケータ #1 は、負荷を処理するために、アクティブなエグゼキューターがどのようにスケールアップし、必要最大数のエグゼキューターに追いついたかを示しています。
- インジケータ #2 は、`batchProcessingTimeInMs` が低くなってから、アクティブなエグゼキューターがどのようにスケールインしたかを示しています。

これらのメトリクスを使用して、現在のエグゼキューターレベルの並列処理をモニタリングし、それに応じて自動スケーリング構成の最大ワーカー数を調整できます。

最高のパフォーマンスを得る方法

Spark は、Amazon Kinesis ストリーム内の読み取りのために、シャードごとに 1 つのタスクを作成しようとします。各シャードのデータはパーティションになります。次に、各ワーカーのコア数に応じて、これらのタスクをエグゼキューター/ワーカーに分散します (ワーカーあたりのコア数は、選択したワーカータイプ (`G.025X` や `G.1X` など) によって異なります)。ただし、タスクがどのように分散されるかは非決定論的です。すべてのタスクは、それぞれのコアで並列実行されます。使用可能なエグゼキューターコアの数よりも多くのシャードがある場合、タスクはキューに入れられます。

上記のメトリクスとシャード数を組み合わせて、バーストのための余裕がある安定した負荷が実現するようにエグゼキュターをプロビジョニングできます。おおよそのワーカー数を決定するために、ジョブを数回繰り返すことをお勧めします。不安定で急激なワークロードの場合も、自動スケーリングと最大ワーカー数を設定することで同じことを実行できます。

windowSize は、ビジネスの SLA 要件に従って設定してください。例えば、処理されたデータが 120 秒を超えてはならないことをビジネスで義務付けている場合は、平均コンシューマーラグが 120 秒未満になるように windowSize を少なくとも 60 秒に設定します (前述のコンシューマーラグに関するセクションを参照)。そこから、numRecords とシャード数に応じて、DPU の容量を計画し、batchProcessingTimeInMs がほとんどいつも windowSize の 70% 未満になるようにします。

> **Note**
>
> ホットシャードはデータスキューの原因となる可能性があります。つまり、一部のシャード/パーティションが他のシャード/パーティションよりもはるかに大きくなる可能性があります。これにより、並列実行されている一部のタスクに時間がかかり、ストラグラータスクが発生する場合があります。その結果、前のバッチのすべてのタスクが完了するまで次のバッチを開始できず、これが batchProcessingTimeInMillis と最大ラグに影響することになります。

ゼロ ETL 統合

ゼロ ETL とは、AWS によるフルマネージド型の統合のセットであり、一般的な取り込みとレプリケーションのユースケースで ETL データパイプラインを構築する必要性を最小限に抑えます。Amazon SageMaker Lakehouse と Amazon Redshift で、複数の運用、トランザクション、アプリケーションソースからデータを利用できるようにします。ゼロ ETL 統合では、分析、AI/ML、レポート作成用の新しいデータを取得できます。ビジネスダッシュボード、最適化されたゲームエクスペリエンス、データ品質のモニタリング、顧客行動分析などのユースケース向けに、より正確でタイムリーなインサイトが得られます。ビジネス全体で、データ駆動型の予測をより自信を持って実行し、カスタマーエクスペリエンスを向上させ、データ駆動型のインサイトを促進できます。

Amazon Redshift は、高速でフルマネージドのペタバイト規模のデータウェアハウスサービスで、既存のビジネスインテリジェンスツールを使用してすべてのデータを簡単かつコスト効率よく効率的に分析できます。

Amazon SageMaker Lakehouse は、Amazon Simple Storage Service (S3) データレイクと Amazon Redshift データウェアハウス全体ですべてのデータを統合し、データの 1 つのコピーで強力な分析と AI/ML アプリケーションを容易に構築できるようにします。SageMaker Lakehouse では、Apache Iceberg と互換性のあるすべてのツールやエンジンで、インプレースのデータに柔軟にアクセスしてクエリを実行できます。SageMaker Lakehouse では、Apache Iceberg 互換ツールやエンジンを使用して、インプレースのデータにアクセスしてクエリを実行する柔軟性もあります。さらに、すべての分析ツールとエンジンのすべてのデータに適用される、統合されたきめ細かなアクセスコントロールによってデータを保護することができます。アクセス許可を一度定義すれば、組織全体で確実にデータを共有できます。

AWS Glue のゼロ ETL 機能

AWS Glue のゼロ ETL 統合により、AWS データサービスやサードパーティーアプリケーションから AWS 送信先へのデータインジェストとレプリケーションが簡素化されます。

AWS のゼロ ETL ソースでサポートされる AWS Glue サービスには、次のものがあります。

- Amazon DynamoDB

ゼロ ETL でサポートされているサードパーティーアプリケーションには、次のようなものがあります。

- Facebook 広告
- Instagram 広告
- Salesforce
- Salesforce Marketing Cloud Account Engagement
- SAP OData
- ServiceNow
- Zendesk
- Zoho CRM

AWS のゼロ ETL ターゲットでサポートされる AWS Glue サービスには、次のものがあります。

- Amazon Redshift
- Amazon SageMaker Lakehouse

> **Note**
>
> AWS Glue で Amazon DynamoDB ソースとのゼロ ETL 統合を作成する場合、ターゲットは Amazon SageMaker Lakehouse でサポートされます。

ゼロ ETL 統合を設定するための前提条件

ソースとターゲットの統合を設定するには、AWS Glue がソースからデータにアクセスしてターゲットに書き込むために使用する IAM ロールの設定や、中間地点またはターゲットの場所にあるデータを暗号化するための KMS キーの使用など、いくつかの前提条件が必要です。

トピック

- ソースリソースの設定
- ターゲットリソースの設定
- Amazon Redshift データウェアハウスの作成
- ゼロ ETL 統合用の VPC のセットアップ
- ゼロ ETL クロスアカウント統合の設定

ソースリソースの設定

ソースの必要に応じて、次のセットアップタスクを実行します。

ソースロールのセットアップ

このセクションでは、ゼロ ETL 統合が接続にアクセスできるようにソースロールを渡す方法について説明します。これは SaaS ソースにのみ適用されます。

> **Note**
>
> アクセスを少数の接続のみに制限するには、まず接続を作成して接続 ARN を取得します。「ゼロ ETL 統合のソースの設定」を参照してください。

統合が接続にアクセスするためのアクセス許可を持つロールを作成します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "GlueConnections",
            "Effect": "Allow",
            "Action": [
                "glue:GetConnections",
                "glue:GetConnection"
            ],
            "Resource": [
                "arn:aws:glue:*:<accountId>:catalog",
                "arn:aws:glue:us-east-1:<accountId>:connection/*"
            ]
        },
        {
            "Sid": "GlueActionBasedPermissions",
            "Effect": "Allow",
            "Action": [
                // Fetch entities:
                "glue:ListEntities",
                // Refresh connection credentials:
                "glue:RefreshOAuth2Tokens"
            ],
            "Resource": [
```

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

信頼ポリシー:

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

ターゲットリソースの設定

必要に応じて、AWS Glue Data Catalog または Amazon RedshiftA データウェアハウス統合ターゲットに以下のセットアップタスクを実行します。

AWS Glue データベースターゲットとの統合の場合:

- AWS Glue データベースのセットアップ
- カタログリソースベースアクセス (RBAC) ポリシーの指定
- ターゲット IAM ロールの作成

Amazon Redshift データウェアハウスターゲットとの統合の場合:

- https://docs.aws.amazon.com/glue/latest/dg/zero-etl-prerequisites.html#zero-etl-setup-target-redshift-data-warehouse

AWS Glue データベースのセットアップ

AWS Glue データベースを使用する統合の場合:

Amazon S3 の場所を使用して AWS Glue Data Catalog にターゲットデータベースを設定するには:

1. AWS Glue コンソールのホームページで、[データカタログ] から [データベース] を選択します。
2. 右上にある [データベースを追加] を選択します。既にデータベースを作成している場合は、Amazon S3 URI がある場所がデータベースに設定されていることを確認してください。
3. 名前と場所 (Amazon S3 URI) を入力します。ゼロ ETL 統合には場所が必要であることに注意してください。完了したら、[データベースの作成] をクリックします。

> **Note**
>
> Amazon S3 バケットは、AWS Glue データベースと同じリージョンに存在する必要があります。

AWS Glue で新しいデータベースを作成する方法については、「Getting started with the AWS Glue Data Catalog」を参照してください。

`create-database` CLI を使用して AWS Glue にデータベースを作成することもできます。`--database-input` の `LocationUri` は必須であることにご注意ください。

Iceberg テーブルの最適化

ターゲットデータベースで AWS Glue によってテーブルが作成されると、圧縮を有効にして Amazon Athena でのクエリを高速化できます。圧縮用のリソース (IAM ロール) の設定については、「テーブル最適化の前提条件」を参照してください。

統合によって作成された AWS Glue テーブルで圧縮を設定する方法の詳細については、「Iceberg テーブルの最適化」を参照してください。

カタログリソースベースアクセス (RBAC) ポリシーの指定

AWS Glue データベースを使用する統合の場合、ソースとターゲット間の統合を許可するには、カタログ RBAC ポリシーに次のアクセス許可を追加します。

> **Note**
>
> クロスアカウント統合の場合、Alice (統合を作成するユーザー) ロールポリシーとカタログリソースポリシーの両方がリソースで glue:CreateInboundIntegration を許可する必要があります。同じアカウントの場合は、リソースポリシーまたはリソースで glue:CreateInboundIntegration を許可するロールポリシーのいずれかで十分です。どちらのシナリオでも、glue.amazonaws.com を glue:AuthorizeInboundIntegration に許可する必要があります。

[データカタログ] から [カタログの設定] にアクセスできます。次に、次のアクセス許可を指定し、不足している情報を入力します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    { // Allow Alice to create Integration on Target Database
      "Principal": {
        "AWS": [
          "arn:aws:iam::<source-account-id>:user/Alice"
        ]
      },
      "Effect": "Allow",
      "Action": [
        "glue:CreateInboundIntegration"
      ],
      "Resource": [
        "arn:aws:glue:<region>:<Target-Account-Id>:catalog",
        "arn:aws:glue:<region>:<Target-Account-Id>:database/DatabaseName"
      ],
      "Condition": {
        "StringLike": {
          "aws:SourceArn": "arn:aws:dynamodb:<region>:<Account>:table/<table-name>"
        }
      }
    },
    { // Allow Glue to Authorize the Inbound Integration on behalf of Bob
      "Principal": {
        "Service": [
          "glue.amazonaws.com"
        ]
```

```
      },
      "Effect": "Allow",
      "Action": [
        "glue:AuthorizeInboundIntegration"
      ],
      "Resource": [
        "arn:aws:glue:<region>:<Target-Account-Id>:catalog",
        "arn:aws:glue:<region>:<Target-Account-Id>:database/DatabaseName"
      ],
      "Condition": {
        "StringEquals": {
          "aws:SourceArn": "arn:aws:dynamodb:<region>:<account-id>:table/<table-name>"
        }
      }
    }
  ]
}
```

ターゲット IAM ロールの作成

以下の許可と信頼関係を持つターゲット IAM ロールを作成します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": "s3:ListBucket",
            "Resource": "arn:aws:s3:::<target iceberg table s3 bucket>",
            "Effect": "Allow"
        },
        {
            "Action": [
                "s3:GetObject",
                "s3:PutObject",
                "s3:DeleteObject"
            ],
            "Resource": "arn:aws:s3:::<target iceberg table s3 bucket>/prefix/*",
            "Effect": "Allow"
        },
        {
            "Action": [
                "glue:GetDatabase"
            ],
```

```
            "Resource": [
                "arn:aws:glue:<region>:<account-id>:catalog",
                "arn:aws:glue:<region>:<account-id>:database/DatabaseName"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "glue:CreateTable",
                "glue:GetTable",
                "glue:UpdateTable",
                "glue:GetTableVersion",
                "glue:GetTableVersions",
                "glue:GetResourcePolicy"
            ],
            "Resource": [
                "arn:aws:glue:<region>:<account-id>:catalog",
                "arn:aws:glue:<region>:<account-id>:database/<DatabaseName>",
                "arn:aws:glue:<region>:<account-id>:table/<DatabaseName>/*"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "cloudwatch:PutMetricData"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "cloudwatch:namespace": "AWS/Glue/ZeroETL"
                }
            },
            "Effect": "Allow"
        },
        {
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": "*",
            "Effect": "Allow"
        }
    ]
```

```
}
```

AWS Glue サービスがロールを引き受けるには、次の信頼ポリシーを追加します。

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

Amazon Redshift データウェアハウスの作成

ゼロ ETL 統合ターゲットが Amazon Redshift データウェアハウスの場合、データウェアハウスをまだ作成していない場合は作成します。Amazon Redshift Serverless ワークグループを作成するには、「名前空間を伴うワークグループの作成」を参照してください。Amazon Redshift クラスターを作成するには、「クラスターの作成」を参照してください。

正常に統合するには、ターゲットの Amazon Redshift ワークグループまたはクラスターで `enable_case_sensitive_identifier` パラメータが有効になっている必要があります。大文字と小文字の区別を有効にする方法の詳細については、Amazon Redshift 管理ガイドの「データウェアハウスの大文字と小文字の区別を有効にする」を参照してください。

Amazon Redshift ワークグループまたはクラスターのセットアップが完了したら、データウェアハウスを設定する必要があります。詳細については、Amazon Redshift 管理ガイドの「ゼロ ETL 統合の開始方法」を参照してください。

ゼロ ETL 統合用の VPC のセットアップ

ゼロ ETL 統合用の VPC のセットアップをするには、以下を行います。

1. [VPC] > [VPC] で、[VPC を作成] を選択します。
   a. [VPC など] を選択します。
   b. VPC 名を設定します。

c. IPv4 CIDR: 10.0.0.0/16 を設定します。

d. AZ の数を 1 に設定します。

e. パブリックサブネットとプライベートサブネットの数を 1 に設定します。

f. [NAT ゲートウェイ] を None に設定します。

g. [VPC エンドポイント] を S3 ゲートウェイに設定します。

h. [DNS ホスト名] と [DNS 解決] を有効にします。

2. [エンドポイント] で、[エンドポイントの作成] を選択します。
3. VPC のプライベートサブネットにこれらのサービスのエンドポイントを作成します (デフォルトのセキュリティグループを使用します)。

a. com.amazonaws.us-east-1.lambda

b. com.amazonaws.us-east-1.glue

c. com.amazonaws.us-east-1.sts

AWS Glue 接続を作成するには:

1. AWS Glue > [データ接続] に移動し、[接続を作成] を選択します。
2. [ネットワーク] を選択します。
3. 作成した VPC、サブネット (プライベート)、デフォルトのセキュリティグループを選択します。

VPC のターゲットロールの設定

ターゲットロールには、ゼロ ETL 統合に必要な他のアクセス許可に加えて、以下のアクセス許可が必要です

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CustomerVpc",
            "Effect": "Allow",
            "Action": [
                "ec2:CreateTags",
                "ec2:DeleteTags",
                "ec2:DescribeRouteTables",
                "ec2:DescribeVpcEndpoints",
                "ec2:DescribeSecurityGroups",
```

```
                "ec2:DescribeSubnets",
                "ec2:CreateNetworkInterface",
                "ec2:DeleteNetworkInterface",
                "glue:GetConnection"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

ターゲットレッグリソースプロパティの設定

CLI を使用している場合は、ターゲットレッグリソースプロパティを、作成したターゲット AWS Glue データベースに設定します。ターゲットロール ARN と AWS Glue 接続名を渡します。

```
aws glue create-integration-resource-property \
--resource-arn arn:aws:glue:us-east-1:<account-id>:database/exampletarget \
--target-processing-properties '{"RoleArn" : "arn:aws:iam::<account-id>:role/example-role", "ConnectionName":"example-vpc-3"}' \
--endpoint-url https://example.amazonaws.com --region us-east-1
```

発生する可能性のあるクライアントエラー

VPC を使用して設定された統合で発生する可能性のあるクライアントエラーは以下のとおりです。

エラーメッセージ	必要なアクション
指定されたロールは、接続時に glue:GetConnection を実行する権限がありません。このアクセス許可をロールポリシーに追加し、統合が復旧するのを待ちます。	ロールポリシーの更新
指定されたロールには、ec2:DescribeSubnets を実行する権限がありません。このアクセス許可をロールポリシーに追加し、統合が復旧するのを待ちます。	ロールポリシーの更新

エラーメッセージ	必要なアクション
指定されたロールには、ec2:DescribeSecurityGroups を実行する権限がありません。このアクセス許可をロールポリシーに追加し、統合が復旧するのを待ちます。	ロールポリシーの更新
指定されたロールには、ec2:DescribeVpcEndpoints を実行する権限がありません。このアクセス許可をロールポリシーに追加し、統合が復旧するのを待ちます。	ロールポリシーの更新
指定されたロールには、ec2:DescribeRouteTables を実行する権限がありません。このアクセス許可をロールポリシーに追加し、統合が復旧するのを待ちます。	ロールポリシーの更新
提供されたロールには、ec2:CreateTags を実行する権限がありません。このアクセス許可をロールポリシーに追加し、統合が復旧するのを待ちます。	ロールポリシーの更新
指定されたロールには、ec2:CreateNetworkInterface を実行する権限がありません。このアクセス許可をロールポリシーに追加し、統合が復旧するのを待ちます。	ロールポリシーの更新
指定された接続サブネットには有効な S3 エンドポイントまたは NAT ゲートウェイが含まれていません。サブネットを更新し、統合が復旧するのを待ちます。	VPC サブネットエンドポイントの更新
接続サブネットが見つかりません。接続サブネットを更新し、統合が復旧するのを待ちます。	AWS Glue 接続の更新

エラーメッセージ	必要なアクション
接続セキュリティグループが見つかりません。接続セキュリティグループを更新し、統合が復旧するのを待ちます。	AWS Glue 接続の更新
提供された VPC 接続を介して S3 に接続することはできません。サブネット設定を更新し、統合が復旧するのを待ちます。	VPC サブネットエンドポイントの更新
提供された VPC 接続を介して Lambda に接続することはできません。サブネット設定を更新し、統合が復旧するのを待ちます。	VPC サブネットエンドポイントの更新

ゼロ ETL クロスアカウント統合の設定

ゼロ ETL クロスアカウント統合を設定するには:

1. 「カタログリソースベースアクセス (RBAC) ポリシーの指定」の説明に従って、ターゲットリソースポリシーを設定します。ソースアカウントロールがターゲットリソースで明示的に許可されていることを確認します。
2. ソースアカウントロール (統合の作成に使用されるロール) に以下が設定されていることを確認します。

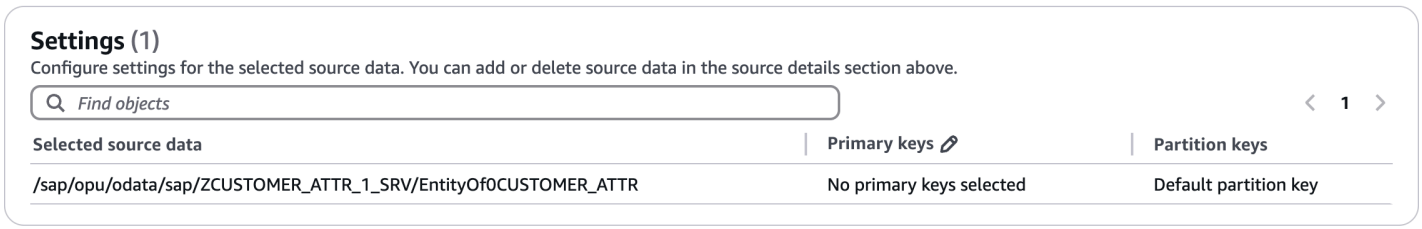
```
{
    "Version": "2012-10-17",
    "Statement": [
    {
        "Sid": "Stmt123456789012",
        "Action": [
            "glue:CreateInboundIntegration"
        ],
        "Effect": "Allow",
        "Resource": [
            "arn:aws:glue:<region>:<target-account-id>:catalog",
            "arn:aws:glue:<region>:<target-account-id>:database/DatabaseName"
        ]
    }]
}
```

3. 「統合の作成」の説明に従って統合を作成します。

ゼロ ETL 統合のソースの設定

特別な SAP エンティティのサポート

ほとんどの SaaS エンティティでは、データ処理中に有効なプライマリキーセットを決定しますが、有効なプライマリキーセットを入力として提供する追加のステップ、特に `EntityOf` で始まる SAP エンティティが必要になるものもあります。`EntityOf` エンティティを選択すると、プライマリキーのセットを指定するように指示されます。



Amazon DynamoDB ソースの設定

ソース Amazon DynamoDB テーブルからデータにアクセスするには、AWS Glue でテーブルを記述し、そこからデータをエクスポートするためのアクセスが必要です。Amazon DynamoDB では、リソースベースアクセス (RBAC) ポリシーの設定を許可する機能が最近導入されました。

次のリソースベースアクセス (RBAC) ポリシーの例では、統合にワイルドカード (*) を使用しています。

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Sid": "1111",
        "Effect": "Allow",
        "Principal": {
            "Service": "glue.amazonaws.com"
        },
        "Resource":"*",
        "Action": [
            "dynamodb:ExportTableToPointInTime",
            "dynamodb:DescribeTable",
            "dynamodb:DescribeExport"
```

```
        ],
        "Condition": {
            "StringEquals": {
                "aws:SourceAccount": "<account-id>"
            },
            "StringLike": {
                "aws:SourceArn": "arn:aws:glue:<region>:<account-id>:integration:*"
            }
        }
    }]
}
```

1. レプリケートする DynamoDB の場合は、上記の RBAC ポリシーテンプレートをテーブルのリソースベースのポリシーに貼り付け、フィールドに入力します。
2. ポリシーを制限したい場合は、統合を作成した後にポリシーを更新し、integrationArn 全体を指定して、StringLike の代わりに StringEquals 条件を使用する必要があります。
3. ポイントインタイムリカバリ (PITR) が DynamoDB テーブルに対して有効になっていることを確認します。
4. リソースベースアクセス (RBAC) ポリシーに Describe Export を追加していることを確認してください。

次のコマンドを使用して、RBAC ポリシーをテーブルに追加することもできます。

```
aws dynamodb put-resource-policy \
--resource-arn arn:aws:dynamodb:<region>:<account-id>:table/<ddb-table-name> \
--policy file://resource-policy-with-condition.json \
--region <region>
```

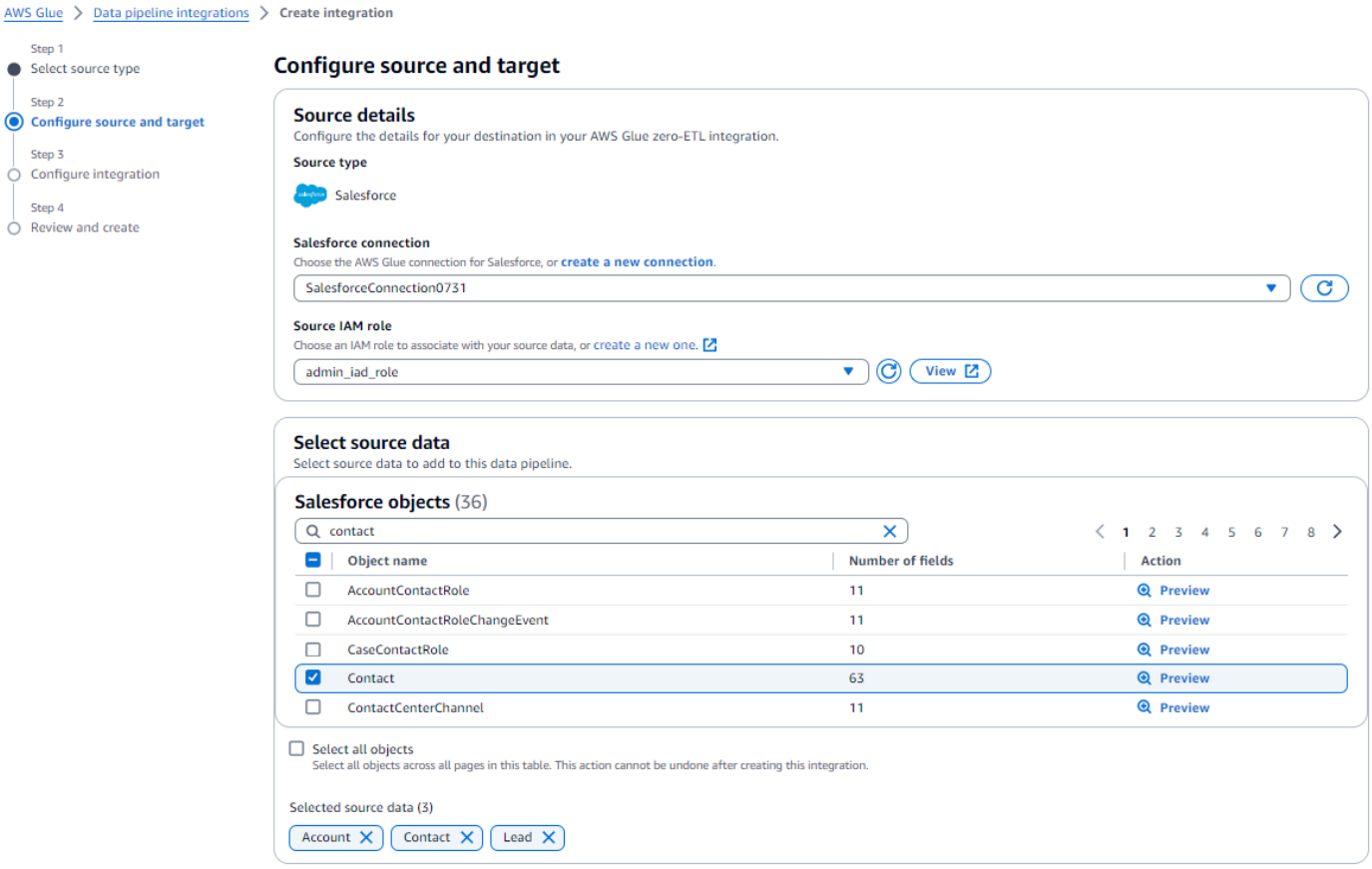
ポリシーが正しく適用されていることを確認するには、次のコマンドを使用してテーブルのリソースポリシーを取得します。

```
aws dynamodb get-resource-policy \
--resource-arn arn:aws:dynamodb:<region>:<account-id>:table/<ddb-table-name>
--region <region>
```

Salesforce ソースの設定

Salesforceソースへの接続を作成するには、「Salesforce への接続」を参照してください。

接続を作成したら、レプリケートするソースデータを指定できます。



ゼロ ETL 統合を使用すると、サポートされているエンティティに対して DDL オペレーションを実行できます。サポートされていないエンティティのリストについては、「Salesforce でサポートされていないエンティティとフィールド」を参照してください。

Salesforce Marketing Cloud Account Engagement ソースの設定

Salesforce Marketing Cloud Account Engagement ソースの接続を作成するには、「 Salesforce Marketing Cloud Account Engagement への接続 」を参照してください。

ゼロ ETL 統合を使用すると、以下のサポートされているエンティティに対して DDL オペレーションを実行できます。

エンティティラベル	エンティティ名
キャンペーン	キャンペーン
リスト	リスト

エンティティラベル	エンティティ名
動的コンテンツ	dynamic-content
List Membership	list-membership
Prospect	prospect
ユーザー	ユーザー
EmailTemplate	email-template
EngagementStudioProgram	engagement-studio-program
Landing Page	landing-page
List Email	list-email

追加の Salesforce 設定

Salesforceのデータ連携機能を使うためには、Glueデータベースに対して、Lake Formationで適切なアクセス許可の設定が必要です。アクセス許可がない場合は、ログから `IngestionFailed` を取得し、次のエラーが発生します。

```
"errorMessage": "Insufficient lake formation permissions on Target Glue database."
```

SAP OData ソースの設定

SAP OData ソースの接続を作成するには、「 SAP OData への接続 」を参照してください。

ゼロ ETL 統合の SAP OData コネクタは、`EntityOf` で始まるエンティティをサポートしていません。

ServiceNow ソースの設定

ServiceNow ソースの接続を作成するには、「 への接続 ServiceNow 」を参照してください。

Zendesk ソースの設定

Zendesk ソースの接続を作成するには、「 Zendesk への接続 」を参照してください。

ゼロ ETL 統合を使用すると、サポートされているエンティティに対して以下の DDL オペレーションを実行できます。

エンティティラベル	エンティティ名	Create supported	Update supported	Delete supported
チケット	tickets	Y	Y	Y
ユーザー	ユーザー	Y	Y	Y
Satisfaction Rating	satisfaction-rating	Y	Y	N
記事	記事	Y	Y	N
組織	組織	Y	Y	Y
呼び出し	calls	Y	Y	N
Call Legs	legs	Y	Y	N

Zoho CRM ソースの設定

Zoho CRM ソースの接続を作成するには、「Zoho CRM への接続」を参照してください。

ゼロ ETL 統合を使用すると、サポートされているエンティティに対して以下の DDL オペレーションを実行できます。

エンティティラベル	エンティティ名	DML-Insert supported	DML-Modify supported	DML-Delete supported	DDL-Insert supported	DDL-Modify supported	DDL-Delete supported
Leads	lead	Y	Y	Y	Y	Y	Y
アカウント	アカウント	Y	Y	Y	Y	Y	Y
問い合わせ	contact	Y	Y	Y	Y	Y	Y

エンティティラベル	エンティティ名	DML-Insert supported	DML-Modify supported	DML-Delete supported	DDL-Insert supported	DDL-Modify supported	DDL-Delete supported
キャンペーン	キャンペーン	Y	Y	Y	Y	Y	Y
タスク	タスク	Y	Y	Y	Y	Y	Y
イベント	イベント	Y	Y	Y	Y	Y	Y
呼び出し	call	Y	Y	Y	Y	Y	Y
解決方法	solution	Y	Y	Y	Y	Y	Y
製品	product	Y	Y	Y	Y	Y	Y
Vendors	ベンダー	Y	Y	Y	Y	Y	Y
Quotes	quote	Y	Y	Y	Y	Y	Y
Sales Orders	sales-order	Y	Y	Y	Y	Y	Y
Purchase Orders	purchase-order	Y	Y	Y	Y	Y	Y
請求書	請求書	Y	Y	Y	Y	Y	Y
Cases	case	Y	Y	Y	Y	Y	Y
Price Books	price-book	Y	Y	Y	Y	Y	Y

Facebook 広告ソースの設定

Facebook 広告ソースの接続を作成するには、「Facebook 広告への接続」を参照してください。

ゼロ ETL 統合を使用すると、サポートされているエンティティに対して以下の DDL オペレーションを実行できます。

エンティティラベル	エンティティ名	Create supported	Update supported	Delete supported
Adset	*/adsets	Y	Y	Y
キャンペーン	*/campaigns	Y	Y	Y
Ads	*/ads	Y	Y	Y

Instagram 広告ソースの設定

Instagram 広告ソースの接続を作成するには、「Instagram 広告への接続」を参照してください。

ゼロ ETL 統合を使用すると、サポートされているエンティティに対して以下の DDL オペレーションを実行できます。

エンティティ名	Create supported	Update supported	Delete supported
*/adsets	Y	Y	Y
*/campaigns	Y	Y	Y
*/ads	Y	Y	Y

Salesforce でサポートされていないエンティティとフィールド

次の Salesforce エンティティまたはフィールドは、Salesforce ソースとのゼロ ETL 統合ではサポートされていません。

```
AccountChangeEvent, AccountContactRoleChangeEvent, AccountHistory, AccountShare,
 ActiveFeatureLicenseMetric, ActivePermSetLicenseMetric, ActiveProfileMetric,
 ActivityFieldHistory, amzsec__asi_Telemetry_Data_Store__ChangeEvent,
 amzsec__asi_Telemetry_Data_Store__History, amzsec__asi_Telemetry_Data_Store__Share,
 amzsec__asi_Telemetry_Job_Log__ChangeEvent, amzsec__asi_Telemetry_Job_Log__History,
 amzsec__asi_Telemetry_Job_Log__Share, amzsec__asi_Telemetry_Requirement__ChangeEvent,
 amzsec__asi_Telemetry_Requirement__History, amzsec__asi_Telemetry_Requirement__Share,
 ApexClass, ApexComponent, ApexLog, ApexPage, ApexTestQueueItem, ApexTestResult,
 ApexTrigger, AssetChangeEvent, AssetHistory, AssetRelationshipHistory,
 AssetShare, AssignmentRule, AssociatedLocationHistory, AsyncApexJob,
```

```
AuditTrailFileExportShare, AuthorizationFormConsentChangeEvent,
AuthorizationFormConsentHistory, AuthorizationFormConsentShare,
AuthorizationFormDataUseHistory, AuthorizationFormDataUseShare,
AuthorizationFormHistory, AuthorizationFormShare, AuthorizationFormTextHistory,
AuthProvider, AuthSession, BatchJobHistory, BatchJobPartFailedRecordHistory,
BatchJobPartHistory, BatchJobShare, BrandTemplate, BriefcaseAssignmentChangeEvent,
BriefcaseDefinitionChangeEvent, BusinessBrandShare, BusinessHours,
BusinessProcess, CalcMatrixColumnRangeHistory, CalcProcStepRelationshipHistory,
CalculationMatrixColumnHistory, CalculationMatrixHistory, CalculationMatrixRowHistory,
CalculationMatrixShare, CalculationMatrixVersionHistory, CalculationProcedureHistory,
CalculationProcedureShare, CalculationProcedureStepHistory,
CalculationProcedureVariableHistory, CalculationProcedureVersionHistory, Calendar,
CalendarViewShare, CallCenter, CallCoachConfigModifyEvent, CampaignChangeEvent,
CampaignHistory, CampaignMemberChangeEvent, CampaignMemberStatusChangeEvent,
CampaignShare, CaseChangeEvent, CaseHistory, CaseHistory2 CaseHistory2ChangeEvent,
CaseRelatedIssueChangeEvent, CaseRelatedIssueHistory, CaseShare, CaseStatus,
CaseTeamMember, CaseTeamRole, CaseTeamTemplate, CaseTeamTemplateMember,
CaseTeamTemplateRecord, CategoryNode, ChangeRequestChangeEvent, ChangeRequestHistory,
ChangeRequestRelatedIssueChangeEvent, ChangeRequestRelatedIssueHistory,
ChangeRequestRelatedItemChangeEvent, ChangeRequestRelatedItemHistory,
ChangeRequestShare, ChatRetirementRdyMetrics, ChatterActivity, ClientBrowser,
CollaborationGroup, CollaborationGroupMember, CollaborationGroupMemberRequest,
CollaborationInvitation, CommSubscriptionChannelTypeHistory,
CommSubscriptionChannelTypeShare, CommSubscriptionConsentChangeEvent,
CommSubscriptionConsentHistory, CommSubscriptionConsentShare, CommSubscriptionHistory,
CommSubscriptionShare, CommSubscriptionTimingHistory, Community, ConnectedApplication,
ContactChangeEvent, ContactHistory, ContactPointAddressChangeEvent,
ContactPointAddressHistory, ContactPointAddressShare, ContactPointConsentChangeEvent,
ContactPointConsentHistory, ContactPointConsentShare, ContactPointEmailChangeEvent,
ContactPointEmailHistory, ContactPointEmailShare, ContactPointPhoneChangeEvent,
ContactPointPhoneHistory, ContactPointPhoneShare, ContactPointTypeConsentChangeEvent,
ContactPointTypeConsentHistory, ContactPointTypeConsentShare, ContactRequestShare,
ContactShare, ContentDocumentChangeEvent, ContentDocumentHistory,
ContentDocumentLink, ContentDocumentLinkChangeEvent, ContentDocumentSubscription,
ContentFolderItem, ContentFolderLink, ContentFolderMember, ContentNote,
ContentNotification, ContentTagSubscription, ContentUserSubscription,
ContentVersionChangeEvent, ContentVersionComment, ContentVersionHistory,
ContentVersionRating, ContentWorkspace, ContentWorkspaceMember,
ContentWorkspacePermission, ContentWorkspaceSubscription, ContractChangeEvent,
ContractHistory, ContractLineItemChangeEvent, ContractLineItemHistory,
ContractStatus, Conversation, ConversationParticipant, CronJobDetail,
CronTrigger, CustomBrand, CustomBrandAsset, CustomerShare, CustomHttpHeader,
DashboardComponent, DataUseLegalBasisHistory, DataUseLegalBasisShare,
DataUsePurposeHistory, DataUsePurposeShare, DecisionTableRecordset, DeleteEvent,
```

```
DocumentAttachmentMap, Domain, DomainSite, DTRecordsetReplicaShare, EmailBounceEvent,
EmailMessageChangeEvent, EmailServicesAddress, EmailServicesFunction,
EmailTemplate, EmailTemplateChangeEvent, EngagementAttendeeChangeEvent,
EngagementAttendeeHistory, EngagementChannelTypeHistory, EngagementChannelTypeShare,
EngagementInteractionChangeEvent, EngagementInteractionHistory,
EngagementInteractionShare, EngagementInterface, EngagementTopicChangeEvent,
EngagementTopicHistory, EntitlementChangeEvent, EntitlementHistory,
EntitlementTemplate, EntityMilestoneHistory, EntitySubscription, EventChangeEvent,
EventRelationChangeEvent, EventRelayConfigChangeEvent, ExpressionSetHistory,
ExpressionSetShare, ExpressionSetVersionHistory, ExternalEventMappingShare,
FeedAttachment, FeedLike, FeedPollChoice, FeedPollVote, FeedSignal,
FieldPermissions, FieldSecurityClassification, FiscalYearSettings,
FlowInterviewLogShare, FlowInterviewShare, FlowOrchestrationEvent,
FlowOrchestrationInstanceShare, FlowOrchestrationStageInstanceShare,
FlowOrchestrationStepInstanceShare, FlowOrchestrationWorkItemShare,
FlowRecordShare, FlowRecordVersionChangeEvent, FlowTestResultShare, Folder,
Group, GroupMember, Holiday, IdeaComment, IdpEventLog, ImageHistory, ImageShare,
IncidentChangeEvent, IncidentHistory, IncidentRelatedItemChangeEvent,
IncidentRelatedItemHistory, IncidentShare, IndividualChangeEvent,
IndividualHistory, IndividualShare, KnowledgeableUser, LeadChangeEvent, LeadHistory,
LeadShare, LeadStatus, LightningExitByPageMetrics, LightningToggleMetrics,
LightningUsageByAppTypeMetrics, LightningUsageByBrowserMetrics,
LightningUsageByFlexiPageMetrics, LightningUsageByPageMetrics, ListEmailChangeEvent,
ListEmailShare, ListView, LocationChangeEvent, LocationHistory, LocationShare,
LocationTrustMeasureShare, LoginHistory, LoginIp, MacroChangeEvent,
MacroHistory, MacroInstructionChangeEvent, MacroShare, MacroUsageShare,
ManagedContentVariantChangeEvent, MessagingEndUserHistory, MessagingEndUserShare,
MessagingSessionHistory, MessagingSessionShare, MilestoneType, MLEngagementEvent,
ObjectPermissions, OpportunityChangeEvent, OpportunityContactRoleChangeEvent,
OpportunityFieldHistory, OpportunityLineItemChangeEvent, OpportunityShare,
OpportunityStage, OrderChangeEvent, OrderHistory, OrderItemChangeEvent,
OrderItemHistory, OrderShare, OrderStatus, Organization, OrgEmailAddressSecurity,
OrgWideEmailAddress, OutgoingEmail, OutgoingEmailRelation, PackageLicense,
PartnerRole, PartyConsentChangeEvent, PartyConsentHistory, PartyConsentShare,
Period, PermissionSet, PermissionSetAssignment, PermissionSetTabSetting,
Pricebook2ChangeEvent, Pricebook2History, PricebookEntryChangeEvent,
PricebookEntryHistory, PrivacyJobSessionShare, PrivacyObjectSessionShare,
PrivacyRTBFRequestHistory, PrivacyRTBFRequestShare, PrivacySessionRecordFailureShare,
ProblemChangeEvent, ProblemHistory, ProblemIncidentChangeEvent,
ProblemIncidentHistory, ProblemRelatedItemChangeEvent, ProblemRelatedItemHistory,
ProblemShare, ProcessDefinition, ProcessExceptionEvent, ProcessExceptionShare,
ProcessInstanceChangeEvent, ProcessInstanceStep, ProcessInstanceStepChangeEvent,
ProcessNode, Product2ChangeEvent, Product2History, ProductEntitlementTemplate,
Profile, ProfileSkillEndorsementHistory, ProfileSkillHistory,
```

```
 ProfileSkillShare, ProfileSkillUserHistory, PromptActionShare, PromptErrorShare,
 QueueSobject, QuickTextChangeEvent, QuickTextHistory, QuickTextShare,
 QuickTextUsageShare, RecentlyViewed, RecommendationChangeEvent,
 RecordActionHistory, RecordAlertHistory, RecordAlertShare, RecordType,
 ScorecardShare, SellerHistory, SellerShare, ServiceContractChangeEvent,
 ServiceContractHistory, ServiceContractShare, SetupAuditTrail, SetupEntityAccess,
 SharingRecordCollectionShare, Site, SiteHistory, SiteRedirectMapping,
 SocialPersonaHistory, SocialPostChangeEvent, SocialPostHistory, SocialPostShare,
 SolutionHistory, SolutionStatus, StaticResource, StreamingChannelShare,
 TableauHostMappingShare, TaskChangeEvent, TaskPriority, TaskStatus,
 ThreatDetectionFeedback, TimelineObjectDefinitionChangeEvent, TodayGoalShare,
 Topic, TopicUserEvent, Translation, User, UserAppMenuCustomizationShare,
 UserChangeEvent, UserDefinedLabelAssignmentShare, UserDefinedLabelShare,
 UserEmailPreferredPersonShare, UserLicense, UserLogin, UserPackageLicense,
 UserPreference, UserPrioritizedRecordShare, UserProvisioningRequestShare,
 UserRole, UserShare, VideoCallChangeEvent, VideoCallParticipantChangeEvent,
 VideoCallRecordingChangeEvent, VideoCallShare, VisualforceAccessMetrics,
 VoiceCallChangeEvent, VoiceCallRecordingChangeEvent, VoiceCallShare, Vote,
 WebLink, WorkAccessShare, WorkBadgeDefinitionHistory, WorkBadgeDefinitionShare,
 WorkOrderChangeEvent, WorkOrderHistory, WorkOrderLineItemChangeEvent,
 WorkOrderLineItemHistory, WorkOrderLineItemStatus, WorkOrderShare, WorkOrderStatus,
 WorkPlanChangeEvent, WorkPlanHistory, WorkPlanShare, WorkPlanTemplateChangeEvent,
 WorkPlanTemplateEntryChangeEvent, WorkPlanTemplateEntryHistory,
 WorkPlanTemplateHistory, WorkPlanTemplateShare, WorkStepChangeEvent, WorkStepHistory,
 WorkStepStatus, WorkStepTemplateChangeEvent, WorkStepTemplateHistory,
 WorkStepTemplateShare, WorkThanksShare
```

ServiceNow でサポートされていないエンティティとフィールド

次以下の ServiceNow エンティティまたはフィールドは、ServiceNow ソースとのゼロ ETL 統合ではサポートされていません。

```
ais_acl_overrides, ais_async_genius_result, ais_async_request, ais_connection,
 ais_genius_result_configuration_parameters, ais_partition_health,
 ais_partition_health_response, ais_publish_history, ais_relevancy_training_execution,
 ais_relevancy_training_staging, ais_search_profile_relevancy_model,
 catalog_draft_entities, clone_log, clone_log0000, clone_log0001, clone_log0002,
 clone_log0003, clone_log0004, clone_log0005, clone_log0006, clone_log0007,
 cmdb_ie_context, cmdb_ie_log, cmdb_ie_run, cmdb_ire_partial_payloads_index,
 cmdb_qb_result_base$par1, cmdb$par1, discovery_log, discovery_log0000,
 discovery_log0001, discovery_log0002, discovery_log0003, discovery_log0004,
 discovery_log0005, discovery_log0006, discovery_log0007, ecc_agent_log,
 entitlement_data, entl_subscription_map, gs_entitlement_plugin_mapping,
```

```
ih_transaction_exclusion, import_log, import_log0000, import_log0001, import_log0002,
import_log0003, import_log0004, import_log0005, import_log0006, import_log0007,
jrobin_archive, jrobin_database, jrobin_datasource, jrobin_definition, jrobin_graph,
jrobin_graph_line, jrobin_graph_set, jrobin_graph_set_member, jrobin_shard,
jrobin_shard_location, license_role_discovery_run, logger_configuration_validation,
m2m_analytics_event_logger, m2m_user_consent_info, ml_artifact_object_store,
np$sys_gen_ai_filter_sample, np$sys_ui_element, np$sys_ui_list_element,
one_api_service_plan_feature_invocation, one_api_service_plan_invocation,
open_nlu_predict_log, open_nlu_predict_log0000, open_nlu_predict_log0001,
open_nlu_predict_log0002, open_nlu_predict_log0003, open_nlu_predict_log0004,
open_nlu_predict_log0005, open_nlu_predict_log0006, open_nlu_predict_log0007,
pa_diagnostic_log, pa_diagnostic_log0000, pa_diagnostic_log0001,
pa_diagnostic_log0002, pa_diagnostic_log0003, pa_diagnostic_log0004,
pa_diagnostic_log0005, pa_diagnostic_log0006, pa_diagnostic_log0007, pa_favorites,
pa_job_log_rows, pa_job_log_rows0000, pa_job_log_rows0001, pa_job_log_rows0002,
pa_job_log_rows0003, pa_job_log_rows0004, pa_job_log_rows0005, pa_job_log_rows0006,
pa_job_log_rows0007, pa_migration_ignored_scores, pa_scores_l1, pa_scores_l2,
pa_scores_migration_groups, par_dashboard_conversion_backup, promin_log,
promin_log0000, promin_log0001, promin_log0002, promin_log0003, promin_log0004,
promin_log0005, promin_log0006, promin_log0007, promin_request_object,
proposed_change_verification_log, proposed_change_verification_log0000,
proposed_change_verification_log0001, proposed_change_verification_log0002,
proposed_change_verification_log0003, proposed_change_verification_log0004,
proposed_change_verification_log0005, proposed_change_verification_log0006,
proposed_change_verification_log0007, protected_table_log, protected_table_log0000,
protected_table_log0001, protected_table_log0002, protected_table_log0003,
protected_table_log0004, protected_table_log0005, protected_table_log0006,
protected_table_log0007, pwd_history, qb_query_results, scan_log,
scan_log0000, scan_log0001, scan_log0002, scan_log0003, scan_log0004,
scan_log0005, scan_log0006, scan_log0007, schema_validator_error,
sla_repair_log_entry, sla_repair_log_entry0000, sla_repair_log_entry0001,
sla_repair_log_entry0002, sla_repair_log_entry0003, sla_repair_log_entry0004,
sla_repair_log_entry0005, sla_repair_log_entry0006, sla_repair_log_entry0007,
sla_repair_log_message, sla_repair_log_message0000, sla_repair_log_message0001,
sla_repair_log_message0002, sla_repair_log_message0003, sla_repair_log_message0004,
sla_repair_log_message0005, sla_repair_log_message0006, sla_repair_log_message0007,
sn_bm_client_activity, sn_ci_analytics_st_actionable_notifs,
sn_ci_analytics_st_conv_completion_by_cat, sn_ci_analytics_st_conv_dynamic_property,
sn_ci_analytics_st_conversation, sn_ci_analytics_st_count_by_date,
sn_ci_analytics_st_event_occurrence, sn_ci_analytics_st_event_property_value_trend,
sn_ci_analytics_st_issue_auto_resolution, sn_ci_analytics_st_no_clicks,
sn_ci_analytics_st_no_results, sn_ci_analytics_st_property_summary_by_event,
sn_ci_analytics_st_session_count_per_locale, sn_ci_analytics_st_session_duration,
sn_ci_analytics_st_spokes_usage, sn_ci_analytics_st_topic_execution_stats,
```

```
sn_ci_analytics_st_topic_occurrence, sn_ci_analytics_st_trending_content,
sn_ci_analytics_st_trending_queries, sn_ci_analytics_st_users,
sn_cs_plugin_signatures, sn_cs_telemetry_log, sn_dfc_application, sn_dfc_product,
sn_employee_position, sn_entitlement_st_subscription_application_family,
sn_entitlement_st_subscription_application_users, sn_hr_sp_st_relevant_for_you,
sn_instance_clone_log, sn_instance_clone_log0000, sn_instance_clone_log0001,
sn_instance_clone_log0002, sn_instance_clone_log0003, sn_instance_clone_log0004,
sn_instance_clone_log0005, sn_instance_clone_log0006, sn_instance_clone_log0007,
sn_km_mr_st_kb_knowledge, sn_me_st_topic, sn_rf_conditional_definition,
sn_rf_evaluation_type, sn_rf_evaluation_type_input, sn_rf_recommendation_action,
sn_rf_recommendation_experience, sn_rf_recommendation_history,
sn_rf_recommendation_rule, sn_rf_record_display_configuration, sn_rf_trend_definition,
sn_sub_man_st_account_level_entitlement, sn_sub_man_st_gen_ai_metadata,
sn_sub_man_st_instance_used_assist_count, sn_sub_man_st_now_assist_creator_instances,
sn_sub_man_st_now_assists_aggregate, sn_sub_man_st_subscribed_groups,
sn_sub_man_st_subscription_insights, sn_sub_man_st_subscription_license_detail_metric,
sn_sub_man_st_unallocated_group_recommendation, sn_sub_man_st_unconfirmed_user_group,
sn_wn_user_app_activity, sn_wn_user_content_activity, snc_monitorable_item,
snpar_sched_export_v_scheduled_export_visualization, spotlight,
spotlight_audit, spotlight_copy_log_row, spotlight_copy_log_row0000,
spotlight_copy_log_row0001, spotlight_copy_log_row0002, spotlight_copy_log_row0003,
spotlight_copy_log_row0004, spotlight_copy_log_row0005, spotlight_copy_log_row0006,
spotlight_copy_log_row0007, spotlight_job_log_row, spotlight_job_log_row0000,
spotlight_job_log_row0001, spotlight_job_log_row0002, spotlight_job_log_row0003,
spotlight_job_log_row0004, spotlight_job_log_row0005, spotlight_job_log_row0006,
spotlight_job_log_row0007, st_dfc_performance_metric, st_license_detail_metric,
st_on_call_hour, st_sc_wizard_question, st_sys_catalog_items_and_variable_sets,
st_sys_design_system_icon, subscription_instance_stats, svc_container_config,
svc_environment_config, svc_layer_config, svc_model_assoc_ci,
svc_model_checkpoint_attr, svc_model_obj_cluster, svc_model_obj_constraint,
svc_model_obj_deployable, svc_model_obj_element, svc_model_obj_impact,
svc_model_obj_impactrule, svc_model_obj_package, svc_model_obj_path,
svc_model_obj_relation, svc_model_obj_service, sys_administrative_script_transaction,
sys_amb_message, sys_amb_processor, sys_analytics_batch_state,
sys_analytics_config, sys_analytics_data_points_error, sys_analytics_event,
sys_analytics_logger, sys_analytics_logger_field, sys_app_payload_loader_rule,
sys_app_payload_unloader_rule, sys_app_scan_payload, sys_app_scan_variable,
sys_app_scan_variable_type, sys_archive_destroy_log, sys_archive_destroy_run,
sys_archive_log, sys_archive_run, sys_atf_transaction_log, sys_attachment_doc,
sys_attachment_doc_v2, sys_attachment_soft_deleted, sys_audit, sys_audit_relation,
sys_auth_policy_api_allowed, sys_aw_registered_scripting_modal,
sys_cache_flush, sys_data_egress_source, sys_dm_delete_count,
sys_export_set_log, sys_export_set_log0000, sys_export_set_log0001,
sys_export_set_log0002, sys_export_set_log0003, sys_export_set_log0004,
```

```
sys_export_set_log0005, sys_export_set_log0006, sys_export_set_log0007,
sys_flow_compiled_flow, sys_flow_compiled_flow_chunk, sys_flow_context_chunk,
sys_flow_context_chunk_archive, sys_flow_context_inputs_chunk,
sys_flow_execution_history, sys_flow_log, sys_flow_log0000, sys_flow_log0001,
sys_flow_log0002, sys_flow_log0003, sys_flow_plan_context_binding,
sys_flow_report_doc, sys_flow_report_doc_chunk, sys_flow_report_doc_chunk_archive,
sys_flow_runtime_state_chunk, sys_flow_runtime_value_chunk,
sys_flow_subflow_plan_chunk, sys_flow_trigger_plan_chunk, sys_flow_val_listener,
sys_flow_value, sys_flow_value_chunk, sys_gen_ai_config_example,
sys_gen_ai_feature_mapping, sys_gen_ai_strategy_mapping, sys_gen_ai_usage_log,
sys_generative_ai_capability_definition, sys_generative_ai_log,
sys_generative_ai_response_validator, sys_generative_ai_validator,
sys_geo_routing, sys_geo_routing_config, sys_hop_token, sys_hub_action_plan_chunk,
sys_hub_snapshot_chunk, sys_journal_field, sys_journal_field_edit, sys_json_chunk,
sys_kaa_policy, sys_kaa_subidentity_assertion, sys_kaa_user_policy_mapping,
sys_mapplication, sys_mass_encryption_job, sys_notification_execution_log,
sys_notification_execution_log0000, sys_notification_execution_log0001,
sys_notification_execution_log0002, sys_notification_execution_log0003,
sys_notification_execution_log0004, sys_notification_execution_log0005,
sys_notification_execution_log0006, sys_notification_execution_log0007,
sys_nowmq_message, sys_nowmq_provider_param_definition, sys_orchestrator_action,
sys_pd_asset_configuration, sys_pd_context_chunk, sys_pd_context_log,
sys_pd_snapshot_chunk, sys_pd_trigger_license, sys_processing_framework_job,
sys_query_index_hint, sys_query_rewrite, sys_query_string_log,
sys_replication_queue, sys_replication_queue0, sys_replication_queue1,
sys_replication_queue2, sys_replication_queue3, sys_replication_queue4,
sys_replication_queue5, sys_replication_queue6, sys_replication_queue7,
sys_request_performance, sys_rollback_blacklisted, sys_rollback_conflict,
sys_rollback_incremental, sys_rollback_log, sys_rollback_log0000,
sys_rollback_log0001, sys_rollback_log0002, sys_rollback_log0003,
sys_rollback_log0004, sys_rollback_log0005, sys_rollback_log0006,
sys_rollback_log0007, sys_rollback_run, sys_rollback_schema_change,
sys_rollback_schema_conflict, sys_rollback_sequence, sys_scheduler_assignment,
sys_scheduler_memory_pressure_job_log, sys_script_adapter_rule,
sys_script_batch_adapter_rule, sys_search_source_filter, sys_service_authentication,
sys_signing_job, sys_suggestion_reader, sys_sync_history_review, sys_trend,
sys_unreferenced_preview, sys_unreferenced_record_rule, sys_upgrade_manifest,
sys_upgrade_state, sys_ux_asset_cache_buster, sys_ux_lib_component_prop,
sys_ux_lib_presource, sys_ux_page_action, sys_ux_page_action_binding,
sysevent_queue_runtime, syslog, syslog_app_scope0000, syslog_app_scope0001,
syslog_app_scope0002, syslog_app_scope0003, syslog_app_scope0004,
syslog_app_scope0005, syslog_app_scope0006, syslog_app_scope0007, syslog_email0000,
syslog_email0001, syslog_email0002, syslog_email0003, syslog_email0004,
syslog_email0005, syslog_email0006, syslog_email0007, syslog_transaction,
```

```
syslog_transaction0000, syslog_transaction0001, syslog_transaction0002,
syslog_transaction0003, syslog_transaction0004, syslog_transaction0005,
syslog_transaction0006, syslog_transaction0007, syslog0000, syslog0001, syslog0002,
syslog0003, syslog0004, syslog0005, syslog0006, syslog0007, ts_attachment,
ts_c_1_0, ts_c_1_1, ts_c_1_2, ts_c_1_3, ts_c_1_4, ts_c_1_5, ts_c_1_6, ts_c_1_7,
ts_c_1_8, ts_c_1_9, ts_c_10_0, ts_c_10_1, ts_c_10_2, ts_c_10_3, ts_c_10_4,
ts_c_10_5, ts_c_10_6, ts_c_10_7, ts_c_10_8, ts_c_10_9, ts_c_11_0, ts_c_11_1,
ts_c_11_2, ts_c_11_3, ts_c_11_4, ts_c_11_5, ts_c_11_6, ts_c_11_7, ts_c_11_8,
ts_c_11_9, ts_c_2_0, ts_c_2_1, ts_c_2_2, ts_c_2_3, ts_c_2_4, ts_c_2_5, ts_c_2_6,
ts_c_2_7, ts_c_2_8, ts_c_2_9, ts_c_3_0, ts_c_3_1, ts_c_3_2, ts_c_3_3, ts_c_3_4,
ts_c_3_5, ts_c_3_6, ts_c_3_7, ts_c_3_8, ts_c_3_9, ts_c_39_0, ts_c_39_1, ts_c_39_2,
ts_c_39_3, ts_c_39_4, ts_c_39_5, ts_c_39_6, ts_c_39_7, ts_c_39_8, ts_c_39_9,
ts_c_40_0, ts_c_40_1, ts_c_40_2, ts_c_40_3, ts_c_40_4, ts_c_40_5, ts_c_40_6,
ts_c_40_7, ts_c_40_8, ts_c_40_9, ts_c_41_0, ts_c_41_1, ts_c_41_2, ts_c_41_3,
ts_c_41_4, ts_c_41_5, ts_c_41_6, ts_c_41_7, ts_c_41_8, ts_c_41_9, ts_c_42_0,
ts_c_42_1, ts_c_42_2, ts_c_42_3, ts_c_42_4, ts_c_42_5, ts_c_42_6, ts_c_42_7,
ts_c_42_8, ts_c_42_9, ts_c_43_0, ts_c_43_1, ts_c_43_2, ts_c_43_3, ts_c_43_4,
ts_c_43_5, ts_c_43_6, ts_c_43_7, ts_c_43_8, ts_c_43_9, ts_c_44_0, ts_c_44_1,
ts_c_44_2, ts_c_44_3, ts_c_44_4, ts_c_44_5, ts_c_44_6, ts_c_44_7, ts_c_44_8,
ts_c_44_9, ts_c_45_0, ts_c_45_1, ts_c_45_2, ts_c_45_3, ts_c_45_4, ts_c_45_5,
ts_c_45_6, ts_c_45_7, ts_c_45_8, ts_c_45_9, ts_c_6_0, ts_c_6_1, ts_c_6_2,
ts_c_6_3, ts_c_6_4, ts_c_6_5, ts_c_6_6, ts_c_6_7, ts_c_6_8, ts_c_6_9, ts_c_7_0,
ts_c_7_1, ts_c_7_2, ts_c_7_3, ts_c_7_4, ts_c_7_5, ts_c_7_6, ts_c_7_7, ts_c_7_8,
ts_c_7_9, ts_c_8_0, ts_c_8_1, ts_c_8_2, ts_c_8_3, ts_c_8_4, ts_c_8_5, ts_c_8_6,
ts_c_8_7, ts_c_8_8, ts_c_8_9, ts_c_attachment, ts_chain, ts_deleted_doc,
ts_document, ts_field, ts_index_stats, ts_phrase, ts_search_stats, ts_v4_attachment,
ts_word, ua_app_metadata, ua_audit_stats, ua_extra_page, ua_monitor_property,
ua_monitor_property_audit, ua_shared_service, ua_sn_table_inventory, ua_sp_known_bot,
ua_upload_log, v_ais_result_improvement_rule_condition_builder_values,
v_cluster_nodes, v_cxs_search_resource, v_db_index, v_db_trigger, v_file_load_order,
v_interaction_context, v_iostats, v_mysql_proclist, v_mysql_status, v_mysql_variables,
v_on_call_report_cache, v_pa_par_combined_dashboard, v_pd_activity_condition_to_run,
v_pd_activity_start_rule_with_condition, v_pd_lane_condition_to_run,
v_sql_debug, v_st_kb_category, v_st_kb_most_viewed, v_st_kb_recently_viewed,
v_st_km_genai_mra_similar_task, v_st_popular_item, v_st_recent_item, v_st_sc_cat_item,
v_st_sc_catalog, v_st_sc_category, validator_run_summary, vtb_card_history,
wf_log, wf_log0000, wf_log0001, wf_log0002, wf_log0003, wf_log0004, wf_log0005,
wf_log0006, wf_log0007, spotlight_criteria, pa_snapshots, pa_widget_indicators,
pa_widgets, sn_cim_register, global, gsw_change_log, gsw_content, gsw_content_group,
gsw_content_information, gsw_status_of_content, multi_factor_browser_fingerprint,
multi_factor_criteria, pa_filters, password_policy, plan_execution,
plan_mysql, plan_oracle, plan_postgres, sc_rest_api_without_access_policy,
sn_actsub_activity, sn_actsub_activity_fanout, sn_actsub_activity_stream,
sn_actsub_activity_type, sn_actsub_atype_attributes, sn_actsub_atype_notif_pref,
```

```
sn_actsub_module, sn_actsub_notif_object, sn_actsub_subobject_stream,
sn_actsub_subscribable_object, sn_actsub_subscription_notif_pref,
sn_actsub_user_stream, sn_appclient_store_outbound_http_quota,
sn_appcreator_app_template, sn_critical_update, sn_docker_spoke_images,
sn_employee_app, sn_employee_app_access, sn_employee_app_access_criteria,
sn_entitlement_genai_assist_counts, sn_entitlement_genai_creator_user_counts,
sn_entitlement_genai_creator_users, sn_mif_instance, sn_mif_sync_data,
sn_mif_sync_status, sn_mif_table_registration, sn_mif_trust_config,
sn_vsc_best_practice_configurations, sn_vsc_best_practice_goals,
sn_vsc_changed_hardening_settings, sn_vsc_changed_scan_findings,
sn_vsc_check_security_area, sn_vsc_elevation_event, sn_vsc_event,
sn_vsc_export_event, sn_vsc_export_setting, sn_vsc_harc_compliance_status_lookup,
sn_vsc_hardening_compliance_scores, sn_vsc_impersonation_event,
sn_vsc_instance_hardening_settings, sn_vsc_login_event,
sn_vsc_scan_comparisons, sn_vsc_scan_summary, sn_vsc_security_check_categories,
sn_vsc_security_check_configurations, sn_vsc_security_configuration_groups,
sn_vsc_security_privacy_capabilities, sn_vsc_updated_settings,
sn_vsc_user_comparisons, sys_app_hash_inventory, sys_coalesce_strategy_deferred,
sys_flow_secure_data, sys_formula_function, sys_geocoding_request,
sys_global_file_hash, sys_import_set_row, sys_index, sys_index_explain,
sys_installation_schedule, sys_installation_schedule_item, sys_offline_app,
sys_package, sys_package_dependency_item, sys_package_dependency_m2m,
sys_plugins, sys_querystat, sys_reap_package, sys_scoped_plugin,
sys_stage_storage_alias, sys_storage_alias, sys_storage_table_alias, sys_store_app,
sys_table_partition, sys_upgrade_history_log, sys_user_public_credential,
sys_webauthn_authentication_request, sys_webauthn_registration_request,
syslog_app_scope, syslog_email, syslog_page_timing, ua_instance_state_config,
v_expression_cache, v_private_cache, v_shared_cache, sys_amb_message0002,
sys_amb_message0004, sys_amb_message0005, sys_metadata, v_par_unified_report_viz
```

ゼロ ETL 統合ターゲットの設定

ゼロ ETL 統合のターゲットを設定する際には、AWS が提供するいくつかのオプションがあります。ターゲットにできるのは、暗号化された Amazon Redshift データウェアハウスまたは Amazon SageMaker Lakehouse カタログです。

ゼロ ETL 統合のターゲットを選択する前に、次のいずれかのターゲットリソースを設定する必要があります。

ゼロ ETL 統合のターゲットの設定オプションは次のとおりです。

- 通常の Amazon S3 ストレージで設定された Amazon SageMaker Lakehouse カタログとデータベース。「通常の S3 ストレージを使用した Amazon SageMaker Lakehouse カタログの設定」を参照してください。
- Amazon S3 Tables バケットで設定された Amazon SageMaker Lakehouse カタログ。「ターゲットとしての Amazon S3 Tables の設定」を参照してください。
- Amazon Redshift マネージドストレージで設定された Amazon SageMaker Lakehouse カタログ。「Amazon Redshift マネージドストレージを使用した Amazon SageMaker Lakehouse カタログの設定」を参照してください。
- Redshift 名前空間によって識別される Amazon Redshift データウェアハウス。「Amazon Redshift データウェアハウスターゲットの設定」を参照してください。

> **Note**
>
> 作成後にゼロ ETL 統合のターゲットを変更することはできません。

通常の S3 ストレージを使用した Amazon SageMaker Lakehouse カタログの設定

このセクションでは、ゼロ ETL 統合で Amazon SageMaker Lakehouse カタログターゲットのストレージとして、通常の Amazon S3 バケットを設定するための前提条件とセットアップ手順について説明します。

統合を設定するための前提条件

通常の S3 ストレージを使用して Amazon SageMaker Lakehouse カタログとのゼロ ETL 統合を作成する前に、次のセットアップタスクを完了する必要があります。

1. AWS Glue データベースをセットアップする
2. カタログ RBAC ポリシーを指定する
3. ターゲット IAM ロールを作成する

通常の Amazon S3 ストレージで Amazon SageMaker Lakehouse カタログを設定したら、ターゲットとの統合の設定 に進んで統合セットアップを完了できます。

ターゲットとしての Amazon S3 Tables の設定

このセクションでは、ゼロ ETL 統合のターゲットとして Amazon S3 Tables を設定するための前提条件とセットアップ手順を説明します。

統合を設定するための前提条件

Amazon S3 Tables をターゲットとするゼロ ETL 統合を作成する前に、次のセットアップタスクを完了する必要があります。

1. Amazon S3 Tables バケットのセットアップ
2. カタログ RBAC ポリシーを指定する
3. ターゲット IAM ロールを作成する

Amazon S3 Tables バケットのセットアップ

1. 「Getting started with Amazon S3 Tables」の手順に従って、アカウントに S3 テーブルバケットを作成します。
2. 「Integrating AWS services with Amazon S3 Tables」の手順に従って、S3-Table バケットと Analytics の統合を有効にします。

カタログ RBAC ポリシーを指定する

ソースと Amazon S3 Tables カタログターゲット間の統合を可能にするには、カタログ RBAC ポリシーに次のアクセス許可を追加する必要があります。

ターゲット AWS Glue Catalog リソースポリシーには、AuthorizeInboundIntegration への Glue サービスのアクセス許可を含める必要があります。さらに、CreateInboundIntegration アクセス許可が、統合を作成するソースプリンシパルまたはターゲット AWS Glue リソースポリシーで必要になります。

> **Note**
>
> クロスアカウントシナリオでは、ソースプリンシパルとターゲット AWS Glue カタログリソースポリシーの両方に、リソースに対する glue:CreateInboundIntegration アクセス許可を含める必要があります。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
    // Optional for same account but mandatory for cross account scenarios
    // Allow Alice to create Integration on Target Catalog
      "Principal": {
        "AWS": [
          "arn:aws:iam::<source-account-id>:user/Alice"
        ]
      },
      "Effect": "Allow",
      "Action": [
        "glue:CreateInboundIntegration"
      ],
      "Resource": [
        "arn:aws:glue:<region>:<Target-Account-Id>:catalog/<s3tablescatalog>/*"
      ],
      "Condition": {
        "StringLike": {
          "aws:SourceArn": "arn:aws:dynamodb:<region>:<Account>:table/<table-name>"
        }
      }
    },
    { // Required: Allow Glue to Authorize the Inbound Integration on behalf of Bob
      "Principal": {
        "Service": [
          "glue.amazonaws.com"
        ]
      },
      "Effect": "Allow",
      "Action": [
        "glue:AuthorizeInboundIntegration"
      ],
      "Resource": [
        "arn:aws:glue:<region>:<Target-Account-Id>:catalog/<s3tablescatalog>/*"
      ],
      "Condition": {
        "StringEquals": {
          "aws:SourceArn": "arn:aws:dynamodb:<region>:<account-id>:table/<table-name>"
        }
      }
    }
```

```
  ]
}
```

> **Note**
>
> <s3tablescatalog> を S3 テーブルのカタログ名に置き換えます。

ターゲット IAM ロールを作成する

以下の許可と信頼関係を持つターゲット IAM ロールを作成します。

IAM ポリシーの例:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "s3tables:ListTableBuckets",
                "s3tables:GetTableBucket",
                "s3tables:GetTableBucketEncryption",
                "s3tables:GetNamespace",
                "s3tables:CreateNamespace",
                "s3tables:ListNamespaces",
                "s3tables:CreateTable",
                "s3tables:GetTable",
                "s3tables:GetTableEncryption",
                "s3tables:ListTables",
                "s3tables:GetTableMetadataLocation",
                "s3tables:UpdateTableMetadataLocation",
                "s3tables:GetTableData",
                "s3tables:PutTableData"
            ],
            "Resource": "arn:aws:s3tables:<region>:<account-id>:bucket/*",
            "Effect": "Allow"
        },
        {
            "Action": [
                "cloudwatch:PutMetricData"
            ],
            "Resource": "*",
```

```
            "Condition": {
                "StringEquals": {
                    "cloudwatch:namespace": "AWS/Glue/ZeroETL"
                }
            },
            "Effect": "Allow"
        },
        {
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": "*",
            "Effect": "Allow"
        }
    ]
}
```

AWS Glue サービスがターゲット IAM ロールを引き受けることができるようにするには、次の信頼ポリシーを追加します。

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

> **Note**
>
> S3-Tables バケットリソースポリシーに、このターゲット IAM ロールの明示的な DENY ステートメントがないことを確認します。明示的な DENY があると、任意の ALLOW 許可が上書きされ、統合が正しく機能しなくなります。

Amazon Redshift マネージドストレージを使用した Amazon SageMaker Lakehouse カタログの設定

このセクションでは、ゼロ ETL 統合のターゲットとして、Amazon Redshift マネージドストレージ (RMS) を使用した Amazon SageMaker Lakehouse カタログを設定するための前提条件とセットアップ手順について説明します。

統合を設定するための前提条件

Redshift マネージドストレージを使用して Amazon SageMaker Lakehouse カタログとのゼロ ETL 統合を作成する前に、次のセットアップタスクを完了する必要があります。

1. Amazon Redshift クラスターまたは Serverless ワークグループをセットアップする
2. Lake Formation と Amazon Redshift の統合を登録する
3. Lake Formation でマネージドカタログを作成する
4. IAM 許可を設定する

Amazon Redshift マネージドストレージのセットアップ

ゼロ ETL 統合用の Amazon Redshift マネージドストレージのセットアップをするには、以下を行います。

1. 既存の Amazon Redshift クラスターまたは Serverless ワークグループを使用するか、新たに作成します。正常に統合できるように、ターゲットの Amazon Redshift ワークグループまたはクラスターで `enable_case_sensitive_identifier` パラメータが有効になっていることを確認します。大文字と小文字の区別を有効にする方法の詳細については、Amazon Redshift 管理ガイドの「データウェアハウスの大文字と小文字の区別を有効にする」を参照してください。
2. Redshift から AWS Lake Formation のカタログに統合を登録します。「Registering Amazon Redshift clusters and namespaces to the AWS Glue Data Catalog」を参照してください。
3. AWS Lake Formation でフェデレーションカタログまたはマネージドカタログを作成します。詳細については、以下を参照してください。
   - Bringing Amazon Redshift data into the AWS Glue Data Catalog
   - Creating an Amazon Redshift managed catalog in the AWS Glue Data Catalog
4. ターゲットロールの IAM アクセス許可を設定します。ロールには、Redshift リソースと Lake Formation リソースの両方にアクセスするためのアクセス許可が必要です。ロールに最低限必要となるアクセス許可は以下のとおりです。

- Redshift クラスターまたはワークグループへのアクセス許可
- Lake Formation カタログへのアクセス許可
- カタログでテーブルを作成および管理するためのアクセス許可
- モニタリングのための CloudWatch および CloudWatch Logs のアクセス許可

Amazon Redshift マネージドストレージを使用して Amazon SageMaker Lakehouse カタログを設定したら、ターゲットとの統合の設定 に進んで統合セットアップを完了できます。

Amazon Redshift データウェアハウスターゲットの設定

このセクションでは、ゼロ ETL 統合のターゲットとして Amazon Redshift データウェアハウスを設定するための前提条件とセットアップ手順について説明します。

統合を設定するための前提条件

Amazon Redshift データウェアハウスターゲットとのゼロ ETL 統合を作成する前に、次のセットアップタスクを完了する必要があります。

1. Amazon Redshift クラスターまたは Serverless ワークグループをセットアップする
2. 大文字と小文字の区別を設定する
3. IAM 許可を設定する

Amazon Redshift データウェアハウスのセットアップ

ゼロ ETL 統合用の Amazon Redshift データウェアハウスを設定するには、次を行います。

1. [Amazon Redshift コンソール] に移動し、[クラスターの作成] をクリックするか、既存のクラスターを使用します。Amazon Redshift Serverless の場合、[ワークグループを作成] をクリックします。
2. 新しいクラスターを作成する場合は、適切なクラスターサイズを選択し、クラスターが暗号化されていることを確認します。Serverless の場合は、要件に応じてワークグループ設定を構成します。
3. 正常に統合するため、ターゲットの Amazon Redshift ワークグループまたはクラスターで `enable_case_sensitive_identifier` パラメータが有効になっていることを確認します。大文字と小文字の区別を有効にする方法の詳細については、Amazon Redshift 管理ガイドの「データウェアハウスの大文字と小文字の区別を有効にする」を参照してください。

4. ゼロ ETL 統合が Amazon Redshift データウェアハウスにアクセスできるように IAM アクセス許可を設定します。以下のアクセス許可を持つ IAM ロールを作成する必要があります。
   - Amazon Redshift クラスターまたはワークグループへのアクセス許可
   - Amazon Redshift でデータベースとテーブルを作成および管理するためのアクセス許可
   - モニタリングのための CloudWatch およびAmazon CloudWatch Logs のアクセス許可
5. Amazon Redshift ワークグループまたはクラスターのセットアップが完了したら、ゼロ ETL 統合のためにデータウェアハウスを設定する必要があります。詳細については、Amazon Redshift 管理ガイドの「ゼロ ETL 統合の開始方法」を参照してください。

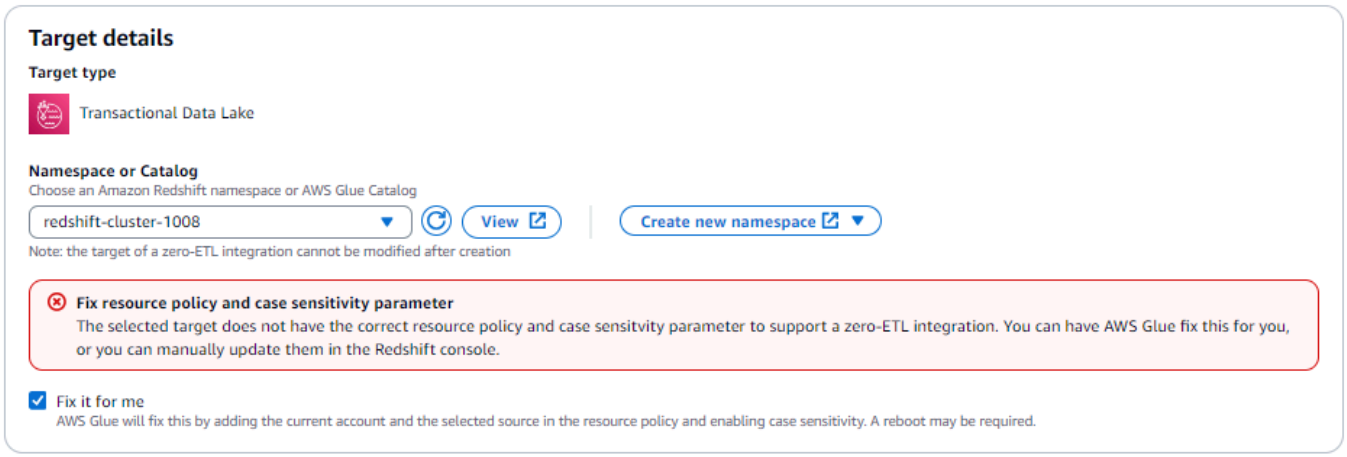
> **Note**
>
> Amazon Redshift データウェアハウスをターゲットとして使用すると、統合により、指定されたデータベースにスキーマが作成され、レプリケートされたデータが保存されます。スキーマ名は統合名から導出されます。

Amazon Redshift データウェアハウスを設定したら、ターゲットとの統合の設定 に進み、統合セットアップを完了できます。

ターゲットとの統合の設定

ターゲットリソースを設定し、接続を選択し、ソース IAM ロールを指定したら、次の手順に従って統合設定を完了します。

1. 前のステップで設定したターゲットを指定します。
2. AWS Glue [修正をお願いする] オプションを選択します。Amazon Redshift ターゲットの場合、これは次のようになります。
   - Amazon Redshift クラスターまたは Serverless ワークグループで承認されたサービスプリンシパルを適用します。
   - 承認された AWS Glue ソース ARN を Amazon Redshift クラスターまたは Serverless ワークグループに適用します。
   - 新しいパラメータグループを `enable_case_sensitive_identifier = true` に関連付けます。



3. 統合名を指定し、[統合の作成と起動] を選択します。
4. 統合がアクティブ状態になったら、統合の詳細ページに移動し、[統合からデータベースを作成] を選択します。
5. 最後に、Redshift クエリエディタに移動し、データベースに接続してスナップショットと増分データを検証できます。

> **Note**
>
> 名前空間またはカタログ名には、小文字の英数字とアンダースコアのみを使用できます。これは、AWS Glue データカタログで任意の名前 (特殊文字を含む) を使用してデータベースを作成できることとは異なります。

パーティション仕様とスキーマのネスト解除ガイド

DynamoDB や SaaS アプリケーションなどの NoSQL データソースを使用する場合、データから次のような分析に固有の課題が見えてくることがあります。

1. 同じテーブル内のレコードのスキーマが異なる場合がある
2. 同じテーブル内のネストされたレコードは、異なる方法で表現できる
3. マップや配列などの複雑なネスト構造には、効率的なクエリを実行するために変換が必要
4. 大規模なクエリパフォーマンスを確保するには、最適なデータ整理が必要

AWS Glue ゼロ ETL 統合は、次の 2 つの優れた機能を通じてこれらの課題に対処します。

- スキーマのネスト解除: 複雑なネストされたデータ構造を分析しやすい形式に自動的にフラット化します。ネスト解除のレベルを設定して、データ構造の保持とクエリの簡素化による最適化とのバランスを取ります。
- データパーティショニング: 指定された列または時間ベースのディメンションに基づいてデータを論理パーティションに整理し、クエリの実行中にパーティションプルーニングを有効にすることでクエリのパフォーマンスを向上させ、コストを削減します。

このようなデータソースを効果的にクエリするために、AWS Glue ゼロ ETL では、ターゲット AWS Glue データベースでレプリケートされるソースデータのスキーマ処理とパーティショニングスキームがすぐに利用できる状態で提供されます。CreateIntegrationTableProperty API を使用して、各テーブルのスキーマのネスト解除とパーティショニングを設定できます。これにより、分析ワークロードのデータをどのように構造し、整理するかをファインチューニングできます。

デフォルトのネスト解除とパーティショニングの動作

1. ターゲットテーブルにネスト解除オプションが指定されていない場合、AWS Glue ゼロ ETL はデフォルトの完全なネスト解除になります
2. ターゲットテーブルに PartitionSpec が指定されていない場合、AWS Glue ゼロ ETL はデフォルトのバケットパーティショニングになります

スキーマのネスト解除

ゼロ ETL を介して分析サービスと統合する場合、ネストされた構造をターゲットテーブルでどのように表現するかを選択できます。AWSGlue Zero-ETL には、複雑なデータ構造をより分析しやすい形式にフラット化するスキーマのネスト解除オプションが用意されています。

ネスト解除オプション

ソースとのゼロ ETL 統合を作成するときは、次のネスト解除オプションから選択できます。これらのオプションは、CreateIntegrationTableProperty API を呼び出すときに使用する特定の列挙値に対応しています。

ネスト解除なし (デフォルト)

API 値:**`NO_UNNEST`**

DynamoDB 項目の元のネストされた構造を保存します。マップとリストは、ターゲットに構造化列として保存されます。

最適なケース: 分析ツールでネストされたデータを操作できる場合に、DynamoDB データの正確な構造を保持します。

ネストを 1 レベル解除する

API 値:**`TOP_LEVEL`**

ネストされたマップの最上位レベルを個々の列にフラット化します。リスト構造はネストされたままです。

最適なケース: DynamoDB 項目に一貫したスキーマがある場合に、データ構造の保持とクエリの簡素化のバランスを取ります。

すべてのレベルのネストを解除する

API 値:**`FULL`**

ネストされたすべての構造 (マップとリスト) を再帰的にフラット化し、命名用のドット表記を使用して個々の列にします。

最適なケース: 深くネストされた構造とフラットなスキーマを優先する分析ツールを使用する場合に、クエリを最大限に簡素化します。

> **Note**
>
> 完全なネスト解除によって、DynamoDB データに可変構造または深くネストされた構造がある場合、多数の列を持つ非常に幅の広いテーブルになる可能性があります。

Example API でのネスト解除オプションの使用

CreateIntegrationTableProperty API を使用してスキーマのネスト解除を設定する場合は、UnnestSpec パラメータでネスト解除オプションを指定します。

```
aws glue create-integration-table-property \
  --resource-arn "arn:aws:glue:us-east-1:123456789012:integration/my-integration" \
  --table-name "my-table" \
  --table-properties '{
    "unnestSpec": "FULL"
  }'
```

ネスト解除の例

次の構造を持つ DynamoDB 項目を考えてみましょう。

```
{
  "ProductId": "P12345",
  "ProductDetails": {
    "Name": "Smartphone",
    "Brand": "TechCo",
    "Specifications": {
      "Color": "Black",
      "Storage": "128GB"
    }
  },
  "Reviews": [
    {
      "Rating": 5,
      "Comment": "Great product!"
    },
    {
      "Rating": 4,
      "Comment": "Good value."
    }
  ]
}
```

ネスト解除なしの例

ネストを解除しない場合、ターゲットテーブルには以下の列が含まれます。

- ProductId (文字列)
- ProductDetails (構造体)
- Reviews (構造体の配列)

クエリでは、構造体と配列のアクセスパターンを使用する必要があります。

```
SELECT
  ProductId,
  ProductDetails.Name,
  ProductDetails.Specifications.Color
FROM product_table;
```

ネストを 1 レベル解除する例

ネストを 1 レベル解除する場合、ターゲットテーブルには以下の列が含まれます。

- ProductId (文字列)
- ProductDetails_Name (文字列)
- ProductDetails_Brand (文字列)
- ProductDetails_Specifications (構造体)
- Reviews (構造体の配列)

クエリは第 1 レベルで簡略化されます。

```
SELECT
  ProductId,
  ProductDetails_Name,
  ProductDetails_Specifications.Color
FROM product_table;
```

すべてのレベルのネストを解除する例

すべてのレベルのネストを解除すると、ターゲットテーブルには以下の列と値が含まれます。

列名 (タイプ)	値
ProductId (文字列)	P12345
ProductDetails_Name (文字列)	Smartphone
ProductDetails_Brand (文字列)	TechCo
ProductDetails_Specifications_Color (文字列)	Black
ProductDetails_Specifications_Storage (文字列)	128GB
Reviews_0_Rating (数値)	5
Reviews_0_Comment (文字列)	Great product!
Reviews_1_Rating (数値)	4

列名 (タイプ)	値
Reviews_1_Comment (文字列)	Good value.

クエリは次のように完全にフラット化されます。

```
SELECT
  ProductId,
  ProductDetails_Name,
  ProductDetails_Specifications_Color
FROM product_table;
```

ゼロ ETL 統合のデータパーティショニング

データパーティショニングとは

データパーティショニングは、大きなデータセットをパーティションと呼ばれるより小さく管理しやすいセグメントに分割する手法です。AWS Glue ゼロ ETL 統合のコンテキストでは、パーティション分割は、特定の列の値またはそれらの値の変換に基づいて、ターゲット位置にデータを整理します。

データパーティショニングの利点

効果的なデータパーティショニングを分析ワークロードで使用すると、主に次のような利点があります。

- クエリパフォーマンスの向上: クエリは無関係なパーティションをスキップ (パーティションプルーニング) するため、スキャンする必要があるデータの量を減らすことができます。
- コストの削減: スキャンするデータを減らすことで、分析クエリのコンピューティングコストと I/O コストを低減できます。
- スケーラビリティの向上: パーティション分割によってデータセグメントの並列処理が可能になり、分析ワークロードをより効率的にスケーリングできます。
- データライフサイクル管理の簡素化: 保持ポリシーをパーティションレベルで管理できるため、古いデータのアーカイブや削除が容易になります。

主要なパーティショニングの概念

パーティション列

レコードがどのようにパーティションに編成されるかを決定するために使用されるデータの列。有効なパーティション列は、一般的なクエリパターンと一致し、適切なカーディナリティを持つ必要があります。

パーティション関数

実際のパーティション境界を作成するためにパーティション列の値に適用される変換。例として、アイデンティティ (raw 値を使用)、時間ベースの関数 (年、月、日、時間)、バケット化などがあります。

パーティションプルーニング

クエリに関連するデータを含まないパーティションをクエリエンジンが識別してスキップするプロセス。これにより、パフォーマンスが大幅に向上します。

パーティションの粒度

データパーティショニングの詳細レベル。より細かい粒度 (より多くのパーティション) にすると、クエリのパフォーマンスは向上しますが、メタデータのオーバーヘッドが増加する可能性があります。より粗い粒度 (より少ないパーティション) にすると、メタデータのオーバーヘッドは軽減されますが、必要以上に多くのデータがスキャンされる可能性があります。

AWS Glue ゼロ ETL 統合でのパーティショニング

AWS Glue ゼロ ETL 統合では、高度なパーティショニング機能を提供する Apache Iceberg テーブル形式を使用します。ゼロ ETL 統合を作成すると、以下を行うことができます。

- データソースに最適化されたデフォルトパーティショニング戦略を使用する
- クエリパターンに応じたカスタムパーティショニング仕様を定義する
- 変換をパーティション列に適用する (特にタイムスタンプベースのパーティショニングに役立つ)
- 複数のパーティション戦略を組み合わせてマルチレベルパーティショニングを行う

パーティショニングの設定は、ゼロ ETL 統合を設定するときに `CreateIntegrationTableProperty` API を通じて指定されます。設定すると、AWS Glue はこれらのパーティショニング戦略を自動的に適用して、ターゲットの場所にデータを整理します。

パーティション仕様の API リファレンス

CreateIntegrationTableProperties API で次のパラメータを使用し、パーティショニングを設定します。

PartitionSpec

ターゲットロケーションでデータがパーティショニングされる法を定義するパーティション仕様の配列。

```
{
  "partitionSpec": [
    {
      "fieldName": "timestamp_col",
      "functionSpec": "month",
      "conversionSpec": "epoch_milli"
    },
    {
      "fieldName": "category",
      "functionSpec": "identity"
    },
    {
      "fieldName": "user_id",
      "functionSpec": "bucket",
      "bucketCount": 16
    },
    {
      "fieldName": "product_code",
      "functionSpec": "truncate",
      "width": 4
    }
  ]
}
```

FieldName

パーティショニングに使用する列名を指定する UTF-8 文字列 (1 ~ 128 バイト)。

FunctionSpec

パーティショニング関数を指定します。有効な値:

- `identity` - 変換なしでソース値を直接使用する

- year - タイムスタンプ値から年を抽出する (例: 2023)
- month - タイムスタンプ値から月を抽出する (例: 2023-01)
- day - タイムスタンプ値から日を抽出する (例: 2023-01-15)
- hour - タイムスタンプ値から時間を抽出する (例: 2023-01-15-14)
- bucket - ハッシュベースのパーティショニングを使用して、指定された数のバケットに値を分散します。bucketCount パラメータが必要です。
- truncate - 文字列値を指定された幅に切り捨てます。width パラメータが必要です。

> **Note**
>
> 時間ベースの関数 (year、month、day、hour) では、ソースタイムスタンプ形式を指定する ConversionSpec パラメータが必要です。bucket と truncate 関数には、説明に指定されている追加のパラメータが必要です。

ConversionSpec

ソースデータのタイムスタンプ形式を指定する UTF-8 文字列。次の値を指定できます:

- epoch_sec – UNIX エポックタイムスタンプ (秒単位)
- epoch_milli – UNIX エポックタイムスタンプ (ミリ秒単位)
- iso – ISO 8601 形式のタイムスタンプ

BucketCount

bucket 関数の使用時に作成するバケットの数を指定する整数。functionSpec を bucket に設定した場合、このパラメータは必須です。値は正の整数である必要があります。ハッシュベースのバケット化によって、指定された数のバケットに値が均等に分散されます。

Example バケットパーティショニングの例

```
{
  "partitionSpec": [
    {
      "fieldName": "user_id",
      "functionSpec": "bucket",
      "bucketCount": 16
    }
  ]
```

```
}
```

[幅]

truncate 関数の使用時に切り捨て幅を指定する整数。functionSpec を truncate に設定した場合、このパラメータは必須です。値は、文字列値の先頭から保持される文字数を表す正の整数である必要があります。

Example 切り捨てパーティショニングの例

```
{
  "partitionSpec": [
    {
      "fieldName": "product_code",
      "functionSpec": "truncate",
      "width": 4
    }
  ]
}
```

パーティショニング戦略

デフォルトのパーティショニング

パーティション列が指定されていない場合、AWS Glue Zero-ETL はデータソースに最適化されたデフォルトのパーティショニング戦略を適用します。

- プライマリキーベースのパーティショニング: プライマリキーを持つソース (DynamoDB テーブルなど) の場合、AWS Glue Zero-ETL はプライマリキーとバケット化によってデータを自動的にパーティショニングし、パーティションの爆発を防ぎます。

デフォルトのパーティショニングは、手動で設定しなくても、一般的なクエリパターンが適切に機能するように設計されています。ただし、特定のクエリパターンやパフォーマンス要件には、カスタムパーティショニング戦略を定義することをお勧めします。

ユーザー定義のパーティショニング戦略

AWS Glue Zero-ETL では、PartitionSpec パラメータを使用してカスタムパーティショニング戦略を定義できます。1 つ以上のパーティション列を指定し、各列に異なるパーティショニング関数を適用できます。

ID パーティショニングは、列の raw 値を使用してパーティションを作成します。この戦略は、カテゴリ、リージョン、ステータスフィールドなど、低から中程度のかーディナリティを持つ列に役立ちます。

Example ID パーティショニングの例

```
{
  "partitionSpec": [
    {
      "fieldName": "category",
      "functionSpec": "identity"
    }
  ]
}
```

これにより、「カテゴリ」列の一意の値ごとに個別のパーティションが作成されます。

> **Warning**
>
> 高カーディナリティ列 (プライマリキーやタイムスタンプなど) では ID パーティショニングを使用しないでください。使用すると、パーティションの爆発が発生して、パフォーマンスが低下し、メタデータのオーバーヘッドが増加する可能性があります。

時間ベースのパーティショニングは、さまざまな粒度 (年、月、日、または時間) のタイムスタンプ値に基づいてデータを整理します。この戦略は時系列データに最適であり、効率的な時間範囲クエリを実行可能にします。

時間ベースのパーティショニングを使用する場合、AWS Glue ゼロ ETL はパーティション関数を適用する前に、さまざまなタイムスタンプ形式を標準化された形式に自動的に変換できます。この変換は、`ConversionSpec` パラメータを使用して指定します。

Example 時間ベースのパーティショニングの例

```
{
  "partitionSpec": [
    {
      "fieldName": "created_at",
      "functionSpec": "month",
      "conversionSpec": "epoch_milli"
    }
```

```
  ]
}
```

この関数は、Unix エポックタイムスタンプをミリ秒単位で含む「created_at」列に基づいてデータを月単位でパーティショニングします。

AWS Glue ゼロ ETL は、次の時間ベースのパーティション関数をサポートしています。

- year: 年別にデータをパーティショニング (例: 2023、2024)
- month: データを月別にパーティショニング (例: 2023-01、2023-02)
- day: データを日別にパーティショニング (例: 2023-01-01、2023-01-02)
- hour: データを時間単位でパーティショニング (例: 2023-01-01-01、2023-01-01-02)

AWS Glue ゼロ ETL は、`ConversionSpec` パラメータで以下のタイムスタンプ形式をサポートしています。

- epoch_sec: 秒単位の Unix エポックタイムスタンプ
- epoch_milli: ミリ秒単位の Unix エポックタイムスタンプ
- iso: ISO 8601 形式のタイムスタンプ

> **Note**
>
> 元の列値はソースデータ内で変更されません。AWSGlue はターゲットデータベースのテーブルで、パーティション列の値のみをタイムスタンプタイプに変換します。変換は、パーティショニングプロセスのみに適用されます。

バケットパーティショニングでは、ハッシュベースのディストリビューションを使用して、固定数のバケットにデータを均等に分散します。この戦略は、パーティションの数を制御してデータを均等に分散する高カーディナリティ列に役立ちます。

Example バケットパーティショニングの例

```
{
  "partitionSpec": [
    {
      "fieldName": "user_id",
```

```
      "functionSpec": "bucket",
      "bucketCount": 16
    }
  ]
}
```

これにより、「user_id」列のハッシュ値に基づいて 16 個のバケットが作成され、実際のユーザー ID 値に関係なく均等に分散されます。

バケットパーティショニングは、以下に対して特に効果的です。

- ユーザー ID、セッション ID、プライマリキーなどの高カーディナリティ列
- 予測可能なパーティション数が必要なシナリオ
- 固定パーティション境界全体で並列処理が有益なワークロード

切り捨てパーティショニングは、切り捨てられた文字列値に基づいてパーティションを作成し、最初の N 文字のみを保持します。この戦略は、類似した値をグループ化する文字列の列に役立ちます。

Example 切り捨てパーティショニングの例

```
{
  "partitionSpec": [
    {
      "fieldName": "product_code",
      "functionSpec": "truncate",
      "width": 4
    }
  ]
}
```

これにより、「product_code」列の最初の 4 文字に基づいてパーティションが作成されます。たとえば、「ELEC12345」と「ELEC67890」は、同じ「ELEC」パーティションにグループ化されます。

切り捨てパーティショニングは、以下に対して特に効果的です。

- 製品コード、SKU、または意味のあるプレフィックスを持つその他の構造化識別子
- 地域グループ化が有益な地域コードまたは郵便番号
- プレフィックスベースのグループ化がクエリパターンと一致する文字列の列

マルチレベルパーティショニングは、複数のパーティション戦略を組み合わせて階層パーティショニングスキームを作成します。これは、同じデータセットに対してさまざまなタイプのクエリを最適化するのに役立ちます。

Example マルチレベルパーティショニングの例

```
{
  "partitionSpec": [
    {
      "fieldName": "created_at",
      "functionSpec": "month",
      "conversionSpec": "iso"
    },
    {
      "fieldName": "region",
      "functionSpec": "identity"
    },
    {
      "fieldName": "user_id",
      "functionSpec": "bucket",
      "bucketCount": 8
    }
  ]
}
```

この関数は、3 レベルのパーティショニングスキームを作成します。最初は月ごと (「created_at」列から)、次にリージョンごと、最後にユーザー ID バケットごとに作成します。これにより、日付範囲、特定の地域、ユーザーセグメント、またはこれらのディメンションの任意の組み合わせでフィルタリングする効率的なクエリが実行可能になります。

マルチレベルパーティショニングスキームを設計する際は、次の点を考慮してください。

- パーティション階層の最初に選択性の高い列を配置する
- パーティションの粒度とパーティション数のバランスを取る
- パーティション分割スキームを最も一般的なクエリパターンに合わせる

ベストプラクティス

パーティション列の選択

- identity パーティション関数を持つ高カーディナリティ列を使用しないでください。ID パーティショニングを持つ高カーディナリティ列を使用すると、小さなパーティションが多く作成され、取り込みパフォーマンスが大幅に低下する可能性があります。高カーディナリティ列には次のものが含まれる場合があります。
  - プライマリキー
  - タイムスタンプフィールド (LastModifiedTimestamp や CreatedDate など)
  - システム生成されたタイムスタンプ
- 同じ列で複数のタイムスタンプパーティションを選択しないでください。例:

```
"partitionSpec": [
      {"fieldName": "col1", "functionSpec": "year", "conversionSpec" :
 "epoch_milli"},
      {"fieldName": "col1", "functionSpec": "month", "conversionSpec" :
 "epoch_milli"},
      {"fieldName": "col1", "functionSpec": "day", "conversionSpec" : "epoch_milli"},
      {"fieldName": "col1", "functionSpec": "hour", "conversionSpec" : "epoch_milli"}
]
```

パーティション FunctionSpec/ConversionSpec の選択

- タイムスタンプベースのパーティション関数を使用する際に、タイムスタンプベースのパーティショニングのために選択された列値の形式を表す正しい ConversionSpec (epoch_sec | epoch_milli | iso) を指定します。AWSGlue ゼロ ETL はこのパラメータを使用し、パーティショニングする前にソースデータをタイムスタンプ形式に正しく変換します。
- データ量に基づいて適切な粒度 (年/月/日/時間) を使用します。
- ISO タイムスタンプ を使用する際、タイムゾーンの影響を考慮してください。AWSGlue ゼロ ETL は、UTC タイムゾーンで選択したタイムスタンプ列のすべてのレコード値を入力します。
- bucket パーティショニングを使用する場合は、クエリ並列処理の要件に合ったバケット数を選択します。一般的なバケット数は 2 の累乗 (8、16、32) であり、最適なハッシュ分散が可能になります。

- `truncate` パーティショニングでは、データパターンに基づいて意味のあるグループ化を作成する幅の値を選択します。切り捨て幅が文字列値の重要なプレフィックス部分をキャプチャしていることを確認します。
- `bucket` または `truncate` 関数を非常に小さなデータセットの唯一のパーティション戦略として使用しないでください。不要なパーティションオーバーヘッドが発生する可能性があります。

エラー処理

NEEDS_ATTENTION 状態

統合は次の場合に NEEDS_ATTENTION 状態になります。

- 指定されたパーティション列がソースに存在しない場合
- パーティション列のタイムスタンプ変換が失敗した場合

制限

パーティショニングの制限事項

- 統合を作成した後にパーティションの仕様を変更することはできません。別のパーティショニング戦略を使用するには、新しい統合を作成する必要があります。
- パーティション列の最大数は 10 に制限されています。

クロスアカウント統合の制限事項

- クロスアカウント統合を作成する場合、AWS Glue コンソールでは、統合が存在しないアカウントでホストされているターゲット AWS Glue テーブルに対して UnnestSpec と PartitionSpec を設定する CreateIntegrationTableProperty API を呼び出すことはできません。

  回避方法: CreateIntegrationTableProperty API は、ターゲットデータベースが存在するアカウントから CX が呼び出す必要があります。

複数の統合の制限事項

- 異なるスキーマのネスト解除/パーティション設定で同じソースをレプリケートする必要がある場合は、統合ごとに新しい AWS Glue データベースを別々に作成する必要があります。後で、目的

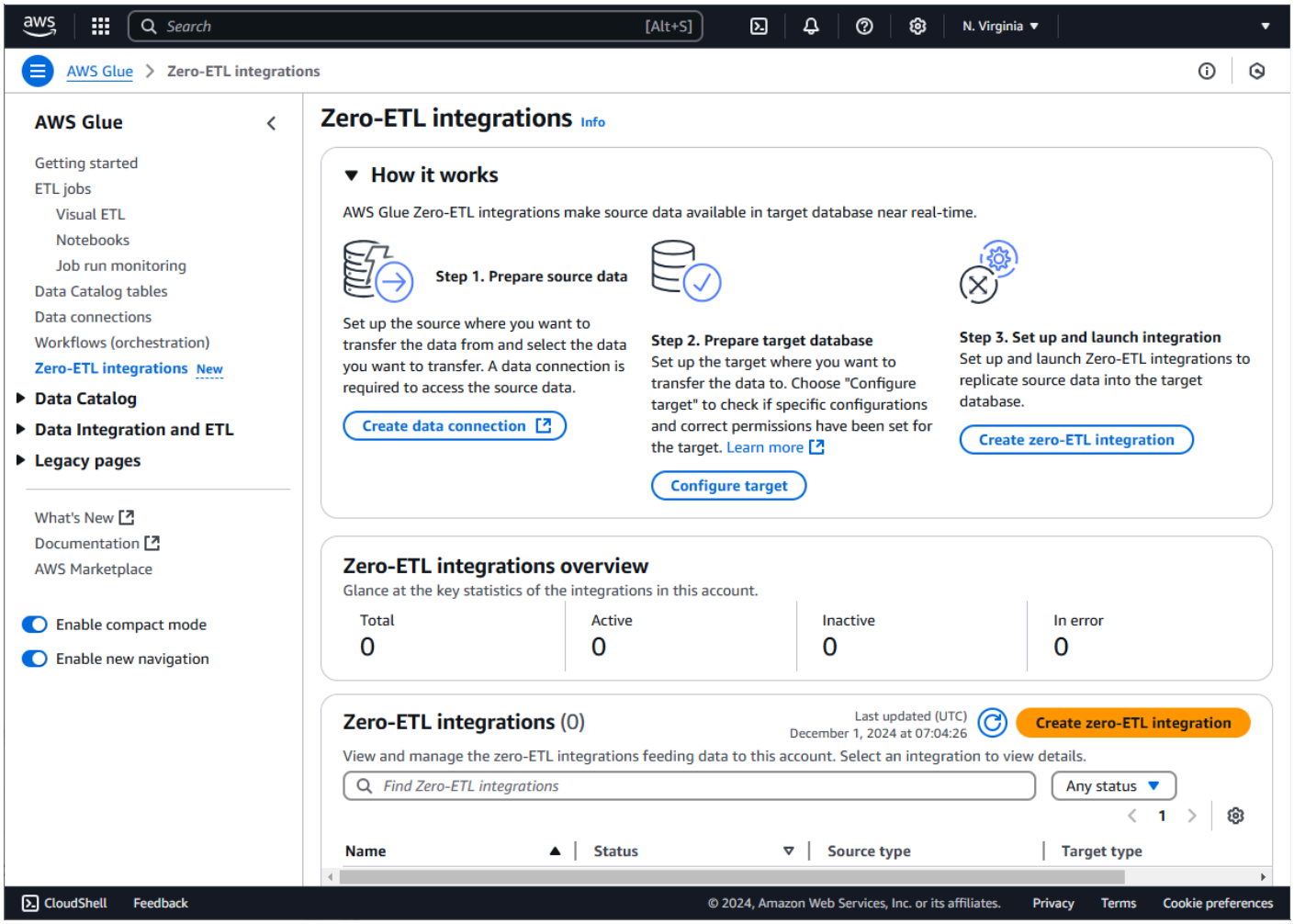
のスキーマのネスト解除とパーティショニング設定を使用して、個別の AWS Glue データベースからテーブルごとに CreateIntegrationTableProperty を呼び出します。

一般的な統合タスク

統合の作成

このセクションでは、統合を作成するための一般的なステップについて説明します。この例では、Amazon DynamoDB をソースとして使用します。

1. AWS Glue コンソールのホームページで、[ゼロ ETL 統合] を選択します。
2. [ゼロ ETL 統合] ホームページですべての統合を表示できます。新しい統合を作成するには、[ゼロ ETL 統合の作成] を選択します。



3. [ソースタイプ] を選択するように求められます。ソースを選択し、[次へ] をクリックします。SaaS 統合ソースのソース設定セクションを参照してください。
4. [ソースとターゲットの設定] ページで、レプリケートするテーブルまたはエンティティを選択します。Amazon DynamoDB の場合は、PITR および RBAC ポリシーが設定されていることを確認します。
5. 統合ターゲットを指定します。
   - AWS Glue データカタログターゲットの場合は、データをレプリケートする AWS Glue データベースを選択します。
   - Amazon Redshift データウェアハウスターゲットの場合は、Redshift クラスター名前空間または Redshift Serverless ワークグループ名前空間を選択します。

   詳細については、「ターゲットとの統合の設定」を参照してください。
6. 前提条件で作成したターゲット IAM ロールを指定します。
7. ターゲットに保存されているデータに対して、任意のターゲット KMS キーを設定する場合は、有効な KMS キーを指定します。同様に、ターゲットネットワーク接続を設定する場合は、AWS Glue 接続を選択します。
8. [ターゲットの固定] ボタンは、このドキュメントの「前提条件」セクションのステップの一部を設定します。つまり、1) カタログ RBAC ポリシーを提供し、2) Amazon S3 URI が指定されていない場合は生成され、それ以外の場合は指定された URI が使用されます。
9. Redshift データウェアハウスターゲットとの統合の場合:
10. [ソースとターゲットの設定] ページの [出力設定] セクションで、ターゲット内のデータに使用するスキーマのネスト解除オプションを選択します。データにカスタマーパーティションキーを使用する場合は、[カスタマーパーティションキーの指定] を選択し、最大 10 個のキーを指定します。それ以外の場合は、レプリケートされる DynamoDB テーブルに割り当てられたパーティションキーを使用できます。
11. [セキュリティとデータの暗号化] セクションでは、データをターゲットにレプリケートする中間プロセスで使用される KMS キーを指定できます。それ以外の場合は、AWS マネージド KMS キーが使用されます。現在、15 分のレプリケーション設定のみをサポートしています。[統合の詳細] にゼロ ETL 統合の名前を入力します。
12. 提供されたすべての詳細が正しいことを確認します。すべてが確認されたら、[統合の作成と起動] をクリックします。
13. ゼロ ETL ホームページで、作成した統合を選択すると、統合の詳細が表示されます。「ステータス」は、統合の状態を示します。

統合の変更

既存の統合を変更できます。

1. 統合の詳細ページの右上隅にある [編集] を選択します。
2. [ソースとターゲットの編集] ページで、ターゲット IAM ロールとターゲットネットワーク接続を変更できます。他のフィールドは、統合の作成後に編集できません。[次へ] をクリックします。
3. 統合の名前と説明は、[統合と設定の編集] ページで編集することもできます。[次へ] をクリックします。
4. 編集内容を確認し、確認したら [統合の更新] をクリックします。

統合の削除

削除は統合の使用を終了する状態です。一度削除すると、統合を元に戻すことはできません。統合を削除すると、すべての内部メタデータと保存された中間データが消去されます。

このプロセス中に、ターゲットテーブルにデータを書き込んでいる実行中のタスクはすべて終了します。AWS Glue は、ターゲット AWS Glue データベース (データカタログ内) およびアカウントの Amazon S3 バケット内の関連データを削除またはクリーンアップしません。必要に応じて、明示的にクリーンアップする必要があります。

統合を削除するには:

1. 統合の詳細ページで、[削除] をクリックします。
2. 「削除」と入力し、[削除] をクリックします。注: これは元に戻せないアクションです。
3. 統合の詳細ページで、ステータスに「削除中」と表示されます。統合が実際に削除されると、ゼロ ETL 統合ホームページに表示されなくなります。

API を使用した統合の作成

次の API を使用して、AWS Glue でゼロ ETL 統合を作成および管理できます。

- CreateIntegration
- CreateIntegrationTableProperties
- CreateIntegrationResourceProperty
- UpdateIntegrationTableProperties

- UpdateIntegrationResourceProperty
- ModifyingIntegration
- DeleteIntegration
- DeleteIntegrationTableProperties
- DescribeIntegrations
- DescribeInboundIntegrations
- GetIntegrationTableProperties
- GetIntegrationResourceProperty

詳細については、「AWS Glue の統合 API」を参照してください。

統合のモニタリング

統合状態

次の統合状態は、統合の説明を示しています。

- `Creating` — 統合を作成中です。
- `Active` — 統合がトランザクションデータをターゲットに送信中です。
- `Modifying` — 統合を変更中です。
- `Syncing` — 統合で復旧可能なエラーが発生したため、データを再シードしています。
- `Needs attention` — 統合でイベントまたはエラーが発生したため、解決するには手動介入が必要です。問題を解決するには、統合詳細のエラーメッセージの指示に従ってください。
- `Failed` — 統合で、復旧不能なイベントまたはエラーが発生しました。統合を手動で削除して再作成する必要があります。
- `Deleting` — 統合を削除中です。

統合に関する Amazon CloudWatch Logs の表示

AWS Glue ゼロ ETL 統合は、データ移動を可視化するために Amazon CloudWatch Logs を生成します。それぞれの正常な取り込み、ソースでの問題のあるデータレコードが原因で発生した障害、また

はスキーマの変更やアクセス許可の不足によるデータ書き込みエラーに関連するログイベントは、カスタマーアカウントで作成されたデフォルトのロググループに出力されます。

作成された各統合で、その統合のログイベントは Amazon CloudWatch の `/aws-glue/zeroETL-integrations/logs/` に収集されます。ロググループ内のログメッセージは、ログストリームに分割されます。作成された各統合には、その統合のすべてのログが書き込まれる専用のログストリームがあります。例えば、IntegrationArn `arn:aws:glue:us-east-1:123456789012:integration:03cabe77-79e7-4b7a-b3da-8c160bea6bbf` の統合のログは、/aws-glue/zeroETL-integrations/logs/03cabe77-79e7-4b7a-b3da-8c160bea6bbf にあります。{IntegrationId} は、統合の作成時に生成された {integrationArn} から参照できます。

> **Note**
>
> クロスアカウントシナリオでは、ソース処理ログは統合が存在するソースアカウントで出力され、ターゲット処理ログはターゲットデータベースが存在するターゲットアカウントで出力されます。

ログ記録を有効にするために必要な IAM 許可

統合を作成する際、統合の CloudWatch ログ記録を有効にするには、ソースロールとターゲットロールに次の IAM 許可が必要です。AWS Glue ゼロ ETL 統合では、ソースロールとターゲットロールで提供されるこれらの許可を使用して、CloudWatch ログをカスタマーアカウントに出力します。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
"Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents",
             ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

ログメッセージ

ログ形式: ゼロ ETL 統合では、次の 4 種類のログメッセージを出力します。

```
// Ingestion started
{
"integrationArn": "arn:aws:glue:us-east-2:123456789012:integration/1a012bba-123a-1bba-
ab1c-173de3b12345",
...
    "messageType": "IngestionStarted",
    "details": {
        "tableName": "testDDBTable",
        "message": "Ingestion Job started"
    }
}
// Data processing stats on successful table ingestion
{
...
    "messageType": "IngestionProcessingStats",
    "details": {
        "tableName": "testDDBTable",
        "insert_count": 100,
        "update_count": 10,
        "delete_count": 10
    }
}
// Ingestion failure logs for failed table-processing
{
...
    "messageType": "IngestionFailed",
    "details": {
        "tableName": "testDDBTable",
        "errorMessage": "Failed to ingest data with error: Target Glue database not
 found.",
        "error_code" : "client_error"
    }
}
// Ingestion completed notification with lastSyncedTimestamp
{
...
    "messageType": "IngestionCompleted",
    "details": {
        "tableName": "testDDBTable",
        "message": "Ingestion Job completed"
```

```
        "lastSyncedTimestamp": "1132344255745"
    }
}
```

統合に関する Amazon CloudWatch メトリクスの表示

統合が完了すると、AWS Glue ジョブ実行ごとにアカウントで生成された以下の Amazon Cloudwatch メトリクスが表示されます。

CloudWatch メトリクス名前空間: "AWS/Glue/ZeroETL"

メトリクスのディメンション:

- `integrationArn`
- `loadType`
- `tableName`

メトリクス名:

- `InsertCount` - ターゲット Iceberg テーブルに挿入されたレコードの数。
- `UpdateCount` - ターゲット Iceberg テーブルで更新されたレコードの数。
- `DeleteCount` - ターゲット Iceberg テーブルから削除されたレコードの数。
- `IngestionSucceeded` - 統合の取り込みが成功した場合、1 をカウントします。
- `IngestionFailed` - 統合の取り込みに失敗した場合、1 をカウントします。
- `LastSyncTimestamp` - ソースがターゲットに同期されるまでのタイムスタンプ。

Amazon EventBridge によるイベント通知の管理

ゼロ ETL 統合では、Amazon EventBridge を使用してイベント通知を管理し、統合の変更に関する最新情報を伝えます。Amazon EventBridge は、アプリケーションをさまざまなソースからのデータに接続するために使用できるサーバーレスのイベントバスサービスです。この場合、イベントソースは AWS Glue です。環境内の監視された変更であるイベントは、AWS Glue から EventBridge に自動的に送信されます。イベントは、ほぼリアルタイムに配信されます。

EventBridge は、イベントルールを記述するための環境を用意します。ルールでは、特定のイベントに対して実行するアクションを指定できます。ターゲットを設定することもできます。ターゲッ

トは、EventBridge がイベントを送信できるリソースです。ターゲットには、API 送信先、Amazon CloudWatch ロググループなどを含めることができます。ルールの詳細については、「Amazon EventBridge ルール」を参照してください。ターゲットの詳細については、「Amazon EventBridge ターゲット」を参照してください。

すべてのゼロ ETL 通知をキャプチャするには、以下に一致する EventBridge ルールを作成します。

```
{
  "source": [{
    "prefix": "aws.glue-zero-etl“
  }],
  "detail-type": [{
    "prefix": "Glue Zero ETL“
  }]
}
```

次の表には、ゼロ ETL 統合イベントと、追加のメタデータをまとめています。

顧客向けの詳細タイプ	説明
Glue ゼロ ETL 取り込みの完了	エンティティの個々の実行が正常に完了しました。
Glue ゼロ ETL 取り込みの失敗	エンティティの個々の実行が正常に完了しませんでした (クライアントまたはシステムエラー)。
Glue ゼロ ETL 統合の再同期	統合が再同期されました。
Glue ゼロ ETL 統合の失敗	エラーのため統合ステータスが FAILED に変更されました。
Glue ゼロ ETL 統合に注意が必要	エラーのため統合ステータスが NEEDS_ATTENTION に変更されました。

制限

以下は、ゼロ ETL 統合の一般的な制限または考慮事項です。

- リソースプロパティは、対応するリソースと 1 対 1 の関係を持ちます。そのため、そのリソースを使用して作成されたすべての統合は、単一のリソースプロパティに従う必要があります。したがって、リソースプロパティを変更すると、そのリソースに関連付けられているすべての統合に影響します。
- テーブルプロパティは、リソース内の対応するテーブルまたはオブジェクトと 1 対 1 の関係を持ちます。そのため、同じリソースとの間で同じテーブルを処理するすべての統合は、単一のテーブルプロパティに従う必要があります。
- ソースの列の名前を変更することはできません。列の名前が変更された場合、スキーマ検出が AWS Glue によって正確に行われる保証はなく、統合への影響も明確ではありません。
- 統合と AWS Lake Formation マネージドテーブルの連携には、次の考慮事項が適用されます。デフォルトでは、IAM/AWS Glue ポリシーを使用してテーブルとデータベースを管理します。
- AWS Lake Formation を使用してそのデータベースのテーブル作成を管理する場合は、テーブルとデータベースを作成、変更、削除するのに十分な Lake Formation 許可がロールに付与されていることを確認する必要があります。
- 現時点では、ゼロ ETL の概要ページにはメトリクスは含まれていません。

ゼロ ETL 統合のソース固有の制限は次のとおりです。

- SAP OData ソースとのゼロ ETL 統合では、`Entity0f` で始まるエンティティがサポートされるようになりました。プライマリキーを上書きする機能は、現在 SAPOData `Entity0f` オブジェクトでのみサポートされています。このプロパティが設定されると、変更することはできません。
- Amazon DynamoDB から Amazon SageMaker Lakehouse へのゼロ ETL 統合 (S3 経由) では、最大 50TB の DynamoDB テーブルサイズがサポートされます。
- ソース DynamoDB テーブルは、Amazon 所有またはカスタマー管理の KMS AWS キーで暗号化する必要があります。Amazon が管理する暗号化は、ソース DynamoDB テーブルではサポートされていません。
- SAP OData は、デルタトークンを使用して機能し、ここでは OAuth クライアントとエンティティの組み合わせ、または基本認証とエンティティの組み合わせに対してデルタトークンを 1 つのみ持つことができます。同じクライアントとの 2 つの異なる統合で同じエンティティを使用することは避けてください。
- 次の Salesforce エンティティまたはフィールドは、Salesforce ソースとのゼロ ETL 統合ではサポートされていません。

```
AccountChangeEvent, AccountContactRoleChangeEvent, AccountHistory, AccountShare,
 ActiveFeatureLicenseMetric, ActivePermSetLicenseMetric, ActiveProfileMetric,
```

```
ActivityFieldHistory, amzsec__asi_Telemetry_Data_Store__ChangeEvent,
amzsec__asi_Telemetry_Data_Store__History, amzsec__asi_Telemetry_Data_Store__Share,
amzsec__asi_Telemetry_Job_Log__ChangeEvent, amzsec__asi_Telemetry_Job_Log__History,
amzsec__asi_Telemetry_Job_Log__Share,
amzsec__asi_Telemetry_Requirement__ChangeEvent,
amzsec__asi_Telemetry_Requirement__History,
amzsec__asi_Telemetry_Requirement__Share, ApexClass, ApexComponent, ApexLog,
ApexPage, ApexTestQueueItem, ApexTestResult, ApexTrigger, AssetChangeEvent,
AssetHistory, AssetRelationshipHistory, AssetShare, AssignmentRule,
AssociatedLocationHistory, AsyncApexJob, AuditTrailFileExportShare,
AuthorizationFormConsentChangeEvent, AuthorizationFormConsentHistory,
AuthorizationFormConsentShare, AuthorizationFormDataUseHistory,
AuthorizationFormDataUseShare, AuthorizationFormHistory, AuthorizationFormShare,
AuthorizationFormTextHistory, AuthProvider, AuthSession, BatchJobHistory,
BatchJobPartFailedRecordHistory, BatchJobPartHistory, BatchJobShare,
BrandTemplate, BriefcaseAssignmentChangeEvent, BriefcaseDefinitionChangeEvent,
BusinessBrandShare, BusinessHours, BusinessProcess, CalcMatrixColumnRangeHistory,
CalcProcStepRelationshipHistory, CalculationMatrixColumnHistory,
CalculationMatrixHistory, CalculationMatrixRowHistory, CalculationMatrixShare,
CalculationMatrixVersionHistory, CalculationProcedureHistory,
CalculationProcedureShare, CalculationProcedureStepHistory,
CalculationProcedureVariableHistory, CalculationProcedureVersionHistory, Calendar,
CalendarViewShare, CallCenter, CallCoachConfigModifyEvent, CampaignChangeEvent,
CampaignHistory, CampaignMemberChangeEvent, CampaignMemberStatusChangeEvent,
CampaignShare, CaseChangeEvent, CaseHistory, CaseHistory2 CaseHistory2ChangeEvent,
CaseRelatedIssueChangeEvent, CaseRelatedIssueHistory, CaseShare, CaseStatus,
CaseTeamMember, CaseTeamRole, CaseTeamTemplate, CaseTeamTemplateMember,
CaseTeamTemplateRecord, CategoryNode, ChangeRequestChangeEvent,
ChangeRequestHistory, ChangeRequestRelatedIssueChangeEvent,
ChangeRequestRelatedIssueHistory, ChangeRequestRelatedItemChangeEvent,
ChangeRequestRelatedItemHistory, ChangeRequestShare, ChatRetirementRdyMetrics,
ChatterActivity, ClientBrowser, CollaborationGroup, CollaborationGroupMember,
CollaborationGroupMemberRequest, CollaborationInvitation,
CommSubscriptionChannelTypeHistory, CommSubscriptionChannelTypeShare,
CommSubscriptionConsentChangeEvent, CommSubscriptionConsentHistory,
CommSubscriptionConsentShare, CommSubscriptionHistory, CommSubscriptionShare,
CommSubscriptionTimingHistory, Community, ConnectedApplication, ContactChangeEvent,
ContactHistory, ContactPointAddressChangeEvent, ContactPointAddressHistory,
ContactPointAddressShare, ContactPointConsentChangeEvent,
ContactPointConsentHistory, ContactPointConsentShare,
ContactPointEmailChangeEvent, ContactPointEmailHistory, ContactPointEmailShare,
ContactPointPhoneChangeEvent, ContactPointPhoneHistory, ContactPointPhoneShare,
ContactPointTypeConsentChangeEvent, ContactPointTypeConsentHistory,
ContactPointTypeConsentShare, ContactRequestShare, ContactShare,
```

```
ContentDocumentChangeEvent, ContentDocumentHistory, ContentDocumentLink,
ContentDocumentLinkChangeEvent, ContentDocumentSubscription, ContentFolderItem,
ContentFolderLink, ContentFolderMember, ContentNote, ContentNotification,
ContentTagSubscription, ContentUserSubscription, ContentVersionChangeEvent,
ContentVersionComment, ContentVersionHistory, ContentVersionRating,
ContentWorkspace, ContentWorkspaceMember, ContentWorkspacePermission,
ContentWorkspaceSubscription, ContractChangeEvent, ContractHistory,
ContractLineItemChangeEvent, ContractLineItemHistory, ContractStatus,
Conversation, ConversationParticipant, CronJobDetail, CronTrigger, CustomBrand,
CustomBrandAsset, CustomerShare, CustomHttpHeader, DashboardComponent,
DataUseLegalBasisHistory, DataUseLegalBasisShare, DataUsePurposeHistory,
DataUsePurposeShare, DecisionTableRecordset, DeleteEvent, DocumentAttachmentMap,
Domain, DomainSite, DTRecordsetReplicaShare, EmailBounceEvent,
EmailMessageChangeEvent, EmailServicesAddress, EmailServicesFunction,
EmailTemplate, EmailTemplateChangeEvent, EngagementAttendeeChangeEvent,
EngagementAttendeeHistory, EngagementChannelTypeHistory, EngagementChannelTypeShare,
EngagementInteractionChangeEvent, EngagementInteractionHistory,
EngagementInteractionShare, EngagementInterface, EngagementTopicChangeEvent,
EngagementTopicHistory, EntitlementChangeEvent, EntitlementHistory,
EntitlementTemplate, EntityMilestoneHistory, EntitySubscription, EventChangeEvent,
EventRelationChangeEvent, EventRelayConfigChangeEvent, ExpressionSetHistory,
ExpressionSetShare, ExpressionSetVersionHistory, ExternalEventMappingShare,
FeedAttachment, FeedLike, FeedPollChoice, FeedPollVote, FeedSignal,
FieldPermissions, FieldSecurityClassification, FiscalYearSettings,
FlowInterviewLogShare, FlowInterviewShare, FlowOrchestrationEvent,
FlowOrchestrationInstanceShare, FlowOrchestrationStageInstanceShare,
FlowOrchestrationStepInstanceShare, FlowOrchestrationWorkItemShare,
FlowRecordShare, FlowRecordVersionChangeEvent, FlowTestResultShare, Folder,
Group, GroupMember, Holiday, IdeaComment, IdpEventLog, ImageHistory, ImageShare,
IncidentChangeEvent, IncidentHistory, IncidentRelatedItemChangeEvent,
IncidentRelatedItemHistory, IncidentShare, IndividualChangeEvent,
IndividualHistory, IndividualShare, KnowledgeableUser, LeadChangeEvent, LeadHistory,
LeadShare, LeadStatus, LightningExitByPageMetrics, LightningToggleMetrics,
LightningUsageByAppTypeMetrics, LightningUsageByBrowserMetrics,
LightningUsageByFlexiPageMetrics, LightningUsageByPageMetrics, ListEmailChangeEvent,
ListEmailShare, ListView, LocationChangeEvent, LocationHistory, LocationShare,
LocationTrustMeasureShare, LoginHistory, LoginIp, MacroChangeEvent,
MacroHistory, MacroInstructionChangeEvent, MacroShare, MacroUsageShare,
ManagedContentVariantChangeEvent, MessagingEndUserHistory, MessagingEndUserShare,
MessagingSessionHistory, MessagingSessionShare, MilestoneType, MLEngagementEvent,
ObjectPermissions, OpportunityChangeEvent, OpportunityContactRoleChangeEvent,
OpportunityFieldHistory, OpportunityLineItemChangeEvent, OpportunityShare,
OpportunityStage, OrderChangeEvent, OrderHistory, OrderItemChangeEvent,
OrderItemHistory, OrderShare, OrderStatus, Organization, OrgEmailAddressSecurity,
```

```
OrgWideEmailAddress, OutgoingEmail, OutgoingEmailRelation,
PackageLicense, PartnerRole, PartyConsentChangeEvent, PartyConsentHistory,
PartyConsentShare, Period, PermissionSet, PermissionSetAssignment,
PermissionSetTabSetting, Pricebook2ChangeEvent, Pricebook2History,
PricebookEntryChangeEvent, PricebookEntryHistory, PrivacyJobSessionShare,
PrivacyObjectSessionShare, PrivacyRTBFRequestHistory, PrivacyRTBFRequestShare,
PrivacySessionRecordFailureShare, ProblemChangeEvent, ProblemHistory,
ProblemIncidentChangeEvent, ProblemIncidentHistory, ProblemRelatedItemChangeEvent,
ProblemRelatedItemHistory, ProblemShare, ProcessDefinition, ProcessExceptionEvent,
ProcessExceptionShare, ProcessInstanceChangeEvent, ProcessInstanceStep,
ProcessInstanceStepChangeEvent, ProcessNode, Product2ChangeEvent, Product2History,
ProductEntitlementTemplate, Profile, ProfileSkillEndorsementHistory,
ProfileSkillHistory, ProfileSkillShare, ProfileSkillUserHistory, PromptActionShare,
PromptErrorShare, QueueSobject, QuickTextChangeEvent, QuickTextHistory,
QuickTextShare, QuickTextUsageShare, RecentlyViewed, RecommendationChangeEvent,
RecordActionHistory, RecordAlertHistory, RecordAlertShare, RecordType,
ScorecardShare, SellerHistory, SellerShare, ServiceContractChangeEvent,
ServiceContractHistory, ServiceContractShare, SetupAuditTrail, SetupEntityAccess,
SharingRecordCollectionShare, Site, SiteHistory, SiteRedirectMapping,
SocialPersonaHistory, SocialPostChangeEvent, SocialPostHistory, SocialPostShare,
SolutionHistory, SolutionStatus, StaticResource, StreamingChannelShare,
TableauHostMappingShare, TaskChangeEvent, TaskPriority, TaskStatus,
ThreatDetectionFeedback, TimelineObjectDefinitionChangeEvent, TodayGoalShare,
Topic, TopicUserEvent, Translation, User, UserAppMenuCustomizationShare,
UserChangeEvent, UserDefinedLabelAssignmentShare, UserDefinedLabelShare,
UserEmailPreferredPersonShare, UserLicense, UserLogin, UserPackageLicense,
UserPreference, UserPrioritizedRecordShare, UserProvisioningRequestShare,
UserRole, UserShare, VideoCallChangeEvent, VideoCallParticipantChangeEvent,
VideoCallRecordingChangeEvent, VideoCallShare, VisualforceAccessMetrics,
VoiceCallChangeEvent, VoiceCallRecordingChangeEvent, VoiceCallShare, Vote,
WebLink, WorkAccessShare, WorkBadgeDefinitionHistory, WorkBadgeDefinitionShare,
WorkOrderChangeEvent, WorkOrderHistory, WorkOrderLineItemChangeEvent,
WorkOrderLineItemHistory, WorkOrderLineItemStatus, WorkOrderShare,
WorkOrderStatus, WorkPlanChangeEvent, WorkPlanHistory, WorkPlanShare,
WorkPlanTemplateChangeEvent, WorkPlanTemplateEntryChangeEvent,
WorkPlanTemplateEntryHistory, WorkPlanTemplateHistory, WorkPlanTemplateShare,
WorkStepChangeEvent, WorkStepHistory, WorkStepStatus, WorkStepTemplateChangeEvent,
WorkStepTemplateHistory, WorkStepTemplateShare, WorkThanksShare
```

- 次以下の ServiceNow エンティティまたはフィールドは、ServiceNow ソースとのゼロ ETL 統合ではサポートされていません。

```
ais_acl_overrides, ais_async_genius_result, ais_async_request,
 ais_connection, ais_genius_result_configuration_parameters,
 ais_partition_health, ais_partition_health_response, ais_publish_history,
 ais_relevancy_training_execution, ais_relevancy_training_staging,
 ais_search_profile_relevancy_model, catalog_draft_entities, clone_log,
 clone_log0000, clone_log0001, clone_log0002, clone_log0003, clone_log0004,
 clone_log0005, clone_log0006, clone_log0007, cmdb_ie_context, cmdb_ie_log,
 cmdb_ie_run, cmdb_ire_partial_payloads_index, cmdb_qb_result_base
$par1, cmdb$par1, discovery_log, discovery_log0000, discovery_log0001,
 discovery_log0002, discovery_log0003, discovery_log0004, discovery_log0005,
 discovery_log0006, discovery_log0007, ecc_agent_log, entitlement_data,
 entl_subscription_map, gs_entitlement_plugin_mapping, ih_transaction_exclusion,
 import_log, import_log0000, import_log0001, import_log0002, import_log0003,
 import_log0004, import_log0005, import_log0006, import_log0007, jrobin_archive,
 jrobin_database, jrobin_datasource, jrobin_definition, jrobin_graph,
 jrobin_graph_line, jrobin_graph_set, jrobin_graph_set_member, jrobin_shard,
 jrobin_shard_location, license_role_discovery_run, logger_configuration_validation,
 m2m_analytics_event_logger, m2m_user_consent_info, ml_artifact_object_store,
 np$sys_gen_ai_filter_sample, np$sys_ui_element, np$sys_ui_list_element,
 one_api_service_plan_feature_invocation, one_api_service_plan_invocation,
 open_nlu_predict_log, open_nlu_predict_log0000, open_nlu_predict_log0001,
 open_nlu_predict_log0002, open_nlu_predict_log0003, open_nlu_predict_log0004,
 open_nlu_predict_log0005, open_nlu_predict_log0006, open_nlu_predict_log0007,
 pa_diagnostic_log, pa_diagnostic_log0000, pa_diagnostic_log0001,
 pa_diagnostic_log0002, pa_diagnostic_log0003, pa_diagnostic_log0004,
 pa_diagnostic_log0005, pa_diagnostic_log0006, pa_diagnostic_log0007, pa_favorites,
 pa_job_log_rows, pa_job_log_rows0000, pa_job_log_rows0001, pa_job_log_rows0002,
 pa_job_log_rows0003, pa_job_log_rows0004, pa_job_log_rows0005, pa_job_log_rows0006,
 pa_job_log_rows0007, pa_migration_ignored_scores, pa_scores_l1, pa_scores_l2,
 pa_scores_migration_groups, par_dashboard_conversion_backup, promin_log,
 promin_log0000, promin_log0001, promin_log0002, promin_log0003, promin_log0004,
 promin_log0005, promin_log0006, promin_log0007, promin_request_object,
 proposed_change_verification_log, proposed_change_verification_log0000,
 proposed_change_verification_log0001, proposed_change_verification_log0002,
 proposed_change_verification_log0003, proposed_change_verification_log0004,
 proposed_change_verification_log0005, proposed_change_verification_log0006,
 proposed_change_verification_log0007, protected_table_log, protected_table_log0000,
 protected_table_log0001, protected_table_log0002, protected_table_log0003,
 protected_table_log0004, protected_table_log0005, protected_table_log0006,
 protected_table_log0007, pwd_history, qb_query_results, scan_log,
 scan_log0000, scan_log0001, scan_log0002, scan_log0003, scan_log0004,
 scan_log0005, scan_log0006, scan_log0007, schema_validator_error,
 sla_repair_log_entry, sla_repair_log_entry0000, sla_repair_log_entry0001,
```

```
sla_repair_log_entry0002, sla_repair_log_entry0003, sla_repair_log_entry0004,
sla_repair_log_entry0005, sla_repair_log_entry0006, sla_repair_log_entry0007,
sla_repair_log_message, sla_repair_log_message0000, sla_repair_log_message0001,
sla_repair_log_message0002, sla_repair_log_message0003, sla_repair_log_message0004,
sla_repair_log_message0005, sla_repair_log_message0006, sla_repair_log_message0007,
sn_bm_client_activity, sn_ci_analytics_st_actionable_notifs,
sn_ci_analytics_st_conv_completion_by_cat, sn_ci_analytics_st_conv_dynamic_property,
sn_ci_analytics_st_conversation, sn_ci_analytics_st_count_by_date,
sn_ci_analytics_st_event_occurrence, sn_ci_analytics_st_event_property_value_trend,
sn_ci_analytics_st_issue_auto_resolution, sn_ci_analytics_st_no_clicks,
sn_ci_analytics_st_no_results, sn_ci_analytics_st_property_summary_by_event,
sn_ci_analytics_st_session_count_per_locale, sn_ci_analytics_st_session_duration,
sn_ci_analytics_st_spokes_usage, sn_ci_analytics_st_topic_execution_stats,
sn_ci_analytics_st_topic_occurrence, sn_ci_analytics_st_trending_content,
sn_ci_analytics_st_trending_queries, sn_ci_analytics_st_users,
sn_cs_plugin_signatures, sn_cs_telemetry_log, sn_dfc_application, sn_dfc_product,
sn_employee_position, sn_entitlement_st_subscription_application_family,
sn_entitlement_st_subscription_application_users, sn_hr_sp_st_relevant_for_you,
sn_instance_clone_log, sn_instance_clone_log0000, sn_instance_clone_log0001,
sn_instance_clone_log0002, sn_instance_clone_log0003, sn_instance_clone_log0004,
sn_instance_clone_log0005, sn_instance_clone_log0006, sn_instance_clone_log0007,
sn_km_mr_st_kb_knowledge, sn_me_st_topic, sn_rf_conditional_definition,
sn_rf_evaluation_type, sn_rf_evaluation_type_input, sn_rf_recommendation_action,
sn_rf_recommendation_experience, sn_rf_recommendation_history,
sn_rf_recommendation_rule, sn_rf_record_display_configuration,
sn_rf_trend_definition, sn_sub_man_st_account_level_entitlement,
sn_sub_man_st_gen_ai_metadata, sn_sub_man_st_instance_used_assist_count,
sn_sub_man_st_now_assist_creator_instances, sn_sub_man_st_now_assists_aggregate,
sn_sub_man_st_subscribed_groups, sn_sub_man_st_subscription_insights,
sn_sub_man_st_subscription_license_detail_metric,
sn_sub_man_st_unallocated_group_recommendation,
sn_sub_man_st_unconfirmed_user_group, sn_wn_user_app_activity,
sn_wn_user_content_activity, snc_monitorable_item,
snpar_sched_export_v_scheduled_export_visualization, spotlight,
spotlight_audit, spotlight_copy_log_row, spotlight_copy_log_row0000,
spotlight_copy_log_row0001, spotlight_copy_log_row0002, spotlight_copy_log_row0003,
spotlight_copy_log_row0004, spotlight_copy_log_row0005, spotlight_copy_log_row0006,
spotlight_copy_log_row0007, spotlight_job_log_row, spotlight_job_log_row0000,
spotlight_job_log_row0001, spotlight_job_log_row0002, spotlight_job_log_row0003,
spotlight_job_log_row0004, spotlight_job_log_row0005, spotlight_job_log_row0006,
spotlight_job_log_row0007, st_dfc_performance_metric, st_license_detail_metric,
st_on_call_hour, st_sc_wizard_question, st_sys_catalog_items_and_variable_sets,
st_sys_design_system_icon, subscription_instance_stats,
svc_container_config, svc_environment_config, svc_layer_config,
```

```
svc_model_assoc_ci, svc_model_checkpoint_attr, svc_model_obj_cluster,
svc_model_obj_constraint, svc_model_obj_deployable, svc_model_obj_element,
svc_model_obj_impact, svc_model_obj_impactrule, svc_model_obj_package,
svc_model_obj_path, svc_model_obj_relation, svc_model_obj_service,
sys_administrative_script_transaction, sys_amb_message, sys_amb_processor,
sys_analytics_batch_state, sys_analytics_config, sys_analytics_data_points_error,
sys_analytics_event, sys_analytics_logger, sys_analytics_logger_field,
sys_app_payload_loader_rule, sys_app_payload_unloader_rule, sys_app_scan_payload,
sys_app_scan_variable, sys_app_scan_variable_type, sys_archive_destroy_log,
sys_archive_destroy_run, sys_archive_log, sys_archive_run, sys_atf_transaction_log,
sys_attachment_doc, sys_attachment_doc_v2, sys_attachment_soft_deleted, sys_audit,
sys_audit_relation, sys_auth_policy_api_allowed, sys_aw_registered_scripting_modal,
sys_cache_flush, sys_data_egress_source, sys_dm_delete_count,
sys_export_set_log, sys_export_set_log0000, sys_export_set_log0001,
sys_export_set_log0002, sys_export_set_log0003, sys_export_set_log0004,
sys_export_set_log0005, sys_export_set_log0006, sys_export_set_log0007,
sys_flow_compiled_flow, sys_flow_compiled_flow_chunk, sys_flow_context_chunk,
sys_flow_context_chunk_archive, sys_flow_context_inputs_chunk,
sys_flow_execution_history, sys_flow_log, sys_flow_log0000, sys_flow_log0001,
sys_flow_log0002, sys_flow_log0003, sys_flow_plan_context_binding,
sys_flow_report_doc, sys_flow_report_doc_chunk, sys_flow_report_doc_chunk_archive,
sys_flow_runtime_state_chunk, sys_flow_runtime_value_chunk,
sys_flow_subflow_plan_chunk, sys_flow_trigger_plan_chunk, sys_flow_val_listener,
sys_flow_value, sys_flow_value_chunk, sys_gen_ai_config_example,
sys_gen_ai_feature_mapping, sys_gen_ai_strategy_mapping, sys_gen_ai_usage_log,
sys_generative_ai_capability_definition, sys_generative_ai_log,
sys_generative_ai_response_validator, sys_generative_ai_validator,
sys_geo_routing, sys_geo_routing_config, sys_hop_token, sys_hub_action_plan_chunk,
sys_hub_snapshot_chunk, sys_journal_field, sys_journal_field_edit, sys_json_chunk,
sys_kaa_policy, sys_kaa_subidentity_assertion, sys_kaa_user_policy_mapping,
sys_mapplication, sys_mass_encryption_job, sys_notification_execution_log,
sys_notification_execution_log0000, sys_notification_execution_log0001,
sys_notification_execution_log0002, sys_notification_execution_log0003,
sys_notification_execution_log0004, sys_notification_execution_log0005,
sys_notification_execution_log0006, sys_notification_execution_log0007,
sys_nowmq_message, sys_nowmq_provider_param_definition, sys_orchestrator_action,
sys_pd_asset_configuration, sys_pd_context_chunk, sys_pd_context_log,
sys_pd_snapshot_chunk, sys_pd_trigger_license, sys_processing_framework_job,
sys_query_index_hint, sys_query_rewrite, sys_query_string_log,
sys_replication_queue, sys_replication_queue0, sys_replication_queue1,
sys_replication_queue2, sys_replication_queue3, sys_replication_queue4,
sys_replication_queue5, sys_replication_queue6, sys_replication_queue7,
sys_request_performance, sys_rollback_blacklisted, sys_rollback_conflict,
sys_rollback_incremental, sys_rollback_log, sys_rollback_log0000,
```

```
sys_rollback_log0001, sys_rollback_log0002, sys_rollback_log0003,
sys_rollback_log0004, sys_rollback_log0005, sys_rollback_log0006,
sys_rollback_log0007, sys_rollback_run, sys_rollback_schema_change,
sys_rollback_schema_conflict, sys_rollback_sequence, sys_scheduler_assignment,
sys_scheduler_memory_pressure_job_log, sys_script_adapter_rule,
sys_script_batch_adapter_rule, sys_search_source_filter, sys_service_authentication,
sys_signing_job, sys_suggestion_reader, sys_sync_history_review, sys_trend,
sys_unreferenced_preview, sys_unreferenced_record_rule, sys_upgrade_manifest,
sys_upgrade_state, sys_ux_asset_cache_buster, sys_ux_lib_component_prop,
sys_ux_lib_presource, sys_ux_page_action, sys_ux_page_action_binding,
sysevent_queue_runtime, syslog, syslog_app_scope0000, syslog_app_scope0001,
syslog_app_scope0002, syslog_app_scope0003, syslog_app_scope0004,
syslog_app_scope0005, syslog_app_scope0006, syslog_app_scope0007, syslog_email0000,
syslog_email0001, syslog_email0002, syslog_email0003, syslog_email0004,
syslog_email0005, syslog_email0006, syslog_email0007, syslog_transaction,
syslog_transaction0000, syslog_transaction0001, syslog_transaction0002,
syslog_transaction0003, syslog_transaction0004, syslog_transaction0005,
syslog_transaction0006, syslog_transaction0007, syslog0000, syslog0001,
syslog0002, syslog0003, syslog0004, syslog0005, syslog0006, syslog0007,
ts_attachment, ts_c_1_0, ts_c_1_1, ts_c_1_2, ts_c_1_3, ts_c_1_4, ts_c_1_5,
ts_c_1_6, ts_c_1_7, ts_c_1_8, ts_c_1_9, ts_c_10_0, ts_c_10_1, ts_c_10_2,
ts_c_10_3, ts_c_10_4, ts_c_10_5, ts_c_10_6, ts_c_10_7, ts_c_10_8, ts_c_10_9,
ts_c_11_0, ts_c_11_1, ts_c_11_2, ts_c_11_3, ts_c_11_4, ts_c_11_5, ts_c_11_6,
ts_c_11_7, ts_c_11_8, ts_c_11_9, ts_c_2_0, ts_c_2_1, ts_c_2_2, ts_c_2_3,
ts_c_2_4, ts_c_2_5, ts_c_2_6, ts_c_2_7, ts_c_2_8, ts_c_2_9, ts_c_3_0,
ts_c_3_1, ts_c_3_2, ts_c_3_3, ts_c_3_4, ts_c_3_5, ts_c_3_6, ts_c_3_7, ts_c_3_8,
ts_c_3_9, ts_c_39_0, ts_c_39_1, ts_c_39_2, ts_c_39_3, ts_c_39_4, ts_c_39_5,
ts_c_39_6, ts_c_39_7, ts_c_39_8, ts_c_39_9, ts_c_40_0, ts_c_40_1, ts_c_40_2,
ts_c_40_3, ts_c_40_4, ts_c_40_5, ts_c_40_6, ts_c_40_7, ts_c_40_8, ts_c_40_9,
ts_c_41_0, ts_c_41_1, ts_c_41_2, ts_c_41_3, ts_c_41_4, ts_c_41_5, ts_c_41_6,
ts_c_41_7, ts_c_41_8, ts_c_41_9, ts_c_42_0, ts_c_42_1, ts_c_42_2, ts_c_42_3,
ts_c_42_4, ts_c_42_5, ts_c_42_6, ts_c_42_7, ts_c_42_8, ts_c_42_9, ts_c_43_0,
ts_c_43_1, ts_c_43_2, ts_c_43_3, ts_c_43_4, ts_c_43_5, ts_c_43_6, ts_c_43_7,
ts_c_43_8, ts_c_43_9, ts_c_44_0, ts_c_44_1, ts_c_44_2, ts_c_44_3, ts_c_44_4,
ts_c_44_5, ts_c_44_6, ts_c_44_7, ts_c_44_8, ts_c_44_9, ts_c_45_0, ts_c_45_1,
ts_c_45_2, ts_c_45_3, ts_c_45_4, ts_c_45_5, ts_c_45_6, ts_c_45_7, ts_c_45_8,
ts_c_45_9, ts_c_6_0, ts_c_6_1, ts_c_6_2, ts_c_6_3, ts_c_6_4, ts_c_6_5,
ts_c_6_6, ts_c_6_7, ts_c_6_8, ts_c_6_9, ts_c_7_0, ts_c_7_1, ts_c_7_2, ts_c_7_3,
ts_c_7_4, ts_c_7_5, ts_c_7_6, ts_c_7_7, ts_c_7_8, ts_c_7_9, ts_c_8_0, ts_c_8_1,
ts_c_8_2, ts_c_8_3, ts_c_8_4, ts_c_8_5, ts_c_8_6, ts_c_8_7, ts_c_8_8, ts_c_8_9,
ts_c_attachment, ts_chain, ts_deleted_doc, ts_document, ts_field, ts_index_stats,
ts_phrase, ts_search_stats, ts_v4_attachment, ts_word, ua_app_metadata,
ua_audit_stats, ua_extra_page, ua_monitor_property, ua_monitor_property_audit,
ua_shared_service, ua_sn_table_inventory, ua_sp_known_bot, ua_upload_log,
```

```
v_ais_result_improvement_rule_condition_builder_values, v_cluster_nodes,
v_cxs_search_resource, v_db_index, v_db_trigger, v_file_load_order,
v_interaction_context, v_iostats, v_mysql_proclist, v_mysql_status,
v_mysql_variables, v_on_call_report_cache, v_pa_par_combined_dashboard,
v_pd_activity_condition_to_run, v_pd_activity_start_rule_with_condition,
v_pd_lane_condition_to_run, v_sql_debug, v_st_kb_category, v_st_kb_most_viewed,
v_st_kb_recently_viewed, v_st_km_genai_mra_similar_task, v_st_popular_item,
v_st_recent_item, v_st_sc_cat_item, v_st_sc_catalog, v_st_sc_category,
validator_run_summary, vtb_card_history, wf_log, wf_log0000, wf_log0001,
wf_log0002, wf_log0003, wf_log0004, wf_log0005, wf_log0006, wf_log0007,
spotlight_criteria, pa_snapshots, pa_widget_indicators, pa_widgets,
sn_cim_register, global, gsw_change_log, gsw_content, gsw_content_group,
gsw_content_information, gsw_status_of_content, multi_factor_browser_fingerprint,
multi_factor_criteria, pa_filters, password_policy, plan_execution,
plan_mysql, plan_oracle, plan_postgres, sc_rest_api_without_access_policy,
sn_actsub_activity, sn_actsub_activity_fanout, sn_actsub_activity_stream,
sn_actsub_activity_type, sn_actsub_atype_attributes, sn_actsub_atype_notif_pref,
sn_actsub_module, sn_actsub_notif_object, sn_actsub_subobject_stream,
sn_actsub_subscribable_object, sn_actsub_subscription_notif_pref,
sn_actsub_user_stream, sn_appclient_store_outbound_http_quota,
sn_appcreator_app_template, sn_critical_update, sn_docker_spoke_images,
sn_employee_app, sn_employee_app_access, sn_employee_app_access_criteria,
sn_entitlement_genai_assist_counts, sn_entitlement_genai_creator_user_counts,
sn_entitlement_genai_creator_users, sn_mif_instance, sn_mif_sync_data,
sn_mif_sync_status, sn_mif_table_registration, sn_mif_trust_config,
sn_vsc_best_practice_configurations, sn_vsc_best_practice_goals,
sn_vsc_changed_hardening_settings, sn_vsc_changed_scan_findings,
sn_vsc_check_security_area, sn_vsc_elevation_event, sn_vsc_event,
sn_vsc_export_event, sn_vsc_export_setting, sn_vsc_harc_compliance_status_lookup,
sn_vsc_hardening_compliance_scores, sn_vsc_impersonation_event,
sn_vsc_instance_hardening_settings, sn_vsc_login_event,
sn_vsc_scan_comparisons, sn_vsc_scan_summary, sn_vsc_security_check_categories,
sn_vsc_security_check_configurations, sn_vsc_security_configuration_groups,
sn_vsc_security_privacy_capabilities, sn_vsc_updated_settings,
sn_vsc_user_comparisons, sys_app_hash_inventory, sys_coalesce_strategy_deferred,
sys_flow_secure_data, sys_formula_function, sys_geocoding_request,
sys_global_file_hash, sys_import_set_row, sys_index, sys_index_explain,
sys_installation_schedule, sys_installation_schedule_item, sys_offline_app,
sys_package, sys_package_dependency_item, sys_package_dependency_m2m,
sys_plugins, sys_querystat, sys_reap_package, sys_scoped_plugin,
sys_stage_storage_alias, sys_storage_alias, sys_storage_table_alias, sys_store_app,
sys_table_partition, sys_upgrade_history_log, sys_user_public_credential,
sys_webauthn_authentication_request, sys_webauthn_registration_request,
syslog_app_scope, syslog_email, syslog_page_timing, ua_instance_state_config,
```

```
v_expression_cache, v_private_cache, v_shared_cache, sys_amb_message0002,
sys_amb_message0004, sys_amb_message0005, sys_metadata, v_par_unified_report_viz
```

AWS Glue Data Quality

AWS Glue Data Quality を使用するとデータ品質の測定とモニタリングが行えて、ビジネス上の意思決定を適切に行えるようになります。オープンソースの DeeQu フレームワーク上に構築された AWS Glue Data Quality は、マネージド型のサーバーレス体験を提供します。AWS GlueData Quality はデータ品質定義言語 (DQDL) を使用します。DQDL は、データ品質のルールの定義に使用されるドメイン固有の言語です。DQDL とサポートされているルールタイプの詳細については、「データ品質定義言語 (DQDL) リファレンス」を参照してください。

製品の詳細および価格に関するその他詳細は、AWS Glue Data Quality のサービスページを参照してください。

利点と主な特徴

AWS Glue Data Quality のメリットと主な特徴は次のとおりです。

- サーバーレス - インストール、パッチ適用、またはメンテナンスは不要です。
- すぐに開始できる — AWS Glue Data Quality はデータをすばやく分析し、ユーザーに代わってデータ品質ルールを作成します。[Create Data Quality Rules] と [Recommend rules] をクリックするだけですぐに始められます。
- データ品質問題の検出 — 異常や、検出が困難なデータ品質問題を、機械学習 (ML) を使用して検出します。
- ルールを即席で作れる — 追加設定のいらない 25 以上の DQ ルールを使用してすぐに始められ、具体的なニーズに合わせてルールを作成できます。
- 品質を評価し、自信を持って判断を下せる — ルールを評価すると、データの状態に関する概要を示した Data Quality スコアを確認できます。この Data Quality スコアを使用すれば、自信を持ってビジネス上の判断を下せます。
- 不良データに的を絞る — AWS Glue Data Quality を使用すると、品質スコアを下げる原因となったレコードを正確に特定できます。それらをすぐに特定し、隔離して、修正できます。
- 従量制料金 — AWS Glue Data Quality の使用には年間ライセンスは不要です。
- ロックインなし — AWS Glue Data Quality はオープンソースの DeeQu 上に構築されているため、作成中のルールをオープン言語で維持できます。
- データ品質チェック - データ品質のチェックを Data Catalog と AWS Glue ETL パイプラインで行えるため、保管中および転送中のデータの品質を管理できます。

- ML ベースのデータ品質検出 — 異常や、検出が困難なデータ品質問題を、機械学習 (ML) を使用して検出します。
- ルールを表現するためのオープン言語 - データ品質ルールが一貫してシンプルに作成できます。ビジネスユーザーは、理解できるわかりやすい言語でデータ品質ルールを簡単に表現できます。エンジニアにとっては、コードの生成、一貫したバージョン管理の実装およびデプロイの自動化を可能にする柔軟性がこの言語によって提供されます。

仕組み

AWS Glue Data Quality のエントリポイントは 2 つあります。AWS Glue Data Catalog と AWS Glue ETL ジョブです。このセクションでは、ユースケースの概要と、各エントリポイントでサポートされている AWS Glue の機能について説明します。

AWS Glue Data Catalog のデータ品質

AWS Glue Data Quality は、AWS Glue Data Catalog に保存されているオブジェクトを評価します。プログラマー以外の人でもデータ品質ルールを簡単に設定できます。例えば、データスチュワードやビジネスアナリストなどです。

この方法は、以下のようなユースケースに適しています。

- 既に AWS Glue Data Catalog でカタログ化しているデータセットに対してデータ品質タスクを実行したい場合
- データガバナンスに取り組んでおり、データレイク内のデータ品質に関する問題を、継続的に特定または評価する必要がある場合

データカタログのデータ品質は、次のインターフェイスで管理できます。

- AWS Glue マネジメントコンソール
- AWS Glue API

AWS Glue Data Catalog で AWS Glue Data Quality を使用する手順については、「Data Catalog で AWS Glue Data Quality を使用する」を参照してください。

AWS Glue ETL ジョブのデータ品質

AWS Glue ETL ジョブ向けの AWS Glue Data Quality を使用すると、プロアクティブなデータ品質タスクを実行できます。プロアクティブなタスクは、データセットをデータレイクにロードする前に不良データを特定し、除外するのに役立ちます。

動画: ETL パイプライン向けの AWS Glue Data Quality 入門

ETL ジョブ向けの Data Quality は以下のようなユースケースに使用できます。

- データ品質タスクを ETL ジョブに組み込む場合
- データ品質タスクを定義するコードを ETL スクリプトで記述する場合
- ビジュアルデータパイプラインに送信されるデータの品質を管理する場合

ETL ジョブのデータ品質は、次のインターフェイスで管理できます。

- AWS Glue Studio、AWS Glue Studio、ノートブック、AWS Glue インタラクティブセッション
- ETL スクリプト用の AWS Glue ライブラリ
- AWS Glue API

ETL ジョブ向け Data Quality の使用方法については、「AWS Glue Studio ユーザーガイド」の「チュートリアル: Data Quality の使用開始」を参照してください。

データカタログ向け Data Quality と ETL ジョブ向け Data Quality の比較

こちらの表は、AWS Glue Data Quality がサポートしている各エントリポイントの機能の概要を示したものです。

機能	データカタログ向け Data Quality	ETL ジョブ向け Data Quality
データソース	Amazon S3、Amazon Redshift、データカタログと互換性のある JDBC ソース、および Apache Iceberg、Apache Hudi、Delta Lake などトランザクションデータレイク	AWS Glue でサポートされているすべてのデータソース。カスタムのコネクタとサードパーティーのコネクタを含みます。

機能	データカタログ向け Data Quality	ETL ジョブ向け Data Quality
	形式。テーブルが AWS Lake Formation で管理されている場合、Iceberg、Delta、HUDI テーブルはサポートされないことに注意してください。AWS Glue Data Catalog にカタログ登録されている Amazon Athena ビューはサポートされていません。	
Data Quality ルールの推奨事項	サポート	サポートされていません
DQDL ルールの作成と実行	サポート	サポート
Auto scaling	サポート外	サポート
AWS Glue Flex のサポート	サポート外	サポート
スケジューリング	Data Quality ルールを評価するときおよび Step Functions を使用する場合にサポート	Step Functions とワークフローを使用する場合にサポート
データ品質チェックに失敗したレコードの特定	サポート外	サポート
Amazon EventBridge との統合	サポート	サポート
AWS CloudWatch との統合	サポート	サポート
データ品質評価の結果を Amazon S3 に書き込む	サポート	サポート
増分的なデータ品質	プッシュダウン述語によりサポート	AWS Glue ブックマークによりサポート

機能	データカタログ向け Data Quality	ETL ジョブ向け Data Quality
AWS CloudFormation のサポート	サポート	サポート
ML ベースの異常検出	サポート外	サポート
動的ルール	サポート外	サポート

考慮事項

AWS Glue Data Quality を使用するときは、事前に次の点を確認します。

- データ品質ルールは、ネストされたデータソースやリストタイプのデータソースを評価することはできません。「ネストされた Struct のフラット化」を参照してください。

用語

次の表では、AWS Glue Data Quality に関連する用語が定義されています。

データ品質定義言語 (DQDL)

AWS Glue Data Quality のルール作成に使用できるドメイン固有の言語。

DQDL の詳細については、データ品質定義言語 (DQDL) リファレンス のガイドを参照してください。

データ品質

データセットがある目的をどの程度果たしているかを示します。 AWS GlueData Quality は、データセットに対するルールを評価してデータ品質を測定します。各ルールは、データの鮮度や整合性などの特性を確認します。データ品質を数値化するには、データ品質スコアを使用できます。

データ品質スコア

AWS Glue Data Quality でルールセットを評価する際に、合格する (結果が true になる) データ品質ルールの割合。

ルール

データにおける特性を確認し、ブール値を返す DQDL 式。詳細については、「ルールの構造」を参照してください。

アナライザー

データ統計を収集する DQDL 式。アナライザーは、ML アルゴリズムで使用できるデータ統計を収集し、異常や、時間の経過に伴って検出が困難となるデータ品質問題を検出できます。

ルールセット

一連のデータ品質ルールで構成される AWS Glue リソース。ルールセットは、AWS Glue Data Catalog 内のテーブルと関連付ける必要があります。ルールセットを保存すると、AWS Glue は Amazon リソースネーム (ARN) をルールセットに割り当てます。

データ品質スコア

AWS Glue Data Quality でルールセットを評価する際に、合格する (結果が true になる) データ品質ルールの割合。

監視

ルールやアナライザーから収集されたデータ統計を経時的に分析することで、AWS Glue によって得られる未確認のインサイト。

制限

AWS Glue Data Quality サービスの制限:

- ルールセットには 2,000 個のルールを含めることができます。ルールセットが大きい場合は、複数のルールセットに分割することをお勧めします。
- ルールセットのサイズは 65 KB です。ルールセットが大きい場合は、複数のルールセットに分割することをお勧めします。
- AWS Glue Data Quality は、ルールまたはアナライザーを作成するときに統計を収集します。これらの統計の保存にはコストはかかりません。ただし、アカウントごとに保存できる統計は 100,000 に制限されており、これらの統計は最大 2 年間保持されます。

AWS Glue Data Quality のリリースノート

このトピックでは、AWS Glue Data Quality に導入された機能について説明します。

一般提供を開始: 新機能

以下は、AWS Glue Data Quality の一般提供の開始に伴い利用できるようになった新機能です。

- データ品質チェックで不合格になったレコードを特定する機能が AWS Glue Studio でサポート
- 新しいデータ品質のルールタイプ。2 つのデータセット間におけるデータの参照整合性の検証、2 つのデータセット間におけるデータの比較、データタイプのチェックなど
- AWS Glue Data Catalog でのユーザーエクスペリエンスの向上
- Apache Iceberg、Apache Hudi、Delta Lake のサポート
- Amazon Redshift のサポート
- Amazon EventBridge による通知の簡素化
- ルールセット作成のための AWS CloudFormation サポート
- パフォーマンスの向上。ETL のキャッシュオプションと AWS Glue Studio によりデータ品質を評価する際のパフォーマンスが高速化

2023 年 11 月 27 日 (プレビュー)

- ML を利用した異常検出機能が AWS Glue ETL と AWS Glue Studio で利用できるようになりました。これにより、異常や検出が難しいデータ品質の問題を検出できるようになりました。
- 動的ルールでは、動的なしきい値 (例: `RowCount> avg(last(10))`) を設定できます

2024 年 3 月 12 日

- DQDL の改良点
  - NULL、BLANKS、WHITESPACES_ONLY などのキーワードのサポート
  - AWS Glue データ品質が複合ルールを処理する方法を指定するオプション
  - ColumnValues ルールタイプでは、比較中に NULL 値を渡すことはできません
  - DQDL での NOT 演算子のサポート

2024 年 6 月 26 日

- DQDL の改良点

- DQDL は Where 句のサポートを開始し、DQ ルールを適用する前にデータをフィルタリングできるようになりました

2024 年 8 月 7 日

- 異常検出と動的ルールの一般提供が開始

2024 年 11 月 22 日

- 複雑な複合ルールでは、ネストされたサポートを含む複雑なビジネスルールを作成することができます。
- ファイルのデータ品質を管理するための新しいルールタイプ
  - FileFreshness
  - FileSize
  - FileUniqueness
  - FileMatch
- Visual ETL ジョブのデフォルトのデータ品質チェック

2024 年 12 月 6 日

- AWS Glue Data Quality は、AWS Glue ETL 5.0 で Amazon SageMaker AI LakeHouse テーブルと AWS Lake Formation マネージド Iceberg、Delta、HUDI テーブルをサポートするようになりました。

2025 年 6 月 30 日

- 集計メトリクス – AWS Glue ETL ベースの DQ ジョブの API レベルで渡されたレコード数、失敗したレコード数などの集計メトリクスを取得できるようになりました。

AWS Glue Data Quality での異常検出

Data Quality における機械学習の事例

エンジニアは数百ものデータパイプラインを同時に管理します。各パイプラインはさまざまなソースからデータを抽出し、データレイクまたはその他のデータリポジトリに読み込むことができます。意思決定に役立つ高品質のデータが配信されるように、データ品質ルールを確立します。これらのルールは、現在のビジネス状態を反映した一定の基準に基づいてデータを評価します。ただし、ビジネス環境が変わると、データプロパティがシフトするため、これらの基準が古くなり、データ品質が低下します。

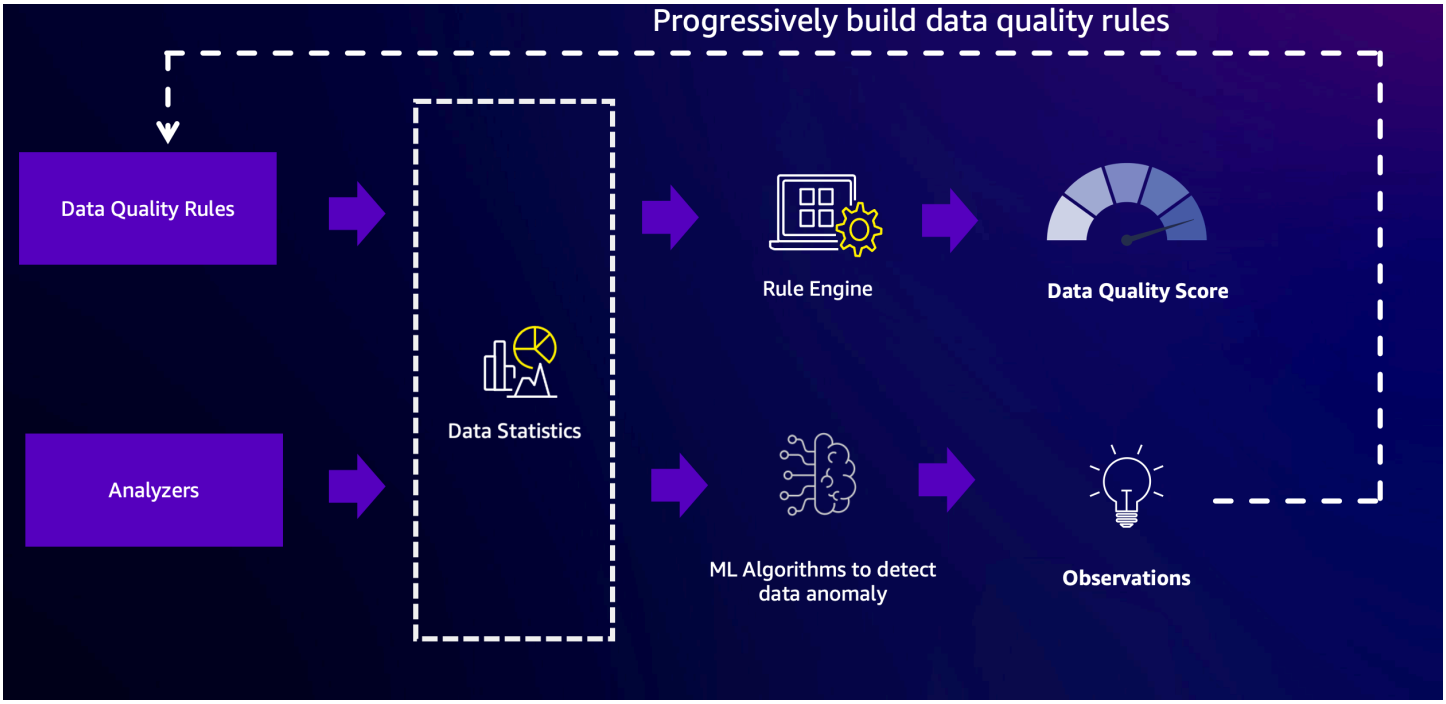
例えば、ある小売会社のデータエンジニアが、日次売上が 100 万 USD のしきい値を超えなければならない、という日次売上の検証ルールを設定したとします。その数か月後、日次売上高は 200 万 USD を超えたため、元のしきい値は適切な数値ではなくなりました。データエンジニアは、通知がなく、ルールを手動で分析および更新するために必要な労力が不足していたため、最新のしきい値を反映するルールへ更新ができませんでした。同月後半、ビジネスユーザーたちは売上が 25% 減少していることに気付きました。数時間の調査の結果、データエンジニアは、一部のストアからデータを抽出する ETL パイプラインが、エラーを生成せずに故障していたことを発見しました。ところが古いしきい値で確立したルールは正常に動作し続け、この問題を検出しませんでした。

あるいは、これらの異常を検出できるプロアクティブアラートがあれば、ユーザーがこの問題を検出できた可能性があります。さらに、ビジネスの季節性を追跡すると、データ品質に関する重大な問題が浮き彫りになったでしょう。例えば、小売売上は週末や祝祭日が最も高く、平日だと比較的低いと考えられます。このパターンからの逸脱は、データ品質の問題やビジネス状況の変化を示している可能性があります。データ品質ルールは、季節性をキャプチャして偏差を検出するために過去のパターンから学習できる高度なアルゴリズムが必要なため、季節性パターンを検出することはできません。

最後に、ルール作成プロセスの技術的な性質とルール作成にかかる時間が要因となり、ユーザーがルールの作成と維持を行うのは困難です。そのため、ユーザーはルールを定義する前に、まずデータインサイトを調べることを好みます。顧客は異常を簡単に特定できる能力が必要であり、それがあればデータ品質の問題をプロアクティブに検出して、自信を持ってビジネス上の意思決定を行うことができます。

仕組み

> **Note**
>
> 異常検出は AWS Glue ETL でのみサポートされています。これは、Data Catalog ベースのデータ品質ではサポートされていません。



AWS Glue Data Quality は、ルールベースのデータ品質と異常検出機能を組み合わせて、高品質のデータを提供します。開始するには、まずルールとアナライザーを設定し、異常検出を有効にする必要があります。

ルール

ルール - ルールは、データに対する期待値を、データ品質定義言語 (DQDL) と呼ばれるオープン言語で表します。次にルールの例を示します。このルールは、「passenger_count」列に空の値または NULL 値がない場合に成功します。

```
Rules = [
    IsComplete "passenger_count"
]
```

アナライザー

重要な列については理解しているが、特定のルールを書くためのデータについてよくわからない場合は、アナライザーを使用してそれらの列をモニタリングできます。アナライザーは、明示的なルールを定義せずにデータ統計を収集する方法です。アナライザーの設定例を以下に示します。

```
Analyzers = [
AllStatistics "fare_amount",
```

```
DistinctValuesCount "pulocationid",
RowCount
]
```

この例では、3 つのアナライザーが設定されています:

1. 最初のアナライザー `AllStatistics "fare_amount"` は、`fare_amount` フィールドで使用可能なすべての統計をキャプチャします。
2. 2 番目のアナライザー `DistinctValuesCount "pulocationid"` は、`pulocationid` 列の個別の値の数をキャプチャします。
3. 3 番目のアナライザー `RowCount` は、データセット内のレコードの総数をキャプチャします。

アナライザーは、複雑なルールを指定せずに関連するデータ統計を収集するシンプルな方法として役立ちます。これらの統計をモニタリングすることで、データ品質に関するインサイトを得て、さらなる調査や特定のルールの作成が必要になる可能性のある潜在的な問題や異常を特定できます。

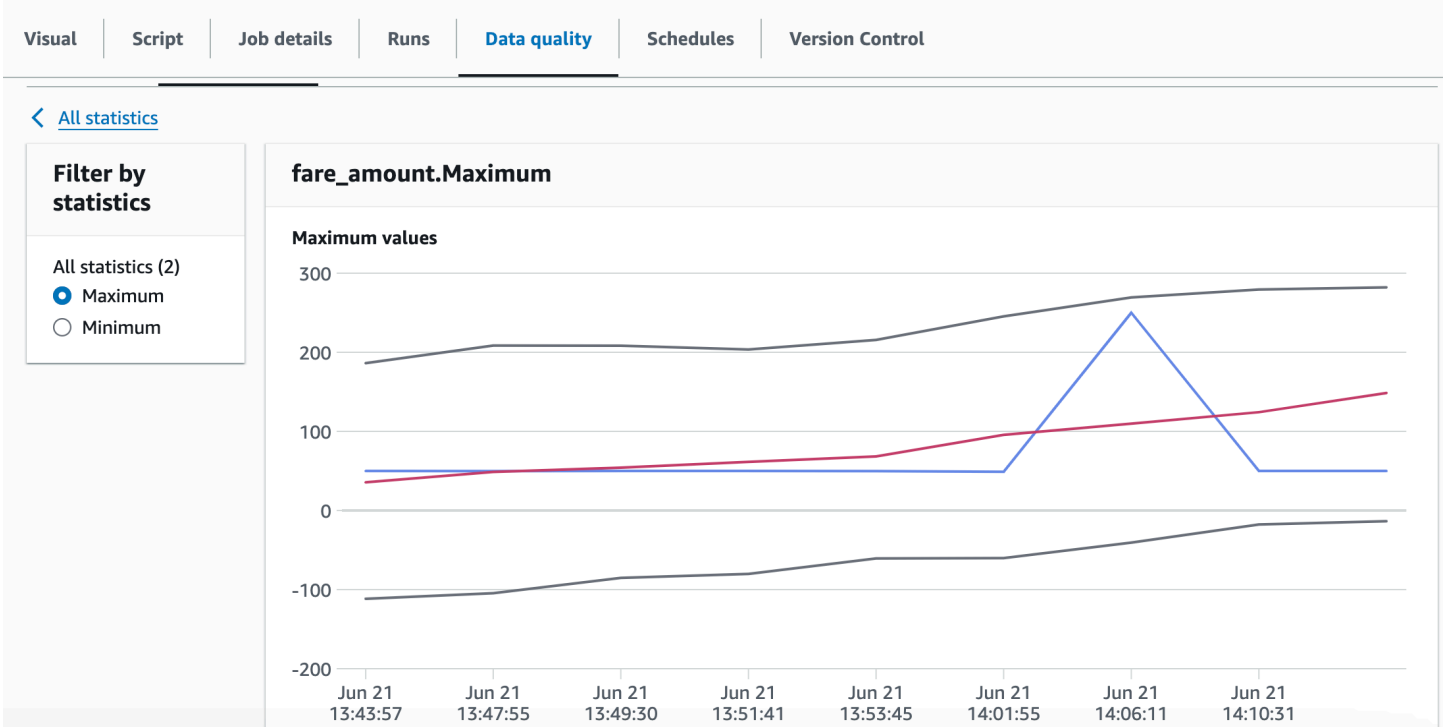
データ統計

AWS Glue Data Quality のアナライザーとルールはどちらも、データ統計 (データプロファイルとも呼ばれる) を収集します。これらの統計は、データの特性と品質に関するインサイトを提供します。収集された統計は AWS Glue サービス内に経時的に保存されるため、データプロファイルの変更を追跡および分析できます。

適切な API を呼び出すことで、これらの統計を簡単に取得してさらなる分析ができるほか、Amazon S3 に書き込むことで長期間保管できます。この機能を使用すると、データ処理ワークフローにデータプロファイリングを統合し、収集された統計をデータ品質モニタリング、異常検出など、さまざまな目的のために活用できます。

データプロファイルを Amazon S3 に保存することで、Amazon のオブジェクトストレージサービスのスケーラビリティ、耐久性およびコスト効率を活用できます。さらに、他の AWS サービスやサードパーティーのツールを使用してデータプロファイルを分析および視覚化できるため、データ品質に関するより深いインサイトを得て、データ管理とガバナンスについて情報に基づいた意思決定を行うことができます。

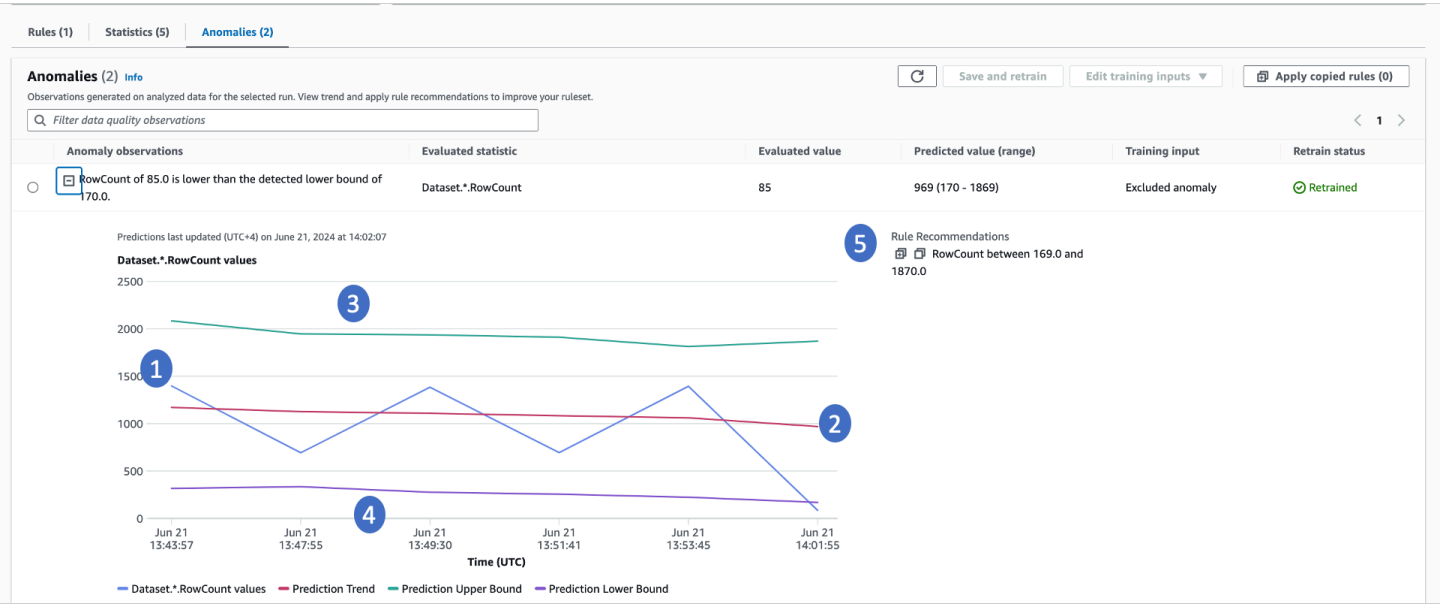
次は、経時的に保存されるデータ統計の例です。



> **Note**
>
> AWS Glue Data Quality は、同じ列に [ルール] と [アナライザー] の両方がある場合でも、統計を 1 回だけ収集するため、統計の生成プロセスが効率的になります。

異常検出

AWS Glue Data Quality では、異常を検出するために最低 3 つのデータポイントが必要です。機械学習アルゴリズムを使用して過去の傾向から学習し、将来の値を予測します。実際の値が予測範囲内にない場合、AWS Glue Data Quality は異常観測を作成します。実際の値と傾向が視覚的に表示されます。以下のグラフには 4 つの値が表示されます。



1. 経時的に表された実際の統計とその傾向。
2. 実際の傾向から学習することによって導出された傾向。傾向の方向性を理解するのに役立ちます。
3. 統計の上限可能値。
4. 統計の下限可能値。
5. これらの問題を将来検出できる推奨データ品質ルール。

異常に関して注意すべき重要な点がいくつかあります:

- 異常が生成されても、データ品質スコアは影響されません。
- 異常が検出されると、その後の実行では正常と見なされます。機械学習アルゴリズムは、明示的に除外されない限り、この異常値を入力と見なします。

再トレーニング

適切な異常を検出するには、異常検出モデルの再トレーニングが不可欠です。異常が検出されると、AWS Glue Data Quality はこの異常をモデルに通常の値として含めます。異常検出を正確に機能させるには、異常を承認または拒否してフィードバックを提供することが重要です。AWSGlue Data Quality では、モデルにフィードバックを提供するためのメカニズムが、AWS Glue Studio と API の両方で提供されています。詳細については、AWS Glue ETL パイプラインでの異常検出の設定に関するドキュメントを参照してください。

異常検出アルゴリズムの詳細

- 異常検出アルゴリズムは、経時的なデータ統計を調べます。このアルゴリズムは、使用可能なすべてのデータポイントを考慮し、明示的に除外された統計を無視します。
- これらのデータ統計は AWS Glue サービスに保存され、暗号化するための AWS KMS キーを指定できます。AWS Glue Data Quality 統計を暗号化するための AWS KMS キーの提供方法については、「セキュリティガイド」を参照してください。
- 時間コンポーネントは異常検出アルゴリズムにとって不可欠です。AWS Glue Data Quality は、過去の値に基づいて上限と下限を決定します。この値を決定するうえで、時間コンポーネントが考慮されます。同じ値でも、1 分間隔、1 時間間隔、または 1 日間隔では制限が異なります。

季節性の取得

AWS Glue Data Quality の異常検出アルゴリズムは、季節的なパターンをキャプチャできます。例えば、平日のパターンは週末のパターンとは異なることを理解できます。これは以下の例で確認できます。ここでは、AWS Glue Data Quality がデータ値の季節的な傾向を検出しています。この機能を有効にするために何か特定のことを行う必要はありません。時間の経過とともに、AWS Glue Data Quality は季節的な傾向を学習し、これらのパターンが破られたときに異常を検出します。



コスト

異常の検出にかかる時間に基づいて料金が課金されます。統計ごとに 1 DPU の料金が発生し、異常の検出にかかる時間に対して課金されます。詳細な例については、「AWS Glue の料金」を参照してください。

主な考慮事項

統計の保存に対して料金は発生しません。ただし、アカウントごとに保存できる統計は 100,000 に制限されています。これらの統計は最大 2 年間保存されます。

AWS Glue Data Quality の IAM アクセス許可の設定

このトピックでは、IAM 管理者が AWS Glue Data Quality 向けの AWS Identity and Access Management (IAM) ポリシーで使用できるアクションとリソースの理解に役立つ情報を説明します。また、AWS Glue データカタログで AWS Glue Data Quality を使用するために必要な最小限の権限を含むサンプル IAM ポリシーも含まれています。

AWS Glue のセキュリティに関する追加情報については、「「AWS Glue」 のセキュリティ」を参照してください。

AWS Glue Data Quality の IAM アクセス許可

次の表は、AWS Glue Data Quality で特定のオペレーションを実行する際にユーザーに必要になるアクセス許可を一覧化したものです。AWS Glue Data Quality の権限を詳細に設定するときは、IAM ポリシーステートメントの `Action` 要素でこれらのアクションを指定します。

AWS Glue Data Quality アクション

[アクション]	説明	リソースタイプ
`glue:CreateDataQualityRuleset`	品質評価のルールセットを、1 つ作成するためのアクセス許可を付与します。	`::dataQualityRuleset/<name>`
`glue:DeleteDataQualityRuleset`	品質評価のルールセットを、1 つ削除するためのアクセス許可を付与します。	`::dataQualityRuleset/<name>`

[アクション]	説明	リソースタイプ
glue:GetDataQualityRuleset	品質評価のルールセットを、1 つ取得するためのアクセス許可を付与します。	::dataQualityRuleset/<name>
glue:ListDataQualityRulesets	品質評価のルールセットを、すべて削除するためのアクセス許可を付与します。	::dataQualityRuleset/*
glue:UpdateDataQualityRuleset	品質評価のルールセットを、1 つ更新するためのアクセス許可を付与します。	::dataQualityRuleset/<name>
glue:GetDataQualityResult	品質評価タスクの実行結果を、1 つ取得するためのアクセス許可を付与します。この IAM アクションは、次の APIS への許可も付与します。 • BatchGetDataQualityQualityResult • ListDataQualityStatistics • ListDataQualityStatisticAnnotations	::dataQualityRuleset/<name>
glue:ListDataQualityResults	品質評価タスクの実行結果を、すべて取得するためのアクセス許可を付与します。	::dataQualityRuleset/*
glue:CancelDataQualityRuleRecommendationRun	実行中の品質評価推奨タスクを、1 つ停止するためのアクセス許可を付与します。	::dataQualityRuleset/*
glue:GetDataQualityRuleRecommendationRun	実行済みの品質評価推奨タスクを、1 つ取得するためのアクセス許可を付与します。	::dataQualityRuleset/*

[アクション]	説明	リソースタイプ
glue:ListDataQualityRuleRecommendationRuns	実行中の品質評価推奨タスクを、すべて取得するためのアクセス許可を付与します。	::dataQualityRuleset/*
glue:StartDataQualityRuleRecommendationRun	品質評価推奨タスクを、1 つ実行開始するためのアクセス許可を付与します。	::dataQualityRuleset/*
glue:CancelDataQualityRulesetEvaluationRun	実行中の品質評価タスクを、1 つ停止するためのアクセス許可を付与します。	::dataQualityRuleset/*
glue:GetDataQualityRulesetEvaluationRun	実行済みの品質評価タスクを、1 つ取得するためのアクセス許可を付与します。	::dataQualityRuleset/*
glue:ListDataQualityRulesetEvaluationRuns	実行済みの品質評価タスクを、すべて取得するためのアクセス許可を付与します。	::dataQualityRuleset/*
glue:StartDataQualityRulesetEvaluationRun	品質評価タスクを、1 つ実行開始するためのアクセス許可を付与します。	::dataQualityRuleset/<name>
glue:PublishDataQuality	データ品質の結果を公開する許可を付与します。	::dataQualityRuleset/<name>
glue:GetDataQualityModel	データ品質モデルを取得する許可を付与します。	::dataQualityRuleset/<name>, ::job/<name>
glue:GetDataQualityModelResult	データ品質モデルの結果を取得する許可を付与します。	::dataQualityRuleset/<name>, ::job/<name>

[アクション]	説明	リソースタイプ
glue:PutDataQualityStatisticAnnotation	統計に注釈を追加する許可を付与します。この IAM アクションは、次の APIS への許可も付与します。 • BatchPutDataQualityStatisticAnnotation	::dataQualityRuleset/<name>, ::job/<name>
glue:PutDataQualityProfileAnnotation	プロファイルのすべての統計に注釈を付ける許可を付与します。	::dataQualityRuleset/<name>, ::job/<name>

評価実行のスケジューリングに必要な IAM の設定

IAM アクセス許可

スケジュール済みの Data Quality 評価実行を実行するには、権限ポリシーに IAM:PassRole アクションを追加します。

AWS EventBridge スケジューラに必要なアクセス許可

[アクション]	説明	リソースタイプ
iam:PassRole	承認済みのロールをユーザーが渡すことを許可するアクセス許可を、IAM に付与します。	StartDataQualityRulesetEvaluationRun の呼び出しに使用されたロールの ARN

これらのアクセス許可がないと、次のエラーが発生します。

```
"errorCode": "AccessDenied"
"errorMessage": "User: arn:aws:sts::account_id:assumed-role/AWSGlueServiceRole is not
authorized to perform: iam:PassRole on resource: arn:aws:iam::account_id:role/service-
role/AWSGlueServiceRole
because no identity-based policy allows the iam:PassRole action"
```

IAM の信頼されたエンティティ

スケジュール済みの StartDataQualityEvaluationRun を作成して実行するには、AWS Glue と AWS EventBridge スケジューラサービスが、信頼されたエンティティに登録されている必要があります。

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

IAM ポリシーの例

AWS Glue Data Quality の IAM ロールには、次のタイプのアクセス許可が必要です。

- AWS Glue Data Quality オペレーションのアクセス許可。これにより、推奨されるデータ品質ルールを取得したり、AWS Glue データカタログ内のテーブルに対してデータ品質タスクを実行したりできます。このセクションの IAM ポリシーの例には、AWS Glue Data Quality オペレーションに必要な最低限のアクセス許可が含まれています。
- データカタログテーブルとその基盤となるデータへのアクセス許可。これらのアクセス許可は、ユースケースによって異なります。例えば、Amazon S3 でカタログ化するデータの場合、アクセス許可には Amazon S3 へのアクセスを含める必要があります。

> **Note**
>
> そのため、このセクションで説明するアクセス許可に加えて、Amazon S3 のアクセス許可を設定する必要があります。

推奨データ品質ルールの取得に最低限必要なアクセス許可

このサンプルポリシーには、推奨データ品質ルールを生成するために必要なアクセス許可が含まれています。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowGlueRuleRecommendationRunActions",
      "Effect": "Allow",
      "Action": [
        "glue:GetDataQualityRuleRecommendationRun",
        "glue:PublishDataQuality",
        "glue:CreateDataQualityRuleset"
      ],
      "Resource": "arn:aws:glue:us-east-1:111122223333:dataQualityRuleset/*"
    },
 {
     "Sid": "AllowCatalogPermissions",
     "Effect": "Allow",
     "Action": [
        "glue:GetPartitions",
        "glue:GetTable"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "AllowS3GetObjectToRunRuleRecommendationTask",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
```

```
      "Resource": "arn:aws:s3:::aws-glue-*"
    },
 {   // Optional for Logs
        "Sid": "AllowPublishingCloudwatchLogs",
        "Effect": "Allow",
        "Action": [
          "logs:CreateLogStream",
          "logs:CreateLogGroup",
          "logs:PutLogEvents"
        ],
        "Resource": "*"
    },
  ]
}
```

品質評価タスクを実行するために最低限必要なアクセス許可

このサンプルポリシーには、データ品質評価タスクを実行するために必要なアクセス許可が含まれています。

以下のポリシーステートメントはオプションで、ユースケースに応じて設定します。

- `AllowCloudWatchPutMetricDataToPublishTaskMetrics` - Amazon CloudWatchにデータ品質の実行メトリクスを発行する場合は必須です。
- `AllowS3PutObjectToWriteTaskResults` - Amazon S3 にデータ品質の実行結果を書き込む場合は必須です。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowGlueGetDataQualityRuleset",
      "Effect": "Allow",
      "Action": [
        "glue:GetDataQualityRuleset"
      ],
      "Resource": "arn:aws:glue:us-east-1:111122223333:dataQualityRuleset/<YOUR-
RULESET-NAME>"
    },
    {
      "Sid": "AllowGlueRulesetEvaluationRunActions",
```

```
      "Effect": "Allow",
      "Action": [
        "glue:GetDataQualityRulesetEvaluationRun",
        "glue:PublishDataQuality"
      ],
      "Resource": "arn:aws:glue:us-east-1:111122223333:dataQualityRuleset/*"
    },
{
      "Sid": "AllowCatalogPermissions",
      "Effect": "Allow",
      "Action": [
        "glue:GetPartitions",
        "glue:GetTable"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "AllowS3GetObjectForRulesetEvaluationRun",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": "arn:aws:s3:::aws-glue-*"
    },
    {
      "Sid": "AllowCloudWatchPutMetricDataToPublishTaskMetrics",
      "Effect": "Allow",
      "Action": [
        "cloudwatch:PutMetricData"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "cloudwatch:namespace": "Glue Data Quality"
        }
      }
    },
    {
      "Sid": "AllowS3PutObjectToWriteTaskResults",
      "Effect": "Allow",
      "Action": [
        "s3:PutObject*"
```

```
      ],
      "Resource": "arn:aws:s3:::<YOUR-BUCKET-NAME>/*"
    }
  ]
}
```

データ品質 ETL ジョブを実行するために最低限必要なアクセス許可

このサンプルポリシーには、データ品質 ETL ジョブを実行するために必要なアクセス許可が含まれています。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowGluePublishDataQualityResult",
      "Effect": "Allow",
      "Action": [
        "glue:PublishDataQuality"
      ],
      "Resource": "arn:aws:glue:us-east-1:111122223333:dataQualityRuleset/*"
    },
    //Optional to retrieve results, observation generation,
    //dynamic rules and DetectAnomalies
    {
      "Sid": "AllowGlueGetDataQualityResult",
      "Effect": "Allow",
      "Action": [
        "glue:GetDataQualityResult"
      ],
      "Resource": "arn:aws:glue:us-east-1:111122223333:dataQualityRuleset/*"
    },
    //Optional to allow annotating statistics
    {
      "Sid": "AllowGlueDataQualityStatisticAnnotation",
      "Effect": "Allow",
      "Action": [
        "glue:PutDataQualityStatisticAnnotation"
      ],
      "Resource": [
        "arn:aws:glue:us-east-1:111122223333:dataQualityRuleset/*",
        "arn:aws:glue:us-east-1:111122223333::job/{JobName}"
      ]
```

```
    },
    //Optional to allow annotating all statistics in a profile
    {
      "Sid": "AllowGlueDataQualityProfileAnnotation",
      "Effect": "Allow",
      "Action": [
        "glue:PutDataQualityProfileAnnotation"
      ],
      "Resource": [
        "arn:aws:glue:us-east-1:111122223333:dataQualityRuleset/*",
        "arn:aws:glue:us-east-1:111122223333::job/{JobName}"
      ]
    }
}
```

Data Catalog で AWS Glue Data Quality を使用する

この開始方法セクションでは、AWS Glue Data Quality コンソールから AWS Glue を使用する際に役立つ手順をご紹介します。品質評価ルールの推奨事項の生成や、データに照らしたルールセットの評価など、重要なタスクを完了する方法について確認できます。

トピック

- 前提条件
- 手順の例
- ルールの推奨事項の生成
- ルールの推奨のモニタリング
- 推奨ルールセットの編集
- 新しいルールセットの作成
- ルールセットを実行してデータ品質を評価する
- データ品質スコアと結果を表示する
- 関連トピック

前提条件

AWS Glue Data Quality を使用する前に、Data Catalog と AWS Glue のクローラーの使用に慣れておく必要があります。AWS Glue Data Quality を使用すれば、Data Catalog データベースのテーブルの品質を評価できます。また、以下も必要になります:

- データ品質ルールセットの評価時に対照する Data Catalog のテーブル。
- ルールの推奨事項の生成やデータ品質タスクの実行時に、AWS Glue が使用する IAM ロール。このロールには、さまざまな AWS Glue Data Quality プロセスをユーザーに代わって実行するための、必要なリソースにアクセスする権限が必要です。この対象となるリソースには AWS Glue、Amazon S3、および CloudWatch が含まれます。AWS Glue Data Quality の最低限必要なアクセス許可を含むポリシーの例については、「IAM ポリシーの例」を参照してください。

  AWS Glue 用の IAM ロール の詳細については、「AWS Glue サービスの IAM ポリシーを作成する」と「AWS Glue 用の IAM ロールを作成する」を参照してください。「Authorization for AWS Glue Data Quality actions」のページでは、Data Quality 専用の AWS Glue アクセス許可すべてのリストもご覧いただけます。
- さまざまなデータを含むテーブルが 1 つ以上あるデータベース。このチュートリアルで使用するテーブルには `yyz-tickets` という名前が付いており、テーブル `tickets` が付随しています。このデータは、トロント市が公開している駐車違反切符の情報を集めたものです。独自のテーブルを作成するときは、最適な推奨ルールを設定できるよう、さまざまな有効なデータがそこに入力されていることを確認します。

手順の例

サンプルのデータセットを使用した手順の例については、AWS Glue Data Quality に関するブログ記事を参照してください。

ルールの推奨事項の生成

ルールの推奨事項を利用すると、コードを記述することなく、すぐに Data Quality の使用を始められます。AWS Glue Data Quality を使用すると、データの分析、ルールの特定、品質評価タスクで評価されるルールセットの作成を行えます。推奨事項の実行は、実行から 90 日後に自動的に削除されます。

品質評価ルールの推奨事項を生成するには

1. AWS Glue コンソール (https://console.aws.amazon.com/glue/) を開きます。
2. ナビゲーションペインで、[Tables] (テーブル) を選択します。次に、品質評価ルールの推奨事項を生成する対象となるテーブルを選択します。
3. テーブルの詳細ページで、[Data quality] タブを選択し、そのテーブルの AWS Glue Data Quality ルールにアクセスします。
4. [Data quality] タブで、[Add rules and monitor data quality] を選択します。

5. [Ruleset builder] ページで、ページ上部のアラートに、「ルールの推奨事項が実行されていない場合は推奨事項タスクを開始する」との指示が表示されます。
6. [Recommend rules] を選択してモーダルを開き、推奨事項タスクのパラメータを入力します。
7. AWS Glue へのアクセス許可を持つ IAM ロールを選択します。このロールには、さまざまな AWS Glue Data Quality プロセスを、ユーザーに代わって実行するために必要なリソースに、アクセスするための許可が必要です。
8. 希望する設定に従ってフィールドに入力したら、[Recommend rules] を選択し、推奨事項タスクの実行を開始します。推奨事項の実行が進行中の場合、または完了した場合は、こちらのアラートで実行を管理できます。ステータスの変化を確認する際にページの更新が必要になる場合があります。完了済みと進行中の推奨事項タスクの実行状況は [実行履歴] のページに表示されます。ここには過去 90 日分の推奨事項の実行がすべて表示されます。

推奨ルールの意味

AWS Glue Data Quality は、入力テーブルの各列のデータに基づいてルールを生成します。このルールを使用して、品質要件を維持するために、データのフィルタリングが可能な潜在的境界を特定します。以下は生成されたルールのリストです。ルールの意味と、このルールがデータに適用されたときどのような処理が行われるのか、理解するのに役立つ例が記載されています。

生成されたデータ品質定義言語 (DQDL) のルールタイプの一覧は、「データ品質定義言語 (DQDL)」を参照してください。

- `IsComplete "SET_FINE_AMOUNT"` – 特定の行に対して列が入力されていることを確認する `IsComplete` ルールです。データ内で列を非オプションとしてタグ付けするときにこのルールを使用します。
- `Uniqueness "TICKET_NUMBER" > 0.95` – 列内のデータが一意のしきい値を満たしていることを確認する `Uniqueness` ルールです。この例では、`"TICKET_NUMBER"` の特定の行に入力されたデータは他のすべての行と内容が最大 95% 同じであると判断されましたが、これはこのルールを示唆しています。
- `ColumnValues "PROVINCE" in ["ON", "QC", "AB", "NY",...]` – `ColumnValues` ルールは、既存の列の内容に基づいて、この列の有効な値を定義します。この例では、各行のデータは州または地域の 2 文字のナンバープレートコードです。
- `ColumnLength "INFRACTION_DESCRIPTION" between 15 and 31` — `ColumnLength` ルールは、列のデータに文字数の制限を適用します。このルールは、文字列の列に記録された最小長と最大長に基づき、サンプルデータから生成されます。

ルールの推奨のモニタリング

品質評価ルールの推奨事項を実行すると、[Add rules and monitor data quality] ページに、情報と、上部のバーで実行できる追加のアクションが表示されます。

ルールの推奨が進行中の場合、推奨タスクが完了する前に [実行を停止] を選択できます。タスクが進行中の場合、ステータスは [in progress] と表示され、実行が開始された日付と時刻が表示されます。

ルールの推奨が完了すると、ルール推奨のバーに、推奨されたルールの数、最後の推奨実行のステータス、完了した日付とタイムスタンプが表示されます。

推奨ルールを追加するときは、[Insert Rule Recommendation] を選択します。過去に推奨されたルールを閲覧するには、具体的な日付を選択します。新しい推奨を実行するには、[More actions] を選択して、次に [Recommended rules] を選択します。

デフォルトの設定は、[Manage user settings] を選択して設定します。Amazon S3 のデフォルトパスを設定すると、ルールセットを保存したり、データカタログを実行するためのデフォルトロールを設定したりできます。

推奨ルールセットの編集

AWS Glue Data Quality は利用可能な既存のデータに基づいてルールを生成するため、自動化された提案に予期せぬルールや望ましくないルールが表示される場合があります。推奨ルールセットを最大限活用するには、ルールセットを評価し、修正を加えることが必要になります。本チュートリアルのこちらのステップでは、前回のステップで生成したルールを修正し、一部のデータに、より制限の厳しい条件を適用します。また、他のルールの制限を緩和し、正確な一意のデータを後で追加できるようにすることも可能です。

推奨されたルールセットを編集する

1. AWS Glue コンソールで [データカタログ] を選択し、次にナビゲーションペインで [Databases tables] を選択します。テーブル `tickets` を選択します。
2. テーブルの詳細ページで、[Data quality] タブを選択し、そのテーブルの AWS Glue Data Quality のルールと設定にアクセスします。
3. [Rulesets] セクションで、ルールの推奨事項の生成 で生成したルールセットを選択します。
4. [アクション] を選択し、次にコンソールウィンドウで [編集] を選択します。ルールセットエディタがコンソールにロードされます。ここには、ルールの編集用ペインと DQDL のクイックリファレンスが含まれています。

5. スクリプトの 2 行目を削除します。これにより、データベースのサイズを特定の行数に制限する条件が緩和されます。編集後、ファイルの 1～3 行目に以下の内容が含まれているはずです。

```
Rules = [
    IsComplete "TAG_NUMBER_MASKED",
    ColumnLength "TAG_NUMBER_MASKED" between 6 and 9,
```

6. スクリプトの 25 行目を削除します。これにより、記録された州の 96% が ON であるという条件は緩和されます。編集後、ファイルの 24 行目からルールセットの最後までに以下の内容が含まれているはずです。

```
ColumnValues "PROVINCE" in ["ON", "QC", "AB", "NY", "AZ", "NS", "BC", "MI", "PQ",
 "MB", "PA", "FL", "SK", "NJ", "OH", "NB", "IL", "MA", "CA",
    "VA", "TX", "NF", "MD", "PE", "CT", "NC", "GA", "IN", "OR", "MN", "TN", "WI",
 "KY", "MO", "WA", "NH", "SC", "CO", "OK", "VT", "RI", "ME", "AL",
    "YT", "IA", "DE", "AR", "LA", "XX", "WV", "MT", "KS", "NT", "DC", "NV", "NE",
 "UT", "MS", "NM", "ID", "SD", "ND", "AK", "NU", "GO", "WY", "HI"],
ColumnLength "PROVINCE" = 2
]
```

7. 14 行目を以下の内容に変更します。

```
IsComplete "TIME_OF_INFRACTION",
```

これにより、データベースを違反時間が記録されたチケットのみに制限して、列の条件を強化します。違反時間が記録されていないチケットは、このデータセットでは常に無効なデータとみなされます。これは、さらなるデータ使用や品質ルールを決定する際の調査など、パーティショニングや変換の方が適している場合とは異なります。

8. コンソールページの下部で [Update Ruleset] を選択します。

新しいルールセットの作成

ルールセットは、データに対応させて評価するための品質評価ルールをグループ化したものです。AWS Glue コンソールでは、データ品質定義言語 (DQDL) を使用してカスタムルールセットを作成できます。

品質評価ルールセットを作成するには

1. AWS Glue のコンソールで [データカタログ] を選択し、次に [データベース] を選択し、ナビゲーションペインで [テーブル] を選択します。テーブル `tickets` を選択します。
2. [Data quality] (データ品質) タブを開きます。
3. [Rulesets] セクションで、[Create ruleset] を選択します。DQDL エディタがコンソールで起動します。ここには、直接編集できるテキスト領域と、DQDL ルールおよびテーブルスキーマのクイックリファレンスがあります。
4. DQDL エディタのテキスト領域にルールを追加していきます。チュートリアルから直接ルールを記述することもできますし、データ品質ルールエディタの DQDL ルールビルダー機能を使用することもできます。

   > **Note**
   >
   > DQDL ルールビルダーの使用方法
   >
   > 1. リストからルールタイプを選択し、プラス記号を選択してサンプルの構文をエディタペインに挿入します。
   > 2. プレースホルダーの列名を、実際の列名で置き換えます。テーブルの列名は、[Create ruleset] タブで入手できます。
   > 3. 必要に応じて式のパラメータを更新します。DQDL がサポートする式の完全なリストは、「表現」でご確認ください。

   一例を挙げると、次のルールは `tickets` テーブルの `ticket_number` 列のデータ検証に関する制約です。次のルールを追加するには、DQDL ルールビルダーを使用するか、ルールセットを直接編集します。

   ```
   IsComplete "ticket_number",
   IsUnique "ticket_number",
   ColumnValues "ticket_number" > 9000000000
   ```

5. [Run name] フィールドに新しいルールセットの名前を入力します。
6. [Save ruleset] を選択します。

複数のデータセットのデータ品質を評価する

ReferentialIntegrity ルールセットと DatasetMatch ルールセットを使用することで、複数のデータセットにデータ品質のルールを設定できます。ReferentialIntegrity は、プライマリデータセット内のデータが他のデータセット内に存在するかをチェックします。

参照のデータセットを追加するときは、[スキーマ] タブを選択し、次に [Update reference tables] を選択します。データベースとテーブルを選択するよう要求されます。テーブルを追加したら、データ品質ルールを設定できます。AggregateMatch、RowCountMatch、ReferentialIntegrity、SchemaMatch、DatasetMatch などのルールタイプは、複数のデータセットを横断してデータ品質チェックを実行する機能をサポートしています。

ルールセットを実行してデータ品質を評価する

品質評価タスクを実行すると、AWS Glue Data Quality はデータと対応させてルールセットを評価し、データ品質のスコアを計算します。このスコアは、入力に対して合格したデータ品質ルールの割合を表します。

品質評価タスクを実行するには

1. AWS Glue のコンソールで [データカタログ] を選択し、次に [データベース] を選択し、ナビゲーションペインで [テーブル] を選択します。テーブル `tickets` を選択します。
2. [Data quality] タブを選択します。
3. [Rulesets] リストから、テーブルに照らして評価を行うルールセットを選択します。このステップでは、生成されたルールではなく、自分で既に作成または変更したルールセットを使用することが推奨されます。[Run] (実行) を選択します。
4. モーダルで、自分の IAM ロールを選択します。このロールには、さまざまな AWS Glue Data Quality プロセスを、ユーザーに代わって実行するために必要なリソースに、アクセスするための許可が必要です。この IAM ロールをデフォルトとして保存するか、[Default Setting] のページに進んでこれを変更します。
5. [Data quality actions] (データ品質アクション) で、必要に応じて [Publish metrics to Amazon CloudWatch] (Amazon CloudWatch にメトリクスを公開する) を選択します。これを選択した場合、AWS Glue Data Quality は、評価に合格したルールと不合格だったルール、それぞれの数についてのメトリクスをパブリッシュします。この方法で保存されたメトリクスに対してアクションを実行するときは、CloudWatch アラームを使用します。アラートの設定用に、主要なメトリクスも Amazon EventBridge に公開されます。詳細については、「Setting up alerts, deployments, and scheduling」を参照してください。

6. [Run Frequency] で、[オンデマンドで実行する] または [Schedule the ruleset] を選択します。ルールセットをスケジュールした場合、タスク名を入力するよう求められます。スケジュールは Amazon EventBridge で作成されます。スケジュールの編集は Amazon EventBridge で行えます。
7. データ品質評価の結果を Amazon S3 に保存するには、[Data quality results location] を選択します。このタスクのために先に選択した IAM ロールには、この口ケーションに対する書き込みのアクセス権限が必要です。
8. [Additional Configurations] の [Requested number of workers] に、AWS Glue でデータ品質タスクに割り当てる数を入力します。
9. オプションで、データソースにフィルターを設定できます。これにより、読み取るデータを減らせます。また、フィルターを使用して、パーティション情報を選択し、これを API 呼び出しを介してパラメータとして渡すことにより、増分検証を実行することもできます。パフォーマンスを改善するには、パーティション述語を指定します。
10. [Run] (実行) を選択します。[Data quality task runs] (実行中のデータ品質タスク) リストに、新しいタスクが表示されます。タスクの [実行ステータス] 列に [完了] と表示されたら、品質スコアの結果を確認できます。最新のステータスを確認するため、コンソール画面の更新が必要になる場合があります。
11. 品質評価結果の詳細の列を表示するには、[+] アイコンをクリックしてルールセットを展開します。結果には、評価に合格したルールと不合格だったルール、そして不合格の原因が表示されます。

データ品質スコアと結果を表示する

作成された全ルールセットの最新の実行を確認するには

1. AWS Glue コンソールのナビゲーションペインで、[Tables] (テーブル) を選択します。次に、品質評価タスクを実行する対象のテーブルを選択します。
2. [Data quality] タブを選択します。
3. [Data quality snapshot] には、時間の経過に伴う実行の一般的傾向が示されています。デフォルトでは、すべてのルールセットにおける最後の 10 回の実行が表示されます。ルールセットごとにフィルタリングするには、ドロップダウンリストから希望するルールセットを選択します。実行数が 10 回未満の場合は、完了した実行のうち表示可能なものがすべて表示されます。
4. [Data quality] テーブルには、各ルールセットが、最新の実行 (もしある場合) のスコアと共に表示されます。ルールセットを展開すると、そのルールセットに含まれるルールが、その実行のルール結果と併せて表示されます。

特定のルールセットにおける最新の実行を確認するには

1. AWS Glue コンソールのナビゲーションペインで、[Tables] (テーブル) を選択します。次に、品質評価タスクを実行する対象のテーブルを選択します。
2. [Data quality] タブを選択します。
3. [Data quality] テーブルで、特定のルールセットを選択します。
4. [Ruleset details] ページで、[Run history] タブを選択します。

   この特定のルールセットに対するすべての評価実行が、このタブ内のテーブルに一覧表示されます。スコアの履歴と実行のステータスを確認できます。
5. 特定の実行に関する詳細情報を表示するには、[実行 ID] を選択して [Evaluation run details] のページに進みます。このページでは、実行の詳細と、個々のルール結果のステータスに関する詳細を確認できます。

関連トピック

- DQDL ルールタイプリファレンス
- データ品質定義言語 (DQDL) リファレンス

AWS Glue Studio を使用したデータ品質の評価

AWS Glue Data Quality は、定義したルールに基づき、データ品質の評価とモニタリングを行います。これにより、アクションが必要なデータを簡単に特定できます。AWS Glue Studio では、ビジュアルジョブにデータ品質ノードを追加して、データカタログ内のテーブルにデータ品質ルールを作成できます。これにより、経時的に進化するデータセットの変化をモニタリングし、評価できます。AWS Glue Studio で AWS Glue Data Quality を操作する方法の概要については、次の動画を参照してください。

AWS Glue Data Quality を使用する手順の概要を次に示します。

1. データ品質ルールの作成 – 設定した組み込みルールセットを選択して、DQDL ビルダーを使用してデータ品質ルールのセットを作成します。
2. データ品質ジョブの設定 – データ品質結果と出力オプションに基づいてアクションを定義します。

3. [Save and run a data quality job] – ジョブを作成して実行します。ジョブを保存すると、そのジョブ用に作成したルールセットが保存されます。
4. データ品質結果のモニタリングとレビュー – ジョブの実行が完了した後にデータ品質結果をレビューします。必要に応じて、ジョブを将来の日付にスケジュールすることもできます。

利点

データアナリスト、データエンジニア、データサイエンティストは、AWS Glue Studio でデータ品質評価ノードを使用し、ビジュアルジョブエディタでデータ品質を分析、設定、モニタリング、改善できます。データ品質ノードの使用には、次のような利点があります。

- データ品質の問題を検出可能 - データセットの特性をチェックするルールを作成することで、問題を確認できます。
- 簡単に開始可能 - 事前構築済みのルールとアクションで開始できます。
- 緊密な統合 - AWS Glue Data Quality は AWS Glue データカタログ上で実行されるため、AWS Glue Studio でデータ品質ノードを使用できます。

AWS Glue Studio で ETL ジョブのデータ品質を評価する

このチュートリアルでは、AWS Glue Studio で AWS Glue Data Quality の使用を開始します。次の方法について説明します。

- データ品質定義言語 (DQDL) ルールビルダーを使用して、ルールを作成します。
- データ品質アクション、出力するデータ、およびデータ品質結果を出力するロケーションを指定します。
- データ品質結果を確認する。

例を使用して練習するには、ブログ記事「Getting started with AWS Data Quality for ETL pipelines」を参照してください。

ステップ 1: データ品質評価変換ノードをビジュアルジョブに追加する

このステップでは、ビジュアルジョブエディタに、データ品質評価ノードを追加します。

データ品質ノードを追加するには

1. AWS Glue Studio コンソールで、[ジョブを作成する] セクションから [Visual with a source and target] を選択し、続いて [作成] を選択します。
2. データ品質変換を適用するノードを選択します。通常これは、変換ノードまたはデータソースです。
3. [+] アイコンを選択して、左側のリソースパネルを開きます。検索バーでデータ品質評価を検索し、取得した検索結果からデータ品質評価を選択します。
4. ビジュアルジョブエディタは、選択したノードから分岐したデータ品質評価変換ノードを表示します。コンソールの右側で、自動的に [Transform] (変換) タブが開きます。親ノードを変更する必要がある場合は、[Node properties] タブを選択し、ドロップダウンメニューで親ノードを選択します。

   ノードの親を新たに選択すると、この親ノードと データ品質評価ノード間に、新しい接続が確立されます。不要な親ノードをすべて削除します。1 つの データ品質評価ノードに接続できるのは、親ノード 1 つだけです。
5. データ品質評価変換は複数の親をサポートしているため、複数のデータセットにわたってデータ品質ルールを検証できます。複数のデータセットをサポートするルールには、ReferentialIntegrity、DatasetMatch、SchemaMatch、RowCountMatch、AggregateMatch などがあります。

   データ品質評価変換に複数の入力を追加する場合は、「プライマリ」入力を選択する必要があります。プライマリ入力は、データ品質を検証するデータセットです。他のすべてのノードまたは入力は、参照として扱われます。

   データ品質評価変換を使用して、データ品質チェックで不合格になった特定のレコードを識別できます。不良レコードにフラグを立てる新しい列がプライマリデータセットに追加されるため、プライマリデータセットを選択することをお勧めします。
6. 入力データソースのエイリアスを指定できます。ReferentialIntegrity ルールを使用する場合、エイリアスを使用して入力ソースを参照する別の方法があります。プライマリソースとして指定できるデータソースは 1 つだけのため、追加するデータソースにはそれぞれエイリアスが必要です。

   次の例では、ReferentialIntegrity ルールによってエイリアス名で入力データソースを指定し、プライマリデータソースと 1 対 1 の比較を実行します。

```
Rules = [
```

```
 ReferentialIntegrity “Aliasname.name” = 1
]
```

ステップ 2: DQDL を使用してルールを作成する

このステップでは、DQDL を使用してルールを作成します。このチュートリアルでは、完全性ルールタイプを使用して 1 つのルールを作成します。このルールタイプは、特定の表現に対して列内の完全な (null 以外の) 値の割合を確認します。DQDL の使用の詳細については、「DQDL」を参照してください。

1. [変換] タブで、[Insert] ボタンを選択し、[Rule type] を追加します。これにより、ルールタイプがルール エディタに追加されるため、そこでルールのパラメータを入力できます。

   > **Note**
   >
   > ルールを編集する際は、ルールがかっこ内にあり、互いにカンマで区切られていることを確認してください。以下は、完全なルール表現の例です。
   >
   > ```
   > Rules= [
   >     Completeness "year">0.8, Completeness "month">0.8
   > ]
   > ```
   >
   > この例では、「年」と「月」の名前が付いた列に対して、完全性に関するパラメータを指定しています。このルールがパスするには、これらの列の完全性が 80% を超えているか、各列で 80% を超えるインスタンスにデータが含まれている必要があります。

   この例では、[Completeness] (完全性) のルールタイプを検索して挿入します。これにより、ルールタイプがルール エディタに追加されます。このルールタイプの構文は、`Completeness <COL_NAME> <EXPRESSION>` の形式を取ります。

   ほとんどのルールタイプにおいて、ブール型の応答を作成する場合は、パラメータとして表現を指定する必要があります。サポートされている DQDL 表現の詳細については、「DQDL expressions」を参照してください。次に、列名を追加します。

2. DQDL ルール ビルダーで、[スキーマ] タブを選択します。検索バーを使用して、入力スキーマの中の列名を特定します。入力スキーマには、列名とデータ型が表示されます。

3. ルールエディターで、ルールタイプの右側をクリックして、列を挿入する場所にカーソルを移動します。または、ルールの中に列名を入力することもできます。

   例えば、入力スキーマリストの列リストで、列の横にある [Insert] ボタンを選択します (この例では [year])。これにより、列がルールに追加されます。

4. 次に、ルールエディターで、ルールを評価するための表現を追加します。完全性ルールタイプは特定の表現に対して列内の完全な (null 以外の) 値の割合を確認するため、`> 0.8` のような値を入力します。このルールは、列に完全な値 (null 以外) が 80% を超える割合で含まれているかどうかを確認します。

ステップ 3: データ品質出力の設定

データ品質ルールを作成したら、データ品質ノード出力を指定する追加のオプションを選択できます。

1. [Data quality transform output] (データ品質変換の出力) で、次のオプションから選択します。

   - [Original data] – 元の入力データを出力することを選択します。このオプションを選択すると、新しい子ノード「rowLevelOutcomes」がジョブに追加されます。スキーマは、変換の入力として渡されたプライマリデータセットのスキーマと一致します。このオプションは、品質の問題が発生したときに、データを渡してジョブを失敗させる場合に便利です。

     別のユースケースとしては、データ品質チェックに失敗した不良レコードを検出する場合があります。不良レコードを検出するには、オプション [Add new columns to indicate data quality errors] を選択します。このアクションにより、「rowLevelOutcomes」変換のスキーマに 4 つの新しい列が追加されます。

     - DataQualityRulesPass (文字列配列) — データ品質チェックに合格したルールの配列を提供します。
     - DataQualityRulesFail (文字列配列) — データ品質チェックに失敗したルールの配列を提供します。
     - DataQualityRulesSkip (文字列配列) — スキップされたルールの配列を提供します。次のルールはデータセットレベルで適用されるため、エラーレコードを識別できません。
       - AggregateMatch
       - ColumnCount
       - ColumnExists
       - ColumnNamesMatchPattern

- CustomSql
- RowCount
- RowCountMatch
- StandardDeviation
- Mean
- ColumnCorrelation

- DataQualityEvaluationResult — 行レベルで「合格」または「不合格」のステータスを表示します。全体的な結果は不合格でも、特定の記録は合格である可能性があることに注意してください。例えば、RowCount ルールは失敗でも、その他のルールはすべて成功である可能性があります。このような場合、このフィールドのステータスは「合格」です。

2. [Data quality results] – 設定したルールと、それに関するステータス (合格/不合格) を出力することを選択します。このオプションは、結果を Amazon S3 またはその他のデータベースに書き込む場合に便利です。
3. [Data quality output settings] (オプション) – [Data quality output settings] を選択し、[Data quality result location] フィールドを表示します。続いて [参照] を選択し、データ品質の出力先として設定する Amazon S3 の場所を検索します。

ステップ 4. データ品質アクションの設定

このアクションを使用すると、特定の基準に基づいてメトリクスを CloudWatch に公開したり、ジョブを停止したりできます。アクションは、ルールを作成した後にのみ使用できます。このオプションを選択すると、同じメトリクスが Amazon EventBridge にも公開されます。このオプションを使用すると、通知用アラートを作成できます。

- [On ruleset failure] — ジョブ実行中にルールセットが失敗した場合の対処方法を選択できます。データ品質が低下した場合にジョブを失敗させる場合は、次のいずれかのオプションを選択して、ジョブを失敗させるタイミングを選択します。デフォルトではこのアクションは選択されておらず、データ品質ルールが失敗した場合でもそのジョブの実行は完了します。
  - [なし] — [なし] (デフォルト) を選択した場合、ジョブは失敗せず、ルールセットが失敗しても実行が継続されます。
  - [ターゲットにデータをロードした後にジョブを失敗させる] — ジョブは失敗し、データは保存されません。結果を保存するには、データ品質の結果を保存する Amazon S3 の場所を選択します。

- [Fail job without loading to target data] — このオプションは、データ品質エラーが発生するとすぐにジョブを失敗させます。データ品質変換の結果を含め、データターゲットはロードされません。

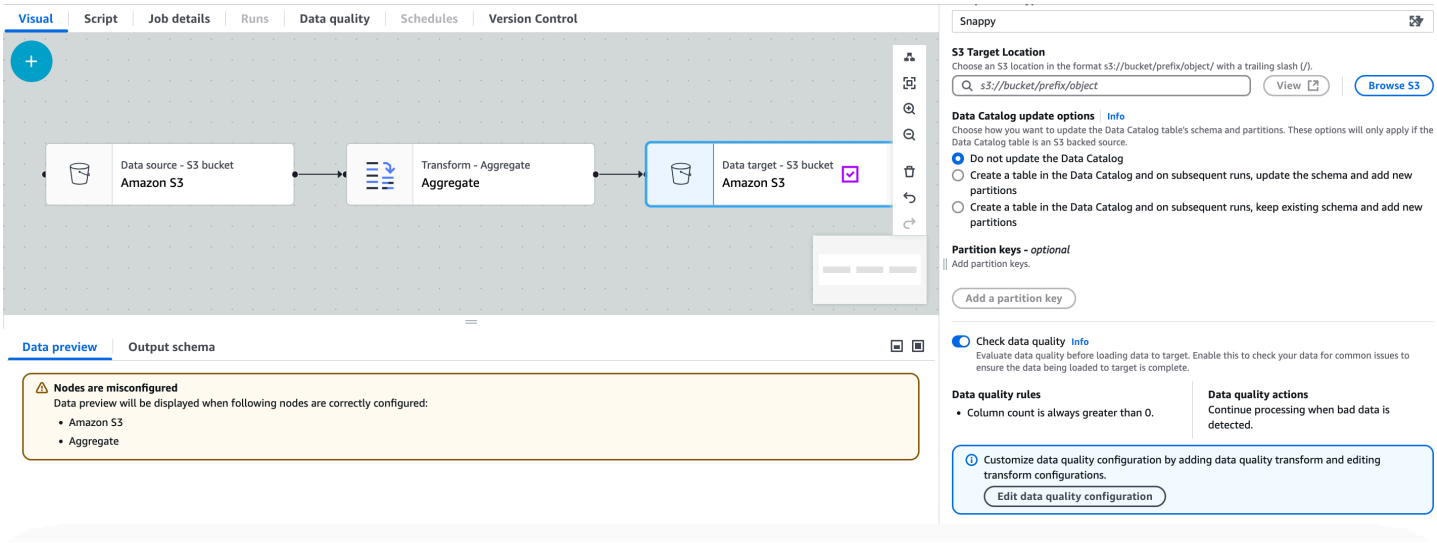
ステップ 5: データ品質結果を表示する

ジョブの実行が完了したら、[Data quality] タブを選択して、データ品質結果を表示します。

1. データ品質結果はジョブを実行するたびに表示されます。各ノードには、データ品質ステータスと、そのステータスに関する詳細が表示されます。ノードを選択すると、すべてのルールと各ルールのステータスが表示されます。
2. [Download results] を選択して、ジョブの実行とデータ品質結果に関する情報が記載された CSV ファイルをダウンロードします。
3. qデータ品質結果と実行されたジョブが複数存在する場合は、日付と時間の範囲を指定することで、結果をフィルタリングできます。[Filter by a date and time range] を選択すると、フィルターウィンドウが展開します。
4. 相対範囲 (relative range) と絶対範囲 (absolute range) のどちらかを選択します。絶対範囲を選択した場合は、カレンダーで日付を選択した後に、開始時刻と終了時刻を入力します。完了したら、[適用] を選択します。

自動データ品質

Amazon S3 をターゲットとする AWS Glue ETL ジョブを作成すると、AWS Glue ETL は自動的に Data Quality ルールを有効にし、ロードされるデータに少なくとも 1 つの列があるかどうかをチェックします。このルールは、ロードされるデータが空であったり、破損していたりしないようにするためのものです。ただし、このルールが失敗してもジョブは失敗しませんが、データ品質スコアが低下します。さらに、異常検出はデフォルトで有効になっており、データ内の列数をモニタリングします。列数にバリエーションや異常がある場合、AWS Glue ETL はこれらの異常をユーザーに通知します。この機能によって、データに関する潜在的な問題を特定し、適切なアクションを実行することができます。Data Quality ルールとその設定を表示するには、AWS Glue ETL ジョブで Amazon S3 ターゲットをクリックします。提供されたスクリーンショットが示すように、ルールの設定が表示されます。



データ品質ルールを追加するには、[データ品質設定の編集] を選択します。

集約されたメトリクス

ダッシュボードを構築するには、ルールレベルまたはルールセットレベルで合格、失敗、スキップされたレコードの数などの集計メトリクスが必要になる場合があります。各ルールの集約メトリクスとルールメトリクスを取得するには、まず EvaluateDataQuality 関数に publishAggregatedMetrics オプションを追加して集約メトリクスを有効にします。

additional_options publishAggregatedMetrics に使用できるオプションは、ENABLED と DISABLED です。例:

```
EvaluateDataQualityMultiframe = EvaluateDataQuality().process_rows(
    frame=medicare_dyf,
    ruleset=EvaluateDataQuality_ruleset,
    publishing_options={
        "dataQualityEvaluationContext": "EvaluateDataQualityMultiframe",
        "enableDataQualityCloudWatchMetrics": False,
        "enableDataQualityResultsPublishing": False,
    },
    additional_options={"publishAggregatedMetrics.status": "ENABLED"},
)
```

指定しない場合、publishAggregatedMetrics.status はデフォルトで DISABLED になり、ruleMetrics と集計メトリクスが計算されるようになります。この機能は現在、AWS Glue インタラクティブセッションおよび Glue ETL ジョブでサポートされています。これは、Glue Catalog Data Quality API ではサポートされていません。

集計メトリクスの結果の取得

additionalOptions "publishAggregatedMetrics.status": "ENABLED" の場合、次の 2 つの場所で結果を取得できます。

1. AggregatedMetrics と RuleMetrics は、resultId を指定すると GetDataQualityResult() で返されます。AggregatedMetrics と RuleMetrics には以下が含まれます。

   集約されたメトリクス:

   - 処理された行の合計数
   - 合格した行の合計数
   - 失敗した行の合計数
   - 処理されたルールの合計数
   - 合格したルールの合計数
   - 失敗したルールの合計数

```
[redacted] % aws --region us-east-1 glue get-data-quality-result --result-id "dqresult-09655f0234b3e3d129381bd162d0347ed25d1c05"
{
    "ResultId": "dqresult-09655f0234b3e3d129381bd162d0347ed25d1c05",
    "ProfileId": "dqprofile-[redacted]",
    "Score": 0.6666666666666666,
    "RulesetName": "EvaluateDataQuality_node1741974822533",
    "EvaluationContext": "EvaluateDataQuality_node1741974822533",
    "StartedOn": 1750687515.374,
    "CompletedOn": 1750687537.485,
    "JobName": "[redacted]_aggregatedmetrics_test",
    "JobRunId": "jr_[redacted]",
    "RuleResults": [
        {
            "Name": "Rule_1",
            "Description": "IsUnique \"customer_identifier\"",
            "Result": "PASS",
            "EvaluatedMetrics": {
                "Column.customer_identifier.Uniqueness": 1.0
            },
            "EvaluatedRule": "IsUnique \"customer_identifier\"",
            "RuleMetrics": {
                "rows_passed": 10.0,
                "rows_failed": 0.0,
                "rows_skipped": 0.0,
                "total_rows": 10.0
            }
        },
        {
            "Name": "Rule_2",
            "Description": "RowCount > 10",
            "EvaluationMessage": "Value: 10 does not meet the constraint requirement!",
            "Result": "FAIL",
            "EvaluatedMetrics": {
                "Dataset.*.RowCount": 10.0
            },
            "EvaluatedRule": "RowCount > 10",
            "RuleMetrics": {
                "rows_passed": 0.0,
                "rows_failed": 0.0,
                "rows_skipped": 10.0,
                "total_rows": 10.0
            }
        },
        {
            "Name": "Rule_3",
            "Description": "Completeness \"customer_identifier\" > 0.5",
            "Result": "PASS",
            "EvaluatedMetrics": {
                "Column.customer_identifier.Completeness": 1.0
            },
            "EvaluatedRule": "Completeness \"customer_identifier\" > 0.5",
            "RuleMetrics": {
                "rows_passed": 10.0,
                "rows_failed": 0.0,
                "rows_skipped": 0.0,
                "total_rows": 10.0
            }
        }
    ],
    "AggregatedMetrics": {
        "TotalRowsProcessed": 10.0,
        "TotalRowsPassed": 10.0,
        "TotalRowsFailed": 0.0,
        "TotalRulesProcessed": 3.0,
        "TotalRulesPassed": 2.0,
        "TotalRulesFailed": 1.0
    }
}
```

また、ルールレベルでは、次のメトリクスが提供されます。

ルールメトリクス:

- 合格した行
- 失敗した行
- スキップされた行
- 処理された行の合計数

2. AggregatedMetrics は追加のデータフレームとして返され、RuleOutcomes データフレームは RuleMetrics を含むように拡張されます。

実装例

次の例では、Scala で集計メトリクスを実装する方法を示しています。

```
// Script generated for node Evaluate Data Quality
val EvaluateDataQuality_node1741974822533_ruleset = """
  # Example rules: Completeness "colA" between 0.4 and 0.8, ColumnCount > 10
  Rules = [
      IsUnique "customer_identifier",
      RowCount > 10,
      Completeness "customer_identifier" > 0.5
  ]
"""

val EvaluateDataQuality_node1741974822533 =
 EvaluateDataQuality.processRows(frame=ChangeSchema_node1742850392012,
 ruleset=EvaluateDataQuality_node1741974822533_ruleset,
 publishingOptions=JsonOptions("""{"dataQualityEvaluationContext":
 "EvaluateDataQuality_node1741974822533", "enableDataQualityCloudWatchMetrics":
 "true", "enableDataQualityResultsPublishing": "true"}"""),
 additionalOptions=JsonOptions("""{"compositeRuleEvaluation.method":"ROW","observations.scope":
 "publishAggregatedMetrics.status": "ENABLED"}""" ))

println("---------------------------------ROW LEVEL
 OUTCOMES---------------------------------")
val rowLevelOutcomes_node = EvaluateDataQuality_node1741974822533("rowLevelOutcomes")

rowLevelOutcomes_node.show(10)

 println("---------------------------------RULE LEVEL
 OUTCOMES---------------------------------")

val ruleOutcomes_node = EvaluateDataQuality_node1741974822533("ruleOutcomes")

ruleOutcomes_node.show()

 println("---------------------------------AGGREGATED
 METRICS---------------------------------")

val aggregatedMetrics_node = EvaluateDataQuality_node1741974822533("aggregatedMetrics")
```

```
aggregatedMetrics_node.show()
```

サンプル結果

結果は次のように返されます。

```
{
    "Rule": "IsUnique \"customer_identifier\"",
    "Outcome": "Passed",
    "FailureReason": null,
    "EvaluatedMetrics": {
        "Column.customer_identifier.Uniqueness": 1
    },
    "EvaluatedRule": "IsUnique \"customer_identifier\"",
    "PassedCount": 10,
    "FailedCount": 0,
    "SkippedCount": 0,
    "TotalCount": 10
}
{
    "Rule": "RowCount > 10",
    "Outcome": "Failed",
    "FailureReason": "Value: 10 does not meet the constraint requirement!",
    "EvaluatedMetrics": {
        "Dataset.*.RowCount": 10
    },
    "EvaluatedRule": "RowCount > 10",
    "PassedCount": 0,
    "FailedCount": 0,
    "SkippedCount": 10,
    "TotalCount": 10
}
{
    "Rule": "Completeness \"customer_identifier\" > 0.5",
    "Outcome": "Passed",
    "FailureReason": null,
    "EvaluatedMetrics": {
        "Column.customer_identifier.Completeness": 1
    },
    "EvaluatedRule": "Completeness \"customer_identifier\" > 0.5",
    "PassedCount": 10,
    "FailedCount": 0,
    "SkippedCount": 0,
```

```
    "TotalCount": 10
}
```

集計メトリクスは次のとおりです。

```
{ "TotalRowsProcessed": 10, "PassedRows": 10, "FailedRows": 0, "TotalRulesProcessed":
 3, "RulesPassed": 2, "RulesFailed": 1 }
```

Data Quality ルールビルダー

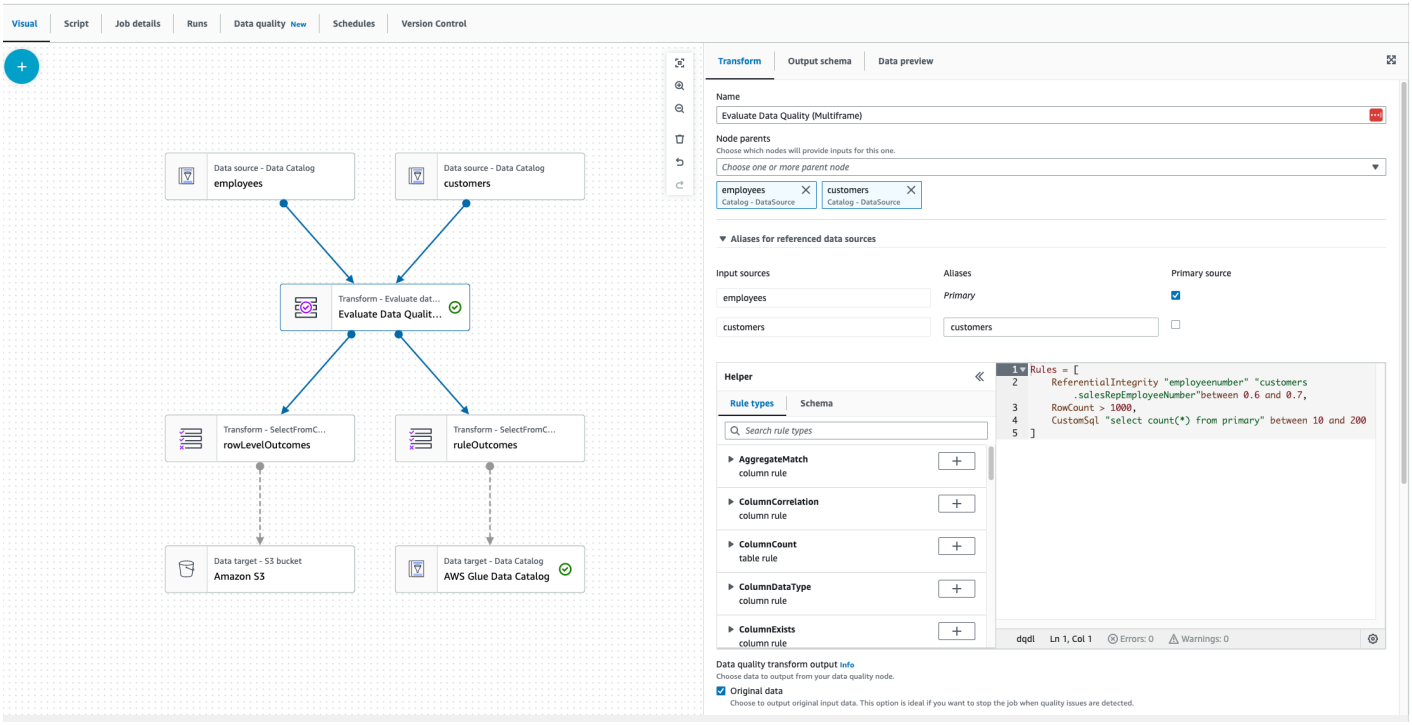
データ品質定義言語 (DQDL) ルールビルダーを使用すると、データを評価するデータ品質ルールを作成できます。最初にルールタイプを選択し、次にルールエディタでパラメータを指定します。ルールエディタには、ルールを作成する際のエラーや警告も表示されます。

DQDL のガイドには、DQDL 構文、組み込みのルールタイプ、および例を使用してルールを作成する方法に関する、包括的なドキュメントが用意されています。

データ品質評価ノード

データ品質評価変換ノードと DQDL ルールビルダーを使用する際に、作業スペースを拡張できます。

- [変換] タブを画面全体に展開するには、ノードの詳細パネルの右上隅にある展開アイコンを選択します。
- DQDL ルールエディタを展開するには、[<<] アイコンを選択してルールエディタを展開し、[Rule types] タブと [スキーマ] タブを折りたたみます。



コンポーネント

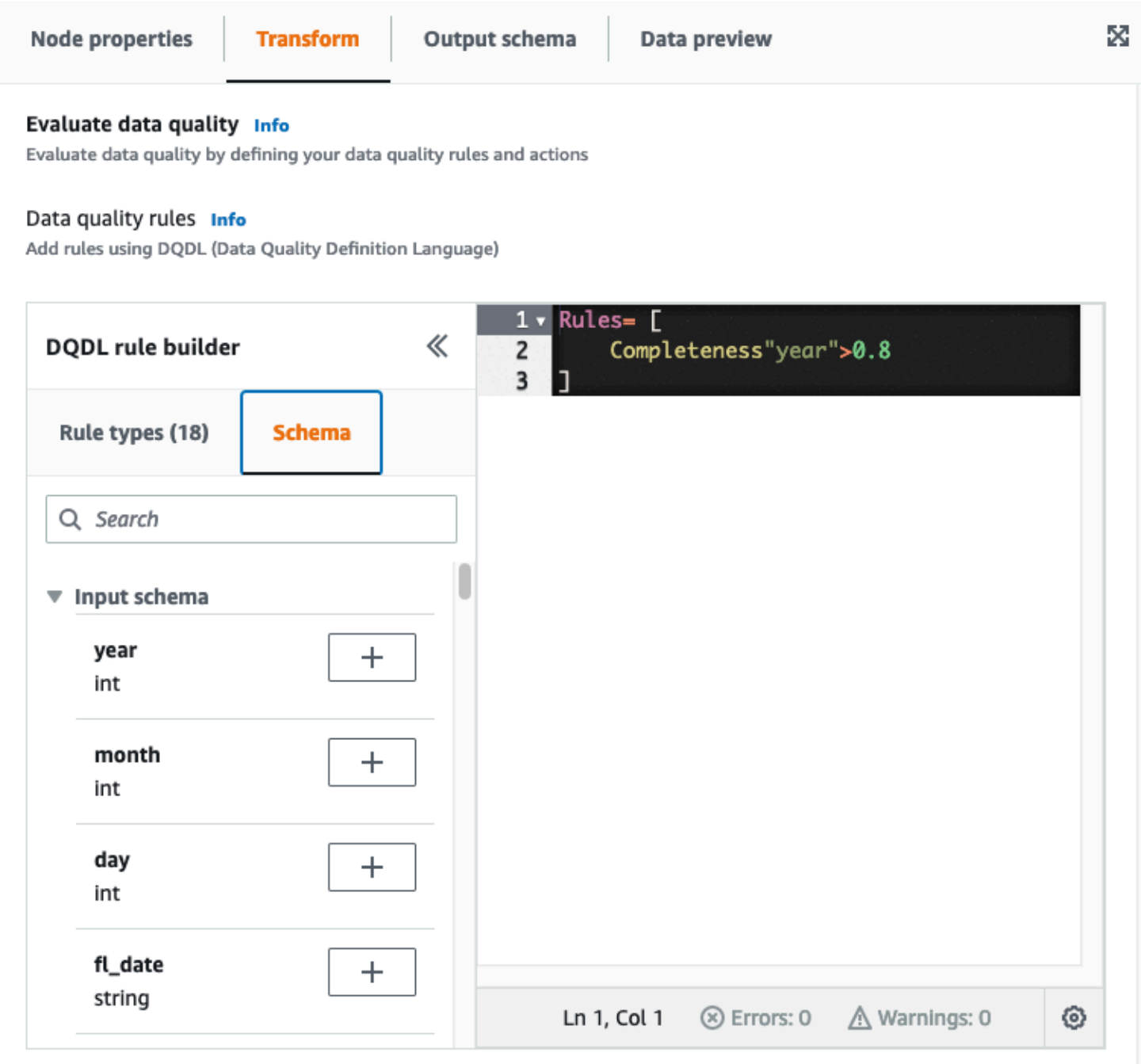
AWS Glue Studio には 26 のルールタイプが組み込まれています。各ルールタイプには、その使用方法の説明と例があります。

データ品質ルールタイプ

AWS Glue Studio には、ルールを簡単に作成するためのルールタイプが組み込まれています。ルールタイプの詳細については、「DQDL ルールタイプリファレンス」を参照してください。

Schema

[Schema] (スキーマ) タブには、親ノードの列名とデータ型が表示されます。複数のノードのスキーマが表示されます。入力スキーマを表示したり、列名で検索したり、ルール工ディタに列を挿入したりできます。



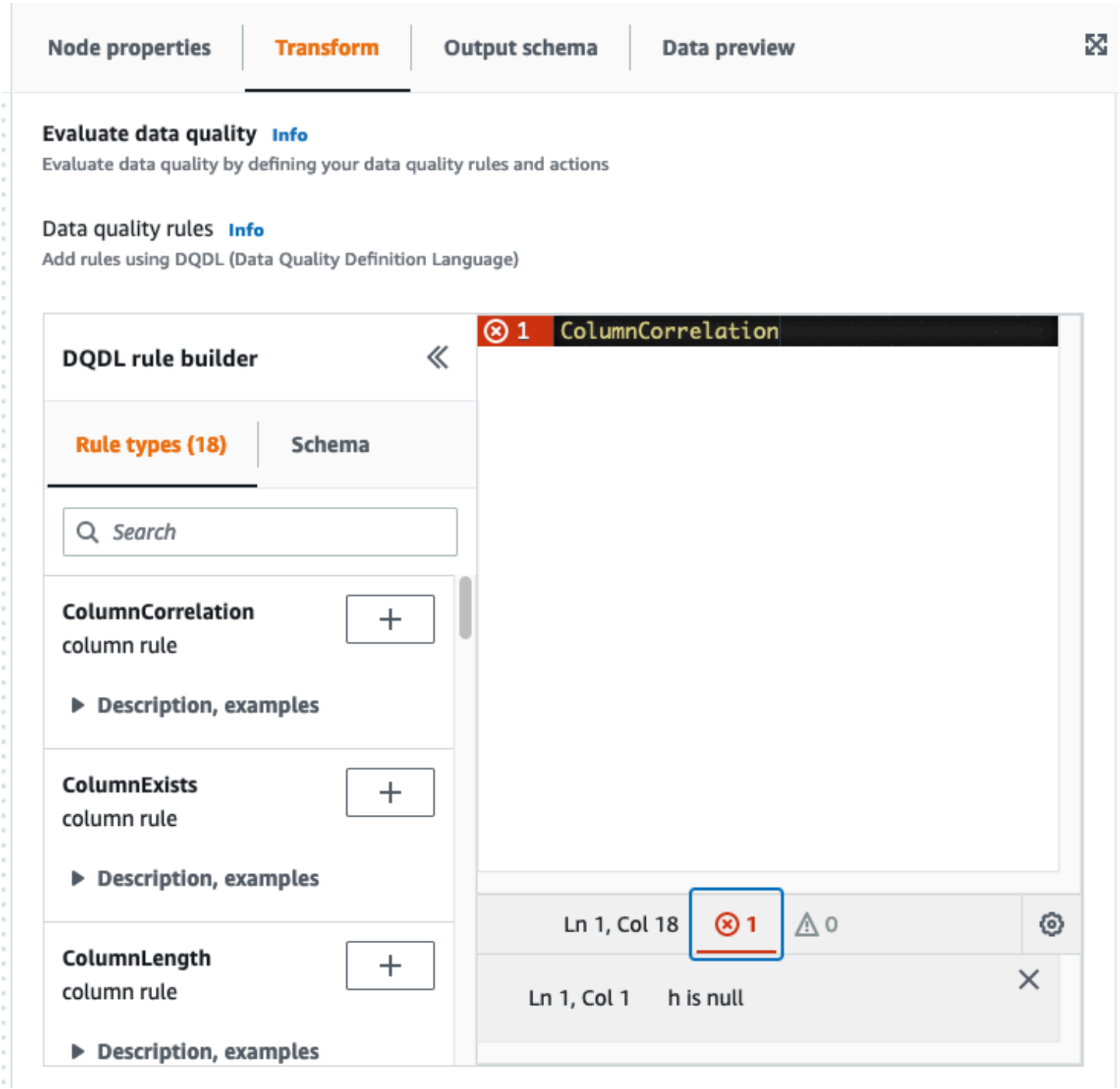
ルールエディタ

ルールエディタは、ルールを作成および編集できるテキストエディタす。DQDL ルールビルダーからルールタイプを選択すると、そのルールタイプがルールエディタに追加されます。その後、テキストを変更することで、必要に応じてパラメータを指定したり、ルールを追加したり、ルールを編集したりできます。AWS Glue Studio はルールエディタ内のルールを検証し、エラーと警告がある場合はそれらを表示します。

エラーおよび警告

ルールが DQDL ルール構文に従っていない場合、エラーがあることを視覚的に示すいくつかのインジケータがルールエディタに表示されます。

- ルールエディタにエラーアイコンが表示され、エラーのある行が赤くなります。
- ルールエディタでは、赤いエラーアイコンの横にエラーの数が表示されます。
- エラーのある行を選択すると、エラーの説明と場所 (行と列) がルールエディタの下部に表示されます。



データ品質アクション

デフォルトでは、このアクションは選択されておらず、データ品質ルールが失敗した場合でもそのジョブの実行は完了します。

次のアクションの中から選択します。アクションを使用して、特定の基準に基づいて結果を CloudWatch に公開したり、ジョブを停止したりできます。アクションは、ルールを作成した後にのみ使用できます。

- [Publish results to CloudWatch] – ジョブを実行し、その結果を CloudWatch に追加します。
- [Fail job when data quality fails] – データ品質ルールが失敗すると、結果的にジョブも失敗します。

データ品質変換出力

- [Original data] – 元の入力データを出力することを選択します。このオプションは、品質の問題が検出されたときにジョブを停止する場合に理想的です。
- [Data quality metrics] – 設定したルールとその合格または不合格のステータスを出力するように選択します。このオプションは、カスタムアクションを実行する場合に便利です。

データ品質出力設定

Amazon S3 の場所をデータ品質出力先として指定し、データ品質結果の場所を設定します。

AWS Glue ETL ジョブでの異常検出の設定

AWS Glue Studio で異常検出を開始するには、AWS Glue Studio ジョブを開き、[データ品質変換の評価] をクリックします。

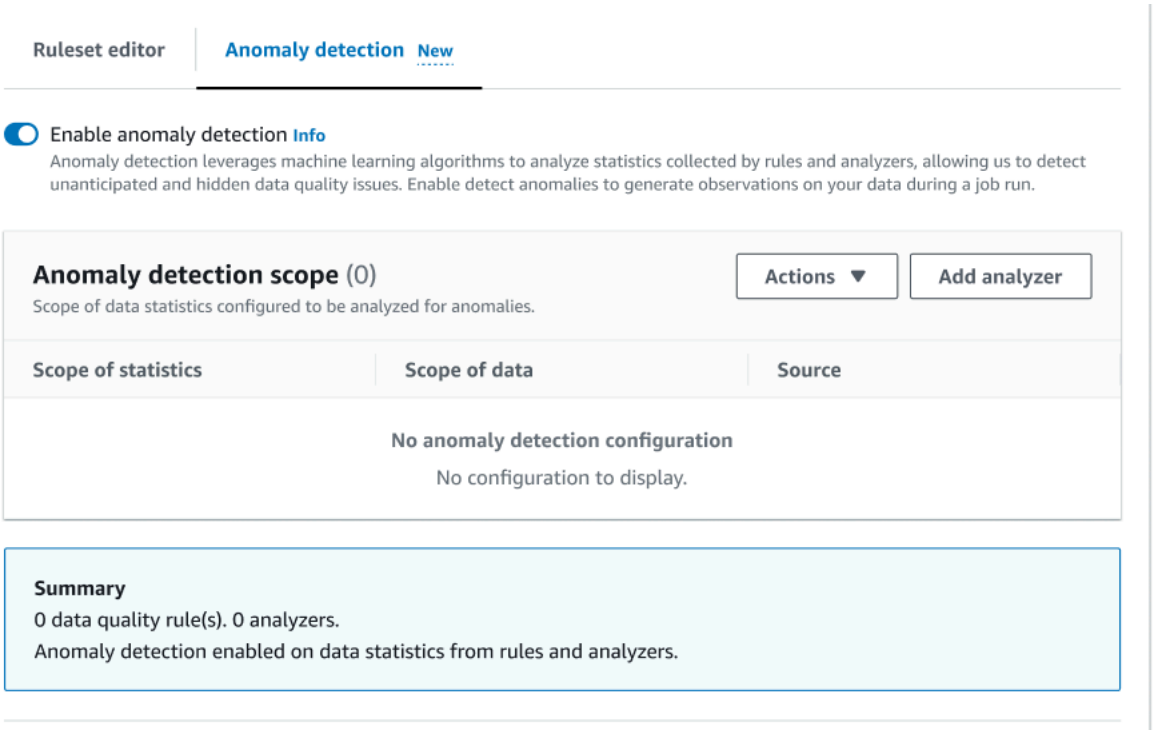
この機能を有効にすることで、AWS Glue Data Quality は時間の経過とともにデータを分析して異常を検出します。データに関する貴重なデータ統計と観測結果が提供されるため、特定された異常に対してアクションできるようになります。

この機能の内部の仕組みを理解するには、「異常検出」のドキュメントを参照してください。

異常検出の有効化

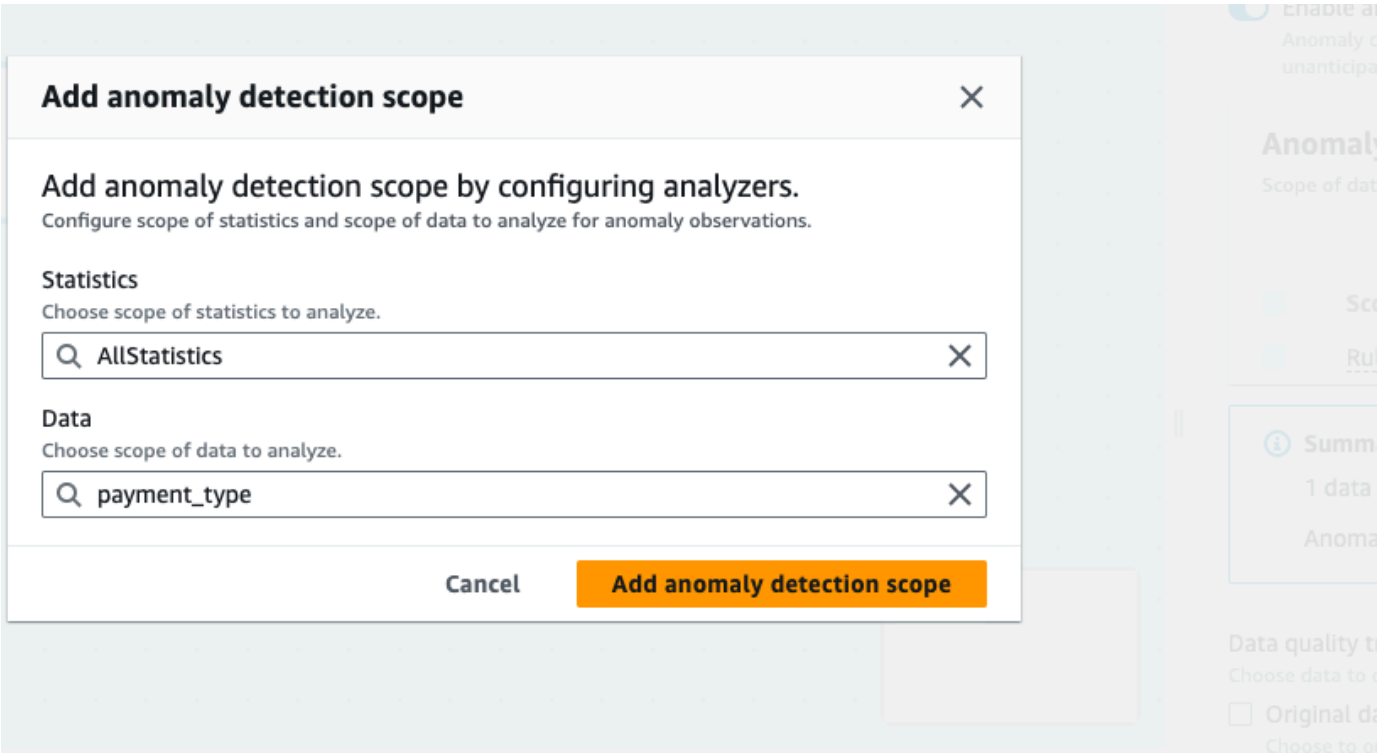
AWS Glue Studio で異常検出を有効にするには:

1. ジョブの [Data Quality] ノードを選択し、[異常検出] タブをクリックします。[異常検出を有効にする] をオンに切り替えます。



2. [アナライザーを追加] を選択して、異常を監視するデータを定義します。入力できるフィールドは [統計] と [データ] の 2 つです。

   - 統計は、データの形状やその他の特性に関する情報です。一度に 1 つまたは複数の統計を選択することも、[すべての統計] を選択することもできます。統計には、Completeness (完全性)、Uniqueness (一意性)、Mean (平均)、Sum (合計)、StandardDeviation (標準偏差)、Entropy (エントロピー)、DistinctValuesCount (個別値数)、UniqueValueRatio (固有値比率) などがあります。詳細については、「アナライザー」ドキュメントを参照してください。
   - データはデータセット内の列です。すべての列を選択することも、個々の列を選択することもできます。



3. [異常検出範囲を追加] を選択して変更を保存します。アナライザーを作成したら、[異常検出範囲] セクションで確認できます。

   [アクション] メニューを使用してアナライザーを編集したり、[ルールセットエディター] タブを選択してルールセットエディターのメモ帳でアナライザーを直接編集したりすることもできます。保存したアナライザーは、作成したルールの下に表示されます。

```
Rules = [

]

Analyzers = [
    Completeness “id”
]
```

更新されたルールセットとアナライザーが設定されると、AWS Glue Data Quality は受信データストリームを継続的にモニタリングします。設定によっては、アラートやジョブの停止によって潜在的な異常を示すことがあります。このプロアクティブモニタリングは、データパイプライン全体のデータ品質と整合性を確保するのに役立ちます。

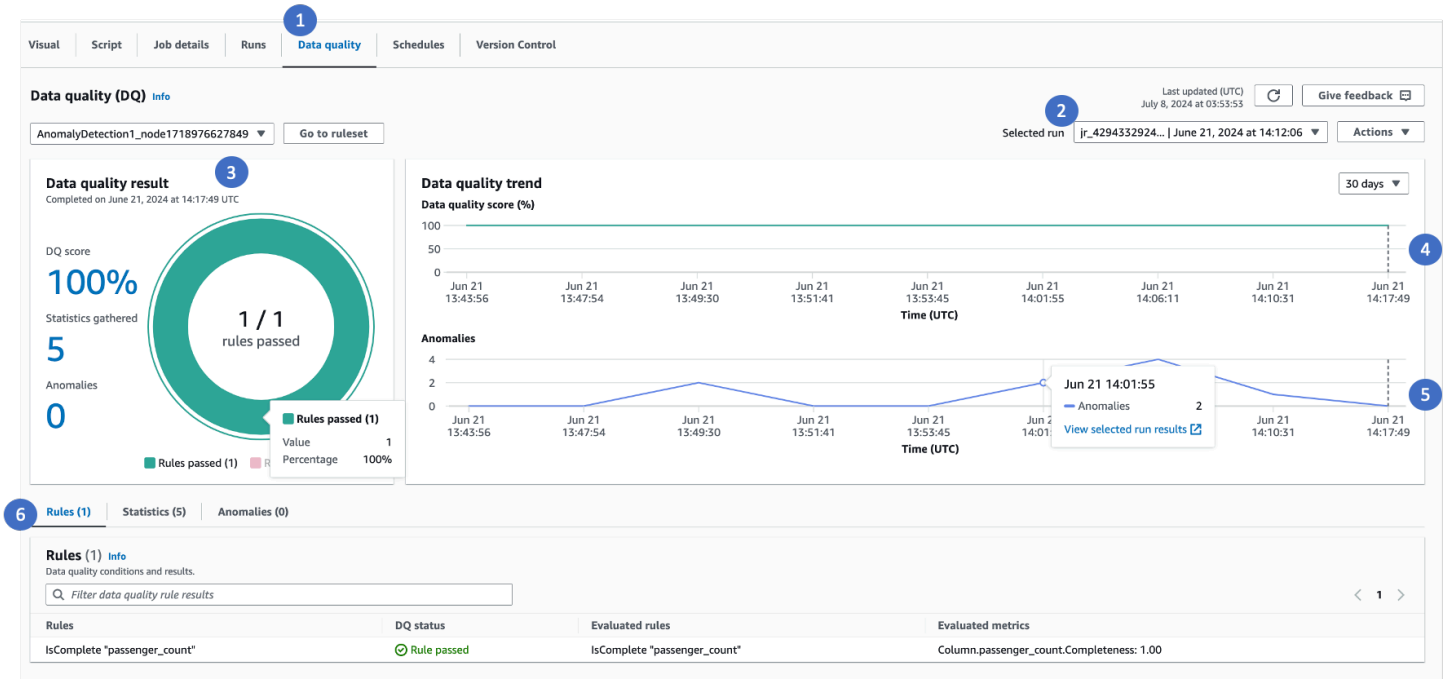
次のセクションでは、システムによって識別される異常を効果的にモニタリングする方法について説明します。また、AWS Glue Data Quality によって収集されたデータ統計を表示および分析する方法についても説明します。さらに、異常検出機能の強化につながる、機械学習モデルへのフィードバックの提供方法も理解できます。このフィードバックループは、モデルの精度を向上させ、特定のビジネス要件やデータパターンに合った異常を効果的に検出するために不可欠です。

データ品質のスコアと異常を表示

このセクションでは、Data Quality のダッシュボードとそこで提供される各種機能について説明します。

高レベルのデータ品質メトリクスと傾向を視覚化して理解する

ジョブが成功したら、[Data Quality] タブを選択して、データ品質のスコアと異常を表示します。



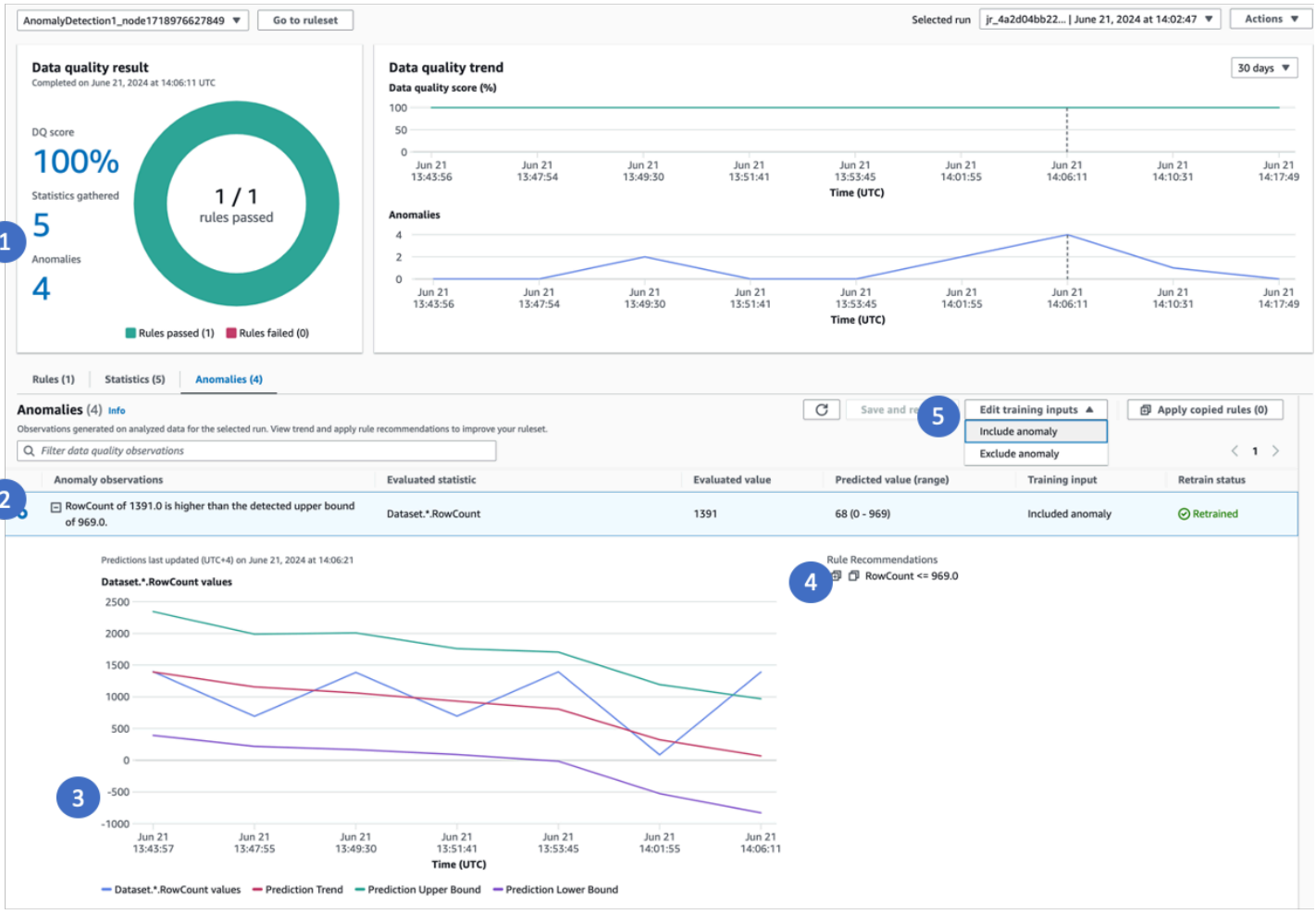
[Data Quality] タブの次のコンポーネントは、役に立つ情報を提供します。

1. [Data Quality] タブを選択すると、データ品質メトリクスが表示されます。
2. 特定のジョブ実行 ID を選択して、Data Quality のスコアを表示します。
3. このペインには 3 つの重要な情報が表示されます。異常、データ統計、またはルールを表示するために、個別に選択して特定のテーブルに移動することができます。
   - ルールが設定されている場合の Data Quality のスコア。
   - ルールとアナライザーによって収集された統計の数。

- 検出された異常の総数。

4. このトレンドチャートは、データ品質が時間の経過とともにどのように傾向しているかを表示しています。傾向にカーソルを合わせると、データ品質のスコアが悪化した特定の時点に移動できます。
5. 経時的な異常の傾向は、時間の経過とともに検出された異常の数を示します。
6. タブ:
   - ルールタブは、すべてのルールとステータスのリストを表示するデフォルトのタブです。評価済みルールは、動的ルールが評価された実際の値を表示する場合に便利です。
   - 統計タブにはすべての統計が一覧表示されるため、メトリクスとトレンドを経時的に表示できます。
   - 異常タブには、検出された異常のリストが表示されます。

異常の表示と異常検出アルゴリズムのトレーニング

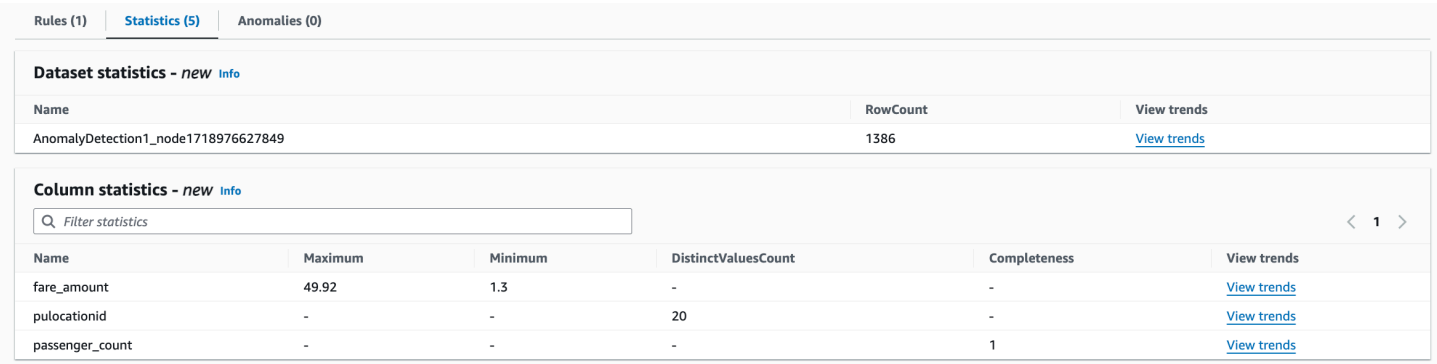


上記のイメージの呼び出し:

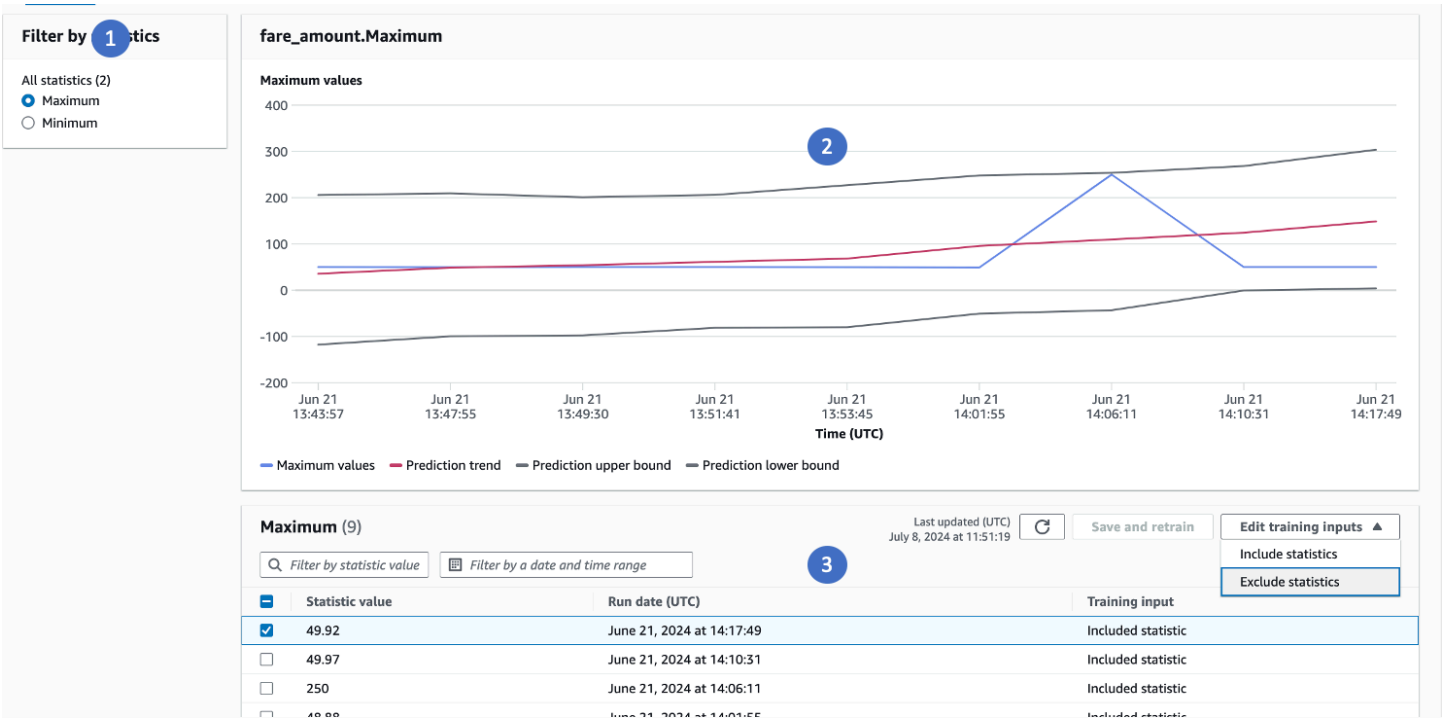
1. 異常が検出されたら、異常をクリックするか、異常タブを選択します。
2. AWS Glue Data Quality は、異常、実際の値、予測範囲の詳細な説明を提供します
3. AWS Glue Data Quality はトレンドラインを表示します。実際の値、実際の値 (赤い線) に基づく派生的傾向、上限値および下限値が示されています。
4. AWS Glue Data Quality では、将来のパターンをキャプチャするために使用できるデータ品質ルールを推奨しています。推奨されるすべてのルールをコピーしてデータ品質ノードに適用し、これらのパターンを効果的にキャプチャできます。
5. 機械学習 (ML) モデルに入力を提供して異常値を除外し、将来の実行で異常を正確かつ確実に検出できます。異常を明示的に除外しない場合、AWS Glue Data Quality はそれらを将来の予測のモデルの一部として自動的に考慮します。指定したモデル入力が反映されるのは最新の実行のみであることに注意してください。例えば、過去数回の実行から異常なポイントを除外した場合、最新の実行でモデル入力を表示および更新しない限り、モデルはそれらの変更を反映しません。モデルは、最新の実行で必要な調整を行うまで、以前に提供された入力を引き続き使用します。異常値の除外を積極的に管理することで、特定のデータパターンと要件に対する異常の構成要素への ML モデルの理解を絞り込み、時間の経過とともにより正確な異常検出を行うことができます。

時間の経過に伴うデータ統計の表示とトレーニング入力の提供

データ統計やデータプロファイルを表示し、時間の経過とともにどのように進行しているかを表示したい場合があります。これを行うには、[統計] を選択するか、[統計] タブを開きます。その後、AWS Glue Data Quality によって収集された最新のデータ統計を表示できます。



[トレンドの表示] をクリックすると、各統計が時間の経過とともにどのように進行しているかが表示されます。



1. 特定の列の統計を選択できます
2. 傾向の進行状況を確認できます
3. 異常な値を選択して、除外したり含めたりできます。このフィードバックを提供することで、アルゴリズムは識別された異常なデータポイントを除外または含め、モデルを再トレーニングします。この再トレーニングプロセスにより、どの値を異常と見なすべきかについて、モデルが提供したフィードバックから学習するため、正確な異常検出を行うことができます。

   このフィードバックループを通じて、アルゴリズムの特定のデータパターンとビジネス要件に対する異常の構成要素への理解を絞り込むことができます。再トレーニングされたモデルでは、異常としてフラグを付けるべきではない値を除外したり、見逃された値を含めたりすることで、予想されるデータポイントと真に異常なデータポイントの区別がしやすくなります。

AWS Glue Studio ノートブックにおける ETL ジョブ向けの Data Quality

このチュートリアルでは、AWS Glue Studio ノートブックで、抽出、変換、ロード (ETL) ジョブに AWS Glue Data Quality を使用する方法について説明します。

AWS Glue Studio でノートブックを使用してジョブスクリプトを編集し、完全なジョブを実行せずに出力を表示できます。マークダウンを追加したり、ノートブックを `.ipynb` ファイルやジョブス

クリプトとして保存したりすることもできます。ソフトウェアをローカルにインストールしたり、サーバーを管理したりすることなくノートブックを起動できます。コードに問題がなければ、AWS Glue Studio を使用して、ノートブックを AWS Glue ジョブに簡単に変換できます。

この例で使用されているデータセットは、2 つの Data.CMS.gov データセット (Inpatient Prospective Payment System Provider Summary for the Top 100 Diagnosis-Related Groups - FY2011 および Inpatient Charge Data FY 2011) からダウンロードされた、メディケアプロバイダーの支払いデータで構成されています。

ダウンロードした後、データセットを修正してファイルの最後にいくつかの誤ったレコードを追加しました。この変更されたファイルは、パブリックな Amazon S3 バケット (`s3://awsglue-datasets/examples/medicare/Medicare_Hospital_Provider.csv`) 内に置かれています。

前提条件

- 宛先 Amazon S3 バケットに書き込むための Amazon S3 アクセス許可を持つ AWS Glue ロール
- 新しいノートブック (「AWS Glue Studio 中でのノートブックの使用開始」を参照)

AWS Glue Studio での ETL ジョブの作成

ETL ジョブを作成するには

1. セッションバージョンを AWS Glue 3.0 に変更します。

   変更するには、以下の方法で定型コードのセルをすべて削除し、セルを実行します。この定型コードは、新しいノートブックを作成すると、最初のセルに自動的に入力されることに注意してください。

   ```
   %glue_version 3.0
   ```

2. 次のコードをコピーして、セルで実行します。

   ```
   import sys
   from awsglue.transforms import *
   from awsglue.utils import getResolvedOptions
   from pyspark.context import SparkContext
   from awsglue.context import GlueContext
   from awsglue.job import Job

   sc = SparkContext.getOrCreate()
   ```

```
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
```

3. 次のセルで、AWS Glue Data Quality を評価する `EvaluateDataQuality` クラスをインポートします。

```
from awsgluedq.transforms import EvaluateDataQuality
```

4. 次のセルで、公開 Amazon S3 バケットに保存されている .csv ファイルを使用して、ソースデータを読み取ります。

```
medicare = spark.read.format(
"csv").option(
"header", "true").option(
"inferSchema", "true").load(
's3://awsglue-datasets/examples/medicare/Medicare_Hospital_Provider.csv')
medicare.printSchema()
```

5. データを AWS Glue DynamicFrame に変換します。

```
from awsglue.dynamicframe import DynamicFrame
medicare_dyf = DynamicFrame.fromDF(medicare,glueContext,"medicare_dyf")
```

6. データ品質定義言語 (DQDL) を使用して、ルールセットを作成します。

```
EvaluateDataQuality_ruleset = """
    Rules = [
        ColumnExists "Provider Id",
        IsComplete "Provider Id",
        ColumnValues  " Total Discharges " > 15
]
    ]
"""
```

7. データセットをルールセットに照らして検証します。

```
EvaluateDataQualityMultiframe = EvaluateDataQuality().process_rows(
    frame=medicare_dyf,
    ruleset=EvaluateDataQuality_ruleset,
    publishing_options={
        "dataQualityEvaluationContext": "EvaluateDataQualityMultiframe",
        "enableDataQualityCloudWatchMetrics": False,
        "enableDataQualityResultsPublishing": False,
    },
    additional_options={"performanceTuning.caching": "CACHE_NOTHING"},
)
```

8. 結果を確認します。

```
ruleOutcomes = SelectFromCollection.apply(
    dfc=EvaluateDataQualityMultiframe,
    key="ruleOutcomes",
    transformation_ctx="ruleOutcomes",
)

ruleOutcomes.toDF().show(truncate=False)
```

出力:

```
---------------------------------------+-------
+-----------------------------------------------------
+-----------------------------------------+
|Rule                                    |Outcome|FailureReason
                  |EvaluatedMetrics                            |
+---------------------------------------+-------
+-----------------------------------------------------
+-----------------------------------------+
|ColumnExists "Provider Id"              |Passed |null
                  |{}                                          |
|IsComplete "Provider Id"                |Passed |null
                  |{Column.Provider Id.Completeness -> 1.0}    |
|ColumnValues " Total Discharges " > 15|Failed |Value: 11.0 does not meet the
 constraint requirement!|{Column. Total Discharges .Minimum -> 11.0}|
```

```
+--------------------------------------+-------
+--------------------------------------------------
+-----------------------------------------+
```

9. 合格した行をフィルタリングし、データ品質の行レベルの結果から不合格の行を確認します。

```
owLevelOutcomes = SelectFromCollection.apply(
dfc=EvaluateDataQualityMultiframe,
key="rowLevelOutcomes",
transformation_ctx="rowLevelOutcomes",
)

rowLevelOutcomes_df = rowLevelOutcomes.toDF() # Convert Glue DynamicFrame to
 SparkSQL DataFrame
rowLevelOutcomes_df_passed =
 rowLevelOutcomes_df.filter(rowLevelOutcomes_df.DataQualityEvaluationResult ==
 "Passed") # Filter only the Passed records.
rowLevelOutcomes_df.filter(rowLevelOutcomes_df.DataQualityEvaluationResult ==
 "Failed").show(5, truncate=False) # Review the Failed records
```

出力:

```
+----------------------------------------+-----------
+-------------------------------------+--------------------------+-------------
+--------------+-----------------+-----------------------------------
+------------------+-------------------------+-----------------------
+-------------------------+-------------------------
+-----------------------------------------+---------------------------
+---------------------------+
|DRG Definition                           |Provider Id|Provider Name
           |Provider Street Address   |Provider City|Provider State|Provider Zip
 Code|Hospital Referral Region Description| Total Discharges | Average Covered
 Charges | Average Total Payments |Average Medicare Payments|DataQualityRulesPass
      |DataQualityRulesFail                    |DataQualityRulesSkip        |
DataQualityEvaluationResult|
+----------------------------------------+-----------
+-------------------------------------+--------------------------+-------------
+--------------+-----------------+-----------------------------------
+------------------+-------------------------+-----------------------
+-------------------------+-------------------------
```

```
+----------------------------------------+---------------------------
+--------------------------+
|039 - EXTRACRANIAL PROCEDURES W/O CC/MCC|10005      |MARSHALL MEDICAL CENTER SOUTH
        |2505 U S HIGHWAY 431 NORTH|BOAZ         |AL            |35957
 |AL - Birmingham                       |14               |$15131.85
  |$5787.57               |$4976.71                |[IsComplete "Provider Id"]|
[ColumnValues " Total Discharges " > 15]|[ColumnExists "Provider Id"]|Failed
              |
|039 - EXTRACRANIAL PROCEDURES W/O CC/MCC|10046      |RIVERVIEW REGIONAL MEDICAL
 CENTER    |600 SOUTH THIRD STREET    |GADSDEN      |AL            |35901
   |AL - Birmingham                       |14               |$67327.92
   |$5461.57               |$4493.57                |[IsComplete "Provider Id"]|
[ColumnValues " Total Discharges " > 15]|[ColumnExists "Provider Id"]|Failed
              |
|039 - EXTRACRANIAL PROCEDURES W/O CC/MCC|10083      |SOUTH BALDWIN REGIONAL
 MEDICAL CENTER|1613 NORTH MCKENZIE STREET|FOLEY        |AL            |36535
        |AL - Mobile                           |15               |$25411.33
         |$5282.93               |$4383.73                |[IsComplete "Provider
 Id"]|[ColumnValues " Total Discharges " > 15]|[ColumnExists "Provider Id"]|Failed
                   |
|039 - EXTRACRANIAL PROCEDURES W/O CC/MCC|30002      |BANNER GOOD SAMARITAN MEDICAL
 CENTER |1111 EAST MCDOWELL ROAD   |PHOENIX      |AZ            |85006
 |AZ - Phoenix                          |11               |$34803.81
  |$7768.90               |$6951.45                |[IsComplete "Provider Id"]|
[ColumnValues " Total Discharges " > 15]|[ColumnExists "Provider Id"]|Failed
              |
|039 - EXTRACRANIAL PROCEDURES W/O CC/MCC|30010      |CARONDELET ST  MARYS HOSPITAL
        |1601 WEST ST MARY'S ROAD  |TUCSON       |AZ            |85745
 |AZ - Tucson                           |12               |$35968.50
  |$6506.50               |$5379.83                |[IsComplete "Provider Id"]|
[ColumnValues " Total Discharges " > 15]|[ColumnExists "Provider Id"]|Failed
              |
+----------------------------------------+-----------
+-------------------------------------+-------------------------+-------------
+--------------+-----------------+-----------------------------------
+------------------+-------------------------+----------------------
+-------------------------+-------------------------
+----------------------------------------+---------------------------
+--------------------------+
only showing top 5 rows
```

AWS Glue Data Quality に 4 つの新しい列 (DataQualityRulesPass、DataQualityRulesFail、DataQualityRulesSkip、DataQualityEvaluationResult) が追加されたことに注意してください。これには、合格したレコード、不合格のレコード、行レベルの評価でスキップされたルール、および行レベルの全体的な結果が表示されます。

10. Amazon S3 バケットに出力を書き込んでデータを分析し、結果を視覚化します。

```
#Write the Passed records to the destination.

glueContext.write_dynamic_frame.from_options(
       frame = rowLevelOutcomes_df_passed,
       connection_type = "s3",
       connection_options = {"path": "s3://glue-sample-target/output-dir/
medicare_parquet"},
       format = "parquet")
```

データ品質定義言語 (DQDL) リファレンス

データ品質定義言語 (DQDL) は、AWS Glue Data Quality でのルール定義に使用するドメイン固有の言語です。

このガイドでは、DQDL の理解を助けるために、この言語の主要な概念について解説します。また、DQDL ルールタイプのリファレンスの中で、構文や例も提供しています。このガイドを開始する際には、あらかじめ AWS Glue Data Quality を理解しておくことをお勧めします。詳細については、「AWS Glue Data Quality」を参照してください。

> **Note**
>
> DynamicRules は AWS Glue ETL でのみサポートされています。

目次

- DQDL 構文
  - ルールの構造
  - 複合ルール

    - 複合ルールの仕組み
  - 表現
    - NULL、EMPTY、WHITESPACES_ONLY のキーワード
    - Where 句によるフィルタリング
  - 動的ルール
  - アナライザー
  - コメント
- DQDL ルールタイプリファレンス
  - AggregateMatch
  - ColumnCorrelation
  - ColumnCount
  - ColumnDataType
  - ColumnExists
  - ColumnLength
  - ColumnNamesMatchPattern
  - ColumnValues
  - Completeness
  - CustomSQL
  - DataFreshness
  - DatasetMatch
  - DistinctValuesCount
  - Entropy
  - IsComplete
  - IsPrimaryKey
  - IsUnique
  - 平均値
  - ReferentialIntegrity
  - RowCount
  - RowCountMatch
  - StandardDeviation

- Sum
- SchemaMatch
- Uniqueness
- UniqueValueRatio
- DetectAnomalies
- FileFreshness
- FileMatch
- FileUniqueness
- FileSize

DQDL 構文

DQDL のドキュメントでは大文字と小文字が区別されます。また、個々の 品質評価ルールをグループ化したルールセットが含まれます。ルールセットを作成するには、Rules という名前 (大文字) の、角括弧で区切られたリストを作成する必要があります。リストの中には、次の例のようにカンマで区切られた 1 つ以上の DQDL ルールを含めます。

```
Rules = [
   IsComplete "order-id",
   IsUnique "order-id"
]
```

ルールの構造

DQDL ルールの構造は、そのルールタイプによって異なります。ただし、DQDL の各ルールは、以下のような基本的な形式を取ります。

```
<RuleType> <Parameter> <Parameter> <Expression>
```

RuleType は設定するルールタイプの名前で、大文字と小文字が区別されます。例えば、IsComplete、IsUnique、または CustomSql です。ルールタイプごとに、ルールのパラメータが異なります。DQDL のルールタイプと、そのパラメータの詳細なリファレンスについては、「DQDL ルールタイプリファレンス」を参照してください。

複合ルール

DQDL では、ルールを組み合わせる際に、以下の論理演算子を使用できます。これらのルールは複合ルールと呼ばれます。

and

and 論理演算子は、接続するルールがすべて true である場合に限り、結果に true を返します。それ以外の場合、ルールの組み合わせ結果は false になります。and 演算子前後に配置する各ルールは、丸括弧で囲む必要があります。

次の例では、and 演算子を使用して 2 つの DQDL ルールを組み合わせています。

```
(IsComplete "id") and (IsUnique "id")
```

or

or 論理演算子は、接続するルールが 1 つ以上 true であれば、結果に true を返します。or 演算子前後に配置する各ルールは、丸括弧で囲む必要があります。

次の例では、or 演算子を使用して 2 つの DQDL ルールを組み合わせています。

```
(RowCount "id" > 100) or (IsPrimaryKey "id")
```

同じ演算子を使用して複数のルールを接続でき、次のようなルールの組み合わせが可能です。

```
(Mean "Star_Rating" > 3) and (Mean "Order_Total" > 500) and (IsComplete "Order_Id")
```

論理演算子を 1 つの式にまとめることができます。例:

```
(Mean "Star_Rating" > 3) and ((Mean "Order_Total" > 500) or (IsComplete "Order_Id"))
```

より複雑な入れ子になったルールを作成することもできます。

```
(RowCount > 0) or ((IsComplete "colA") and (IsUnique "colA"))
```

複合ルールの仕組み

デフォルトでは、複合ルールはデータセットまたはテーブル全体の個別のルールとして評価されたら、結果が結合されます。つまり、最初に列全体を評価してから演算子を適用します。このデフォルト動作は以下の例で説明します。

```
# Dataset

+------+------+
|myCol1|myCol2|
+------+------+
|     2|     1|
|     0|     3|
+------+------+

# Overall outcome

+-------------------------------------------------------------+-------+
|Rule                                                         |Outcome|
+-------------------------------------------------------------+-------+
|(ColumnValues "myCol1" > 1) OR (ColumnValues "myCol2" > 2)|Failed |
+-------------------------------------------------------------+-------+
```

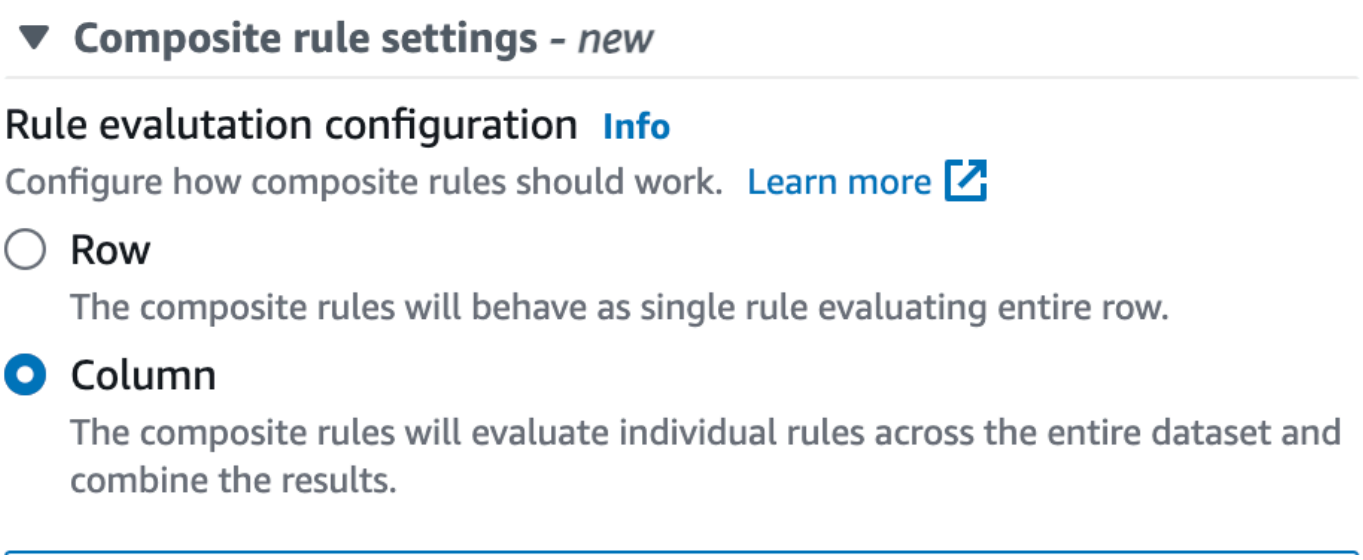
上の例では、AWS Glue Data Quality は最初に失敗に終わる (`ColumnValues "myCol1" > 1`) を評価します。次に (`ColumnValues "myCol2" > 2`) を評価しますが、これも失敗します。両方の結果の組み合わせは FAILED として表示されます。

ただし、行全体を評価する必要がある SQL のような動作を好む場合、以下のコードスニペットの `additionalOptions` で示すように `ruleEvaluation.scope` パラメータを明示的に設定する必要があります。

```
object GlueApp {
  val datasource = glueContext.getCatalogSource(
    database="<db>",
    tableName="<table>",
    transformationContext="datasource"
  ).getDynamicFrame()

  val ruleset = """
    Rules = [
        (ColumnValues "age" >= 26) OR (ColumnLength "name" >= 4)
```

```
    ]
  """

  val dq_results = EvaluateDataQuality.processRows(
    frame=datasource,
    ruleset=ruleset,
    additionalOptions=JsonOptions("""
        {
          "compositeRuleEvaluation.method":"ROW"
        }
      """
    )
  )
}
```

AWS Glue Data Catalog では、次に示すように、ユーザーインターフェイスでこのオプションを簡単に設定できます。



いったん設定すると、複合ルールは行全体を評価する単一のルールとして動作します。次の例では、この動作を示します。

```
# Row Level outcome

+------+------+-------------------------------------------------------------
+---------------------------+
```

```
|myCol1|myCol2|DataQualityRulesPass                                          |
DataQualityEvaluationResult|
+------+------+-------------------------------------------------------------
+---------------------------+
|2     |1     |[(ColumnValues "myCol1" > 1) OR (ColumnValues "myCol2" > 2)]|Passed
                |
|0     |3     |[(ColumnValues "myCol1" > 1) OR (ColumnValues "myCol2" > 2)]|Passed
                |
+------+------+-------------------------------------------------------------
+---------------------------+
```

一部のルールは全体的な結果がしきい値または比率に依存するため、この機能ではサポートできません。以下に一覧表示します。

比率に依存するルール:

- Completeness
- DatasetMatch
- ReferentialIntegrity
- Uniqueness

しきい値に依存するルール:

次のルールにしきい値が含まれる場合、ルールはサポートされません。ただし、`with threshold` に関連しないルールは引き続きサポートされます。

- ColumnDataType
- ColumnValues
- CustomSQL

表現

ルールタイプがブール型で応答を生成しない場合、この応答を取得するには、パラメータで表現を指定する必要があります。例えば、次のルールでは、表現に対応させながら列内のすべての値の mean 値 (平均) をチェックし、真または偽の結果を返します。

```
Mean "colA" between 80 and 100
```

一部のルールタイプ (IsUnique や IsComplete など) は、既にブール型の応答を返しています。

次の表に、DQDL ルールで使用できる式を一覧で示します。

サポートされている DQDL 表現

式	説明	例
=*x*	ルールタイプの応答が *x* と等しい場合、true を返します。	`Completeness "colA" = "1.0", ColumnValues "colA" = "2022-06-30"`
!=*x*	ルールタイプの応答が *x* と等しくない場合、x は true を返します。	`ColumnValues "colA" != "a", ColumnValues "colA" != "2022-06-30"`
> *x*	ルールタイプの応答が *x* より大きい場合、true を返します。	`ColumnValues "colA" > 10`
< *x*	ルールタイプの応答が *x* より小さい場合、true を返します。	`ColumnValues "colA" < 1000, ColumnValues "colA" < "2022-06-30"`
>= *x*	ルールタイプの応答が *x* 以上の場合、true を返します。	`ColumnValues "colA" >= 10`
<= *x*	ルールタイプの応答が *x* 以下の場合、true を返します。	`ColumnValues "colA" <= 1000`
between *x* and *y*	ルールタイプの応答が指定された範囲内 (除外) にある場合、true を返します。この式のタイプは、数値および日付タイプにのみ使用できます。	`Mean "colA" between 8 and 100, ColumnValues "colA" between "2022-05-31" and "2022-06-30"`

式	説明	例
x と *y* の間ではない	ルールタイプの応答が指定された範囲外 (含有) にある場合、true を返します。この式のタイプは、数値および日付のタイプにのみ使用できます。	`ColumnValues "colA" not between "2022-05-31" and "2022-06-30"`
in [*a, b, c, ...*]	ルールタイプの応答が指定されたセットに含まれている場合に、true を返します。	`ColumnValues "colA" in [ 1, 2, 3 ], ColumnValues "colA" in [ "a", "b", "c" ]`
not in [*a, b, c, ...*]	ルールタイプの応答が指定されたセットに含まれていない場合、true を返します。	`ColumnValues "colA" not in [ 1, 2, 3 ], ColumnValues "colA" not in [ "a", "b", "c" ]`
matches */ab+c/i*	ルールタイプの応答が正規表現と一致する場合に、true を返します。	`ColumnValues "colA" matches "[a-zA-Z]*"`
not matches */ab+c/i*	ルールタイプのレスポンスが正規表現と一致しない場合、true を返します。	`ColumnValues "colA" not matches "[a-zA-Z]*"`
now( )	日付表現を作成する ColumnValues ルールタイプでのみ機能します。	`ColumnValues "load_date" > (now() - 3 days)`

式	説明	例
matches/in […]/not matches/not in [...] with threshold	ルール条件に一致する値をパーセンテージで指定します。ColumnValues 、ColumnDataType 、CustomSQL のルールタイプのみで使用できます。	`ColumnValues "colA" in ["A", "B"] with threshold > 0.8, ColumnValues "colA" matches "[a-zA-Z]*" with threshold between 0.2 and 0.9 ColumnDataType "colA" = "Timestamp" with threshold > 0.9`

NULL、EMPTY、WHITESPACES_ONLY のキーワード

文字列の列に null、空、空白のみの文字列があるかどうかを検証する場合、次のキーワードを使用できます。

- NULL / null – 文字列の列の null 値には、このキーワードは true を返します。

  50% 以上のデータに null 値が含まれていない場合、`ColumnValues "colA" != NULL with threshold > 0.5` は true を返します。

  null 値または長さが 5 を超えるすべての行に対し、`(ColumnValues "colA" = NULL) or (ColumnLength "colA" > 5)` は true を返します。これには、「compositeRuleEvaluation.Method」=「ROW」オプションを使用する必要があることに注意してください。
- EMPTY / empty – 文字列の列に空の文字列 (“”) の値には、このキーワードは true を返します。一部のデータ形式は、文字列の列に null を空の文字列に変換します。このキーワードは、データの空の文字列を除外するのに便利です。

  行が空、「a」、「b」のいずれかの場合、`(ColumnValues "colA" = EMPTY) or (ColumnValues "colA" in ["a", "b"])` は true を返します。これには、「compositeRuleEvaluation.Method」=「ROW」オプションを使用する必要があることに注意してください。
- WHITESPACES_ONLY / whitespaces_only – 文字列の列で空白 (“ ”) のみの値が含まれる文字列には、このキーワードは true を返します。

行が「a」、「b」、空白のいずれかでない場合、ColumnValues "colA" not in ["a", "b", WHITESPACES_ONLY] は true を返します。

サポートされているルール:

- ColumnValues

数値または日付ベースの式には、列に null があるかどうかを検証する場合、次のキーワードを使用できます。

- NULL / null – 文字列の列の null 値には、このキーワードは true を返します。

  列の日付が 2023-01-01 または null の場合、ColumnValues "colA" in [NULL, "2023-01-01"] は true を返します。

  null 値または 1 から 9 までの値が含まれているすべての行に対し、(ColumnValues "colA" = NULL) or (ColumnValues "colA" between 1 and 9) は true を返します。これには、「compositeRuleEvaluation.Method」=「ROW」オプションを使用する必要があることに注意してください。

  サポートされているルール:

  - ColumnValues

Where 句によるフィルタリング

> **Note**
>
> Where 句は AWS Glue 4.0 でのみサポートされています。

ルールの作成時にデータをフィルタリングできます。これは、条件付きルールを適用する場合に便利です。

```
<DQDL Rule> where "<valid SparkSQL where clause> "
```

フィルターは where キーワード、その後に引用符 ("") で囲まれた有効な Spark SQL ステートメントを指定する必要があります。

しきい値を持つルールに Where 句を追加するルールの場合、しきい値条件の前に Where 句を指定する必要があります。

```
<DQDL Rule> where "valid SparkSQL statement>" with threshold <threshold condition>
```

この構文では次のようなルールを記述できます。

```
Completeness "colA" > 0.5 where "colB = 10"
ColumnValues "colB" in ["A", "B"] where "colC is not null" with threshold > 0.9
ColumnLength "colC" > 10 where "colD != Concat(colE, colF)"
```

提供された Spark SQL ステートメントが有効であることを検証します。無効な場合、ルールの評価は失敗し、次の形式の `IllegalArgumentException` がスローされます。

```
Rule <DQDL Rule> where "<invalid SparkSQL>" has provided an invalid where clause :
<SparkSQL Error>
```

行レベルのエラーレコード識別が有効になっている場合の句の動作

AWS Glue Data Quality を使用すると、失敗した特定のレコードを識別できます。行レベルの結果をサポートするルールに Where 句を適用すると、Where 句で除外された行に `Passed` というラベルが付けられます。

フィルタリングされた行に個別に`SKIPPED`というラベルを付ける場合は、ETL ジョブに以下`additionalOptions`を設定できます。

```
object GlueApp {
  val datasource = glueContext.getCatalogSource(
    database="<db>",
    tableName="<table>",
    transformationContext="datasource"
  ).getDynamicFrame()

  val ruleset = """
    Rules = [
       IsComplete "att2" where "att1 = 'a'"
    ]
  """
```

```
  val dq_results = EvaluateDataQuality.processRows(
    frame=datasource,
    ruleset=ruleset,
    additionalOptions=JsonOptions("""
        {
          "rowLevelConfiguration.filteredRowLabel":"SKIPPED"
        }
      """
    )
  )
}
```

例として、次のルールとデータフレームを参照してください:

```
IsComplete att2 where "att1 = 'a'"
```

id	att1	att2	行レベルの結果 (デフォルト)	行レベルの結果 (スキップオプション)	コメント
1	a	f	PASSED	PASSED	
2	b	d	PASSED	SKIPPED	att1 は "a" ではないため、行はフィルターで除外されます
3	a	null	FAILED	FAILED	
4	a	f	PASSED	PASSED	
5	b	null	PASSED	SKIPPED	att1 は "a" ではないため、行はフィルターで除外されます
6	a	f	PASSED	PASSED	

動的ルール

> **Note**
>
> 動的ルールは AWS Glue ETL でのみサポートされており、 AWS Glue Data Catalog ではサポートされていません。

動的ルールを作成して、ルールによって生成された現在のメトリクスと履歴値を比較できるようになりました。これらの履歴比較は、式に `last()` 演算子を使用することで可能になります。例えば、現在の実行で行数が同じデータセットの最新の行数よりも多い場合、`RowCount > last()` ルールは成功します。`last()` は、考慮すべき事前メトリクスの数を記述する自然数引数をオプションで取り、`last(k)` では `k >= 1` が直近の k メトリクスを参照します。

- データポイントがない場合、`last(k)` はデフォルト値 0.0 を返します。
- 利用可能な k メトリクスよりも少ない場合は、`last(k)` はそれ以前のメトリクスをすべて返します。

有効な式を作成するには `last(k)` を使用します。`k > 1` では、複数の履歴結果を 1 つの数値にまとめる集計関数が必要です。例えば `RowCount > avg(last(5))` は、現在のデータセットの行数が、同じデータセットの直近の 5 つの行数の平均より真に大きいかどうかを確認します。現在のデータセットの行数をリストと有意に比較できないため、`RowCount > last(5)` はエラーを生成します。

サポートされている集計関数:

- `avg`
- `median`
- `max`
- `min`
- `sum`
- `std` (標準偏差)
- `abs` (絶対値)
- `index(last(k), i)` では直近の k から i 番目に新しい値を選択できます。i はゼロインデックスであるため、`index(last(3), 0)` は最新のデータポイントを返しますが、データ

ポイントが 3 つしかなく、4 番目に新しいデータポイントにインデックスを付けようとするため、`index(last(3), 3)` はエラーになります。

式の例

ColumnCorrelation

- `ColumnCorrelation "colA" "colB" < avg(last(10))`

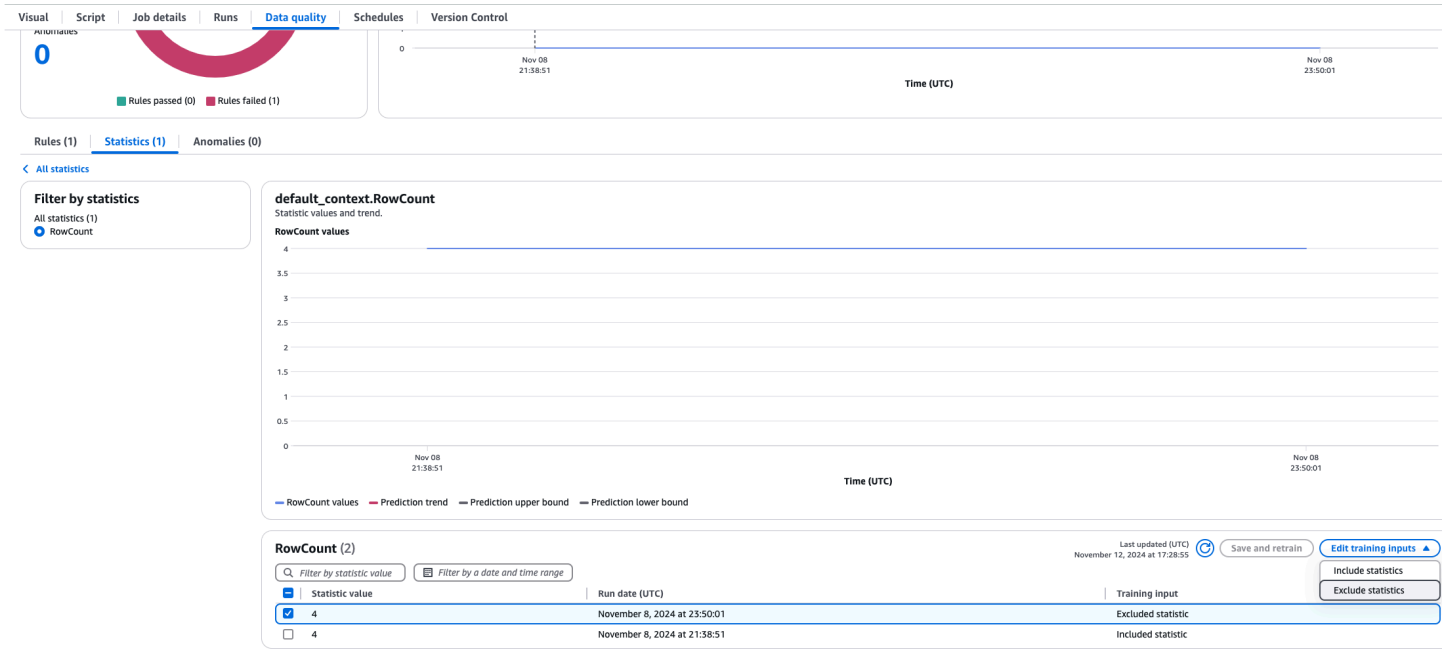
DistinctValuesCount

- `DistinctValuesCount "colA" between min(last(10))-1 and max(last(10))+1`

数値条件またはしきい値を使用するほとんどのルールタイプは動的ルールをサポートします。使用しているルールタイプで動的ルールがサポートされているかどうかを確認するには、提供されている「アナライザーとルール」の表を参照してください。

動的ルールからの統計の除外

場合によっては、動的ルール計算からデータ統計を除外する必要があります。履歴データのロードを行い、それを平均値に影響を与えたくないとします。これを行うには、AWS Glue ETL でジョブを開き、[データの品質] タブを選択し、[統計] を選択し、除外する統計を選択します。統計の表とともにトレンドチャートを表示できます。除外する値を選択し、[統計を除外] を選択します。これで、除外された統計は動的ルール計算に含まれなくなります。



アナライザー

> **Note**
>
> アナライザーは、AWS Glue Data Catalog ではサポートされていません。

DQDL ルールは、アナライザーと呼ばれる関数を使用してデータに関する情報を収集します。この情報をルールのブール式に使用して、ルールが成功するか失敗するかを決定します。例えば、RowCount ルール `RowCount > 5` は、行数アナライザーを使用してデータセット内の行数を検出し、その数を式 `> 5` と比較して、現在のデータセットに 5 行以上存在するかどうかを確認します。

ルールを作成する代わりに、アナライザーを作成して、異常の検出に使用できる統計を生成させることを推奨する場合があります。このような場合は、アナライザーを作成できます。アナライザーは、次の点でルールとは異なります。

特徴	アナライザー	ルール
ルールセットの一部	あり	はい
統計を生成	あり	はい

特徴	アナライザー	ルール
観察結果を生成	あり	はい
条件を評価してアサートできる	いいえ	はい
障害発生時にジョブを停止したり、ジョブの処理を続行したりするなどのアクションを設定できます。	いいえ	はい

アナライザーはルールなしで独立して存在できるため、アナライザーをすばやく構成し、データ品質ルールを段階的に構築できます。

ルールセットの Analyzers ブロックに入力できるルールタイプもあります。これにより、アナライザーに必要なルールを実行し、条件のチェックを行わずに情報を収集できます。アナライザーの中には、ルールに関連付けられておらず、Analyzers ブロックにしか入力できないものもあります。次の表は、各項目がルールとしてサポートされているか、スタンドアロンアナライザーとしてサポートされているか、および各ルールタイプの詳細を示しています。

アナライザーを使用したルールセットの例

以下のルールセットでは以下のものが使用されます。

- 直近の 3 回のジョブ実行で、データセットが平均値を上回っているかどうかを確認する動的ルール
- データセットの Name 列内の異なる値の数を記録する DistinctValuesCount アナライザー
- Name の最小サイズと最大サイズを経時的に追跡する ColumnLength アナライザー

アナライザーメトリクスの結果は、ジョブ実行の [データ品質] タブで確認できます。

```
Rules = [
   RowCount > avg(last(3))
]
Analyzers = [
   DistinctValuesCount "Name",
   ColumnLength "Name"
]
```

AWS Glue Data Quality は、次のアナライザーをサポートしています。

アナライザー名	機能
RowCount	データセットの行数を計算します
Completeness	列の完全性の割合を計算します
Uniqueness	列の一意性の割合を計算します
Mean	数値列の平均を計算します
Sum	数値列の合計を計算します
StandardDeviation	数値列の標準偏差を計算します
Entropy	数値列のエントロピーを計算します
DistinctValuesCount	列内の個別値の数を計算します
UniqueValueRatio	列内の一意の値の比率を計算します
ColumnCount	データセット内の列数を計算します
ColumnLength	列の長さを計算します
ColumnValues	数値列の最小値と最大値を計算します。数値列以外の最小の ColumnLength と最大の ColumnLength を計算します
ColumnCorrelation	特定の列の列相関を計算します
CustomSql	カスタム SQL によって返される統計を計算します
AllStatistics	次の統計を計算します: • RowCount、ColumnCount • すべての列: Completeness、Uniqueness

アナライザー名	機能
	• 数値: Min、Max、Entropy、Mean、Standard Dev、Sum • 文字列: MinLength、MaxLength

コメント

「#」文字を使用して DQDL ドキュメントにコメントを追加できます。「#」文字の後から行末までに含まれるすべての文字は DQDL に無視されます。

```
Rules = [
    # More items should generally mean a higher price, so correlation should be
 positive
    ColumnCorrelation "price" "num_items" > 0
]
```

DQDL ルールタイプリファレンス

このセクションでは、AWS Glue Data Quality がサポートする各ルールタイプのリファレンスを提供します。

> **Note**
>
> - 現在、DQDL はリストタイプまたはネストされた列データをサポートしていません。
> - 以下の表の括弧内の値は、ルール引数で指定された情報に置き換えられます。
> - ルールでは通常、式に追加の引数が必要です。

Ruletype	説明	引数	報告されたメトリクス	ルールとしてサポートされていますか?	アナライザーとしてサポートされていますか?	行レベルの結果を返しますか?	動的ルールをサポートしますか?	観察結果を生成	Where 句の構文をサポートしますか？
Aggregat Match	売上総額などのサマリーメトリクスを比較して、2 つのデータセットが一致しているかをチェックします。金融機関が、すべてのデータがソースシステムから取り込まれているかを比	1 つ以上の集計	1 番目と 2 番目の集計列名が一致する場合: Column. [Column]. gregatetch 1 番目と 2 番目の集計列の名前が異なる場合: Column. [Column1, lumn2]. gregatetch	はい	いいえ	いいえ	いいえ	いいえ	なし

Ruletype	説明	引数	報告されたメトリクス	ルールとしてサポートされていますか?	アナライザーとしてサポートされていますか?	行レベルの結果を返しますか?	動的ルールをサポートしますか?	観察結果を生成	Where句の構文をサポートしますか？
	較する際などに便利です。								

Ruletype	説明	引数	報告されたメトリクス	ルールとしてサポートされていますか?	アナライザーとしてサポートされていますか?	行レベルの結果を返しますか?	動的ルールをサポートしますか?	観察結果を生成	Where句の構文をサポートしますか？

AllStatistics	データセット内の指定した列の複数のメトリクスを収集するスタンドアロンアナライザー。	単一の列名	すべてのタイプの列の場合:
			`Dataset.RowCou`
			`Column.[Column].mpleteness`
			`Column.[Column].iquenes`
			文字列値列のその他のメトリクス:
				いいえ	はい	いいえ	いいえ	いいえ	なし

Ruletype	説明	引数	報告されたメトリクス	ルールとしてサポートされていますか?	アナライザーとしてサポートされていますか?	行レベルの結果を返しますか?	動的ルールをサポートしますか?	観察結果を生成	Where 句の構文をサポートしますか？
			Columnl gth metrics 数値列のその他のメトリクス: Columnl ues metrics						
ColumnC relation	2 つの列にどの程度の相関性があるかを確認します。	列名はちょうど 2 つです	Multico mn. [Colum n1,Colu 2].Colu Correla on	あり	はい	いいえ	はい	いいえ	はい
ColumnC nt	抜け落ちた列がないかを確認します。	なし	Dataset .Columr unt	あり	はい	いいえ	はい	はい	なし

Ruletype	説明	引数	報告されたメトリクス	ルールとしてサポートされていますか?	アナライザーとしてサポートされていますか?	行レベルの結果を返しますか?	動的ルールをサポートしますか?	観察結果を生成	Where 句の構文をサポートしますか ?
ColumnDataType	列がデータ型に準拠しているかをチェックします。	列名は 1 つだけです	Column.[Column].ColumnDataType.Compliance	はい	いいえ	なし	はい (行レベルのしきい値式の場合)	いいえ	はい

Ruletype	説明	引数	報告されたメトリクス	ルールとしてサポートされていますか?	アナライザーとしてサポートされていますか?	行レベルの結果を返しますか?	動的ルールをサポートしますか?	観察結果を生成	Where 句の構文をサポートしますか？
ColumnExists	データセットに列が存在するかをチェックします。これにより、セルフサービスのデータプラットフォームを構築しているユーザーは、特定の列が利用可能であることを確	列名は 1 つだけです	該当なし	はい	いいえ	いいえ	いいえ	いいえ	なし

Ruletype	説明	引数	報告されたメトリクス	ルールとしてサポートされていますか?	アナライザーとしてサポートされていますか?	行レベルの結果を返しますか?	動的ルールをサポートしますか?	観察結果を生成	Where句の構文をサポートしますか？
	認できます。								

Ruletype	説明	引数	報告されたメトリクス	ルールとしてサポートされていますか?	アナライザーとしてサポートされていますか?	行レベルの結果を返しますか?	動的ルールをサポートしますか?	観察結果を生成	Where 句の構文をサポートしますか？
ColumnLength	データの長さが一貫しているかをチェックします。	列名は 1 つだけです	Column.[Column].MaximumLength Column.[Column].MinimumLength 行レベルのしきい値が指定されている場合のその他のメトリクス: Column.[Column].ColumnValues.Compliance	あり	はい	はい (行レベルのしきい値が指定されている場合)	なし	はい。最小長と最大長を分析して観察結果のみを生成します。	はい

Ruletype	説明	引数	報告されたメトリクス	ルールとしてサポートされていますか?	アナライザーとしてサポートされていますか?	行レベルの結果を返しますか?	動的ルールをサポートしますか?	観察結果を生成	Where 句の構文をサポートしますか?
ColumnN esMatchP ttern	列名が定義済みのパターンと一致しているかをチェックします。ガバナンスチームが列名の一貫性を保つ際などに便利です。	列名の正規表現	Dataset .Columr mesPatt nMatchF io	はい	いいえ	いいえ	いいえ	いいえ	なし

Ruletype	説明	引数	報告されたメトリクス	ルールとしてサポートされていますか?	アナライザーとしてサポートされていますか?	行レベルの結果を返しますか?	動的ルールをサポートしますか?	観察結果を生成	Where 句の構文をサポートしますか?
ColumnValues	データが定義済みの値と一致しているかをチェックします。このルールは正規表現に対応しています。	列名は 1 つだけです	Column.[Column].ximum Column.[Column].nimum 行レベルのしきい値が指定されている場合のその他のメトリクス: Column.[Column].lumnVals.Compliance	あり	はい	はい (行レベルのしきい値が指定されている場合)	なし	はい。最小値と最大値を分析して観察結果のみを生成します。	はい

Ruletype	説明	引数	報告されたメトリクス	ルールとしてサポートされていますか?	アナライザーとしてサポートされていますか?	行レベルの結果を返しますか?	動的ルールをサポートしますか?	観察結果を生成	Where 句の構文をサポートしますか?
Complete ess	データに空白または NULL がないかをチェックします。	列名は 1 つだけです	Column. [C olumn]. mpleter s	あり	はい	はい	はい	はい	はい
CustomS	ユーザーは、ほぼすべてのタイプのデータ品質チェックを SQL に実装できます。	SQL ステートメント (オプション) 行レベルのしきい値	Dataset .Custom L 行レベルのしきい値が指定されている場合のその他のメトリクス: Dataset .Custom L.Compl nce	はい	なし	はい (行レベルのしきい値が指定されている場合)	はい	いいえ	なし

Ruletype	説明	引数	報告されたメトリクス	ルールとしてサポートされていますか?	アナライザーとしてサポートされていますか?	行レベルの結果を返しますか?	動的ルールをサポートしますか?	観察結果を生成	Where 句の構文をサポートしますか?
DataFres ness	データが最新であるかをチェックします。	列名は 1 つだけです	Column [C olumn]. taFresh ss.Comp ance	はい	いいえ	はい	いいえ	いいえ	はい
DatasetM tch	2 つのデータセットを比較して、同期しているかを識別します。	参照データセットの名前 列のマッピング (オプション) 一致を確認する列	Dataset [Referer Dataset ias].Da setMatc	はい	いいえ	はい	はい	いいえ	なし

Ruletype	説明	引数	報告されたメトリクス	ルールとしてサポートされていますか?	アナライザーとしてサポートされていますか?	行レベルの結果を返しますか?	動的ルールをサポートしますか?	観察結果を生成	Where 句の構文をサポートしますか?
DistinctValuesCount	重複する値がないかをチェックします。	列名は 1 つだけです	Column.[Column].DistinctValuesCount	あり	はい	はい	はい	はい	はい
DetectAnomalies	別のルールタイプで報告されたメトリクスに異常がないかチェックします。	ルールタイプ	ルールタイプ引数で報告されたメトリクス (1 つまたは複数)	はい	いいえ	いいえ	いいえ	いいえ	なし
エントロピー	データのエントロピーをチェックします。	列名は 1 つだけです	Column.[Column].Entropy	あり	はい	いいえ	はい	いいえ	はい

Ruletype	説明	引数	報告されたメトリクス	ルールとしてサポートされていますか?	アナライザーとしてサポートされていますか?	行レベルの結果を返しますか?	動的ルールをサポートしますか?	観察結果を生成	Where 句の構文をサポートしますか?
IsComple e	すべてのデータが完全であるかをチェックします。	列名は 1 つだけです	Column. [C olumn]. mpleter s	はい	いいえ	はい	いいえ	いいえ	はい
IsPrimary Key	列がプライマリキー (NULL および一意ではない) であるかをチェックします。	列名は 1 つだけです	1 列の場合: Column. [C olumn]. iquenes 複数列の場合: Multicc mn. [Comma Delimit Columns Uniquer s	はい	いいえ	はい	いいえ	いいえ	はい

Ruletype	説明	引数	報告されたメトリクス	ルールとしてサポートされていますか?	アナライザーとしてサポートされていますか?	行レベルの結果を返しますか?	動的ルールをサポートしますか?	観察結果を生成	Where 句の構文をサポートしますか?
IsUnique	データがすべて一意であるかをチェックします。	列名は 1 つだけです	Column.[Column].iquenes	はい	いいえ	はい	いいえ	いいえ	はい
平均値	平均値が、設定済みのしきい値と一致するかをチェックします。	列名は 1 つだけです	Column.[Column].an	あり	はい	はい	はい	いいえ	はい

Ruletype	説明	引数	報告されたメトリクス	ルールとしてサポートされていますか?	アナライザーとしてサポートされていますか?	行レベルの結果を返しますか?	動的ルールをサポートしますか?	観察結果を生成	Where 句の構文をサポートしますか？
ReferentialIntegrity	2 つのデータセットに参照整合性があるかをチェックします。	データセットの 1 つまたは複数の列名 参照データセットの 1 つまたは複数の列名	Column.[Reference Dataset Alias].ReferentialIntegrity	はい	いいえ	はい	はい	いいえ	なし
RowCou	レコード数がしきい値と一致するかをチェックします。	なし	Dataset.RowCou	あり	はい	いいえ	はい	はい	はい

Ruletype	説明	引数	報告されたメトリクス	ルールとしてサポートされていますか?	アナライザーとしてサポートされていますか?	行レベルの結果を返しますか?	動的ルールをサポートしますか?	観察結果を生成	Where 句の構文をサポートしますか？
RowCount	2 つのデータセットのレコード数が一致するかをチェックします。	参照データセットのエイリアス	Dataset [Referer Dataset ias].Ro ountMat	はい	いいえ	いいえ	はい	いいえ	なし
Standard eviation	標準偏差がしきい値と一致するかをチェックします。	列名は 1 つだけです	Column [C olumn]. andardD iation	あり	はい	はい	はい	いいえ	はい

Ruletype	説明	引数	報告されたメトリクス	ルールとしてサポートされていますか?	アナライザーとしてサポートされていますか?	行レベルの結果を返しますか?	動的ルールをサポートしますか?	観察結果を生成	Where 句の構文をサポートしますか？
SchemaMatch	2 つのデータセットのスキーマが一致するかをチェックします。	参照データセットのエイリアス	Dataset[Referer Dataset ias].SchemaMatch	はい	いいえ	いいえ	はい	いいえ	なし
合計	合計が、設定済みのしきい値と一致するかをチェックします。	列名は 1 つだけです	Column[Column].m	あり	はい	いいえ	はい	いいえ	はい

Ruletype	説明	引数	報告されたメトリクス	ルールとしてサポートされていますか?	アナライザーとしてサポートされていますか?	行レベルの結果を返しますか?	動的ルールをサポートしますか?	観察結果を生成	Where 句の構文をサポートしますか？
Uniquene s	データセットの一意性がしきい値と一致するかをチェックします。	列名は 1 つだけです	Column. [C olumn]. iquenes	あり	はい	はい	はい	いいえ	はい
UniqueV ueRatio	一意の値の比率がしきい値と一致するかをチェックします。	列名は 1 つだけです	Column. [C olumn]. iqueVal Ratio	あり	はい	はい	はい	いいえ	はい

Ruletype	説明	引数	報告されたメトリクス	ルールとしてサポートされていますか?	アナライザーとしてサポートされていますか?	行レベルの結果を返しますか?	動的ルールをサポートしますか?	観察結果を生成	Where 句の構文をサポートしますか？
FileFreshness	Amazon S3 内のファイルが最新かどうかを確認します。	ファイルまたはフォルダのパスとしきい値。	Dataset.FileFreshness.Compliance Dataset.FileCount	はい	いいえ	いいえ	いいえ	いいえ	なし

Ruletype	説明	引数	報告されたメトリクス	ルールとしてサポートされていますか?	アナライザーとしてサポートされていますか?	行レベルの結果を返しますか?	動的ルールをサポートしますか?	観察結果を生成	Where句の構文をサポートしますか？
FileMatch	ファイルの内容がチェックサムまたは他のファイルと一致するかどうかを確認します。このルールはチェックサムを使用して、2つのファイルが同じかどうかを検証します。	ソースファイルまたはフォルダパスとターゲットファイルまたはフォルダパス。	統計は生成されません。	はい	いいえ	いいえ	いいえ	いいえ	なし

Ruletype	説明	引数	報告されたメトリクス	ルールとしてサポートされていますか?	アナライザーとしてサポートされていますか?	行レベルの結果を返しますか?	動的ルールをサポートしますか?	観察結果を生成	Where 句の構文をサポートしますか?
FileSize	ファイルのサイズが指定された条件と一致するかどうかを確認します。	ファイルまたはフォルダのパスとしきい値。	Dataset.FileSize.Compliance Dataset.FileCount Dataset.MaximumFileSize Dataset.MinimumFileSize	はい	いいえ	いいえ	いいえ	いいえ	なし
FileUniqueness	チェックサムを使用してファイルが一意かどうかを確認します。	ファイルまたはフォルダのパスとしきい値。	Dataset.FileUniquenessRatio Dataset.FileCount	はい	いいえ	いいえ	いいえ	いいえ	なし

トピック

- AggregateMatch
- ColumnCorrelation
- ColumnCount
- ColumnDataType
- ColumnExists
- ColumnLength
- ColumnNamesMatchPattern
- ColumnValues
- Completeness
- CustomSQL
- DataFreshness
- DatasetMatch
- DistinctValuesCount
- Entropy
- IsComplete
- IsPrimaryKey
- IsUnique
- 平均値
- ReferentialIntegrity
- RowCount
- RowCountMatch
- StandardDeviation
- Sum
- SchemaMatch
- Uniqueness
- UniqueValueRatio
- DetectAnomalies
- FileFreshness
- FileMatch
- FileUniqueness

- FileSize

AggregateMatch

特定の式を参照して、2 つの列の集計の比率をチェックします。このルールタイプは、複数のデータセットで機能します。2 つの列の集計が評価され、1 番目の列の集計結果を 2 番目の列の集計結果で割ることで比率が算出されます。比率と指定された式が照合され、ブール型応答が生成されます。

構文

列の集計

```
AggregateMatch <AGG_OPERATION> (<OPTIONAL_REFERENCE_ALIAS>.<COL_NAME>)
```

- AGG_OPERATION – 集計に使用する操作。現在サポートされている形式は、`sum` と `avg` です。

  列でサポートされている型: Byte (バイト)、Decimal (十進数)、Double (倍精度浮動小数点数)、Float (浮動小数点数)、Integer (整数)、Long (整数)、Short (整数)
- OPTIONAL_REFERENCE_ALIAS – 列がプライマリデータセットではなく参照データセットのものである場合は、このパラメータを指定する必要があります。このルールを AWS Glue データカタログで使用している場合は、参照エイリアスは "<database_name>.<table_name>.<column_name> の形式に従います。

  列でサポートされている型: Byte (バイト)、Decimal (十進数)、Double (倍精度浮動小数点数)、Float (浮動小数点数)、Integer (整数)、Long (整数)、Short (整数)
- COL_NAME — 集計する列の名前。

  列でサポートされている型: Byte (バイト)、Decimal (十進数)、Double (倍精度浮動小数点数)、Float (浮動小数点数)、Integer (整数)、Long (整数)、Short (整数)

例: 平均

```
"avg(rating)"
```

例: 合計

```
"sum(amount)"
```

例: 参照データセットの列の平均

```
"avg(reference.rating)"
```

ルール

```
AggregateMatch <AGG_EXP_1> <AGG_EXP_2> <EXPRESSION>
```

- AGG_EXP_1 – 最初の列の集計。

  列でサポートされている型: Byte (バイト)、Decimal (十進数)、Double (倍精度浮動小数点数)、Float (浮動小数点数)、Integer (整数)、Long (整数)、Short (整数)

  列でサポートされている型: Byte (バイト)、Decimal (十進数)、Double (倍精度浮動小数点数)、Float (浮動小数点数)、Integer (整数)、Long (整数)、Short (整数)
- AGG_EXP_2 – 2 番目の列の集計。

  列でサポートされている型: Byte (バイト)、Decimal (十進数)、Double (倍精度浮動小数点数)、Float (浮動小数点数)、Integer (整数)、Long (整数)、Short (整数)

  列でサポートされている型: Byte (バイト)、Decimal (十進数)、Double (倍精度浮動小数点数)、Float (浮動小数点数)、Integer (整数)、Long (整数)、Short (整数)
- EXPRESSION – ルールタイプの応答に対して実行し、論地値を生成するための式。詳細については、「表現」を参照してください。

例: 合計を使用した Aggregate Match

次のルール例では、amount 列の値の合計が、total_amount 列の値の合計と完全に等しいかをチェックします。

```
AggregateMatch "sum(amount)" "sum(total_amount)" = 1.0
```

例: 平均を使用した Aggregate Match

次のルール例では、ratings 列の値の平均が、reference データセットの ratings 列における値の平均の、90% 以上と等しいかをチェックします。参照データセットは、ETL またはデータカタログ体験の、追加のデータソースとして提供されます。

AWS Glue ETL では、以下を使用できます。

```
AggregateMatch "avg(ratings)" "avg(reference.ratings)" >= 0.9
```

AWS Glue データカタログでは、以下を使用できます。

```
AggregateMatch "avg(ratings)" "avg(database_name.tablename.ratings)" >= 0.9
```

Null 動作

AggregateMatch ルールは、集計方法 (合計/平均) の計算で NULL 値の行を無視します。例:

```
+---+-----------+
|id |units      |
+---+-----------+
|100|0          |
|101|null       |
|102|20         |
|103|null       |
|104|40         |
+---+-----------+
```

units 列の平均は (0 + 20 + 40) / 3 = 20 になります。行 101 および 103 はこの計算の対象になりません。

ColumnCorrelation

特定の表現と対応させて、2 つの列の間での相関関係をチェックします。AWSGlue Data Quality では、ピアソン相関係数を使用して 2 つの列間の線形相関を測定します。この結果では、-1 から 1 までの数値により、相関関係の強さと方向性が示されます。

[Syntax] (構文)

```
ColumnCorrelation <COL_1_NAME> <COL_2_NAME> <EXPRESSION>
```

- COL_1_NAME – 品質評価ルールの評価に対応させる、最初の列の名前。

  列でサポートされている型: Byte (バイト)、Decimal (十進数)、Double (倍精度浮動小数点数)、Float (浮動小数点数)、Integer (整数)、Long (整数)、Short (整数)
- COL_2_NAME – 品質評価ルールの評価に対応させる、2 番目の列の名前。

列でサポートされている型: Byte (バイト)、Decimal (十進数)、Double (倍精度浮動小数点数)、Float (浮動小数点数)、Integer (整数)、Long (整数)、Short (整数)

- EXPRESSION – ルールタイプの応答に対して実行し、論地値を生成するための式。詳細については、「表現」を参照してください。

例: 列の相関

次のルール例では、列 height と weight の間の相関係数が、強い正の (係数値が 0.8 より大きい) 相関を示しているかをチェックします。

```
ColumnCorrelation "height" "weight" > 0.8
```

```
ColumnCorrelation "weightinkgs" "Salary" > 0.8 where "weightinkgs > 40"
```

動的ルールの例

- `ColumnCorrelation "colA" "colB" between min(last(10)) and max(last(10))`
- `ColumnCorrelation "colA" "colB" < avg(last(5)) + std(last(5))`

Null 動作

ColumnCorrelation ルールは、相関の計算で NULL 値を含む行を無視します。例:

```
+---+-----------+
|id |units      |
+---+-----------+
|100|0          |
|101|null       |
|102|20         |
|103|null       |
|104|40         |
+---+-----------+
```

行 101 および 103 は無視され、ColumnCorrelation は 1.0 になります。

ColumnCount

特定の式を参照しながら、プライマリデータセットの列数を確認します。この式では、> や < などの演算子を使用して、行数または行の範囲を指定できます。

[Syntax] (構文)

```
ColumnCount <EXPRESSION>
```

- EXPRESSION – ルールタイプの応答に対して実行し、論地値を生成するための式。詳細については、「表現」を参照してください。

例: 列数の数値による確認

次のルール例では、列数が特定の範囲内にあるかを確認します。

```
ColumnCount between 10 and 20
```

動的ルールの例

- `ColumnCount >= avg(last(10))`
- `ColumnCount between min(last(10))-1 and max(last(10))+1`

ColumnDataType

特定の列の値を Apache Spark で指定されたタイプにキャストできるかどうかを確認します。`with threshold` 式を承諾し、列内の値のサブセットをチェックします。

[Syntax] (構文)

```
ColumnDataType <COL_NAME> = <EXPECTED_TYPE>
```

- COL_NAME – データ品質ルールを評価する対象となる列の名前。

  サポートされている列の型: String 型

  列でサポートされている型: Byte (バイト)、Decimal (十進数)、Double (倍精度浮動小数点数)、Float (浮動小数点数)、Integer (整数)、Long (整数)、Short (整数)

- EXPECTED_TYPE — 列内の想定されている値の型。

  サポートされている値: Boolean、Date、Timestamp、Integer、Double、Float、Long

  列でサポートされている型: Byte (バイト)、Decimal (十進数)、Double (倍精度浮動小数点数)、Float (浮動小数点数)、Integer (整数)、Long (整数)、Short (整数)
- EXPRESSION – 想定されている型の、値の割合を指定するオプションの式。

  列でサポートされている型: Byte (バイト)、Decimal (十進数)、Double (倍精度浮動小数点数)、Float (浮動小数点数)、Integer (整数)、Long (整数)、Short (整数)

例: 文字列としての、列データ型の整数。

次のルール例では、所定の列の値 (string 型) が整数としてキャストできるかどうかを確認します。

```
ColumnDataType "colA" = "INTEGER"
```

例: 文字列としての列データ型の整数が、値のサブセットをチェックします。

次のルール例は、所定の列の値 (string 型) の 90% 以上が整数としてキャストできるかどうかを確認します。

```
ColumnDataType "colA" = "INTEGER" with threshold > 0.9
```

ColumnExists

列が存在するかどうかを確認します。

[Syntax] (構文)

```
ColumnExists <COL_NAME>
```

- COL_NAME – データ品質ルールを評価する対象となる列の名前。

  列でサポートされている型: 任意の型

例: 列の存在

次のルール例では、`Middle_Name` という名前の列が存在するかどうかを確認します。

```
ColumnExists "Middle_Name"
```

ColumnLength

列内にある各行の長さが、特定の表現に適合しているかどうかを確認します。

[Syntax] (構文)

```
ColumnLength <COL_NAME><EXPRESSION>
```

- COL_NAME – データ品質ルールを評価する列の名前。

  列でサポートされる型: String
- EXPRESSION – ブール値を生成するためにルールタイプのレスポンスに対して実行する式。詳細については、「表現」を参照してください。

例: 列内の行の長さ

次のルール例では、Postal_Code という名前の列内にある各行の値が、5 文字の長さであるかどうかを確認します。

```
ColumnLength "Postal_Code" = 5
ColumnLength "weightinkgs" = 2 where "weightinkgs > 10"
```

Null 動作

ColumnLength ルールは、NULL を長さ 0 の文字列として扱います。NULL 行には、次の内容が適用されます。

```
ColumnLength "Postal_Code" > 4 # this will fail
```

```
ColumnLength "Postal_Code" < 6 # this will succeed
```

次の複合ルールの例では、NULL 値を明示的に満たさない方法が示されています。

```
(ColumnLength "Postal_Code" > 4) AND (ColumnValues "Postal_Code" != NULL)
```

ColumnNamesMatchPattern

プライマリデータセットの全列の名前が、所定の正規表現と一致するかをチェックします。

[Syntax] (構文)

```
ColumnNamesMatchPattern <PATTERN>
```

- PATTERN – データ品質ルールを評価する際の基準となるパターン。

  列でサポートされている型: Byte (バイト)、Decimal (十進数)、Double (倍精度浮動小数点数)、Float (浮動小数点数)、Integer (整数)、Long (整数)、Short (整数)

例: 列名がパターンと一致する

次のルール例では、すべての列がプレフィックス「aws_」から始まっているかをチェックします。

```
ColumnNamesMatchPattern "aws_.*"
ColumnNamesMatchPattern "aws_.*" where "weightinkgs > 10"
```

ColumnValues

列内の値に対して式を実行します。

[Syntax] (構文)

```
ColumnValues <COL_NAME> <EXPRESSION>
```

- COL_NAME – データ品質ルールを評価する対象となる列の名前。

  列でサポートされている型: 任意の型
- EXPRESSION – ルールタイプの応答に対して実行し、論地値を生成するための式。詳細については、「表現」を参照してください。

例: 許可される値

次のルール例では、指定された列の各値が、許可された値 (null、空、空白のみの文字列を含む) のセットの中に含まれているかどうかを確認します。

```
ColumnValues "Country" in [ "US", "CA", "UK", NULL, EMPTY, WHITESPACES_ONLY ]
ColumnValues "gender" in ["F", "M"] where "weightinkgs < 10"
```

例: 正規表現

次のルール例では、列内の値が正規表現と照合されます。

```
ColumnValues "First_Name" matches "[a-zA-Z]*"
```

例: 日付値

次のルール例では、日付列内の値が日付表現に適合しているかをチェックします。

```
ColumnValues "Load_Date" > (now() - 3 days)
```

例: 数値

次のルール例では、列内の値が特定の数値制約に適合しているかどうかをチェックします。

```
ColumnValues "Customer_ID" between 1 and 2000
```

Null 動作

すべての ColumnValues ルール (!= および NOT IN 以外) では、NULL 行はルールを満たしません。null 値が原因でルールが失敗した場合、失敗理由には次の内容が表示されます。

```
Value: NULL does not meet the constraint requirement!
```

次の複合ルールの例では、NULL 値を明示的に満たす方法が示されています。

```
(ColumnValues "Age" > 21) OR (ColumnValues "Age" = NULL)
```

!= および not in 構文を使用する否定 ColumnValues ルールは、NULL 行に渡されます。例:

```
ColumnValues "Age" != 21
```

```
ColumnValues "Age" not in [21, 22, 23]
```

次の例では、NULL 値を明示的に満たさない方法を示されています。

```
(ColumnValues "Age" != 21) AND (ColumnValues "Age" != NULL)
```

```
ColumnValues "Age" not in [21, 22, 23, NULL]
```

Completeness

列内の完全な (NULL 以外の) 値を指定された式と照合し、パーセンテージで示します。

[Syntax] (構文)

```
Completeness <COL_NAME> <EXPRESSION>
```

- COL_NAME – データ品質ルールを評価する対象となる列の名前。

  列でサポートされている型: 任意の型
- EXPRESSION – ルールタイプの応答に対して実行し、論地値を生成するための式。詳細については、「表現」を参照してください。

例: NULL 値のパーセンテージ

次のルール例では、列内に完全な値が 95% 以上存在するかどうかをチェックします。

```
Completeness "First_Name" > 0.95
Completeness "First_Name" > 0.95 where "weightinkgs > 10"
```

動的ルールの例

- `Completeness "colA" between min(last(5)) - 1 and max(last(5)) + 1`
- `Completeness "colA" <= avg(last(10))`

Null 動作

CSV データ形式に関するメモ: CSV 列の空白行は、複数の動作を表示することがあります。

- 列が `String` 型の場合、空白行は空の文字列として認識され、`Completeness` ルールを満たします。
- 列が `Int` のような別のデータ型の場合、空白行は `NULL` として認識され、`Completeness` ルールを満たしません。

CustomSQL

このルールタイプは、次の 2 つのユースケースをサポートするように拡張されました。

- データセットに対してカスタム SQL ステートメントを実行し、その戻り値を指定された式と対応させて確認します。
- SELECT ステートメントで列名を指定するカスタム SQL ステートメントを実行し、それを何らかの条件と比較して行レベルの結果を取得します。

[Syntax] (構文)

```
CustomSql <SQL_STATEMENT> <EXPRESSION>
```

- SQL_STATEMENT – 二重引用符で囲まれた単一の数値を返す SQL ステートメント。
- EXPRESSION – ルールタイプの応答に対して実行し、論地値を生成するための式。詳細については、「表現」を参照してください。

例: ルール全体の結果を取得するカスタム SQL

このルール例では、データセットのレコード数を取得するための、SQL ステートメントを使用します。その後、このルールは、レコード数が 10 から 20 の間であることを確認します。

```
CustomSql "select count(*) from primary" between 10 and 20
```

例: 行レベルの結果を取得するカスタム SQL

このサンプルルールでは、SELECT ステートメントで列名を指定し、それを何らかの条件と比較して行レベルの結果を取得する SQL ステートメントを使用しています。しきい値条件式は、ルール全体が失敗するまでに失敗するレコード数のしきい値を定義します。ルールには条件とキーワードの両方を一緒に含めることはできないことに注意してください。

```
CustomSql "select Name from primary where Age  > 18"
```

or

```
CustomSql "select Name from primary where Age > 18" with threshold  > 3
```

> **Important**
>
> primary エイリアスは、評価するデータセットの名前の代替です。コンソールでビジュアル ETL ジョブを使用する場合、primary は常に、EvaluateDataQuality.apply() 変換に渡されている DynamicFrame を表します。AWS Glue データカタログを使用して、テーブルに対して 品質評価タスクを実行する場合、primary はテーブルを表します。

AWS Glue データカタログを使用している場合は、実際のテーブル名を使用することもできます。

```
CustomSql "select count(*) from database.table" between 10 and 20
```

複数のテーブルを結合して、異なるデータ要素を比較することもできます。

```
CustomSql "select count(*) from database.table inner join database.table2 on id1 = id2"
 between 10 and 20
```

AWS Glue ETL では、CustomSQL はデータ品質チェックに失敗したレコードを特定できます。これを機能させるには、データ品質を評価する主テーブルの一部であるレコードを返す必要があります。クエリの一部として返されたレコードは成功と見なされ、返されなかったレコードは不合格と見なされます。これは、CustomSQL クエリの結果を元のデータセットに結合することで機能します。SQL クエリの複雑さによっては、パフォーマンスに影響する場合があります。

これを実行するには:

- 主テーブルから少なくとも 1 つの列を選択する必要があります。
  - `select count(*) from primary` は、OVERALL CustomSQL DQ ルールでは有効なクエリですが、行レベルのカスタム SQL の有効なクエリではありません。
  - このルールは、評価中に次のエラーをスローします。 `The output from CustomSQL must contain at least one column that matches the input dataset for AWS Glue Data Quality to provide row level results. The SQL query is a valid query but the columns from the SQL result are not present in the Input Dataset. Ensure that matching columns are returned from the SQL.`
- SQL クエリで、テーブルから「プライマリキー」を選択するか、複合キーを形成する列の組み合わせを選択します。そうしない場合、重複行とのマッチングによって結果に矛盾が生じたり、パフォーマンスが低下したりする可能性があります。
- 参照テーブルからキーを選択するのではなく、主テーブルにあるキーのみを選択します。

次のルールにより、経過日数が 100 未満のレコードは成功と見なされ、それ以上のレコードは不合格とマークされます。

```
CustomSql "select id from primary where age < 100"
```

次の CustomSQL ルールは、レコードの 50% の経過日数が 10 を超えた場合は合格とし、失敗したレコードも特定します。この CustomSQL によって返されたレコードは合格と見なされ、返されなかったレコードは不合格と見なされます。

```
CustomSQL "select ID, CustomerID from primary where age > 10" with threshold > 0.5
```

メモ: データセットにないレコードを返すと、CustomSQL ルールは失敗します。

DataFreshness

現在の時刻と日付列の値との差を評価して、列内のデータがどの程度新しいかをチェックします。このルールタイプで時間ベースの式を指定することで、列の値を最新に保つことができます。

[Syntax] (構文)

```
DataFreshness <COL_NAME> <EXPRESSION>
```

- COL_NAME – データ品質ルールを評価する対象となる列の名前。

  列でサポートされる型: Date
- EXPRESSION – 時間または日付の数値表現。表現の中では、時間単位を指定する必要があります。

例: データの新しさ

次のルール例では、データの新しさをチェックします。

```
DataFreshness "Order_Date" <= 24 hours
DataFreshness "Order_Date" between 2 days and 5 days
```

Null 動作

DataFreshness ルールは、NULL 値がある行を満たしません。null 値が原因でルールが失敗した場合、失敗理由には次の内容が表示されます。

```
80.00 % of rows passed the threshold
```

満たさなかった 20% の行には、NULL を含む行が対象となる場合。

次の複合ルールの例では、NULL 値を明示的に満たす方法が示されています。

```
(DataFreshness "Order_Date" <= 24 hours) OR (ColumnValues "Order_Date" = NULL)
```

Amazon S3 オブジェクトのデータの鮮度

Amazon S3 ファイルの作成時間に基づいて、データの鮮度を検証する必要がある場合があります。これを行うには、次のコードを使用してタイムスタンプを取得してデータフレームに追加し、データの鮮度のチェックを適用します。

```
df = glueContext.create_data_frame.from_catalog(database = "default", table_name =
 "mytable")
df = df.withColumn("file_ts", df["_metadata.file_modification_time"])

Rules = [
 DataFreshness "file_ts" < 24 hours
]
```

DatasetMatch

プライマリデータセットのデータが参照データセットのデータと一致するかをチェックします。これら 2 つのデータセットは、入力されたキー列マッピングを使用して結合されます。これらの列のみでデータが等しいかを確認する場合は、追加の列マッピングを提供できます。DataSetMatch を操作するときは、結合キーは NULL ではなく一意でなければなりません (プライマリキーであること)。これらの条件を満たしていないと、「Provided key map not suitable for given data frames」というエラーメッセージが表示されます。一意の結合キーを使用できない場合は、AggregateMatch のような他のルール型を使用してサマリーデータで照合することを検討します。

[Syntax] (構文)

```
DatasetMatch <REFERENCE_DATASET_ALIAS> <JOIN CONDITION WITH
 MAPPING> <OPTIONAL_MATCH_COLUMN_MAPPINGS> <EXPRESSION>
```

- REFERENCE_DATASET_ALIAS – プライマリデータセットのデータと比較する、参照データセットのエイリアス。

- KEY_COLUMN_MAPPINGS – データセットのキーとなる列名のカンマ区切りリスト。列名が両方のデータセットで同一でない場合は、-> で区切る必要があります。
- OPTIONAL_MATCH_COLUMN_MAPPINGS – 特定の列のみでデータの一致を確認したい場合はこのパラメータを指定します。これは、キー列のマッピングと同じ構文を使用します。このパラメータを指定しないと、残りすべての列のデータと照合されます。残りの非キー列は、両方のデータセットで同じ名前を持っている必要があります。
- EXPRESSION – ルールタイプの応答に対して実行し、論地値を生成するための式。詳細については、「表現」を参照してください。

例: ID 列を使用したセットデータセットの照合

次のルール例では、[ID] 列を使用して 2 つのデータセットを結合し、プライマリデータセットの 90% 以上が参照データセットと一致していることを確認します。この場合、すべての列を比較します。

```
DatasetMatch "reference" "ID" >= 0.9
```

例: 複数のキー列を使用したセットデータセットの照合

次の例では、プライマリデータセットと参照データセットで、キー列の名前が異なります。ID_1 と ID_2 は、結合して、プライマリデータセットの複合キーを構成します。ID_ref1 と ID_ref2 は、結合して、参照データセットの複合キーを構成します。このシナリオでは、特殊な構文を使用して列名を指定できます。

```
DatasetMatch "reference" "ID_1->ID_ref1,ID_ref2->ID_ref2" >= 0.9
```

例: 複数のキー列を使用してセットデータセットを照合し、特定の列が一致していることを確認する

この例は、前述の例に基づいています。金額を含む列のみが一致しているかを、確認しようとしています。この列の名前は、プライマリデータセットでは Amount1、参照データセットでは Amount2 です。これらを完全に一致させたいと思います。

```
DatasetMatch "reference" "ID_1->ID_ref1,ID_2->ID_ref2" "Amount1->Amount2" >= 0.9
```

DistinctValuesCount

列内の異なる値の数を、指定された式に対応させて確認します。

[Syntax] (構文)

```
DistinctValuesCount <COL_NAME> <EXPRESSION>
```

- COL_NAME – データ品質ルールを評価する対象となる列の名前。

  列でサポートされている型: 任意の型
- EXPRESSION – ルールタイプの応答に対して実行し、論地値を生成するための式。詳細については、「表現」を参照してください。

例: 列内の異なる値のカウント

次のルール例では、State という名前の列に 3 つ以上の異なる値が含まれていることを確認します。

```
DistinctValuesCount "State" > 3
DistinctValuesCount "Customer_ID" < 6  where "Customer_ID < 10"
```

動的ルールの例

- `DistinctValuesCount "colA" between avg(last(10))-1 and avg(last(10))+1`
- `DistinctValuesCount "colA" <= index(last(10),2) + std(last(5))`

Entropy

列のエントロピー値が、特定の式と一致するかどうかを確認します。エントロピーは、メッセージに含まれる情報のレベルを測定します。エントロピーでは、列内の値に関する特定の確率分布に基づき、値を区別するのに必要なビット数を表します。

[Syntax] (構文)

```
Entropy <COL_NAME> <EXPRESSION>
```

- COL_NAME – データ品質ルールを評価する対象となる列の名前。

  列でサポートされている型: 任意の型
- EXPRESSION – ルールタイプの応答に対して実行し、論地値を生成するための式。詳細については、「表現」を参照してください。

例: 列のエントロピー

次のルール例では、Feedback という名前の列のエントロピー値が、1 より大きいことを確認します。

```
Entropy "Star_Rating" > 1
Entropy "First_Name" > 1  where "Customer_ID < 10"
```

動的ルールの例

- `Entropy "colA" < max(last(10))`
- `Entropy "colA" between min(last(10)) and max(last(10))`

IsComplete

列のすべての値が完全 (NULL 以外) かどうかをチェックします。

[Syntax] (構文)

```
IsComplete <COL_NAME>
```

- COL_NAME – データ品質ルールを評価する対象となる列の名前。

  列でサポートされている型: 任意の型

例: NULL 値

次の例では、email という名前が付いた列内で、値がすべて NULL 以外であるかどうかをチェックします。

```
IsComplete "email"
IsComplete "Email" where "Customer_ID between 1 and 50"
IsComplete "Customer_ID"  where "Customer_ID < 16 and Customer_ID != 12"
IsComplete "passenger_count" where "payment_type<>0"
```

Null 動作

CSV データ形式に関するメモ: CSV 列の空白行は、複数の動作を表示することがあります。

- 列が String 型の場合、空白行は空の文字列として認識され、Completeness ルールを満たします。
- 列が Int のような別のデータ型の場合、空白行は NULL として認識され、Completeness ルールを満たしません。

IsPrimaryKey

列にプライマリキーが含まれているかどうかを確認します。列内の値がすべて一意であり、かつそれらすべてが完全な (NULL 以外) 場合、その列にはプライマリキーが含まれます。複数の列を持つプライマリキーを確認することもできます。

[Syntax] (構文)

```
IsPrimaryKey <COL_NAME>
```

- COL_NAME – データ品質ルールを評価する対象となる列の名前。

  列でサポートされている型: 任意の型

例; プライマリキー

次のルール例では、Customer_ID という名前の列にプライマリキーが含まれているかどうかを確認します。

```
IsPrimaryKey "Customer_ID"
IsPrimaryKey "Customer_ID" where "Customer_ID < 10"
```

例:複数の列を持つ主プライマリキー。以下のいずれの例も有効です。

```
IsPrimaryKey "colA" "colB"
IsPrimaryKey "colA" "colB" "colC"
IsPrimaryKey colA "colB" "colC"
```

IsUnique

列のすべての値が一意であるかどうかをチェックし、ブール型の値を返します。

[Syntax] (構文)

```
IsUnique <COL_NAME>
```

- COL_NAME – データ品質ルールを評価する対象となる列の名前。

  列でサポートされている型: 任意の型

例

次のルール例では、email という名前の列で、すべての値が一意であるかどうかをチェックします。

```
IsUnique "email"
IsUnique "Customer_ID" where "Customer_ID < 10"]
```

次のルール例は、複数の列をチェックします。

```
IsUnique "vendorid" "tpep_pickup_datetime"
```

平均値

列内にあるすべての値の mean 値 (平均) が、特定の表現に適合するかどうかを確認します。

[Syntax] (構文)

```
Mean <COL_NAME> <EXPRESSION>
```

- COL_NAME – データ品質ルールを評価する対象となる列の名前。

  列でサポートされている型: Byte (バイト)、Decimal (十進数)、Double (倍精度浮動小数点数)、Float (浮動小数点数)、Integer (整数)、Long (整数)、Short (整数)
- EXPRESSION – ルールタイプの応答に対して実行し、論地値を生成するための式。詳細については、「表現」を参照してください。

例: 平均値

次のルール例では、列のすべての値の平均が、しきい値を超えているかどうかを確認します。

```
Mean "Star_Rating" > 3
```

```
Mean "Salary" < 6200 where "Customer_ID < 10"
```

動的ルールの例

- `Mean "colA" > avg(last(10)) + std(last(2))`
- `Mean "colA" between min(last(5)) - 1 and max(last(5)) + 1`

Null 動作

Mean ルールは、平均の計算で NULL 値を含む行を無視します。例:

```
+---+-----------+
|id |units      |
+---+-----------+
|100|0          |
|101|null       |
|102|20         |
|103|null       |
|104|40         |
+---+-----------+
```

units 列の平均は (0 + 20 + 40) / 3 = 20 になります。行 101 および 103 はこの計算の対象になりません。

ReferentialIntegrity

プライマリデータセットに存在する一連の列の値が、どの程度、参照データセットの一連の列における値のサブセットになっているかをチェックします。

[Syntax] (構文)

```
ReferentialIntegrity <PRIMARY_COLS> <REFERENCE_DATASET_COLS> <EXPRESSION>
```

- PRIMARY_COLS — プライマリデータセット内の列名のカンマ区切りリスト。

  列でサポートされている型: Byte (バイト)、Decimal (十進数)、Double (倍精度浮動小数点数)、Float (浮動小数点数)、Integer (整数)、Long (整数)、Short (整数)
- REFERENCE_DATASET_COLS – このパラメータには、ピリオドで区切られた 2 つの要素が含まれます。前半の部分は、参照データセットのエイリアスです。後半の部分は、中かっこで囲まれた参照データセット内の列名のカンマ区切りリストです。

列でサポートされている型: Byte (バイト)、Decimal (十進数)、Double (倍精度浮動小数点数)、Float (浮動小数点数)、Integer (整数)、Long (整数)、Short (整数)

- EXPRESSION – ルールタイプの応答に対して実行し、論地値を生成するための式。詳細については、「表現」を参照してください。

例: 郵便番号列の参照整合性の確認

次のルール例では、zipcode 列の値の 90% 以上が reference データセットの zipcode 列にあることを確認します。

```
ReferentialIntegrity "zipcode" "reference.zipcode" >= 0.9
```

例: 都市列と州列の参照整合性の確認

次の例では、都市と州の情報を含む列が、プライマリデータセットと参照データセットの中にあります。列の名前は、両方のデータセットで異なっています。このルールは、プライマリデータセットの列の値セットが、参照データセットの列の値セットと完全に等しいかをチェックします。

```
ReferentialIntegrity "city,state" "reference.{ref_city,ref_state}" = 1.0
```

動的ルールの例

- `ReferentialIntegrity "city,state" "reference.{ref_city,ref_state}" > avg(last(10))`
- `ReferentialIntegrity "city,state" "reference.{ref_city,ref_state}" between min(last(10)) - 1 and max(last(10)) + 1`

RowCount

特定の表現を参照しながら、データセットの行数を確認します。この式では、> や < などの演算子を使用して、行数または行の範囲を指定します。

[Syntax] (構文)

```
RowCount <EXPRESSION>
```

- EXPRESSION – ルールタイプの応答に対して実行し、論地値を生成するための式。詳細については、「表現」を参照してください。

例: 行数の数値による確認

次のルール例では、行数が特定の範囲内にあるかどうかを確認します。

```
RowCount between 10 and 100
RowCount between 1 and 50 where "Customer_ID < 10"
```

動的ルールの例

```
RowCount > avg(last(10)) *0.8
```

RowCountMatch

プライマリデータセットの行数と参照データセットの行数の比率を、所定の式に照らして確認します。

[Syntax] (構文)

```
RowCountMatch <REFERENCE_DATASET_ALIAS> <EXPRESSION>
```

- REFERENCE_DATASET_ALIAS – 行数の比較に使用する参照データセットのエイリアス。

  列でサポートされている型: Byte (バイト)、Decimal (十進数)、Double (倍精度浮動小数点数)、Float (浮動小数点数)、Integer (整数)、Long (整数)、Short (整数)
- EXPRESSION – ルールタイプの応答に対して実行し、論地値を生成するための式。詳細については、「表現」を参照してください。

例: 参照データセットに対する行数の確認

次のルール例は、プライマリデータセットの行数が、参照データセットの行数の 90% 以上あるかをチェックします。

```
RowCountMatch "reference" >= 0.9
```

StandardDeviation

列のすべての値の標準偏差が、特定の表現に適合するかを確認します。

[Syntax] (構文)

```
StandardDeviation <COL_NAME> <EXPRESSION>
```

- COL_NAME – データ品質ルールを評価する対象となる列の名前。

  列でサポートされている型: Byte (バイト)、Decimal (十進数)、Double (倍精度浮動小数点数)、Float (浮動小数点数)、Integer (整数)、Long (整数)、Short (整数)
- EXPRESSION – ルールタイプの応答に対して実行し、論地値を生成するための式。詳細については、「表現」を参照してください。

例: 標準偏差

次のルール例では、colA という名前の列内の値の標準偏差が、指定された値より小さいかどうかを確認します。

```
StandardDeviation "Star_Rating" < 1.5
StandardDeviation "Salary" < 3500 where "Customer_ID < 10"
```

動的ルールの例

- `StandardDeviation "colA" > avg(last(10) + 0.1`
- `StandardDeviation "colA" between min(last(10)) - 1 and max(last(10)) + 1`

Null 動作

StandardDeviation ルールは、標準偏差の計算で NULL 値を含む行を無視します。例:

```
+---+-----------+-----------+
|id |units1     |units2     |
+---+-----------+-----------+
|100|0          |0          |
|101|null       |0          |
|102|20         |20         |
|103|null       |0          |
|104|40         |40         |
```

```
+---+-----------+-----------+
```

units1 列の標準偏差は行 101 および 103 を対象にせず、結果は 16.33 になります。units2 列の標準偏差は 16 になります。

Sum

列のすべての値の合計値を、特定の表現を参照しながら確認します。

[Syntax] (構文)

```
Sum <COL_NAME> <EXPRESSION>
```

- COL_NAME – データ品質ルールを評価する対象となる列の名前。

  列でサポートされている型: Byte (バイト)、Decimal (十進数)、Double (倍精度浮動小数点数)、Float (浮動小数点数)、Integer (整数)、Long (整数)、Short (整数)
- EXPRESSION – ルールタイプの応答に対して実行し、論地値を生成するための式。詳細については、「表現」を参照してください。

例: 合計

次のルール例では、列のすべての値の合計が、特定のしきい値を超えているかどうかを確認します。

```
Sum "transaction_total" > 500000
Sum "Salary" < 55600 where "Customer_ID < 10"
```

動的ルールの例

- `Sum "ColA" > avg(last(10))`
- `Sum "colA" between min(last(10)) - 1 and max(last(10)) + 1`

Null 動作

Sum ルールは、合計の計算で NULL 値を含む行を無視します。例:

```
+---+-----------+
|id |units      |
+---+-----------+
```

```
|100|0          |
|101|null       |
|102|20         |
|103|null       |
|104|40         |
+---+-----------+
```

units 列の合計は行 101 および 103 を対象にせず、(0 + 20 + 40) = 60 になります。

SchemaMatch

プライマリデータセットのスキーマが参照データセットのスキーマと一致しているかをチェックします。スキーマのチェックは列ごとに行われます。名前と型が同じであれば、2 つの列のスキーマは一致します。行の順序は関係ありません。

[Syntax] (構文)

```
SchemaMatch <REFERENCE_DATASET_ALIAS> <EXPRESSION>
```

- REFERENCE_DATASET_ALIAS – スキーマの比較に使用する参照データセットのエイリアスです。

  列でサポートされている型: Byte (バイト)、Decimal (十進数)、Double (倍精度浮動小数点数)、Float (浮動小数点数)、Integer (整数)、Long (整数)、Short (整数)
- EXPRESSION – ルールタイプの応答に対して実行し、論地値を生成するための式。詳細については、「表現」を参照してください。

例: SchemaMatch

次のルール例は、プライマリデータセットのスキーマが参照データセットのスキーマと厳密に一致しているかをチェックします。

```
SchemaMatch "reference" = 1.0
```

Uniqueness

特定の表現と対応させて、列内にある一意の値の個数を、パーセンテージにより確認します。一意の値とは、1 つのみ存在する値です。

[Syntax] (構文)

```
Uniqueness <COL_NAME> <EXPRESSION>
```

- COL_NAME – データ品質ルールを評価する対象となる列の名前。

  列でサポートされている型: 任意の型
- EXPRESSION – ルールタイプの応答に対して実行し、論地値を生成するための式。詳細については、「表現」を参照してください。

例

次のルール例では、列内での一意の値の割合が、特定の数値基準と一致するかどうかを確認します。

```
Uniqueness "email" = 1.0
Uniqueness "Customer_ID" != 1.0 where "Customer_ID < 10"
```

次のルール例は、複数の列をチェックします。

```
Uniqueness "vendorid" "tpep_pickup_datetime" = 1
```

動的ルールの例

- `Uniqueness "colA" between min(last(10)) and max(last(10))`
- `Uniqueness "colA" >= avg(last(10))`

UniqueValueRatio

特定の表現を参照して、列での一意な値の比率をチェックします。個別値の比率は、列内の一意の値の個数を、個別なすべての値の個数で割ったものです。一意の値とは 1 つだけ含まれる値であり、一方、個別の値は少なくとも 1 つ含まれている値です。

例えば、[a, a, b] のセットには 1 つの一意の値 (b) と 2 つの個別の値 (a と b) が含まれています。したがって、このセットの一意な値の比率は ½ = 0.5 です。

[Syntax] (構文)

```
UniqueValueRatio <COL_NAME> <EXPRESSION>
```

- COL_NAME – データ品質ルールを評価する対象となる列の名前。

  列でサポートされている型: 任意の型
- EXPRESSION – ルールタイプの応答に対して実行し、論地値を生成するための式。詳細については、「表現」を参照してください。

例: 一意な値の比率

この例では、列の一意な値の比率を、値の範囲と比較します。

```
UniqueValueRatio "test_score" between 0 and 0.5
UniqueValueRatio "Customer_ID" between 0 and 0.9 where "Customer_ID < 10"
```

動的ルールの例

- `UniqueValueRatio "colA" > avg(last(10))`
- `UniqueValueRatio "colA" <= index(last(10),2) + std(last(5))`

DetectAnomalies

特定のデータ品質ルールの異常を検出します。DetectAnomalies ルールを実行するたびに、そのルールの評価値が保存されます。十分なデータが収集されると、異常検出アルゴリズムはその特定のルールに関するすべての履歴データを取得し、異常検出を実行します。DetectAnomalies ルールは、異常が検出されると失敗します。どのような異常が検出されたかについての詳細は、「観察結果」で確認できます。

[Syntax] (構文)

```
DetectAnomalies <RULE_NAME> <RULE_PARAMETERS>
```

RULE_NAME – 異常の評価、検出を希望するルールの名前。サポートされているルール:

- "RowCount"
- "Completeness"
- "Uniqueness"
- "Mean"

- "Sum"
- "StandardDeviation"
- "Entropy"
- "DistinctValuesCount"
- "UniqueValueRatio"
- "ColumnLength"
- "ColumnValues"
- "ColumnCorrelation"
- "CustomSQL"
- "ColumnCount"

RULE_PARAMETERS — 一部のルールでは、実行に追加のパラメータが必要です。必要なパラメータについては、該当するルールのドキュメントを参照してください。

例: RowCount の異常

例えば、RowCount の異常を検出したい場合は、RowCount をルール名として指定します。

```
DetectAnomalies "RowCount"
```

例: ColumnLength の異常

例えば、ColumnLength の異常を検出したい場合は、ルール名として ColumnLength を指定し、列名を指定します。

```
DetectAnomalies "ColumnLength" "id"
```

FileFreshness

FileFreshness を使用すると、指定した条件に基づいてデータファイルが最新になります。データファイルまたはフォルダ全体が最新であることを確認するために、ファイルの最終変更時刻を使用します。

このルールは 2 つのメトリクスを収集します:

- 設定したルールに基づく FileFreshness コンプライアンス
- ルールでスキャンされたファイル数

```
{"Dataset.*.FileFreshness.Compliance":1,"Dataset.*.FileCount":1}
```

異常検出では、これらのメトリクスは考慮されません。

ファイルの鮮度の確認

次のルールは、過去 24 時間に tickets.parquet が作成されていることを保証します。

```
FileFreshness "s3://amzn-s3-demo-bucket/artifacts/file/tickets/tickets.parquet" >
 (now() - 24 hours)
```

フォルダの鮮度の確認

次のルールは、フォルダ内のすべてのファイルが過去 24 時間以内に作成または変更された場合に適用されます。

```
FileFreshness "s3://bucket/" >= (now() -1 days)
FileFreshness "s3://amzn-s3-demo-bucket/artifacts/file/tickets/" >= (now() - 24 hours)
```

しきい値によるフォルダまたはファイルの鮮度の確認

次のルールは、フォルダ「tickets」内のファイルの 10% が過去 10 日間に作成または変更された場合に適用されます。

```
FileFreshness "s3://amzn-s3-demo-bucket/artifacts/file/tickets/" < (now() - 10 days)
 with threshold > 0.1
```

特定の日付のファイルまたはフォルダの確認

特定の日におけるファイルの鮮度を確認できます。

```
FileFreshness "s3://amzn-s3-demo-bucket/artifacts/file/tickets/" > "2020-01-01"
FileFreshness "s3://amzn-s3-demo-bucket/artifacts/file/tickets/" between "2023-01-01"
 and "2024-01-01"
```

ファイルまたはフォルダを時間とともに確認する

FileFreshness を使用して、特定の時間に基づいてファイルが到着したことを確認できます。

```
FileFreshness "s3://amzn-s3-demo-bucket/artifacts/file/tickets/" between now() and
 (now() - 45 minutes)
```

```
FileFreshness "s3://amzn-s3-demo-bucket/artifacts/file/tickets/" between "9:30 AM" and
 "9:30 PM"
FileFreshness "s3://amzn-s3-demo-bucket/artifacts/file/tickets/" > (now() - 10 minutes)
FileFreshness "s3://amzn-s3-demo-bucket/artifacts/file/tickets/" > now()
FileFreshness "s3://amzn-s3-demo-bucket/artifacts/file/tickets/" between (now() - 2
 hours) and (now() + 15 minutes)
FileFreshness "s3://amzn-s3-demo-bucket/artifacts/file/tickets/" between (now() - 3
 days) and (now() + 15 minutes)
FileFreshness "s3://amzn-s3-demo-bucket/artifacts/file/tickets/" between "2001-02-07"
 and (now() + 15 minutes)
FileFreshness "s3://amzn-s3-demo-bucket/artifacts/file/tickets/" > "21:45""
FileFreshness "s3://amzn-s3-demo-bucket/artifacts/file/tickets/" > "2024-01-01"
FileFreshness "s3://amzn-s3-demo-bucket/artifacts/file/tickets/" between "02:30" and
 "04:30"
FileFreshness "s3://amzn-s3-demo-bucket/artifacts/file/tickets/" between "9:30 AM" and
 "22:15"
```

主な考慮事項:

- FileFreshness では、日単位、時間単位、分単位を使用してファイルを評価できます
- 時間については、午前/午後と 24 時間に対応しています
- 上書きが指定されていない限り、時刻は UTC で計算されます
- 日付は時刻 00:00 に UTC で計算されます

時間ベースの FileFreshness は次のように機能します。

```
FileFreshness "s3://amzn-s3-demo-bucket/artifacts/file/tickets/" > "21:45"
```

- まず、UTC 形式で時刻「21:45」と今日の日付を組み合わせて日時フィールドを作成します
- 次に、日時は指定したタイムゾーンに変換されます
- 最後に、このルールが評価されます

オプションのファイルベースのルールタグ:

タグを使用すると、このルールの動作を制御できます。

recentFiles

このタグは、最新のファイルを最初に保持することで、処理されるファイルの数を制限します。

```
FileFreshness "s3://amzn-s3-demo-bucket/" between (now() - 100 minutes) and (now() + 10
 minutes) with recentFiles = 1
```

timeZone

許可されたタイムゾーンの上書きについては、サポートされているタイムゾーンの「Allowed Time Zones」を参照してください。

```
FileFreshness "s3://path/" > "21:45" with timeZone = "America/New_York"
```

```
FileFreshness "s3://path/" > "21:45" with timeZone = "America/Chicago"
```

```
FileFreshness "s3://path/" > "21:45" with timeZone = "Europe/Paris"
```

```
FileFreshness "s3://path/" > "21:45" with timeZone = "Asia/Shanghai"
```

```
FileFreshness "s3://path/" > "21:45" with timeZone = "Australia/Darwin"
```

データフレームから直接ファイル名を推測する

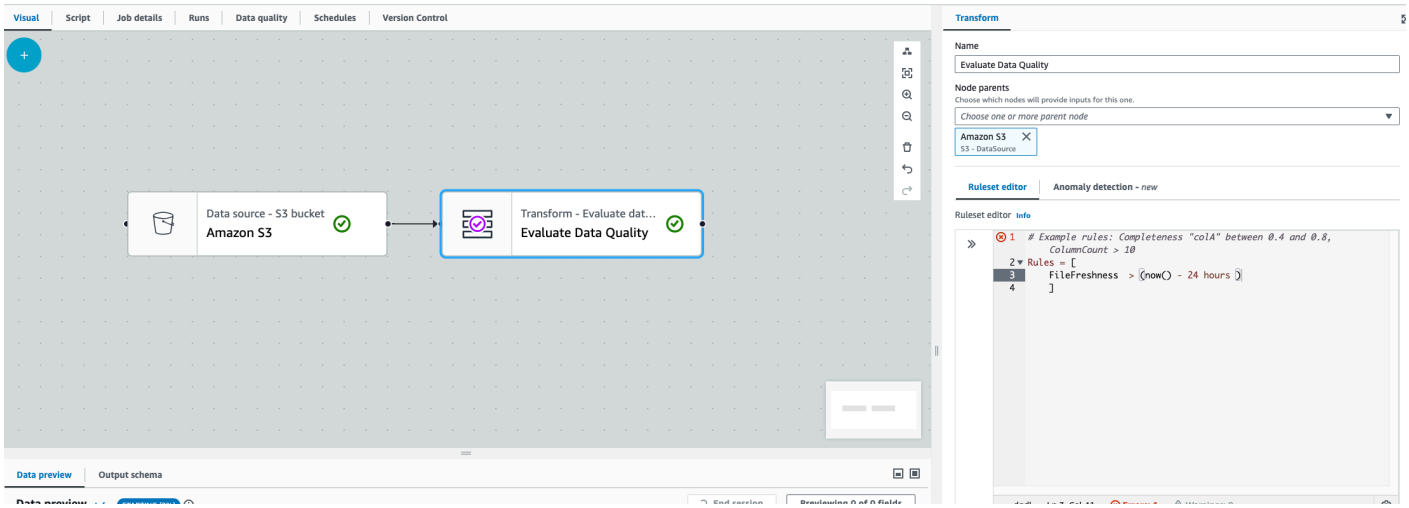
必ずしもファイルパスを指定する必要はありません。例えば、AWS Glue Data Catalog でルールを作成する場合、カタログテーブルが使用しているフォルダを見つけるのが難しい場合があります。AWSGlue Data Quality は、データフレームへの入力に使用される特定のフォルダまたはファイルを検索し、それらが最新かどうかを検出できます。

> **Note**
>
> この機能は、ファイルが DynamicFrame または DataFrame に正常に読み取られた場合にのみ機能します。

```
FileFreshness > (now() - 24 hours)
```

このルールは、動的フレームまたはデータフレームの入力に使用されるフォルダパスまたはファイルを検索します。これは、Amazon S3 パスまたは Amazon S3 ベースの AWS Glue Data Catalog テーブルで機能します。考慮事項がいくつかあります:

1. AWS Glue ETL では、Amazon S3 または AWS Glue Data Catalog 変換の直後に [EvaluateDataQuality] 変換が必要です。



2. このルールは AWS Glue インタラクティブセッションでは機能しません。

両方の場合でこれを試行した場合、または AWS Glue がファイルを見つけられなかった場合、AWS Glue は次のエラーがスローします: “`Unable to parse file path from DataFrame`”

FileMatch

FileMatch ルールを使用すると、ファイルを他のファイルやチェックサムと比較できます。これは、いくつかのシナリオで役立ちます:

1. 外部ソースから受信したファイルの検証: FileMatch を使用して、チェックサムと比較することで、外部ソースから正しいファイルを受け取ったことを確認できます。これにより、取り込んでいるデータの整合性を検証できます。
2. 2 つの異なるフォルダのデータの比較: FileMatch を使用して、2 つのフォルダ間でファイルを比較できます。

このルールは、ルールでスキャンされたファイルの数という 1 つのメトリクスを収集します。

```
{"Dataset.*.FileCount":1}
```

チェックサムを使用してファイルを検証します:

FileMatch はファイルとセットのチェックサムを受け入れ、少なくとも 1 つのチェックサムがファイルと一致することを確認します。

```
FileMatch "s3://amzn-s3-demo-bucket/file.json" in ["3ee0d8617ac041793154713e5ef8f319"]
 with hashAlgorithm = "MD5"
FileMatch "s3://amzn-s3-demo-bucket/file.json" in ["3ee0d8617ac041793154713e5ef8f319"]
 with hashAlgorithm = "SHA-1"
FileMatch "s3://amzn-s3-demo-bucket/file.json" in ["3ee0d8617ac041793154713e5ef8f319"]
 with hashAlgorithm = "SHA-256"
FileMatch "s3://amzn-s3-demo-bucket/file.json" in ["3ee0d8617ac041793154713e5ef8f319"]
```

以下の標準アルゴリズムがサポートされています:

- MD5
- SHA-1
- SHA-256

アルゴリズムを指定しない場合、デフォルトは SHA-256 です。

一連のチェックサムを使用してフォルダ内のすべてのファイルを検証します:

```
FileMatch "s3://amzn-s3-demo-bucket /" in ["3ee0d8617ac041793154713e5ef8f319",
 "7e8617ac041793154713e5ef8f319"] with hashAlgorithm = "MD5"
FileMatch "s3://amzn-s3-demo-bucket /internal-folder/" in
 ["3ee0d8617ac041793154713e5ef8f319", "7e8617ac041793154713e5ef8f319"]
```

異なるフォルダ内のファイルを比較する

```
FileMatch "s3://original_bucket/" "s3://archive_bucket/"
FileMatch "s3://original_bucket/internal-folder/" "s3://original_bucket/other-folder/"
```

FileMatch は、original_bucket 内のファイルの内容をチェックし、archive_bucket の内容と一致することを確認します。内容が完全に一致しない場合、このルールは失敗します。また、内部フォルダまたは個々のファイルの内容を確認することもできます。

FileMatch は、個々のファイルを相互にチェックすることもできます。

```
FileMatch "s3://amzn-s3-demo-bucket /file_old.json" "s3://amzn-s3-demo-bucket /
file_new.json"
```

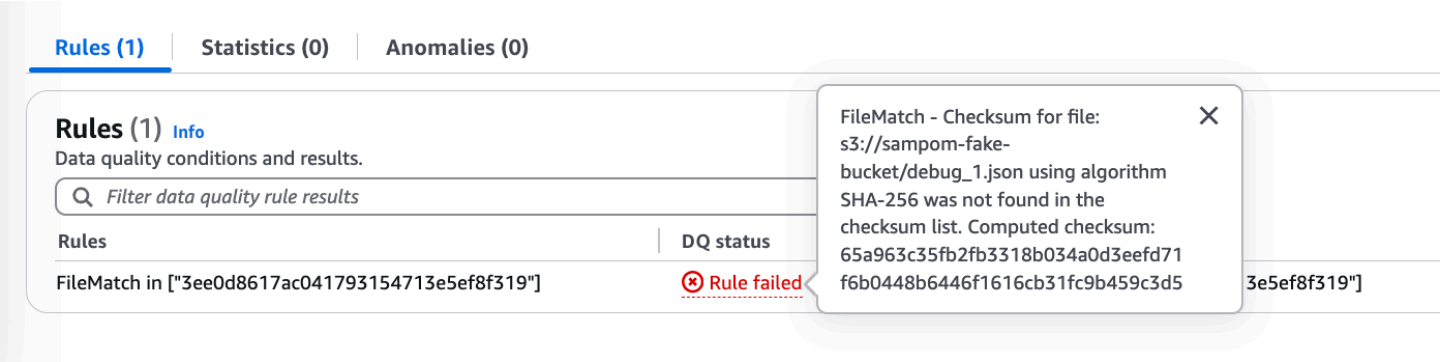
データフレームから直接ファイル名を推測する

必ずしもファイルパスを指定する必要はありません。例えば、AWS Glue Data Catalog (Amazon S3 がバックアップした) でルールを作成する場合、カタログテーブルが使用しているフォルダを見つけるのが難しい場合があります。 AWSGlue Data Quality は、データフレームへの入力に使用される特定のフォルダまたはファイルを見つけることができます。

> **Note**
>
> この機能は、ファイルが DynamicFrame または DataFrame に正常に読み取られた場合にのみ機能します。

```
FileMatch in ["3ee0d8617ac041793154713e5ef8f319"] with hashAlgorithm = "MD5"
FileMatch in ["3ee0d8617ac041793154713e5ef8f319"] with hashAlgorithm = "SHA-1"
FileMatch in ["3ee0d8617ac041793154713e5ef8f319"] with hashAlgorithm = "SHA-256"
FileMatch in ["3ee0d8617ac041793154713e5ef8f319"]
```

指定されたチェックサムが計算されたものと異なる場合、FileMatch はその違いを警告します。



オプションのファイルベースのルールタグ:

タグを使用すると、ルールの動作を制御できます。

recentFiles

このタグは、最新のファイルを最初に保持することで、処理されるファイルの数を制限します。

```
FileMatch "s3://amzn-s3-demo-bucket/file.json" in ["3ee0d8617ac04179sam4713e5ef8f319"]
 with recentFiles = 1
```

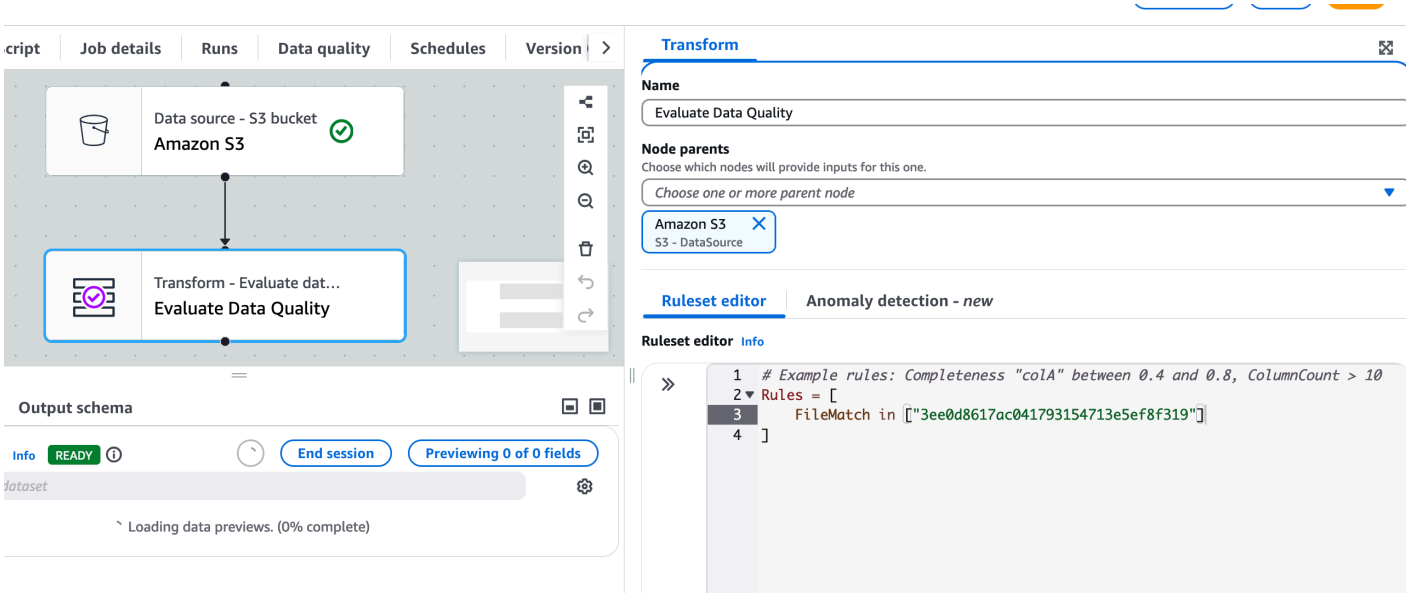
matchFileName

このタグは、ファイルの名前が重複しないようにするものです。デフォルトの動作は false です。

```
FileMatch "s3://amzn-s3-demo-bucket/file.json" in ["3ee0d8617ac04179sam4713e5ef8f319"]
 with matchFileName = "true"
```

考慮事項がいくつかあります:

1. AWS Glue ETL では、Amazon S3 または AWS Glue Data Catalog 変換の直後に [EvaluateDataQuality] 変換が必要です。



2. このルールは AWS Glue インタラクティブセッションでは機能しません。

FileUniqueness

ファイルの一意性によって、データプロデューサーから受け取ったデータに重複したファイルがないことを確認できます。

次のデータ統計を収集します。

1. ルールでスキャンされたファイル数
2. ファイルの一意性の比率

```
Dataset.*.FileUniquenessRatio: 1.00, Dataset.*.FileCount: 8.00
```

フォルダ内の重複ファイルを検索します:

```
FileUniqueness "s3://bucket/" > 0.5
```

```
FileUniqueness "s3://bucket/folder/" = 1
```

重複を検出するために、データフレームから直接フォルダ名を推測します:

必ずしもファイルパスを指定する必要はありません。例えば、AWS Glue Data Catalog でルールを作成する場合、カタログテーブルが使用しているフォルダを見つけるのが難しい場合があります。AWSGlue Data Quality は、データフレームへの入力に使用される特定のフォルダまたはファイルを見つけることができます。

> **Note**
>
> 推論を使用する場合、ファイルベースのルールは DynamicFrame または DataFrame に正常に読み取られたファイルのみを検出できます。

```
FileUniqueness > 0.5
```

オプションのファイルベースのルールタグ:

タグを使用すると、ルールの動作を制御できます。

recentFiles

このタグは、最新のファイルを最初に保持することで、処理されるファイルの数を制限します。

```
FileUniqueness "s3://amzn-s3-demo-bucket/" > 0.5 with recentFiles = 1
```

matchFileName

このタグは、ファイルの名前が重複しないようにするものです。デフォルトの動作は false です。

```
FileUniqueness "s3://amzn-s3-demo-bucket/" > 0.5 with matchFileName = "true"
```

考慮事項がいくつかあります:

1. AWS Glue ETL では、Amazon S3 または AWS Glue Data Catalog 変換の直後に [EvaluateDataQuality] 変換が必要です。
2. このルールは AWS Glue インタラクティブセッションでは機能しません。

FileSize

FileSize ルールタイプを使用すると、ファイルが特定のファイルサイズ基準を満たすようにすることができます。これは、次のユースケースに役立ちます:

1. プロデューサーが空のファイルや実質的に小さいファイルを処理用に送信していないことを確認します。
2. ターゲットのバケットに、パフォーマンスの問題につながる可能性のある小さなファイルがないことを確認します。

FileSize は、次のメトリクスを収集します:

1. コンプライアンス: 制定したルールのしきい値を満たすファイルの割合を返します。
2. ファイル数: ルールによってスキャンされたファイルの数
3. バイト単位の最小ファイルサイズ
4. バイト単位の最大データサイズ

```
Dataset.*.FileSize.Compliance: 1.00,
Dataset.*.FileCount: 8.00,
Dataset.*.MaximumFileSize: 327413121.00,
Dataset.*.MinimumFileSize: 204558920.00
```

これらのメトリクスでは異常検出はサポートされていません。

ファイルのサイズを検証する

file.dat が 2 MB を超えると、このルールは適用されます。

```
FileSize "s3://amzn-s3-demo-bucket/file.dat" > 2 MB
```

サポートされている単位には、B (バイト)、MB (メガバイト)、GB (ギガバイト)、TB (テラバイト) があります。

フォルダ内のファイルのサイズを検証する

```

FileSize "s3://bucket/" > 5 B
FileSize "s3://bucket/" < 2 GB
```

s3://amzn-s3-demo-bucket にあるファイルの 70% が 2 GB ~ 1 TB の間であれば、このルールにパスします。

```
FileSize "s3://amzn-s3-demo-bucket/" between 2 GB and 1 TB  with threshold > 0.7
```

データフレームから直接ファイル名を推測する

必ずしもファイルパスを指定する必要はありません。例えば、Data Catalog でルールを作成する場合、カタログテーブルが使用しているフォルダを見つけるのが難しい場合があります。 AWSGlue Data Quality は、データフレームへの入力に使用される特定のフォルダまたはファイルを見つけることができます。

> **Note**
>
> この機能は、ファイルが DynamicFrame または DataFrame に正常に読み取られた場合にのみ機能します。

```
FileSize < 10 MB with threshold > 0.7
```

オプションのファイルベースのルールタグ:

タグを使用すると、ルールの動作を制御できます。

recentFiles

このタグは、最新のファイルを最初に保持することで、処理されるファイルの数を制限します。

```
FileSize "s3://amzn-s3-demo-bucket/" > 5 B with recentFiles = 1
```

matchFileName

このタグは、ファイルの名前が重複しないようにするものです。デフォルトの動作は false です。

```
FileSize "s3://amzn-s3-demo-bucket/" > 5 B with matchFileName = "true"
```

考慮事項がいくつかあります:

1. AWS Glue ETL では、Amazon S3 または Data Catalog 変換の直後に Evaluate DataQuality Transform を実行する必要があります。

2. このルールは AWS Glue インタラクティブセッションでは機能しません。

API を使用したデータ品質の測定および管理

このトピックでは、API を使用してデータ品質を測定および管理する方法について説明します。

目次

- 前提条件
- AWS Glue Data Quality 推奨事項の操作
- AWS Glue Data Quality ルールセットの操作
- AWS Glue Data Quality 実行の操作
- AWS Glue Data Quality 評価結果の操作

前提条件

- お使いの boto3 のバージョンが最新で、最新の AWS Glue Data Quality API が含まれていることを確認します。
- お使いの AWS CLI のバージョンが最新で、最新の CLI が含まれていることを確認します。

AWS Glue ジョブを使用してこれらの API を実行している場合は、次の方法で boto3 ライブラリを最新バージョンに更新できます。

```
–additional-python-modules boto3==<version>
```

AWS Glue Data Quality 推奨事項の操作

AWS Glue Data Quality 推奨事項の実行を開始するには

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""
    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 AWS Glue client.
        """
        self.glue_client = glue_client
```

```
   def start_data_quality_rule_recommendation_run(self, database_name, table_name,
role_arn):
       """
       Starts a recommendation run that is used to generate rules when you don't know
what rules to write. AWS Glue Data Quality
       analyzes the data and comes up with recommendations for a potential ruleset.
You can then triage the ruleset
       and modify the generated ruleset to your liking.

       :param database_name: The name of the AWS Glue database which contains the
dataset.
       :param table_name: The name of the AWS Glue table against which we want a
recommendation
       :param role_arn: The Amazon Resource Name (ARN) of an AWS Identity and Access
Management (IAM) role that grants permission to let AWS Glue access the resources it
needs.

       """
       try:
           response = self.client.start_data_quality_rule_recommendation_run(
               DataSource={
                   'GlueTable': {
                       'DatabaseName': database_name,
                       'TableName': table_name
                   }
               },
               Role=role_arn
           )
       except ClientError as err:
           logger.error(
               "Couldn't start data quality recommendation run %s. Here's why: %s:
%s", name,
               err.response['Error']['Code'], err.response['Error']['Message'])
           raise
       else:
           return response['RunId']
```

推奨事項の実行では、pushDownPredicates または catalogPartitionPredicates を使用することで、パフォーマンスを高め、カタログソースの特定のパーティションのみに推奨事項を実行できます。

```
client.start_data_quality_rule_recommendation_run(
           DataSource={
```

```
                'GlueTable': {
                    'DatabaseName': database_name,
                    'TableName': table_name,
                    'AdditionalOptions': {
                        'pushDownPredicate': "year=2022"
                    }
                }
            },
            Role=role_arn,
            NumberOfWorkers=2,
            CreatedRulesetName='<rule_set_name>'
  )
```

AWS Glue Data Quality 推奨事項実行の結果を取得するには

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""
    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 AWS Glue client.
        """
        self.glue_client = glue_client

    def get_data_quality_rule_recommendation_run(self, run_id):
        """
        Gets the specified recommendation run that was used to generate rules.

        :param run_id: The id of the data quality recommendation run

        """
        try:
            response =
 self.client.get_data_quality_rule_recommendation_run(RunId=run_id)
        except ClientError as err:
            logger.error(
                "Couldn't get data quality recommendation run %. Here's why: %s: %s",
 run_id,
                err.response['Error']['Code'], err.response['Error']['Message'])
            raise
        else:
            return response
```

上記の反応により得られたオブジェクトから、実行時に推奨された RuleSet を抽出し、以降のステップで使用できます。

```
print(response['RecommendedRuleset'])

Rules = [
    RowCount between 2000 and 8000,
    IsComplete "col1",
    IsComplete "col2",
    StandardDeviation "col3" between 58138330.8 and 64258155.09,
    ColumnValues "col4" between 1000042965 and 1214474826,
    IsComplete "col5"
]
```

推奨事項の実行 (フィルタリングと一覧表示が可能) の一覧を表示するには

```
response = client.list_data_quality_rule_recommendation_runs(
    Filter={
        'DataSource': {
            'GlueTable': {
                'DatabaseName': '<database_name>',
                'TableName': '<table_name>'
            }
        }
)
```

既存の AWS Glue Data Quality の推奨事項タスクをキャンセルするには

```
response = client.cancel_data_quality_rule_recommendation_run(
    RunId='dqrun-d4b6b01957fdd79e59866365bf9cb0e40fxxxxxxx'
)
```

AWS Glue Data Quality ルールセットの操作

AWS Glue Data Quality ルールセットを作成するには

```
response = client.create_data_quality_ruleset(
    Name='<ruleset_name>',
    Ruleset='Rules = [IsComplete "col1", IsPrimaryKey "col2", RowCount between 2000 and
 8000]',
    TargetTable={
```

```
        'TableName': '<table_name>',
        'DatabaseName': '<database_name>'
    }
)
```

データ品質ルールセットを取得するには

```
response = client.get_data_quality_ruleset(
    Name='<ruleset_name>'
)
print(response)
```

この API を使用すると、ルールセットを抽出できます。

```
print(response['Ruleset'])
```

テーブルのデータ品質ルールセットをすべて一覧表示するには

```
response = client.list_data_quality_rulesets()
```

API 内のフィルター条件を使用すると、特定のデータベースまたはテーブルに添付されたすべてのルールセットをフィルタリングできます。

```
response = client.list_data_quality_rulesets(
    Filter={
        'TargetTable': {
            'TableName': '<table_name>',
            'DatabaseName': '<database_name>'
        }
    },
)
```

データ品質ルールセットを更新するには

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""
    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 AWS Glue client.
        """
```

```
        self.glue_client = glue_client

    def update_data_quality_ruleset(self, ruleset_name, ruleset_string):
        """
        Update an AWS Glue Data Quality Ruleset

        :param ruleset_name: The name of the AWS Glue Data Quality ruleset to update
        :param ruleset_string: The DQDL ruleset string to update the ruleset with

        """
        try:
            response = self.client.update_data_quality_ruleset(
                Name=ruleset_name,
                Ruleset=ruleset_string
            )
        except ClientError as err:
            logger.error(
                "Couldn't update the AWS Glue Data Quality ruleset. Here's why: %s:
 %s",
                err.response['Error']['Code'], err.response['Error']['Message'])
            raise
        else:
            return response
```

データ品質ルールセットを削除するには

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""
    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 AWS Glue client.
        """
        self.glue_client = glue_client

    def delete_data_quality_ruleset(self, ruleset_name):
        """
        Delete a AWS Glue Data Quality Ruleset

        :param ruleset_name: The name of the AWS Glue Data Quality ruleset to delete

        """
        try:
            response = self.client.delete_data_quality_ruleset(
```

```
                Name=ruleset_name
            )
        except ClientError as err:
            logger.error(
                "Couldn't delete the AWS Glue Data Quality ruleset. Here's why: %s:
%s",
                err.response['Error']['Code'], err.response['Error']['Message'])
            raise
        else:
            return response
```

AWS Glue Data Quality 実行の操作

AWS Glue Data Quality 実行を開始するには

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""
    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 AWS Glue client.
        """
        self.glue_client = glue_client

    def start_data_quality_ruleset_evaluation_run(self, database_name, table_name,
 role_name, ruleset_list):
        """
        Start an AWS Glue Data Quality evaluation run

        :param database_name: The name of the AWS Glue database which contains the
 dataset.
        :param table_name: The name of the AWS Glue table against which we want to
 evaluate.
        :param role_arn: The Amazon Resource Name (ARN) of an AWS Identity and Access
 Management (IAM) role that grants permission to let AWS Glue access the resources it
 needs.
        :param ruleset_list: The list of AWS Glue Data Quality ruleset names to
 evaluate.

        """
        try:
            response = client.start_data_quality_ruleset_evaluation_run(
                DataSource={
                    'GlueTable': {
```

```
                        'DatabaseName': database_name,
                        'TableName': table_name
                    }
                },
                Role=role_name,
                RulesetNames=ruleset_list
            )
        except ClientError as err:
            logger.error(
                "Couldn't start the AWS Glue Data Quality Run. Here's why: %s: %s",
                err.response['Error']['Code'], err.response['Error']['Message'])
            raise
        else:
            return response['RunId']
```

pushDownPredicate または catalogPartitionPredicate パラメータに合格すると、データ品質の実行の対象を、カタログテーブル内の特定のパーティションセットのみにできます。例:

```
response = client.start_data_quality_ruleset_evaluation_run(
    DataSource={
        'GlueTable': {
            'DatabaseName': '<database_name>',
            'TableName': '<table_name>',
            'AdditionalOptions': {
                'pushDownPredicate': 'year=2023'
            }
        }
    },
    Role='<role_name>',
    NumberOfWorkers=5,
    Timeout=123,
    AdditionalRunOptions={
        'CloudWatchMetricsEnabled': False
    },
    RulesetNames=[
        '<ruleset_name>',
    ]
)
```

ルールセットの複合ルールの評価方法を ROW または COLUMN レベルで設定することもできます。複合ルールの仕組みの詳細については、ドキュメントの「複合ルールの仕組み」を参照してください。

リクエストで複合ルールの評価の仕組みを設定する方法の例:

```
response = client.start_data_quality_ruleset_evaluation_run(
    DataSource={
        'GlueTable': {
            'DatabaseName': '<database_name>',
            'TableName': '<table_name>',
            'AdditionalOptions': {
                'pushDownPredicate': 'year=2023'
            }
        }
    },
    Role='<role_name>',
    NumberOfWorkers=5,
    Timeout=123,
    AdditionalRunOptions={
        'CompositeRuleEvaluationMethod':ROW
    },
    RulesetNames=[
        '<ruleset_name>',
    ]
)
```

AWS Glue Data Quality の実行に関する情報を取得するには

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""
    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 AWS Glue client.
        """
        self.glue_client = glue_client

    def get_data_quality_ruleset_evaluation_run(self, run_id):
        """
        Get details about an AWS Glue Data Quality Run

        :param run_id: The AWS Glue Data Quality run ID to look up

        """
        try:
            response = self.client.get_data_quality_ruleset_evaluation_run(
                RunId=run_id
```

```
            )
        except ClientError as err:
            logger.error(
                "Couldn't look up the AWS Glue Data Quality run ID. Here's why: %s:
 %s",
                err.response['Error']['Code'], err.response['Error']['Message'])
            raise
        else:
            return response
```

AWS Glue Data Quality 実行の結果を取得するには

所定の AWS Glue Data Quality 実行では、次の方法を使用して実行の評価の結果を抽出できます。

```
response = client.get_data_quality_ruleset_evaluation_run(
    RunId='d4b6b01957fdd79e59866365bf9cb0e40fxxxxxxx'
)

resultID = response['ResultIds'][0]

response = client.get_data_quality_result(
    ResultId=resultID
)

print(response['RuleResults'])
```

AWS Glue Data Quality 実行のすべてを一覧表示するには

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""
    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 AWS Glue client.
        """
        self.glue_client = glue_client

    def list_data_quality_ruleset_evaluation_runs(self, database_name, table_name):
        """
        Lists all the AWS Glue Data Quality runs against a given table

        :param database_name: The name of the database where the data quality runs
        :param table_name: The name of the table against which the data quality runs
 were created
```

```
        """
        try:
            response = self.client.list_data_quality_ruleset_evaluation_runs(
                Filter={
                    'DataSource': {
                        'GlueTable': {
                            'DatabaseName': database_name,
                            'TableName': table_name
                        }
                    }
                }
            )
        except ClientError as err:
            logger.error(
                "Couldn't list the AWS Glue Quality runs. Here's why: %s: %s",
                err.response['Error']['Code'], err.response['Error']['Message'])
            raise
        else:
            return response
```

フィルター句を変更すると、特定の時間内の結果のみ、あるいは特定のテーブルに対する実行のみを表示できます。

進行中の AWS Glue Data Quality 実行を停止するには

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""
    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 AWS Glue client.
        """
        self.glue_client = glue_client

    def cancel_data_quality_ruleset_evaluation_run(self, result_id):
        """
        Cancels a given AWS Glue Data Quality run

        :param result_id: The result id of a AWS Glue Data Quality run to cancel

        """
        try:
            response = self.client.cancel_data_quality_ruleset_evaluation_run(
```

```
                ResultId=result_id
            )
        except ClientError as err:
            logger.error(
                "Couldn't cancel the AWS Glue Data Quality run. Here's why: %s: %s",
                err.response['Error']['Code'], err.response['Error']['Message'])
            raise
        else:
            return response
```

AWS Glue Data Quality 評価結果の操作

AWS Glue Data Quality 実行の結果を取得するには

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""
    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 AWS Glue client.
        """
        self.glue_client = glue_client

    def get_data_quality_result(self, result_id):
        """
        Outputs the result of an AWS Glue Data Quality Result

        :param result_id: The result id of an AWS Glue Data Quality run

        """
        try:
            response = self.client.get_data_quality_result(
                ResultId=result_id
            )
        except ClientError as err:
            logger.error(
                "Couldn't get the AWS Glue Data Quality result. Here's why: %s: %s",
                err.response['Error']['Code'], err.response['Error']['Message'])
            raise
        else:
            return response
```

指定のデータ品質結果について収集された統計を表示するには:

```
import boto3
from botocore.exceptions import ClientError
import logging

logger = logging.getLogger(__name__)
class GlueWrapper:
    """Encapsulates AWS Glue actions."""
    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 AWS Glue client.
        """
        self.glue_client = glue_client

    def get_profile_for_data_quality_result(self, result_id):
        """
        Outputs the statistic profile for a AWS Glue Data Quality Result

        :param result_id: The result id of a AWS Glue Data Quality run

        """
        try:
            response = self.glue_client.get_data_quality_result(
                ResultId=result_id
            )

            # the profile contains all statistics gathered for the result
            profile_id = response['ProfileId']
            profile = self.glue_client.list_data_quality_statistics(
                ProfileId = profile_id
            )
            return profile
        except ClientError as err:
            logger.error(
                "Couldn't retrieve Data Quality profile. Here's why: %s: %s",
                err.response['Error']['Code'], err.response['Error']['Message'])
            raise
```

複数のデータ品質実行で収集された統計の時系列を表示するには:

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""
    def __init__(self, glue_client):
        """
```

```
        :param glue_client: A Boto3 AWS Glue client.
        """
        self.glue_client = glue_client

    def get_statistics_for_data_quality_result(self, profile_id):
        """
        Outputs an array of datapoints for each statistic in the input result.

        :param result_id: The profile id of a AWS Glue Data Quality run

        """
        try:
            profile = self.glue_client.list_data_quality_statistics(
                ProfileId = profile_id
            )
            statistics = [self.glue_client.list_data_quality_statistics(
                StatisticId = s['StatisticId']
            ) for s in profile['Statistics']]
            return statistics
        except ClientError as err:
            logger.error(
                "Couldn't retrieve Data Quality statistics. Here's why: %s: %s",
                err.response['Error']['Code'], err.response['Error']['Message'])
            raise
```

特定の統計の異常検出モデルを表示するには:

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""
    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 AWS Glue client.
        """
        self.glue_client = glue_client

    def get_model_training_result_for_statistic(self, statistic_id, profile_id):
        """
        Outputs the details (bounds) of anomaly detection training for the given
 statistic at the given profile.

        :param statistic_id the model's statistic (the timeseries it is tracking)
        :param profile_id the profile associated with the model (a point in the
 timeseries)
```

```
        """
        try:
            model = self.glue_client.get_data_quality_model_result(
                ProfileId = profile_id, StatisticId = statistic_id
            )
            return model
        except ClientError as err:
            logger.error(
                "Couldn't retrieve Data Quality model results. Here's why: %s: %s",
                err.response['Error']['Code'], err.response['Error']['Message'])
            raise
```

統計モデルの異常検出ベースラインからデータポイントを除外するには:

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""
    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 AWS Glue client.
        """
        self.glue_client = glue_client

    def apply_exclusions_to_statistic(self, statistic_id, profile_ids):
        """
        Annotate some points along a given statistic timeseries.

        This example excludes the provided values; INCLUDE can also be used to undo
 this action.

        :param statistic_id the statistic timeseries to annotate
        :param profile_id the profiles we want to exclude (points in the timeseries)

        """

        try:
            response = self.glue_client.batch_put_data_quality_statistic_annotation(
                    InclusionAnnotations = [
                        {'ProfileId': prof_id,
                        'StatisticId': statistic_id,
                        'InclusionAnnotation': 'EXCLUDE'} for prof_id in profile_ids
                    ]
            )
```

```
            return response['FailedInclusionAnnotations']
        except ClientError as err:
            logger.error(
                "Couldn't store Data Quality annotations. Here's why: %s: %s",
                err.response['Error']['Code'], err.response['Error']['Message'])
            raise
```

特定の統計の異常検出モデルトレーニングのステータスを表示するには:

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""
    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 AWS Glue client.
        """
        self.glue_client = glue_client

    def get_model_training_status_for_statistic(self, statistic_id, profile_id):
        """
        Outputs the status of anomaly detection training for the given statistic at the
 given profile.

        :param statistic_id the model's statistic (the timeseries it is tracking)
        :param profile_id the profile associated with the model (a point in the
 timeseries)

        """
        try:
            model = self.glue_client.get_data_quality_model(
                ProfileId = profile_id, StatisticId = statistic_id
            )
            return model
        except ClientError as err:
            logger.error(
                "Couldn't retrieve Data Quality statistics. Here's why: %s: %s",
                err.response['Error']['Code'], err.response['Error']['Message'])
            raise
```

異常検出ベースラインから特定のデータ品質実行からすべての結果を除外するには:

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""
    def __init__(self, glue_client):
```

```
        """
        :param glue_client: A Boto3 AWS Glue client.
        """
        self.glue_client = glue_client

    def apply_exclusions_to_profile(self, profile_id):
        """
        Exclude datapoints produced by a run across statistic timeseries.

        This example excludes the provided values; INCLUDE can also be used to undo
 this action.

        :param profile_id the profiles we want to exclude (points in the timeseries)

        """
        try:
            response = self.glue_client.put_data_quality_profile_annotation(
                    ProfileId = profile_id,
                    InclusionAnnotation = "EXCLUDE"
            )
            return response
        except ClientError as err:
            logger.error(
                "Couldn't store Data Quality annotations. Here's why: %s: %s",
                err.response['Error']['Code'], err.response['Error']['Message'])
            raise
```

指定のデータ品質実行の結果を取得し、結果を表示するには:

AWS Glue Data Quality runID では、次のように resultID を抽出して実際の結果を取得することができます。

```
response = client.get_data_quality_ruleset_evaluation_run(
    RunId='dqrun-abca77ee126abe1378c1da1ae0750d7dxxxx'
)

resultID = response['ResultIds'][0]

response = client.get_data_quality_result(
    ResultId=resultID
)

print(resp['RuleResults'])
```

アラート、デプロイ、スケジュールの設定

このトピックでは、AWS Glue Data Quality の、アラート、デプロイ、スケジュールの設定方法について説明します。

目次

- Amazon EventBridge 統合でのアラートと通知の設定
  - イベントパターンの追加設定のオプション
  - 通知をメールとしてフォーマットする
- CloudWatch 統合でアラートと通知をセットアップする
- データ品質評価の結果をクエリしてダッシュボードを作成する
- AWS CloudFormation を使用したデータ品質ルールのデプロイ
- データ品質ルールのスケジューリング

Amazon EventBridge 統合でのアラートと通知の設定

AWS Glue Data Quality は EventBridge イベントのパブリッシュに対応しています。このイベントは、Data Quality ルールセットの評価実行が完了すると生成されます。これにより、Data Quality のルールが失敗した場合のアラートを、簡単に設定できます。

以下は、データカタログの Data Quality ルールセットを評価する際の、イベントのサンプルです。この情報を使用すれば、Amazon EventBridge で公開されているデータを確認できます。追加の API 呼び出しを発行すると、さらに詳細を入手できます。例えば、特定の実行の詳細を取得するには、結果 ID を使用して `get_data_quality_result` API を呼び出します。

```
{
    "version":"0",
    "id":"abcdef00-1234-5678-9abc-def012345678",
    "detail-type":"Data Quality Evaluation Results Available",
    "source":"aws.glue-dataquality",
    "account":"123456789012",
    "time":"2017-09-07T18:57:21Z",
    "region":"us-west-2",
    "resources":[],
    "detail":{
        "context": {
                    "contextType": "GLUE_DATA_CATALOG",
```

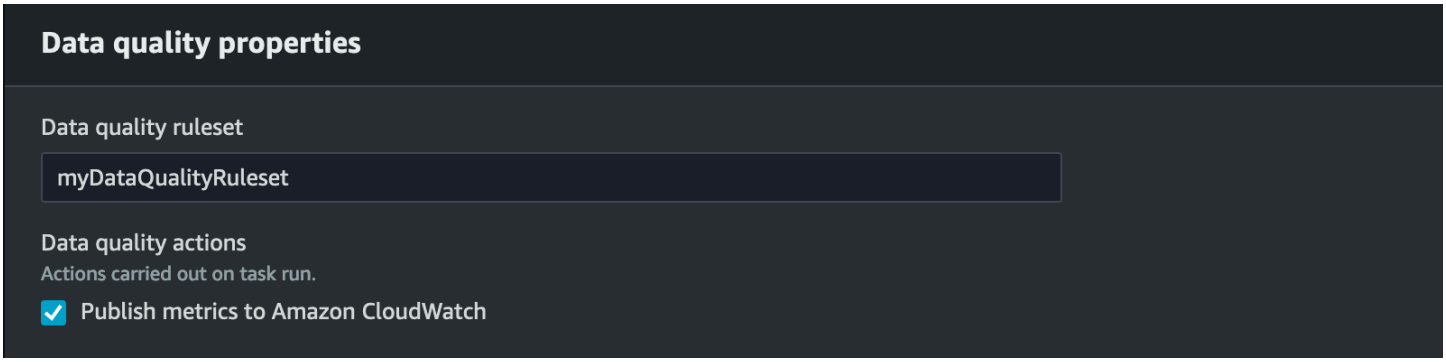
```
                    "runId":"dqrun-12334567890",
                    "databaseName": "db-123",
                    "tableName": "table-123",
                    "catalogId": "123456789012"
                    },
        "resultID": "dqresult-12334567890",
        "rulesetNames": ["rulset1"],
        "state":"SUCCEEDED",
        "score": 1.00,
        "rulesSucceeded": 100,
        "rulesFailed": 0,
        "rulesSkipped": 0
    }
}
```

こちらは、AWS Glue ETL または AWS Glue Studio ノートブックで Data Quality ルールセットを評価するときにパブリッシュする、サンプルのイベントです。

```
{
    "version":"0",
    "id":"abcdef00-1234-5678-9abc-def012345678",
    "detail-type":"Data Quality Evaluation Results Available",
    "source":"aws.glue-dataquality",
    "account":"123456789012",
    "time":"2017-09-07T18:57:21Z",
    "region":"us-west-2",
    "resources":[],
    "detail":{
        "context": {
                    "contextType": "GLUE_JOB",
                    "jobId": "jr-12334567890",
                    "jobName": "dq-eval-job-1234",
                    "evaluationContext": "",
                    }
        "resultID": "dqresult-12334567890",
        "rulesetNames": ["rulset1"],
        "state":"SUCCEEDED",
        "score": 1.00
        "rulesSucceeded": 100,
        "rulesFailed": 0,
        "rulesSkipped": 0
    }
}
```

データカタログと ETL ジョブの両方で Data Quality 評価を実行する場合は、EventBridge のパブリッシュを実行するために、デフォルトで選択されている [Publish metrics to Amazon CloudWatch] を選択したままにしておきます。

EventBridge 通知のセットアップ



生成されたイベントを受信してターゲットを定義するには、Amazon EventBridge ルールを設定する必要があります。ルールを作成するには

1. [Amazon EventBridge console] (Amazon EventBridge コンソール) を開きます。
2. ナビゲーションバーの [バス] セクションで [ルール] を選択します。
3. [Create Rule] (ルールの作成) を選択します。
4. [ルールの詳細を定義] では以下のとおり入力および選択します。
   a. [名前] に `myDQRule` と入力します。
   b. 説明を記入します (任意)。
   c. [イベントバス] では、お使いのイベントバスを選択します。お持ちでない場合は、デフォルトのままにしておきます。
   d. [ルールタイプ] では [イベントパターンを持つルール] を選択し、続いて [次へ] を選択します。
5. [イベントパターンを構築] では以下のとおり入力および選択します。
   a. [イベントソース] では [AWS イベントまたは EventBridge パートナーイベント] を選択します。
   b. [サンプルイベント] セクションは飛ばします。
   c. [作成のメソッド] では [パターンフォームを使用する] を選択します。
   d. イベントパターンの場合
      i. イベントソースに [AWS のサービス] を選択します。
      ii. AWS サービスに [Glue Data Quality] を選択します。
      iii. イベントタイプに [Data Quality Evaluation Results Available] を選択します。

iv. 具体的な状態に [FAILED] を選択します。そうすると、次のようなイベントパターンが表示されます。

```
{
  "source": ["aws.glue-dataquality"],
  "detail-type": ["Data Quality Evaluation Results Available"],
  "detail": {
    "state": ["FAILED"]
  }
}
```

v. 設定オプションの詳細については、「イベントパターンの追加設定のオプション」を参照してください。

6. [ターゲットを選択] では以下のとおり入力および選択します。
   a. [ターゲットタイプ] で [AWS のサービス] を選択します。
   b. [ターゲットを選択] ドロップダウンで、(SNS、Lambda、SQS などへの) 接続に使用する AWS サービスを選択して、[次へ] を選択します。
7. [Configure tag(s)] で、[Add new tags] をクリックし、オプションのタグを追加して、[次へ] を選択します。
8. すべての選択を示す概要ページが表示されます。一番下で [ルールの作成] を選択します。

イベントパターンの追加設定のオプション

イベントを、成功または失敗に応じてフィルタリングするだけでなく、さまざまなパラメータに応じてさらにフィルタリングするとよい場合があります。

これを行うには、[イベントパターン] のセクションに進み、[パターンを編集] を選択して追加のパラメータを指定します。イベントパターンのフィールドは大文字と小文字を区別する必要があります。以下は、イベントパターンの設定の例です。

特定のルールセットを評価する特定のテーブルからイベントを取り込むときは、次のタイプのパターンを使用します。

```
{
  "source": ["aws.glue-dataquality"],
  "detail-type": ["Data Quality Evaluation Results Available"],
  "detail": {
    "context": {
```

```
      "contextType": ["GLUE_DATA_CATALOG"],
      "databaseName": "db-123",
       "tableName": "table-123",
    },
    "rulesetNames": ["ruleset1", "ruleset2"]
    "state": ["FAILED"]
  }
}
```

ETL 体験の特定のジョブからイベントを取り込むときは、次のタイプのパターンを使用します。

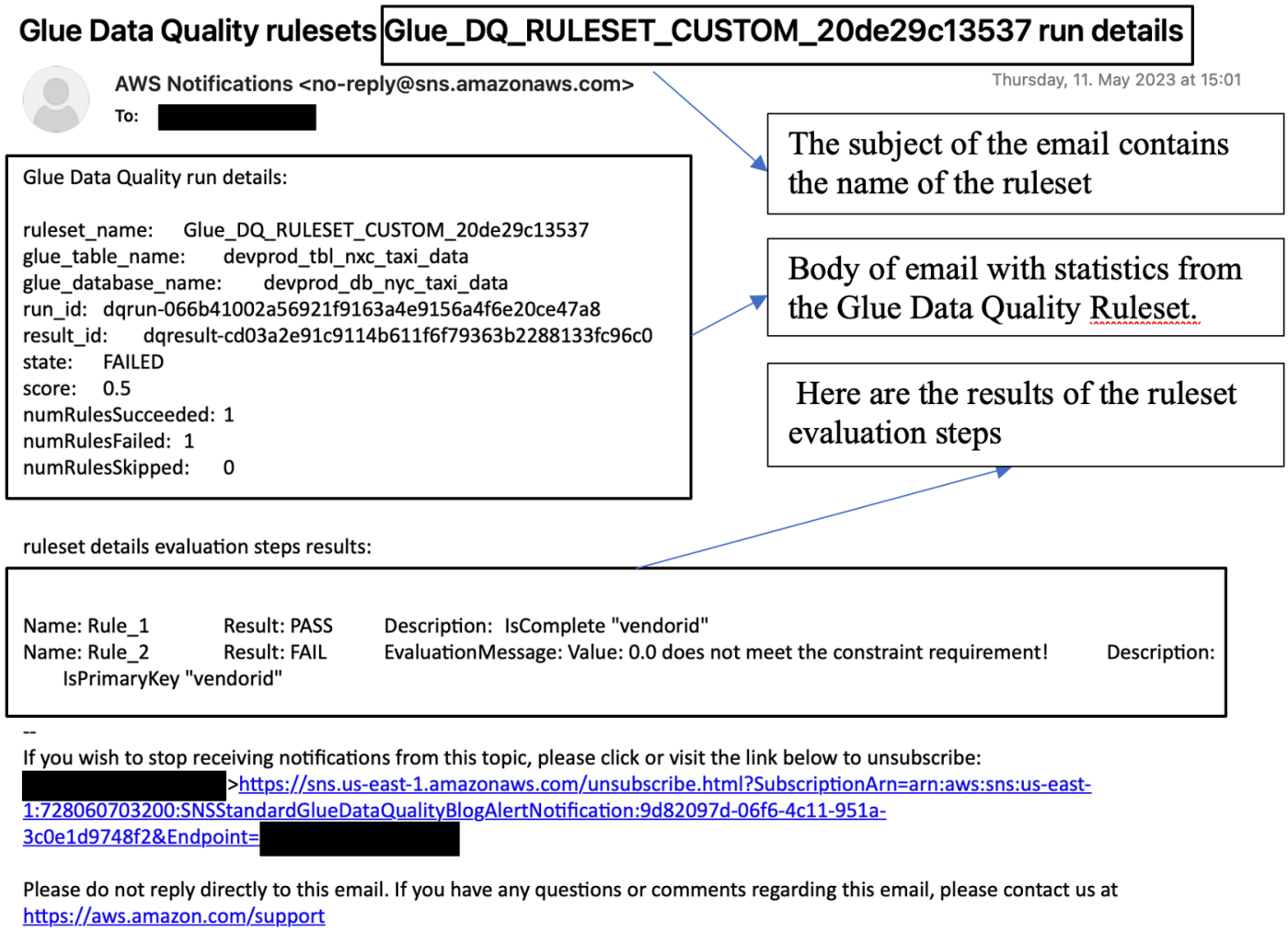
```
{
  "source": ["aws.glue-dataquality"],
  "detail-type": ["Data Quality Evaluation Results Available"],
  "detail": {
    "context": {
      "contextType": ["GLUE_JOB"],
      "jobName": ["dq_evaluation_job1", "dq_evaluation_job2"]
    },
    "state": ["FAILED"]
  }
}
```

スコアが特定のしきい値 (70% など) を下回るイベントを取り込むときは、次のタイプのパターンを使用します。

```
{
  "source": ["aws.glue-dataquality"],
  "detail-type": ["Data Quality Evaluation Results Available"],
  "detail": {
    "score": [{
      "numeric": ["<=", 0.7]
    }]
  }
}
```

通知をメールとしてフォーマットする

自社のビジネスチームに、適切にフォーマットしたメール通知を送信することが必要になる場合があります。Amazon EventBridge と AWS を使用すればこれを実行できます。



次のサンプルコードを使用すると、データ品質通知をフォーマットしてメールを生成できます。

```
import boto3
import json
from datetime import datetime

sns_client = boto3.client('sns')
glue_client = boto3.client('glue')

sns_topic_arn = 'arn:aws:sns:<region-code>:<account-id>:<sns-topic-name>'



def lambda_handler(event, context):
    log_metadata = {}
    message_text = ""
```

```
    subject_text = ""

    if event['detail']['context']['contextType'] == 'GLUE_DATA_CATALOG':
        log_metadata['ruleset_name'] = str(event['detail']['rulesetNames'][0])
        log_metadata['tableName'] = str(event['detail']['context']['tableName'])
        log_metadata['databaseName'] = str(event['detail']['context']['databaseName'])
        log_metadata['runId'] = str(event['detail']['context']['runId'])
        log_metadata['resultId'] = str(event['detail']['resultId'])
        log_metadata['state'] = str(event['detail']['state'])
        log_metadata['score'] = str(event['detail']['score'])
        log_metadata['numRulesSucceeded'] = str(event['detail']['numRulesSucceeded'])
        log_metadata['numRulesFailed'] = str(event['detail']['numRulesFailed'])
        log_metadata['numRulesSkipped'] = str(event['detail']['numRulesSkipped'])

        message_text += "Glue Data Quality run details:\n"
        message_text += "ruleset_name: {}\n".format(log_metadata['ruleset_name'])
        message_text += "glue_table_name: {}\n".format(log_metadata['tableName'])
        message_text += "glue_database_name: {}\n".format(log_metadata['databaseName'])
        message_text += "run_id: {}\n".format(log_metadata['runId'])
        message_text += "result_id: {}\n".format(log_metadata['resultId'])
        message_text += "state: {}\n".format(log_metadata['state'])
        message_text += "score: {}\n".format(log_metadata['score'])
        message_text += "numRulesSucceeded:
{}\n".format(log_metadata['numRulesSucceeded'])
        message_text += "numRulesFailed: {}\n".format(log_metadata['numRulesFailed'])
        message_text += "numRulesSkipped: {}\n".format(log_metadata['numRulesSkipped'])

        subject_text = "Glue Data Quality ruleset {} run
details".format(log_metadata['ruleset_name'])

    else:
        log_metadata['ruleset_name'] = str(event['detail']['rulesetNames'][0])
        log_metadata['jobName'] = str(event['detail']['context']['jobName'])
        log_metadata['jobId'] = str(event['detail']['context']['jobId'])
        log_metadata['resultId'] = str(event['detail']['resultId'])
        log_metadata['state'] = str(event['detail']['state'])
        log_metadata['score'] = str(event['detail']['score'])

        log_metadata['numRulesSucceeded'] = str(event['detail']['numRulesSucceeded'])
        log_metadata['numRulesFailed'] = str(event['detail']['numRulesFailed'])
        log_metadata['numRulesSkipped'] = str(event['detail']['numRulesSkipped'])

        message_text += "Glue Data Quality run details:\n"
        message_text += "ruleset_name: {}\n".format(log_metadata['ruleset_name'])
```

```
        message_text += "glue_job_name: {}\n".format(log_metadata['jobName'])
        message_text += "job_id: {}\n".format(log_metadata['jobId'])
        message_text += "result_id: {}\n".format(log_metadata['resultId'])
        message_text += "state: {}\n".format(log_metadata['state'])
        message_text += "score: {}\n".format(log_metadata['score'])
        message_text += "numRulesSucceeded:
{}\n".format(log_metadata['numRulesSucceeded'])
        message_text += "numRulesFailed: {}\n".format(log_metadata['numRulesFailed'])
        message_text += "numRulesSkipped: {}\n".format(log_metadata['numRulesSkipped'])

        subject_text = "Glue Data Quality ruleset {} run
details".format(log_metadata['ruleset_name'])

   resultID = str(event['detail']['resultId'])
   response = glue_client.get_data_quality_result(ResultId=resultID)
   RuleResults = response['RuleResults']
   message_text += "\n\nruleset details evaluation steps results:\n\n"
   subresult_info = []

   for dic in RuleResults:
        subresult = "Name: {}\t\tResult: {}\t\tDescription: \t{}".format(dic['Name'],
dic['Result'], dic['Description'])
        if 'EvaluationMessage' in dic:
            subresult += "\t\tEvaluationMessage: {}".format(dic['EvaluationMessage'])
        subresult_info.append({
            'Name': dic['Name'],
            'Result': dic['Result'],
            'Description': dic['Description'],
            'EvaluationMessage': dic.get('EvaluationMessage', '')
        })
        message_text += "\n" + subresult

   log_metadata['resultrun'] = subresult_info



   sns_client.publish(
        TopicArn=sns_topic_arn,
        Message=message_text,
        Subject=subject_text
   )

   return {
        'statusCode': 200,
```

```
        'body': json.dumps('Message published to SNS topic')
    }
```

CloudWatch 統合でアラートと通知をセットアップする

当社では、Amazon EventBridge を使用してデータ品質アラートをセットアップする方法を推奨しています。Amazon EventBridge ならアラートのセットアップが 1 回で済むためです。ただし使い慣れているからという理由で Amazon CloudWatch を選ぶユーザーもいます。このようなユーザーのために当社は Amazon CloudWatch との統合をご用意しています。

各 AWS Glue Data Quality 評価は、データ品質の実行のたびに、`glue.data.quality.rules.passed` (合格したルールの数を示す) と `glue.data.quality.rules.failed` (不合格だったルールの数を示す) から成る 1 組のメトリクスを発行します。所定のデータ品質の実行がしきい値を下回った場合、発行されたこのメトリクスを使用してアラームを作成し、ユーザーに警告できます。Amazon SNS 通知からメールを送信するアラームをセットアップするには、以下の手順に従います。

Amazon SNS 通知からメールを送信するアラームをセットアップするには、以下の手順に従います。

1. Amazon CloudWatch コンソールを開きます。
2. [メトリクス] で、[すべてのメトリクス] を選択します。[カスタム名前空間] に、Glue Data Quality というタイトルの追加の名前空間が表示されています。

   > **Note**
   >
   > AWS Glue Data Quality の実行を始めるときは、[Publish metrics to Amazon CloudWatch] のチェックボックスがオンになっていることを確認します。オンになっていないと、この実行用のメトリクスが Amazon CloudWatch にパブリッシュされません。

   `Glue Data Quality` 名前空間で、テーブルごと、ルールセットごとに発行されるメトリクスを確認できます。このトピックでは `glue.data.quality.rules.failed` ルールを使用し、この値が 1 を超えた場合にアラームを発行する (つまり、不合格と評価されたルールの数が 1 を超えた場合に通知する) ようにします。
3. アラームを作成するには、[アラーム] で、[すべてのアラーム] を選択します。
4. [アラームの作成] を選択します。

5. メトリクスの選択 を選択します。
6. 作成したテーブルに対応する `glue.data.quality.rules.failed` メトリクスを選択し、次に [メトリクスの選択] を選択します。
7. [メトリクスと条件の指定] タブの [メトリクス] セクションで、以下のとおり選択します。
   a. [統計] で、[合計] を選択します。
   b. [期間] は、[1 minute] を選択します。
8. [条件] セクションで、以下のとおり選択および入力します。
   a. [Threshold type] で [静的] を選択します。
   b. [Whenever glue.data.quality.rules.failed is...] では、[以上] を選択します。
   c. [than...] に、しきい値として 1 と入力します。

   上記を選択したことにより、`glue.data.quality.rules.failed` メトリクスで 1 以上の値が発行されると、アラームがトリガーされることになります。ただし、データがない場合は許容範囲として処理されます。
9. [Next] を選択します。
10. [アクションの設定] で、以下のとおり選択および入力します。
    a. [アラーム状態トリガー] セクションで、[アラーム状態] を選択します。
    b. [Send a notification to the following SNS topic] セクションで、[Create a new topic to send a notification via a new SNS topic] を選択します。
    c. [Email endpoints that will receive the notification] に、自身のメールアドレスを入力します。[トピックの作成] をクリックします。
    d. [Next] を選択します。
11. [アラーム名] に `myFirstDQAlarm` と入力し、[次へ] を選択します。
12. すべての選択を示す概要ページが表示されます。一番下で [アラームを作成] を選択します。

作成したアラームは、Amazon CloudWatch アラームダッシュボードで確認できます。

データ品質評価の結果をクエリしてダッシュボードを作成する

データ品質評価の結果を表示する、ダッシュボードの作成が必要になる場合があります。これには、以下の 2 つの方法があります。

以下のコードを使用して、データを Amazon S3 に書き込むように Amazon EventBridge をセットアップする

```
import boto3
import json
from datetime import datetime


s3_client = boto3.client('s3')
glue_client = boto3.client('glue')


s3_bucket = 's3-bucket-name'

def write_logs(log_metadata):
    try:
        filename = datetime.now().strftime("%m%d%Y%H%M%S") + ".json"
        key_opts = {
            'year': datetime.now().year,
            'month': "{:02d}".format(datetime.now().month),
            'day': "{:02d}".format(datetime.now().day),
            'filename': filename
        }
        s3key = "gluedataqualitylogs/year={year}/month={month}/day={day}/
{filename}".format(**key_opts)
        s3_client.put_object(Bucket=s3_bucket, Key=s3key,
 Body=json.dumps(log_metadata))
    except Exception as e:
        print(f'Error writing logs to S3: {e}')


def lambda_handler(event, context):
    log_metadata = {}
    message_text = ""
    subject_text = ""

    if event['detail']['context']['contextType'] == 'GLUE_DATA_CATALOG':
        log_metadata['ruleset_name'] = str(event['detail']['rulesetNames'][0])
        log_metadata['tableName'] = str(event['detail']['context']['tableName'])
        log_metadata['databaseName'] = str(event['detail']['context']['databaseName'])
        log_metadata['runId'] = str(event['detail']['context']['runId'])
        log_metadata['resultId'] = str(event['detail']['resultId'])
        log_metadata['state'] = str(event['detail']['state'])
        log_metadata['score'] = str(event['detail']['score'])
        log_metadata['numRulesSucceeded'] = str(event['detail']['numRulesSucceeded'])
        log_metadata['numRulesFailed'] = str(event['detail']['numRulesFailed'])
```

```
        log_metadata['numRulesSkipped'] = str(event['detail']['numRulesSkipped'])

        message_text += "Glue Data Quality run details:\n"
        message_text += "ruleset_name: {}\n".format(log_metadata['ruleset_name'])
        message_text += "glue_table_name: {}\n".format(log_metadata['tableName'])
        message_text += "glue_database_name: {}\n".format(log_metadata['databaseName'])
        message_text += "run_id: {}\n".format(log_metadata['runId'])
        message_text += "result_id: {}\n".format(log_metadata['resultId'])
        message_text += "state: {}\n".format(log_metadata['state'])
        message_text += "score: {}\n".format(log_metadata['score'])
        message_text += "numRulesSucceeded:
{}\n".format(log_metadata['numRulesSucceeded'])
        message_text += "numRulesFailed: {}\n".format(log_metadata['numRulesFailed'])
        message_text += "numRulesSkipped: {}\n".format(log_metadata['numRulesSkipped'])

        subject_text = "Glue Data Quality ruleset {} run
details".format(log_metadata['ruleset_name'])

    else:
        log_metadata['ruleset_name'] = str(event['detail']['rulesetNames'][0])
        log_metadata['jobName'] = str(event['detail']['context']['jobName'])
        log_metadata['jobId'] = str(event['detail']['context']['jobId'])
        log_metadata['resultId'] = str(event['detail']['resultId'])
        log_metadata['state'] = str(event['detail']['state'])
        log_metadata['score'] = str(event['detail']['score'])

        log_metadata['numRulesSucceeded'] = str(event['detail']['numRulesSucceeded'])
        log_metadata['numRulesFailed'] = str(event['detail']['numRulesFailed'])
        log_metadata['numRulesSkipped'] = str(event['detail']['numRulesSkipped'])

        message_text += "Glue Data Quality run details:\n"
        message_text += "ruleset_name: {}\n".format(log_metadata['ruleset_name'])
        message_text += "glue_job_name: {}\n".format(log_metadata['jobName'])
        message_text += "job_id: {}\n".format(log_metadata['jobId'])
        message_text += "result_id: {}\n".format(log_metadata['resultId'])
        message_text += "state: {}\n".format(log_metadata['state'])
        message_text += "score: {}\n".format(log_metadata['score'])
        message_text += "numRulesSucceeded:
{}\n".format(log_metadata['numRulesSucceeded'])
        message_text += "numRulesFailed: {}\n".format(log_metadata['numRulesFailed'])
        message_text += "numRulesSkipped: {}\n".format(log_metadata['numRulesSkipped'])

        subject_text = "Glue Data Quality ruleset {} run
details".format(log_metadata['ruleset_name'])
```

```
    resultID = str(event['detail']['resultId'])
    response = glue_client.get_data_quality_result(ResultId=resultID)
    RuleResults = response['RuleResults']
    message_text += "\n\nruleset details evaluation steps results:\n\n"
    subresult_info = []

    for dic in RuleResults:
        subresult = "Name: {}\t\tResult: {}\t\tDescription: \t{}".format(dic['Name'],
dic['Result'], dic['Description'])
        if 'EvaluationMessage' in dic:
            subresult += "\t\tEvaluationMessage: {}".format(dic['EvaluationMessage'])
        subresult_info.append({
            'Name': dic['Name'],
            'Result': dic['Result'],
            'Description': dic['Description'],
            'EvaluationMessage': dic.get('EvaluationMessage', '')
        })
        message_text += "\n" + subresult

    log_metadata['resultrun'] = subresult_info

    write_logs(log_metadata)


    return {
        'statusCode': 200,
        'body': json.dumps('Message published to SNS topic')
    }
```

データを Amazon S3 に書き込むと、AWS Glue クローラーを使用して Athena に登録し、テーブルをクエリできます。

データ品質評価中に Amazon S3 のロケーションを設定する

AWS Glue データカタログまたは AWS Glue ETL でデータ品質タスクを実行しているときは、Amazon S3 のロケーションを指定してデータ品質評価の結果を Amazon S3 に書き込むことが可能です。以下の構文を使用すると、ターゲットを参照してテーブルを作成し、データ品質評価の結果を読み取れます。

`CREATE EXTERNAL TABLE` と `MSCK REPAIR TABLE` のクエリは個別に実行する必要がありますので注意してください。

```
CREATE EXTERNAL TABLE <my_table_name>(
    catalogid string,
    databasename string,
    tablename string,
    dqrunid string,
    evaluationstartedon timestamp,
    evaluationcompletedon timestamp,
    rule string,
    outcome string,
    failurereason string,
    evaluatedmetrics string)
PARTITIONED BY (
    `year` string,
    `month` string,
    `day` string)
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
WITH SERDEPROPERTIES (

 'paths'='catalogId,databaseName,dqRunId,evaluatedMetrics,evaluationCompletedOn,evaluationStart
STORED AS INPUTFORMAT 'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION 's3://glue-s3-dq-bucket-us-east-2-results/'
TBLPROPERTIES (
    'classification'='json',
    'compressionType'='none',
    'typeOfData'='file');
```

```
MSCK REPAIR TABLE <my_table_name>;
```

上記のテーブルを作成すれば、Amazon Athena を使用して分析クエリを実行できるようになります。

AWS CloudFormation を使用したデータ品質ルールのデプロイ

データ品質ルールの作成には、AWS CloudFormation を使用します。詳細については、「AWS Glue 用の AWS CloudFormation」を参照してください。

データ品質ルールのスケジューリング

データ品質ルールのスケジューリングには、次の方法を使用します。

- データカタログからデータ品質ルールをスケジュールします。ノーコードのユーザーは、この方法を使用することでデータ品質スキャンを簡単にスケジューリングできます。AWSGlue Data Quality では、Amazon EventBridge にスケジュールを作成します。データ品質ルールをスケジュールするには、以下の手順に従います。
  - ルールセットに進み、[実行] をクリックします。
  - [Run frequency] で希望するスケジュールを選択し、タスク名を入力します。このタスク名は、EventBridge でのスケジュールの名前です。
- Amazon EventBridge と AWS Step Functions を使用して、データ品質ルールの評価と推奨事項をオーケストレートします。

AWS Glue Data Quality の保管中のデータ暗号化

AWS Glue Data Quality は、デフォルトで暗号化を提供し、AWS が所有する暗号化キーを使用して保管中のお客様の機密データを保護します。

AWS 所有キー

AWS Glue Data Quality はこれらのキーを使用して、お客様の Data Quality アセットを自動的に暗号化します。AWS 所有キーは表示、管理、使用することはできず、その使用を監査することもできません。しかし、データを暗号化するキーを保護するために何らかの操作を行ったり、プログラムを変更したりする必要はありません。詳細については、AWS KMS デベロッパーガイドの「AWS 所有キー」を参照してください。

保管中のデータをデフォルトで暗号化することで、機密データの保護におけるオーバーヘッドと複雑な作業を減らすのに役立ちます。同時に、セキュリティを重視したアプリケーションを構築することで、暗号化のコンプライアンスと規制の厳格な要件を満たすことができます。

この暗号化層を無効にしたり、別の暗号化タイプを選択したりすることはできませんが、Data Quality リソースを作成する際にカスタマーマネージドキーを選択することで、既存の AWS 所有暗号化キーに 2 番目の暗号化層を追加できます。

カスタマーマネージドキー

カスタマーマネージドキー: AWS Glue Data Quality は、ユーザーが作成、所有、および管理する対称カスタマーマネージドキーの使用をサポートします。これにより、既存の AWS 所有の暗号化に 2 番目の暗号化層が追加されます。この暗号化層はユーザーが完全に制御できるため、次のようなタスクを実行できます:

- キーポリシーの策定と維持
- IAM ポリシーの策定と維持
- キーポリシーの有効化と無効化
- キー暗号化マテリアルのローテーション
- タグの追加
- キーエイリアスの作成
- 削除のためのキースケジューリング

詳細については、AWS KMS デベロッパーガイドの「カスタマーマネージドキー」を参照してください。

次の表は、AWS Glue Data Quality がさまざまな Data Quality アセットを暗号化する方法をまとめたものです。

データ型	AWS 所有のキー暗号化	カスタマーマネージドキーの暗号化
Data Quality ルールセット 永続 DQ ルールセットに参照される DQDL ルールセット文字列。これらの永続ルールセットは、現時点では AWS Glue Data Catalog エクスペリエンスでのみ使用されます。	有効	有効
Data Quality ルール/アナライザーの結果 ルールセット内の各ルールの合格/不合格ステータスと、ルールとアナライザーの両方で収集されたメトリクスを含む結果アーティファクト。	有効	有効
観測値	有効	有効

データ型	AWS 所有のキー暗号化	カスタマーマネージドキーの暗号化
観測値は、データ内で異常が検出されたときに生成されます。これには、予想される上限と下限に関する情報と、これらの境界に基づく推奨ルールが含まれます。生成されると、Data Quality の結果とともに表示されます。		
統計 メトリクスの値 (RowCount、Completeness など)、列名、その他のメタデータなど、ルールセットに指定されたデータを評価した後に収集されたメトリクスに関する情報が含まれます。	有効	有効
異常検出統計モデル 統計モデルには、顧客データの以前の評価に基づいて生成された特定のメトリクスの上限および下限の時系列が含まれます。	有効	有効

> **Note**
>
> AWS Data Quality は、個人を特定できるデータを無料で保護するために、AWS が所有するキーを使用して保管中の暗号化を自動的に有効にします。ただし、カスタマーマネージドキーの使用には AWS KMS 料金が適用されます。料金の詳細については、「AWS KMS 料金表」を参照してください。
> AWS KMS の詳細については、「AWS KMS」をご参照ください。

カスタマーマネージドキーを作成

対称カスタマーマネージドキーを作成するには、AWS Management Console または AWS KMS API を使用します。

対称カスタマーマネージドキーを作成するには

- 「AWS Key Management Service デベロッパーガイド」の「対称暗号化 AWS KMS キー」を参照してください。

キーポリシー

キーポリシーは、カスタマーマネージドキーへのアクセスを制御します。すべてのカスタマーマネージドキーには、キーポリシーが 1 つだけ必要です。このポリシーには、そのキーを使用できるユーザーとその使用方法を決定するステートメントが含まれています。キーポリシーは、カスタマーマネージドキーの作成時に指定できます。詳細については、「AWS Key Management Service デベロッパーガイド」の「AWS KMS のキーポリシー」を参照してください。

カスタマーマネージドキーを Data Quality リソースで使用するには、キーポリシーで次の API オペレーションを許可する必要があります:

- `kms:Decrypt` - `GenerateDataKeyWithoutPlaintext` を使用して AWS KMS キーによって暗号化された暗号文を復号します。
- `kms:DescribeKey` — Amazon Location がキーを検証できるように、カスタマーマネージドキーの詳細を提供します。
- `kms:GenerateDataKeyWithoutPlaintext` - AWS KMS の外部で使用する一意の対称データキーを返します。このオペレーションは、ユーザー指定の対称暗号化 KMS キーで暗号化されたデータキーのコピーを返します。キーのバイトはランダムであり、呼び出し元や KMS キーとは関係ありません。顧客が行う必要がある KMS の呼び出しを減らすために使用されます。
- `kms:ReEncrypt*` - 暗号文を復号し、AWS KMS 内で完全に再暗号化します。このオペレーションを使用して、KMS キーを手動でローテーションしたり、暗号文を保護する KMS キーを変更したりするなど、データが暗号化される KMS キーを変更できます。これを使用して、暗号文の暗号化コンテキストを変更するなど、同じ KMS キーで暗号文を再暗号化することもできます。

Amazon Location に追加できるポリシーステートメントの例を以下に示します。

```
"Statement" : [
```

```
    {
        "Sid" : "Allow access to principals authorized to use AWS Glue Data Quality",
        "Effect" : "Allow",
        "Principal" : {
            "AWS": "arn:aws:iam::<account_id>:role/ExampleRole"
        },
        "Action" : [
            "kms:Decrypt",
            "kms:DescribeKey",
            "kms:GenerateDataKeyWithoutPlaintext",
            "kms:ReEncrypt*"
        ],
        "Resource" : "*",
        "Condition" : {
            "StringEquals" : {
                "kms:ViaService" : "glue.amazonaws.com",
                "kms:CallerAccount" : "111122223333"
            }
        },
    {
        "Sid": "Allow access for key administrators",
        "Effect": "Allow",
        "Principal": {
            "AWS": "arn:aws:iam::111122223333:root"
          },
        "Action" : [
            "kms:*"
         ],
        "Resource": "arn:aws:kms:region:111122223333:key/key_ID"
    },
    {
        "Sid" : "Allow read-only access to key metadata to the account",
        "Effect" : "Allow",
        "Principal" : {
            "AWS" : "arn:aws:iam::111122223333:root"
        },
        "Action" : [
            "kms:Describe*",
            "kms:Get*",
            "kms:List*",
        ],
        "Resource" : "*"
    }
```

```
]
```

AWS Glue Data Quality での KMS キーの使用に関する注意事項

AWS Glue Data Quality はキー移行をサポートしていません。つまり、Data Quality アセットをキー A で暗号化し、キー B に切り替える場合、キー A で暗号化されたデータはキー B を使用するように再暗号化されません。キー B に切り替えることはできますが、キー A で以前に暗号化されたデータにアクセスするにはキー A へのアクセスを維持する必要があります。

ポリシーでのアクセス許可の指定に関する詳細については、「AWS Key Management Service デベロッパーガイド」の「キーポリシーでの AWS サービスのアクセス許可」を参照してください。

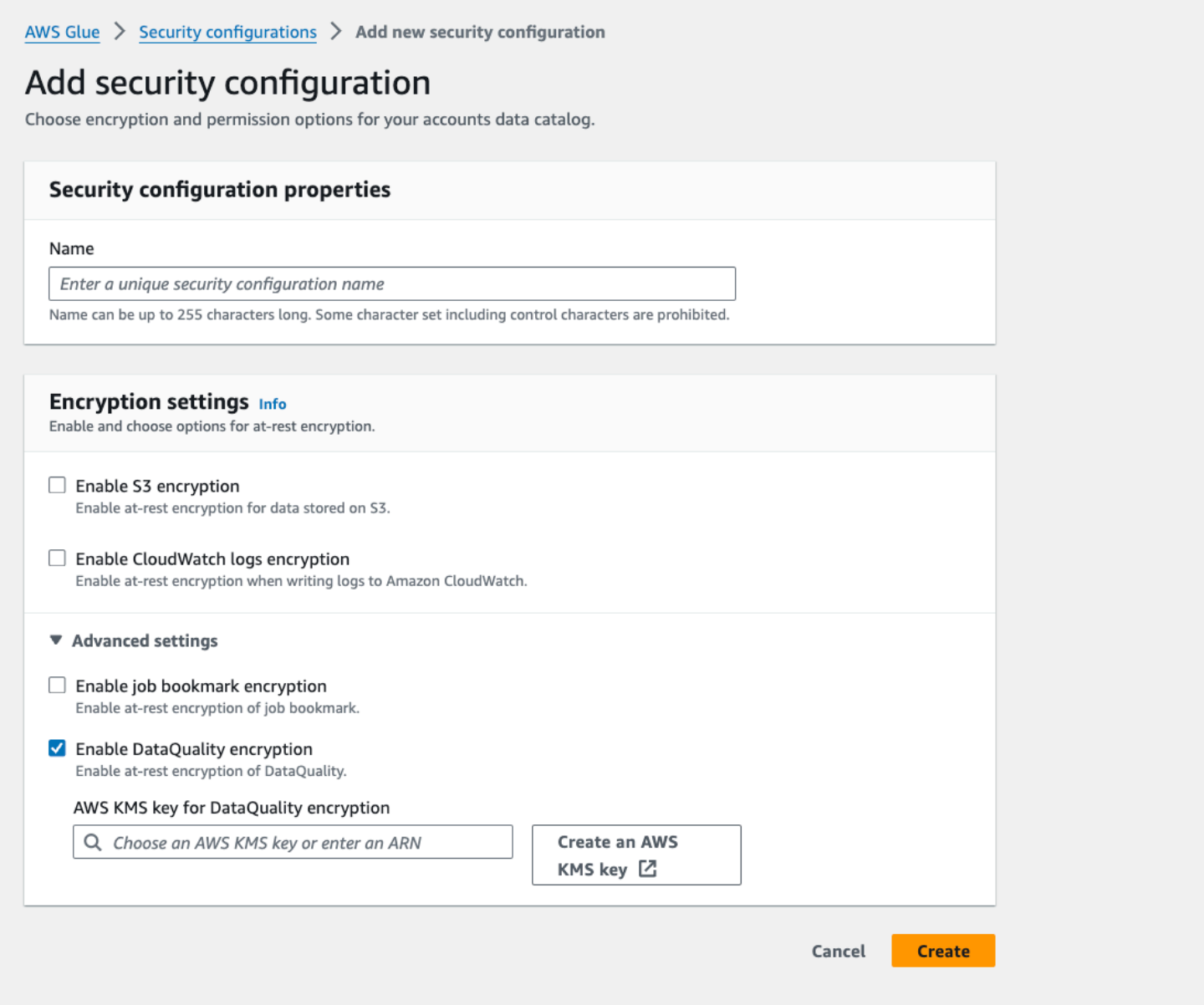
キーアクセスのトラブルシューティングについては、「AWS Key Management Service デベロッパーガイド」の「キーアクセスのトラブルシューティング」を参照してください。

セキュリティ設定を作成する

AWS Glue のセキュリティ設定リソースには、暗号化されたデータを書き込むときに必要なプロパティが含まれています。

データ品質アセットを暗号化するには

1. [詳細設定] 下の [暗号化設定] で、[データ品質暗号化を有効にする] を選択します。
2. [KMS キー]を選択するか、[AWS KMS キーの作成] を選択します。



AWS Glue Data Quality 暗号化コンテキスト

暗号化コンテキストは、データに関する追加のコンテキスト情報が含まれたキーと値のペアのオプションのセットです。

AWS KMS は、暗号化コンテキストを追加の認証済みデータとして使用して、認証済み暗号化をサポートします。データの暗号化リクエストに暗号化コンテキストを組み込むと、AWS KMS は暗号化コンテキストを暗号化後のデータにバインドします。データを復号化するには、そのリクエストに (暗号化時と) 同じ暗号化コンテキストを含めます。

AWS Glue Data Quality 暗号化コンテキストの例

```
"encryptionContext": {
    "kms-arn": "arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE",
    "branch-key-id": "111122223333+arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE",
    "hierarchy-version": "1",
    "aws-crypto-ec:aws:glue:securityConfiguration": "111122223333:customer-security-
configuration-name",
    "create-time": "2024-06-07T13:47:23:000861Z",
    "tablename": "AwsGlueMlEncryptionKeyStore",
    "type": "beacon:ACTIVE"
}
```

モニタリングに暗号化コンテキストを使用する

対称カスタマーマネージドキーを使用してトラッカーまたはジオフェンスコレクションを暗号化する場合は、監査レコードとログで暗号化コンテキストを使用して、カスタマーマネージドキーがどのように使用されているかを特定することもできます。暗号化コンテキストは、AWS CloudTrail または Amazon CloudWatch Logs が生成したログにも表示されます。

AWS Glue Data Quality の暗号化キーのモニタリング

AWS Glue Data Quality リソースで AWS KMS カスタマーマネージドキーを使用する場合、AWS CloudTrail または Amazon CloudWatch Logs を使用して、AWS Glue Data Quality が AWS KMS に送信するリクエストを追跡できます。

次の例は、カスタマーマネージドキーで暗号化されたデータにアクセスするために、AWS Glue Data Quality によって呼び出される KMS オペレーションを監視する `GenerateDataKeyWithoutPlainText` および `Decrypt` の AWS CloudTrail イベントです。

Decrypt

```
{
    "eventVersion": "1.09",
    "userIdentity": {
        "type": "AssumedRole",
        "arn": "arn:aws:sts::111122223333:role/CustomerRole",
        "accountId": "111122223333",
        "invokedBy": "glue.amazonaws.com"
    },
```

```
    "eventTime": "2024-07-02T20:03:10Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "Decrypt",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "glue.amazonaws.com",
    "userAgent": "glue.amazonaws.com",
    "requestParameters": {
        "keyId": "arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE",
        "encryptionAlgorithm": "SYMMETRIC_DEFAULT",
        "encryptionContext": {
            "kms-arn": "arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE",
            "branch-key-id": "111122223333+arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE",
            "hierarchy-version": "1",
            "aws-crypto-ec:aws:glue:securityConfiguration": "111122223333:customer-
security-configuration-name",
            "create-time": "2024-06-07T13:47:23:000861Z",
            "tablename": "AwsGlueMlEncryptionKeyStore",
            "type": "branch:ACTIVE",
            "version": "branch:version:ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE"
        }
    },
    "responseElements": null,
    "requestID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "eventID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```

GenerateDataKeyWithoutPlaintext

```
{
    "eventVersion": "1.09",
    "userIdentity": {
        "type": "AssumedRole",
        "arn": "arn:aws:sts::111122223333:role/CustomerRole",
        "accountId": "111122223333",
        "invokedBy": "glue.amazonaws.com"
    },
    "eventTime": "2024-07-02T20:03:10Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "GenerateDataKeyWithoutPlaintext",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "glue.amazonaws.com",
    "userAgent": "glue.amazonaws.com",
    "requestParameters": {
        "keyId": "arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE",
        "encryptionAlgorithm": "SYMMETRIC_DEFAULT",
        "encryptionContext": {
            "kms-arn": "arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE",
            "branch-key-id": "111122223333+arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE",
            "hierarchy-version": "1",
            "aws-crypto-ec:aws:glue:securityConfiguration": "111122223333:customer-
security-configuration-name",
            "create-time": "2024-06-07T13:47:23:000861Z",
            "tablename": "AwsGlueMlEncryptionKeyStore",
            "type": "branch:version:ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE"
        }
    },
    "responseElements": null,
    "requestID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "eventID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
        }
    ],
```

```
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```

ReEncyrpt

```
{
    "eventVersion": "1.09",
    "userIdentity": {
        "type": "AssumedRole",
        "arn": "arn:aws:sts::111122223333:role/CustomerRole",
        "accountId": "111122223333",
        "invokedBy": "glue.amazonaws.com"
    },
    "eventTime": "2024-07-17T21:34:41Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "ReEncrypt",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "glue.amazonaws.com",
    "userAgent": "glue.amazonaws.com",
    "requestParameters": {
        "destinationEncryptionContext": {
             "kms-arn": "arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE",
            "branch-key-id": "111122223333+arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE",
            "hierarchy-version": "1",
            "aws-crypto-ec:aws:glue:securityConfiguration": "111122223333:customer-
security-configuration-name",
            "create-time": "2024-06-07T13:47:23:000861Z",
            "tablename": "AwsGlueMlEncryptionKeyStore",
            "type": "branch:ACTIVE"
            "version": "branch:version:12345678-SAMPLE"
        },
        "destinationKeyId": "arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE",
        "sourceAAD": "1234567890-SAMPLE+Z+lqoYOHj7VtWxJLrvh+biUFbliYDAQkobM=",
        "sourceKeyId": "arn:aws:kms:ap-southeast-2:585824196334:key/17ca05ca-a8c1-40d7-
b7fd-30abb569a53a",
        "destinationEncryptionAlgorithm": "SYMMETRIC_DEFAULT",
        "sourceEncryptionContext": {
```

```
            "kms-arn": "arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE",
            "branch-key-id": "111122223333+arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE",
            "hierarchy-version": "1",
            "aws-crypto-ec:aws:glue:securityConfiguration": "111122223333:customer-
security-configuration-name",
            "create-time": "2024-06-07T13:47:23:000861Z",
            "tablename": "AwsGlueMlEncryptionKeyStore",
            "type": "branch:version:ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE"
        },
        "destinationAAD": "1234567890-SAMPLE",
        "sourceEncryptionAlgorithm": "SYMMETRIC_DEFAULT"
    },
    "responseElements": null,
    "requestID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "eventID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
        },
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```


詳細はこちら

次のリソースは、保管時のデータ暗号化についての詳細を説明しています。

- AWS Key Management Service 基本概念の詳細については、「AWS Key Management Service デベロッパーガイド」を参照してください。
- AWS Key Management Service のセキュリティのベストプラクティスの詳細については、「AWS Key Management Service デベロッパーガイド」を参照してください。

AWS Glue Data Quality のエラーのトラブルシューティング

AWS Glue Data Quality でエラーが発生した場合は、次の解決方法を用いて問題の原因を突き止め、修正します。

目次

- エラー: AWS Glue Data Quality のモジュールが見つからない
- エラー: AWS Lake Formation のアクセス許可が不十分である
- エラー: ルールセットに一意の名前が付いていない
- エラー: テーブルに特殊文字が含まれている
- エラー: 大規模なルールセットによるオーバーフローエラー
- エラー: ルール全体のステータスが失敗である
- AnalysisException: Unable to verify existence of default database
- エラーメッセージ: Provided key map not suitable for given data frames
- ユーザークラスの例外: java.lang.RuntimeException : Failed to fetch data. Check the logs in CloudWatch to get more details
- LAUNCH ERROR: Error downloading from S3 for bucket
- InvalidInputException (status: 400): DataQuality rules cannot be parsed
- エラー: EventBridge が、設定したスケジュールに基づいて Glue DQ ジョブをトリガーしない
- CustomSQL エラー
- 動的ルール
- ユーザークラスの例外: org.apache.spark.sql.AnalysisException: org.apache.hadoop.hive.ql.metadata.HiveException
- UNCLASSIFIED_ERROR; IllegalArgumentException : 解析エラー: ルールまたはアナライザーが指定されていません。、入力時に実行可能な代替手段がありません。

エラー: AWS Glue Data Quality のモジュールが見つからない

エラーメッセージ: No module named 'awsgluedq'.

解決方法: このエラーは、AWS Glue Data Quality をサポートされていないバージョンで実行すると発生します。AWSGlue Data Quality は、Glue のバージョン 3.0 移行でのみサポートされています。

エラー: AWS Lake Formation のアクセス許可が不十分である

エラーメッセージ: Exception in User Class: `com.amazonaws.services.glue.model.AccessDeniedException`: Insufficient Lake Formation permission(s) on impact_sdg_involvement (Service: AWS Glue; Status Code: 400; Error Code: AccessDeniedException; Request ID: 465ae693-b7ba-4df0-a4e4-6b17xxxxxxxx; Proxy: null).

解決方法: AWS Lake Formation に十分なアクセス許可を付与する必要があります。

エラー: ルールセットに一意の名前が付いていない

エラーメッセージ: Exception in User Class: ...services.glue.model.AlreadyExistsException: Another ruleset with the same name already exists.

解決方法: ルールセットはグローバルであり、一意である必要があります。

エラー: テーブルに特殊文字が含まれている

エラーメッセージ: Exception in User Class: org.apache.spark.sql.AnalysisException: cannot resolve "C" given input columns: [primary.data_end_time, primary.data_start_time, primary.end_time, primary.last_updated, primary.message, primary.process_date, primary.rowhash, primary.run_by, primary.run_id, primary.start_time, primary.status]; line 1 pos 44;.

解決方法: 現在、「.」のような特殊文字を含むテーブルでは AWS Glue Data Quality を実行できない、という制限があります。

エラー: 大規模なルールセットによるオーバーフローエラー

エラーメッセージ: Exception in User Class: java.lang.StackOverflowError.

解決方法: 2,000 以上のルールを含む大規模なルールセットを使用すると、このエラーが発生する可能性があります。ルールを複数のルールセットに分割します。

エラー: ルール全体のステータスが失敗である

エラーの状態: My Ruleset is successful, but my overall rule status is failed.

解決方法: このエラーは、パブリッシュの最中に Amazon CloudWatch にメトリクスをパブリッシュするオプションを選択したことが原因と考えられます。データセットが VPC にある場合、お使いの VPC が、AWS Glue に対して、メトリクスを Amazon CloudWatch にパブリッシュすることを許可していない可能性があります。その場合は、VPC が Amazon CloudWatch にアクセスできるようエンドポイントをセットアップする必要があります。

AnalysisException: Unable to verify existence of default database

エラー状態: AnalysisException: Unable to verify existence of default database: com.amazonaws.services.glue.model.AccessDeniedException: Insufficient Lake Formation permission(s) on default (Service: AWS Glue; Status Code: 400; Error Code: AccessDeniedException; Request ID: XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX; Proxy: null)

解決方法: AWS Glue ジョブのカタログ統合では、AWS Glue は AWS Glue `GetDatabase API` を使用してデフォルトのデータベースが存在するかを常にチェックします。`DESCRIBE` Lake Formation アクセス許可が付与されないとき、または `GetDatabase IAM` アクセス許可が付与されたとき、デフォルトのデータベースの存在を確認するときのジョブは失敗します。

解決するには:

1. Lake Formation にデフォルトのデータベース用の `DESCRIBE` アクセス許可を追加します。
2. AWS Glue ジョブにアタッチされている IAM ロール を Lake Formation でデータベース作成者として設定します。これにより、デフォルトのデータベースが自動的に作成され、必要な Lake Formation へのアクセス許可がロールに付与されます。
3. `--enable-data-catalog` オプションを無効にします (AWS Glue Studio では [Use Data Catalog as the Hive metastore] と表示されます)。

   ジョブに Spark SQL Data Catalog 統合を必要としない場合は、無効にできます。

エラーメッセージ: Provided key map not suitable for given data frames

エラー状態: Provided key map not suitable for given data frames.

解決方法: 現在、DataSetMatch ルールタイプを使用しており、結合キーが重複しています。結合キーは一意でなければならず、また NULL であってはなりません。一意の結合キーを使用できない場合は、AggregateMatch のような他のルール型を使用してサマリーデータで照合することを検討します。

ユーザークラスの例外: java.lang.RuntimeException : Failed to fetch data. Check the logs in CloudWatch to get more details

エラー状態: Exception in User Class: java.lang.RuntimeException : Failed to fetch data. Check the logs in CloudWatch to get more details.

解決方法: このエラーは、Amazon S3 ベースのテーブルで、Amazon RDS または Amazon Redshift と比較する DQ ルールを作成しているときに発生します。この場合、AWS Glue は接続をロードできません。代わりに、Amazon Redshift または Amazon RDS データセットで DQ ルールを設定します。これは既知のバグです。

LAUNCH ERROR: Error downloading from S3 for bucket

エラー状態: LAUNCH ERROR: Error downloading from S3 for bucket: `aws-glue-ml-data-quality-assets-us-east-1, key: jars/aws-glue-ml-data-quality-etl.jar.Access Denied (Service: Amazon S3; Status Code: 403; Please refer logs for details)` .

解決方法: AWS Glue Data Quality に渡されるロール内のアクセス許可は、上記の Amazon S3 の口ケーションからの読み取りを許可する必要があります。この IAM ポリシーを、次のロールに添付します。

```
{
  "Sid": "allowS3",
  "Effect": "Allow",
  "Action": "s3:GetObject",
  "Resource": "arn:aws:s3:::aws-glue-ml-data-quality-assets-<region>/*"
}
```

アクセス許可の詳細については、「Data Quality authorization」を参照してください。これらのライブラリは、データセットのデータ品質を評価する際に必要になります。

InvalidInputException (status: 400): DataQuality rules cannot be parsed

エラー状態: InvalidInputException (status: 400): DataQuality rules cannot be parsed.

解決方法: このエラーには、さまざまな原因が考えられます。その 1 つが、ルールに一重引用符が含まれていることです。二重引用符で囲まれていることを確認してください。例:

```
Rules = [
ColumnValues "tipo_vinculo" in ["CODO", "DOCO", "COCO", "DODO"] AND "categoria" = 'ES'
    AND "cod_bandera" = 'CEP'
```

上記を下記のように変更します。

```
Rules = [
(ColumnValues "tipovinculo" in [ "CODO", "DOCO", "COCO", "DODO"]) AND (ColumnValues
 "categoria" = "ES")
    AND (ColumnValues "codbandera" = "CEP")
]
```

エラー: EventBridge が、設定したスケジュールに基づいて Glue DQ ジョブをトリガーしない

エラー状態: Eventbridge is not triggering AWS Glue Data Quality jobs based on the schedule I setup.

解決方法: ジョブをトリガーするロールに適切なアクセス許可が付与されていない可能性があります。ジョブのトリガーに使用するロールに、「評価実行のスケジューリングに必要な IAM の設定」に記載されているアクセス許可が付与されていることを確認します。

CustomSQL エラー

エラー状態: `The output from CustomSQL must contain at least one column that matches the input dataset for AWS Glue Data Quality to provide row level results. The SQL query is a valid query but no columns from the SQL result are present in the Input Dataset. Ensure that matching columns are returned from the SQL。`

解決策: SQL クエリは有効ですが、主テーブルの列のみを選択していることを確認してください。主テーブルから列の sum、count などの集計関数を選択すると、このエラーが発生する可能性があります。

エラー状態: `There was a problem when executing your SQL statement: cannot resolve "Col"`。

解決策: この列は主テーブルには存在しません。

エラー状態: `The columns that are returned from the SQL statement should only belong to the primary table. "In this case, some columns ( Col ) belong to reference table"`。

解決策: SQL クエリで主テーブルを他の参照テーブルと結合する場合は、主テーブルの行レベルの結果を生成するために、SELECT ステートメントに主テーブルの列名のみが含まれていることを確認してください。

動的ルール

エラー状態: `: Dynamic rules require job context, and cannot be evaluated in interactive session or data preview.`。

原因: ルールセットに動的 DQ ルールが含まれている場合、このエラーメッセージがデータプレビュー結果や他のインタラクティブセッションに表示されることがあります。動的ルールは特定のジョブ名と評価コンテキストに関連する履歴メトリクスを参照するため、インタラクティブセッションでは評価できません。

解決策: AWS Glue ジョブを実行すると履歴メトリクスが生成され、同じジョブについて後でジョブを実行する際に参照できます。

エラー状態:

- `[RuleType] rule only supports simple atomic operands in thresholds.`.
- `Function last not yet implemented for [RuleType] rule.`

解決策: 動的ルールは通常、数値式に含まれるすべての DQDL ルールタイプでサポートされています (「DQDL リファレンス」を参照)。ただし、ColumnValues と ColumnLength など、複数のメトリクスを生成するルールはまだサポートされていないものもあります。

エラー状態: `Binary expression operands must resolve to a single number.`。

原因: 動的ルールは `RowCount > avg(last(5)) * 0.9` のようなバイナリ式をサポートしています。ここでのバイナリ式とは、`avg(last(5)) * 0.9` のことです。このルールは、オペランド `avg(last(5))` と `0.9` の両方が単一の数値に解決されるため有効です。間違った例としては、`RowCount > last(5) * 0.9` などがあります。`last(5)` は現在の行数と有意義な比較ができないリストが生成されるからです。

解決策: 集計関数を使用して、リスト値のオペランドを 1 つの数値に減らします。

エラー状態:

- `Rule threshold results in list, and a single value is expected. Use aggregation functions to produce a single value. Valid example: sum(last(10)), avg(last(10)).`
- `Rule threshold results in empty list, and a single value is expected.`

原因: 動的ルールを使用すると、データセットの一部の機能とその履歴値を比較できます。最後の関数では、正の整数の引数が指定されていれば、複数の履歴値を取得できます。例えば `last(5)` は、ルールのジョブ実行で確認された直近の 5 つの値を取得します。

解決策: 集計関数を使用してこれらの値を 1 つの数値に減らし、現在のジョブ実行で測定された値と有意義な比較を行う必要があります。

有効な例:

- `RowCount >= avg(last(5))`
- `RowCount > last(1)`
- `RowCount < last()`

無効な例: `RowCount > last(5)`。

エラー状態:

- `Function index used in threshold requires positive integer argument.`
- `Index argument must be an integer. Valid syntax example: RowCount > index(last(10, 2)), which means RowCount must be greater than third most recent execution from last 10 job runs.`

解決策: 動的ルールを作成する場合、index 集計機能を使用してリストから 1 つの履歴値を選択できます。例えば、RowCount > index(last(5), 1) は、現在のジョブで観測された行数が、そのジョブで観測された行数の中で 2 番目に新しい行数より真に大きいかどうかを確認します。index のインデックスは 0 です。

エラー状態: IllegalArgumentException: Parsing Error: Rule Type: DetectAnomalies is not valid。

解決策: 異常検出は AWS Glue 4.0 でのみ使用できます。

エラー状態: IllegalArgumentException: Parsing Error: Unexpected condition for rule of type ... no viable alternative at input ...。

注: ... は動的です。例えば、IllegalArgumentException: Parsing Error: Unexpected condition for rule of type RowCount with number return type, line 4:19 no viable alternative at input '>last' などです。

解決策: 異常検出は AWS Glue 4.0 でのみ使用できます。

ユーザークラスの例外: org.apache.spark.sql.AnalysisException: org.apache.hadoop.hive.ql.metadata.HiveException

エラー状態: Exception in User Class: org.apache.spark.sql.AnalysisException: org.apache.hadoop.hive.ql.metadata.HiveException: Unable to fetch table mailpiece_submitted. StorageDescriptor#InputFormat cannot be null for table: mailpiece_submitted (Service: null; Status Code: 0; Error Code: null; Request ID: null; Proxy: null)

原因: AWS Glue Data Catalog で Apache Iceberg を使用していますが、AWS Glue の Data Catalog の入力フォーマット属性が空です。

解決策: この問題は、DQ ルールで CustomSQL ルールタイプを使用しているときに発生します。この問題を解決する 1 つの方法は、「プライマリ」を使用するか、<database>.<table> in Custom ruletype に glue_catalog. のカタログ名を追加することです。

UNCLASSIFIED_ERROR; IllegalArgumentException : 解析エラー: ルールまたはアナライザーが指定されていません。、入力時に実行可能な代替手段がありません。

エラー状態: `UNCLASSIFIED_ERROR; IllegalArgumentException: Parsing Error: No rules or analyzers provided., no viable alternative at input`

解決方法: DQDL は解析できません。これが発生するインスタンスはいくつかあります。複合ルールを使用している場合は、正しい括弧があることを確認してください。

```
(RowCount >= avg(last(10)) * 0.6) and (RowCount <= avg(last(10)) * 1.4) instead of
      RowCount >= avg(last(10)) * 0.6 and RowCount <= avg(last(10)) * 1.4
```

AWS Glue の Amazon Q データ統合

AWS Glue の Amazon Q データ統合は、データエンジニアや ETL 開発者が自然言語を使用してデータ統合ジョブを構築できるようにする、AWS Glue の新しい生成 AI 機能です。エンジニアや開発者は Amazon Q にジョブの作成、問題のトラブルシューティング、AWS Glue やデータ統合に関する質問への回答を依頼できます。

Amazon Q とは

> **Note**
>
> Amazon Bedrock を搭載: AWS により不正使用検知が自動化されています。Amazon Q データ統合は Amazon Bedrock に構築されているため、ユーザーは Amazon Bedrock に実装されているコントロールを最大限に活用し、安全性、セキュリティ、人工知能 (AI) の責任ある使用を実施できます。

Amazon Q は、生成人工知能 (AI) を活用した会話型アシスタントであり、AWS アプリケーションの理解、構築、拡張、運用を支援します。Amazon Q を支えるモデルは高品質の AWS コンテンツで強化されているため、より完全かつ実用的で参考になる回答が得られ、AWS での構築を加速できるようにします。詳細については、「Amazon Q とは」を参照してください。

AWS Glue の Amazon Q データ統合とは

AWS Glue の Amazon Q データ統合には以下の機能が含まれています。

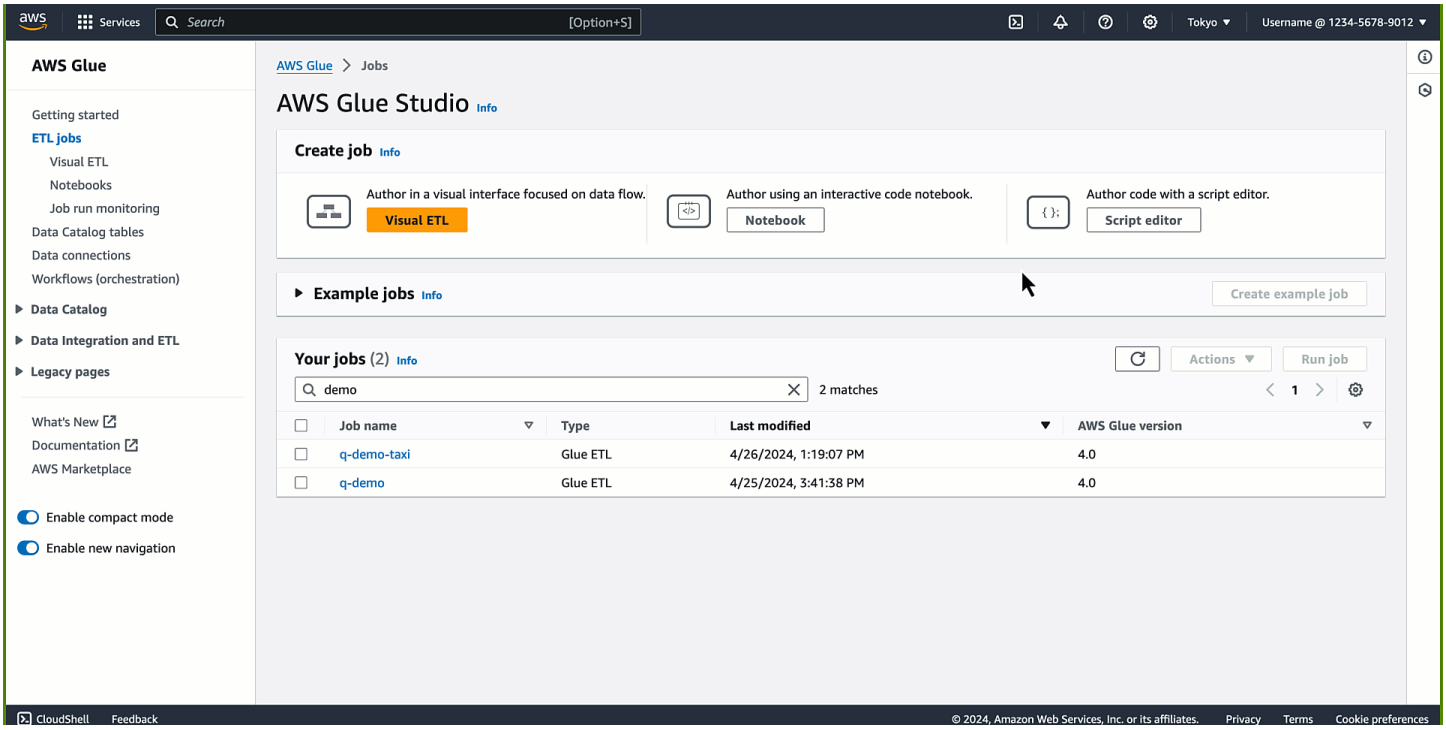
- チャット — AWS Glue の Amazon Q データ統合では、AWS Glue についてや、AWS Glueの送信元と送信先のコネクター、AWS Glue ETL ジョブ、データカタログ、クローラーや AWS Lake Formation、その他の機能に関するドキュメント、ベストプラクティスなど、データ統合分野に関する自然言語の質問に英語で回答できます。AWS Glue の Amazon Q データ統合ではステップバイステップの手順が示され、情報の発信元への参照が含まれています。
- データ統合コードの生成 — AWS Glue の Amazon Q データ統合は、AWS Glue ETL スクリプトに関する質問に回答し、英語で自然言語の質問を与えた新しいコードを生成できます。
- トラブルシューティング — AWS Glue の Amazon Q データ統合は、AWS Glue ジョブのエラーを理解しやすくすることを目的として構築され、問題の根本原因と解決のためのステップバイステップの手順を提供します。

> **Note**
>
> AWS Glue の Amazon Q データ統合では、会話中に将来の応答を通知するために会話のコンテキストを使用することはありません。AWS Glue の Amazon Q データ統合に関する各会話は、以前または将来の会話とは無関係です。

AWS Glue の Amazon Q データ統合の操作について

Amazon Q パネルでは、Amazon Q に対して AWS Glue ETL スクリプト用のコードの生成、AWS Glue 機能に関する質問への回答、エラーのトラブルシューティングを行うようにリクエストできます。応答は PySpark の ETL スクリプトであり、スクリプトをカスタマイズ、レビュー、実行するステップバイステップの手順が説明されています。質問に対して、データ統合ナレッジベースに基づいて、概要と参照用のソース URL を含む回答が生成されます。

たとえば、Amazon Q に「Snowflake から読み取り、フィールドの名前を変更し、Redshift に書き込む Glue スクリプトを提供してください」と依頼すると、それに応じて AWS Glue の Amazon Q データ統合から、要求されたアクションを実行できる AWS Glue ジョブスクリプトが返されます。生成されたコードをレビューして、要求した意図を満たしていることを確認できます。問題がなければ、本番に AWS Glue ジョブとしてデプロイできます。統合に、エラーや障害の説明や解決策の提案を依頼し、解決策を提案することで、ジョブのトラブルシューティングを行うことができます。Amazon Q は、AWS Glue やデータ統合のベストプラクティスに関する質問に回答することができます。



以下は、AWS Glue の Amazon Q データ統合が AWS Glue でのビルドにどのように役立つかを示す質問例です。

AWS Glue ETL コード生成:

- S3 から JSON を読み取り、適用マッピングを使用してフィールドを変換し、Amazon Redshift に書き込む AWS Glue スクリプトを記述します
- DynamoDB から読み込み、DropNullFields 変換を適用し、Parquet として S3 に書き込むための AWS Glue スクリプトを作成する方法を教えてください。
- MySQL から読み取り、ビジネスロジックに基づいていくつかのフィールドを削除し、Snowflake に書き込む AWS Glue スクリプトを教えてください
- DynamoDB から読み取り、S3 に JSON として書き込む AWS Glue ジョブを記述してください
- S3 向けの AWS Glue Data Catalog の AWS Glue スクリプトの開発を手伝ってください
- S3 から JSON を読み取り、null を削除して Redshift に書き込む AWS Glue ジョブを記述してください

AWS Glue 機能の説明:

- AWS Glue Data Quality の使用方法を教えてください。
- AWS Glue ジョブブックマークの使い方を教えてください。

- AWS Glue 自動スケーリングを有効にするにはどうすればよいですか?
- AWS Glue 動的フレームと Spark データフレームの違いは何ですか?
- AWS Glue はどのようなタイプの接続をサポートしていますか ?

AWS Glue のトラブルシューティング:

- AWS Glue ジョブのメモリ不足 (OOM) エラーのトラブルシューティング方法を教えてください。
- AWS Glue Data Quality の設定時に表示されるエラーメッセージにはどのようなものがありますか。また、その修正方法を教えてください。
- Amazon S3 アクセスが拒否されましたというエラーが発生した AWS Glue ジョブを修正するにはどうすればよいですか?
- AWS Glue ジョブのデータシャッフルの問題を解決する方法を教えてください。

Amazon Q データ統合を操作するためのベストプラクティス

Amazon Q データ統合を操作するためのベストプラクティスを次に示します。

- Amazon Q データ統合を利用する際の質問は、具体的にします。複雑なリクエストの場合は反復し、返される内容が正確であることを確認します。
- データ統合プロンプトを自然言語で提供する場合は、生成アシスタントが必要な点を正確に把握できるように、可能な限り具体的に質問します。「S3 からデータを抽出する」と尋ねる代わりに、「S3 から JSON ファイルを抽出する AWS Glue スクリプトを記述する」など、より詳細な情報を提供します。
- 生成されたスクリプトは実行する前に調べて、正確であることを確認します。生成されたスクリプトにエラーがある場合、または意図した内容と違う場合は、アシスタントに修正方法を指示します。
- 生成 AI は新しいテクノロジーであり、応答にはハルシネーションと呼ばれる誤りがある場合があります。現在の環境やワークロードで使用する前に、すべてのコードをテストしてエラーや脆弱性がないかを確認する必要があります。

AWS Glue サービス改善における Amazon Q データ統合

AWS Glue で Amazon Q データ統合が AWS のサービスに関する最も関連性の高い情報を提供するのに役立つように、Amazon Q からの特定のコンテンツを使用する場合があります。これには、Amazon Q への質問とその回答などがあり、サービスの改善に役立ちます。

使用するコンテンツとオプトアウト方法については、「Amazon Q Developer ユーザーガイド」の「Amazon Q Developer サービスの改善」を参照してください。

考慮事項

AWS Glue の Amazon Q データ統合を使用する前に、以下の項目について検討してください。

- 現在、コード生成は PySpark カーネルでのみ機能します。生成されたコードは Python Spark に基づく AWS Glue ジョブ用です。
- AWS Glue でサポートされている Amazon Q データ統合のコード生成機能の組み合わせについては、「サポートされているコード生成機能」を参照してください。

AWS Glue の Amazon Q データ統合の設定

次のセクションでは、AWS Glue の Amazon Q データ統合の設定について説明します。

トピック

- IAM 許可の設定

IAM 許可の設定

このトピックでは、Amazon Q チャット体験向けに構成した IAM 許可および AWS Glue Studio ノートブック体験について説明します。

トピック

- Amazon Q チャットの IAM 許可の構成
- AWS Glue Studio ノートブックの IAM 許可の構成

Amazon Q チャットの IAM 許可の構成

AWS Glue の Amazon Q データ統合が使用する API に許可を付与するには、適切な AWS Identity and Access Management (IAM) 許可が必要です。次のカスタム AWS ポリシーを IAM アイデンティティ (ユーザー、ロール、グループなど) にアタッチすることにより、許可を取得できます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:StartCompletion",
                "glue:GetCompletion"
            ],
            "Resource": [
                "arn:aws:glue:*:*:completion/*"
            ]
        }
    ]
}
```

AWS Glue Studio ノートブックの IAM 許可の構成

AWS Glue Studio ノートブックで Amazon Q データ統合を有効にするには、次の許可がノートブック IAM ロールにアタッチされていることを確認します。

> **Note**
>
> codewhisperer プレフィックスは、Amazon Q Developer とマージされたサービスのレガシー名です。詳細については、「Amazon Q Developer の名前変更 - 変更の概要」を参照してください。

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

```
                "glue:StartCompletion",
                "glue:GetCompletion"
            ],
            "Resource": [
                "arn:aws:glue:*:*:completion/*"
            ]
        },
        {
            "Sid": "AmazonQDeveloperPermissions",
            "Effect": "Allow",
            "Action": [
                "codewhisperer:GenerateRecommendations"
            ],
            "Resource": "*"
        }
    ]
}
```

> **Note**
>
> AWS Glue の Amazon Q データ統合には、プログラムで使用できる AWS SDK 経由で利用できる API がありません。Amazon Q チャットパネルまたは AWS Glue Studio ノートブックを介してこの体験を実現するため、IAM ポリシーで `StartCompletion` および `GetCompletion` の 2 つの API が使用されます。

権限の割り当て

アクセスを提供するには、ユーザー、グループ、またはロールにアクセス許可を追加します。

- AWS IAM Identity Center のユーザーおよびグループ: 許可セットを作成します。「AWS IAM Identity Center のユーザーガイド」の「許可セットの作成」の手順に従ってください。
- アイデンティティプロバイダーを介して IAM で管理されているユーザー: ID フェデレーションのロールを作成します。詳細については、「IAM ユーザーガイド」の「サードパーティー ID プロバイダー (フェデレーション) 用のロールの作成」を参照してください。
- IAM ユーザー:
  - ユーザーが担当できるロールを作成します。手順については、「IAM ユーザーガイド」の「IAM ユーザー用ロールの作成」を参照してください。

- (お奨めできない方法) ポリシーをユーザーに直接アタッチするか、ユーザーをユーザーグループに追加する。詳細については、「IAM ユーザーガイド」の「ユーザー (コンソール) へのアクセス権限の追加」を参照してください。

サポートされているコード生成機能

Amazon Q データ統合のコード生成機能の組み合わせは次のとおりです。

ソースとターゲット	[Transformation] (変換)
以下のフォーマットタイプで使用する S3: json、csv、parquet、hudi、delta	削除
AWS Glue Data Catalog	集計
Redlake	DropDuplicates
Amazon DynamoDB	Join
MySQL	フィルター
Oracle	RenameColumns
PostgresSQL	FillNull
Microsoft SQL Server	DropNull
Amazon DocumentDB / MongoDB	WithColumns
Snowflake	SQL クエリ
Google BigQuery	Union
Teradata	Select
Amazon OpenSearch Service	
Vertica	
SAP HANA	

ソースとターゲット	[Transformation] (変換)
Amazon Redshift	

インタラクションの例

AWS Glue の Amazon Q データ統合は、Amazon Q パネルに質問を入力できるようにします。AWS Glue が提供するデータ統合機能に関する質問を入力できます。詳細な回答は、参照ドキュメントとともに返されます。

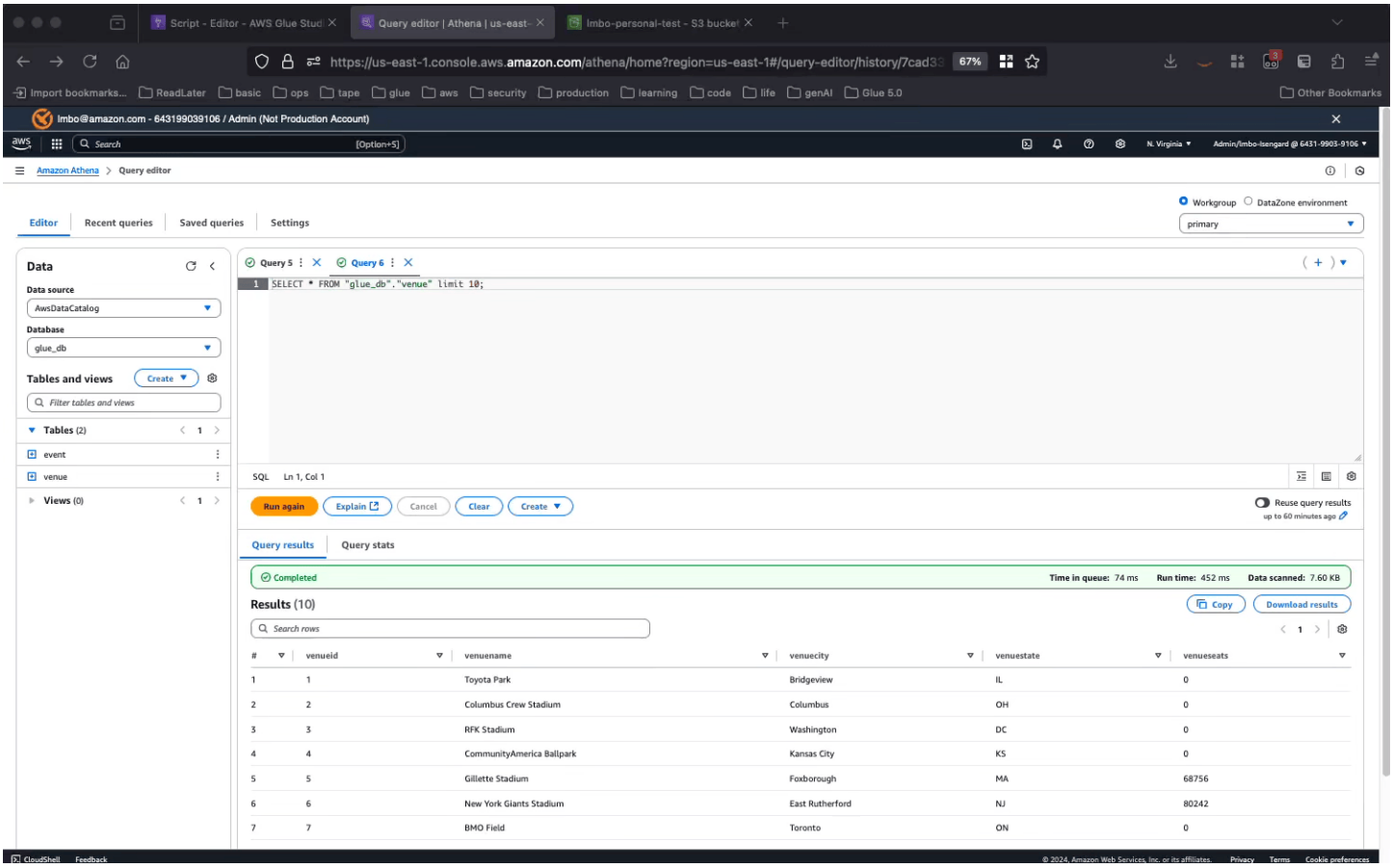
もう 1 つのユースケースは、AWS Glue ETL ジョブスクリプトの生成です。データの抽出、変換、読み込みジョブの実行方法について質問できます。生成された PySpark スクリプトが返されます。

トピック

- Amazon Q チャットインタラクション
- AWS Glue Studio ノートブックのインタラクション

Amazon Q チャットインタラクション

AWS Glue コンソールで、新しいジョブの作成を開始し、Amazon Q に以下を依頼します。:「私のデータベース glue_db で 2 つの Glue カタログテーブルの会場とイベントに接続する Glue ETL フローを作成し、会場の venueid とイベントの e_venueid の結果を結合します。次に、venuestate=='DC' という条件で会場のステータスをフィルターして、CSV 形式で s3://amzn-s3-demo-bucket/codegen/BDB-9999/output/ に書き込みます」



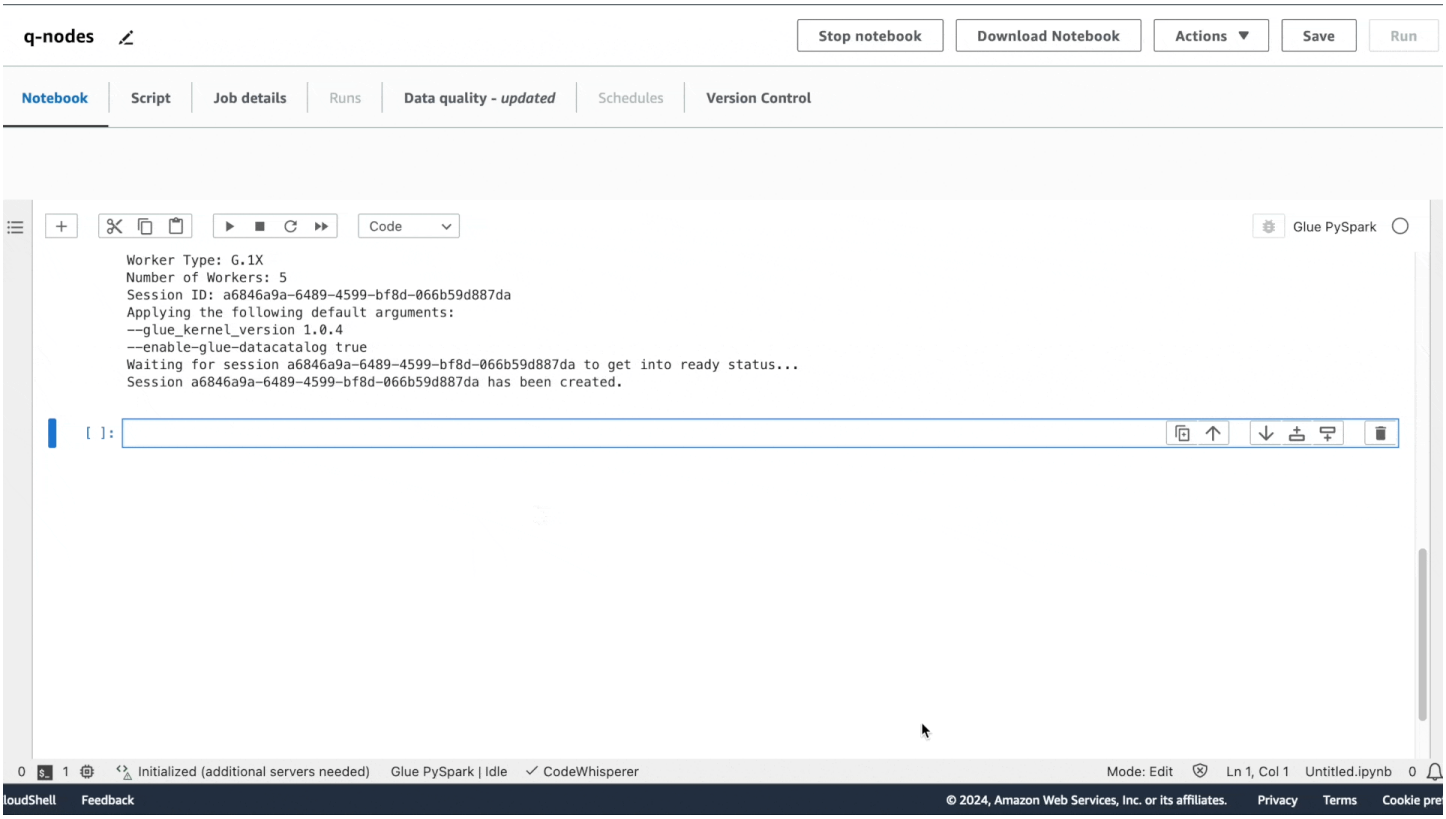
コードが生成されていることがわかります。このレスポンスを使用すると、目的に合わせて AWS Glue コードを作成する方法を理解できます。生成されたコードをスクリプトエディタにコピー/貼り付けて、プレースホルダーを設定できます。IAM ロールと AWS Glue 接続をジョブで設定したら、そのジョブを保存して実行します。ジョブが完了すると、サマリーデータが想定したとおりに Amazon S3 に保持され、ダウンストリームのワークロードで使用できることを確認できます。

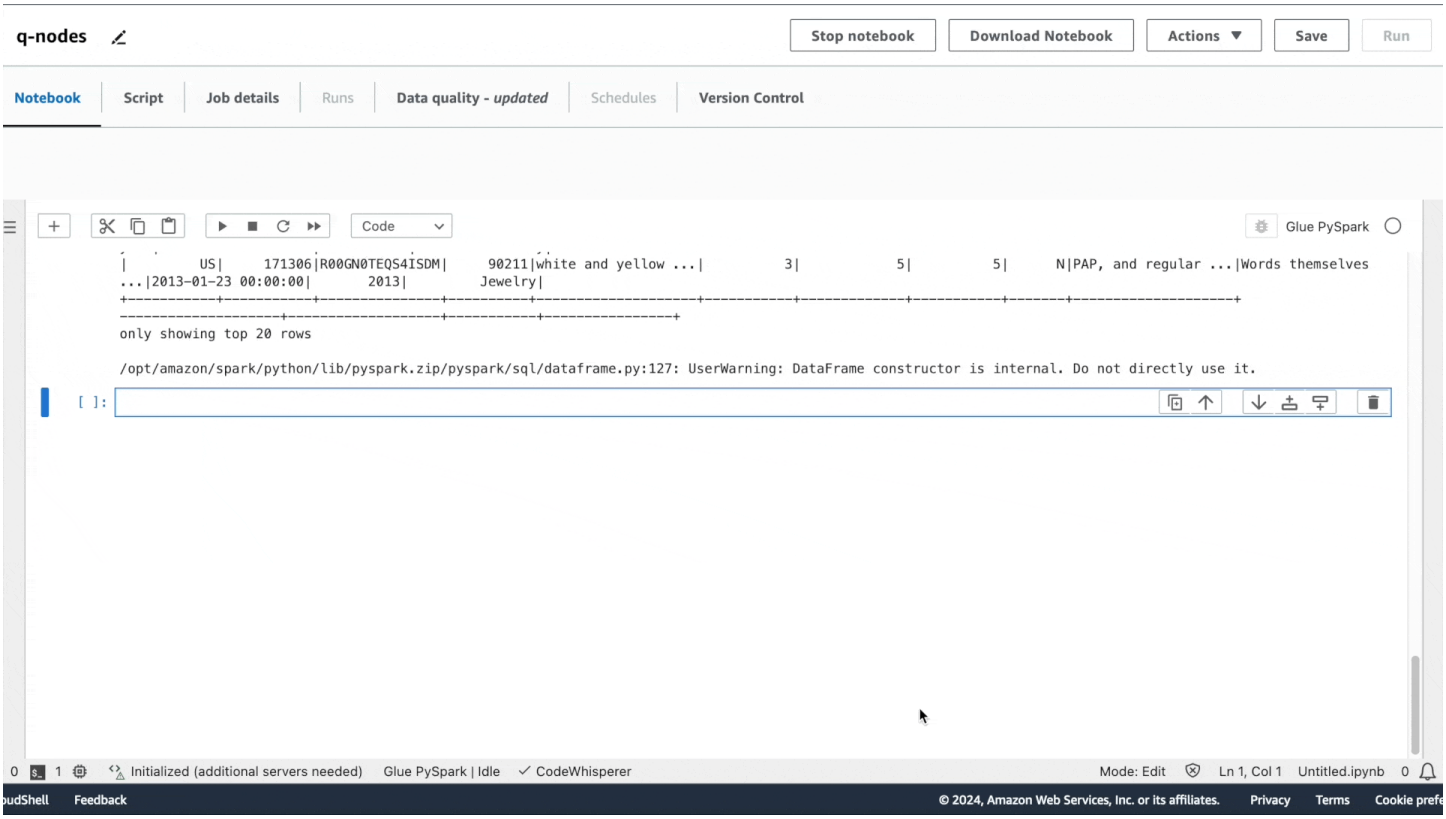
AWS Glue Studio ノートブックのインタラクション

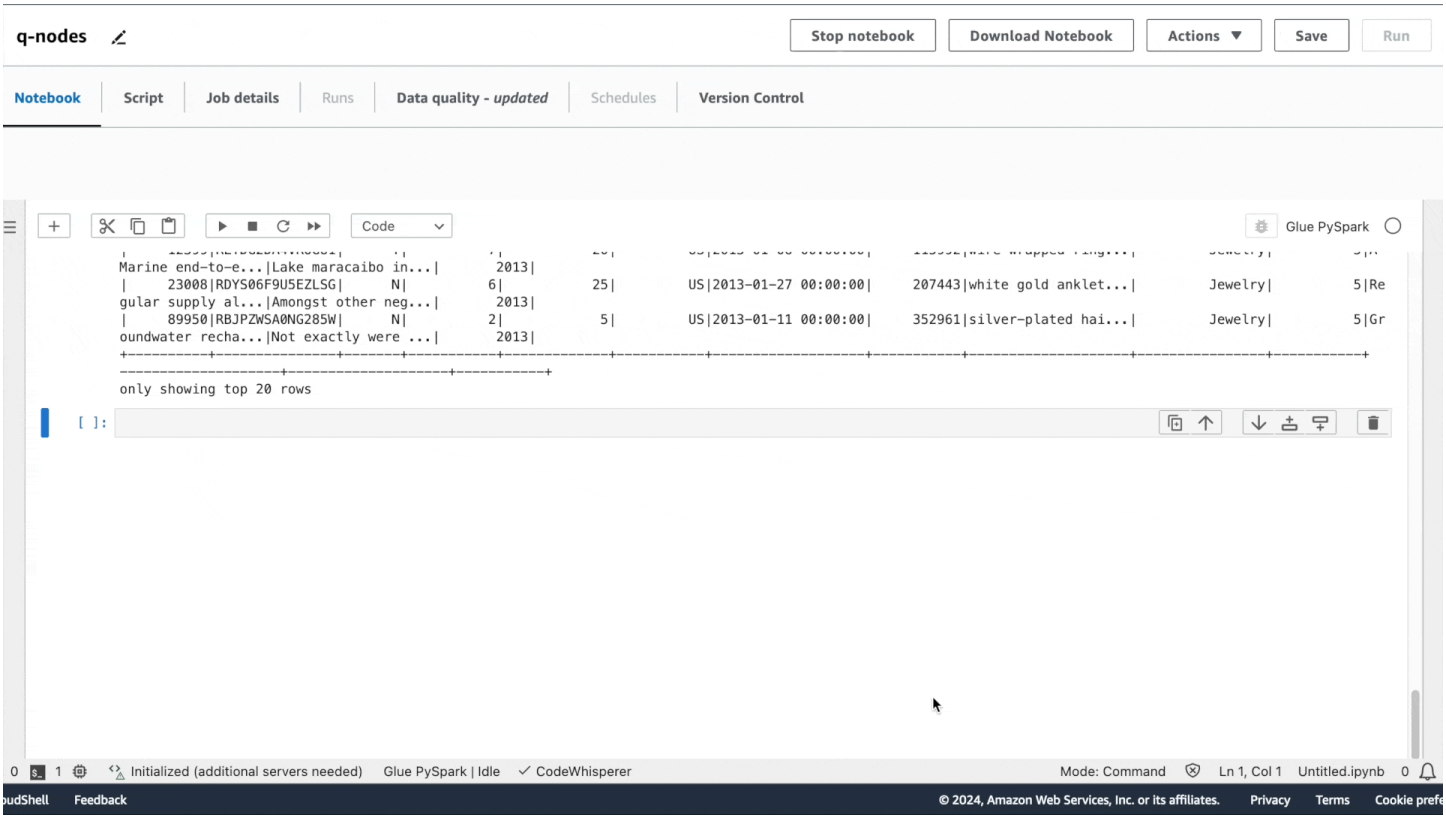
> **Note**
>
> AWS Glue Studio ノートブックの Amazon Q Data 統合エクスペリエンスは、引き続き DynamicFrame ベースのデータ統合フローに焦点を当てています。

新しいセルを追加してコメントを入力して、達成したい内容を説明します。[Tab] と [Enter] を押すと、推奨コードが表示されます。

最初の目的はデータを抽出することです。「Glue Data Catalog テーブルを読み取るコードを提供してください」、「star_rating>3 でフィルター変換を適用するコードを提供してください」、「フレームを Parquet として S3 に書き込むコードを提供してください」。




q-nodes
Stop notebook
Download Notebook
Actions
Save
Run
Notebook
Script
Job details
Runs
Data quality - updated
Schedules
Version Control
Code
Glue PySpark
only showing top 20 rows
/opt/amazon/spark/python/lib/pyspark.zip/pyspark/sql/dataframe.py:127: UserWarning: DataFrame constructor is internal. Do not directly use it.
[ ]:
Initialized (additional servers needed)
Glue PySpark | Idle
CodeWhisperer
Mode: Edit
Ln 1, Col 1
Untitled.ipynb
Feedback
© 2024, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms


q-nodes
Stop notebook
Download Notebook
Actions
Save
Run
Notebook
Script
Job details
Runs
Data quality - updated
Schedules
Version Control
Code
Glue PySpark
only showing top 20 rows
[ ]:
Initialized (additional servers needed)
Glue PySpark | Idle
CodeWhisperer
Mode: Command
Ln 1, Col 1
Untitled.ipynb
Feedback
© 2024, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms

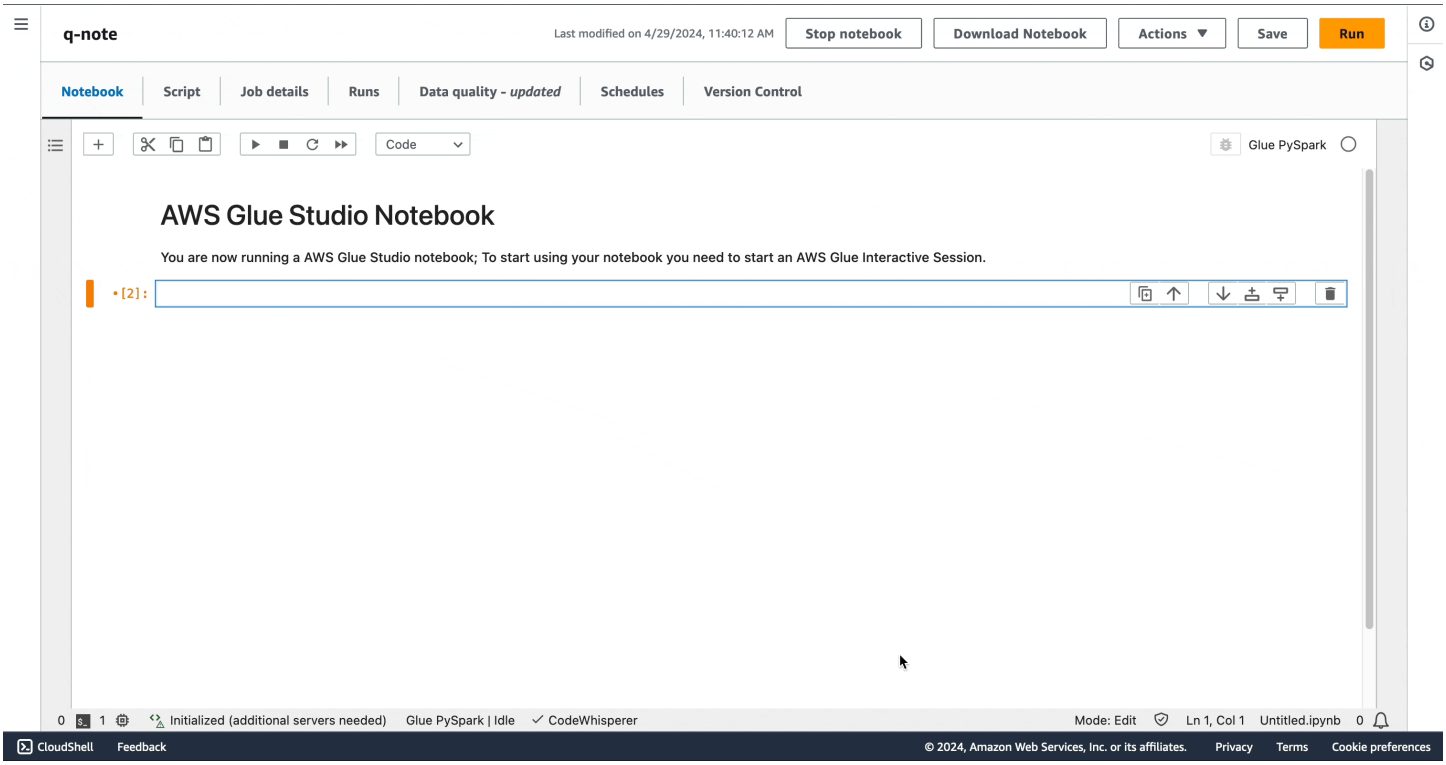
Amazon Q チャットエクスペリエンスと同様に、コードが推奨されます。[Tab] を押すと、推奨コードが選択されます。

生成されたコードのソースに適したオプションを入力して、各セルを実行できます。show( ) メソッドを使用して、実行の任意の時点でデータセットのサンプルをプレビューすることもできます。

プログラムで選択するか [実行] を選択することで、ノートブックをジョブとして実行できます。

複雑なプロンプト

1 つの複雑なプロンプトで完全なスクリプトを生成できます。S3 には JSON データがあり、Oracle には組み合わせが必要なデータがあります。両方のソースから読み取り、結合を行い、結果を Redshift に書き込む Glue スクリプトを指定してください。」



ノートブックでは、AWS Glue の Amazon Q データ統合によって、Amazon Q チャットで生成されたのと同じコードスニペットが生成されていることに気付くかもしれません。

[実行] またはプログラムで選択することで、ノートブックをジョブとして実行できます。

AWS Glue での Amazon Q データ統合によるコンテキスト認識の使用

Amazon Q Data Integration で、クエリベースのコンテキスト認識と PySpark DataFrame コード生成を使用することで、データ処理ジョブをこれまでよりもさらに効率的に作成できるようになりました。例えば、次のプロンプトを使用することで PySpark コードを生成できます。「create a job to load sales data from Redshift table 'analytics.salesorder' using connection 'erp_conn', filter order_amount below 50 dollars, and save to Amazon S3 in parquet format.」

Amazon Q は、プロンプトに基づいてスクリプトを生成し、接続設定、スキーマの詳細、データベース/テーブル名、変換の列仕様など、質問の中で提示された情報を使用してデータ統合ワークフローを設定します。接続オプションのパスワードなどの機密情報は引き続き編集されます。

プロンプトから必要な情報が提供されていない場合、Amazon Q はプレースホルダーを配置します。プレースホルダーは、コードを実行する前に、生成されたコードを適切な値で更新する必要があります。

コンテキスト認識の使用方法の例を以下に紹介します。

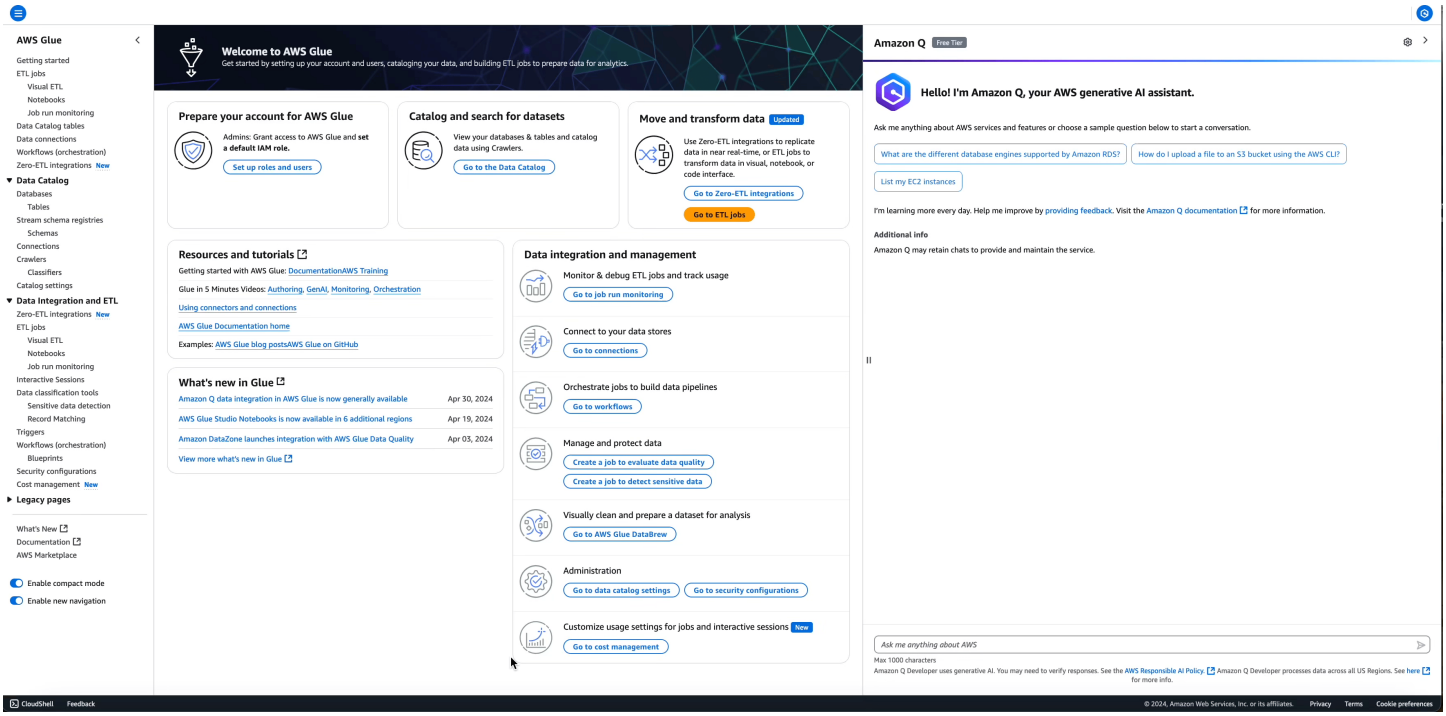
インタラクションの例

Amazon S3 ファイルの読み取りと Amazon S3 バケットへの書き込み

プロンプト: `Create an AWS Glue spark job that reads a file s3://amzn-s3-demo-bucket-input-bucket-name/input-filename.csv and saves it into the parquet file under directory s3://amzn-s3-demo-bucket-output-bucket-name/output-s3-prefix/`

```
import sys
from awsglue.transforms import *
from pyspark.context import SparkContext
from pyspark.sql import SparkSession

sc = SparkContext.getOrCreate()
spark = SparkSession.builder.getOrCreate()

# Script generated for node S3DataSource
S3DataSource_dsource1 = spark.read.format("csv").load(
    "s3://amzn-s3-demo-bucket-input-bucket-name/input-filename.csv"
```

```
)

# Script generated for node S3DataSink
S3DataSource_dsource1.write.format("parquet").mode("append").save(
    "s3://amzn-s3-demo-bucket-output-bucket-name/output-s3-prefix/"
)
```



Lakehouse からデータを取得しデータベースに書き込む

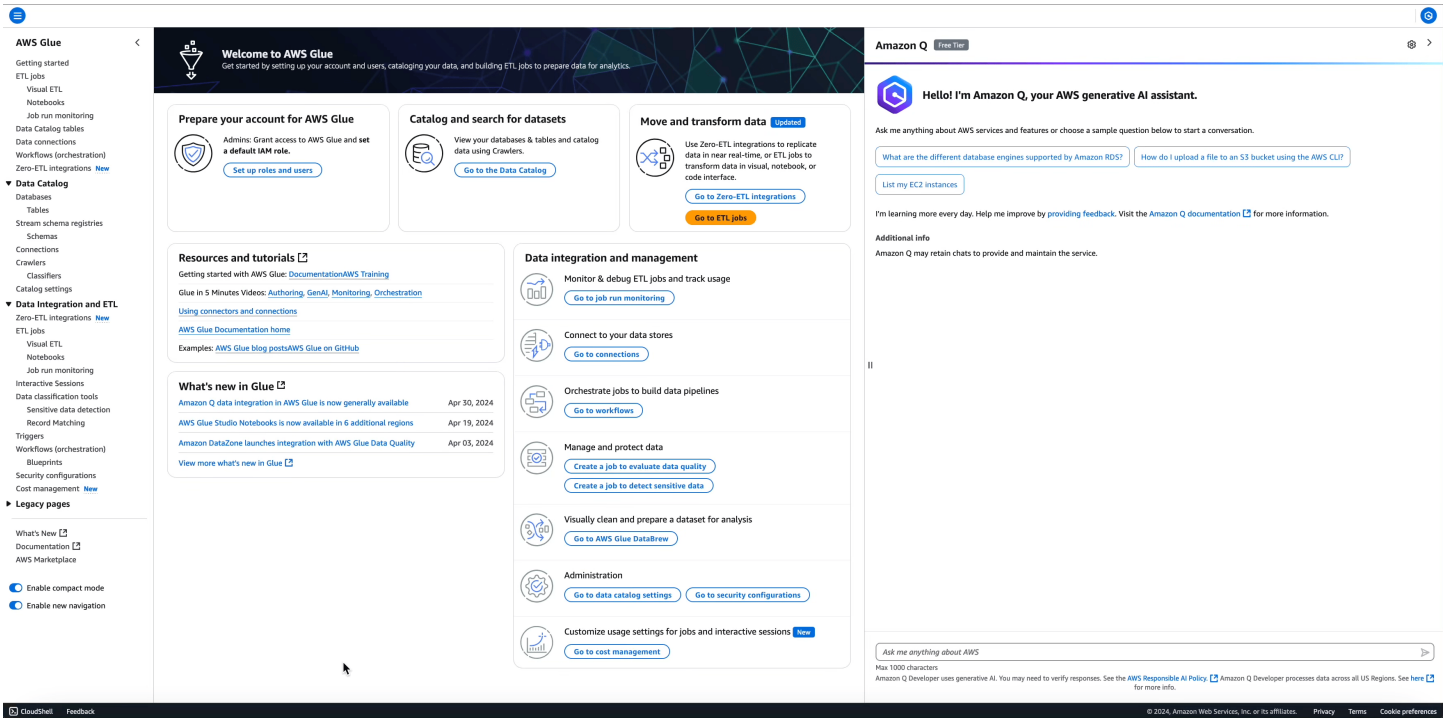
プロンプト: `write an ETL script to read from a Lakehouse table my-table in database my-database and write it to a RDS MySQL table my-target-table`

ユーザーが情報を提供しなかったフィールド (例えば connectionName が MySQL データシンクに必須であり、生成されたコードにデフォルトでプレースホルダー <connection-name> が付く場合) では、スクリプトを実行する前に必須の情報を入力するためのプレースホルダーが保持されます。

生成されたスクリプト:

```
import sys
from awsglue.transforms import *
from pyspark.context import SparkContext
from pyspark.sql import SparkSession
from connectivity.adapter import CatalogConnectionHelper
```

```
sc = SparkContext.getOrCreate()
spark = SparkSession.builder.getOrCreate()

# Script generated for node S3DataSource
S3DataSource_dsource1 = spark.read.format("parquet").load(
    "s3://amzn-lakehouse-demo-bucket/my-database/my-table"
)

# Script generated for node ConnectionV2DataSink
ConnectionV2DataSink_dsink1_additional_options = {"dbtable": "my-target-table"}
CatalogConnectionHelper(spark).write(
    S3DataSource_dsource1,
    "mysql",
    "<connection-name>",
    ConnectionV2DataSink_dsink1_additional_options,
)
```



例: 完全な ETL ワークフロー

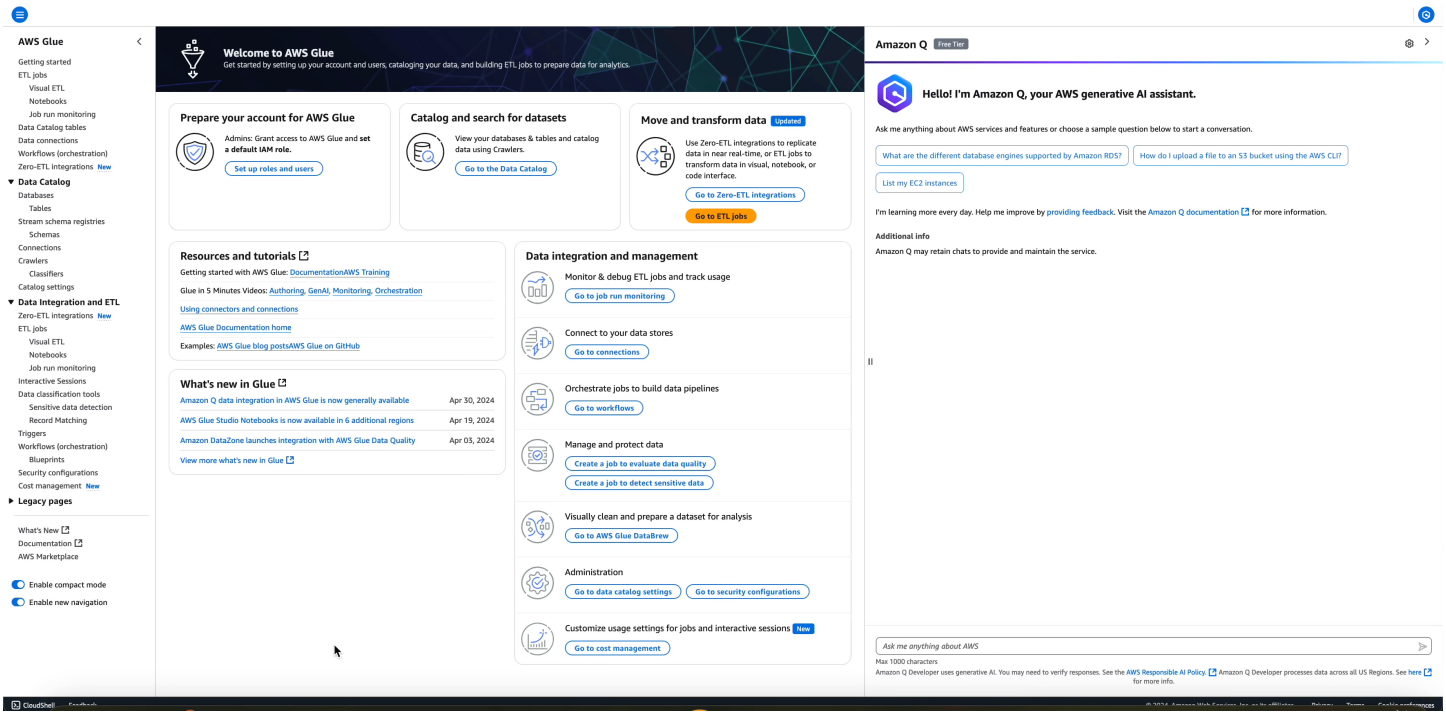
以下は、次のプロンプトを使って、AWS Glue スクリプトを作成して完全な ETL ワークフローを作成するよう AWS Glue に指示する方法の例を示したものです。`Create a AWS Glue ETL Script read from two AWS Glue Data Catalog tables venue and event in my database glue_db_4fthqih3vvk1if, join the results on the field venueid, filter on venue state with condition as venuestate=='DC' after joining`

```
the results and write output to an Amazon S3 S3 location s3://amz-s3-demo-
bucket/output/ in CSV format
```

このワークフローには、異なるデータソース (AWS Glue データカタログの 2 つのテーブル) からの読み取りの他に、読み取り後のいくつかの変換が含まれ、2 つの読み取り結果を結合して何らかの条件に基づいてフィルター処理し、変換された出力を CSV 形式で Amazon S3 の宛先に書き込みます。

生成されたジョブは、データソース、変換、シンクオペレーションの詳細情報を、次のようにユーザーの質問から抽出された対応する情報で入力します。

```
import sys
from awsglue.transforms import *
from pyspark.context import SparkContext
from pyspark.sql import SparkSession

sc = SparkContext.getOrCreate()
spark = SparkSession.builder.getOrCreate()

# Script generated for node CatalogDataSource
CatalogDataSource_dsource1 = spark.sql("select * from
 `glue_db_4fthqih3vvk1if`.`venue`")

# Script generated for node CatalogDataSource
CatalogDataSource_dsource2 = spark.sql("select * from
 `glue_db_4fthqih3vvk1if`.`event`")

# Script generated for node JoinTransform
JoinTransform_transform1 = CatalogDataSource_dsource1.join(
    CatalogDataSource_dsource2,
    (CatalogDataSource_dsource1["venueid"] == CatalogDataSource_dsource2["venueid"]),
    "inner",
)

# Script generated for node FilterTransform
FilterTransform_transform2 = JoinTransform_transform1.filter("venuestate=='DC'")

# Script generated for node S3DataSink
FilterTransform_transform2.write.format("csv").mode("append").save(
    "s3://amz-s3-demo-bucket/output//output/"
)
```



制限

- コンテキストの持ち越し:
  - コンテキスト認識機能は、同じ会話内の以前のユーザークエリのコンテキストのみを継承します。直前のクエリ以外のコンテキストは保持されません。
- ノード設定のサポート:
  - 現在、コンテキスト認識はさまざまなノードの必須設定のサブセットのみをサポートしています。
  - オプションのフィールドのサポートは、今後のリリースで予定されています。
- 可用性:
  - コンテキスト認識と DataFrame サポートは、Q Chat および SageMaker Unified Studio ノートブックで利用できます。ただし、これらの機能は AWS Glue Studio ノートブックではまだ使用できません。

AWS Glue でのオーケストレーション

以下のセクションでは、のジョブのオーケストレーションの情報を提供しますAWS Glue。

トピック

- トリガーを使用したジョブとクローラの開始
- AWS Glue で設計図とワークフローを使用して複雑な ETL アクティビティを実行する
- AWS Glue のブループリントの開発

トリガーを使用したジョブとクローラの開始

AWS Glue では、トリガーと呼ばれる Data Catalog オブジェクトを作成できます。このオブジェクトを使用して、手動または自動で 1 つ以上のクローラを開始したり、抽出、変換、ロード (ETL) ジョブを実行したりできます。トリガーを使用して、依存関係にあるジョブとクローラのチェーンを設計できます。

> **Note**
>
> この同じことは、ワークフローを定義することでも可能です。ワークフローは、複雑なマルチジョブ ETL オペレーションを作成する場合に適しています。詳しくは、「the section called “設計図とワークフローを使用した複雑な ETL アクティビティの実行”」を参照してください。

トピック

- AWS Glue トリガー
- トリガーの追加
- トリガーの有効化と無効化

AWS Glue トリガー

起動されると、トリガーは指定されたジョブとクローラを開始できます。トリガーは、オンデマンドで、スケジュールに基づいて、またはイベントの組み合わせに基づいて起動します。

> **Note**
>
> 1 つのトリガーでアクティブにできるクローラは 2 つだけです。複数のデータストアをクロールする場合は、複数のクローラを同時に実行するのではなく、クローラごとに複数のソースを使用します。

トリガーは作成されると複数の状態のいずれかになります。たとえば、CREATED、ACTIVATED、または DEACTIVATED になります。ACTIVATING などの移行状態もあります。トリガーの起動を一時的に停止するために、トリガーを無効化できます。その後、再度有効化できます。

以下の 3 種類のトリガーがあります。

予定

cron に基づく時間ベースのトリガー。

スケジュールに基づいて、一連のジョブまたはクローラのトリガーを作成できます。ジョブまたはクローラが実行される頻度、実行される曜日、実行される時間などの制約を指定できます。これらの制約は cron に基づいています。トリガーにスケジュールを設定するときは、cron の機能と制限を考慮してください。たとえば、毎月 31 日にクローラを実行することを選択した場合、いくつかの月には 31 日間はないことに注意してください。cron の詳細については、ジョブとクローラの時間ベースのスケジュール を参照してください。

条件付き

前のジョブまたはクローラや、複数のジョブまたはクローラが、条件のリストを満たしたときに起動するトリガー。

条件付きトリガーを作成するときは、監視対象のジョブとクローラのリストを指定します。監視対象のジョブまたはクローラごとに、監視するステータス (成功、失敗、タイムアウトなど) を指定します。監視対象のジョブまたはクローラが指定されたステータスで終了すると、トリガーが起動します。監視対象のイベントのいずれかまたはすべてが発生したときに起動するようにトリガーを設定できます。

たとえば、ジョブ J1 とジョブ J2 の両方が正常に完了したときにジョブ J3 を開始するトリガー T1 を設定し、ジョブ J1 またはジョブ J2 のいずれかが失敗した場合にジョブ J4 を開始する別のトリガー T2 を設定できます。

以下の表に、トリガーの監視対象のジョブとクローラの完了状態 (イベント) を示しています。

ジョブの完了状態	クローラの完了状態
• SUCCEEDED • STOPPED • FAILED • TIMEOUT	• SUCCEEDED • FAILED • CANCELLED

オンデマンド

有効化されると起動するトリガー。オンデマンドトリガーが ACTIVATED または DEACTIVATED 状態になることはありません。それらのトリガーは常に CREATED 状態のままです。

それらのトリガーは作成されるとすぐに起動可能な状態になるように、作成時に、スケジュールトリガーと条件付きトリガーを有効化するフラグを設定できます。

> **Important**
>
> 他のジョブまたはクローラの完了の結果として実行されるジョブまたはクローラは、依存関係にあると言われます。依存関係にあるジョブまたはクローラは、完了したジョブまたはクローラがトリガーによって開始された場合にのみ開始されます。依存関係チェーンのすべてのジョブまたはクローラは、1 つのスケジュールまたはオンデマンドトリガーの子であることが必要です。

トリガーを使用してジョブパラメータを渡す

トリガーは、開始するジョブにパラメータを渡すことができます。パラメータとしては、ジョブ引数、タイムアウト値、セキュリティ設定などがあります。トリガーが複数のジョブを開始する場合、パラメータは各ジョブに渡されます。

以下に示しているのは、トリガーによって渡されるジョブ引数のルールです。

- キーと値のペアのキーがデフォルトのジョブ引数と一致する場合、渡された引数はデフォルトの引数を上書きします。キーがデフォルトの引数と一致しない場合、その引数は追加の引数としてジョブに渡されます。
- キーと値のペアのキーが上書不可の引数と一致する場合、渡された引数は無視されます。

詳細については、AWS Glue API の「the section called “トリガー”」を参照してください。

トリガーの追加

トリガーは、AWS Glue コンソール、AWS Command Line Interface (AWS CLI)、または AWS Glue API を使用して追加できます。

> **Note**
>
> 現在、トリガーを使用するとき、AWS Glue コンソールはジョブのみをサポートし、クローラはサポートしません。AWS CLI または AWS Glue API を使用すると、ジョブとクローラの両方でトリガーを設定できます。

トリガーを追加するには (コンソール)

1. AWS Management Consoleにサインインし、AWS Glue コンソール (https://console.aws.amazon.com/glue/) を開きます。
2. ナビゲーションペインの [ETL] で、[トリガー] を選択します。[トリガーを追加] を選択します。
3. 以下のプロパティを指定します。

   名前

   トリガーに一意の名前を付けます。

   トリガータイプ

   次のいずれかを指定します。

   - [Schedule (スケジュール)]: トリガーは特定の頻度と時間で起動します。
   - [ジョブイベント]: 条件付きトリガー。トリガーは、リスト内のいずれかまたはすべてのジョブが、指定されたステータスと一致すると起動します。トリガーを起動するには、監視されたジョブがトリガーによって開始されている必要があります。どのジョブを選択した場合でも、監視できるジョブイベントは 1 つ (完了ステータス) のみです。
   - [オンデマンド]: トリガーは、有効化されると起動します。
4. トリガーウィザードを完了します。[Review] (確認) ページで [Schedule] (スケジュール) および [Job events] (ジョブイベント) (条件付き) は、[Enable trigger on creation] (作成時にトリガーを有効化する) を選択すると直ちにトリガーを有効化できます。

トリガーを追加するには (AWS CLI)

- 以下のようなコマンドを入力します。

```
aws glue create-trigger --name MyTrigger --type SCHEDULED --schedule  "cron(0 12 *
 * ? *)" --actions CrawlerName=MyCrawler --start-on-creation
```

このコマンドは、MyTrigger という名前のスケジュールトリガーを作成します。このトリガーは、毎日 UTC 午後 12 時に実行され、MyCrawler という名前のクローラを開始します。トリガーは有効化された状態で作成されます。

詳細については、the section called “AWS Glue トリガー” を参照してください。

ジョブとクローラの時間ベースのスケジュール

AWS Glue では、ジョブとクローラの時間ベースのスケジュールを定義できます。これらのスケジュールの定義は、Unix と同様の cron 構文を使用します。協定世界時 (UTC) で時間を指定します。スケジュールの最小精度は 5 分です。

スケジュールを使用して実行するようにジョブとクローラを設定する方法の詳細については、「トリガーを使用したジョブとクローラの開始」を参照してください。

cron 式

cron 式には 6 つの必須フィールドがあり、それらは空白で区切られます。

[Syntax] (構文)

```
cron(Minutes Hours Day-of-month Month Day-of-week Year)
```

フィールド	値	ワイルドカード
分	0 ~ 59	, - * /
時間	0 ~ 23	, - * /
日	1 ~ 31	, - * ? / L W
月	1 ~ 12 または JAN ~ DEC	, - * /

フィールド	値	ワイルドカード
曜日	1 ~ 7 または SUN ~ SAT	, - * ? / L
年	1970 ~ 2199	, - * /

ワイルドカード

- , (カンマ) のワイルドカードには、追加の値が含まれます。Month フィールドの、JAN,FEB,MAR は、1 月、2 月、3 月を含みます。
- - (ダッシュ) のワイルドカードは、範囲を指定します。Day フィールドの、「1–15」は、指定した月の 1 日から 15 日を含みます。
- [*] (アスタリスク) のワイルドカードには、フィールドのすべての値が含まれます。Hours フィールドの、* にはすべての時間が含まれています。
- / (スラッシュ) のワイルドカードは、増分を指定します。Minutes フィールドで、「**1/10**」と入力して、その時間の最初の分から始めて、10 分毎を指定できます (11 分、21 分、31 分など)。
- [?] (疑問符) のワイルドカードは、任意を意味します。Day-of-month フィールドで 7 と入力し、7 日が何曜日であってもかまわない場合、Day-of-week フィールドに ? を入力できます。
- Day-of-month フィールドまたは Day-of-week フィールドにある [L] のワイルドカードは、月または週の最終日を指定します。
- Day-of-month フィールドの、ワイルドカード W は、平日を指定します。Day-of-month フィールドで、3W は月の 3 番目の平日に最も近い日を指定します。

制限

- Cron 式の Day-of-month フィールドと Day-of-week フィールドを同時に指定することはできません。一方のフィールドに値を指定する場合、もう一方のフィールドで [?] (疑問符) を使用する必要があります。
- 5 分より短い間隔を導き出す cron 式はサポートされていません。

例

スケジュールを作成するときは、以下のサンプルの cron 文字列を使用できます。

分	時間	日	月	曜日	年	意味
0	10	*	*	?	*	毎日午前 10:00 (UTC) に実行
15	12	*	*	?	*	毎日午後 12:15 (UTC) に実行
0	18	?	*	MON-FRI	*	毎週月曜日から金曜日まで午後 6:00 (UTC) に実行
0	8	1	*	?	*	毎月 1 日の午前 8:00 (UTC) に実行
0/15	*	*	*	?	*	15 分ごとに実行
0/10	*	?	*	MON-FRI	*	月曜日から金曜日まで 10 分ごとに実行
0/5	8 ~ 17	?	*	MON-FRI	*	毎週月曜日から金曜日まで午前 8:00 から午後 5:55 (UTC) の間

分	時間	日	月	曜日	年	意味
						に 5 分ごとに実行

たとえば、毎日 12:15 UTC のスケジュールで実行するには、次のように指定します。

```
cron(15 12 * * ? *)
```

トリガーの有効化と無効化

トリガーは、AWS Glue コンソール、AWS Command Line Interface (AWS CLI)、または AWS Glue API を使用して有効化または無効化できます。

トリガーを有効化または無効化するには (コンソール)

1. AWS Management Consoleにサインインし、AWS Glue コンソール (https://console.aws.amazon.com/glue/) を開きます。
2. ナビゲーションペインの [ETL] で、[トリガー] を選択します。
3. 目的のトリガーの横にあるチェックボックスをオンにし、[Action (アクション)] メニューで [トリガーの有効化] を選択して、トリガーを有効化します。または、[トリガーを無効化] を選択して、トリガーを無効化します。

トリガーを有効化または無効化するには (AWS CLI)

- 以下のいずれかのコマンドを入力します。

  ```
  aws glue start-trigger --name MyTrigger

  aws glue stop-trigger --name MyTrigger
  ```

  トリガーは、開始することで有効化され、停止することで無効化されます。オンデマンドトリガーは、有効化されると、すぐに起動します。

詳細については、the section called “AWS Glue トリガー” を参照してください。

AWS Glue で設計図とワークフローを使用して複雑な ETL アクティビティを実行する

組織が実行する複雑な抽出、変換、ロード (ETL) プロセスの中には、依存関係のある複数の AWS Glue ジョブとクローラによる実装が最適なものがあります。AWS Glue ワークフローを使用すると、マルチジョブ、マルチクローラの ETL プロセスを設計でき、このプロセスは単一のエンティティとして実行と追跡が可能です。ワークフローを作成し、その中でジョブ、クローラ、トリガーを指定した後に、オンデマンドまたはスケジュールにより、そのワークフローを実行できるようになります。

トピック

- AWS Glue のワークフローの概要
- AWS Glue でワークフローを手動により作成および構築する
- Amazon EventBridge イベントによる AWS Glue ワークフローの開始
- ワークフローを開始した EventBridge イベントの表示
- AWS Glue でのワークフローの実行とモニタリング
- ワークフロー実行の停止
- ワークフロー実行の修復と再開
- AWS Glue でのワークフローの実行プロパティの取得と設定
- AWS Glue APIを使用したワークフローのクエリ
- AWS Glue での設計図とワークフローの制限
- AWS Glue での設計図エラーをトラブルシューティングする
- AWS Glue ブループリントでのペルソナおよびロール用のアクセス許可

AWS Glue のワークフローの概要

AWS Glue では、複数のクローラ、ジョブ、およびトリガーを伴う複雑な ETL (抽出、変換、ロード) アクティビティを作成して可視化できます。各ワークフローは、含まれるすべてのジョブならびにクローラーの実行とモニタリングを管理します。ワークフローは、各コンポーネントについて、その実行の進捗状況とステータスを記録します。これにより、タスク全体の概要と各ステップの詳細を把握できます。AWS Glue コンソールは、ワークフローの状態をグラフで表示します。

ワークフローは、AWS Glue 設計図から作成することができます。あるいは、AWS Management Console または AWS Glue API を使用しながら、ワークフローとコンポーネントを毎回手動で作成

することも可能です。設計図の詳細については、「the section called “設計図の概要”」を参照してください。

ワークフロー内のトリガーは、ジョブおよびクローラの両方を開始でき、同時に、ジョブもしくはクローラの完了によって起動することができます。トリガーを使用することで、相互に依存するジョブとクローラの大規模なチェーンを作成できます。ワークフロー内でジョブとクローラの依存関係を定義するトリガーに加えて、各ワークフローには開始トリガーもあります。開始トリガーには、以下の 3 種類があります。

- スケジュール – 定義したスケジュールに基づいて、ワークフローが開始されます。スケジュールは、日次、週次、月次その他に設定することも、cron 式に基づきカスタムスケジュールを構成することも可能です。
- オンデマンド – ワークフローは、AWS Glue コンソール、API、あるいは AWS CLI から手動で開始されます。
- EventBridge イベント – ワークフローは、単一の Amazon EventBridge イベントまたは、バッチ化された Amazon EventBridge イベントの発生時に開始されます。このトリガータイプでは、AWS Glue はイベント駆動型アーキテクチャのイベントコンシューマとして機能します。すべての EventBridge イベントタイプがワークフローを開始できます。一般的なユースケースとしては、Amazon S3 バケット に新しいオブジェクトが保存された(S3 の PutObject オペレーション)場合が挙げられます。

  イベントのバッチにより開始される場合、ワークフローは、指定された数のイベントを受信するか、指定された時間が経過するまで待機します。EventBridge イベントトリガーを作成する際には、オプションでバッチ条件を指定できます。バッチ条件を指定する際には、バッチサイズ (イベント数) を指定する必要があります。また、オプションでバッチウィンドウ (秒数) を指定することもできます。デフォルトで、バッチウィンドウは最大長である 900 秒 (15 分) に設定されています。ワークフローは、最初に満たされたバッチ条件によって開始されます。バッチウィンドウは、最初のイベントが受信された時点で開始されます。トリガーの作成時にバッチ条件を指定しない場合、バッチサイズはデフォルトで 1 に設定されます。

  ワークフローが開始されると、バッチ条件はリセットされます。イベントトリガーは、ワークフローを再度開始するために、後続のバッチ条件が満たされるかどうかの監視を開始します。

  次の表は、バッチサイズとバッチウィンドウが連携して、ワークフローをトリガーする様子を示しています。

バッチサイズ	バッチウィンドウ	トリガ条件の結果
10		ワークフローは、10 個の EventBridge イベントの受信と、前のイベントから 15 分間の経過のうち、先に発生したいずれかによりトリガーされます。(ウィンドウサイズを指定しない場合のデフォルト設定は 15 分です。)
10	2 分	ワークフローは、10 個の EventBridge イベントの受信と、前のイベントから 2 分間の経過のうち、先に発生したいずれかによりトリガーされます。
1		ワークフローは、1 つ目のイベントの到着時にトリガーされます。ウィンドウサイズの設定は無効です。EventBridge イベントトリガーの作成時にバッチ条件を指定しない場合、バッチサイズはデフォルトで 1 に設定されます。

`GetWorkflowRun` API オペレーションは、ワークフローをトリガーしたバッチ条件を返します。

ワークフローの作成時は、ワークフローの開始方法に関係なく、ワークフローの同時実行の最大数を指定できます。

イベントまたはイベントのバッチが実行を開始したワークフローが最終的に失敗した場合、対象のイベントまたはイベントのバッチは、以後ワークフロー実行のためには使用されなくなります。新しいワークフローの実行は、次のイベントまたはイベントのバッチを受信した場合にのみ開始されます。

> **Important**
>
> ワークフロー内のジョブ、クローラ、トリガーの総数を、100 以下に制限します。100 を超える値を含めると、ワークフローの実行を再開または停止しようとしたときにエラーが発生する場合があります。

ワークフローに設定された同時実行の上限数を超過している場合は、仮にイベント条件が満たされていても、ワークフローの実行は開始されません。ワークフローの同時実行の制限は、想定されるイ

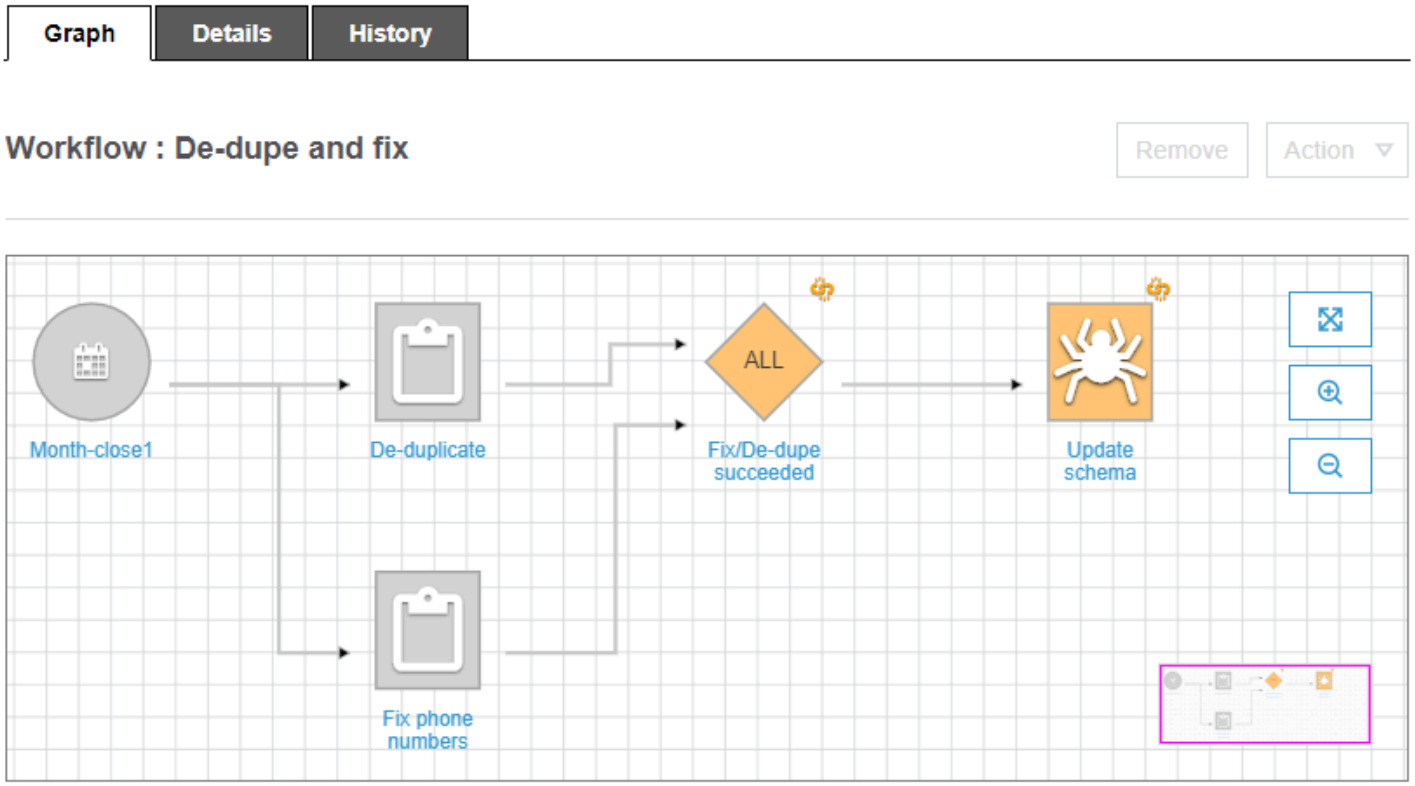
ベントボリュームに基づいて調整することをお勧めします。AWS Glue は、同時実行の制限を超えたために失敗したワークフローの実行を再試行しません。同様に、想定されるイベント量に基づいて、ワークフロー内のジョブとクローラの同時実行の制限を調整することをお勧めします。

ワークフロー実行のプロパティ

ワークフローの実行全体の状態を共有して管理するには、ワークフローのデフォルトの実行プロパティを定義できます。これらのプロパティ (名前と値のペア) は、ワークフローのすべてのジョブで使用できます。AWS Glue API を使用すると、ジョブはワークフローの実行プロパティを取得し、これらのプロパティを変更してワークフローの以降のジョブで使用できます。

ワークフローのグラフ

次の図に、AWS Glue コンソールが表示する、ワークフローの基本的なグラフを示します。ワークフローには、いくつものコンポーネントが含まれている場合があります。



このワークフローは、スケジュールトリガー `Month-close1` により開始されます。このトリガーは、2 つのジョブ (`De-duplicate` および `Fix phone numbers`) を開始します。これら 2 つのジョブが正常に完了すると、イベントトリガー `Fix/De-dupe succeeded` によってクローラ `Update schema` が開始されます。

ワークフローの静的ビューと動的ビュー

各ワークフローには、静的ビューと動的ビューの表記があります。静的ビューは、ワークフローの設計を示します。動的ビューは、各ジョブや各クローラに関する最新の実行情報を含むランタイムビューです。実行情報には、成功ステータスとエラーの詳細が含まれます。

ワークフローの実行中は、その動的ビューがコンソールに表示されます。また、完了済みのジョブと未実行のジョブがグラフで示されます。実行中のワークフローの動的ビューは、AWS Glue API を使用して取得することもできます。(詳細については、AWS Glue APIを使用したワークフローのクエリを参照してください)。

以下も参照してください。

- the section called “設計図からのワークフローの作成”
- the section called “ワークフローの手動による作成と構築”
- ワークフロー (ワークフロー API 用)

AWS Glue でワークフローを手動により作成および構築する

AWS Glue コンソールにより、ワークフローのノードを一度に 1 つずつ手動で作成し構築できます。

ワークフローは、ジョブ、クローラ、トリガーで構成されます。ワークフローの手動作成を開始する前に、ワークフローに含めるジョブとクローラを作成します。ワークフローのクローラは、オンデマンドで実行するように指定するのが最適です。トリガーは、ワークフローの構築中に新規作成できます。または、既存のトリガーをワークフロー内に 複製することもできます。トリガーをクローンすると、トリガーに関連付けられたすべてのカタログオブジェクト (トリガーを起動するジョブまたはクローラ、トリガーにより開始されるジョブまたはクローラ) がワークフローに追加されます。

Important

ワークフロー内のジョブ、クローラ、トリガーの総数を 100 以下に制限します。100 を超える値を含めると、ワークフローの実行を再開または停止しようとしたときにエラーが発生することがあります。

ワークフローを構築するには、ワークフローのグラフにトリガーを追加し、トリガーごとに監視対象のイベントやアクションを定義します。まず、開始トリガーとしてオンデマンドトリガーまたはスケジュールトリガーを追加し、次にイベント (条件付き) トリガーを追加してグラフを完成します。

ステップ 1: ワークフローを作成する

1. AWS Management Consoleにサインインし、AWS Glue コンソール (https://console.aws.amazon.com/glue/) を開きます。
2. ナビゲーションペインの [ETL] で、[ワークフロー] を選択します。
3. [ワークフロー] を選択し、[Add a new ETL workflow (新しい ETL ワークフローの追加)] フォームに入力します。

   オプションとして追加したデフォルトの実行プロパティは、ワークフローのすべてのジョブに対する引数として使用できます。詳細については、AWS Glue でのワークフローの実行プロパティの取得と設定 を参照してください。
4. [Add workflow (ワークフローの追加)] を選択します。

   新しいワークフローが [ワークフロー] ページのリストに表示されます。

ステップ 2: 開始トリガーを追加する

1. [ワークフロー] ページで、新しいワークフローを選択します。次に、ページの下部で [Graph] (グラフ) タブが選択されていることを確認します。
2. [トリガーを追加] を選択し、[トリガーを追加] ダイアログボックスで、次のいずれかの操作を行います。

   - [Clone existing (既存の複製)] を選択し、複製するトリガーを選択します。その後、[Add] (追加) を選択します。

     トリガーがグラフに表示されます。トリガーで監視するジョブやクローラおよびトリガーで開始するジョブやクローラも表示されます。

     トリガーを間違えて選択した場合は、そのトリガーをグラフで選択し、[削除] を選択します。
   - [Add new (新規追加)] を選択し、[トリガーを追加] フォームに入力します。
     1. [Trigger type] (トリガータイプ) で、[Schedule] (スケジュール)、[On demand] (オンデマンド)、または [EventBridge event] (EventBridge イベント) のいずれかを選択します。

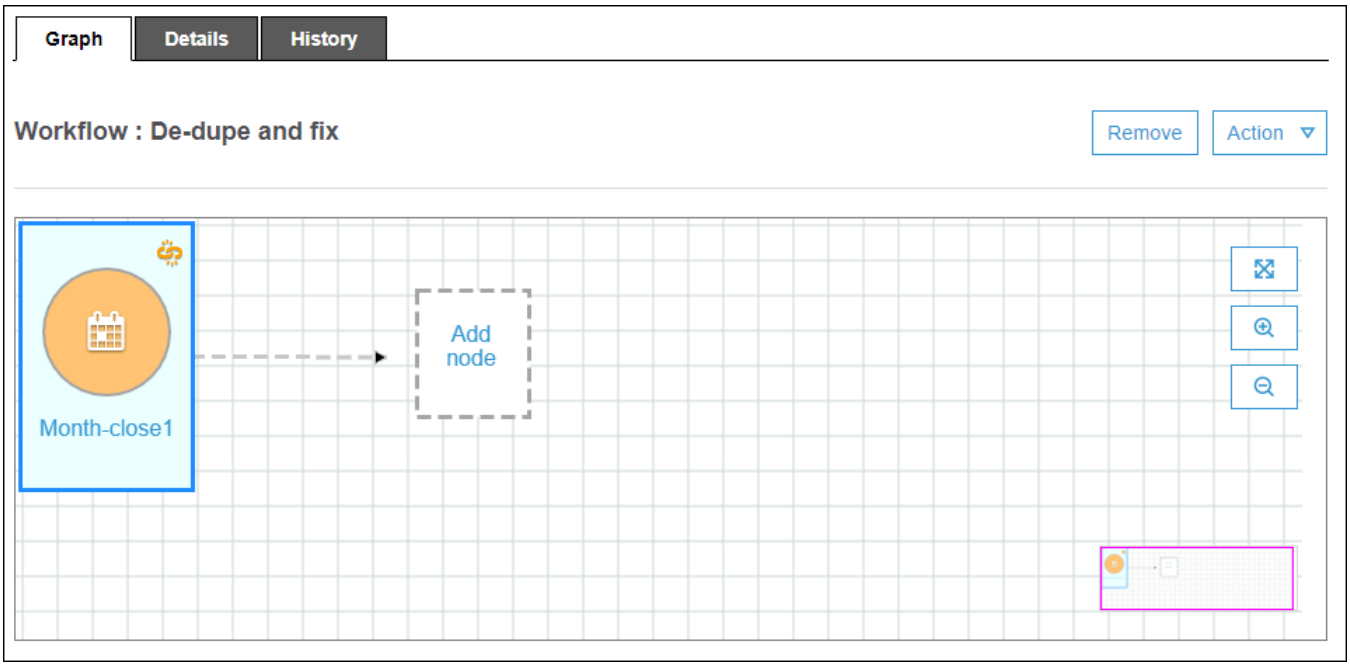
        トリガータイプを [Schedule] (スケジュール) にした場合、[Frequency] (頻度) オプションの中から 1 つを選択します。`cron` 式を入力するには、[Custom] (カスタム) をクリックします。

トリガータイプに [EventBridge event] (EventBridge イベント) を選択した場合は、[Number of events] (イベント数) (バッチサイズ) を入力し、オプションで [Time delay] (遅延時間) (バッチウィンドウ) を入力します。[Time delay] (遅延時間) を省略した場合、デフォルトでバッチウィンドウには 15 分が指定されます。詳細については、AWS Glue のワークフローの概要 を参照してください。

2. [追加] を選択します。

トリガーがグラフ上に表示されます。プレースホルダーノード ([Add node (ノードの追加)] というラベルが付いたノード) も一緒に表示されます。以下の例では、開始トリガーは `Month-close1` という名前のスケジュールされたトリガーです。

この段階では、トリガーは保存されていません。



3. 新しいトリガーを追加した場合は、以下のステップを実行します。

   a. 次のいずれかを行います。

      - プレースホルダーノード ([Add node (ノードの追加]) を選択します。
      - 開始トリガーが選択されていることを確認し、 [アクション] メニューの [Add jobs/crawlers to trigger (トリガーにジョブ/クローラを追加)] を選択します。

   b. Add job(s) and crawler(s) to trigger (トリガーにジョブおよびクローラを追加)] ダイアログボックスで、1 つ以上のジョブまたはクローラを選択し、 [追加] を選択します。

トリガーが保存されます。選択したジョブまたはクローラが、トリガーからのコネクタと共にグラフに表示されます。

ジョブやクローラを間違って追加した場合は、トリガーまたはコネクタを選択して、[削除] を選択できます。

ステップ 3: さらにトリガーを追加する

[Event] (イベント) タイプのトリガーをさらに追加して、ワークフローの構築を続けます。グラフのキャンバスを拡大または縮小するには、グラフの右側にあるアイコンを使用します。追加するトリガーごとに、以下の手順を実行します。

> **Note**
>
> ワークフローの保存のために、行うべきアクションはありません。最後のトリガーを追加し、そのトリガーにアクションを割り当てると、ワークフローが完了し保存されます。後の任意のタイミングでこの作業に戻り、さらにノードを追加することができます。

1. 次のいずれかを行います。

   - 既存のトリガーを複製するには、グラフで選択されているノードがないことを確認し、 [アクション] メニューの [トリガーを追加] を選択します。
   - グラフ上の特定のジョブまたはクローラを監視する新しいトリガーを追加するには、そのジョブまたはクローラのノードを選択し、[トリガーを追加] プレースホルダーノードを選択します。

     後のステップで、このトリガーで監視するジョブやクローラをさらに追加できます。

2. [トリガーを追加] ダイアログボックスで、次のいずれかの操作を行います。

   - [Add new (新規追加)] を選択し、[トリガーを追加] フォームに入力します。その後、[Add] (追加) を選択します。

     トリガーがグラフに表示されます。後のステップでトリガーを完了します。
   - [Clone existing (既存の複製)] を選択し、複製するトリガーを選択します。その後、[Add] (追加) を選択します。

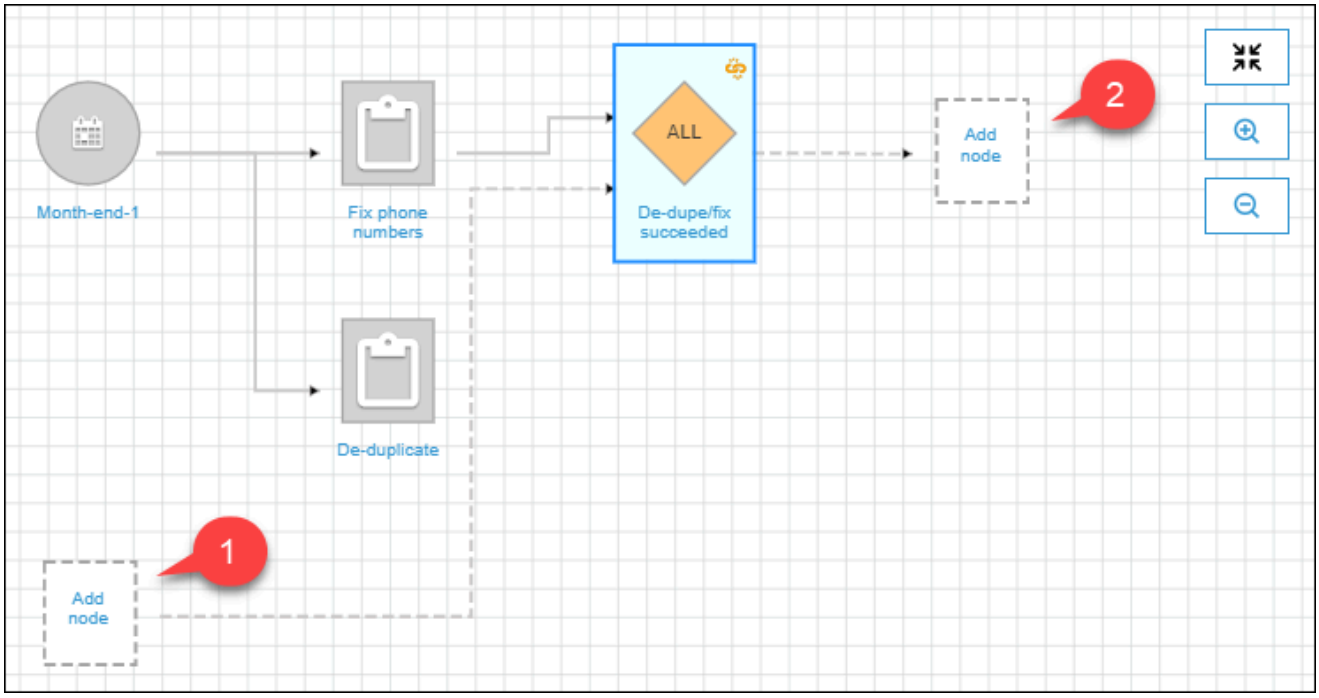
トリガーがグラフに表示されます。トリガーで監視するジョブやクローラおよびトリガーで開始するジョブやクローラも表示されます。

トリガーを間違って選択した場合は、そのトリガーをグラフで選択し、[削除] を選択します。

3. 新しいトリガーを追加した場合は、以下のステップを実行します。

   a. 新しいトリガーを選択します。

      次のように、トリガー `De-dupe/fix succeeded` が選択された状態のグラフが表示され、プレースホルダーノードが (1) イベント用、ならびに (2) アクション用として表示されます。



   b. (トリガーでイベントを既に監視しており、監視対象のジョブやクローラをを追加する場合のオプション) 監視対象イベントのプレースホルダーノードを選択し、[Add job(s) and crawler(s) to watch (監視するジョブやクローラの追加)] ダイアログボックスで 1 つ以上のジョブまたはクローラを選択します。監視対象のイベント (成功、失敗など) を選択して、[追加] を選択します。

   c. トリガーが選択されていることを確認し、アクション用のプレースホルダーノードを選択します。

   d. [Add job(s) and crawler(s) to watch (監視対象のジョブやクローラの追加)] ダイアログボックスで 1 つ以上のジョブまたはクローラを選択し、[追加] を選択します。

選択したジョブやクローラがグラフに表示されます。トリガーからのコネクタも表示されます。

Express ワークフローと サービス統合の詳細については、以下を参照してください。

- AWS Glue のワークフローの概要
- AWS Glue でのワークフローの実行とモニタリング
- AWS Glue で設計図からワークフローを作成する

Amazon EventBridge イベントによる AWS Glue ワークフローの開始

Amazon EventBridge (別名 CloudWatch Events) を使用すると、AWS サービスを自動化して、アプリケーションの可用性の問題やリソースの変更などのシステム的なイベントに自動的に対応できます。AWS のサービスからのイベントは、ほぼリアルタイムに EventBridge に提供されます。簡単なルールを記述して、注目するイベントと、イベントがルールに一致した場合に自動的に実行するアクションを指定できます。

EventBridge の機能を使用することで、AWS Glue は、イベント駆動型アーキテクチャにおけるイベントのプロデューサおよびコンシューマとして動作します。ワークフローに対して AWS Glue では、コンシューマーとして、あらゆるタイプの EventBridge イベントをサポートしています。このための最も一般的なユースケースとしては、Amazon S3 バケットが新しいオブジェクトを受け取る場合が挙げられます。データが、不規則または定義されていない間隔で到着した際には、可能なかぎり素早くそれを処理できます。

> **Note**
>
> AWS Glue は、EventBridge メッセージの配信を保証していません。EventBridge により重複したメッセージが配信された場合にも、AWS Glue は、重複の排除を行いません。対応すべき処理は、ユースケースに基づいて管理する必要があります。
> 不要なイベントが送信されないように、EventBridge ルール適切に設定します。

開始する前に

Amazon S3 データイベントを使用してワークフローを開始する場合は、対象の S3 バケットのイベントが AWS CloudTrail および EventBridge に口グ記録されたことを確認します。そのために

は、CloudTrail の証跡を作成する必要があります。詳細については、「Creating a trail for your AWS account」を参照してください。

EventBridge イベントを使用してワークフローを開始するには

> **Note**
>
> 以下のコマンドで、下記のような置き換えを行います。
>
> - *<workflow-name>* には、ワークフローに割り当てる名前。
> - *<trigger-name>* には、トリガーに割り当てる名前。
> - *<bucket-name>* には、Amazon S3 バケットの名前。
> - *<account-id>* には、有効な AWS アカウント ID。
> - *<region>* には、リージョン名 (例: us-east-1)
> - *<rule-name>* には、EventBridge ルールに割り当てる名前。

1. EventBridge ルールとターゲットを作成および表示するための、AWS Identity and Access Management (IAM) アクセス許可があることを確認します。以下は、アタッチできるサンプルのポリシーです。オペレーションとリソースを制限するために、このポリシーの範囲を狭めたい場合もあります。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "events:PutRule",
                "events:DisableRule",
                "events:DeleteRule",
                "events:PutTargets",
                "events:RemoveTargets",
                "events:EnableRule",
                "events:List*",
                "events:Describe*"
            ],
            "Resource": "*"
        }
    ]
```

```
}
```

2. AWS Glue にイベントが渡される際に、EventBridge サービスが引き受けることができる IAM ロールを作成します。

   a. IAM コンソールの [Create role] (ロールの作成) ページで、[AWS Service] を選択します。次に、[CloudWatch Events] (CloudWatch イベント) サービスを選択します。

   b. [Create role] (ロールの作成) ウィザードを完了します。ポリシー `CloudWatchEventsBuiltInTargetExecutionAccess` および `CloudWatchEventsInvocationAccess` が、ウィザードにより自動的にアタッチされます。

   c. 次のインラインポリシーをロールにアタッチします。このポリシーにより、EventBridge サービスはイベントを AWS Glue に送れるようになります。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "glue:notifyEvent"
      ],
      "Resource": [
        "arn:aws:glue:<region>:<account-id>:workflow/<workflow-name>"
      ]
    }
  ]
}
```

3. 次のコマンドを入力して、ワークフローを作成します。

   この他のオプションのコマンドラインパラメータについては、AWS CLI コマンドリファレンスの「create-workflow」を参照してください。

```
aws glue create-workflow --name <workflow-name>
```

4. 次のコマンドを入力して、ワークフローのための EventBridge イベントトリガーを作成します。これがワークフローの開始トリガーになります。*<actions>*は、実行するアクション (開始するジョブとクローラ) に置き換えます。

引数 actions の記述方法については、AWS CLIコマンドリファレンスの「create-trigger」を参照してください。

```
aws glue create-trigger --workflow-name <workflow-name> --type EVENT --
name <trigger-name> --actions <actions>
```

単一の EventBridge イベントではなく、バッチされたイベントによってワークフローをトリガーする場合は、代わりに次のコマンドを入力します。

```
aws glue create-trigger --workflow-name <workflow-name> --type EVENT
 --name <trigger-name> --event-batching-condition BatchSize=<number-of-
events>,BatchWindow=<seconds> --actions <actions>
```

event-batching-condition 引数の BatchSize は必須であり、BatchWindow はオプションです。BatchWindow が省略された場合、ウィンドウはデフォルトで (最大ウィンドウサイズの) 900 秒に設定されます。

Example

次の例では、3 つの EventBridge イベントを受信した後、または最初のイベントが到着してから 5 分経過後の、いずれか早いタイミングで eventtest ワークフロー開始するトリガーを作成しています。

```
aws glue create-trigger --workflow-name eventtest --type EVENT --name objectArrival
 --event-batching-condition BatchSize=3,BatchWindow=300 --actions JobName=test1
```

5. Amazon EventBridge でルールを作成します。

   a. 任意のテキストエディタで JSON オブジェクトを作成し、ルールの詳細を記述します。

      次の例では、イベントソースとして Amazon S3 を、イベント名として PutObject を、リクエストパラメータとしてバケット名を、それぞれ指定しています。このルールは、新しいオブジェクトがバケットで受信された場合にワークフローを開始します。

```
{
  "source": [
    "aws.s3"
  ],
  "detail-type": [
```

```
    "AWS API Call via CloudTrail"
  ],
  "detail": {
    "eventSource": [
      "s3.amazonaws.com"
    ],
    "eventName": [
      "PutObject"
    ],
    "requestParameters": {
      "bucketName": [
        "<bucket-name>"
      ]
    }
  }
}
```

新しいオブジェクトがバケット内のフォルダに到着した際にワークフローを開始させるには、次のコードを requestParameters に置き換えます。

```
    "requestParameters": {
      "bucketName": [
        "<bucket-name>"
      ]
      "key" : [{ "prefix" : "<folder1>/<folder2>/*"}}]
  }
```

b. 任意のツールを使用して、ルールを記述した JSON オブジェクトの文字列をエスケープします。

```
{\n  \"source\": [\n    \"aws.s3\"\n  ],\n  \"detail-type\": [\n    \"AWS API
 Call via CloudTrail\"\n  ],\n  \"detail\": {\n    \"eventSource\": [\n
 \"s3.amazonaws.com\"\n    ],\n    \"eventName\": [\n      \"PutObject\"\n
  ],\n    \"requestParameters\": {\n      \"bucketName\": [\n        \"<bucket-
name>\"\n      ]\n    }\n  }\n}
```

c. 次のコマンドを実行して、JSON パラメータテンプレートを作成します。このテンプレートを編集して、後続の put-rule コマンドの入力パラメータを指定します。出力をファイルに保存します。この例では、ruleCommand というファイルに保存しています。

```
aws events put-rule --name <rule-name> --generate-cli-skeleton >ruleCommand
```

`--generate-cli-skeleton` パラメータの詳細については、AWS Command Line Interface ユーザーガイドの「Generating AWS CLI skeleton and input parameters from a JSON or YAML input file」を参照してください。

出力ファイルは以下のようになります。

```
{
    "Name": "",
    "ScheduleExpression": "",
    "EventPattern": "",
    "State": "ENABLED",
    "Description": "",
    "RoleArn": "",
    "Tags": [
        {
            "Key": "",
            "Value": ""
        }
    ],
    "EventBusName": ""
}
```

d. このファイルを編集して、少なくとも Name、EventPattern、および State パラメータを指定しながら、必要に応じて他のパラメータを削除します。EventPattern パラメータでは、前のステップで作成したルール詳細のために、エスケープされた文字列を指定します。

```
{
    "Name": "<rule-name>",
    "EventPattern": "{\n  \"source\": [\n    \"aws.s3\"\n  ],\n  \"detail-
type\": [\n    \"AWS API Call via CloudTrail\"\n  ],\n  \"detail\": {\n
 \"eventSource\": [\n      \"s3.amazonaws.com\"\n    ],\n    \"eventName\": [\n
      \"PutObject\"\n    ],\n    \"requestParameters\": {\n      \"bucketName
\": [\n        \"<bucket-name>\"\n      ]\n    }\n  }\n}",
    "State": "DISABLED",
    "Description": "Start an AWS Glue workflow upon new file arrival in an
 Amazon S3 bucket"
}
```

> **Note**
>
> ワークフローの構築が完了するまで、ルールを無効にしたままにしておくことをお勧めします。

e. 次の put-rule コマンドを入力し、ファイル ruleCommand から入力パラメータを読み取ります。

```
aws events put-rule --name <rule-name> --cli-input-json file://ruleCommand
```

次の出力は、正しく処理されたことを示しています。

```
{
    "RuleArn": "<rule-arn>"
}
```

6. 次のコマンドを入力して、ターゲットにルールをアタッチします。ターゲットは、AWS Glue のワークフローです。*<role-name>*は、この手順の最初に作成したロールに置き換えます。

```
aws events put-targets --rule <rule-name> --targets
 "Id"="1","Arn"="arn:aws:glue:<region>:<account-id>:workflow/<workflow-
name>","RoleArn"="arn:aws:iam::<account-id>:role/<role-name>" --region <region>
```

次の出力は、正しく処理されたことを示しています。

```
{
    "FailedEntryCount": 0,
    "FailedEntries": []
}
```

7. 次のコマンドを入力して、ルールとターゲットの接続が正常に行われたことを確認します。

```
aws events list-rule-names-by-target --target-arn arn:aws:glue:<region>:<account-
id>:workflow/<workflow-name>
```

接続に成功した場合は次のような出力になります。*<rule-name>* は作成したルールの名前です。

```
{
    "RuleNames": [
        "<rule-name>"
    ]
}
```

8. AWS Management Consoleにサインインし、AWS Glue コンソール (https://console.aws.amazon.com/glue/) を開きます。
9. ワークフローを選択し、ワークフローグラフで、開始トリガーとそのアクション (ワークフローが開始するジョブまたはクローラ) の表示を確認します。その後、ステップ 3: さらにトリガーを追加する の手順に進みます。または、AWS Glue API か AWS Command Line Interface を使用して、ワークフローに他のコンポーネントを追加します。
10. ワークフローの指定を完了したら、ルールを有効にします。

```
aws events enable-rule --name <rule-name>
```

これで、EventBridge イベントまたはイベントのバッチによって、ワークフローを開始する準備が整いました。

以下も参照してください。

- Amazon EventBridge ユーザーガイド
- AWS Glue のワークフローの概要
- AWS Glue でワークフローを手動により作成および構築する

ワークフローを開始した EventBridge イベントの表示

ワークフローを開始した Amazon EventBridge イベントのイベント ID を表示できます。ワークフローがバッチされたイベントによって開始された場合は、そのバッチ内のすべてのイベントの、イベント ID を表示することが可能です。

バッチサイズが 1 より大きいワークフローでは、どのバッチ条件 (バッチサイズでのイベントの到着数、またはバッチウィンドウの有効期限) によってワークフローが開始されたかを確認することもできます。。

ワークフローを開始した EventBridge イベントを表示するには (コンソール)

1. AWS Management Consoleにサインインし、AWS Glue コンソール (https://console.aws.amazon.com/glue/) を開きます。
2. ナビゲーションペインで、[Workflows] (ワークフロー) をクリックします。
3. ワークフローを選択します。下部で [History] (履歴) タブを選択します。
4. ワークフロー実行を選択し、[View run details] (実行の詳細を表示する) をクリックします。
5. 実行の詳細ページにある [Run properties] (実行プロパティ) フィールドで、[aws:eventIds] キーを検索します。

   このキーの値は、EventBridge イベント ID のリストです。

ワークフローを開始した EventBridge イベントを表示するには (AWS API)

- Python スクリプトに次のコードを含めます。

```
workflow_params =
 glue_client.get_workflow_run_properties(Name=workflow_name,RunId=workflow_run_id)
batched_events = workflow_params['aws:eventIds']
```

  `batched_events` は、それぞれがイベント ID である文字列のリストです。

以下も参照してください。

- Amazon EventBridge ユーザーガイド
- the section called “ワークフローの概要”

AWS Glue でのワークフローの実行とモニタリング

ワークフローの開始トリガーがオンデマンドトリガーである場合は、AWS Glue コンソールからワークフローを開始できます。ワークフローの実行とモニタリングを行うには、以下の手順を完了します。ワークフローが失敗した場合は、実行グラフを表示し、どのノードで失敗したのかを判断します。ワークフローが設計図から作成された場合は、その設計図の実行を表示して、ワークフローの作成に使用された設計図パラメータ値を確認することで、トラブルシューティングの助けになります。(詳細については、the section called “設計図の実行の表示” を参照してください)。

ワークフローを実行およびモニタリングするには、AWS Glue コンソール、API、あるいは AWS Command Line Interface (AWS CLI) を使用します

ワークフローを実行およびモニタリングするには (コンソール)

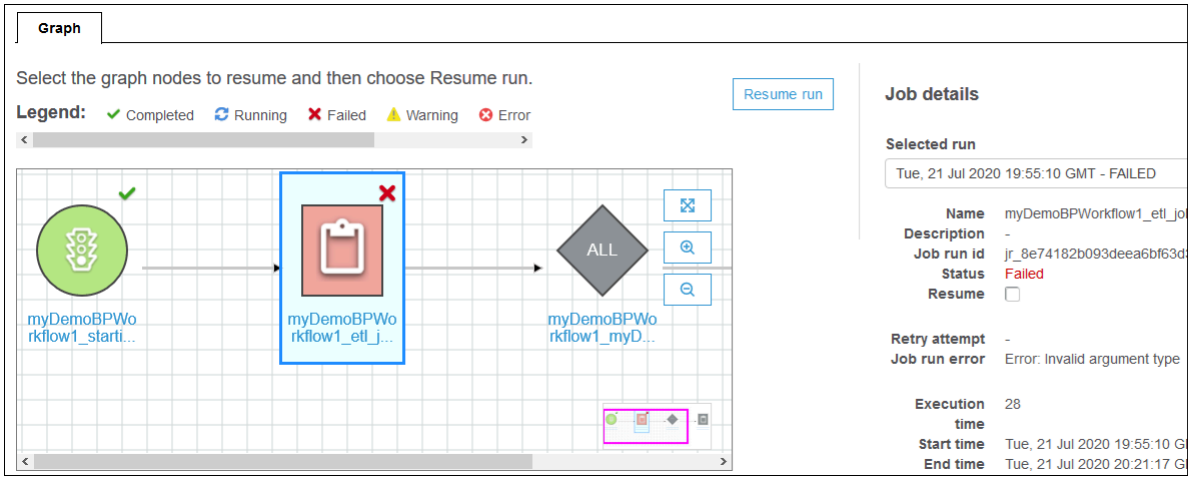
1. https://console.aws.amazon.com/glue/ で AWS Glue コンソール を開きます。
2. ナビゲーションペインの [ETL] で、[ワークフロー] を選択します。
3. ワークフローを選択します。[アクション] メニューの [実行] を選択します。
4. ワークフローのリストで、[Last run status] (最終実行ステータス) 列を確認します。進行中のワークフローのステータスを表示するには、[Refresh] (更新) ボタンをクリックします。
5. ワークフローの実行中、もしくはワークフローが完了 (または失敗) した後、以下の手順を実行して実行の詳細を表示します。

   a. ワークフローが選択されていることを確認し、[History] (履歴) タブを選択します。

   b. 現在の、または最も新しいワークフロー実行を選択し、[View run details] (実行の詳細を表示する) をクリックします。

      ワークフローの実行グラフが、現在の実行ステータスを表示します。

   c. グラフ内の任意のノードを選択して、ノードの詳細とステータスを表示します。



ワークフローを実行およびモニタリングするには (AWS CLI)

1. 次のコマンドを入力します。*<workflow-name>* は、実行するワークフローに置き換えます。

```
aws glue start-workflow-run --name <workflow-name>
```

ワークフローが正常に開始されると、コマンドから実行 ID が返されます。

2. get-workflow-run コマンドを実行して、ワークフロー実行の状態を表示します。ワークフロー名と実行 ID を指定します。

```
aws glue get-workflow-run --name myWorkflow --run-id
 wr_d2af14217e8eae775ba7b1fc6fc7a42c795aed3cbcd8763f9415452e2dbc8705
```

以下に、このコマンドの出力例を示します。

```
{
    "Run": {
        "Name": "myWorkflow",
        "WorkflowRunId":
 "wr_d2af14217e8eae775ba7b1fc6fc7a42c795aed3cbcd8763f9415452e2dbc8705",
        "WorkflowRunProperties": {
            "run_state": "COMPLETED",
            "unique_id": "fee63f30-c512-4742-a9b1-7c8183bdaae2"
        },
        "StartedOn": 1578556843.049,
        "CompletedOn": 1578558649.928,
        "Status": "COMPLETED",
        "Statistics": {
            "TotalActions": 11,
            "TimeoutActions": 0,
            "FailedActions": 0,
            "StoppedActions": 0,
            "SucceededActions": 9,
            "RunningActions": 0,
            "ErroredActions": 0
        }
    }
}
```

以下も参照してください。

- the section called “ワークフローの概要”
- the section called “設計図の概要”

ワークフロー実行の停止

ワークフロー実行を停止するには、AWS Glue コンソール、AWS Command Line Interface (AWS CLI)、または AWS Glue API を使用できます。ワークフロー実行を停止すると、すべての実行中のジョブとクローラーは即座に終了され、まだ開始されていないジョブとクローラーは開始されることがありません。すべての実行中のジョブとクローラーが停止するまでに最大 1 分かかる場合があります。ワークフロー実行のステータスは [実行中] から [停止中] に変わり、ワークフロー実行が完全に停止すると、ステータスは [停止済み] になります。

ワークフロー実行の停止後に実行グラフを表示し、完了したジョブとクローラー、および開始しなかったジョブとクローラーを確認できます。これにより、データの整合性を確保するためのステップを実行する必要があるかどうかを判断できます。ワークフロー実行を停止すると、自動ロールバックオペレーションは実行されません。

ワークフロー実行を停止するには (コンソール)

1. https://console.aws.amazon.com/glue/ で AWS Glue コンソール を開きます。
2. ナビゲーションペインの [ETL] で、[ワークフロー] を選択します。
3. 実行中のワークフローを選択し、[履歴] タブを選択します。
4. ワークフロー実行を選択し、[実行の停止] を選択します。

   実行ステータスが [停止中] に変わります。
5. (オプション) ワークフロー実行を選択し、[実行の詳細を表示する] を選択して、実行グラフを確認します。

ワークフロー実行を停止するには (AWS CLI)

- 次のコマンドを入力します。*<workflow-name>* はワークフローの名前に置き換え、*<run-id>* は停止するワークフロー実行の実行 ID に置き換えます。

```
aws glue stop-workflow-run --name <workflow-name> --run-id <run-id>
```

次に stop-workflow-run コマンドの例を示します。

```
aws glue stop-workflow-run --name my-workflow --run-id
 wr_137b88917411d128081069901e4a80595d97f719282094b7f271d09576770354
```

ワークフロー実行の修復と再開

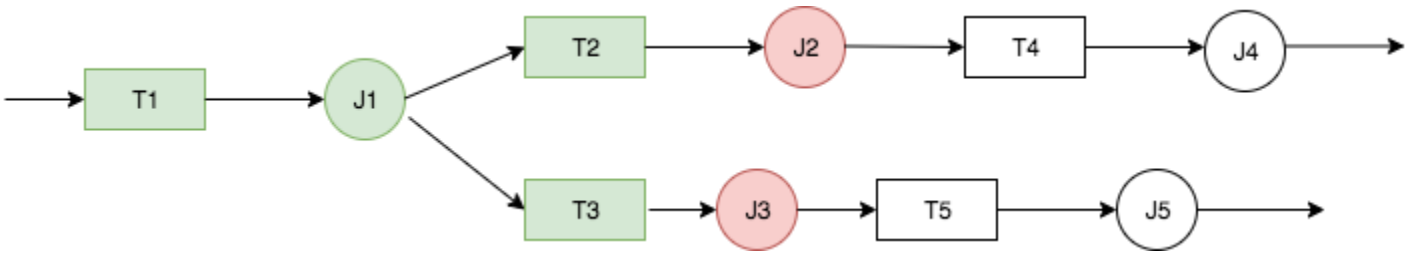
ワークフロー内の 1 つ以上のノード (ジョブまたはクローラ) が正常に完了しない場合、ワークフローは部分的にしか実行されなかったことを意味します。この根本原因を見つけて修正した上で、ワークフローの実行を再開するノードを 1 つ以上選択し、対象のワークフローを再開します。選択したノードと、それらのノードの下流にあるすべてのノードが実行されます。

トピック

- ワークフロー実行の再開の仕組み
- ワークフロー実行を再開する
- ワークフロー実行の再開に関する注意と制限事項

ワークフロー実行の再開の仕組み

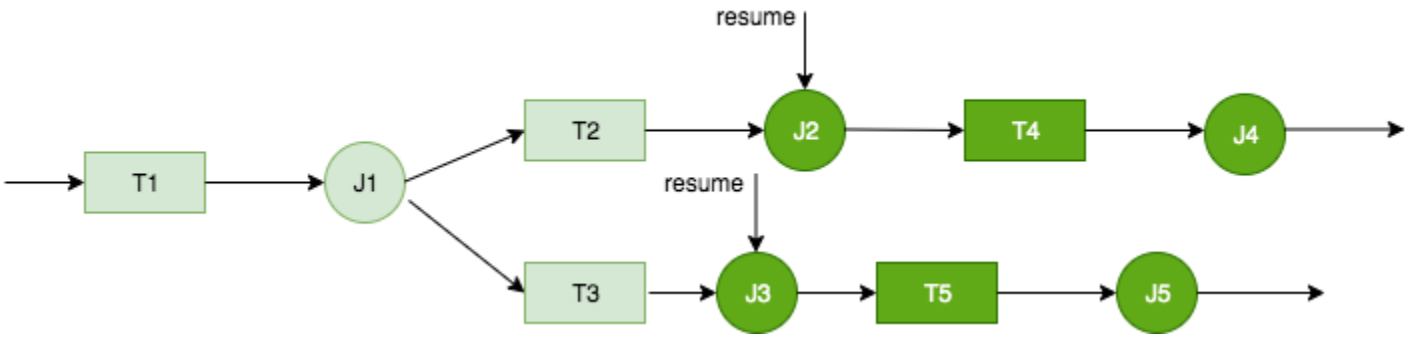
次の図にある、ワークフロー W1 について考えてみます。



ワークフロー実行は以下のように進行します。

1. トリガー T1 が、ジョブ J1 を開始します。
2. J1 が正常に完了すると、T2 と T3 がトリガーされ、それぞれジョブ J2 と J3 が実行されます。
3. ジョブ J2 と J3 が失敗します。
4. トリガー T4 と T5 は、J2 と J3 の正常な完了に依存するため、これらのトリガーは起動せずジョブ J4 と J5 も実行されません。ワークフロー W1 は部分的にのみ実行されます。

ここで、J2 と J3 の失敗の原因となった問題が修正されたとします。ワークフローの実行を再開する開始点として、J2 および J3 が選択されます。



ワークフローの実行は以下のように再開されます。

1. ジョブ J2 および J3 が正常に実行されます。
2. T4 と T5 がトリガーされます。
3. ジョブ J4 および J5 が正常に実行されます。

実行が再開されたワークフローは、新しい実行 ID を持つ別のワークフローとして追跡されます。ワークフロー履歴を表示することで、以前に行われたすべてのワークフロー実行の実行 ID を表示できます。次のスクリーンショットの例では、実行 ID `wr_c7a22...` (2 行目) を持つワークフローには、完了できなかったノードがあります。ユーザーが問題を解決し、ワークフローの実行を再開しました。その結果が、実行 ID `wr_a07e55...` (1 行目) として示されています。



> **Note**
>
> この説明の残りの部分では、「再開されたワークフロー実行」という表現は、以前のワークフロー実行が再開された際に作成されたワークフロー実行のことを指します。「元のワークフロー実行」とは、部分的にしか実行されず、再開する必要があったワークフロー実行を指します。

再開されたワークフロー実行のグラフ

再開されたワークフロー実行では、ノードのサブセットのみが実行されますが、実行のグラフは完全な形で表示されます。つまり、再開されたワークフローで実行されなかったノードは、元のワークフロー実行の実行グラフからコピーされています。元のワークフロー実行で実行され、グラフにコピーされたジョブおよびクローラーのノードには、実行の詳細が含まれます。

前の図のワークフロー W1 について、もう一度考えてみます。ワークフロー実行が J2 および J3 から再開されると、再開されたワークフロー実行の実行グラフには、すべてのジョブ (J1 ~ J5)、およびすべてのトリガー (T1 ~ T5) が表示されます。J1 の実行に関する詳細が、元のワークフロー実行からコピーされます。

ワークフロー実行のスナップショット

ワークフローの実行が開始されると、AWS Glue は、その時点でのワークフロー設計グラフのスナップショットを作成します。スナップショットの使用期間は、ワークフローの実行中です。実行の開始後にトリガーを変更した場合、その変更は、実行中のワークフローには影響しません。スナップショットを使用すると、ワークフローの実行を一貫した方法で行うことができます。

スナップショットでは、トリガーのみが不変になります。ワークフローの実行中に下流のジョブとクローラに加えた変更は、その時点の実行に対しても有効になります。

ワークフロー実行を再開する

ワークフローの実行を再開するには、以下の手順に従います。ワークフロー実行は、AWS Glue コンソール、API、または AWS Command Line Interface (AWS CLI) により再開させることができます。

ワークフロー実行を再開するには (コンソール)

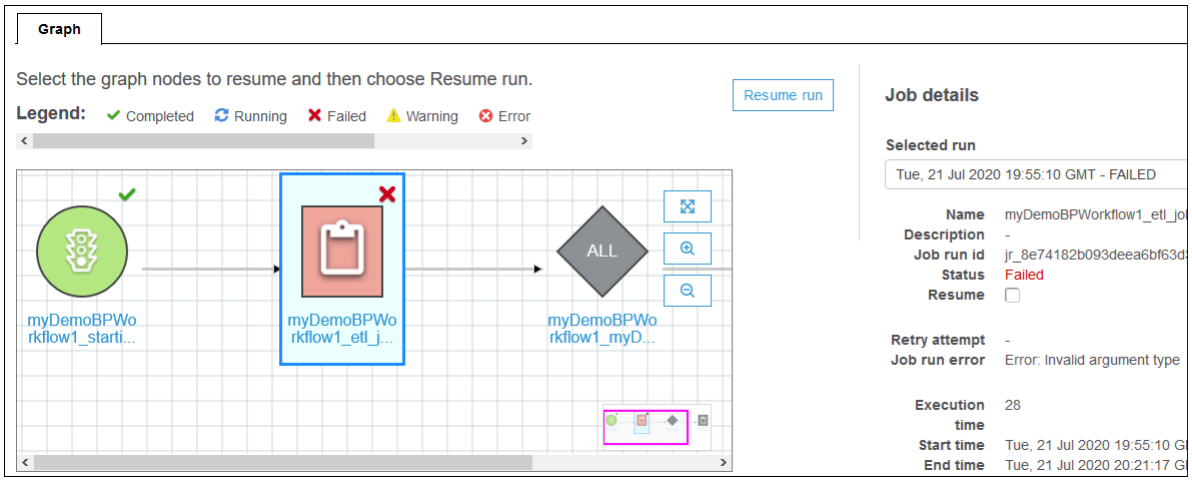
1. https://console.aws.amazon.com/glue/ で AWS Glue コンソール を開きます。

   ワークフローを表示し、その実行を再開するためのアクセス許可を持つユーザーとして、サインインします。

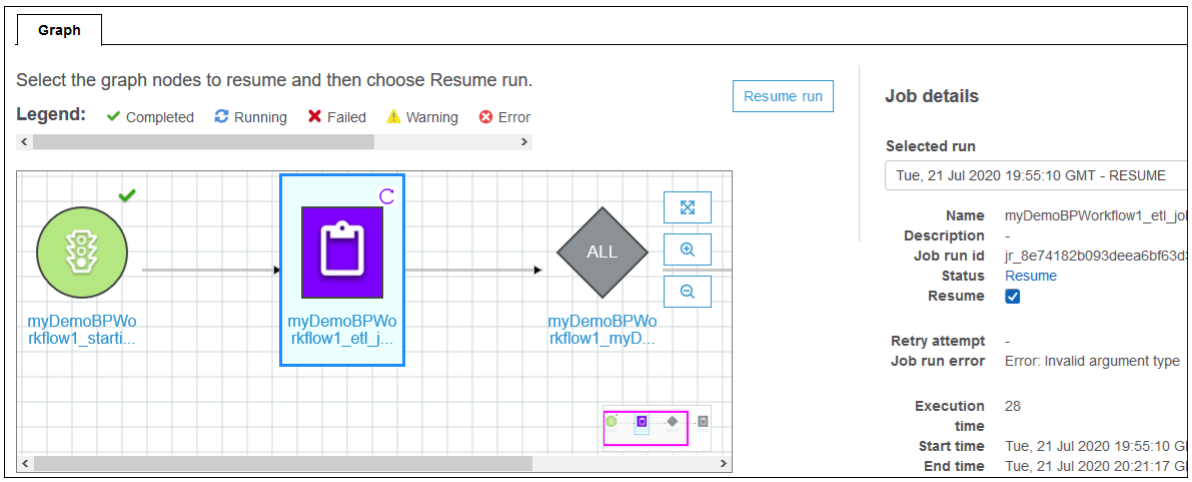
   > **Note**
   >
   > ワークフローの実行を再開するには、`glue:ResumeWorkflowRun` AWS Identity and Access Management(IAM) のアクセス許可が必要です。

2. ナビゲーションペインで、[Workflows] (ワークフロー) をクリックします。
3. ワークフローを選択し、[History] (履歴) タブを選択します。

4. 部分的にしか実行されていないワークフロー実行を選択し、[View run details] (実行の詳細を表示する) をクリックします。
5. 実行グラフで、再起動しワークフロー実行の再開ポイントとする、最初の (または唯一の) ノードを選択します。
6. グラフの右側にある詳細ペインで、[Resume] (再開) チェックボックスをオンにします。



ノードの色が変わり、右上に小さな [Resume] (再開) アイコンが表示されます。



7. 再起動する追加のノードについて、前述の 2 つのステップを完了します。
8. [Resume run] (実行を再開) をクリックします。

ワークフロー実行を再開するには (AWS CLI)

1. `glue:ResumeWorkflowRun` IAM アクセス許可が付与されていることを確認します。
2. 再起動するノードのノード ID を取得します。

a. 元のワークフロー実行のために、get-workflow-run コマンドを実行します。ワークフロー名と実行 ID を指定し、次の例に示すように、--include-graph オプションを追加します。コンソールの [History] (履歴) タブから実行 ID を取得します。もしくは get-workflow コマンドを実行し取得します。

```
aws glue get-workflow-run --name cloudtrailtest1 --run-id
 wr_a07e55f2087afdd415a404403f644a4265278f68b13ba3da08c71924ebe3c3a8 --include-
graph
```

このコマンドは、グラフのノードとエッジを大きな JSON オブジェクトとして返します。

b. ノードオブジェクトの Type および Name プロパティごとに、対象としているノードを見つけます。

次に、出力されるノードオブジェクトの例を示します。

```
{
    "Type": "JOB",
    "Name": "test1_post_failure_4592978",
    "UniqueId":
 "wnode_d1b2563c503078b153142ee76ce545fe5ceef66e053628a786ddd74a05da86fd",
    "JobDetails": {
        "JobRuns": [
            {
                "Id":
 "jr_690b9f7fc5cb399204bc542c6c956f39934496a5d665a42de891e5b01f59e613",
                "Attempt": 0,
                "TriggerName": "test1_aggregate_failure_649b2432",
                "JobName": "test1_post_failure_4592978",
                "StartedOn": 1595358275.375,
                "LastModifiedOn": 1595358298.785,
                "CompletedOn": 1595358298.785,
                "JobRunState": "FAILED",
                "PredecessorRuns": [],
                "AllocatedCapacity": 0,
                "ExecutionTime": 16,
                "Timeout": 2880,
                "MaxCapacity": 0.0625,
                "LogGroupName": "/aws-glue/python-jobs"
            }
        ]
    }
```

```
}
```

c. ノードオブジェクトの UniqueId プロパティから、ノード ID を取得します。

3. resume-workflow-run コマンドを実行します。次の例に示すように、ワークフロー名、実行 ID、およびノード ID のスペース区切りのリストを指定します。

```
aws glue resume-workflow-run --name cloudtrailtest1 --run-id
 wr_a07e55f2087afdd415a404403f644a4265278f68b13ba3da08c71924ebe3c3a8 --node-
ids wnode_ca1f63e918fb855e063aed2f42ec5762ccf71b80082ae2eb5daeb8052442f2f3
 wnode_d1b2563c503078b153142ee76ce545fe5ceef66e053628a786ddd74a05da86fd
```

このコマンドは、再開された (新しい) ワークフローの実行の ID と、開始されるノードのリストを出力します。

```
{
    "RunId": "wr_2ada0d3209a262fc1156e4291134b3bd643491bcfb0ceead30bd3e4efac24de9",
    "NodeIds": [
        "wnode_ca1f63e918fb855e063aed2f42ec5762ccf71b80082ae2eb5daeb8052442f2f3"
    ]
}
```

この例での resume-workflow-run コマンドでは再起動するノード 2 つをリストしていますが、出力例では 1 つのノードのみが再起動されると表示していることに注意してください。これは、1 つのノードが他のノードの下流にあり、いずれにせよ、この下流のノードはワークフローの通常のフローによって再開されるためです。

ワークフロー実行の再開に関する注意と制限事項

ワークフロー実行を再開する場合には、次の注意事項と制限事項に留意してください。

- 実行を再開できるのは、COMPLETED 状態のワークフローのみです。

> **Note**
>
> ワークフロー実行の中で 1 つ以上のノードが未完了の場合でも、ワークフロー実行の状態は COMPLETED となります。実行グラフをチェックして、正常に完了しなかったノードを見つける必要があります。

- 元のワークフロー実行により実行が試みられた任意のジョブまたはクローラのノードから、ワークフロー実行を再開することができます。トリガーノードからは、ワークフロー実行を再開することはできません。
- ノードを再起動しても、その状態はリセットされません。処理が部分的に完了しているデータのロールバックは行われません。
- 同じワークフロー実行を、複数回再開することが可能です。再開されたワークフローの実行が部分的にしか実行されない場合は、その問題に対処した上で実行を再開します。
- 再起動のために 2 つのノードを選択し、それらが互いに依存している場合には、上流にあるノードが下流ノードより先に実行されます。実際、下流にあるノードはワークフローの通常のフローによって実行されるため、このノードを選択することは冗長的となります。

AWS Glue でのワークフローの実行プロパティの取得と設定

ワークフローの実行プロパティを使用して、AWS Glue ワークフローのジョブ間の状態を共有して管理します。デフォルトの実行プロパティは、ワークフローの作成時に設定できます。次に、ジョブの実行時に、ジョブは実行プロパティの値を取得して必要に応じて変更し、ワークフローの以降のジョブに対する入力として使用できます。ジョブが実行プロパティを変更した場合、新しい値が有効なのはワークフローの実行中に限られます。デフォルトの実行プロパティは影響を受けません。

AWS Glue ジョブがワークフローに含まれていない場合、これらのプロパティは設定されません。

次の Python コード例は抽出、変換、ロード (ETL) ジョブの一部で、ワークフローの実行プロパティを取得する方法を示しています。

```
import sys
import boto3
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from awsglue.context import GlueContext
from pyspark.context import SparkContext

glue_client = boto3.client("glue")
args = getResolvedOptions(sys.argv, ['JOB_NAME','WORKFLOW_NAME', 'WORKFLOW_RUN_ID'])
workflow_name = args['WORKFLOW_NAME']
workflow_run_id = args['WORKFLOW_RUN_ID']
workflow_params = glue_client.get_workflow_run_properties(Name=workflow_name,
                                       RunId=workflow_run_id)["RunProperties"]

target_database = workflow_params['target_database']
```

```
target_s3_location = workflow_params['target_s3_location']
```

次のコードは、target_format 実行プロパティを 'csv' に設定することで続行します。

```
workflow_params['target_format'] = 'csv'
glue_client.put_workflow_run_properties(Name=workflow_name, RunId=workflow_run_id,
 RunProperties=workflow_params)
```

詳細については、次を参照してください:

- GetWorkflowRunProperties アクション (Python: get_workflow_run_properties)
- PutWorkflowRunProperties アクション (Python: put_workflow_run_properties)

AWS Glue APIを使用したワークフローのクエリ

AWS Glue には、ワークフローを管理するための高度な API が用意されています。AWS Glue API を使用してワークフローの静的ビューまたは実行中のワークフローの動的ビューを取得できます。詳細については、ワークフローを参照してください。

トピック

- 静的ビューのクエリ
- 動的ビューのクエリ

静的ビューのクエリ

GetWorkflow API オペレーションを使用して、ワークフローの設計を示す静的ビューを取得します。このオペレーションは、ノードおよびエッジで構成される有向グラフを返します。ノードは、トリガー、ジョブ、またはクローラを表します。エッジは、ノード間の関係を定義します。AWS Glue コンソールのグラフでは、エッジがコネクタ (矢印) で表されます。

このオペレーションは、NetworkX、igraph、JGraphT、Java Universal Network/Graph (JUNG) Framework などの一般的なグラフ処理ライブラリでも使用できます。これらのライブラリが表すグラフはすべてが類似しているため、必要な変換は最小限です。

この API から返される静的ビューは、ワークフローに関連付けられているトリガーの最新の定義に基づく最新のビューです。

グラフ定義

ワークフローのグラフ G は順序付きペア (N、E) で、N はノードのセットを表し、E はエッジのセットを表します。ノード はグラフ内の頂点で、一意の数値で識別されます。ノードのタイプは、トリガー、ジョブ、またはクローラです。例: `{name:T1, type:Trigger, uniqueId:1}`, `{name:J1, type:Job, uniqueId:2}`。

エッジ は、2 タプルの形式`(src, dest)` です。 `src` と `dest` はノードで、`src` から `dest`への有向エッジがあります。

静的ビューのクエリの例を示します。

ジョブ J1 の完了時にジョブ J2 をトリガーする条件付きのトリガー T を考えます。

```
J1 ---> T ---> J2
```

ノード: J1、T、J2

エッジ: (J1、T)、(T、J2)

動的ビューのクエリ

GetWorkflowRun API オペレーションを使用して、実行中のワークフローの動的ビューを取得します。このオペレーションは、グラフの同じ静的ビューを、ワークフローの実行に関連するメタデータと共に返します。

実行の際、GetWorkflowRun 呼び出しのジョブを表すノードでは、最新のワークフロー実行の一部として開始されたジョブ実行のリストを含んでいます。このリストを使用して、各ジョブの実行ステータスをグラフ自体に表示できます。まだ実行されていないダウンストリームの依存関係の場合、このフィールドは null に設定されます。グラフ化された情報により、任意の時点における任意のワークフローの現在の状態を確認できます。

この API から返される動的ビューは、ワークフローの実行が開始された時点における静的ビューに基づいています。

ランタイムノードの例: `{name:T1, type: Trigger, uniqueId:1}`、`{name:J1, type:Job, uniqueId:2, jobDetails:{jobRuns}}`、`{name:C1, type:Crawler, uniqueId:3, crawlerDetails:{crawls}}`

例 1: 動的ビュー

次の例は、単純な 2 トリガーワークフローを示しています。

- ノード: t1、j1、t2、j2
- エッジ: (t1, j1)、(j1, t2)、(t2, j2)

GetWorkflow レスポンスの内容は次のとおりです。

```
{
    Nodes : [
        {
            "type" : Trigger,
            "name" : "t1",
            "uniqueId" : 1
        },
        {
            "type" : Job,
            "name" : "j1",
            "uniqueId" : 2
        },
        {
            "type" : Trigger,
            "name" : "t2",
            "uniqueId" : 3
        },
        {
            "type" : Job,
            "name" : "j2",
            "uniqueId" : 4
        }
    ],
    Edges : [
        {
            "sourceId" : 1,
            "destinationId" : 2
        },
        {
            "sourceId" : 2,
            "destinationId" : 3
        },
        {
            "sourceId" : 3,
            "destinationId" : 4
        }
}
```

GetWorkflowRun レスポンスの内容は次のとおりです。

```
{
    Nodes : [
        {
            "type" : Trigger,
            "name" : "t1",
            "uniqueId" : 1,
            "jobDetails" : null,
            "crawlerDetails" : null
        },
        {
            "type" : Job,
            "name" : "j1",
            "uniqueId" : 2,
            "jobDetails" : [
                {
                    "id" : "jr_12334",
                    "jobRunState" : "SUCCEEDED",
                    "errorMessage" : "error string"
                }
            ],
            "crawlerDetails" : null
        },
        {
            "type" : Trigger,
            "name" : "t2",
            "uniqueId" : 3,
            "jobDetails" : null,
            "crawlerDetails" : null
        },
        {
            "type" : Job,
            "name" : "j2",
            "uniqueId" : 4,
            "jobDetails" : [
                {
                    "id" : "jr_1233sdf4",
                    "jobRunState" : "SUCCEEDED",
                    "errorMessage" : "error string"
                }
            ],
            "crawlerDetails" : null
        }
```

```
    ],
    Edges : [
        {
            "sourceId" : 1,
            "destinationId" : 2
        },
        {
            "sourceId" : 2,
            "destinationId" : 3
        },
        {
            "sourceId" : 3,
            "destinationId" : 4
        }
}
```

例 2: 複数のジョブと 1 つの条件付きのトリガー

次の例は、複数のジョブと 1 つの条件付きのトリガー (t3) を持つワークフローを示しています。

```
Consider Flow:
T(t1) ---> J(j1) ---> T(t2) ---> J(j2)
             |                     |
             |                     |
             >+------> T(t3) <-----+
                        |
                        |
                     J(j3)

Graph generated:
Nodes: t1, t2, t3, j1, j2, j3
Edges: (t1, j1), (j1, t2), (t2, j2), (j1, t3), (j2, t3), (t3, j3)
```

AWS Glue での設計図とワークフローの制限

設計図とワークフローについての制限を次に示します。

設計図での制限

設計図については、以下の制限に注意してください。

- 設計図は、Amazon S3 バケットが置かれているのと同じ AWS リージョンに登録される必要があります。

- 設計図を AWS アカウント間で共有する場合は、Amazon S3 内で設計図の ZIP アーカイブに対する読み取りを許可する必要があります。設計図の ZIP アーカイブに対する読み取り許可を持つユーザーは、自分の AWS アカウントに設計図を登録し、それを使用することができます。
- 設計図のパラメータセットは、単一の JSON オブジェクトとして格納されます。このオブジェクトの最大長は 128 KB です。
- 設計図の ZIP アーカイブにおける非圧縮状態での最大サイズは 5 MB です。圧縮状態での最大サイズは 1 MB です。
- ワークフロー内のジョブ、クローラ、トリガーの総数を、100 以下に制限します。100 を超える値を含めると、ワークフローの実行を再開または停止しようとした場合にエラーが発生することがあります。

ワークフローの制限

ワークフローについては、以下の制限に注意してください。これらのコメントの中には、ワークフローを手動で作成しているユーザーに直接関係するものがあります。

- Amazon EventBridge のイベントトリガーの最大バッチサイズは 100 です。最大ウィンドウサイズは 900 秒 (15 分) です。
- トリガーは 1 つのワークフローにのみ関連付けることができます。
- 1 つの開始トリガー (オンデマンドまたはスケジュール) のみが許可されます。
- ワークフロー内のジョブまたはクローラがワークフロー外のトリガーによって開始された場合、ジョブまたはクローラの完了 (成功またはその他) に依存するワークフロー内のトリガーは起動されません。
- 同様に、ワークフロー内のジョブまたはクローラに、ワークフロー内とワークフロー外の両方でジョブまたはクローラの完了 (成功またはそれ以外) に依存するトリガーがあり、さらにジョブまたはクローラがワークフロー内から開始されている場合には、ジョブまたはクローラの完了時、ワークフロー内のトリガーのみが起動されます。

AWS Glue での設計図エラーをトラブルシューティングする

AWS Glue の設計図を使用する際にエラーが発生した場合は、次の解決策を参照しながら問題の原因を突き止め、それを修正してください。

トピック

- エラー: PySpark モジュールが見つからない

- エラー: 設計図の設定ファイルが見つからない
- エラー:インポートされたファイルが見つからない
- エラー:リソースでの iamPassRole の実行が承認されない
- エラー: cron スケジュールが無効
- エラー: 既存のものと同じ名前を使用してのトリガー
- エラー:名前:foo という名前のワークフローが既に存在する
- エラー: 指定された layoutGenerator のパスにモジュールが見つからない
- エラー: Connections フィールドの検証エラー

エラー: PySpark モジュールが見つからない

AWS Glue が、「Unknown error executing layout generator function ModuleNotFoundError: No module named 'pyspark'」というエラーを返します。

設計図アーカイブを解凍すると、以下のいずれかのようになります。

```
$ unzip compaction.zip
Archive:  compaction.zip
   creating: compaction/
  inflating: compaction/blueprint.cfg
  inflating: compaction/layout.py
  inflating: compaction/README.md
  inflating: compaction/compaction.py

$ unzip compaction.zip
Archive:  compaction.zip
  inflating: blueprint.cfg
  inflating: compaction.py
  inflating: layout.py
  inflating: README.md
```

1 つ目のケースでは、設計図に関連するすべてのファイルが compaction という名前のフォルダの下に配置され、その後で compaction.zip という名前の zip ファイルに変換されています。

2 つ目のケースでは、設計図に必要なすべてのファイルをフォルダに配置せずに、zip ファイル compaction.zip に対しルートファイルとして追加しています。

上記の形式のどちらを使用しても、ファイルを作成することができます。ただし、`blueprint.cfg` の中には、レイアウトを生成するスクリプト内の関数名を指す正しいパスが必要です。

例

ケース 1 の場合、blueprint.cfg には以下のように layoutGenerator が必要です

```
layoutGenerator": "compaction.layout.generate_layout"
```

ケース 2 の場合、blueprint.cfg には以下のように layoutGenerator が必要です

```
layoutGenerator": "layout.generate_layout"
```

このパスが正しく指定されていない場合は、次のようなエラーが表示されることがあります。例えば、フォルダをケース 2 で説明した構造としながら、ケース 1 で示した layoutGenerator が使用されている場合、上記のような エラーが発生する場合があります。

エラー: 設計図の設定ファイルが見つからない

AWS Glue が、「Unknown error executing layout generator function FileNotFoundError: [Errno 2] No such file or directory: '/tmp/compaction/blueprint.cfg」というエラーを返します。

blueprint.cfg ファイルは、ZIP アーカイブのルートレベルか、ZIP アーカイブと同じ名前のフォルダ内に配置する必要があります。

設計図の ZIP アーカイブを抽出すると、通常、blueprint.cfg は、下記のいずれかのパスで見つかります。以下のパスのいずれかに配置されていない場合には、上記のエラーが表示されます。

```
$ unzip compaction.zip
Archive:  compaction.zip
   creating: compaction/
  inflating: compaction/blueprint.cfg

$ unzip compaction.zip
Archive:  compaction.zip
  inflating: blueprint.cfg
```

エラー:インポートされたファイルが見つからない

AWS Glue が、「Unknown error executing layout generator function FileNotFoundError: [Errno 2] No such file or directory:* *'demo-project/foo.py」というエラーを返します。

レイアウト生成スクリプトに他のファイルを読み込む機能がある場合は、インポートするファイルの完全なパスを指定する必要があります。例えば、Conversion.py スクリプトは Layout.py から参照されることがあります。詳細については、「Sample Blueprint Project」を参照してください。

エラー:リソースでの iamPassRole の実行が承認されない

AWS Glue が、「User: arn:aws:sts::123456789012:assumed-role/AWSGlueServiceRole/GlueSession is not authorized to perform: iam:PassRole on resource: arn:aws:iam::123456789012:role/AWSGlueServiceRole」というエラーを返します。

ワークフロー内のジョブとクローラが、設計図からワークフローを作成するために渡されたロールと同じロールを引き受ける場合、設計図のロールには、それ自体に対する `iam:PassRole` アクセス許可を含める必要があります。

ワークフロー内のジョブとクローラが、設計図からワークフローのエンティティを作成するために渡されたロール以外のロールを引き受ける場合、設計図のロールには、それ自体のロールではなく引き受けたロールに対する、`iam:PassRole` アクセス許可を含める必要があります。

詳細については、「Permissions for Blueprint Roles」を参照してください。

エラー: cron スケジュールが無効

AWS Glue が、「The schedule cron(0 0 * * * *) is invalid.」というエラーを返します。

有効な cron 式を指定します。詳細については、「Time-Based Schedules for Jobs and Crawlers」を参照してください。

エラー: 既存のものと同じ名前を使用してのトリガー

AWS Glue が、「Trigger with name 'foo_starting_trigger' already submitted with different configuration」というエラーを返します。

設計図でワークフローを作成するために、レイアウトスクリプトの中でトリガーを定義する必要はありません。トリガーの作成は、2 つのアクション間に定義された依存関係に基づいて、設計図ライブラリによって管理されます。

トリガーの名前は、次のように決定されます。

- ワークフローの開始トリガーの場合は、<workflow_name>_starting_trigger のような名前になります。

- ワークフロー内のノード (ジョブ/クローラ) で、1 つまたは複数の上流ノードの完了に依存している場合、AWS Glue はトリガーを、<workflow_name>_<node_name>_trigger のような名前で定義します

このエラーは、既に存在するものと同じ名前で、トリガーが発生したことを意味します。既存のトリガーを削除すれば、ワークフローの作成を再実行できます。

> **Note**
>
> ワークフローを削除しても、ワークフロー内のノードは削除されません。ワークフローが削除されても、トリガーが残されている場合があります。このため、作成したワークフローを削除した後に、同じ名前で同じ設計図から再作成しようとすると、ワークフローが既に存在していることを知らせるエラーは表示されずに、トリガーが既に存在している、というエラーが出力されることがあります。

エラー:名前:foo という名前のワークフローが既に存在する

ワークフロー名は一意である必要があります。別の名前で再試行してください。

エラー: 指定された layoutGenerator のパスにモジュールが見つからない

AWS Glue が、「Unknown error executing layout generator function ModuleNotFoundError: No module named 'crawl_s3_locations'」というエラーを返します。

```
layoutGenerator": "crawl_s3_locations.layout.generate_layout"
```

例えば、上記の LayoutGenerator パスを指定しながら設計図アーカイブを解凍する際には、次のようにする必要があります。

```
$ unzip crawl_s3_locations.zip
Archive:  crawl_s3_locations.zip
   creating: crawl_s3_locations/
  inflating: crawl_s3_locations/blueprint.cfg
  inflating: crawl_s3_locations/layout.py
  inflating: crawl_s3_locations/README.md
```

アーカイブの解凍時に、設計図アーカイブが次のようになっている場合は、上記のエラーが発生することがあります。

```
$ unzip crawl_s3_locations.zip
Archive:  crawl_s3_locations.zip
  inflating: blueprint.cfg
  inflating: layout.py
  inflating: README.md
```

`crawl_s3_locations` という名前のフォルダがなく、`layoutGenerator` パスがモジュール `crawl_s3_locations` を介してレイアウトファイルを参照している場合に、上記のエラーが発生する可能性があります。

エラー: Connections フィールドの検証エラー

AWS Glue が、「Unknown error executing layout generator function TypeError: Value ['foo'] for key Connections should be of type <class 'dict'>!」というエラーを返します。

これは検証に関するエラーです。`Job` クラス内の `Connections` フィールドはディクショナリの使用を想定しており、代わりに値のリストが提供された場合には、エラーの原因となります。

```
User input was list of values
Connections= ['string']

Should be a dict like the following
Connections*=*{'Connections': ['string']}
```

設計図からワークフローを作成する際に発生する、これらの実行時エラーを回避するには、「Testing a Blueprint」に概略されているとおりに、ワークフロー、ジョブ、およびクローラの定義を検証します。

レイアウトスクリプト内での AWS Glue ジョブ、クローラ、ワークフローの定義に関しては、「AWS Glue Blueprint Classes Reference」にある構文を参照してください。

AWS Glue ブループリントでのペルソナおよびロール用のアクセス許可

以下は、AWS Identity and Access Management ブループリントのための典型的なペルソナと、ペルソナおよびロール用に提案されている AWS Glue(IAM) アクセス許可ポリシーです。

トピック

- 設計図のペルソナ
- 設計図のペルソナ用のアクセス許可
- 設計図のロール用のアクセス許可

設計図のペルソナ

以下は、一般的に、AWS Glue ブループリントのライフサイクルに関与するペルソナです

ペルソナ	説明
AWS Glue 開発者	設計図を開発、テスト、公開します。
AWS Glue 管理者	設計図を登録および保守し、アクセス許可を付与します。
データアナリスト	設計図を実行してワークフローを作成します。

(詳細については、the section called “設計図の概要” を参照してください)。

設計図のペルソナ用のアクセス許可

設計図のペルソナごとに、提案されているアクセス許可を以下に示します。

ブループリントのための AWS Glue デベロッパー権限

AWS Glue デベロッパーには設計図の公開に使用される Amazon S3 バケットに対する、書き込みのアクセス許可が必要です。多くの場合、デベロッパーはアップロードした後に設計図を登録します。その場合、デベロッパーには the section called “ブループリントのための AWS Glue 管理者のアクセス許可” に一覧表示されているアクセス許可が必要です。さらに、デベロッパーが登録後に設計図をテストする場合は、the section called “設計図のためのデータアナリストの権限” に一覧表示されているアクセス許可も必要となります。

ブループリントのための AWS Glue 管理者のアクセス許可

次のポリシーでは、AWS Glue ブループリントを登録、表示、保守するためのアクセス許可が付与されます。

> **Important**
>
> 次のポリシーでは、*<s3-bucket-name>* および *<prefix>* を、登録のためにアップロードされた設計図の ZIP アーカイブを指す Amazon S3 パスに置き換えます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:CreateBlueprint",
                "glue:UpdateBlueprint",
                "glue:DeleteBlueprint",
                "glue:GetBlueprint",
                "glue:ListBlueprints",
                "glue:BatchGetBlueprints"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": "arn:aws:s3:::<s3-bucket-name>/<prefix>/*"
        }
    ]
}
```

設計図のためのデータアナリストの権限

次のポリシーは、設計図を実行し、結果のワークフローとワークフローコンポーネントを表示するアクセス許可を付与します。また、ワークフローおよびワークフローコンポーネントを作成するために PassRole が引き受ける、AWS Glue というロールを付与ます。

このポリシーでは、任意のリソースに対するアクセス許可が付与されます。個々の設計図へのきめ細かなアクセスを構成する場合は、設計図の ARN に次の形式を使用します。

```
arn:aws:glue:<region>:<account-id>:blueprint/<blueprint-name>
```

> **Important**
>
> 次のポリシーでは、*<account-id>* を有効な AWS アカウントに置き換え、*<role-name>* を設計図の実行に使用するロールの名前に置き換えます。このロールに必要なアクセス許

可については、「the section called “設計図のロール用のアクセス許可”」を参照してください。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:ListBlueprints",
                "glue:GetBlueprint",
                "glue:StartBlueprintRun",
                "glue:GetBlueprintRun",
                "glue:GetBlueprintRuns",
                "glue:GetCrawler",
                "glue:ListTriggers",
                "glue:ListJobs",
                "glue:BatchGetCrawlers",
                "glue:GetTrigger",
                "glue:BatchGetWorkflows",
                "glue:BatchGetTriggers",
                "glue:BatchGetJobs",
                "glue:BatchGetBlueprints",
                "glue:GetWorkflowRun",
                "glue:GetWorkflowRuns",
                "glue:ListCrawlers",
                "glue:ListWorkflows",
                "glue:GetJob",
                "glue:GetWorkflow",
                "glue:StartWorkflowRun"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "arn:aws:iam::<account-id>:role/<role-name>"
        }
    ]
}
```

設計図のロール用のアクセス許可

設計図からワークフローを作成するために、IAM ロール用として提案されているアクセス許可を次に示します。このロールには glue.amazonaws.com との信頼関係が必要です。

> **Important**
>
> 次のポリシーでは、*<account-id>* を有効な AWS アカウントに置き換え、*<role-name>* をロールの名前に置き換えます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:CreateJob",
                "glue:GetCrawler",
                "glue:GetTrigger",
                "glue:DeleteCrawler",
                "glue:CreateTrigger",
                "glue:DeleteTrigger",
                "glue:DeleteJob",
                "glue:CreateWorkflow",
                "glue:DeleteWorkflow",
                "glue:GetJob",
                "glue:GetWorkflow",
                "glue:CreateCrawler"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "arn:aws:iam::<account-id>:role/<role-name>"
        }
    ]
}
```

> **Note**
>
> ワークフロー内のジョブとクローラが、このロール以外のロールを引き受ける場合、このポリシーには、設計図のロールのものではなく引き受けたロールでの `iam:PassRole` アクセス許可を付与する必要があります。

AWS Glue のブループリントの開発

組織が、類似した一連の ETL ユースケースを実行する場合、それらすべてをパラメータ化した単一のワークフローで処理することでメリットが得られます。このようなニーズに応えるため、AWS Glue では、設計図を定義して、ワークフローの生成に使用できるようになってます。設計図にはパラメータを渡すことができるので、データアナリストは単一の設計図からさまざまなワークフローを作成して、類似の ETL ユースケースを処理できます。作成した設計図は、別の部門、チーム、プロジェクトでの再利用が可能です。

トピック

- AWS Glue のブループリント概要
- AWS Glue のブループリントの開発
- AWS Glue でブループリントを登録する
- AWS Glue でブループリントを表示する
- AWS Glue でブループリントを更新する
- AWS Glue で設計図からワークフローを作成する
- AWS Glue でブループリントの実行を表示する

AWS Glue のブループリント概要

> **Note**
>
> ブループリント機能は、現在、AWS Glue コンソールの次のリージョンではご利用いただけません。アジアパシフィック (ジャカルタ)、中東 (UAE)。

AWS Glue ブループリントにより、AWS Glue ワークフローを作成ならびに共有する手段が提供されます。複雑な ETL プロセスで、使用できるユースケースが類似している場合には、ユースケースごとに AWS Glue ワークフローを作成する代わりにブループリントを 1 つ作成します。

設計図では、ワークフローに含めるジョブとクローラを指定し、ワークフローを作成するために設計図が実行される際にワークフローのユーザーから提供すべきパラメータも指定します。パラメータを使用することで、単一のブループリントにより、類似したさまざまなユースケースのワークフローを生成できるようになります。ワークフローの詳細については、「AWS Glue のワークフローの概要」を参照してください。

設計図に関するユースケースの例を次に示します。

- 既存のデータセットを分割する場合。Amazon Simple Storage Service (Amazon S3) のソースならびにターゲットへのパス、およびパーティション列のリストが、設計図への入力パラメータとなります。
- Amazon DynamoDB テーブルのスナップショットを、Amazon Redshift のような SQL データストア内に作成する場合。ブループリントへの入力パラメータは、DynamoDB テーブル名と AWS Glue の接続となります。このパラメータにより、Amazon Redshift クラスターと送信先のデータベースを指定します。
- 複数の Amazon S3 パスの CSV データを Parquet に変換する場合。AWS Glue ワークフローに、パスごとの個別のクローラとジョブを含める必要があります。AWS Glue Data Catalog 内の送信先データベースと、Amazon S3 パスのコンマ区切りのリストが入力パラメーターとなります。この場合、ワークフローが作成するクローラおよびジョブの数は可変であることに注意してください。

設計図のコンポーネント

設計図は、以下のコンポーネントを含む ZIP アーカイブです。

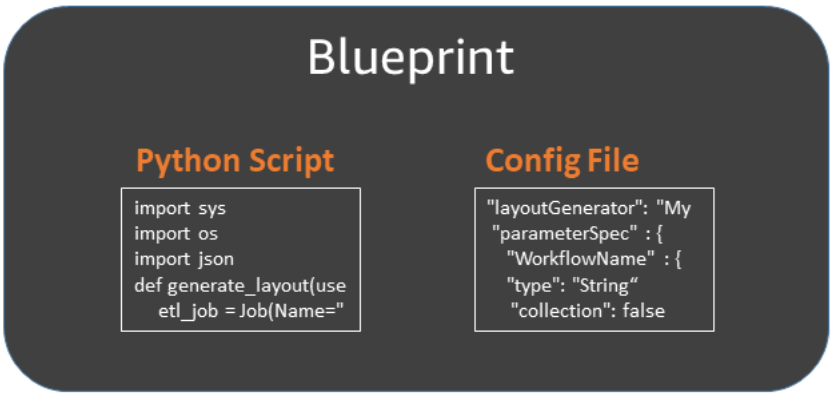
- Python レイアウトジェネレータスクリプト

  ワークフローのレイアウト (ワークフロー用に作成するジョブとクローラ、ジョブとクローラのプロパティ、およびジョブとクローラの依存関係) を指定する関数が含まれています。この関数はブループリントのパラメータを受け取り、AWS Glue がワークフロー生成のために使用するワークフロー構造 (JSON オブジェクト) を返します。ワークフローの生成には Python スクリプトを使用するので、ユースケースに適した独自のロジックを追加できます。
- 設定ファイル

ワークフローのレイアウトを生成する Python 関数の完全な修飾名を指定します。スクリプトで使用されるすべての設計図パラメータの名前、データタイプ、およびその他のプロパティも指定します。

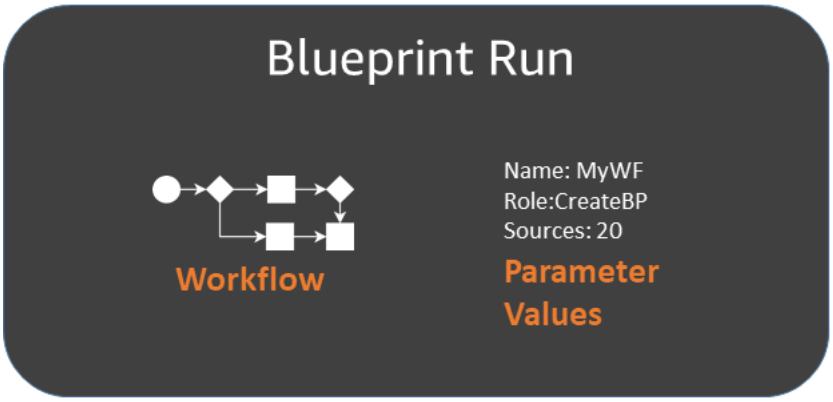
- (オプション) ETL スクリプトとサポートファイル

  高度なユースケースとして、ジョブで使用する ETL スクリプトの場所をパラメータ化できます。ZIP アーカイブにジョブスクリプトファイルを含め、スクリプトのコピー先となる Amazon S3 ロケーションを設計図パラメータとして指定できます。レイアウトジェネレータスクリプトは、指定された場所に ETL スクリプトをコピーし、そのコピー先をジョブスクリプトの場所プロパティとして指定します。スクリプトで処理するための、ライブラリやその他のサポートファイルを含めることもできます。



設計図の実行

ブループリントからワークフローを作成すると、AWS Glue でブループリントが実行され、ワークフローがカプセル化するジョブ、クローラー、およびトリガーを作成する非同期プロセスが開始します。AWS Glue でブループリントの実行を使用して、ワークフローとそのコンポーネントの作成をオーケストレートします。設計図の実行ステータスを表示すると、作成プロセスのステータスを知ることができます。設計図の実行では、設計図パラメータに指定した値の保存も行われます。



ブループリントの実行は、AWS Glue コンソールまたは AWS Command Line Interface (AWS CLI) を使用して表示できます。ワークフローを表示またはトラブルシューティングする場合には、いつでも設計図の実行に戻り、ワークフローの作成に使用された設計図のパラメータ値を確認できます。

設計図のライフサイクル

ブループリントは、AWS Glue により開発、テスト、登録が行われ、これを実行することでワークフローが作成されます。設計図のライフサイクルには、通常 3 つのペルソナが関与します。

ペルソナ	タスク
AWS Glue デベロッパー	• ワークフローレイアウトのスクリプトを記述し、設定ファイルを作成します。 • AWS Glue によって提供されるライブラリを使用して、ブループリントをローカルでテストします。 • スクリプト、設定ファイル、およびサポートファイルを含む ZIP アーカイブを作成し、Amazon S3 内の口ケーションでそのアーカイブを公開します。 • バケットオブジェクトに対する読み取りアクセス許可を、AWS Glue 管理者の AWS アカウントに付与するバケットポリシーを、Amazon S3 バケットに追加します。 • Amazon S3 にある ZIP アーカイブに対する、読み取りのための IAM アクセス許可を、AWS Glue 管理者に付与します。

ペルソナ	タスク
AWS Glue 管理者	• AWS Glue にブループリントを登録します。AWS Glue で予約した Amazon S3 の場所に、ZIP アーカイブのコピーを作成します。 • 設計図に対する IAM アクセス許可をデータアナリストに付与します。
データアナリスト	• 設計図を実行したワークフローの作成と、設計図パラメータ値の指定を行います。設計図の実行ステータスをチェックして、ワークフローとワークフローコンポーネントが正常に生成されたことを確認します。 • ワークフローを実行し、トラブルシューティングします。ワークフローを実行する前に、ワークフローのデザイングラフを AWS Glue コンソールに表示し、そのワークフローを検証します。

以下も参照してください。

- AWS Glue のブループリントの開発
- AWS Glue で設計図からワークフローを作成する
- AWS Glue ブループリントでのペルソナおよびロール用のアクセス許可

AWS Glue のブループリントの開発

AWS Glue デベロッパーは、データアナリストがワークフローを生成するための設計図を作成および公開できます。

トピック

- 設計図の開発の概要
- 設計図の開発のための前提条件
- 設計図コードを記述する
- 設計図プロジェクト例

- 設計図をテストする
- 設計図を公開する
- AWS Glue ブループリントクラスリファレンス
- 設計図の例

以下も参照してください。

- AWS Glue のブループリント概要

設計図の開発の概要

開発プロセスの最初のステップは、設計図からのメリットが活かせるような、共通点のあるユースケースを特定することです。典型的なユースケースとしては、一般化した方法での解決が想定される、反復的な ETL の問題が考えられます。次に、その一般化されたユースケースを実装するために設計図を構成します。一般化されたユースケースを使用して特定のユースケースを定義できるように、設計図の入力パラメータを定義します。

設計図は、その設計図のパラメータ構成ファイルを含むプロジェクトと、生成するワークフローのレイアウトを定義するスクリプトにより構成されます。レイアウトは、作成するジョブとクローラ (または設計図の用語におけるエンティティ) を定義します。

レイアウトスクリプトでは、トリガーを直接指定しません。その代わり、スクリプトが作成するジョブやクローラー間の依存関係を指定するコードを書き込みます。AWS Glue は依存関係の指定に基づき、トリガーを生成します。レイアウトスクリプトからは、すべてのワークフローエンティティの仕様を含む、ワークフローオブジェクトが出力されます。

ワークフローオブジェクトの構築には、以下に示す AWS Glue のブループリントライブラリを使用します。

- `awsglue.blueprint.base_resource` – ライブラリで使用される基本リソースのライブラリ。
- `awsglue.blueprint.workflow` – `Workflow` クラスを定義するためのライブラリ。
- `awsglue.blueprint.job` – `Job` クラスを定義するためのライブラリ。
- `awsglue.blueprint.crawler` – `Crawler` クラスを定義するためのライブラリ。

上記の他には、Python シェルで使用できるライブラリのみが、レイアウト生成用のライブラリとしてサポートされています。

設計図ライブラリで定義されたメソッドを使用することで、公開前の設計図をローカルでテストできます。

設計図をデータアナリストに提供する準備ができたら、スクリプト、パラメータ設定ファイル、および追加のスクリプトやライブラリなどのサポートファイルを、単一のデプロイ可能なアセットの中にパッケージ化します。次に、このアセットを Amazon S3 にアップロードし、AWS Glue への登録を管理者に依頼します。

その他の設計図プロジェクトの例については、「設計図プロジェクト例」および「設計図の例」を参照してください。

設計図の開発のための前提条件

ブループリントを開発するには、AWS Glue と Apache Spark ETL ジョブや Python シェルジョブのスクリプト記述に関する知識が必要です。また、以下のセットアップタスクも完了する必要があります。

- 設計図レイアウトスクリプトで使用する 4 個の AWS Python ライブラリのダウンロード
- AWS SDK のセットアップ。
- AWS CLI のセットアップ。

Python ライブラリをダウンロードする

以下のライブラリを GitHub からダウンロードし、自分のプロジェクトにインストールします。

- https://github.com/awslabs/aws-glue-blueprint-libs/tree/master/awsglue/blueprint/base_resource.py
- https://github.com/awslabs/aws-glue-blueprint-libs/tree/master/awsglue/blueprint/workflow.py
- https://github.com/awslabs/aws-glue-blueprint-libs/tree/master/awsglue/blueprint/crawler.py
- https://github.com/awslabs/aws-glue-blueprint-libs/tree/master/awsglue/blueprint/job.py

AWS Java SDK のセットアップ

AWS Java SDK に対しては、設計図の API を含む `jar` ファイルを追加する必要があります。

1. まだ追加していない場合には、AWS SDK for Java のセットアップを行います。

- Java 1.x の場合は、「AWS SDK for Java デベロッパーガイド」の「AWS SDK for Java の入手方法」にある手順に従ってください。
- Java 2.x の場合は、「AWS SDK for Java 2.x デベロッパーガイド」の「AWS SDK for Java 2.x のセットアップ」にある手順に従ってください。

2. 設計図の API へのアクセス許可が付与されている、クライアント jar ファイルをダウンロードします。

   - Java 1.x 用ファイル: s3://awsglue-custom-blueprints-preview-artifacts/awsglue-java-sdk-preview/AWSGlueJavaClient-1.11.x.jar
   - Java 2.x 用ファイル: s3://awsglue-custom-blueprints-preview-artifacts/awsglue-java-sdk-v2-preview/AwsJavaSdk-Glue-2.0.jar

3. Java クラスパスの先頭にクライアント jar を追加して、AWS Java SDK で提供された AWS Glue クライアントをオーバーライドします。

```
export CLASSPATH=<path-to-preview-client-jar>:$CLASSPATH
```

4. (オプション) 次の Java アプリケーションを使用して SDK をテストします。このアプリケーションは空のリストを出力するように設定されています。

   accessKey および secretKey は自分の認証情報に置き換え、us-east-1 は使用しているリージョンに置き換えます。

```
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSCredentialsProvider;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.services.glue.AWSGlue;
import com.amazonaws.services.glue.AWSGlueClientBuilder;
import com.amazonaws.services.glue.model.ListBlueprintsRequest;

public class App{
    public static void main(String[] args) {
        AWSCredentials credentials = new BasicAWSCredentials("accessKey",
 "secretKey");
        AWSCredentialsProvider provider = new
 AWSStaticCredentialsProvider(credentials);
        AWSGlue glue = AWSGlueClientBuilder.standard().withCredentials(provider)
                .withRegion("us-east-1").build();
```

```
        ListBlueprintsRequest request = new
 ListBlueprintsRequest().withMaxResults(2);
        System.out.println(glue.listBlueprints(request));
    }
}
```

AWS Python SDK をセットアップする

次の手順では、Python バージョン 2.7 以降、またはバージョン 3.9 以降が、コンピュータ上にインストールされていることを前提としています。

1. 次の boto3 wheel ファイルをダウンロードします。ファイルを開くか保存することを促された場合は、次のファイルを保存します。s3://awsglue-custom-blueprints-preview-artifacts/aws-python-sdk-preview/boto3-1.17.31-py2.py3-none-any.whl
2. 次の botocore wheel ファイルをダウンロードします。s3://awsglue-custom-blueprints-preview-artifacts/aws-python-sdk-preview/botocore-1.20.31-py2.py3-none-any.whl
3. 使用している Python のバージョンを確認します。

   ```
   python --version
   ```

4. Python のバージョンに応じて、以下のコマンドを入力します (Linux の場合)。

   - Python 2.7 以降の場合。

     ```
     python3 -m pip install --user virtualenv
     source env/bin/activate
     ```

   - Python 3.9 以降の場合。

     ```
     python3 -m venv python-sdk-test
     source python-sdk-test/bin/activate
     ```

5. botocore wheel ファイルをインストールします。

   ```
   python3 -m pip install <download-directory>/botocore-1.20.31-py2.py3-none-any.whl
   ```

6. boto3 wheel ファイルをインストールします。

   ```
   python3 -m pip install <download-directory>/boto3-1.17.31-py2.py3-none-any.whl
   ```

7. ~/.aws/credentials および ~/.aws/config ファイル内で、認証情報およびデフォルトのリージョンを指定します。詳細については、「AWS CLI ユーザーガイド」の「AWS Command Line Interface の設定。」を参照してください。
8. (オプション) セットアップをテストします。次のコマンドは、空のリストを返すように設定されています。

   us-east-1 は、実際のリージョンに置き換えます。

   ```
   $ python
   >>> import boto3
   >>> glue = boto3.client('glue', 'us-east-1')
   >>> glue.list_blueprints()
   ```

プレビュー AWS CLI をセットアップする

1. まだの場合は、コンピュータ上で AWS Command Line Interface (AWS CLI) のインストールおよび (または)更新を行ってください。この作業は、Python インストーラユーティリティの pip を使用することで、最も簡単に完了できます。

   ```
   pip install awscli --upgrade --user
   ```

   AWS CLI を使用したインストールに関する完全な手順は、「nstalling the AWS Command Line Interface」でご確認いただけます。
2. 次の AWS CLI wheel ファイルをダウンロードします。s3://awsglue-custom-blueprints-preview-artifacts/awscli-preview-build/awscli-1.19.31-py2.py3-none-any.whl
3. AWS CLI wheel ファイルをインストールする。

   ```
   python3 -m pip install awscli-1.19.31-py2.py3-none-any.whl
   ```
4. aws configure コマンドを実行します。AWS 認証情報 (アクセスキー、シークレットキーを含む) と AWS リージョンを設定します。AWS CLI の設定に関する情報は、「Configuring the AWS CLI」でご確認ください。
5. AWS CLI をテストします。次のコマンドは、空のリストを返すように設定されています。

   us-east-1 は、実際のリージョンに置き換えます。

   ```
   aws glue list-blueprints --region us-east-1
   ```

設計図コードを記述する

作成する各設計図プロジェクトには、少なくとも以下のファイルを含む必要があります。

- ワークフローを定義する Python レイアウトスクリプト。このスクリプトには、ワークフロー内のエンティティ (ジョブとクローラ)、および、それらの間の依存関係を定義する関数が含まれています。
- 以下を定義する設定ファイル `blueprint.cfg`。
  - ワークフローレイアウト定義関数を指す完全なパス。
  - 設計図が受け取るパラメータ。

トピック

- 設計図レイアウトスクリプトを作成する
- 設定ファイルを作成する
- 設計図パラメータを指定する

設計図レイアウトスクリプトを作成する

設計図のレイアウトスクリプトには、ワークフロー内のエンティティを生成するための関数を含む必要があります。この関数には好きな名前を付けることができます。AWS Glue は構成ファイルを使用して、関数の完全修飾名を決定します。

レイアウト関数は以下を実行します。

- (オプション) `Job` オブジェクトを作成する `Job` クラスをインスタンス化し、`Command` および `Role` の引数を渡します。これらは、AWS Glue コンソールまたは API を使用しながらジョブを作成する際に指定する、ジョブのプロパティです。
- (オプション) `Crawler` オブジェクトを作成する `Crawler` クラスをインスタンス化し、名前、ロール、ターゲット引数を渡します。
- オブジェクト (ワークフローエンティティ) 間の依存関係を示すために、追加の引数 `DependsOn` および `WaitForDependencies` を `Job()` および `Crawler()` に渡します。これらの引数については、このセクションの後半で説明します。
- `Workflow` に返すためのワークフローオブジェクトを作成する AWS Glue クラスをインスタンス化し、`Name` 引数 、`Entities` 引数 、およびオプションの `OnSchedule` 引数を渡します。引数 `Entities` では、ワークフローに含めるすべてのジョブとクローラを指定します。`Entities` オ

ブジェクトを構築する方法については、このセクションで後述するプロジェクト例をご覧ください。

- Workflow オブジェクトを返します。

Job、Crawler、および Workflow クラスの定義については、「AWS Glue ブループリントクラスリファレンス」を参照してください。

レイアウト関数は、以下の引数を受け取る必要があります。

引数	説明
user_params	設計図パラメータの名前と値に関する Python ディクショナリ。詳細については、設計図パラメータを指定する を参照してください。
system_params	2 つのプロパティ (region および accountId) を含む Python ディクショナリ。

次に、Layout.py ファイルに記述された、レイアウトジェネレータスクリプトの例を示します。

```
import argparse
import sys
import os
import json
from awsglue.blueprint.workflow import *
from awsglue.blueprint.job import *
from awsglue.blueprint.crawler import *


def generate_layout(user_params, system_params):

    etl_job = Job(Name="{}_etl_job".format(user_params['WorkflowName']),
                  Command={
                      "Name": "glueetl",
                      "ScriptLocation": user_params['ScriptLocation'],
                      "PythonVersion": "2"
                  },
                  Role=user_params['PassRole'])
    post_process_job = Job(Name="{}_post_process".format(user_params['WorkflowName']),
                           Command={
                               "Name": "pythonshell",
```

```
                                "ScriptLocation": user_params['ScriptLocation'],
                                "PythonVersion": "2"
                            },
                            Role=user_params['PassRole'],
                            DependsOn={
                                etl_job: "SUCCEEDED"
                            },
                            WaitForDependencies="AND")
    sample_workflow = Workflow(Name=user_params['WorkflowName'],
                            Entities=Entities(Jobs=[etl_job, post_process_job]))
    return sample_workflow
```

このサンプルスクリプトは、必要な設計図ライブラリをインポートし、2 つのジョブを含むワークフローを生成する generate_layout 関数をインクルードします。これは非常にシンプルなスクリプトです。より複雑なスクリプトでは、追加のロジックとパラメータを使用して、多数のジョブとクローラを含むワークフローを生成できます。さらに、ジョブとクローラの数を可変にすることも可能です

DependsOn 引数を使用する

引数 DependsOn は、このエンティティがワークフロー内の他のエンティティに対し維持する依存関係を、ディクショナリ向けに表現したものです。これには、以下の形式が使用されます。

```
DependsOn = {dependency1 : state, dependency2 : state, ...}
```

このディクショナリーのキーはエンティティの名前ではなく、オブジェクトリファレンスを表し、値は監視すべき状態に対応する文字列です。AWS Glue 適切なトリガーを推測します。有効な状態については、「Condition Structure」を参照してください。

例えば、ジョブは、クローラの正常な完了に依存する場合があります。crawler2 という名前のクローラオブジェクトを以下のように定義したとします。

```
crawler2 = Crawler(Name="my_crawler", ...)
```

この時、crawler2 に依存するオブジェクトには、次のようなコンストラクタ引数が含まれます。

```
DependsOn = {crawler2 : "SUCCEEDED"}
```

例:

```
job1 = Job(Name="Job1", ..., DependsOn = {crawler2 : "SUCCEEDED", ...})
```

仮にエンティティで DependsOn が省略されている場合には、そのエンティティはワークフローの開始トリガーに依存します。

WaitForDependencies 引数を使用する

WaitForDependencies 引数は、ジョブまたはクローラエンティティが、依存するすべてのエンティティが完了するまで待機するのか、あるいは、いずれかのエンティティが完了するまで待機するのかを決定します。

許容される値は「AND」または「ANY」です。

OnSchedule 引数を使用する

Workflow クラスコンストラクタの引数 OnSchedule は、ワークフローの開始トリガーを定義する cron 式です。

この引数を指定した場合、AWS Glue はスケジュールと対応付けながら、スケジュールトリガーを作成します。特に指定されていない場合は、ワークフローの開始トリガーはオンデマンドトリガーになります。

設定ファイルを作成する

設計図の設定ファイルは必須のファイルで、ワークフローを生成するためのスクリプトエントリポイントと、設計図が受け取るパラメータを定義します。ファイル名は、blueprint.cfg とする必要があります。

設定ファイルの例を次に示します。

```
{
    "layoutGenerator": "DemoBlueprintProject.Layout.generate_layout",
    "parameterSpec" : {
           "WorkflowName" : {
                "type": "String",
                "collection": false
           },
           "WorkerType" : {
                "type": "String",
                "collection": false,
                "allowedValues": ["G1.X", "G2.X"],
                "defaultValue": "G1.X"
```

```
            },
            "Dpu" : {
                 "type" : "Integer",
                 "allowedValues" : [2, 4, 6],
                 "defaultValue" : 2
            },
            "DynamoDBTableName": {
                 "type": "String",
                 "collection" : false
            },
            "ScriptLocation" : {
                 "type": "String",
                 "collection": false
     }
    }
}
```

layoutGenerator プロパティでは、レイアウトを生成するスクリプト内の関数について、完全な修飾名を指定します。

parameterSpec プロパティでは、この設計図が受け取るパラメータを指定します。詳細については、設計図パラメータを指定する を参照してください。

> **Important**
>
> 設定ファイルに設計図パラメータとしてワークフロー名を含めるか、あるいはレイアウトスクリプトにより一意のワークフロー名を生成する必要があります。

設計図パラメータを指定する

設定ファイルには、parameterSpec JSON オブジェクトの設計図パラメータの仕様が含まれます。parameterSpec には、1 つ以上のパラメータオブジェクトが含まれます。

```
"parameterSpec": {
    "<parameter_name>": {
      "type": "<parameter-type>",
      "collection": true|false,
      "description": "<parameter-description>",
      "defaultValue": "<default value for the parameter if value not specified>"
      "allowedValues": "<list of allowed values>"
    },
```

```
    "<parameter_name>": {
        ...
    }
}
```

以下は、各パラメータオブジェクトでのコーディング規則です。

- パラメータ名と type は必須です。その他のプロパティはすべてオプションです。
- defaultValue プロパティを指定する場合のパラメータはオプションです。それ以外の場合、パラメータは必須であり、この設計図からワークフローを作成するデータアナリストは、その値を提供する必要があります。
- collection プロパティに true を設定した場合、このパラメータは値のコレクションを参照できます。コレクションは、任意のデータ型にすることができます。
- allowedValues を指定すると、ブループリントからワークフローを作成する際にデータアナリストが選択する、値のドロップダウンリストが AWS Glue コンソールに表示されます。

type で使用可能な値を以下に示します。

パラメータのデータ型	メモ
String	-
Integer	-
Double	-
Boolean	指定できる値は true および false です。AWS Glue コンソールの [Create a workflow from <blueprint>] (<blueprint> からのワークフローの作成) ページで、チェックボックスを生成します。
S3Uri	s3:// で開始する Amazon S3 パスを完成させます。[Create a workflow from <blueprint>] (<blueprint> からのワークフローの作成) ページで、テキストフィールドおよび [Browse] (参照) ボタンを生成します。
S3Bucket	Amazon S3 バケット名のみ。[Create a workflow from <blueprint>] (<blueprint> からのワークフローの作成) ページで、バケットピッカーを生成します。

パラメータのデータ型	メモ
IAMRoleArn	AWS Identity and Access Management (IAM ロール) の Amazon リソースネーム (ARN)。[Create a workflow from <blueprint>] (<blueprint> からのワークフローの作成) ページで、ロールピッカーを生成します。
IAMRoleName	IAM ロールの名前。[Create a workflow from <blueprint>] (<blueprint> からのワークフローの作成) ページで、ロールピッカーを生成します。

設計図プロジェクト例

データ形式の変換は、抽出、変換、ロード (ETL) で頻繁に行われるユースケースです。一般的な分析ワークロードでは、CSV や JSON などのテキスト形式よりも、Parquet や ORC などの列ベースのファイル形式の使用が好まれます。このサンプル設計図を使用すると、Amazon S3 上のファイルで、CSV/JSON/その他のデータを Parquet に変換できます。

この設計図では、設計図パラメータで定義された S3 パスのリストを取得し、そのデータを Parquet 形式に変換した上で、別の設計図パラメータで指定された S3 の口ケーションに書き込んでいます。このレイアウトスクリプトは、パスごとにクローラとジョブを作成します。また、このレイアウトスクリプトでは、Conversion.py 内の ETL スクリプトを、別の設計図パラメータで指定された S3 バケットにアップロードしています。その後、このレイアウトスクリプトは、アップロードされたスクリプトを各ジョブの ETL スクリプトとして指定します。プロジェクト用の ZIP アーカイブには、レイアウトスクリプト、ETL スクリプト、および設計図の設定ファイルが含まれています。

その他の設計図プロジェクトの例については、「設計図の例」を参照してください。

以下は、Layout.py ファイル内にあるレイアウトスクリプトです。

```
from awsglue.blueprint.workflow import *
from awsglue.blueprint.job import *
from awsglue.blueprint.crawler import *
import boto3

s3_client = boto3.client('s3')

# Ingesting all the S3 paths as Glue table in parquet format
def generate_layout(user_params, system_params):
    #Always give the full path for the file
```

```
    with open("ConversionBlueprint/Conversion.py", "rb") as f:
        s3_client.upload_fileobj(f, user_params['ScriptsBucket'], "Conversion.py")
    etlScriptLocation = "s3://{}/Conversion.py".format(user_params['ScriptsBucket'])

    crawlers = []
    jobs = []
    workflowName = user_params['WorkflowName']
    for path in user_params['S3Paths']:
      tablePrefix = "source_"
      crawler = Crawler(Name="{}_crawler".format(workflowName),
                        Role=user_params['PassRole'],
                        DatabaseName=user_params['TargetDatabase'],
                        TablePrefix=tablePrefix,
                        Targets= {"S3Targets": [{"Path": path}]})
      crawlers.append(crawler)
      transform_job = Job(Name="{}_transform_job".format(workflowName),
                          Command={"Name": "glueetl",
                                   "ScriptLocation": etlScriptLocation,
                                   "PythonVersion": "3"},
                          Role=user_params['PassRole'],
                          DefaultArguments={"--database_name":
 user_params['TargetDatabase'],
                                            "--table_prefix": tablePrefix,
                                            "--region_name": system_params['region'],
                                            "--output_path":
 user_params['TargetS3Location']},
                          DependsOn={crawler: "SUCCEEDED"},
                          WaitForDependencies="AND")
      jobs.append(transform_job)
    conversion_workflow = Workflow(Name=workflowName, Entities=Entities(Jobs=jobs,
 Crawlers=crawlers))
    return conversion_workflow
```

以下は、対応する設計図の設定ファイル blueprint.cfg です。

```
{
    "layoutGenerator": "ConversionBlueprint.Layout.generate_layout",
    "parameterSpec" : {
        "WorkflowName" : {
            "type": "String",
            "collection": false,
            "description": "Name for the workflow."
        },
```

```
        "S3Paths" : {
            "type": "S3Uri",
            "collection": true,
            "description": "List of Amazon S3 paths for data ingestion."
        },
        "PassRole" : {
            "type": "IAMRoleName",
            "collection": false,
            "description": "Choose an IAM role to be used in running the job/crawler"
        },
        "TargetDatabase": {
            "type": "String",
            "collection" : false,
            "description": "Choose a database in the Data Catalog."
        },
        "TargetS3Location": {
            "type": "S3Uri",
            "collection" : false,
            "description": "Choose an Amazon S3 output path: ex:s3://<target_path>/."
        },
        "ScriptsBucket": {
            "type": "S3Bucket",
            "collection": false,
            "description": "Provide an S3 bucket name(in the same AWS Region) to store
 the scripts."
        }
    }
}
```

以下に示す、ファイル Conversion.py 内のスクリプトは、アップロードされた ETL スクリプトです。ここでは、変換中にパーティション分割スキームが保持されることに注意してください。

```
import sys
from pyspark.sql.functions import *
from pyspark.context import SparkContext
from awsglue.transforms import *
from awsglue.context import GlueContext
from awsglue.job import Job
from awsglue.utils import getResolvedOptions
import boto3

args = getResolvedOptions(sys.argv, [
    'JOB_NAME',
```

```
    'region_name',
    'database_name',
    'table_prefix',
    'output_path'])
databaseName = args['database_name']
tablePrefix = args['table_prefix']
outputPath = args['output_path']

glue = boto3.client('glue', region_name=args['region_name'])

glue_context = GlueContext(SparkContext.getOrCreate())
spark = glue_context.spark_session
job = Job(glue_context)
job.init(args['JOB_NAME'], args)

def get_tables(database_name, table_prefix):
    tables = []
    paginator = glue.get_paginator('get_tables')
    for page in paginator.paginate(DatabaseName=database_name, Expression=table_prefix
+"*"):
        tables.extend(page['TableList'])
    return tables

for table in get_tables(databaseName, tablePrefix):
    tableName = table['Name']
    partitionList = table['PartitionKeys']
    partitionKeys = []
    for partition in partitionList:
        partitionKeys.append(partition['Name'])

    # Create DynamicFrame from Catalog
    dyf = glue_context.create_dynamic_frame.from_catalog(
        name_space=databaseName,
        table_name=tableName,
        additional_options={
            'useS3ListImplementation': True
        },
        transformation_ctx='dyf'
    )

    # Resolve choice type with make_struct
    dyf = ResolveChoice.apply(
        frame=dyf,
        choice='make_struct',
```

```
        transformation_ctx='resolvechoice_' + tableName
    )

    # Drop null fields
    dyf = DropNullFields.apply(
        frame=dyf,
        transformation_ctx="dropnullfields_" + tableName
    )

    # Write DynamicFrame to S3 in glueparquet
    sink = glue_context.getSink(
        connection_type="s3",
        path=outputPath,
        enableUpdateCatalog=True,
        partitionKeys=partitionKeys
    )
    sink.setFormat("glueparquet")

    sink.setCatalogInfo(
        catalogDatabase=databaseName,
        catalogTableName=tableName[len(tablePrefix):]
    )
    sink.writeFrame(dyf)

job.commit()
```

> **Note**
>
> このサンプルの設計図に対し、入力として提供できる Amazon S3 パスは 2 つだけです。これは、AWS Glue トリガーが呼び出すことができるクローラアクションが、2 つだけに制限されているためです。

設計図をテストする

コードの開発時には、ローカルなテストを実行して、ワークフローのレイアウトに誤りがないことを確認する必要があります。

ローカルテストでは AWS Glue ジョブ、クローラ、トリガーは生成されません。代わりに、レイアウトスクリプトをローカルで実行し、`to_json()` および `validate()` メソッドによりオブジェクトを画面表示し、エラーを発見します。これらのメソッドは、ライブラリで定義されている 3 つのクラスすべてで使用できます。

user_params がレイアウト関数に渡す、引数 system_params および AWS Glue を処理するには 2 つの方法があります。テストベンチコードでは、サンプル設計図パラメータ値のディクショナリを作成し、それを引数 user_params としてレイアウト関数に渡すことができます。または、user_params への参照を削除し、その部分をハードコーディングのための文字列に置き換えます。

コードの引数 system_params の中で、region および accountId プロパティを利用している場合には、system_params を独自の辞書に含めて渡すことができます。

設計図をテストするには

1. Python インタプリタをライブラリのあるディレクトリで起動するか、設計図ファイルならびに提供されたライブラリを、希望の統合開発環境 (IDE) にロードします。
2. コードによって提供されたライブラリがインポートされていることを確認します。
3. 任意のエンティティまたは Workflow オブジェクトで validate() または to_json() 呼び出すためのコードを、レイアウト関数に追加します。例えば、コードが mycrawler という名前の Crawler オブジェクトを作成する場合には、以下のように validate() を呼び出せます。

   ```
   mycrawler.validate()
   ```

   mycrawler は以下のように表示できます。

   ```
   print(mycrawler.to_json())
   ```

   オブジェクトで to_json を呼び出す場合、別途 validate() を呼び出す必要はありません。validate() は to_json() により呼び出されます。

   これにより、ワークフローオブジェクトでのこれらのメソッドの呼び出しが、最も効率的に行えます。スクリプトで、ワークフローオブジェクトに my_workflow と名前を付けているのであれば、次のようにワークフローオブジェクトを検証して画面表示します。

   ```
   print(my_workflow.to_json())
   ```

   to_json() と validate() の詳細については、「クラスメソッド」を参照してください。

   また、このセクションの後半の例に示すように、pprint をインポートし、ワークフローオブジェクトを書式を設定しながら表示することも可能です。

4. コードを実行し、エラーを修正した上で、最後に validate() または to_json() の呼び出しを削除します。

Example

次の例では、サンプルの設計図パラメータのディクショナリを構築し、それをレイアウト関数 generate_compaction_workflow の引数 user_params に渡す方法を示しています。また、生成されたワークフローオブジェクトを、書式を指定しながら印刷するための方法も示しています。

```
from pprint import pprint
from awsglue.blueprint.workflow import *
from awsglue.blueprint.job import *
from awsglue.blueprint.crawler import *

USER_PARAMS = {"WorkflowName": "compaction_workflow",
               "ScriptLocation": "s3://amzn-s3-demo-bucket/scripts/threaded-
compaction.py",
               "PassRole": "arn:aws:iam::111122223333:role/GlueRole-ETL",
               "DatabaseName": "cloudtrial",
               "TableName": "ct_cloudtrail",
               "CoalesceFactor": 4,
               "MaxThreadWorkers": 200}


def generate_compaction_workflow(user_params: dict, system_params: dict) -> Workflow:
    compaction_job = Job(Name=f"{user_params['WorkflowName']}_etl_job",
                         Command={"Name": "glueetl",
                                  "ScriptLocation": user_params['ScriptLocation'],
                                  "PythonVersion": "3"},
                         Role="arn:aws:iam::111122223333:role/
AWSGlueServiceRoleDefault",
                         DefaultArguments={"DatabaseName": user_params['DatabaseName'],
                                           "TableName": user_params['TableName'],
                                           "CoalesceFactor":
 user_params['CoalesceFactor'],
                                           "max_thread_workers":
 user_params['MaxThreadWorkers']})

    catalog_target = {"CatalogTargets": [{"DatabaseName": user_params['DatabaseName'],
 "Tables": [user_params['TableName']]}]}

    compacted_files_crawler = Crawler(Name=f"{user_params['WorkflowName']}_post_crawl",
```

```
                                    Targets = catalog_target,
                                    Role=user_params['PassRole'],
                                    DependsOn={compaction_job: "SUCCEEDED"},
                                    WaitForDependencies="AND",
                                    SchemaChangePolicy={"DeleteBehavior": "LOG"})

    compaction_workflow = Workflow(Name=user_params['WorkflowName'],
                                   Entities=Entities(Jobs=[compaction_job],

 Crawlers=[compacted_files_crawler]))
    return compaction_workflow

generated = generate_compaction_workflow(user_params=USER_PARAMS, system_params={})
gen_dict = generated.to_json()

pprint(gen_dict)
```

設計図を公開する

作成した設計図は、Amazon S3 にアップロードする必要があります。これには、設計図を公開するために使用する Amazon S3 バケットに対する、書き込みアクセス許可が必要です。また、ブループリントの登録を行う AWS Glue 管理者には、Amazon S3 バケットからの読み取りアクセスが許可されている必要があります。AWS Identity and Access Management 設計図でのペルソナおよびロールのために提案されている、AWS Glue (IAM) アクセス許可ポリシーについては、「AWS Glue ブループリントでのペルソナおよびロール用のアクセス許可」を参照してください。

設計図を公開するには

1. 必要なスクリプト、リソース、および設計図の設定ファイルを作成します。
2. すべてのファイルを ZIP アーカイブに追加し、その ZIP ファイルを Amazon S3 にアップロードします。設計図を登録し実行するリージョンと、同じリージョンにある S3 バケットを使用します。

   ZIP ファイルは、次のコマンドを使用して、コマンドラインから作成することができます。

   ```
   zip -r folder.zip folder
   ```

3. AWS での希望するアカウントに読み取りアクセス許可を付与する、バケットポリシーを追加します。次に、そのポリシー例を示します。

   ```
   {
   ```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:root"
            },
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::my-blueprints/*"
        }
    ]
}
```

4. Amazon S3 バケットに対する IAM の `s3:GetObject` の許可を、AWS Glue 管理者またはブループリントを登録する任意のユーザーに付与します。管理者に付与するポリシーの例については、「ブループリントのための AWS Glue 管理者のアクセス許可」を参照してください。

ブループリントに関するローカルでのテストが完了したら、AWS Glue 上でもブループリントをテストします。ブループリントを AWS Glue 上でテストするには、登録が完了している必要があります。登録された設計図を表示できるユーザーを制限するには、IAM 認可を使用するか、テスト用に分離されたアカウントを使用します。

以下も参照してください。

- AWS Glue でブループリントを登録する

AWS Glue ブループリントクラスリファレンス

AWS Glue ブループリント用のライブラリでは、ワークフローレイアウトスクリプトで使用する 3 つのクラス (`Job`、`Crawler`、および `Workflow`) を定義します。

トピック

- Job クラス
- Crawler クラス
- Workflow クラス
- クラスメソッド

Job クラス

Job クラスは、AWS Glue での ETL ジョブを表します。

必須のコンストラクター引数

Job クラスのコンストラクタに必須な引数を、以下に示します。

引数名	型	説明
Name	str	ジョブに割り当てる名前。AWS Glue では他のブループリントの実行で作成されたジョブとの区別するため、名前にはランダムに生成されるサフィックスを付けています。
Role	str	ジョブが実行中に引き受けるロールの Amazon リソースネーム (ARN)。
Command	dict	API ドキュメントの JobCommand 構造 に掲載されている Job コマンド。

オプションのコンストラクター引数

Job クラスのコンストラクタで、オプションとなっている引数を以下に示します。

引数名	型	説明
DependsOn	dict	ジョブが依存するワークフローエンティティの一覧。詳細については、DependsOn 引数を使用する を参照してください。
WaitForDependencies	str	実行前のジョブが、依存するすべてのエンティティが完了するまで待機するのか、あるいは任意のエンティティが完了することを待つのかを示します。詳細については、WaitForDependencies 引数を使用する を参照してください。ジョブが 1 つのエンティティのみに依存する場合は、この設定は省略します。

引数名	型	説明
(ジョブプロパティ)	-	Job 構造 API ドキュメント (AWS Glue と CreatedOn 以外)LastModifiedOn の一覧に掲載されている、いずれかのジョブのプロパティ。

Crawler クラス

Crawler クラスは、AWS Glue でのクローラーを表します。

必須のコンストラクター引数

Crawler クラスのコンストラクタに必須な引数を、以下に示します。

引数名	型	説明
Name	str	AWS Glue は、ランダムに生成されたサフィックスをクローラーの名前に付加することで、異なるブループリントの実行によって作成されたクローラを区別します。
Role	str	クローラが実行中に引き受ける必要のあるロールの ARN。
Targets	dict	クロールするターゲットのコレクション。Targets クラスコンストラクターの引数は、API ドキュメントの CrawlerTargets 構造 に定義されています。Targets コンストラクタの引数はすべてオプションです。ただし、少なくとも 1 つを渡す必要があります。

オプションのコンストラクター引数

Crawler クラスのコンストラクタで、オプションとなっている引数を以下に示します。

引数名	型	説明
DependsOn	dict	クローラが依存するワークフローエンティティのリスト。詳細については、DependsOn 引数を使用する を参照してください。
WaitForDependencies	str	実行前のクローラが、依存するすべてのエンティティが完了するまで待機するのか、任意のエンティティが完了することを待つのかを示します。詳細については、WaitForDependencies 引数を使用する を参照してください。クローラが 1 つのエンティティのみに依存する場合は、この設定は省略します。
(クローラのプロパティ)	-	Crawler 構造 API ドキュメントの AWS Glue の一覧に掲載されている、いずれかのクローラのプロパティ (以下のものを除く)。 • State • CrawlElapsedTime • CreationTime • LastUpdated • LastCrawl • Version

Workflow クラス

Workflow クラスは、AWS Glue の ワークフローを表します。ワークフローレイアウトスクリプトは、Workflow オブジェクト。AWS Glue でこのオブジェクトに基づいてワークフローを作成します。

必須のコンストラクター引数

Workflow クラスのコンストラクタに必須な引数を、以下に示します。

引数名	型	説明
Name	str	ワークフローに割り当てる名前。
Entities	Entities	ワークフローに含めるエンティティ (ジョブおよびクローラ) のコレクション。Entities クラスのコンストラクタは、引数 Jobs (Job オブジェクトのリスト)、および、Crawlers 引数 (Crawler オブジェクトのリスト) を受け取ります。

オプションのコンストラクター引数

Workflow クラスのコンストラクタで、オプションとなっている引数を以下に示します。

引数名	型	説明
Description	str	「」を参照してくださいWorkflow 構造
DefaultRunProperties	dict	「」を参照してくださいWorkflow 構造
OnSchedule	str	cron 式

クラスメソッド

上記の 3 つのクラスには、以下のメソッドが含まれています。

validate()

オブジェクトのプロパティを検証し、エラーが見つかった場合はメッセージを出力して終了します。エラーが発見されない場合は、出力を生成しません。Workflow クラスでは、ワークフロー内のすべてのエンティティで、自分自身を呼び出します。

to_json()

オブジェクトを JSON としてシリアル化します。また、validate() を呼び出します。Workflow クラスでは、この JSON オブジェクトにはジョブとクローラのリストと、ジョブとクローラの依存関係の仕様によって生成されたトリガーのリストが含まれます。

設計図の例

AWS Glue ブループリント の Githubリポジトリには、設計図プロジェクトの例が多数提供されています。これらのサンプルは参考用であり、本番での使用は意図されていません。

これらのサンプルプロジェクトのタイトルを以下に示します。

- Compaction: この設計図では、必要なファイルサイズに基づいて、複数の入力ファイルをより大きなチャンクの中に圧縮するジョブを作成します。
- Conversion: この設計図では、さまざまな標準ファイル形式の入力ファイルを、分析ワークロード向けに最適化された Apache Parquet 形式に変換します。
- Crawling Amazon S3 locations:この設計図は、複数の Amazon S3 ロケーションをクロールして、Data Catalog にメタデータテーブルを追加します。
- Custom connection to Data Catalog: この設計図では、AWS Glue のカスタムコネクタを使用しデータストアにアクセスします。レコードを読み取り、レコードスキーマに基づいて AWS Glue データカタログのテーブル定義を設定します。
- Encoding: この設計図では、UTF 以外のファイルを UTF エンコードされたファイルに変換します。
- Partitioning: この設計図では、パーティショニングジョブを作成します。特定のパーティションキーに基づいて出力ファイルをパーティションに配置します。
- Importing Amazon S3 data into a DynamoDB table: この設計図では、Amazon S3 から DynamoDB テーブルにデータをインポートします。
- Standard table to governed: この設計図では、Lake Formation テーブルに AWS Glue データカタログテーブルをインポートします。

AWS Glue でブループリントを登録する

AWS Glue デベロッパーが設計図をコーディングして、Amazon Simple Storage Service (Amazon S3) に ZIP アーカイブをアップロードした後、AWS Glue 管理者はその設計図を登録する必要があります。設計図は、登録されることで使用が可能になります。

ブループリントを登録すると、AWS Glue はブループリントアーカイブを予約済みの Amazon S3 ロケーションにコピーします。その後、アーカイブは、アップロードした場所から削除できます。

設計図を登録するには、アップロードされたアーカイブが保存される Amazon S3 ロケーションに対する、読み取りアクセス許可が必要です。また、AWS Identity and Access Management (IAM) のア

クセス許可 glue:CreateBlueprint も必要です。ブループリントの登録、表示、および保守を行う必要がある AWS Glue 管理者のために、提案されているアクセス許可については、ブループリントのための AWS Glue 管理者のアクセス許可 を参照してください。

ブループリントは AWS Glue コンソール、AWS Glue API、または AWS Command Line Interface (AWS CLI) を使用して登録できます。

設計図を登録するには (コンソール)

1. Amazon S3 にある設計図の ZIP アーカイブに対する、読み取りアクセス許可 (s3:GetObject) が付与されていることを確認します。
2. https://console.aws.amazon.com/glue/ で AWS Glue コンソール を開きます。

   設計図を登録するための、アクセス許可が付与されたユーザーとしてサインインします。設計図の ZIP アーカイブを含む Amazon S3 バケットと同じ、AWS リージョンに切り替えます。
3. ナビゲーションペインで [Blueprints] (ブループリント) を選択します。次に、[Blueprints] (設計図) ページで [Add blueprint] (設計図を追加) をクリックします。
4. 設計図の名前と、必要に応じて説明を入力します。
5. [ZIP archive location (S3)] (ZIP アーカイブの場所 (S3))に、アップロードされた設計図の ZIP アーカイブがある Amazon S3 パスを入力します。パスの先頭は s3:// で開始し、またアーカイブのファイル名を含めます。
6. (オプション) 1 つ以上のタグを追加します。
7. [Add blueprint] (設計図を追加) をクリックします。

   [Blueprints] (設計図) ページが再び開き、設計図のステータスが「CREATING」となっていることを表示します。ステータスが「ACTIVE」または「FAILED」に変わるまで、[Refresh] (更新) ボタンをクリックします。
8. ステータスが「FAILED」となった場合は、設計図を選択した上で、[Actions] (アクション) メニューで [View] (表示) をクリックします。

   詳細ページに、失敗の理由が表示されます。エラーメッセージが、「Unable to access object at location...」または「Access denied on object at location...」となっている場合は、以下の要件を確認してください。

   - サインインしているユーザーには、Amazon S3 内にある設計図の ZIP アーカイブに対する読み取りアクセス許可が必要です。

- ZIP アーカイブを含む Amazon S3 バケットには、オブジェクトに対する読み取りアクセス許可を AWS アカウント ID に付与するためのバケットポリシーが必要です。(詳細については、AWS Glue のブループリントの開発 を参照してください)。
- 使用している Amazon S3 バケットは、コンソールでサインインしているリージョンと同じリージョンにあることが必要です。

9. データアナリストに、設計図へのアクセス許可が付与されていることを確認します。

   データアナリストのために提案された IAM ポリシーは、「設計図のためのデータアナリストの権限」で確認できます。このポリシーでは、任意のリソースに `glue:GetBlueprint` を付与します。リソースレベルでポリシーをより細かく設定している場合には、その新しく作成したリソースに対するアクセス許可をデータアナリストに付与します。

設計図を登録するには (AWS CLI)

1. 次のコマンドを入力します。

```
aws glue create-blueprint --name <blueprint-name> [--description <description>] --
blueprint-location s3://<s3-path>/<archive-filename>
```

2. 設計図のステータスを確認するには、次のコマンドを入力します。ステータスが `ACTIVE` または `FAILED` に変化するまで、このコマンドを繰り返します。

```
aws glue get-blueprint --name <blueprint-name>
```

   ステータスが `FAILED` となり、エラーメッセージに「Unable to access object at location...」または「Access denied on object at location...」が出力された場合は、以下の要件を確認してください。

   - サインインしているユーザーには、Amazon S3 内にある設計図の ZIP アーカイブに対する読み取りアクセス許可が必要です。
   - ZIP アーカイブを含む Amazon S3 バケットには、オブジェクトに対する読み取りアクセス許可を AWS アカウント ID に付与するためのバケットポリシーが必要です。詳細については、設計図を公開する を参照してください。
   - 使用している Amazon S3 バケットは、コンソールでサインインしているリージョンと同じリージョンにあることが必要です。

以下も参照してください。

- AWS Glue のブループリント概要

AWS Glue でブループリントを表示する

設計図を表示して、その説明とステータスおよびパラメータ仕様を確認し、設計図の ZIP アーカイブをダウンロードします。

ブループリントを表示するには、AWS Glue コンソール、AWS Glue API、または AWS Command Line Interface (AWS CLI) を使用します。

設計図を表示するには (コンソール)

1. https://console.aws.amazon.com/glue/ で AWS Glue コンソール を開きます。
2. ナビゲーションペインで [Blueprints] (ブループリント) を選択します。
3. [Blueprints] (設計図) ページで、設計図を選択します。次に、[Actions] (アクション) メニューで [View] (表示) をクリックします。

設計図を表示するには (AWS CLI)

- 設計図の名前、説明、ステータスのみを表示するには、次のコマンドを入力します。*<blueprint-name>*は、表示する設計図の名前に置き換えます。

```
aws glue get-blueprint --name <blueprint-name>
```

次のような出力が表示されます。

```
{
    "Blueprint": {
        "Name": "myDemoBP",
        "CreatedOn": 1587414516.92,
        "LastModifiedOn": 1587428838.671,
        "BlueprintLocation": "s3://amzn-s3-demo-bucket1/demo/
DemoBlueprintProject.zip",
        "Status": "ACTIVE"
    }
```

```
}
```

以下のコマンドを入力することで、パラメータの仕様も表示できます。

```
aws glue get-blueprint --name <blueprint-name>  --include-parameter-spec
```

次のような出力が表示されます。

```
{
    "Blueprint": {
        "Name": "myDemoBP",
        "CreatedOn": 1587414516.92,
        "LastModifiedOn": 1587428838.671,
        "ParameterSpec": "{\"WorkflowName\":{\"type\":\"String\",\"collection
\":false,\"description\":null,\"defaultValue\":null,\"allowedValues\":null},
\"PassRole\":{\"type\":\"String\",\"collection\":false,\"description\":null,
\"defaultValue\":null,\"allowedValues\":null},\"DynamoDBTableName\":{\"type
\":\"String\",\"collection\":false,\"description\":null,\"defaultValue\":null,
\"allowedValues\":null},\"ScriptLocation\":{\"type\":\"String\",\"collection
\":false,\"description\":null,\"defaultValue\":null,\"allowedValues\":null}}",
        "BlueprintLocation": "s3://awsexamplebucket1/demo/
DemoBlueprintProject.zip",
        "Status": "ACTIVE"
    }
}
```

引数 `--include-blueprint` を使用して、出力に URL を含めることもできます。この URL をブラウザに貼り付けて、AWS Glue に保存されているブループリントの zip アーカイブをダウンロードできます。

以下も参照してください。

- AWS Glue のブループリント概要

AWS Glue でブループリントを更新する

レイアウトスクリプトの改訂、設計図のパラメータセットの改訂、またはサポートファイルの改訂が行われた場合は、設計図を更新できます。更新により、新しいバージョンの設計図が作成されます。

設計図を更新した場合でも、その設計図から作成された既存のワークフローに影響は与えません。

設計図の更新には、AWS Glue コンソール、AWS Glue API、または AWS Command Line Interface (AWS CLI) を使用します。

以下に示す手順では、AWS Glue デベロッパーが、設計図の ZIP アーカイブを Amazon S3 で作成およびアップロードし、その更新を行うと想定しています。

設計図を更新するには (コンソール)

1. Amazon S3 にある設計図の ZIP アーカイブに対する、読み取りアクセス許可 (`s3:GetObject`) が付与されていることを確認します。
2. https://console.aws.amazon.com/glue/ で AWS Glue コンソール を開きます。

   設計図を更新するためのアクセス許可が付与されたユーザーとしてサインインします。設計図の ZIP アーカイブを含む Amazon S3 バケットと同じ、AWS リージョンに切り替えます。
3. ナビゲーションペインで [Blueprints] (ブループリント) を選択します。
4. [Blueprints] (設計図) ページで設計図を選択し、[Actions] (アクション) メニューで、[Edit] (編集) をクリックします。
5. [Edit a blueprint] (設計図の編集) ページで、設計図の [Description] (説明)、もしくは [ZIP archive location (S3)] (ZIP アーカイブの場所 (S3)) を更新します。パスにはアーカイブのファイル名を含める必要があります。
6. [Save] を選択します。

   [Blueprints] (設計図) ページが再び開き、設計図のステータスが「`UPDATING`」となっていることを表示します。ステータスが「`ACTIVE`」または「`FAILED`」に変わるまで、[Refresh] (更新) ボタンをクリックします。
7. ステータスが「`FAILED`」となった場合は、設計図を選択した上で、[Actions] (アクション) メニューで [View] (表示) をクリックします。

   詳細ページに、失敗の理由が表示されます。エラーメッセージが、「Unable to access object at location...」または「Access denied on object at location...」となっている場合は、以下の要件を確認してください。

   - サインインしているユーザーには、Amazon S3 内にある設計図の ZIP アーカイブに対する読み取りアクセス許可が必要です。

- ZIP アーカイブを含む Amazon S3 バケットには、オブジェクトに対する読み取りアクセス許可を AWS アカウント ID に付与するためのバケットポリシーが必要です。詳細については、設計図を公開する を参照してください。
- 使用している Amazon S3 バケットは、コンソールでサインインしているリージョンと同じリージョンにあることが必要です。

> **Note**
>
> 更新が失敗した場合、次回の設計図実行では、正常に登録または更新された設計図の最新バージョンが使用されます。

設計図を更新するには (AWS CLI)

1. 次のコマンドを入力します。

```
aws glue update-blueprint --name <blueprint-name> [--description <description>] --
blueprint-location s3://<s3-path>/<archive-filename>
```

2. 設計図のステータスを確認するには、次のコマンドを入力します。ステータスが ACTIVE または FAILED に変化するまで、このコマンドを繰り返します。

```
aws glue get-blueprint --name <blueprint-name>
```

ステータスが FAILED となり、エラーメッセージに「Unable to access object at location...」または「Access denied on object at location...」が出力された場合は、以下の要件を確認してください。

- サインインしているユーザーには、Amazon S3 内にある設計図の ZIP アーカイブに対する読み取りアクセス許可が必要です。
- ZIP アーカイブを含む Amazon S3 バケットには、オブジェクトに対する読み取りアクセス許可を AWS アカウント ID に付与するためのバケットポリシーが必要です。詳細については、設計図を公開する を参照してください。
- 使用している Amazon S3 バケットは、コンソールでサインインしているリージョンと同じリージョンにあることが必要です。

以下も参照してください。

- AWS Glue のブループリント概要

AWS Glue で設計図からワークフローを作成する

AWS Glue ワークフローはコンポーネントを 1 つずつ手動で作成したり、AWS Glue ブループリントからワークフローを作成することもできます。AWS Glue には一般的なユースケースのブループリントが含まれます。AWS Glue デベロッパーが、追加の設計図を作成することも可能です。

Important

ワークフロー内のジョブ、クローラ、トリガーの総数を 100 以下に制限します。100 を超える値を含めると、ワークフローの実行を再開または停止しようとしたときにエラーが発生することがあります。

設計図を使用すると、設計図で定義されている一般化されたユースケースをベースにしながら、特定のユースケース向けのワークフローをすばやく生成できます。設計図パラメータの値を指定することで、特定のユースケースを定義します。例えば、データセットをパーティション化する設計図では、パラメータとして Amazon S3 のソースとターゲットのパスを使用します。

AWS Glue は、ブループリントを実行することで、そのブループリントからワークフローを作成します。設計図の実行では、指定したパラメータ値が保存されます。このパラメータは、ワークフローとそのコンポーネントの作成に関する進行状況と結果を追跡するためにも使用されます。ワークフローをトラブルシューティングする場合は、設計図の実行を表示して、ワークフローの作成に使用された設計図のパラメータ値を確認します。

ワークフローを作成および表示するには、特定の IAM アクセス許可が必要です。提案されている IAM ポリシーについては、「設計図のためのデータアナリストの権限」を参照してください。

ブループリントからワークフローを作成するには、AWS Glue コンソール、AWS Glue API、または AWS Command Line Interface (AWS CLI) を使用します。

設計図からワークフローを作成するには (コンソール)

1. https://console.aws.amazon.com/glue/ で AWS Glue コンソール を開きます。

ワークフローを作成するための、アクセス許可が付与されたユーザーとしてサインインします。

2. ナビゲーションペインで [Blueprints] (ブループリント) を選択します。
3. 設計図を選択し、[Actions] (アクション) メニューで、[Create workflow] (ワークフローの作成) をクリックします。
4. [Create a workflow from <blueprint-name>] (<blueprint-name> からのワークフローの作成) ページで、以下の情報を入力します。

   設計図パラメータ

   これらのパラーメータは、設計図の構成によって異なります。パラメータについては、デベロッパーにお問い合わせください。設計図には、通常、ワークフロー名がパラメータとして含まれています。

   IAM ロール

   ワークフローとそのコンポーネントを作成するために、AWS Glue が引き受けるロールです。ロールには、ワークフロー、ジョブ、クローラ、トリガーを作成および削除するための、アクセス許可が付与されている必要があります。提案されているロールのポリシーについては、「設計図のロール用のアクセス許可」を参照してください。
5. [送信] を選択します。

   [Blueprint Details] (設計図の詳細) ページが表示され、その下部に設計図の実行が一覧表示されます。
6. 設計図の実行の一覧で、最上位にある設計図の実行から、ワークフローの作成ステータスを確認します。

   初期ステータスは、`RUNNING` です。ステータスが `SUCCEEDED` または `FAILED` に遷移するまで、[Refresh] (更新) ボタンをクリックします。
7. 次のいずれかを行います。

   - 完了ステータスが `SUCCEEDED` であれば、[Workflows] (ワークフロー) ページを開き、新しく作成したワークフローを選択して実行できます。ワークフローを実行する前に、設計グラフを確認することができます。
   - 完了ステータスが `FAILED` の場合には、設計図実行を選択した上で [Actions] (アクション) メニューで [View] (表示) をクリックし、エラーメッセージを表示します。

Express ワークフローと サービス統合の詳細については、以下を参照してください。

- AWS Glue のワークフローの概要
- AWS Glue でブループリントを更新する
- AWS Glue でワークフローを手動により作成および構築する

AWS Glue でブループリントの実行を表示する

設計図の実行を表示して、次の情報を表示します。

- 作成されたワークフローの名前。
- ワークフローの作成に使用された設計図のパラメータ値。
- ワークフロー作成オペレーションのステータス。

ブループリントを表示するには、AWS Glue コンソール、AWS Glue API、または AWS Command Line Interface (AWS CLI) を使用します。

設計図の実行を表示するには (コンソール)

1. https://console.aws.amazon.com/glue/ で AWS Glue コンソール を開きます。
2. ナビゲーションペインで [Blueprints] (ブループリント) を選択します。
3. [Blueprints] (設計図) ページで、設計図を選択します。次に、[Actions] (アクション) メニューで [View] (表示) をクリックします。
4. [Blueprint Details] (設計図の詳細) ページの最下部で、設計図の実行を選択し、[Actions] (アクション) メニューから [View] 表示を選択します。

設計図の実行を表示するには (AWS CLI)

- 次のコマンドを入力します。*<blueprint-name>* を設計図の名前に置き換えます。*<blueprint-run-id>* を設計図の実行の ID に置き換えます。

```
aws glue get-blueprint-run --blueprint-name <blueprint-name> --run-id <blueprint-run-id>
```

以下も参照してください。

- AWS Glue のブループリント概要

AWS Glue の場合は AWS CloudFormation

AWS CloudFormation は、多くの AWS リソースを作成できるサービスです。AWS Glue には、AWS Glue Data Catalog でオブジェクトを作成するための API オペレーションが用意されています。ただし、AWS Glue オブジェクトや他の関連する AWS リソースオブジェクトを AWS CloudFormation テンプレートファイルで定義して作成するほうが便利な場合があります。この場合、オブジェクトの作成プロセスを自動化できます。

AWS CloudFormation では、JSON (JavaScript Object Notation) または YAML (YAML Ain't Markup Language) のどちらかの簡略化された構文を使用し、AWS リソースの作成を記述します。AWS CloudFormation テンプレートを使用して、データベース、テーブル、パーティション、クローラー、分類子、接続などのデータカタログオブジェクトを定義できます。ジョブ、トリガー、開発エンドポイントなどの ETL オブジェクトを定義することもできます。必要なすべての AWS リソースを記述するテンプレートを作成すると、これらのリソースが AWS CloudFormation で自動的にプロビジョニングおよび設定されます。

詳細については、AWS CloudFormation ユーザーガイドの「What Is AWS CloudFormation?」および「Working with AWS CloudFormation Templates」を参照してください。

管理者として AWS Glue と互換性がある AWS CloudFormationテンプレートを使用する場合は、依存する AWS CloudFormation および AWS のサービスとアクションにアクセス権を付与する必要があります。AWS CloudFormation リソースを作成するアクセス権限を付与するには、次のポリシーを、AWS CloudFormation を使用するユーザーにアタッチします。

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

次の表は、AWS CloudFormation テンプレートで自動的に実行できるアクションの一覧です。AWS CloudFormation テンプレートに追加できる AWS リソースタイプやプロパティタイプに関する情報へのリンクが含まれています。

AWS Glue リソース:	AWS CloudFormation テンプレート	AWS Glueサンプル
分類子	AWS::Glue::Classifier	Grok 分類子、JSON 分類子、XML 分類子
Connection	AWS::Glue::Connection	MySQL 接続
Crawler	AWS::Glue::Crawler	Amazon S3 クローラー、MySQL クローラー
データベース	AWS::Glue::Database	空のデータベース、テーブルを含むデータベース
開発エンドポイント	AWS::Glue::DevEndpoint	開発エンドポイント
ジョブ	AWS::Glue::Job	Amazon S3 ジョブ、JDBC ジョブ
機械学習変換	AWS::Glue::MLTransform	機械学習変換
データ品質ルールセット	AWS::Glue::DataQualityRuleset	データ品質ルールセット、EventBridge スケジューラを使用したデータ品質ルールセット
パーティション	AWS::Glue::Partition	テーブルのパーティション
テーブル	AWS::Glue::Table	データベース内のテーブル
Trigger トリガー)	AWS::Glue::Trigger	オンデマンドのトリガー、スケジュールされたトリガー、条件付きトリガー

使用を開始するには、以下のサンプルテンプレートを独自のメタデータを使用してカスタマイズします。次に AWS CloudFormation コンソールを使用して AWS CloudFormation スタックを作成し、AWS Glue および関連サービスにオブジェクトを追加します。AWS Glue オブジェクトの多くのフィールドはオプションです。これらのテンプレートは、AWS Glue オブジェクトの使用や機能に必須または必要なフィールドを示しています。

AWS CloudFormation テンプレートは JSON 形式または YAML 形式のいずれかで使用できます。以下の例では、読みやすい YAML を使用しています。各例には、テンプレートで定義されている値を説明するコメント (#) が含まれています。

AWS CloudFormation テンプレートには `Parameters` セクションを含めることができます。このセクションは、サンプルテキストを編集して変更できます。または、YAML ファイルを AWS CloudFormation コンソールに送信してスタックを作成するときに変更できます。テンプレートの `Resources` セクションには、AWS Glue および関連オブジェクトの定義が含まれています。AWS CloudFormation テンプレートの構文定義には、詳細なプロパティ構文を含むプロパティが含まれている場合があります。AWS Glue オブジェクトを作成するために、すべてのプロパティが必要になるわけではありません。これらのサンプルは、AWS Glue オブジェクトを作成するための一般的なプロパティの値の例です。

AWS Glue データベース用のサンプル AWS CloudFormation テンプレート

Data Catalog の AWS Glue データベースにはメタデータテーブルが含まれています。このデータベースは、構成するプロパティが非常に少なく、AWS CloudFormation テンプレートを使用して Data Catalog に作成できます。次のサンプルテンプレートでは、使用を開始する方法と、AWS Glue での AWS CloudFormation スタックの使い方を示します。このサンプルテンプレートで作成されるリソースは `cfn-mysampledatabase` というデータベースのみです。このデータベースは、サンプルのテキストを編集するか、YAML の送信時に AWS CloudFormation コンソールで値を変更することで、変更できます。

次に示すのは、AWS Glue データベースを作成するための一般的なプロパティの値の例です。AWS CloudFormation 用の AWS Glue データベーステンプレートの詳細については、「AWS::Glue::Database」を参照してください。

```
---
AWSTemplateFormatVersion: '2010-09-09'
```

```
# Sample CloudFormation template in YAML to demonstrate creating a database named
 mysampledatabase
# The metadata created in the Data Catalog points to the flights public S3 bucket
#
# Parameters section contains names that are substituted in the Resources section
# These parameters are the names the resources created in the Data Catalog
Parameters:
  CFNDatabaseName:
    Type: String
    Default: cfn-mysampledatabse

# Resources section defines metadata for the Data Catalog
Resources:
# Create an AWS Glue database
  CFNDatabaseFlights:
    Type: AWS::Glue::Database
    Properties:
      # The database is created in the Data Catalog for your account
      CatalogId: !Ref AWS::AccountId
      DatabaseInput:
        # The name of the database is defined in the Parameters section above
        Name: !Ref CFNDatabaseName
        Description: Database to hold tables for flights data
        LocationUri: s3://crawler-public-us-east-1/flight/2016/csv/
        #Parameters: Leave AWS database parameters blank
```

AWS Glue データベース、テーブル、およびパーティション用のサンプル AWS CloudFormation テンプレート

AWS Glue テーブルには、ETL スクリプトで処理するデータの構造と場所を定義するメタデータが含まれています。テーブル内に、データを並列処理するためのパーティションを定義できます。パーティションは、キーを使用して定義したデータのチャンクです。たとえば、キーとして月を使用すると、1 月のすべてのデータが同じパーティションに含まれます。AWS Glue では、データベースにテーブルを含め、テーブルにパーティションを含めることができます。

次のサンプルでは、AWS CloudFormation テンプレートを使用して、データベース、テーブル、およびパーティションを事前設定する方法を示します。元のデータ形式は `csv` であり、カンマ (,) で区切られています。テーブルを作成するには事前にデータベースが必要であり、パーティションを作成するには事前にテーブルが必要であるため、テンプレートでは `DependsOn` ステートメントを使用して、これらのオブジェクトの作成時に相互の依存関係を定義します。

次のサンプルの値では、一般に利用可能な Amazon S3 バケットのフライトデータを含むテーブルを定義します。わかりやすくするために、データのいくつかの列と 1 つのパーティションキーのみが定義されています。4 つのパーティションも Data Catalog に定義されています。基本データのストレージを記述するいくつかのフィールドも StorageDescriptor フィールドに示されています。

```
---
AWSTemplateFormatVersion: '2010-09-09'
# Sample CloudFormation template in YAML to demonstrate creating a database, a table,
 and partitions
# The metadata created in the Data Catalog points to the flights public S3 bucket
#
# Parameters substituted in the Resources section
# These parameters are names of the resources created in the Data Catalog
Parameters:
  CFNDatabaseName:
    Type: String
    Default: cfn-database-flights-1
  CFNTableName1:
    Type: String
    Default: cfn-manual-table-flights-1
# Resources to create metadata in the Data Catalog
Resources:
###
# Create an AWS Glue database
  CFNDatabaseFlights:
    Type: AWS::Glue::Database
    Properties:
      CatalogId: !Ref AWS::AccountId
      DatabaseInput:
        Name: !Ref CFNDatabaseName
        Description: Database to hold tables for flights data
###
# Create an AWS Glue table
  CFNTableFlights:
    # Creating the table waits for the database to be created
    DependsOn: CFNDatabaseFlights
    Type: AWS::Glue::Table
    Properties:
      CatalogId: !Ref AWS::AccountId
      DatabaseName: !Ref CFNDatabaseName
      TableInput:
        Name: !Ref CFNTableName1
```

```
        Description: Define the first few columns of the flights table
        TableType: EXTERNAL_TABLE
        Parameters: {
    "classification": "csv"
  }
#       ViewExpandedText: String
        PartitionKeys:
        # Data is partitioned by month
        - Name: mon
          Type: bigint
        StorageDescriptor:
          OutputFormat: org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat
          Columns:
          - Name: year
            Type: bigint
          - Name: quarter
            Type: bigint
          - Name: month
            Type: bigint
          - Name: day_of_month
            Type: bigint
          InputFormat: org.apache.hadoop.mapred.TextInputFormat
          Location: s3://crawler-public-us-east-1/flight/2016/csv/
          SerdeInfo:
            Parameters:
              field.delim: ","
            SerializationLibrary: org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
# Partition 1
# Create an AWS Glue partition
  CFNPartitionMon1:
    DependsOn: CFNTableFlights
    Type: AWS::Glue::Partition
    Properties:
      CatalogId: !Ref AWS::AccountId
      DatabaseName: !Ref CFNDatabaseName
      TableName: !Ref CFNTableName1
      PartitionInput:
        Values:
        - 1
        StorageDescriptor:
          OutputFormat: org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat
          Columns:
          - Name: mon
            Type: bigint
```

```
          InputFormat: org.apache.hadoop.mapred.TextInputFormat
          Location: s3://crawler-public-us-east-1/flight/2016/csv/mon=1/
          SerdeInfo:
            Parameters:
              field.delim: ","
            SerializationLibrary: org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
# Partition 2
# Create an AWS Glue partition
  CFNPartitionMon2:
    DependsOn: CFNTableFlights
    Type: AWS::Glue::Partition
    Properties:
      CatalogId: !Ref AWS::AccountId
      DatabaseName: !Ref CFNDatabaseName
      TableName: !Ref CFNTableName1
      PartitionInput:
        Values:
        - 2
        StorageDescriptor:
          OutputFormat: org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat
          Columns:
          - Name: mon
            Type: bigint
          InputFormat: org.apache.hadoop.mapred.TextInputFormat
          Location: s3://crawler-public-us-east-1/flight/2016/csv/mon=2/
          SerdeInfo:
            Parameters:
              field.delim: ","
            SerializationLibrary: org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
# Partition 3
# Create an AWS Glue partition
  CFNPartitionMon3:
    DependsOn: CFNTableFlights
    Type: AWS::Glue::Partition
    Properties:
      CatalogId: !Ref AWS::AccountId
      DatabaseName: !Ref CFNDatabaseName
      TableName: !Ref CFNTableName1
      PartitionInput:
        Values:
        - 3
        StorageDescriptor:
          OutputFormat: org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat
          Columns:
```

```
          - Name: mon
            Type: bigint
          InputFormat: org.apache.hadoop.mapred.TextInputFormat
          Location: s3://crawler-public-us-east-1/flight/2016/csv/mon=3/
          SerdeInfo:
            Parameters:
              field.delim: ","
            SerializationLibrary: org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
# Partition 4
# Create an AWS Glue partition
  CFNPartitionMon4:
    DependsOn: CFNTableFlights
    Type: AWS::Glue::Partition
    Properties:
      CatalogId: !Ref AWS::AccountId
      DatabaseName: !Ref CFNDatabaseName
      TableName: !Ref CFNTableName1
      PartitionInput:
        Values:
        - 4
        StorageDescriptor:
          OutputFormat: org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat
          Columns:
          - Name: mon
            Type: bigint
          InputFormat: org.apache.hadoop.mapred.TextInputFormat
          Location: s3://crawler-public-us-east-1/flight/2016/csv/mon=4/
          SerdeInfo:
            Parameters:
              field.delim: ","
            SerializationLibrary: org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
```

AWS Glue Grok 分類子用のサンプル AWS CloudFormation テンプレート

AWS Glue 分類子はデータのスキーマを決定します。1 つのタイプのカスタム分類子では、grok パターンを使用してデータをマッチングします。パターンがマッチすると、カスタム分類子ではテーブルのスキーマを作成し、分類子の定義に設定された値に `classification` を設定します。

このサンプルで作成する分類子では、`message` という列が 1 つあるスキーマを作成し、分類を `greedy` に設定します。

```
---
AWSTemplateFormatVersion: '2010-09-09'
# Sample CFN YAML to demonstrate creating a classifier
#
# Parameters section contains names that are substituted in the Resources section
# These parameters are the names the resources created in the Data Catalog
Parameters:

# The name of the classifier to be created
  CFNClassifierName:
    Type: String
    Default: cfn-classifier-grok-one-column-1

#
#
# Resources section defines metadata for the Data Catalog
Resources:
# Create classifier that uses grok pattern to put all data in one column and classifies
 it as "greedy".
  CFNClassifierFlights:
    Type: AWS::Glue::Classifier
    Properties:
      GrokClassifier:
        #Grok classifier that puts all data in one column
        Name: !Ref CFNClassifierName
        Classification: greedy

        GrokPattern: "%{GREEDYDATA:message}"
        #CustomPatterns: none
```

AWS Glue JSON 分類子用のサンプル AWS CloudFormation テンプレート

AWS Glue 分類子はデータのスキーマを決定します。1 つのタイプのカスタム分類子では、分類のための分類子の JSON データを定義する `JsonPath` 文字列を使用します。「Writing JsonPath Custom Classifiers」に記載されているように、AWS Glue では `JsonPath` の演算子のサブセットがサポートされています。

パターンが一致すると、カスタム識別子を使用してテーブルのスキーマが作成されます。

このサンプルでは、オブジェクトの Records3 配列で各スキーマを使用してスキーマを作成する識別子を作成します。

```
---
AWSTemplateFormatVersion: '2010-09-09'
# Sample CFN YAML to demonstrate creating a JSON classifier
#
# Parameters section contains names that are substituted in the Resources section
# These parameters are the names the resources created in the Data Catalog
Parameters:

# The name of the classifier to be created
  CFNClassifierName:
    Type: String
    Default: cfn-classifier-json-one-column-1

#
#
# Resources section defines metadata for the Data Catalog
Resources:
# Create classifier that uses a JSON pattern.
  CFNClassifierFlights:
    Type: AWS::Glue::Classifier
    Properties:
      JSONClassifier:
        #JSON classifier
        Name: !Ref CFNClassifierName
        JsonPath: $.Records3[*]
```

AWS Glue XML 分類子用のサンプル AWS CloudFormation テンプレート

AWS Glue 分類子はデータのスキーマを決定します。1 つのタイプのカスタム分類子は、XML タグを指定して、解析中の XML ドキュメントの各レコードを含む要素を指定します。パターンがマッチすると、カスタム分類子ではテーブルのスキーマを作成し、分類子の定義に設定された値に classification を設定します。

このサンプルで作成する分類子では、Record タグの各レコードでスキーマを作成し、分類を XML に設定します。

```
---
AWSTemplateFormatVersion: '2010-09-09'
# Sample CFN YAML to demonstrate creating an XML classifier
#
# Parameters section contains names that are substituted in the Resources section
# These parameters are the names the resources created in the Data Catalog
Parameters:

# The name of the classifier to be created
  CFNClassifierName:
    Type: String
    Default: cfn-classifier-xml-one-column-1

#
#
# Resources section defines metadata for the Data Catalog
Resources:
# Create classifier that uses the XML pattern and classifies it as "XML".
  CFNClassifierFlights:
    Type: AWS::Glue::Classifier
    Properties:
      XMLClassifier:
        #XML classifier
        Name: !Ref CFNClassifierName
        Classification: XML
        RowTag: <Records>
```

Amazon S3 の AWS Glue クローラー用のサンプル AWS CloudFormation テンプレート

AWS Glue クローラーでは、データに対応するメタデータテーブルをデータカタログに作成します。次に、これらのテーブル定義を ETL ジョブのソースおよびターゲットとして使用できます。

このサンプルでは、クローラー、必要な IAM ロール、および AWS Glue データベースをデータカタログに作成します。このクローラーを実行すると、クローラーは IAM ロールを取得し、公開フライトデータ用のテーブルをデータベースに作成します。テーブルは、プレフィックス `cfn_sample_1_` を使用して作成されます。このテンプレートで作成された IAM ロールでは、グローバルのアクセス許可が付与されるので、カスタムロールを作成する必要がある場合もあります。

この分類子で定義されるカスタム分類子はありません。AWS Glue の組み込み分類子がデフォルトで使用されます。

このサンプルを AWS CloudFormation コンソールに送信する場合は、IAM ロールを作成することを確認する必要があります。

```
---
AWSTemplateFormatVersion: '2010-09-09'
# Sample CFN YAML to demonstrate creating a crawler
#
# Parameters section contains names that are substituted in the Resources section
# These parameters are the names the resources created in the Data Catalog
Parameters:

# The name of the crawler to be created
  CFNCrawlerName:
    Type: String
    Default: cfn-crawler-flights-1
  CFNDatabaseName:
    Type: String
    Default: cfn-database-flights-1
  CFNTablePrefixName:
    Type: String
    Default: cfn_sample_1_
#
#
# Resources section defines metadata for the Data Catalog
Resources:
#Create IAM Role assumed by the crawler. For demonstration, this role is given all
 permissions.
  CFNRoleFlights:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: "2012-10-17"
        Statement:
          -
            Effect: "Allow"
            Principal:
              Service:
                - "glue.amazonaws.com"
            Action:
```

```
              - "sts:AssumeRole"
      Path: "/"
      Policies:
        -
          PolicyName: "root"
          PolicyDocument:
            Version: "2012-10-17"
            Statement:
              -
                Effect: "Allow"
                Action: "*"
                Resource: "*"
 # Create a database to contain tables created by the crawler
  CFNDatabaseFlights:
    Type: AWS::Glue::Database
    Properties:
      CatalogId: !Ref AWS::AccountId
      DatabaseInput:
        Name: !Ref CFNDatabaseName
        Description: "AWS Glue container to hold metadata tables for the flights
 crawler"
 #Create a crawler to crawl the flights data on a public S3 bucket
  CFNCrawlerFlights:
    Type: AWS::Glue::Crawler
    Properties:
      Name: !Ref CFNCrawlerName
      Role: !GetAtt CFNRoleFlights.Arn
      #Classifiers: none, use the default classifier
      Description: AWS Glue crawler to crawl flights data
      #Schedule: none, use default run-on-demand
      DatabaseName: !Ref CFNDatabaseName
      Targets:
        S3Targets:
          # Public S3 bucket with the flights data
          - Path: "s3://crawler-public-us-east-1/flight/2016/csv"
      TablePrefix: !Ref CFNTablePrefixName
      SchemaChangePolicy:
        UpdateBehavior: "UPDATE_IN_DATABASE"
        DeleteBehavior: "LOG"
      Configuration: "{\"Version\":1.0,\"CrawlerOutput\":{\"Partitions\":
{\"AddOrUpdateBehavior\":\"InheritFromTable\"},\"Tables\":{\"AddOrUpdateBehavior\":
\"MergeNewColumns\"}}}"
```

AWS Glue 接続用のサンプル AWS CloudFormation テンプレート

Data Catalog の AWS Glue 接続には、JDBC データベースに接続するために必要な JDBC およびネットワーク情報が含まれています。この情報は、JDBC データベースに接続して ETL ジョブをクロールまたは実行するときに使用されます。

このサンプルでは、Amazon RDS MySQL データベース (devdb) への接続を作成します。この接続を使用する場合は、IAM ロール、データベース認証情報、およびネットワーク接続の値も指定する必要があります。テンプレートの必須フィールドの詳細を参照してください。

```
---
AWSTemplateFormatVersion: '2010-09-09'
# Sample CFN YAML to demonstrate creating a connection
#
# Parameters section contains names that are substituted in the Resources section
# These parameters are the names the resources created in the Data Catalog
Parameters:

# The name of the connection to be created
  CFNConnectionName:
    Type: String
    Default: cfn-connection-mysql-flights-1
  CFNJDBCString:
    Type: String
    Default: "jdbc:mysql://xxx-mysql.yyyyyyyyyyyyyy.us-east-1.rds.amazonaws.com:3306/
devdb"
  CFNJDBCUser:
    Type: String
    Default: "master"
  CFNJDBCPassword:
    Type: String
    Default: "12345678"
    NoEcho: true
#
#
# Resources section defines metadata for the Data Catalog
Resources:
  CFNConnectionMySQL:
    Type: AWS::Glue::Connection
    Properties:
      CatalogId: !Ref AWS::AccountId
```

```
      ConnectionInput:
        Description: "Connect to MySQL database."
        ConnectionType: "JDBC"
        #MatchCriteria: none
        PhysicalConnectionRequirements:
          AvailabilityZone: "us-east-1d"
          SecurityGroupIdList:
           - "sg-7d52b812"
          SubnetId: "subnet-84f326ee"
        ConnectionProperties: {
          "JDBC_CONNECTION_URL": !Ref CFNJDBCString,
          "USERNAME": !Ref CFNJDBCUser,
          "PASSWORD": !Ref CFNJDBCPassword
        }
        Name: !Ref CFNConnectionName
```

JDBC の AWS Glue クローラー用のサンプル AWS CloudFormation テンプレート

AWS Glue クローラーでは、データに対応するメタデータテーブルをデータカタログに作成します。次に、これらのテーブル定義を ETL ジョブのソースおよびターゲットとして使用できます。

このサンプルでは、クローラー、必要な IAM ロール、および AWS Glue データベースをデータカタログに作成します。このクローラーを実行すると、クローラーは IAM ロールを取得し、MySQL データベースに保存されている公開フライトデータ用のテーブルをデータベースに作成します。テーブルは、プレフィックス `cfn_jdbc_1_` を使用して作成されます。このテンプレートで作成された IAM ロールでは、グローバルのアクセス許可が付与されるので、カスタムロールを作成する必要がある場合もあります。JDBC データに対してはカスタム分類子を定義できません。AWS Glue の組み込み分類子がデフォルトで使用されます。

このサンプルを AWS CloudFormation コンソールに送信する場合は、IAM ロールを作成することを確認する必要があります。

```
---
AWSTemplateFormatVersion: '2010-09-09'
# Sample CFN YAML to demonstrate creating a crawler
#
# Parameters section contains names that are substituted in the Resources section
# These parameters are the names the resources created in the Data Catalog
```

```
Parameters:

# The name of the crawler to be created
  CFNCrawlerName:
    Type: String
    Default: cfn-crawler-jdbc-flights-1
# The name of the database to be created to contain tables
  CFNDatabaseName:
    Type: String
    Default: cfn-database-jdbc-flights-1
# The prefix for all tables crawled and created
  CFNTablePrefixName:
    Type: String
    Default: cfn_jdbc_1_
# The name of the existing connection to the MySQL database
  CFNConnectionName:
    Type: String
    Default: cfn-connection-mysql-flights-1
# The name of the JDBC path (database/schema/table) with wildcard (%) to crawl
  CFNJDBCPath:
    Type: String
    Default: saldev/%
#
#
# Resources section defines metadata for the Data Catalog
Resources:
#Create IAM Role assumed by the crawler. For demonstration, this role is given all
 permissions.
  CFNRoleFlights:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: "2012-10-17"
        Statement:
          -
            Effect: "Allow"
            Principal:
              Service:
                - "glue.amazonaws.com"
            Action:
              - "sts:AssumeRole"
      Path: "/"
      Policies:
        -
```

```
          PolicyName: "root"
          PolicyDocument:
            Version: "2012-10-17"
            Statement:
              -
                Effect: "Allow"
                Action: "*"
                Resource: "*"
 # Create a database to contain tables created by the crawler
  CFNDatabaseFlights:
    Type: AWS::Glue::Database
    Properties:
      CatalogId: !Ref AWS::AccountId
      DatabaseInput:
        Name: !Ref CFNDatabaseName
        Description: "AWS Glue container to hold metadata tables for the flights
 crawler"
 #Create a crawler to crawl the flights data in MySQL database
  CFNCrawlerFlights:
    Type: AWS::Glue::Crawler
    Properties:
      Name: !Ref CFNCrawlerName
      Role: !GetAtt CFNRoleFlights.Arn
      #Classifiers: none, use the default classifier
      Description: AWS Glue crawler to crawl flights data
      #Schedule: none, use default run-on-demand
      DatabaseName: !Ref CFNDatabaseName
      Targets:
        JdbcTargets:
          # JDBC MySQL database with the flights data
          - ConnectionName: !Ref CFNConnectionName
            Path: !Ref CFNJDBCPath
          #Exclusions: none
      TablePrefix: !Ref CFNTablePrefixName
      SchemaChangePolicy:
        UpdateBehavior: "UPDATE_IN_DATABASE"
        DeleteBehavior: "LOG"
    Configuration: "{\"Version\":1.0,\"CrawlerOutput\":{\"Partitions\":
{\"AddOrUpdateBehavior\":\"InheritFromTable\"},\"Tables\":{\"AddOrUpdateBehavior\":
\"MergeNewColumns\"}}}"
```

Amazon S3 から Amazon S3 への AWS Glue ジョブのサンプル AWS CloudFormation テンプレート

Data Catalog の AWS Glue ジョブには、AWS Glue でスクリプトを実行するために必要なパラメータ値が含まれています。

このサンプルで作成するジョブでは、Amazon S3 バケットのフライトデータを csv 形式で読み取り、Amazon S3 の Parquet ファイルに書き込みます。このジョブで実行するスクリプトは既存している必要があります。環境に応じた ETL スクリプトを AWS Glue コンソールで生成できます。このジョブ実行時に、適切なアクセス許可が設定された IAM ロールも指定する必要があります。

テンプレートには、一般的なパラメータ値が示されています。たとえば、AllocatedCapacity (DPU) はデフォルトで 5 になります。

```
---
AWSTemplateFormatVersion: '2010-09-09'
# Sample CFN YAML to demonstrate creating a job using the public flights S3 table in a
 public bucket
#
# Parameters section contains names that are substituted in the Resources section
# These parameters are the names the resources created in the Data Catalog
Parameters:

# The name of the job to be created
  CFNJobName:
    Type: String
    Default: cfn-job-S3-to-S3-2
# The name of the IAM role that the job assumes. It must have access to data, script,
 temporary directory
  CFNIAMRoleName:
    Type: String
    Default: AWSGlueServiceRoleGA
# The S3 path where the script for this job is located
  CFNScriptLocation:
    Type: String
    Default: s3://aws-glue-scripts-123456789012-us-east-1/myid/sal-job-test2
#
#
# Resources section defines metadata for the Data Catalog
Resources:
```

```
# Create job to run script which accesses flightscsv table and write to S3 file as
 parquet.
# The script already exists and is called by this job
  CFNJobFlights:
    Type: AWS::Glue::Job
    Properties:
      Role: !Ref CFNIAMRoleName
      #DefaultArguments: JSON object
      # If script written in Scala, then set DefaultArguments={'--job-language';
 'scala', '--class': 'your scala class'}
      #Connections:  No connection needed for S3 to S3 job
      #  ConnectionsList
      #MaxRetries: Double
      Description: Job created with CloudFormation
      #LogUri: String
      Command:
        Name: glueetl
        ScriptLocation: !Ref CFNScriptLocation
             # for access to directories use proper IAM role with permission to buckets
 and folders that begin with "aws-glue-"
             # script uses temp directory from job definition if required (temp
 directory not used S3 to S3)
             # script defines target for output as s3://aws-glue-target/sal
      AllocatedCapacity: 5
      ExecutionProperty:
        MaxConcurrentRuns: 1
      Name: !Ref CFNJobName
```

Amazon S3 に書き込む JDBC の AWS Glue ジョブ用のサンプル AWS CloudFormation テンプレート

Data Catalog の AWS Glue ジョブには、AWS Glue でスクリプトを実行するために必要なパラメータ値が含まれています。

このサンプルで作成するジョブでは、`cfn-connection-mysql-flights-1` という接続で定義された MySQL JDBC データベースからフライトデータを読み取り、Amazon S3 の Parquet ファイルに書き込みます。このジョブで実行するスクリプトは既存している必要があります。環境に応じた ETL スクリプトを AWS Glue コンソールで生成できます。このジョブ実行時に、適切なアクセス許可が設定された IAM ロールも指定する必要があります。

テンプレートには、一般的なパラメータ値が示されています。たとえば、AllocatedCapacity (DPU) はデフォルトで 5 になります。

```
---
AWSTemplateFormatVersion: '2010-09-09'
# Sample CFN YAML to demonstrate creating a job using a MySQL JDBC DB with the flights
 data to an S3 file
#
# Parameters section contains names that are substituted in the Resources section
# These parameters are the names the resources created in the Data Catalog
Parameters:

# The name of the job to be created
  CFNJobName:
    Type: String
    Default: cfn-job-JDBC-to-S3-1
# The name of the IAM role that the job assumes. It must have access to data, script,
 temporary directory
  CFNIAMRoleName:
    Type: String
    Default: AWSGlueServiceRoleGA
# The S3 path where the script for this job is located
  CFNScriptLocation:
    Type: String
    Default: s3://aws-glue-scripts-123456789012-us-east-1/myid/sal-job-dec4a
# The name of the connection used for JDBC data source
  CFNConnectionName:
    Type: String
    Default: cfn-connection-mysql-flights-1
#
#
# Resources section defines metadata for the Data Catalog
Resources:
# Create job to run script which accesses JDBC flights table via a connection and write
 to S3 file as parquet.
# The script already exists and is called by this job
  CFNJobFlights:
    Type: AWS::Glue::Job
    Properties:
      Role: !Ref CFNIAMRoleName
      #DefaultArguments: JSON object
```

```
      # For example, if required by script, set temporary directory as
 DefaultArguments={'--TempDir'; 's3://aws-glue-temporary-xyc/sal'}
      Connections:
        Connections:
        - !Ref CFNConnectionName
      #MaxRetries: Double
      Description: Job created with CloudFormation using existing script
      #LogUri: String
      Command:
        Name: glueetl
        ScriptLocation: !Ref CFNScriptLocation
             # for access to directories use proper IAM role with permission to buckets
 and folders that begin with "aws-glue-"
             # if required, script defines temp directory as argument TempDir and used
 in script like redshift_tmp_dir = args["TempDir"]
             # script defines target for output as s3://aws-glue-target/sal
      AllocatedCapacity: 5
      ExecutionProperty:
        MaxConcurrentRuns: 1
      Name: !Ref CFNJobName
```

AWS Glue オンデマンドトリガー用のサンプル AWS CloudFormation テンプレート

Data Catalog の AWS Glue トリガーには、トリガーの発動によってジョブ実行を開始するために必要なパラメータ値が含まれています。オンデマンドトリガーは、このトリガーを有効にしたときに発生します。

このサンプルで作成するオンデマンドトリガーでは、cfn-job-S3-to-S3-1 という 1 つのジョブを開始します。

```
---
AWSTemplateFormatVersion: '2010-09-09'
# Sample CFN YAML to demonstrate creating an on-demand trigger
#
# Parameters section contains names that are substituted in the Resources section
# These parameters are the names the resources created in the Data Catalog
Parameters:
  # The existing job to be started by this trigger
  CFNJobName:
```

```
    Type: String
    Default: cfn-job-S3-to-S3-1
  # The name of the trigger to be created
  CFNTriggerName:
    Type: String
    Default: cfn-trigger-ondemand-flights-1
#
# Resources section defines metadata for the Data Catalog
# Sample CFN YAML to demonstrate creating an on-demand trigger for a job
Resources:
# Create trigger to run an existing job (CFNJobName) on an on-demand schedule.
  CFNTriggerSample:
    Type: AWS::Glue::Trigger
    Properties:
      Name:
        Ref: CFNTriggerName
      Description: Trigger created with CloudFormation
      Type: ON_DEMAND
      Actions:
        - JobName: !Ref CFNJobName
        # Arguments: JSON object
      #Schedule:
      #Predicate:
```

AWS Glue のスケジュールされたトリガー用のサンプル AWS CloudFormation テンプレート

Data Catalog の AWS Glue トリガーには、トリガーの発動によってジョブ実行を開始するために必要なパラメータ値が含まれています。スケジュールされたトリガーは、このトリガーを有効にして cron タイマーがポップすると、発生します。

このサンプルで作成するスケジュールされたトリガーでは、cfn-job-S3-to-S3-1 という 1 つのジョブを開始します。このタイマーは、平日の 10 分ごとにジョブを実行する cron 式です。

```
---
AWSTemplateFormatVersion: '2010-09-09'
# Sample CFN YAML to demonstrate creating a scheduled trigger
#
# Parameters section contains names that are substituted in the Resources section
# These parameters are the names the resources created in the Data Catalog
```

```
Parameters:
  # The existing job to be started by this trigger
  CFNJobName:
    Type: String
    Default: cfn-job-S3-to-S3-1
  # The name of the trigger to be created
  CFNTriggerName:
    Type: String
    Default: cfn-trigger-scheduled-flights-1
#
# Resources section defines metadata for the Data Catalog
# Sample CFN YAML to demonstrate creating a scheduled trigger for a job
#
Resources:
# Create trigger to run an existing job (CFNJobName) on a cron schedule.
  TriggerSample1CFN:
    Type: AWS::Glue::Trigger
    Properties:
      Name:
        Ref: CFNTriggerName
      Description: Trigger created with CloudFormation
      Type: SCHEDULED
      Actions:
        - JobName: !Ref CFNJobName
        # Arguments: JSON object
      # # Run the trigger every 10 minutes on Monday to Friday
      Schedule: cron(0/10 * ? * MON-FRI *)
      #Predicate:
```

AWS Glue の条件付きトリガー用のサンプル AWS CloudFormation テンプレート

Data Catalog の AWS Glue トリガーには、トリガーの発動によってジョブ実行を開始するために必要なパラメータ値が含まれています。条件付きトリガーは、このトリガーを有効にして、その条件が満たされる (例: ジョブが正常に完了する) と、発生します。

このサンプルで作成する条件付きトリガーでは、cfn-job-S3-to-S3-1 という 1 つのジョブを開始します。このジョブは、cfn-job-S3-to-S3-2 というジョブが正常に完了すると、開始されます。

```
---
AWSTemplateFormatVersion: '2010-09-09'
# Sample CFN YAML to demonstrate creating a conditional trigger for a job, which starts
 when another job completes
#
# Parameters section contains names that are substituted in the Resources section
# These parameters are the names the resources created in the Data Catalog
Parameters:
  # The existing job to be started by this trigger
  CFNJobName:
    Type: String
    Default: cfn-job-S3-to-S3-1
  # The existing job that when it finishes causes trigger to fire
  CFNJobName2:
    Type: String
    Default: cfn-job-S3-to-S3-2
  # The name of the trigger to be created
  CFNTriggerName:
    Type: String
    Default: cfn-trigger-conditional-1
#
Resources:
# Create trigger to run an existing job (CFNJobName) when another job completes
 (CFNJobName2).
  CFNTriggerSample:
    Type: AWS::Glue::Trigger
    Properties:
      Name:
        Ref: CFNTriggerName
      Description: Trigger created with CloudFormation
      Type: CONDITIONAL
      Actions:
        - JobName: !Ref CFNJobName
        # Arguments: JSON object
      #Schedule: none
      Predicate:
        #Value for Logical is required if more than 1 job listed in Conditions
        Logical: AND
        Conditions:
          - LogicalOperator: EQUALS
            JobName: !Ref CFNJobName2
            State: SUCCEEDED
```

AWS Glue の開発エンドポイント用のサンプル AWS CloudFormation テンプレート

AWS Glue 機械学習変換は、データをクレンジングするためのカスタム変換です。現在、FindMatches という名前の変換が 1 つあります。FindMatches 変換を使用すると、レコードに共通の一意の識別子がなく、正確に一致するフィールドがない場合でも、データセット内の重複レコードまたは一致するレコードを識別できます。

このサンプルでは、機械学習変換を作成します。機械学習変換の作成に必要なパラメータの詳細については、「AWS Lake Formation FindMatches によるレコードのマッチング」を参照してください。

```
---
AWSTemplateFormatVersion: '2010-09-09'
# Sample CFN YAML to demonstrate creating a machine learning transform
#
# Resources section defines metadata for the machine learning transform
Resources:
  MyMLTransform:
    Type: "AWS::Glue::MLTransform"
    Condition: "isGlueMLGARegion"
    Properties:
      Name: !Sub "MyTransform"
      Description: "The bestest transform ever"
      Role: !ImportValue MyMLTransformUserRole
      GlueVersion: "1.0"
      WorkerType: "Standard"
      NumberOfWorkers: 5
      Timeout: 120
      MaxRetries: 1
      InputRecordTables:
        GlueTables:
          - DatabaseName: !ImportValue MyMLTransformDatabase
            TableName: !ImportValue MyMLTransformTable
      TransformParameters:
        TransformType: "FIND_MATCHES"
        FindMatchesParameters:
          PrimaryKeyColumnName: "testcolumn"
          PrecisionRecallTradeoff: 0.5
          AccuracyCostTradeoff: 0.5
          EnforceProvidedLabels: True
      Tags:
```

```
        key1: "value1"
        key2: "value2"
      TransformEncryption:
        TaskRunSecurityConfigurationName: !ImportValue
 MyMLTransformSecurityConfiguration
        MLUserDataEncryption:
          MLUserDataEncryptionMode: "SSE-KMS"
          KmsKeyId: !ImportValue MyMLTransformEncryptionKey
```

AWS Glue Data Quality ルールセット用のサンプル AWS CloudFormation テンプレート

AWS Glue Data Quality ルールセットには、データカタログ内のテーブルで評価できるルールが含まれています。ルールセットをターゲットテーブルに配置したら、データカタログにアクセスして、ルールセット内のルールに対してデータを実行する評価を実行できます。これらのルールは、行数の評価からデータの参照整合性の評価までさまざまです。

次のサンプルは、指定されたターゲットテーブルにさまざまなルールを含むルールセットを作成する CloudFormation テンプレートです。

```
AWSTemplateFormatVersion: '2010-09-09'
# Sample CFN YAML to demonstrate creating a DataQualityRuleset
#
# Parameters section contains names that are substituted in the Resources section
# These parameters are the names the resources created in the Data Catalog
Parameters:

  # The name of the ruleset to be created
  RulesetName:
    Type: String
    Default: "CFNRulesetName"
  RulesetDescription:
    Type: String
    Default: "CFN DataQualityRuleset"
  # Rules that will be associated with this ruleset
  Rules:
    Type: String
    Default: 'Rules = [
        RowCount > 100,
        IsUnique "id",
        IsComplete "nametype"
```

```
        ]'
  # Name of database and table within Data Catalog which the ruleset will
  # be applied too
  DatabaseName:
    Type: String
    Default: "ExampleDatabaseName"
  TableName:
    Type: String
    Default: "ExampleTableName"

# Resources section defines metadata for the Data Catalog
Resources:
  # Creates a Data Quality ruleset under specified rules
  DQRuleset:
    Type: AWS::Glue::DataQualityRuleset
    Properties:
      Name: !Ref RulesetName
      Description: !Ref RulesetDescription
      # The String within rules must be formatted in DQDL, a language
      # used specifically to make rules
      Ruleset: !Ref Rules
      # The targeted table must exist within Data Catalog alongside
      # the correct database
      TargetTable:
        DatabaseName: !Ref DatabaseName
        TableName: !Ref TableName
```

EventBridge スケジューラを使用する AWS Glue Data Quality ルールセット用の AWS CloudFormation テンプレート

AWS Glue Data Quality ルールセットには、データカタログ内のテーブルで評価できるルールが含まれています。ルールセットをターゲットテーブルに配置したら、データカタログにアクセスして、ルールセット内のルールに対してデータを実行する評価を実行できます。手動でデータカタログにアクセスしてルールセットを評価する代わりに、CloudFormation テンプレート内に EventBridge スケジューラを追加して、これらのルールセットの評価を一定間隔でスケジュールすることもできます。

次のサンプルは、前述のルールセットを 5 分ごとに評価する、Data Quality ルールセットと EventBridge スケジューラを作成する CloudFormation テンプレートです。

```
AWSTemplateFormatVersion: '2010-09-09'
# Sample CFN YAML to demonstrate creating a DataQualityRuleset
```

```
#
# Parameters section contains names that are substituted in the Resources section
# These parameters are the names the resources created in the Data Catalog
Parameters:

  # The name of the ruleset to be created
  RulesetName:
    Type: String
    Default: "CFNRulesetName"
  # Rules that will be associated with this Ruleset
  Rules:
    Type: String
    Default: 'Rules = [
        RowCount > 100,
        IsUnique "id",
        IsComplete "nametype"
        ]'
  # The name of the Schedule to be created
  ScheduleName:
    Type: String
    Default: "ScheduleDQRulsetEvaluation"
  # This expression determines the rate at which the Schedule will evaluate
  # your data using the above ruleset
  ScheduleRate:
    Type: String
    Default: "rate(5 minutes)"
  # The Request that being sent must match the details of the Data Quality Ruleset
  ScheduleRequest:
    Type: String
    Default: '
        { "DataSource": { "GlueTable": { "DatabaseName": "ExampleDatabaseName",
         "TableName": "ExampleTableName" } },
         "Role": "role/AWSGlueServiceRoleDefault",
          "RulesetNames": [ ""CFNRulesetName"" ] }
        '

# Resources section defines metadata for the Data Catalog
Resources:
  # Creates a Data Quality ruleset under specified rules
  DQRuleset:
    Type: AWS::Glue::DataQualityRuleset
    Properties:
      Name: !Ref RulesetName
      Description: "CFN DataQualityRuleset"
```

```
      # The String within rules must be formatted in DQDL, a language
      # used specifically to make rules
      Ruleset: !Ref Rules
      # The targeted table must exist within Data Catalog alongside
      # the correct database
      TargetTable:
        DatabaseName: "ExampleDatabaseName"
        TableName: "ExampleTableName"
  # Create a Scheduler to schedule evaluation runs on the above ruleset
  ScheduleDQEval:
    Type: AWS::Scheduler::Schedule
    Properties:
      Name: !Ref ScheduleName
      Description: "Schedule DataQualityRuleset Evaluations"
      FlexibleTimeWindow:
        Mode: "OFF"
      ScheduleExpression: !Ref ScheduleRate
      ScheduleExpressionTimezone: "America/New_York"
      State: "ENABLED"
      Target:
        # The ARN is the API that will be run, since we want to evaluate our ruleset
        # we want this specific ARN
        Arn: "arn:aws:scheduler:::aws-sdk:glue:startDataQualityRulesetEvaluationRun"
        # Your RoleArn must have approval to schedule
        RoleArn: "arn:aws:iam::123456789012:role/AWSGlueServiceRoleDefault"
        # This is the Request that is being sent to the Arn
        Input: '
        { "DataSource": { "GlueTable": { "DatabaseName": "sampledb", "TableName":
"meteorite" } },
          "Role": "role/AWSGlueServiceRoleDefault",
           "RulesetNames": [ "TestCFN" ] }
        '
```

AWS Glue の開発エンドポイント用のサンプル AWS CloudFormation テンプレート

AWS Glue の開発エンドポイントは、AWS Glue スクリプトの開発およびテストに使用できる環境です。

このサンプルで作成する開発エンドポイントでは、正常な作成に最低限必要なネットワークパラメータ値を使用します。開発エンドポイントの設定に必要なパラメータの詳細については、「AWS Glue のための開発用ネットワークの設定」を参照してください。

開発エンドポイントを作成するには、既存の IAM ロール ARN (Amazon リソースネーム) を指定します。開発エンドポイントでノートブックサーバーを作成する場合は、有効な RSA パブリックキーを指定し、対応するプライベートキーを使用可能な状態に保持します。

> **Note**
>
> 作成した開発エンドポイントに関連付けられているすべてのノートブックサーバーを管理します。したがって、開発エンドポイントを削除した場合、ノートブックサーバーを削除するには AWS CloudFormation コンソールで AWS CloudFormation スタックを削除する必要があります。

```
---
AWSTemplateFormatVersion: '2010-09-09'
# Sample CFN YAML to demonstrate creating a development endpoint
#
# Parameters section contains names that are substituted in the Resources section
# These parameters are the names the resources created in the Data Catalog
Parameters:

# The name of the crawler to be created
  CFNEndpointName:
    Type: String
    Default: cfn-devendpoint-1
  CFNIAMRoleArn:
    Type: String
    Default: arn:aws:iam::123456789012/role/AWSGlueServiceRoleGA
#
#
# Resources section defines metadata for the Data Catalog
Resources:
  CFNDevEndpoint:
    Type: AWS::Glue::DevEndpoint
    Properties:
      EndpointName: !Ref CFNEndpointName
      #ExtraJarsS3Path: String
```

```
    #ExtraPythonLibsS3Path: String
    NumberOfNodes: 5
    PublicKey: ssh-rsa public.....key myuserid-key
    RoleArn: !Ref CFNIAMRoleArn
    SecurityGroupIds:
      - sg-64986c0b
    SubnetId: subnet-c67cccac
```

AWS Glue プログラミングガイド

スクリプトには、ソースからデータを抽出して、変換し、ターゲットにロードするコードが含まれています。AWS Glue はジョブを開始するときにスクリプトを実行します。

AWS Glue ETL スクリプトは Python または Scala で記述されます。すべてのジョブタイプは Python で記述できますが、AWS Glue for Spark ジョブは Scala でも記述できます。AWS Glue Studio でジョブのソースコードロジックを自動で生成するときに、スクリプトが作成されます。このスクリプトを編集するか、または、独自のスクリプトを指定して ETL 作業を処理することができます。

独自のカスタムスクリプトの提供

スクリプトは、AWS Glue で抽出、変換、ロード (ETL) 作業を実行します。スクリプトは、自動でジョブのソースコードロジックを生成するときに作成されます。この生成されたスクリプトを編集することもできますし、独自のカスタムスクリプトを指定することもできます。

AWS Glue で独自のカスタムスクリプトを提供するには、以下の一般的な手順に従います。

1. AWS Management Consoleにサインインし、AWS Glue コンソール (https://console.aws.amazon.com/glue/) を開きます。
2. [ETL Jobs] タブをクリックしてから、[ジョブを作成する] セクションを開きます。[script editor] オプションをクリックします。
3. [This job runs] (このジョブ実行) で、次のいずれかを選択します。
   - [Create a new script with boilerplate code]
   - [Upload and edit an existing script]
4. [ジョブの詳細] ページで、カスタムスクリプトを実行するのに必要な [IAM ロール] をクリックします。詳細については、「AWS Glue のアイデンティティとアクセスの管理」を参照してください。
5. スクリプトが参照する接続を選択します。これらのオブジェクトは、目的の JDBC データストアに接続するために必要です。

   Elastic Network Interface は、Virtual Private Cloud (VPC) でインスタンスにアタッチできる、仮想ネットワークインターフェイスです。スクリプトで使用されているデータストアに接続するのに必要な Elastic Network Interface を選択します。

6. ジョブタイプ固有のパラメータなどの追加設定を行います。ジョブタイプの設定の詳細については、「AWS Glue Studio でビジュアル ETL ジョブを作成する」セクションを参照してください。
7. [スクリプト] タブで、カスタムスクリプトを貼り付けるか、記述します。

カスタムスクリプトを記述するプロセスのガイドとして、このセクションの内容を使用してください。

AWS Glue におけるジョブ追加の詳細については、「AWS Glue Studio でビジュアル ETL ジョブを作成する」を参照してください。

詳細な手順については、 コンソールの [Add jobAWS Glue] (ジョブ追加) チュートリアルを参照してください。

Spark スクリプトのプログラミング

AWS Glue では、テストや実行に加えて、抽出、変換、ロード (ETL) スクリプトの書き込みや自動生成を簡単に行えます。このセクションでは、AWS Glue に導入された Apache Spark の拡張機能について説明し、ETL スクリプトを Python と Scala でコーディングし実行する方法の例を示します。

> **Important**
>
> AWS Glue の異なるバージョンでは、Apache Spark の異なるバージョンがサポートされます。カスタムスクリプトは、サポートされている Apache Spark バージョンと互換性がある必要があります。AWS Glue バージョンの詳細については、「Glue version job property」を参照してください。

トピック

- チュートリアル: AWS Glue for Spark スクリプトの記述
- PySpark で AWS Glue ETL スクリプトをプログラムする
- Scala での AWS Glue ETL スクリプトのプログラミング
- AWS Glue for Spark ETL スクリプトのプログラミングの機能と最適化

チュートリアル: AWS Glue for Spark スクリプトの記述

このチュートリアルでは、AWS Glue スクリプトを記述するプロセスの概要を説明します。スクリプトを、ジョブを使用してスケジュールに従って実行したり、インタラクティブセッションを使用してインタラクティブに実行したりできます。ジョブの詳細については、「AWS Glue Studio でビジュアル ETL ジョブを作成する」を参照してください。詳細については、「the section called “AWS Glue インタラクティブセッションの概要”」を参照してください。

AWS Glue Studio ビジュアルエディターは、AWS Glue ジョブを構築するためのグラフィカルなノーコードのインターフェイスを提供します。AWSGlue スクリプトはビジュアルジョブをバックアップします。Glue スクリプトを使用すると、Apache Spark プログラムを操作するための拡張された一連のツールにアクセスできます。AWS Glue スクリプト内からの抽出、変換、ロード (ETL) ワークフローを円滑にする AWS Glue ライブラリだけでなく、Spark の ネイティブ API にもアクセスできます。

このチュートリアルでは、駐車チケットのデータセットを抽出、変換、ロードします。この作業を行うスクリプトは、AWS Glue Studio ビジュアルエディタを紹介する AWS ビッグデータブログの「AWS Glue Studio で ETL を簡単にする」で作成したものと形式および機能が同じです。このスクリプトをジョブ内で実行することで、AWS Glue ETL スクリプトがどのように機能するかをビジュアルジョブと比較して確認できます。これにより、ビジュアルジョブでまだ使用可能ではない追加機能を使用する準備ができます。

このチュートリアルでは、Python 言語とライブラリを使用しています。Scala でも同様の機能が使用可能です。このチュートリアルを完了すると、Scala のサンプルスクリプトを生成して検査し、Scala での AWS Glue ETL スクリプトの記述プロセスを実行する方法を理解できるようになります。

前提条件

このチュートリアルには、次のような前提条件があります。

- AWS CloudFormation テンプレートを実行するように指示する AWS Glue Studio のブログ記事と同じ前提条件。

  このテンプレートは、AWS Glue データカタログを使用して、`s3://aws-bigdata-blog/artifacts/gluestudio/` で使用できる駐車券データセットを管理します。これによって参照される以下のリソースが作成されます。

- AWS Glue StudioRole - AWS Glue ジョブを実行する IAM ロール

- AWS Glue StudioAmazon S3Bucket - ブログ関連のファイルを保存する Amazon S3 バケットの名前
- AWS Glue StudioTicketsYYZDB – AWS Glue データカタログデータベース
- AWS Glue StudioTableTickets - ソースとして使用するデータカタログテーブル
- AWS Glue StudioTableTrials - ソースとして使用するデータカタログテーブル
- AWS Glue StudioParkingTicketCount - 送信先として使用するデータカタログテーブル
- AWS Glue Studio のブログ記事で生成されたスクリプト。ブログ記事が変更された場合、スクリプトは次のテキストからも入手できます。

サンプルスクリプトの生成

AWS Glue Studio ビジュアルエディタを強力なコード生成ツールとして使用することで、記述するスクリプトの足場を構築できます。サンプルスクリプトは、このツールを使用して作成します。

これらのステップをスキップしたい場合は、提供されているスクリプトを使用してください。

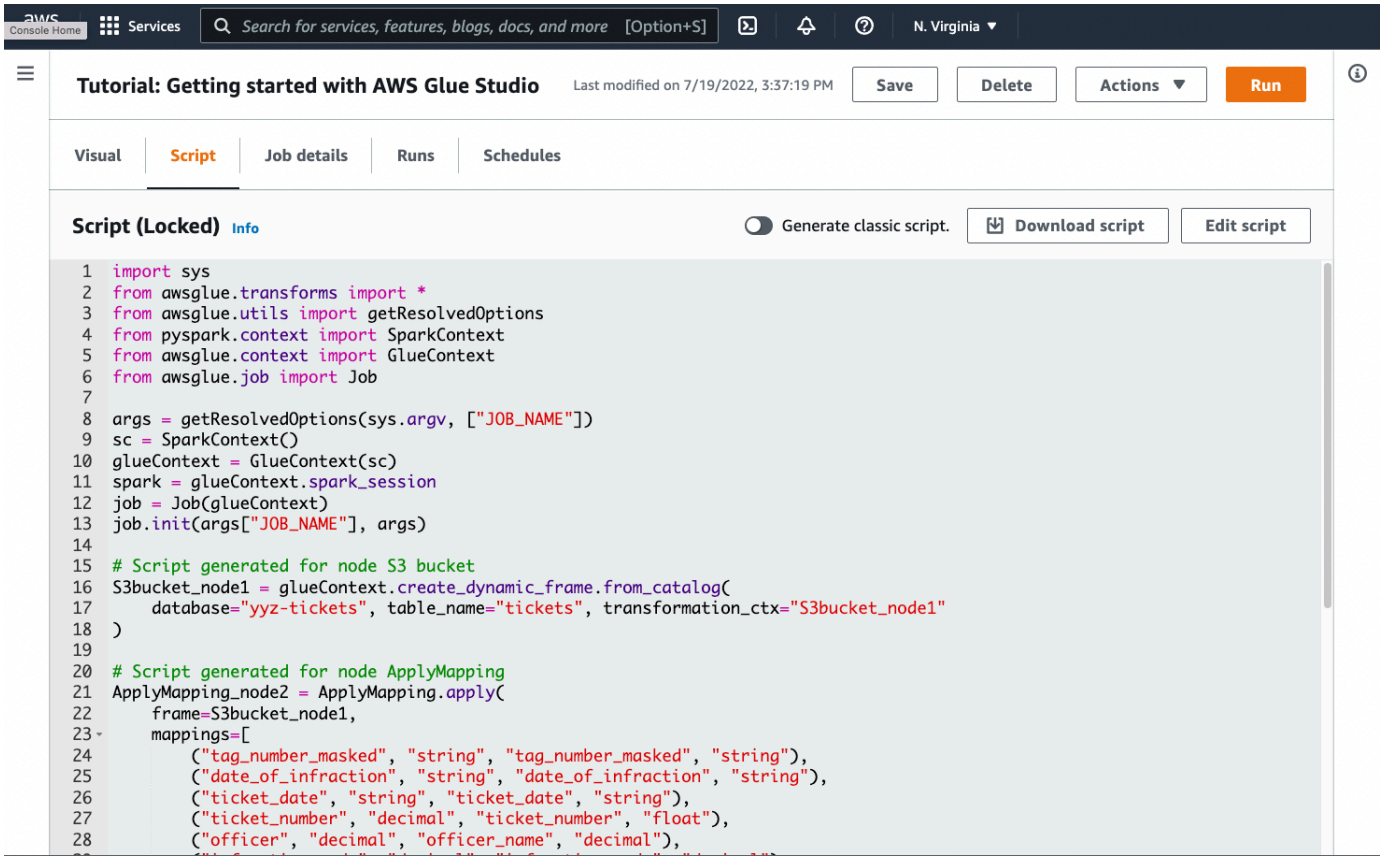
このチュートリアルでのサンプルスクリプト

```
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job

args = getResolvedOptions(sys.argv, ["JOB_NAME"])
sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
job.init(args["JOB_NAME"], args)

# Script generated for node S3 bucket
S3bucket_node1 = glueContext.create_dynamic_frame.from_catalog(
    database="yyz-tickets", table_name="tickets", transformation_ctx="S3bucket_node1"
)

# Script generated for node ApplyMapping
ApplyMapping_node2 = ApplyMapping.apply(
    frame=S3bucket_node1,
```

```
    mappings=[
        ("tag_number_masked", "string", "tag_number_masked", "string"),
        ("date_of_infraction", "string", "date_of_infraction", "string"),
        ("ticket_date", "string", "ticket_date", "string"),
        ("ticket_number", "decimal", "ticket_number", "float"),
        ("officer", "decimal", "officer_name", "decimal"),
        ("infraction_code", "decimal", "infraction_code", "decimal"),
        ("infraction_description", "string", "infraction_description", "string"),
        ("set_fine_amount", "decimal", "set_fine_amount", "float"),
        ("time_of_infraction", "decimal", "time_of_infraction", "decimal"),
    ],
    transformation_ctx="ApplyMapping_node2",
)

# Script generated for node S3 bucket
S3bucket_node3 = glueContext.write_dynamic_frame.from_options(
    frame=ApplyMapping_node2,
    connection_type="s3",
    format="glueparquet",
    connection_options={"path": "s3://DOC-EXAMPLE-BUCKET", "partitionKeys": []},
    format_options={"compression": "gzip"},
    transformation_ctx="S3bucket_node3",
)

job.commit()
```

サンプルスクリプトを生成するには

1. AWS Glue Studio のチュートリアルをを完了します。このチュートリアルを完了するには、「サンプルジョブから AWS Glue Studio でジョブの作成」を参照してください。
2. 次のスクリーンショットに示すように、ジョブページの [Script] (スクリプト) タブに移動します。



3. [Script] (スクリプト) タブのすべての内容をコピーします。[Job details] (Job の詳細) でスクリプト言語を設定することで、生成するコードを Python と Scala の間で自由に切り替えることができます。

ステップ 1. ジョブを作成してスクリプトを貼り付ける

このステップでは、AWS Management Consoleで AWS Glue ジョブを作成します。これにより、AWS Glue でスクリプトを実行するのに必要な構成がセットアップされます。また、スクリプトを保存および編集するための場所も作成されます。

ジョブを作成するには

1. AWS Management Consoleで、AWS Glue のランディングページに移動します。
2. サイドナビゲーションペインで、[Jobs] (ジョブ) を選択します。
3. [Create job] (ジョブの作成) で [Spark script editor] (Spark スクリプトエディタ) を選択してから、[Create] (作成) を選択します。
4. オプション — スクリプトの全文を [Script] (スクリプト) ペインに貼り付けます。この代わりに、チュートリアルに従うこともできます。

ステップ 2. AWS Glue ライブラリをインポートする

スクリプトの外部で定義されたコードや設定とやり取りするには、スクリプトの設定が必要です。この作業は、AWS Glue Studioの背後で実行されます。

このステップでは、次のアクションを実行します。

- `GlueContext` オブジェクトをインポートして初期化します。スクリプト作成の観点から見ると、このインポートが最も重要なものとなります。これにより、ETL スクリプトの開始点となる、ソースデータセットとターゲットデータセットを定義するための標準的な方法が表示されます。`GlueContext` クラスの詳細については、「GlueContext クラス」を参照してください。
- `SparkContext` および `SparkSession` を初期化します。これにより、AWS Glue ジョブ 内部で Spark エンジンを使用するための設定が可能になります。入門用 AWS Glue スクリプトの中では、これらを直接使用する必要はありません
- `getResolvedOptions` を呼び出し、スクリプト内でジョブ引数を使用するための準備を行います。ジョブパラメータの解決方法の詳細については、「the section called “getResolvedOptions”」を参照してください。
- `Job` を初期化します。`Job` オブジェクトにより設定が行われ、AWS Glue のさまざまなオプションの機能の状態が追跡されます。スクリプトは `Job` オブジェクトがなくても実行できますが、ベストプラクティスとして、後にこの機能が統合された場合に混乱が生じないようにオブジェクトを初期化しておきます。

  ジョブのブックマークは、これらの機能の 1 つであり、このチュートリアルではオプションで設定可能です。ジョブのブックマークに関する説明は、「the section called “オプション — ジョブのブックマークを有効にする”」のセクションで確認できます。

この手順では、次のコードを作成します。このコードは、生成されるサンプルスクリプトの一部です。

```
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job

args = getResolvedOptions(sys.argv, ["JOB_NAME"])
sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
```

```
job = Job(glueContext)
job.init(args["JOB_NAME"], args)
```

AWS Glue ライブラリをインポートするには

- コードのこのセクションをコピーして、[Script] (スクリプト) エディタに貼り付けます。

> **Note**
>
> コードのコピーは技術的に推奨されないと思われているかもしれません。ただ、このチュートリアルでは、AWS Glue ETL スクリプト全体で主要な変数に一貫して名前を付けることが推奨されるため、これを提案しています。

ステップ 3. ソースからデータを抽出する

すべての ETL プロセスにおいて、変更するソースデータセットを最初に定義する必要があります。AWS Glue Studio ビジュアルエディタでは、[Source] (ソース) ノードを作成することで、この情報を指定します。

このステップでは、AWS Glue データカタログで設定されたソースからデータを抽出する `create_dynamic_frame.from_catalog` メソッドに `database` および `table_name` を提供します。

`GlueContext` オブジェクトは、前のステップで初期化が済んでいます。このオブジェクトを使用して、ソースの設定に使用される `create_dynamic_frame.from_catalog` などのメソッドを検索します。

この手順では、`create_dynamic_frame.from_catalog` を使用して次のコードを記述します。このコードは、生成されるサンプルスクリプトの一部です。

```
S3bucket_node1 = glueContext.create_dynamic_frame.from_catalog(
    database="yyz-tickets", table_name="tickets", transformation_ctx="S3bucket_node1"
    )
```

ソースからデータを抽出するには

1. AWS Glue データカタログで定義されているソースからデータを抽出するには、ドキュメントを参照し `GlueContext` にあるメソッドを見つけます。これらの方法については、「the section

called “GlueContext”」を参照してください。create_dynamic_frame.from_catalog メソッドを選択します。glueContext でこのメソッドを呼び出します。

2. create_dynamic_frame.from_catalog に関するドキュメントで確認してください。このメソッドには、database および table_name パラメータが必要です。create_dynamic_frame.from_catalog に対し、必要なパラメータを指定します。

   AWS Glue データカタログは前提条件セクションで設定されており、これにはソースデータの場所と形式に関する情報が保存されています。スクリプトに情報を直接提供する必要はありません。

3. オプション — ジョブのブックマークをサポートするには、このメソッドに transformation_ctx パラメータを提供します。ジョブのブックマークに関する説明は、「the section called “オプション — ジョブのブックマークを有効にする”」のセクションで確認できます。

> **Note**
>
> データ抽出用の一般的なメソッド
> the section called “create_dynamic_frame_from_catalog” は、AWS Glue データカタログ内のテーブルへの接続に使用されます
> ソースの構造と場所を記述する設定でジョブを直接提供する必要がある場合は、「the section called “create_dynamic_frame_from_options”」メソッドを参照してください。create_dynamic_frame.from_catalog を使用する場合に比べ、データを説明する詳細なパラメータの指定が必要になります。
> 必要なパラメータを特定するには、format_options および connection_parameters に関する補足文書を参照してください。スクリプトに対しソースデータの形式に関する情報を指定する方法については、「the section called “データ形式に関するオプション”」を参照してください。スクリプトに対しソースデータの場所に関する情報を指定する方法については、「the section called “接続パラメータ”」を参照してください。
> ストリーミングソースから情報を読み取っている場合は、the section called “create_data_frame_from_catalog” または the section called “create_data_frame_from_options” メソッドを介して、ソースの情報をジョブに提供します。これらのメソッドは Apache Spark DataFrames を返すことに注意してください。
> リファレンスドキュメントでは create_dynamic_frame_from_catalog を使用している部分のために、ここで生成されるコードは create_dynamic_frame.from_catalog を呼び出しています。これらのメソッドは、最終的には同じコードを呼び出すもので、よりク

リーンなコードを作成できるようにインクルードされています。これは、aws-glue-libs にある Python のラッパーのソースを表示して確認できます。

ステップ 4. AWS Glue を使用してデータを変換する

ETL プロセスで抽出したソースデータについては、その変更方法を規定する必要があります。この情報を指定するには、AWS Glue Studio ビジュアルエディタで [Transform] (変換) ノードを作成します。

このステップでは、希望する現在のフィールドの名前とタイプのマップを `ApplyMapping` メソッドに提供し、`DynamicFrame` を変換します。

以下の変換を実行します。

- 4 つの `location` および `province`キーを削除します。
- `officer` の名前を `officer_name` に変更します。
- `ticket_number` および `set_fine_amount` の型を `float` に変更します。

`create_dynamic_frame.from_catalog` により、`DynamicFrame` オブジェクトが提供されます。`DynamicFrame` は、AWS Glue 内のデータセットを表します。AWSGlue での変換は、`DynamicFrames` を変更する処理です。

> **Note**
>
> `DynamicFrame` について
> `DynamicFrame` とは抽象化の 1 つであり、データ内のエントリの名前とタイプに関する記述を、データセットに接続できるようにします。Apache Spark にも同様の抽象化が存在し、DataFrame と呼ばれています。DataFrames の説明については、「Spark SQL Guide」(Spark SQL ガイド) を参照してください。
> `DynamicFrames` を使用すると、データセットのスキーマを動的に記述できます。例えば、あるデータセットに価格列が含まれており、一部のエントリは価格を文字列として保存し、他のエントリは価格を double 型で格納している場合を考えます。AWSGlue は、オンザフライでスキーマを計算し、各行の自己記述型レコードを作成します。
> 一貫性のないフィールド (価格など) は、そのフレーム用のスキーマ内で、型 (`ChoiceType`) を使用しながら明示的に表されます。一貫性のないフィールドに対応するには、`DropFields` を使用して削除するか、`ResolveChoice` を使用して解決します。これら

は、DynamicFrame で使用可能な変換です。その後、writeDynamicFrame を使用して、データをデータレイクに再度書き込むことができます。
DynamicFrame クラスにあるメソッドからは、多くの同様な変換を呼び出すことができます。これにより、スクリプトが読みやすくなります。DynamicFrame の詳細については、「the section called “DynamicFrame”」を参照してください。

この手順では、ApplyMapping を使用して次のコードを記述します。このコードは、生成されるサンプルスクリプトの一部です。

```
ApplyMapping_node2 = ApplyMapping.apply(
    frame=S3bucket_node1,
    mappings=[
        ("tag_number_masked", "string", "tag_number_masked", "string"),
        ("date_of_infraction", "string", "date_of_infraction", "string"),
        ("ticket_date", "string", "ticket_date", "string"),
        ("ticket_number", "decimal", "ticket_number", "float"),
        ("officer", "decimal", "officer_name", "decimal"),
        ("infraction_code", "decimal", "infraction_code", "decimal"),
        ("infraction_description", "string", "infraction_description", "string"),
        ("set_fine_amount", "decimal", "set_fine_amount", "float"),
        ("time_of_infraction", "decimal", "time_of_infraction", "decimal"),
    ],
    transformation_ctx="ApplyMapping_node2",
)
```

AWS Glue でデータを変換するには

1. フィールドを変更や削除するための変換を特定するには、ドキュメントを参照してください。詳細については、「the section called “GlueTransform”」を参照してください。ApplyMapping 変換を選択します。ApplyMapping の詳細については、「the section called “ApplyMapping”」を参照してください。ApplyMapping 変換オブジェクトから apply を呼び出します。

   > **Note**
   >
   > ApplyMapping とは
   > ApplyMapping は DynamicFrame にアクセスし、それを変換します。これにより、フィールドの変換を表すタプルのリスト、つまり「マッピング」を取得します。最初の 2 つのタプル要素 (フィールドの名前と型) は、フレーム内のフィールドを識別するために使用されます。その次にある 2 つのパラメータも、フィールドの名前と型です。

ApplyMapping は、ソースフィールドをターゲット名に変換し、新しい DynamicFrame を入力して返します。提供されていないフィールドは戻り値から削除されます。
apply を呼び出す代わりに、DynamicFrame にある apply_mapping メソッドを使用して同様の変換を呼び出すと、さらにスムーズで読みやすいコードを作成できます。詳細については、「the section called “apply_mapping”」を参照してください。

2. 必要なパラメータを特定するには、ApplyMapping のドキュメントを参照してください。「the section called “ApplyMapping”」を参照してください。このメソッドでは、frame および mappings パラメータが必要です。ApplyMapping に対し、必要なパラメータを指定します。
3. オプション — ジョブのブックマークをサポートするには、メソッドに transformation_ctx を提供します。ジョブのブックマークに関する説明は、「the section called “オプション — ジョブのブックマークを有効にする”」のセクションで確認できます。

> **Note**
>
> Apache Spark の機能
> ジョブ内のETLワークフローを効率化するための変換機能が提供されています。また、より一般的な目的のために構築された、ジョブ内の Spark プログラムで使用できるライブラリにもアクセスできます。これらを使用するには、DynamicFrame と DataFrame の間で変換します。
> the section called “toDF” を使用して DataFrame を作成できます。次に、DataFrame で使用可能なメソッドを使用してデータセットを変換できます。これらのメソッドの詳細については、「DataFrame」を参照してください。その後、フレームをターゲットにロードする AWS Glue オペレーションを使用するため、the section called “fromDF” によって変換を元に戻すことができます。

ステップ 5. ターゲットにデータをロードする

通常、変換後のデータは、そのソースとは別の場所に保存します。この操作を実行するには、AWS Glue Studio ビジュアルエディタで、[Target] (ターゲット) ノードを作成します

このステップでは、データを Amazon S3 のターゲットバケットにロードするための write_dynamic_frame.from_options メソッド connection_type、connection_options、format、format_options を提供します。

ステップ 1 において、GlueContext オブジェクトは初期化済みです。AWS Glue では、ソースと同様、ターゲットの設定に使用されるメソッドを提供しています。

この手順では、write_dynamic_frame.from_options を使用して次のコードを記述します。このコードは、生成されるサンプルスクリプトの一部です。

```
S3bucket_node3 = glueContext.write_dynamic_frame.from_options(
    frame=ApplyMapping_node2,
    connection_type="s3",
    format="glueparquet",
    connection_options={"path": "s3://amzn-s3-demo-bucket", "partitionKeys": []},
    format_options={"compression": "gzip"},
    transformation_ctx="S3bucket_node3",
    )
```

データをターゲットにロードするには

1. ターゲットの Amazon S3 バケットに対しデータをロードするメソッドは、ドキュメントで確認してください。これらの方法については、「the section called “GlueContext”」を参照してください。the section called “write_dynamic_frame_from_options” メソッドを選択します。glueContext でこのメソッドを呼び出します。

   > **Note**
   >
   > データロード用の一般的なメソッド
   > write_dynamic_frame.from_options は、データのロードに使用される最も一般的なメソッドです。このメソッドでは、AWS Glueで使用可能なすべてのターゲットがサポートされています。
   > AWS Glue 接続で定義された JDBC ターゲットに書き込む場合は、the section called “write_dynamic_frame_from_jdbc_conf” メソッドを使用します。AWSGlue 接続に、データソースへの接続方法に関する情報が保存されます。これにより、connection_options でそれらの情報を入力する必要がなくなります。ただし、dbtable を提供するためには、引き続き connection_options を使用する必要があります。
   > write_dynamic_frame.from_catalog は、データロード用として、一般的なメソッドではありません。このメソッドは基盤データセットを更新せずに AWS Glue データカタログを更新します。これは、基盤のデータセットを変更する他のプロセスと組み合わ

せて使用するメソッドです。詳細については、「the section called “スキーマを更新し、新規パーティションを追加する”」を参照してください。

2. the section called “write_dynamic_frame_from_options” に関するドキュメントで確認してください。このメソッドで
は、frame、connection_type、format、connection_options、format_options が必要です。glueContext でこのメソッドを呼び出します。

   a. 必要なパラメータを特定するには、format_options および format に関する補足文書を参照してください。データ形式の説明については、「the section called “データ形式に関するオプション”」を参照してください。

   b. 必要なパラメータを特定するには、connection_type および connection_options に関する補足文書を参照してください。接続の説明については、「the section called “接続パラメータ”」を参照してください。

   c. write_dynamic_frame.from_options に対し、必要なパラメータを指定します。このメソッドの設定は、create_dynamic_frame.from_options と似ています。

3. オプション — ジョブのブックマークをサポートするには、transformation_ctx を write_dynamic_frame.from_options に提供します。ジョブのブックマークに関する説明は、「the section called “オプション — ジョブのブックマークを有効にする”」のセクションで確認できます。

ステップ 6。Job オブジェクトのコミット

Job オブジェクトは、ステップ 1 で初期化してあります。ジョブのブックマークを使用する場合など、特定のオプション機能が適切に機能するためにこれを必要とする場合は、スクリプトの最後にライフサイクルを手動で終了する必要がある場合があります。この作業は、AWS Glue Studioの背後で実行されます。

このステップでは、Job オブジェクトの commit メソッドを呼び出します。

この手順では、次のコードを作成します。このコードは、生成されるサンプルスクリプトの一部です。

```
job.commit()
```

Job オブジェクトをコミットするには

1. まだ行っていない場合は、前のセクションで概説したオプションの手順を実行して `transformation_ctx` を含めます。
2. `commit` を呼び出します。

オプション — ジョブのブックマークを有効にする

これまでのすべてのステップでは、`transformation_ctx` パラメータを設定することを促しています。これは、ジョブのブックマークと呼ばれる機能に関連した設定です。

ジョブのブックマークを使用すると、以前の作業を簡単に追跡できるデータセットに対して定期的に実行されるジョブによって、時間とコストを節約できます。ジョブのブックマークは、前回の実行のデータセット全体に関する AWS Glue 変換の進捗状況を追跡します。以前の実行が終了した場所を追跡することで、AWS Glue は未処理の行に対してのみ作業できます。ブックマークの詳細については、「the section called “ジョブのブックマークを使用した処理済みデータの追跡”」を参照してください。

前の例で説明したように、ジョブのブックマークを有効にするには、始めに、提供されている関数に `transformation_ctx` ステートメントを追加します。ジョブのブックマークの状態は実行後も保持されます。`transformation_ctx` パラメータは、その状態にアクセスするために使用されるキーです。これらのステートメント単体では機能を持ちません。また、ジョブのための設定内で、その機能を有効にする必要があります。

この手順では、AWS Management Console を使用してジョブのブックマークを有効にします。

ジョブのブックマークを有効にするには

1. 対象となるジョブの、[Job details] (ジョブの詳細) セクションに移動します。
2. [Job bookmark] (ジョブのブックマーク) で [Enable] (有効化) 設定します。

ステップ 7。コードをジョブとして実行する

このステップでは、チュートリアルを正常に完了したことを確認するために、ジョブを実行します。これは、AWS Glue Studio ビジュアルエディタの中で、ボタンをクリックするだけで完了します。

コードをジョブとして実行するには

1. タイトルバーで　[Untitled job] (無題のジョブ) 選択し、ジョブ名を編集あるいは設定します。

2. [Job details] (ジョブの詳細) タブを開きます。ジョブに [IAM Role] (IAM ロール) を割り当てます。AWS Glue Studio チュートリアルの前提条件にある、AWS CloudFormation テンプレートで作成したものを使用できます。このチュートリアルを完了していれば、通常は、`AWS Glue StudioRole` のように表示されています。
3. [Save] (保存) を選択して、スクリプトを保存します。
4. [Run] (実行) を選択して、ジョブを実行します。
5. [Runs] (実行) タブに移動して、ジョブが完了したことを確認します。
6. `write_dynamic_frame.from_options` のターゲットである *`amzn-s3-demo-bucket`* に移動します。想定どおりに出力されていることを確認します。

ジョブの設定と管理の詳細については、「the section called “独自のカスタムスクリプトの提供”」を参照してください。

詳細情報

Apache Spark のライブラリとメソッドを、AWS Glue スクリプトの中で使用することが可能です。Spark のドキュメントを参照することで、これらの付属ライブラリで何が行えるのか把握できます。詳細については、「examples section of the Spark source repository」(Spark ソースリポジトリのサンプルセクション) を参照してください。

AWS Glue 2.0+ には、デフォルトで、いくつかの一般的な Python ライブラリが含まれています。Scala または Python 環境の AWS Glue ジョブに独自の依存関係をロードするためのメカニズムもあります。Python の依存関係については、「the section called “Python ライブラリ”」を参照してください。

Python で AWS Glue の機能を使用する方法の例の詳細については、「the section called “Python サンプル”」を参照してください。Scala と Python ジョブには同等の機能があるため、Python の例を基にして、同様の処理を Scala で行う方法についても考察することが可能です。

PySpark で AWS Glue ETL スクリプトをプログラムする

AWS Glue 用の Python コード例とユーティリティは、GitHub ウェブサイトの AWS Glue サンプルリポジトリで見つかります。

AWS Glue での Python の使用

AWS Glue では、抽出、変換、ロード (ETL) ジョブをスクリプト化するための PySpark Python 拡張言語機能がサポートされています。このセクションでは、ETL スクリプトと AWS Glue API で Python を使用する方法について説明します。

- AWS Glue で Python を使用するためのセットアップ
- Python での AWS Glue API の呼び出し
- AWS Glue での Python ライブラリの使用
- AWS Glue Python コードサンプル

AWS Glue PySpark 拡張機能

AWS Glue には、PySpark (Python の関連言語) 向けに、以下の拡張機能が追加されています。

- `getResolvedOptions` を使用して、パラメータにアクセスする
- PySpark 拡張子型
- DynamicFrame クラス
- DynamicFrameCollection クラス
- DynamicFrameWriter クラス
- DynamicFrameReader クラス
- GlueContext クラス

AWS Glue PySpark 変換

AWS Glue には、PySpark での ETL オペレーションに使用するため、次の変換クラスが用意されています。

- GlueTransform 基本クラス
- ApplyMapping クラス
- DropFields クラス
- DropNullFields クラス
- ErrorsAsDynamicFrame クラス
- FillMissingValues クラス

- フィルタクラス
- FindIncrementalMatches クラス
- FindMatches クラス
- FlatMap クラス
- Join クラス
- マップクラス
- MapToCollection クラス
- mergeDynamicFrame
- クラスの関連付け
- RenameField クラス
- ResolveChoice クラス
- SelectFields クラス
- SelectFromCollection クラス
- スピゴットクラス
- SplitFields クラス
- SplitRows クラス
- Unbox クラス
- UnnestFrame クラス

AWS Glue で Python を使用するためのセットアップ

Python を使用して Spark ジョブ用の ETL スクリプトを開発します。ETL ジョブでサポートされている Python のバージョンは、ジョブが使用する AWS Glue バージョンによって異なります。AWS Glue バージョンの詳細については、「Glue version job property」を参照してください。

AWS Glue で Python を使用するためのシステムを設定するには

Python をインストールするには、以下の手順を実行して AWS Glue API を呼び出せるようにします。

1. Python がインストールされていない場合は、Python.org のダウンロードページ からダウンロードしてインストールします。
2. AWS CLI のドキュメントで説明されているように AWS Command Line Interface (AWS CLI) をインストールします。

AWS CLI は、Python を使用するのに直接必要なわけではありません。ただし、インストールおよび設定を行うと、アカウントの認証情報を使用して AWS を設定し、その動作を確認するのに便利です。

3. Boto 3 クイックスタートで説明されているように、AWS SDK for Python (Boto 3) をインストールします。

   Boto 3 リソース API は AWS Glue にはまだ使用できません。現時点では、Boto 3 クライアント API のみ使用することができます。

   Boto 3 の詳細については、「AWS SDK for Python (Boto3) Getting Started」を参照してください。

Python のコード例や AWS Glue のユーティリティは、GitHub サイトの AWS Glue サンプルリポジトリで公開されています。

Python での AWS Glue API の呼び出し

Boto 3 リソース API は AWS Glue にはまだ使用できないことに注意してください。現時点では、Boto 3 クライアント API のみ使用することができます。

Python の AWS Glue API 名

Java や他のプログラミング言語での AWS Glue API 名は、通常 CamelCased です。ただし、Python から呼び出されるとき、これらの一般名は、より「Python 的」にするために小文字に変更され、名前の一部がアンダースコア文字で区切られます。AWS Glue API リファレンスドキュメントでは、これらの Python 用の名前を一般的な CamelCased 形式の名前の後に括弧で囲んで一覧表示しています。

ただし、AWS Glue API 名自体は小文字に変換されますが、パラメータ名は大文字のままです。次のセクションで説明されますが、AWS Glue API の呼び出し時にパラメータを名前で渡す必要があるため、この点を覚えておくことが重要です。

AWS Glue の Python パラメータの受け渡しとアクセス

AWS Glue API の Python 呼び出しでは、明示的に名前でパラメータを渡すことが最善です。例:

```
job = glue.create_job(Name='sample', Role='Glue_DefaultRole',
                      Command={'Name': 'glueetl',
```

```
                                'ScriptLocation': 's3://my_script_bucket/scripts/
my_etl_script.py'})
```

Job 構造 または JobRun の構造 の ETL スクリプトに渡す引数として指定する名前と値のタプルのディクショナリを Python が 作成することを理解しておくことは役立ちます。その後 Boto 3 が REST API 呼び出しを経由して JSON 形式でそれらを AWS Glue に渡します。つまり、スクリプトでそれらにアクセスするときは、引数の順序に依存することはできません。

たとえば、Python Lambda ハンドラ関数で JobRun を開始しようとしており、複数のパラメータを指定するとします。コードは以下のようになります。

```
from datetime import datetime, timedelta

client = boto3.client('glue')

def lambda_handler(event, context):
  last_hour_date_time = datetime.now() - timedelta(hours = 1)
  day_partition_value = last_hour_date_time.strftime("%Y-%m-%d")
  hour_partition_value = last_hour_date_time.strftime("%-H")

  response = client.start_job_run(
               JobName = 'my_test_Job',
               Arguments = {
                 '--day_partition_key':   'partition_0',
                 '--hour_partition_key':  'partition_1',
                 '--day_partition_value':  day_partition_value,
                 '--hour_partition_value': hour_partition_value } )
```

これらのパラメータに ETL スクリプトで確実にアクセスするには、AWS Glue の getResolvedOptions 関数を使用して名前で指定し、その後作成されたディクショナリからアクセスします。

```
import sys
from awsglue.utils import getResolvedOptions

args = getResolvedOptions(sys.argv,
                          ['JOB_NAME',
                           'day_partition_key',
                           'hour_partition_key',
                           'day_partition_value',
                           'hour_partition_value'])
```

```
print "The day partition key is: ", args['day_partition_key']
print "and the day partition value is: ", args['day_partition_value']
```

ネストされた JSON 文字列により AWS Glue ETL ジョブに引数を渡す際は、そのパラメータ値を保持するために、ジョブの実行が開始される前にパラメータ文字列をエンコードし、ジョブスクリプトが参照される前にパラメータ文字列をデコードする必要があります。例えば、次の引数文字列を考えてみます。

```
glue_client.start_job_run(JobName = "gluejobname", Arguments={
"--my_curly_braces_string": '{"a": {"b": {"c": [{"d": {"e": 42}}]}}}'
})
```

このパラメータを正しく渡すには、引数を Base64 の文字列としてエンコードする必要があります。

```
import base64
...
sample_string='{"a": {"b": {"c": [{"d": {"e": 42}}]}}}'
sample_string_bytes = sample_string.encode("ascii")

base64_bytes = base64.b64encode(sample_string_bytes)
base64_string = base64_bytes.decode("ascii")
...
glue_client.start_job_run(JobName = "gluejobname", Arguments={
"--my_curly_braces_string": base64_bytes})
...
sample_string_bytes = base64.b64decode(base64_bytes)
sample_string = sample_string_bytes.decode("ascii")
print(f"Decoded string: {sample_string}")
...
```

例: ジョブを作成し実行する

以下の例では、Python を使用して AWS Glue API を呼び出し、ETL ジョブを作成して実行する方法を示します。

ジョブを作成し実行するには

1. AWS Glue クライアントのインスタンスを作成します。

   ```
   import boto3
   glue = boto3.client(service_name='glue', region_name='us-east-1',
   ```

```
                endpoint_url='https://glue.us-east-1.amazonaws.com')
```

2. ジョブを作成します。次のコードに示すように、ETL コマンドの名前として glueetl を使用する必要があります。

```
myJob = glue.create_job(Name='sample', Role='Glue_DefaultRole',
                          Command={'Name': 'glueetl',
                                   'ScriptLocation': 's3://my_script_bucket/
scripts/my_etl_script.py'})
```

3. 前のステップで作成したジョブの新しい実行を開始します。

```
myNewJobRun = glue.start_job_run(JobName=myJob['Name'])
```

4. ジョブのステータスを取得します。

```
status = glue.get_job_run(JobName=myJob['Name'], RunId=myNewJobRun['JobRunId'])
```

5. ジョブ実行の現在の状態を出力します。

```
print(status['JobRun']['JobRunState'])
```

AWS Glue での Python ライブラリの使用

AWS Glue では、AWS Glue ETL で使用する追加の Python モジュールとライブラリをインストールできます。

トピック

- requirements.txt を使用して AWS Glue 5.0 に追加の Python ライブラリをインストールする
- pip を使用して追加の Python モジュールを AWS Glue 2.0 以降にインストールする
- PySpark のネイティブ機能を使用して Python ファイルを含める
- ビジュアル変換を使用するプログラミングスクリプト
- AWS Glue で提供済みの Python モジュール
- 取り込みのためのライブラリの圧縮
- Glue Studio ノートブックでの Python AWS ライブラリのロード
- 開発エンドポイントへの Python ライブラリのロード
- ジョブまたは JobRun での Python ライブラリの使用

requirements.txt を使用して AWS Glue 5.0 に追加の Python ライブラリをインストールする

AWS Glue 5.0 では、Python ライブラリの依存関係を管理するための業界標準である `requirements.txt` を指定できます。そのためにはまず、次の 2 つのジョブパラメータを指定します。

- キー: `--python-modules-installer-option`

  値: `-r`
- キー: `--additional-python-modules`

  値: `s3://path_to_requirements.txt`

AWS Glue 5.0 ノードは、`requirements.txt` で指定された Python ライブラリを最初にロードします。次に requirements.txt の例を示します。

```
awswrangler==3.9.1
elasticsearch==8.15.1
PyAthena==3.9.0
PyMySQL==1.1.1
PyYAML==6.0.2
pyodbc==5.2.0
pyorc==0.9.0
redshift-connector==2.1.3
scipy==1.14.1
scikit-learn==1.5.2
SQLAlchemy==2.0.36
```

pip を使用して追加の Python モジュールを AWS Glue 2.0 以降にインストールする

AWS Glue は Python パッケージインストーラー (pip3) を使用して、AWS Glue の ETL で使用する追加モジュールをインストールします。`--additional-python-modules` パラメータでコンマ区切りの Python モジュールのリストを指定することで、新しいモジュールを追加したり、既存のモジュールのバージョンを変更したりできます。Amazon S3 にディストリビューションをアップロードすることでライブラリのカスタムディストリビューションをインストールできます。その後で、モジュールのリストに Amazon S3 オブジェクトへのパスを含めます。

`--python-modules-installer-option` パラメータを使用すると、pip3 に追加オプションを渡すことができます。例えば `"--upgrade"` を渡すことで、`"--additional-python-modules"` で指定されたパッケージをアップグレードできます。その他の例については、「AWS Glue 2.0 を使用

して Spark ETL ワークロード用のホイールから Python モジュールを構築する」を参照してください。

Python の依存関係が、ネイティブのコンパイル済みコードに過渡的に依存している場合、AWS Glue がジョブ環境でのネイティブコードのコンパイルをサポートしていない、という制限による影響を受ける場合があります。ただし、AWS Glue のジョブは、Amazon Linux 2 環境内での実行が可能です。ディストリビューション版 Wheel を通じて、ネイティブの依存関係をコンパイルされた形式で提供できる場合があります。

例えば、更新したり新しい `scikit-learn` モジュールを追加したりするには、`"--additional-python-modules", "scikit-learn==0.21.3"` のキー/値を使用します。

また、`--additional-python-modules` オプションの中で、Python ホイールモジュールへの Amazon S3 パスを指定できます。以下に例を示します。

```
--additional-python-modules s3://aws-glue-native-spark/tests/j4.2/ephem-3.7.7.1-cp37-cp37m-linux_x86_64.whl,s3://aws-glue-native-spark/tests/j4.2/fbprophet-0.6-py3-none-any.whl,scikit-learn==0.21.3
```

`--additional-python-modules` は、AWS Glue コンソールの [ジョブパラメータ] フィールドを使用して指定するか、AWS SDK でジョブの引数を変更して指定します。ジョブパラメータの設定については、「the section called “ジョブのパラメータ”」を参照してください。

PySpark のネイティブ機能を使用して Python ファイルを含める

AWS Glue では、PySpark を使用して、AWS Glue ETL ジョブに Python ファイルを含めます。可能な場合には、依存関係を管理するために、`--additional-python-modules` を使用することが必要になります。Python ファイルをインクルードするには、`--extra-py-files` ジョブパラメータを使用します。依存関係は Amazon S3 でホストされている必要があり、引数の値は、スペースを含まない Amazon S3 パスのカンマ区切りリストである必要があります。この機能は、Spark で使用する Python の依存関係管理のように動作します。Spark での Python の依存関係管理の詳細については、Apache Spark ドキュメントの「Using PySpark Native Features」(PySpark のネイティブ機能を使う) ページを参照してください。追加のコードがパッケージ化されていない場合や、依存関係を管理するために既存のツールチェーンを使用して Spark プログラムを移行する場合には、`--extra-py-files` が便利です。依存関係ツールをメンテナンス可能にするには、送信する前に依存関係をバンドルする必要があります。

ビジュアル変換を使用するプログラミングスクリプト

AWS Glue Studio のビジュアルインターフェイスを使用して AWS Glue ジョブを作成するとき、マネージドデータトランスフォームノードおよびカスタムビジュアル変換を使用してデータを変換できます。マネージドデータ変換ノードの詳細については、「the section called “AWS Glue マネージド変換によるデータの変換”」を参照してください。カスタムビジュアル変換の詳細については、「the section called “ カスタムビジュアル変換によるデータの変換 ”」を参照してください。ビジュアル変換を使用するスクリプトは、ジョブの [言語] が Python を使用するように設定されている場合にのみ生成できます。

ビジュアル変換を使用して AWS Glue ジョブを生成する場合、AWS Glue Studio はジョブ設定の `--extra-py-files` パラメータを使用し、これらの変換をランタイム環境に組み込みます。ジョブパラメータについては、「the section called “ジョブのパラメータ”」を参照してください。生成されたスクリプトまたはランタイム環境を変更するとき、スクリプトを正常に実行するためにこのジョブ設定を保存する必要があります。

AWS Glue で提供済みの Python モジュール

これらの提供済みモジュールのバージョンを変更するには、`--additional-python-modules` ジョブパラメータにより新しいバージョンを指定します。

AWS Glue version 5.0

AWS Glue バージョン 5.0 には、すぐに使用できる次の Python モジュールが含まれています。

- aiobotocore==2.13.1
- aiohappyeyeballs==2.3.5
- aiohttp==3.10.1
- aioitertools==0.11.0
- aiosignal==1.3.1
- appdirs==1.4.4
- attrs==24.2.0
- boto3==1.34.131
- botocore==1.34.131
- certifi==2024.7.4
- charset-normalizer==3.3.2

- contourpy==1.2.1
- cycler==0.12.1
- fonttools==4.53.1
- frozenlist==1.4.1
- fsspec==2024.6.1
- idna==2.10
- jmespath==0.10.0
- kaleido==0.2.1
- kiwisolver==1.4.5
- matplotlib==3.9.0
- multidict==6.0.5
- numpy==1.26.4
- packaging==24.1
- pandas==2.2.2
- pillow==10.4.0
- pip==22.3.1
- plotly==5.23.0
- pyarrow==17.0.0
- pyparsing==3.1.2
- python-dateutil==2.9.0.post0
- pytz==2024.1
- requests==2.32.2
- s3fs==2024.6.1
- s3transfer==0.10.2
- seaborn==0.13.2
- setuptools==59.6.0
- six==1.16.0
- tenacity==9.0.0

- tzdata==2024.1
- urllib3==1.25.10
- virtualenv==20.4.0
- wrapt==1.16.0
- yarl==1.9.4

AWS Glue version 4.0

AWS Glue バージョン 4.0 には、すぐに使用できる次の Python モジュールが含まれています。

- aiobotocore==2.4.1
- aiohttp==3.8.3
- aioitertools==0.11.0
- aiosignal==1.3.1
- async-timeout==4.0.2
- asynctest==0.13.0
- attrs==22.2.0
- avro-python3==1.10.2
- boto3==1.24.70
- botocore==1.27.59
- certifi==2021.5.30
- chardet==3.0.4
- charset-normalizer==2.1.1
- click==8.1.3
- cycler==0.10.0
- Cython==0.29.32
- docutils==0.17.1
- enum34==1.1.10
- frozenlist==1.3.3
- fsspec==2021.8.1

- idna==2.10
- importlib-metadata==5.0.0
- jmespath==0.10.0
- joblib==1.0.1
- kaleido==0.2.1
- kiwisolver==1.4.4
- matplotlib==3.4.3
- mpmath==1.2.1
- multidict==6.0.4
- nltk==3.7
- numpy==1.23.5
- packaging==23.0
- pandas==1.5.1
- patsy==0.5.1
- Pillow==9.4.0
- pip==23.0.1
- plotly==5.16.0
- pmdarima==2.0.1
- ptvsd==4.3.2
- pyarrow==10.0.0
- pydevd==2.5.0
- pyhocon==0.3.58
- PyMySQL==1.0.2
- pyparsing==2.4.7
- python-dateutil==2.8.2
- pytz==2021.1
- PyYAML==6.0.1
- regex==2022.10.31

- requests==2.23.0
- s3fs==2022.11.0
- s3transfer==0.6.0
- scikit-learn==1.1.3
- scipy==1.9.3
- setuptools==49.1.3
- six==1.16.0
- statsmodels==0.13.5
- subprocess32==3.5.4
- sympy==1.8
- tbats==1.1.0
- threadpoolctl==3.1.0
- tqdm==4.64.1
- typing_extensions==4.4.0
- urllib3==1.25.11
- wheel==0.37.0
- wrapt==1.14.1
- yarl==1.8.2
- zipp==3.10.0

AWS Glue version 3.0

AWS Glue バージョン 3.0 には、すぐに使用できる次の Python モジュールが含まれています。

- aiobotocore==1.4.2
- aiohttp==3.8.3
- aioitertools==0.11.0
- aiosignal==1.3.1
- async-timeout==4.0.2
- asynctest==0.13.0

- attrs==22.2.0
- avro-python3==1.10.2
- boto3==1.18.50
- botocore==1.21.50
- certifi==2021.5.30
- chardet==3.0.4
- charset-normalizer==2.1.1
- click==8.1.3
- cycler==0.10.0
- Cython==0.29.4
- docutils==0.17.1
- enum34==1.1.10
- frozenlist==1.3.3
- fsspec==2021.8.1
- idna==2.10
- importlib-metadata==6.0.0
- jmespath==0.10.0
- joblib==1.0.1
- kiwisolver==1.3.2
- matplotlib==3.4.3
- mpmath==1.2.1
- multidict==6.0.4
- nltk==3.6.3
- numpy==1.19.5
- packaging==23.0
- pandas==1.3.2
- patsy==0.5.1
- Pillow==9.4.0

- pip==23.0
- pmdarima==1.8.2
- ptvsd==4.3.2
- pyarrow==5.0.0
- pydevd==2.5.0
- pyhocon==0.3.58
- PyMySQL==1.0.2
- pyparsing==2.4.7
- python-dateutil==2.8.2
- pytz==2021.1
- PyYAML==5.4.1
- regex==2022.10.31
- requests==2.23.0
- s3fs==2021.8.1
- s3transfer==0.5.0
- scikit-learn==0.24.2
- scipy==1.7.1
- six==1.16.0
- Spark==1.0
- statsmodels==0.12.2
- subprocess32==3.5.4
- sympy==1.8
- tbats==1.1.0
- threadpoolctl==3.1.0
- tqdm==4.64.1
- typing_extensions==4.4.0
- urllib3==1.25.11
- wheel==0.37.0

- wrapt==1.14.1
- yarl==1.8.2
- zipp==3.12.0

AWS Glue version 2.0

AWS Glue バージョン 2.0 には、すぐに使用できる次の Python モジュールが含まれています。

- avro-python3==1.10.0
- awscli==1.27.60
- boto3==1.12.4
- botocore==1.15.4
- certifi==2019.11.28
- chardet==3.0.4
- click==8.1.3
- colorama==0.4.4
- cycler==0.10.0
- Cython==0.29.15
- docutils==0.15.2
- enum34==1.1.9
- fsspec==0.6.2
- idna==2.9
- importlib-metadata==6.0.0
- jmespath==0.9.4
- joblib==0.14.1
- kiwisolver==1.1.0
- matplotlib==3.1.3
- mpmath==1.1.0
- nltk==3.5
- numpy==1.18.1

- pandas==1.0.1
- patsy==0.5.1
- pmdarima==1.5.3
- ptvsd==4.3.2
- pyarrow==0.16.0
- pyasn1==0.4.8
- pydevd==1.9.0
- pyhocon==0.3.54
- PyMySQL==0.9.3
- pyparsing==2.4.6
- python-dateutil==2.8.1
- pytz==2019.3
- PyYAML==5.3.1
- regex==2022.10.31
- requests==2.23.0
- rsa==4.7.2
- s3fs==0.4.0
- s3transfer==0.3.3
- scikit-learn==0.22.1
- scipy==1.4.1
- setuptools==45.2.0
- six==1.14.0
- Spark==1.0
- statsmodels==0.11.1
- subprocess32==3.5.4
- sympy==1.5.1
- tbats==1.0.9
- tqdm==4.64.1
- typing-extensions==4.4.0

- urllib3==1.25.8
- wheel==0.35.1
- zipp==3.12.0

取り込みのためのライブラリの圧縮

ライブラリは、単一の .py ファイルに含まれていない限り、.zip アーカイブにパッケージ化される必要があります。パッケージディレクトリは、アーカイブのルートにあって、パッケージの __init__.py ファイルを含んでいる必要があります。そうすると、Python は通常の方法でパッケージをインポートできるようになります。

ライブラリが 1 つの .py ファイルにある単一の Python モジュールでのみ構成されている場合、.zip ファイルに入れる必要はありません。

Glue Studio ノートブックでの Python AWS ライブラリのロード

Glue Studio ノートブックで Python AWS ライブラリを指定するには、「追加の Python モジュールのインストール」を参照してください。

開発エンドポイントへの Python ライブラリのロード

異なる ETL スクリプトに異なるライブラリセットを使用している場合、各セットに別々の開発エンドポイントをセットアップするか、スクリプトを切り替えるたびに開発エンドポイントがロードするライブラリ .zip ファイルを上書きすることができます。

コンソールを使用して、作成時に開発エンドポイントに 1 つまたは複数のライブラリ .zip ファイルを指定できます。名前と IAM ロールを割り当てた後、[Script Libraries and job parameters (optional)] (スクリプトライブラリおよびジョブパラメータ (任意)) をクリックし、[Python library path] (Python ライブラリパス) ボックスに、ライブラリ .zip ファイルへの完全な Amazon S3 パスを入力します。例:

```
s3://bucket/prefix/site-packages.zip
```

必要に応じてファイルへの複数のフルパスを指定できますが、以下のように、スペースなしでカンマで区切ります。

```
s3://bucket/prefix/lib_A.zip,s3://bucket_B/prefix/lib_X.zip
```

これらの .zip ファイルを後で更新する場合は、コンソールを使用して開発エンドポイントにそのファイルを再インポートできます。該当する開発エンドポイントに移動し、横にあるチェックボックスをオンにして、[Action] (アクション) メニューから [Update ETL libraries] (ETL ライブラリの更新) を選択します。

同様の方法で、AWS Glue API を使用してライブラリファイルを指定できます。CreateDevEndpoint アクション (Python: create_dev_endpoint) を呼び出して開発エンドポイントを作成する場合、ExtraPythonLibsS3Path パラメータでライブラリへの 1 つ以上のフルパスを指定できます。以下のような呼び出しになります。

```
dep = glue.create_dev_endpoint(
             EndpointName="testDevEndpoint",
             RoleArn="arn:aws:iam::123456789012",
             SecurityGroupIds="sg-7f5ad1ff",
             SubnetId="subnet-c12fdba4",
             PublicKey="ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQCtp04H/y...",
             NumberOfNodes=3,
             ExtraPythonLibsS3Path="s3://bucket/prefix/lib_A.zip,s3://bucket_B/prefix/
lib_X.zip")
```

開発エンドポイントを更新するときに、DevEndpointCustomLibraries オブジェクトを使用し UpdateDevEndpoint (update_dev_endpoint) の呼び出し時に UpdateEtlLibraries パラメータを True に設定して、ロードするライブラリも更新できます。

ジョブまたは JobRun での Python ライブラリの使用

コンソールで新しいジョブを作成する際、[Script Libraries and job parameters (optional)] (スクリプトライブラリおよびジョブパラメータ (任意)) をクリックし、開発エンドポイント作成時と同じ方法で Amazon S3 ライブラリの完全なパスを入力することで、1 つ以上のライブラリ .zip ファイルを指定できます。

```
s3://bucket/prefix/lib_A.zip,s3://bucket_B/prefix/lib_X.zip
```

CreateJob (create_job) を呼び出している場合は、以下のようにデフォルトの --extra-py-files パラメータを使用してデフォルトのライブラリへの 1 つ以上のフルパスを指定できます。

```
job = glue.create_job(Name='sampleJob',
                      Role='Glue_DefaultRole',
```

```
                      Command={'Name': 'glueetl',
                               'ScriptLocation': 's3://my_script_bucket/scripts/
my_etl_script.py'},
                      DefaultArguments={'--extra-py-files': 's3://bucket/prefix/
lib_A.zip,s3://bucket_B/prefix/lib_X.zip'})
```

その後 JobRun を開始するときに、デフォルトのライブラリ設定を別のもので上書きできます。

```
runId = glue.start_job_run(JobName='sampleJob',
                           Arguments={'--extra-py-files': 's3://bucket/prefix/
lib_B.zip'})
```

AWS Glue Python コードサンプル

- コード例: データの結合と関係付け
- コード例: ResolveChoice、Lambda、および ApplyMapping を使用したデータ準備

コード例: データの結合と関係付け

この例では、http://everypolitician.org/ から、Amazon Simple Storage Service (Amazon S3) の sample-dataset バケットにデータセットをダウンロードして使用します。s3://awsglue-datasets/examples/us-legislators/allこのデータセットには、米国議会議員や米国下院および上院議員の議席に関する JSON 形式のデータが含まれており、このチュートリアルの目的のため少し変更され、パブリック Amazon S3 バケットで利用可能になりました。

この例のソースコードは、GitHub ウェブサイトの join_and_relationalize.py Glue サンプルリポジトリの AWS Glue ファイルにあります。

このデータを使用して、このチュートリアルでは以下のことを実行する方法を示します。

- AWS Glue クローラを使用して、パブリックな Amazon S3 バケットに保存されているオブジェクトを分類し、それらのスキーマを AWS Glue Data Catalog に保存します。
- クロールの結果のテーブルのメタデータとスキーマを調べます。
- Data Catalog のメタデータを使用して Python の抽出、転送、およびロード (ETL) スクリプトを記述し、以下の操作を行います。
  - 異なるソースファイル内のデータをまとめて単一のデータテーブルに結合します (つまり、データを非正規化します)。
  - 議員のタイプ別に、結合テーブルを別のテーブルにフィルタリングします。

- 生成されたデータを後で分析するために Apache Parquet ファイルに分割して書き出します。

AWS で実行中に Python または PySpark スクリプトをデバッグするための推奨方法は、AWS Glue Studio のノートブックを使用することです。

ステップ 1: Amazon S3 バケット内のデータをクロールする

1. AWS Management Console にサインインし、AWS Glue コンソール (https://console.aws.amazon.com/glue/) を開きます。
2. クローラーの設定 の手順に従って、`s3://awsglue-datasets/examples/us-legislators/all` データセットをクロールできる新しいクローラを、AWS Glue Data Catalog 内のデータベース `legislators` に作成します。サンプルデータは既に、このパブリックな Amazon S3 バケットに用意されています。
3. 新しいクローラを実行し、`legislators` データベースを確認します。

   クローラは、次のメタデータテーブルを作成します。

   - `persons_json`
   - `memberships_json`
   - `organizations_json`
   - `events_json`
   - `areas_json`
   - `countries_r_json`

   これは、議員とその履歴を含むテーブルの半正規化されたテーブルの集合です。

ステップ 2: 開発エンドポイントノートブックに共通スクリプトを追加する

次の共通スクリプトを開発エンドポイントノートブックに貼り付けて、必要な AWS Glue ライブラリをインポートし、単一の `GlueContext` を設定します。

```
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
```

```
from awsglue.job import Job

glueContext = GlueContext(SparkContext.getOrCreate())
```

ステップ 3: Data Catalog 内のデータでスキーマを確認する

次に、簡単な手順で AWS Glue Data Catalog から DynamicFrame を作成し、そのデータのスキーマを調べます。例えば、persons_json テーブルのスキーマを表示するには、ノートブックに以下を追加します。

```
persons = glueContext.create_dynamic_frame.from_catalog(
             database="legislators",
             table_name="persons_json")
print "Count: ", persons.count()
persons.printSchema()
```

プリントコールの出力を以下に示します。

```
Count:  1961
root
|-- family_name: string
|-- name: string
|-- links: array
|    |-- element: struct
|    |    |-- note: string
|    |    |-- url: string
|-- gender: string
|-- image: string
|-- identifiers: array
|    |-- element: struct
|    |    |-- scheme: string
|    |    |-- identifier: string
|-- other_names: array
|    |-- element: struct
|    |    |-- note: string
|    |    |-- name: string
|    |    |-- lang: string
|-- sort_name: string
|-- images: array
|    |-- element: struct
```

```
|    |    |-- url: string
|-- given_name: string
|-- birth_date: string
|-- id: string
|-- contact_details: array
|    |-- element: struct
|    |    |-- type: string
|    |    |-- value: string
|-- death_date: string
```

テーブル内の各人は、米国議会のメンバーです。

memberships_json テーブルのスキーマを表示するには、次のように入力します。

```
memberships = glueContext.create_dynamic_frame.from_catalog(
                 database="legislators",
                 table_name="memberships_json")
print "Count: ", memberships.count()
memberships.printSchema()
```

出力は次のとおりです。

```
Count:  10439
root
|-- area_id: string
|-- on_behalf_of_id: string
|-- organization_id: string
|-- role: string
|-- person_id: string
|-- legislative_period_id: string
|-- start_date: string
|-- end_date: string
```

organizations は政党および上院と下院の 2 つの議会です。organizations_json テーブルのスキーマを表示するには、次のように入力します。

```
orgs = glueContext.create_dynamic_frame.from_catalog(
           database="legislators",
```

```
           table_name="organizations_json")
print "Count: ", orgs.count()
orgs.printSchema()
```

出力は次のとおりです。

```
Count:  13
root
|-- classification: string
|-- links: array
|    |-- element: struct
|    |    |-- note: string
|    |    |-- url: string
|-- image: string
|-- identifiers: array
|    |-- element: struct
|    |    |-- scheme: string
|    |    |-- identifier: string
|-- other_names: array
|    |-- element: struct
|    |    |-- lang: string
|    |    |-- note: string
|    |    |-- name: string
|-- id: string
|-- name: string
|-- seats: int
|-- type: string
```

ステップ 4: データをフィルタリングする

次に、必要なフィールドのみを保持し、id の名前を org_id に変更します。データセットは、小さいため全体を表示することができます。

toDF() は DynamicFrame を Apache Spark に変換するので、Apache Spark SQL に既に存在する DataFrame 変換を適用できます。

```
orgs = orgs.drop_fields(['other_names',
                         'identifiers']).rename_field(
                             'id', 'org_id').rename_field(
                                'name', 'org_name')
```

```
orgs.toDF().show()
```

以下に出力を示します。

```
+--------------+--------------------+--------------------+--------------------+-----
+-----------+--------------------+
|classification|              org_id|            org_name|               links|seats|
     type|               image|
+--------------+--------------------+--------------------+--------------------+-----
+-----------+--------------------+
|         party|            party/al|                  AL|                null| null|
     null|                null|
|         party|      party/democrat|            Democrat|[[website,http://...| null|
     null|https://upload.wi...|
|         party|party/democrat-li...|    Democrat-Liberal|[[website,http://...| null|
     null|                null|
|   legislature|d56acebe-8fdc-47b...|House of Represen...|                null|  435|
lower house|                null|
|         party|   party/independent|         Independent|                null| null|
     null|                null|
|         party|party/new_progres...|     New Progressive|[[website,http://...| null|
     null|https://upload.wi...|
|         party|party/popular_dem...|    Popular Democrat|[[website,http://...| null|
     null|                null|
|         party|    party/republican|          Republican|[[website,http://...| null|
     null|https://upload.wi...|
|         party|party/republican-...|Republican-Conser...|[[website,http://...| null|
     null|                null|
|         party|      party/democrat|            Democrat|[[website,http://...| null|
     null|https://upload.wi...|
|         party|   party/independent|         Independent|                null| null|
     null|                null|
|         party|    party/republican|          Republican|[[website,http://...| null|
     null|https://upload.wi...|
|   legislature|8fa6c3d2-71dc-478...|             Senate|                null|  100|
upper house|                null|
+--------------+--------------------+--------------------+--------------------+-----
+-----------+--------------------+
```

memberships に表示される organizations を表示するには、次のように入力します。

```
memberships.select_fields(['organization_id']).toDF().distinct().show()
```

以下に出力を示します。

```
+--------------------+
|     organization_id|
+--------------------+
|d56acebe-8fdc-47b...|
|8fa6c3d2-71dc-478...|
+--------------------+
```

ステップ 5: すべてをまとめる

ここで AWS Glue を使用して、これらのリレーショナルテーブルを結合し、議員の memberships とそれに対応する organizations の 1 つの完全な履歴テーブルを作成します。

1. まず、persons および memberships を id および person_id と結合します。
2. 次に、結果を orgs と org_id および organization_id と結合します。
3. 次に、冗長なフィールド person_id および org_id を削除します。

これらの操作はすべて、1 行の (拡張された) コードで行うことができます。

```
l_history = Join.apply(orgs,
                       Join.apply(persons, memberships, 'id', 'person_id'),
                       'org_id', 'organization_id').drop_fields(['person_id',
 'org_id'])
print "Count: ", l_history.count()
l_history.printSchema()
```

出力は次のとおりです。

```
Count:  10439
root
|-- role: string
|-- seats: int
|-- org_name: string
|-- links: array
```

```
|    |-- element: struct
|    |    |-- note: string
|    |    |-- url: string
|-- type: string
|-- sort_name: string
|-- area_id: string
|-- images: array
|    |-- element: struct
|    |    |-- url: string
|-- on_behalf_of_id: string
|-- other_names: array
|    |-- element: struct
|    |    |-- note: string
|    |    |-- name: string
|    |    |-- lang: string
|-- contact_details: array
|    |-- element: struct
|    |    |-- type: string
|    |    |-- value: string
|-- name: string
|-- birth_date: string
|-- organization_id: string
|-- gender: string
|-- classification: string
|-- death_date: string
|-- legislative_period_id: string
|-- identifiers: array
|    |-- element: struct
|    |    |-- scheme: string
|    |    |-- identifier: string
|-- image: string
|-- given_name: string
|-- family_name: string
|-- id: string
|-- start_date: string
|-- end_date: string
```

これで、分析に使用できる最終テーブルが作成されました。これは、分析のためのコンパクトで効率的な形式 (つまり Parquet) で記述することができ、AWS Glue、Amazon Athena、または Amazon Redshift Spectrum で SQL を実行できます。

次の呼び出しは、複数のファイルにわたってテーブルを書き込んで、後で解析するときに高速な並列読み込みをサポートします。

```
glueContext.write_dynamic_frame.from_options(frame = l_history,
          connection_type = "s3",
          connection_options = {"path": "s3://glue-sample-target/output-dir/
legislator_history"},
          format = "parquet")
```

すべての履歴データを単一のファイルにまとめるには、データフレームに変換し、再パーティション化して書き出す必要があります。

```
s_history = l_history.toDF().repartition(1)
s_history.write.parquet('s3://glue-sample-target/output-dir/legislator_single')
```

または、上院と下院でそれを分けたい場合。

```
l_history.toDF().write.parquet('s3://glue-sample-target/output-dir/legislator_part',
                               partitionBy=['org_name'])
```

ステップ 6: リレーショナルデータベース向けにデータを変換する

AWS Glue では半構造化データでも Amazon Redshift のようなリレーショナルデータベースに簡単に書き込むことができるのです。これにより、フレーム内のオブジェクトがどれほど複雑であっても、DynamicFrames をフラット化する変換 relationalize が提供されます。

この例の l_history DynamicFrame を使用して、ルートテーブル (hist_root) の名前と一時的な作業パスを relationalize に渡します。これにより、DynamicFrameCollection が返されます。その後、そのコレクション内の DynamicFrames の名前を一覧表示できます。

```
dfc = l_history.relationalize("hist_root", "s3://glue-sample-target/temp-dir/")
dfc.keys()
```

keys 呼び出しの出力は次のとおりです。

```
[u'hist_root', u'hist_root_contact_details', u'hist_root_links',
 u'hist_root_other_names', u'hist_root_images', u'hist_root_identifiers']
```

Relationalize は、履歴テーブルを 6 つの新しいテーブルに分割します。DynamicFrame の各オブジェクトのレコードを含むルートテーブル、および配列の補助テーブルです。リレーショナルデータベースでの配列の処理は、特に配列が大きくなる場合に、最適ではないことがあります。配列を別のテーブルに分けることで、クエリの実行速度が大幅に向上します。

次に、contact_details を調べて分離を確認します。

```
l_history.select_fields('contact_details').printSchema()
dfc.select('hist_root_contact_details').toDF().where("id = 10 or id =
 75").orderBy(['id','index']).show()
```

show 呼び出しの出力は次のとおりです。

```
root
|-- contact_details: array
|    |-- element: struct
|    |    |-- type: string
|    |    |-- value: string
+---+-----+-----------------------+-------------------------+
| id|index|contact_details.val.type|contact_details.val.value|
+---+-----+-----------------------+-------------------------+
| 10|    0|                    fax|                         |
| 10|    1|                       |             202-225-1314|
| 10|    2|                  phone|                         |
| 10|    3|                       |             202-225-3772|
| 10|    4|                twitter|                         |
| 10|    5|                       |          MikeRossUpdates|
| 75|    0|                    fax|                         |
| 75|    1|                       |             202-225-7856|
| 75|    2|                  phone|                         |
| 75|    3|                       |             202-225-2711|
| 75|    4|                twitter|                         |
| 75|    5|                       |                SenCapito|
+---+-----+-----------------------+-------------------------+
```

contact_details フィールドは、元の DynamicFrame の構造体の配列です。これらの配列の各要素は、index によってインデックス化された、補助テーブルの個別の行です。ここで id は、contact_details キーを使用する hist_root テーブルの外部キーです。

```
dfc.select('hist_root').toDF().where(
    "contact_details = 10 or contact_details = 75").select(
       ['id', 'given_name', 'family_name', 'contact_details']).show()
```

出力を次に示します。

```
+--------------------+----------+-----------+---------------+
|                  id|given_name|family_name|contact_details|
+--------------------+----------+-----------+---------------+
|f4fc30ee-7b42-432...|      Mike|       Ross|             10|
|e3c60f34-7d1b-4c0...|   Shelley|     Capito|             75|
+--------------------+----------+-----------+---------------+
```

これらのコマンドでは、toDF() および where 式を使用して、表示する行をフィルタリングすることに注意してください。

したがって、hist_root テーブルを補助テーブルと結合すると、次のことが可能になります。

- 配列をサポートせずにデータベースにデータをロードします。
- SQL を使用して配列内の各項目にクエリを実行します。

AWS Glue 接続を使用して、Amazon Redshift の認証情報を安全に保存してアクセスします。独自の接続の作成方法については、「データへの接続」を参照してください。

DynamicFrames を 1 つずつ切り替えて、接続にデータを書き込みできるようになりました。

```
for df_name in dfc.keys():
  m_df = dfc.select(df_name)
  print "Writing to table: ", df_name
  glueContext.write_dynamic_frame.from_jdbc_conf(frame = m_df, connection settings
 here)
```

接続設定は、リレーショナルデータベースのタイプによって異なります。

- Amazon Redshift への書き込み手順については、「the section called “Redshift 接続”」を参照してください。
- その他のデータベースについては、「AWS Glue for Spark での ETL の接続タイプとオプション」を参照してください。

結論

全体として、AWS Glue は非常に柔軟です。通常は書くのに数日かかるところを、数行のコードで達成できます。ソースからターゲットへの ETL スクリプトの全体は、GitHub の AWS Glue サンプル内の Python ファイル `join_and_relationalize.py` にあります。

コード例: ResolveChoice、Lambda、および ApplyMapping を使用したデータ準備

この例で使用されているデータセットは、2 つの Data.CMS.gov データセット (Inpatient Prospective Payment System Provider Summary for the Top 100 Diagnosis-Related Groups - FY2011 および Inpatient Charge Data FY 2011) からダウンロードされた、メディケアプロバイダの支払いデータで構成されています。ダウンロードした後、データセットを修正してファイルの最後にいくつかの誤ったレコードを追加しました。この変更されたファイルは、パブリックな Amazon S3 バケット (`s3://awsglue-datasets/examples/medicare/Medicare_Hospital_Provider.csv`) 内に置かれています。

この例のソースコードは、`data_cleaning_and_lambda.py` 例AWS Glue GitHub リポジトリのファイルにあります。

AWS で実行中に Python または PySpark スクリプトをデバッグするための推奨方法は、AWS Glue Studio のノートブックを使用することです。

ステップ 1: Amazon S3 バケット内のデータをクロールする

1. AWS Management Consoleにサインインし、AWS Glue コンソール (https://console.aws.amazon.com/glue/) を開きます。
2. クローラーの設定 で説明されているプロセスに従って `s3://awsglue-datasets/examples/medicare/Medicare_Hospital_Provider.csv` ファイルをクロールできる新しいクローラを作成し、得られた結果のメタデータを AWS Glue Data Catalog の `payments` という名前のデータベースに配置します。
3. 新しいクローラを実行し、`payments` データベースを確認します。クローラは、最初のファイルを読み込んでファイルの形式と区切り記号を判断してから、データベースに `medicare` という名前のメタデータテーブルを作成したことが分かります。

   新しい `medicare` テーブルのスキーマは次のようになります。

```
Column  name                                Data type
================================================
drg definition                                string
provider id                                   bigint
```

```
provider name                               string
provider street address                     string
provider city                               string
provider state                              string
provider zip code                           bigint
hospital referral region description        string
total discharges                            bigint
average covered charges                     string
average total payments                      string
average medicare payments                   string
```

ステップ 2: 開発エンドポイントノートブックに共通スクリプトを追加する

次の共通スクリプトを開発エンドポイントノートブックに貼り付けて、必要な AWS Glue ライブラリをインポートし、単一の GlueContext を設定します。

```
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job

glueContext = GlueContext(SparkContext.getOrCreate())
```

ステップ 3: 異なるスキーマ解析を比較する

次に、Apache Spark DataFrame によって認識されたスキーマが、AWS Glue クローラによって記録されたスキーマと同じかどうかを確認できます。以下のコードを実行します。

```
medicare = spark.read.format(
   "com.databricks.spark.csv").option(
   "header", "true").option(
   "inferSchema", "true").load(
   's3://awsglue-datasets/examples/medicare/Medicare_Hospital_Provider.csv')
medicare.printSchema()
```

printSchema 呼び出しの出力は次のとおりです。

```
root
 |-- DRG Definition: string (nullable = true)
 |-- Provider Id: string (nullable = true)
 |-- Provider Name: string (nullable = true)
 |-- Provider Street Address: string (nullable = true)
 |-- Provider City: string (nullable = true)
 |-- Provider State: string (nullable = true)
 |-- Provider Zip Code: integer (nullable = true)
 |-- Hospital Referral Region Description: string (nullable = true)
 |--  Total Discharges : integer (nullable = true)
 |--  Average Covered Charges : string (nullable = true)
 |--  Average Total Payments : string (nullable = true)
 |-- Average Medicare Payments: string (nullable = true)
```

次に、AWS Glue DynamicFrame によって生成されるスキーマを確認します。

```
medicare_dynamicframe = glueContext.create_dynamic_frame.from_catalog(
       database = "payments",
       table_name = "medicare")
medicare_dynamicframe.printSchema()
```

printSchema の出力は次のとおりです。

```
root
 |-- drg definition: string
 |-- provider id: choice
 |    |-- long
 |    |-- string
 |-- provider name: string
 |-- provider street address: string
 |-- provider city: string
 |-- provider state: string
 |-- provider zip code: long
 |-- hospital referral region description: string
 |-- total discharges: long
 |-- average covered charges: string
 |-- average total payments: string
 |-- average medicare payments: string
```

DynamicFrame は、provider id が long 型または string 型のいずれかであるスキーマを生成します。DataFrame スキーマは Provider Id を string 型としてリストし、Data Catalog は provider id を bigint 型としてリストします。

正しいものはどちらでしょうか。ファイルの末尾には、その列に string 値を持つ 2 つのレコード (160,000 レコードのうち) があります。これらは、問題を説明するために導入されたエラーのあるレコードです。

このような問題に対処するために、AWS Glue DynamicFrame では Choice 型の概念を導入しています。この場合、DynamicFrame は、その列に long 値と string 値の両方が存在することを示しています。AWS Glue クローラはデータの 2 MB のプレフィックスのみを考慮しているため、string 値を見落としました。Apache Spark DataFrame はデータセット全体を考慮しましたが、最も一般的な型、つまり string 型を強制的に列に割り当てました。実際、慣れていない複雑な型やバリエーションがある場合にも、Spark は最も一般的なケースを使用することがあります。

provider id 列のクエリを実行するには、Choice 型をまず解決する必要があります。DynamicFrame で、cast:long オプションを指定して resolveChoice 変換メソッドを使用すると、これらの string 値を long 値に変換できます。

```
medicare_res = medicare_dynamicframe.resolveChoice(specs = [('provider
 id','cast:long')])
medicare_res.printSchema()
```

この場合、printSchema の出力は次のようになります。

```
root
 |-- drg definition: string
 |-- provider id: long
 |-- provider name: string
 |-- provider street address: string
 |-- provider city: string
 |-- provider state: string
 |-- provider zip code: long
 |-- hospital referral region description: string
 |-- total discharges: long
 |-- average covered charges: string
 |-- average total payments: string
 |-- average medicare payments: string
```

値がキャストできない string だった場合に、AWS Glue は null を挿入しました。

もう 1 つのオプションは、両方のタイプの値を保持する `struct` に Choice 型を変換することです。

次に、異常だった行を確認してみましょう。

```
medicare_res.toDF().where("'provider id' is NULL").show()
```

次のように表示されています。

```
+--------------------+-----------+--------------+----------------------+-------------
+--------------+-----------------+-------------------------------------+---------------
+----------------------+---------------------+-------------------------+
|      drg definition|provider id|  provider name|provider street address|provider
 city|provider state|provider zip code|hospital referral region description|total
 discharges|average covered charges|average total payments|average medicare payments|
+--------------------+-----------+--------------+----------------------+-------------
+--------------+-----------------+-------------------------------------+---------------
+----------------------+---------------------+-------------------------+
|948 - SIGNS & SYM...|       null|           INC|       1050 DIVISION ST|
 MAUSTON|           WI|           53948|                        WI - Madison|
      12|             $11961.41|             $4619.00|                 $3775.33|
|948 - SIGNS & SYM...|       null| INC- ST JOSEPH|     5000 W CHAMBERS ST|
 MILWAUKEE|           WI|           53210|                        WI - Milwaukee|
        14|             $10514.28|             $5562.50|                 $4522.78|
+--------------------+-----------+--------------+----------------------+-------------
+--------------+-----------------+-------------------------------------+---------------
+----------------------+---------------------+-------------------------+
```

次のように、2 つの不正な形式のレコードを削除します。

```
medicare_dataframe = medicare_res.toDF()
medicare_dataframe = medicare_dataframe.where("'provider id' is NOT NULL")
```

ステップ 4: データのマッピングと Apache Spark Lambda 関数の使用

AWS Glue では、まだユーザー定義関数とも呼ばれる Lambda 関数が直接サポートされていません。しかし、いつでも `DynamicFrame` を Apache Spark `DataFrame` との間で変換して、`DynamicFrames` の特殊な機能に加えて Spark の機能を利用できます。

次に、支払い情報を数字に変換すると、Amazon Redshift や Amazon Athena のような分析エンジンが、より迅速に数値処理を実行できるようになります。

```
from pyspark.sql.functions import udf
from pyspark.sql.types import StringType

chop_f = udf(lambda x: x[1:], StringType())
medicare_dataframe = medicare_dataframe.withColumn(
        "ACC", chop_f(
            medicare_dataframe["average covered charges"])).withColumn(
                "ATP", chop_f(
                    medicare_dataframe["average total payments"])).withColumn(
                        "AMP", chop_f(
                            medicare_dataframe["average medicare payments"]))
medicare_dataframe.select(['ACC', 'ATP', 'AMP']).show()
```

show 呼び出しの出力は次のとおりです。

```
+--------+-------+-------+
|     ACC|    ATP|    AMP|
+--------+-------+-------+
|32963.07|5777.24|4763.73|
|15131.85|5787.57|4976.71|
|37560.37|5434.95|4453.79|
|13998.28|5417.56|4129.16|
|31633.27|5658.33|4851.44|
|16920.79|6653.80|5374.14|
|11977.13|5834.74|4761.41|
|35841.09|8031.12|5858.50|
|28523.39|6113.38|5228.40|
|75233.38|5541.05|4386.94|
|67327.92|5461.57|4493.57|
|39607.28|5356.28|4408.20|
|22862.23|5374.65|4186.02|
|31110.85|5366.23|4376.23|
|25411.33|5282.93|4383.73|
| 9234.51|5676.55|4509.11|
|15895.85|5930.11|3972.85|
|19721.16|6192.54|5179.38|
|10710.88|4968.00|3898.88|
|51343.75|5996.00|4962.45|
+--------+-------+-------+
only showing top 20 rows
```

これらは、すべてデータ内ではまだ文字列です。強力な apply_mapping 変換メソッドを使用して、データをドロップ、名前変更、キャスト、およびネストし、他のデータプログラミング言語やシステムで容易にアクセスできるようにします。

```
from awsglue.dynamicframe import DynamicFrame
medicare_tmp_dyf = DynamicFrame.fromDF(medicare_dataframe, glueContext, "nested")
medicare_nest_dyf = medicare_tmp_dyf.apply_mapping([('drg definition', 'string', 'drg',
 'string'),
                ('provider id', 'long', 'provider.id', 'long'),
                ('provider name', 'string', 'provider.name', 'string'),
                ('provider city', 'string', 'provider.city', 'string'),
                ('provider state', 'string', 'provider.state', 'string'),
                ('provider zip code', 'long', 'provider.zip', 'long'),
                ('hospital referral region description', 'string','rr', 'string'),
                ('ACC', 'string', 'charges.covered', 'double'),
                ('ATP', 'string', 'charges.total_pay', 'double'),
                ('AMP', 'string', 'charges.medicare_pay', 'double')])
medicare_nest_dyf.printSchema()
```

printSchema の出力は次のとおりです。

```
root
 |-- drg: string
 |-- provider: struct
 |    |-- id: long
 |    |-- name: string
 |    |-- city: string
 |    |-- state: string
 |    |-- zip: long
 |-- rr: string
 |-- charges: struct
 |    |-- covered: double
 |    |-- total_pay: double
 |    |-- medicare_pay: double
```

データを Spark DataFrame に戻すと、現在どのような状態かが分かります。

```
medicare_nest_dyf.toDF().show()
```

出力は次のとおりです。

```
+--------------------+--------------------+--------------+--------------------+
```

```
|                 drg|           provider|             rr|             charges|
+--------------------+--------------------+---------------+--------------------+
|039 - EXTRACRANIA...|[10001,SOUTHEAST ...|    AL - Dothan|[32963.07,5777.24...|
|039 - EXTRACRANIA...|[10005,MARSHALL M...|AL - Birmingham|[15131.85,5787.57...|
|039 - EXTRACRANIA...|[10006,ELIZA COFF...|AL - Birmingham|[37560.37,5434.95...|
|039 - EXTRACRANIA...|[10011,ST VINCENT...|AL - Birmingham|[13998.28,5417.56...|
|039 - EXTRACRANIA...|[10016,SHELBY BAP...|AL - Birmingham|[31633.27,5658.33...|
|039 - EXTRACRANIA...|[10023,BAPTIST ME...|AL - Montgomery|[16920.79,6653.8,...|
|039 - EXTRACRANIA...|[10029,EAST ALABA...|AL - Birmingham|[11977.13,5834.74...|
|039 - EXTRACRANIA...|[10033,UNIVERSITY...|AL - Birmingham|[35841.09,8031.12...|
|039 - EXTRACRANIA...|[10039,HUNTSVILLE...|AL - Huntsville|[28523.39,6113.38...|
|039 - EXTRACRANIA...|[10040,GADSDEN RE...|AL - Birmingham|[75233.38,5541.05...|
|039 - EXTRACRANIA...|[10046,RIVERVIEW ...|AL - Birmingham|[67327.92,5461.57...|
|039 - EXTRACRANIA...|[10055,FLOWERS HO...|    AL - Dothan|[39607.28,5356.28...|
|039 - EXTRACRANIA...|[10056,ST VINCENT...|AL - Birmingham|[22862.23,5374.65...|
|039 - EXTRACRANIA...|[10078,NORTHEAST ...|AL - Birmingham|[31110.85,5366.23...|
|039 - EXTRACRANIA...|[10083,SOUTH BALD...|    AL - Mobile|[25411.33,5282.93...|
|039 - EXTRACRANIA...|[10085,DECATUR GE...|AL - Huntsville|[9234.51,5676.55,...|
|039 - EXTRACRANIA...|[10090,PROVIDENCE...|    AL - Mobile|[15895.85,5930.11...|
|039 - EXTRACRANIA...|[10092,D C H REGI...|AL - Tuscaloosa|[19721.16,6192.54...|
|039 - EXTRACRANIA...|[10100,THOMAS HOS...|    AL - Mobile|[10710.88,4968.0,...|
|039 - EXTRACRANIA...|[10103,BAPTIST ME...|AL - Birmingham|[51343.75,5996.0,...|
+--------------------+--------------------+---------------+--------------------+
only showing top 20 rows
```

ステップ 5: Apache Parquet にデータを書き込む

AWS Glue は、リレーショナルデータベースが効果的に消費できる Apache Parquet のような形式でデータを書き込むことを容易にします。

```
glueContext.write_dynamic_frame.from_options(
       frame = medicare_nest_dyf,
       connection_type = "s3",
       connection_options = {"path": "s3://glue-sample-target/output-dir/
medicare_parquet"},
       format = "parquet")
```

AWS Glue PySpark 拡張機能リファレンス

AWS Glue には、PySpark (Python の関連言語) 向けに、以下の拡張機能が追加されています。

- getResolvedOptions を使用して、パラメータにアクセスする

- PySpark 拡張子型
- DynamicFrame クラス
- DynamicFrameCollection クラス
- DynamicFrameWriter クラス
- DynamicFrameReader クラス
- GlueContext クラス

`getResolvedOptions` を使用して、パラメータにアクセスする

AWS Glue `getResolvedOptions(args, options)` ユーティリティ関数を使用すると、ジョブの実行時にスクリプトに渡される引数にアクセスできます。この関数を使用するには、まず AWS Glue `utils` モジュールと `sys` モジュールからインポートします。

```
import sys
from awsglue.utils import getResolvedOptions
```

`getResolvedOptions(args, options)`

- `args` - `sys.argv` に含まれる引数のリスト。
- `options` - 取得したい引数名の Python 配列。

Example JobRun に渡された引数を取得する

スクリプト (Lambda 関数など) の中で、JobRun を作成した場合を考えます。

```
response = client.start_job_run(
             JobName = 'my_test_Job',
             Arguments = {
               '--day_partition_key':   'partition_0',
               '--hour_partition_key':  'partition_1',
               '--day_partition_value':  day_partition_value,
               '--hour_partition_value': hour_partition_value } )
```

渡された引数を取得するには、次のように、`getResolvedOptions` 関数を使用できます。

```
import sys
```

```
from awsglue.utils import getResolvedOptions

args = getResolvedOptions(sys.argv,
                          ['JOB_NAME',
                           'day_partition_key',
                           'hour_partition_key',
                           'day_partition_value',
                           'hour_partition_value'])
print "The day-partition key is: ", args['day_partition_key']
print "and the day-partition value is: ", args['day_partition_value']
```

各引数は 2 つのハイフンで始まり、ハイフンなしでスクリプト内で参照されるように定義されていることに注意してください。引数では、ハイフンではなく、アンダースコアのみを使用します。引数の解決のために、この規則に従う必要があります。

PySpark 拡張子型

AWS Glue PySpark 拡張子で使用される型。

DataType

他の AWS Glue 型の基本クラス。

`__init__(properties={})`

- `properties` – データ型のプロパティ (オプション)。

`typeName(cls)`

AWS Glue 型クラスの種類 (つまり、"型" が末尾から削除されたクラス名) を返します。

- `cls` – AWS Glue から派生した `DataType` クラスインスタンス。

`jsonValue( )`

データ型とクラスのプロパティが含まれる JSON オブジェクトを返します。

```
  {
    "dataType": typeName,
```

```
    "properties": properties
  }
```

AtomicType およびシンプルデリバティブ

DataType クラスから継承して拡張し、すべての AWS Glue アトミックデータ型の基本クラスとして機能します。

`fromJsonValue(cls, json_value)`

JSON オブジェクトからの値を使用してクラスインスタンスを初期化します。

- `cls` – 初期化する AWS Glue 型のクラスインスタンス。
- `json_value` – キーと値のペアのロード元の JSON オブジェクト。

次の型は、AtomicType クラスのシンプルデリバティブです。

- `BinaryType` – バイナリデータ。
- `BooleanType` – ブール値。
- `ByteType` – バイト値。
- `DateType` – 日時値。
- `DoubleType` – 倍精度浮動小数点値。
- `IntegerType` – 整数値。
- `LongType` – 長整数値。
- `NullType` – null 値。
- `ShortType` – 短整数値。
- `StringType` – テキスト文字列。
- `TimestampType` – タイムスタンプ値 (通常は 1970 年 1 月 1 日からの秒数)。
- `UnknownType` – 未確認型の値。

DecimalType(AtomicType)

10 進数 (バイナリ 2 進数ではなく、10 進数で表記される数) を表わすため、AtomicType クラスから継承して拡張します。

__init__(precision=10, scale=2, properties={})

- precision – 10 進数の桁数 (オプション、デフォルトは 10)。
- scale – 小数点以下の桁数 (オプション、デフォルトは 2)。
- properties – 10 進数のプロパティ (オプション)。

EnumType(AtomicType)

有効なオプションの列挙を表すために、AtomicType クラスから継承して拡張します。

__init__(options)

- options – 列挙されているオプションのリスト。

コレクション型

- ArrayType(DataType)
- ChoiceType(DataType)
- MapType(DataType)
- フィールド (オブジェクト)
- StructType(DataType)
- EntityType(DataType)

ArrayType(DataType)

__init__(elementType=UnknownType(), properties={})

- elementType – 配列の要素の型 (オプション、デフォルトは UnknownType)。
- properties – 配列のプロパティ (オプション)。

ChoiceType(DataType)

__init__(choices=[], properties={})

- choices – 選択肢のリスト (オプション)。
- properties – これらのオプションのプロパティ (オプション)。

add(new_choice)

可能な選択肢のリストに、新しい選択肢を追加します。

- new_choice – 可能な選択肢のリストに追加する選択肢。

merge(new_choices)

新しい選択肢のリストを既存の選択肢のリストとマージします。

- new_choices – 既存の選択肢とマージする新しい選択肢のリスト。

MapType(DataType)

__init__(valueType=UnknownType, properties={})

- valueType – マップの値の型 (オプション、デフォルトは UnknownType)。
- properties – マップのプロパティ (オプション)。

フィールド (オブジェクト)

DataType から派生したオブジェクトからフィールドオブジェクトを作成します。

__init__(name, dataType, properties={})

- name – フィールドに割り当てる名前。
- dataType – フィールド作成元のオブジェクト。
- properties – フィールドのプロパティ (オプション)。

StructType(DataType)

データ構造を定義します (struct)。

__init__(fields=[], properties={})

- fields – フィールドのリスト (Field 型) 構造に含めます (オプション)。
- properties – 構造のプロパティ (オプション)。

add(field)

- field – 構造に追加するオブジェクトの Field 型。

hasField(field)

この構造に同じ名前のフィールドがある場合は True を、そうでない場合は False を返します。

- field – フィールド名、または名前が使用される Field 型のオブジェクト。

getField(field)

- field – 名前が使用される Field 型のフィールド名またはオブジェクト。構造に同じ名前のフィールドがある場合は、返されます。

EntityType(DataType)

__init__(entity, base_type, properties)

このクラスは、まだ実装されていません。

その他の型

- DataSource (オブジェクト)
- DataSink (オブジェクト)

DataSource (オブジェクト)

__init__(j_source, sql_ctx, name)

- j_source – データソース。
- sql_ctx – SQL コンテキスト。
- name – データソース名。

setFormat(format, **options)

- format – データソースを設定する形式。

- options – データソースに設定するオプションのコレクション。形式オプションの詳細については、「the section called “データ形式に関するオプション”」を参照してください。

getFrame()

データソースに DynamicFrame を返します。

DataSink (オブジェクト)

__init__(j_sink, sql_ctx)

- j_sink – 作成するシンク。
- sql_ctx – データシンクの SQL コンテキスト。

setFormat(format, **options)

- format – データシンクを設定する形式。
- options – データシンクに設定するオプションのコレクション。形式オプションの詳細については、「the section called “データ形式に関するオプション”」を参照してください。

setAccumulableSize(size)

- size – 設定する累積サイズ (バイト単位)。

writeFrame(dynamic_frame, info="")

- dynamic_frame - 書き込む DynamicFrame。
- info – DynamicFrame に関する情報 (オプション)。

write(dynamic_frame_or_dfc, info="")

DynamicFrame または DynamicFrameCollection を書き込みます。

- dynamic_frame_or_dfc – 書き込む DynamicFrame オブジェクトまたは DynamicFrameCollection オブジェクトのいずれか。

- info – 書き込む DynamicFrame または DynamicFrames に関する情報 (オプション)。

DynamicFrame クラス

Apache Spark の主要な抽象化の 1 つは SparkSQL DataFrame で、これは R と Pandas にある DataFrame 構造に似ています。DataFrame はテーブルと似ており、機能スタイル (マップ/リデュース/フィルター/その他) 操作と SQL 操作 (選択、プロジェクト、集計) をサポートしています。

DataFrames は、強力で広く使用されていますが、抽出、変換、ロード (ETL) 操作に関しては制限があります。最も重要なのは、データをロードする前にスキーマを指定する必要があることです。SparkSQL は、データに対してパスを 2 つ作ることでこれを解決します。この 1 つ目はスキーマの推定を行い、2 つ目はデータをロードします。ただし、この推測は限定されており、実際の煩雑なデータには対応しません。例えば、同じフィールドが異なるレコードの異なるタイプである可能性があります。Apache Spark は、多くの場合、作業を中断して、元のフィールドテキストを使用して型を string として報告します。これは正しくない可能性があり、スキーマの不一致を解決する方法を細かくコントロールする必要があるかもしれません。また、大規模なデータセットの場合、ソースデータに対する追加パスが非常に高価になる可能性があります。

これらの制限に対応するために、AWS Glue では DynamicFrame を導入しています。DynamicFrame は、DataFrame と似ていますが、各レコードが自己記述できるため、最初はスキーマは必要ありません。代わりに、AWS Glue は必要に応じてオンザフライでスキーマを計算し、選択 (または共用) タイプを使用してスキーマの不一致を明示的にエンコードします。これらの不整合を解決して、固定スキーマを必要とするデータストアとデータセットを互換性のあるものにできます。

同様に、DynamicRecord は DynamicFrame 内の論理レコードを表します。これは、Spark DataFrame の行と似ていますが、自己記述型であり、固定スキーマに適合しないデータに使用できます。PySpark で AWS Glue を使用する場合、通常は独立した操作 DynamicRecords は行いません。むしろ、データセットをその DynamicFrame を使ってまとめて変換します。

スキーマの不一致を解決したら、DynamicFrames を DataFrames との間で変換することができます。

— construction —

- __init__
- fromDF
- toDF

__init__

`__init__(jdf, glue_ctx, name)`

- `jdf` - Java 仮想マシン (JVM) 内のデータフレームへの参照。
- `glue_ctx` – GlueContext クラス オブジェクト。
- `name` - オプションの名前文字列。デフォルトでは空。

fromDF

`fromDF(dataframe, glue_ctx, name)`

`DataFrame` フィールドを `DynamicRecord` に変換することにより、`DataFrame` を `DynamicFrame` に変換します。新しい `DynamicFrame` を返します。

`DynamicRecord` は `DynamicFrame` 内の論理レコードを表します。これは、自己記述型であり、固定スキーマに適合しないデータに使用できる点を除いて、Spark `DataFrame` の行に似ています。

この関数は、`DataFrame` で名前と重複する列がすでに解決されていることを前提としています。

- `dataframe` - 変換する Apache Spark SQL `DataFrame` (必須)。
- `glue_ctx` - この変換のコンテキストを指定する GlueContext クラス オブジェクト (必須)。
- `name` - 結果 `DynamicFrame` の名前 (AWS Glue 3.0 以降はオプション)。

toDF

`toDF(options)`

`DynamicRecords` を `DataFrame` フィールドに変換することにより、`DynamicFrame` を Apache Spark `DataFrame` に変換します。新しい `DataFrame` を返します。

`DynamicRecord` は `DynamicFrame` 内の論理レコードを表します。これは、自己記述型であり、固定スキーマに適合しないデータに使用できる点を除いて、Spark `DataFrame` の行に似ています。

- `options` - オプションのリスト。変換プロセスの追加オプションを指定できます。「options」パラメータで使用できる有効なオプションは次のとおりです。
  - `format` – データの形式 (json、csv、parquet など) を指定します。
  - `separater or sep` – CSV ファイルの場合、区切り文字を指定します。

- header – CSV ファイルの場合、最初の行がヘッダーであるかどうかを示します (true/false)。
- inferSchema – Spark にスキーマを自動的に推測するように指示します (true/false)。

「toDF」メソッドで「options」パラメータを使用する例を次に示します。

```
from awsglue.context import GlueContext
from awsglue.dynamicframe import DynamicFrame
from pyspark.context import SparkContext

sc = SparkContext()
glueContext = GlueContext(sc)

csv_dyf = glueContext.create_dynamic_frame.from_options(
  connection_type="s3",
  connection_options={"paths": ["s3://my-bucket/path/to/csv/"]},
  format="csv"
)
csv_cf = csv_dyf.toDF(options={
  "separator": ",",
  "header": "true",
  "inferSchema": "true"
})
```

Project と Cast アクションタイプを選択した場合、ターゲットのタイプを指定します。次に例を示します。

```
>>>toDF([ResolveOption("a.b.c", "KeepAsStruct")])
>>>toDF([ResolveOption("a.b.c", "Project", DoubleType())])
```

— information —

- count
- スキーマ
- printSchema
- show
- repartition
- coalesce

count

`count( )` - 基盤となる DataFrame の行数を返します。

スキーマ

`schema( )` - この DynamicFrame のスキーマを返します。使用できない場合は、基盤となる DataFrame のスキーマを返します。

このスキーマを構成する DynamicFrame タイプの詳細については、「the section called “型”」を参照してください。

printSchema

`printSchema( )` - 基盤となる DataFrame のスキーマを表示します。

show

`show(num_rows)` - 基盤となる DataFrame から、指定された行数を表示します。

repartition

`repartition(numPartitions)` – numPartitions 個のパーティションを含む新しい DynamicFrame を返します。

coalesce

`coalesce(numPartitions)` – numPartitions 個のパーティションを含む新しい DynamicFrame を返します。

— transforms —

- apply_mapping
- drop_fields
- フィルター
- 結合
- マップ
- mergeDynamicFrame
- 関係付け

- rename_field
- resolveChoice
- select_fields
- スピゴット
- split_fields
- split_rows
- アンボックス
- the section called “union”
- ネスト解除
- unnest_ddb_json
- 書き込み

apply_mapping

apply_mapping(mappings, transformation_ctx="", info="", stageThreshold=0, totalThreshold=0)

DynamicFrame に宣言型のマッピングを適用し、それらのマッピングが適用された新しい DynamicFrame を指定したフィールドに返します。未指定のフィールドは新しい DynamicFrame から除外されます。

- mappings — マッピングタプルのリスト (必須)。それぞれが (ソース列、ソースタイプ、ターゲット列、ターゲットタイプ) で構成されます。

  ソース列で、名前にドット「.」が含まれている場合、バックティック「``」で囲む必要があります。例えば、this.old.name (文字列) を thisNewName にマッピングするには、次のタプルを使用します。

  ```
  ("`this.old.name`", "string", "thisNewName", "string")
  ```

- transformation_ctx - 状態情報を識別するために使用される一意の文字列 (オプション)。
- info - この変換のエラー報告に関連付ける文字列 (オプション)。
- stageThreshold - この変換中に発生した、プロセスがエラーを出力するエラーの数 (オプション)。デフォルトはゼロで、エラーを出力しないことを示します。

- `totalThreshold` - この変換までに発生したエラーのうち、プロセスがエラーを出力するエラーの数 (オプション)。デフォルトはゼロで、エラーを出力しないことを示します。

例: apply_mapping を使用してフィールドの名前を変更し、フィールドタイプを変更する

次のコード例は、`apply_mapping` メソッドを使用して、選択したフィールドの名前を変更し、フィールドタイプを変更する方法を示しています。

> **Note**
>
> この例で使用されているデータセットにアクセスするには、「コード例: データの結合と関係付け」を参照し、「ステップ 1: Amazon S3 バケット内のデータをクロールする」の手順に従います。

```
# Example: Use apply_mapping to reshape source data into
# the desired column names and types as a new DynamicFrame

from pyspark.context import SparkContext
from awsglue.context import GlueContext

# Create GlueContext
sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

# Create a DynamicFrame and view its schema
persons = glueContext.create_dynamic_frame.from_catalog(
    database="legislators", table_name="persons_json"
)
print("Schema for the persons DynamicFrame:")
persons.printSchema()

# Select and rename fields, change field type
print("Schema for the persons_mapped DynamicFrame, created with apply_mapping:")
persons_mapped = persons.apply_mapping(
    [
        ("family_name", "String", "last_name", "String"),
        ("name", "String", "first_name", "String"),
        ("birth_date", "String", "date_of_birth", "Date"),
    ]
)
```

```
persons_mapped.printSchema()
```

Output

```
Schema for the persons DynamicFrame:
root
|-- family_name: string
|-- name: string
|-- links: array
|    |-- element: struct
|    |    |-- note: string
|    |    |-- url: string
|-- gender: string
|-- image: string
|-- identifiers: array
|    |-- element: struct
|    |    |-- scheme: string
|    |    |-- identifier: string
|-- other_names: array
|    |-- element: struct
|    |    |-- lang: string
|    |    |-- note: string
|    |    |-- name: string
|-- sort_name: string
|-- images: array
|    |-- element: struct
|    |    |-- url: string
|-- given_name: string
|-- birth_date: string
|-- id: string
|-- contact_details: array
|    |-- element: struct
|    |    |-- type: string
|    |    |-- value: string
|-- death_date: string

Schema for the persons_mapped DynamicFrame, created with apply_mapping:
root
|-- last_name: string
|-- first_name: string
|-- date_of_birth: date
```

drop_fields

drop_fields(paths, transformation_ctx="", info="", stageThreshold=0, totalThreshold=0)

FlatMap クラス 変換を呼び出して、DynamicFrame からフィールドを削除します。指定されたフィールドが削除された新しい DynamicFrame を返します。

- paths - 文字列のリスト。それぞれに、ドロップするフィールドノードへのフルパスが含まれます。ドット表記を使用して、ネストされたフィールドを指定できます。例えば、フィールド first がツリー内のフィールド name の子である場合は、パスに "name.first" を指定します。

  フィールドノードの名前にリテラル . が含まれている場合は、その名前をバックティック (`) で囲む必要があります。
- transformation_ctx - 状態情報を識別するために使用される一意の文字列 (オプション)。
- info - この変換のエラー報告に関連付ける文字列 (オプション)。
- stageThreshold - この変換中に発生した、プロセスがエラーを出力するエラーの数 (オプション)。デフォルトはゼロで、エラーを出力しないことを示します。
- totalThreshold - この変換までに発生したエラーのうち、プロセスがエラーを出力するエラーの数 (オプション)。デフォルトはゼロで、エラーを出力しないことを示します。

例: drop_fields を使用して **DynamicFrame** からフィールドを削除する

このコード例では、drop_fields メソッドを使用して、選択したトップレベルフィールドとネストされたフィールドを DynamicFrame から削除します。

データセットの例

この例では、コード内の EXAMPLE-FRIENDS-DATA テーブルで表される次のデータセットを使用します。

```
{"name": "Sally", "age": 23, "location": {"state": "WY", "county": "Fremont"},
 "friends": []}
{"name": "Varun", "age": 34, "location": {"state": "NE", "county": "Douglas"},
 "friends": [{"name": "Arjun", "age": 3}]}
{"name": "George", "age": 52, "location": {"state": "NY"}, "friends": [{"name":
 "Fred"}, {"name": "Amy", "age": 15}]}
{"name": "Haruki", "age": 21, "location": {"state": "AK", "county": "Denali"}}
{"name": "Sheila", "age": 63, "friends": [{"name": "Nancy", "age": 22}]}
```

コードの例

```
# Example: Use drop_fields to remove top-level and nested fields from a DynamicFrame.
# Replace MY-EXAMPLE-DATABASE with your Glue Data Catalog database name.
# Replace EXAMPLE-FRIENDS-DATA with your table name.

from pyspark.context import SparkContext
from awsglue.context import GlueContext

# Create GlueContext
sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

# Create a DynamicFrame from Glue Data Catalog
glue_source_database = "MY-EXAMPLE-DATABASE"
glue_source_table = "EXAMPLE-FRIENDS-DATA"

friends = glueContext.create_dynamic_frame.from_catalog(
    database=glue_source_database, table_name=glue_source_table
)
print("Schema for friends DynamicFrame before calling drop_fields:")
friends.printSchema()

# Remove location.county, remove friends.age, remove age
friends = friends.drop_fields(paths=["age", "location.county", "friends.age"])
print("Schema for friends DynamicFrame after removing age, county, and friend age:")
friends.printSchema()
```

Output

```
Schema for friends DynamicFrame before calling drop_fields:
root
|-- name: string
|-- age: int
|-- location: struct
|    |-- state: string
|    |-- county: string
|-- friends: array
|    |-- element: struct
|    |    |-- name: string
|    |    |-- age: int

Schema for friends DynamicFrame after removing age, county, and friend age:
```

```
root
|-- name: string
|-- location: struct
|    |-- state: string
|-- friends: array
|    |-- element: struct
|    |    |-- name: string
```

フィルター

filter(f, transformation_ctx="", info="", stageThreshold=0, totalThreshold=0)

指定された述語関数 f を満たす入力 DynamicFrame 内のすべての DynamicRecords を含む、新しい DynamicFrame を返します。

- f - DynamicFrame に適用する述語関数。この関数は DynamicRecord を引数として取り、DynamicRecord がフィルター要件を満たす場合は True を返し、そうでない場合は False を返します (必須)。

  DynamicRecord は DynamicFrame 内の論理レコードを表します。これは、自己記述型であり、固定スキーマに準拠しないデータに使用できる点を除いて、Spark DataFrame の行に似ています。
- transformation_ctx - 状態情報を識別するために使用される一意の文字列 (オプション)。
- info - この変換のエラー報告に関連付ける文字列 (オプション)。
- stageThreshold - この変換中に発生した、プロセスがエラーを出力するエラーの数 (オプション)。デフォルトはゼロで、エラーを出力しないことを示します。
- totalThreshold - この変換までに発生したエラーのうち、プロセスがエラーを出力するエラーの数 (オプション)。デフォルトはゼロで、エラーを出力しないことを示します。

例: フィルターを使用して、フィールドのフィルタリングされた選択内容を取得する

この例では、filter メソッドを使用して、別の DynamicFrame のフィールドのフィルタリングされた選択内容を含む新しい DynamicFrame を作成します。

map メソッドと同様に、filter は、元の DynamicFrame の各レコードに適用される引数として関数を取ります。この関数はレコードを入力として受け取り、ブール値を返します。戻り値が true の

場合、レコードは結果として生じる DynamicFrame に含まれます。false の場合、レコードは除外されます。

> **Note**
>
> この例で使用されているデータセットにアクセスするには、「コード例: ResolveChoice、Lambda、および ApplyMapping を使用したデータ準備」を参照し、「ステップ 1: Amazon S3 バケット内のデータをクロールする」の手順に従います。

```
# Example: Use filter to create a new DynamicFrame
# with a filtered selection of records

from pyspark.context import SparkContext
from awsglue.context import GlueContext

# Create GlueContext
sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

# Create DynamicFrame from Glue Data Catalog
medicare = glueContext.create_dynamic_frame.from_options(
    "s3",
    {
        "paths": [
            "s3://awsglue-datasets/examples/medicare/Medicare_Hospital_Provider.csv"
        ]
    },
    "csv",
    {"withHeader": True},
)

# Create filtered DynamicFrame with custom lambda
# to filter records by Provider State and Provider City
sac_or_mon = medicare.filter(
    f=lambda x: x["Provider State"] in ["CA", "AL"]
    and x["Provider City"] in ["SACRAMENTO", "MONTGOMERY"]
)

# Compare record counts
print("Unfiltered record count: ", medicare.count())
```

```
print("Filtered record count:  ", sac_or_mon.count())
```

Output

```
Unfiltered record count:  163065
Filtered record count:   564
```

結合

join(paths1, paths2, frame2, transformation_ctx="", info="", stageThreshold=0, totalThreshold=0)

別の DynamicFrame と等価結合を実行し、結果の DynamicFrame を返します。

- paths1 - 結合するこのフレームのキーのリスト。
- paths2 - 結合する別のフレームのキーのリスト。
- frame2 - 結合する他の DynamicFrame。
- transformation_ctx - 状態情報を識別するために使用される一意の文字列 (オプション)。
- info - この変換のエラー報告に関連付ける文字列 (オプション)。
- stageThreshold - この変換中に発生した、プロセスがエラーを出力するエラーの数 (オプション)。デフォルトはゼロで、エラーを出力しないことを示します。
- totalThreshold - この変換までに発生したエラーのうち、プロセスがエラーを出力するエラーの数 (オプション)。デフォルトはゼロで、エラーを出力しないことを示します。

例: 結合を使用して **DynamicFrames** を結合する

この例では、join メソッドを使用して 3 つの DynamicFrames で結合を実行します。AWSGlue は、入力されたフィールドキーに基づいて結合を実行します。結果として生じる DynamicFrame には、指定されたキーが一致する元の 2 つのフレームからの行が含まれます。

join 変換では、すべてのフィールドがそのまま保持されることに注意してください。これは、一致するように指定したフィールドが冗長で同じキーが含まれていても、結果として生じる DynamicFrame に表示されることを意味します。この例では、drop_fields を使用して、結合後にこれらの冗長なキーを削除します。

> **Note**
>
> この例で使用されているデータセットにアクセスするには、「コード例: データの結合と関係付け」を参照し、「ステップ 1: Amazon S3 バケット内のデータをクロールする」の手順に従います。

```
# Example: Use join to combine data from three DynamicFrames

from pyspark.context import SparkContext
from awsglue.context import GlueContext

# Create GlueContext
sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

# Load DynamicFrames from Glue Data Catalog
persons = glueContext.create_dynamic_frame.from_catalog(
    database="legislators", table_name="persons_json"
)
memberships = glueContext.create_dynamic_frame.from_catalog(
    database="legislators", table_name="memberships_json"
)
orgs = glueContext.create_dynamic_frame.from_catalog(
    database="legislators", table_name="organizations_json"
)
print("Schema for the persons DynamicFrame:")
persons.printSchema()
print("Schema for the memberships DynamicFrame:")
memberships.printSchema()
print("Schema for the orgs DynamicFrame:")
orgs.printSchema()

# Join persons and memberships by ID
persons_memberships = persons.join(
    paths1=["id"], paths2=["person_id"], frame2=memberships
)

# Rename and drop fields from orgs
# to prevent field name collisions with persons_memberships
orgs = (
    orgs.drop_fields(["other_names", "identifiers"])
```

```
    .rename_field("id", "org_id")
    .rename_field("name", "org_name")
)

# Create final join of all three DynamicFrames
legislators_combined = orgs.join(
    paths1=["org_id"], paths2=["organization_id"], frame2=persons_memberships
).drop_fields(["person_id", "org_id"])

# Inspect the schema for the joined data
print("Schema for the new legislators_combined DynamicFrame:")
legislators_combined.printSchema()
```

Output

```
Schema for the persons DynamicFrame:
root
|-- family_name: string
|-- name: string
|-- links: array
|    |-- element: struct
|    |    |-- note: string
|    |    |-- url: string
|-- gender: string
|-- image: string
|-- identifiers: array
|    |-- element: struct
|    |    |-- scheme: string
|    |    |-- identifier: string
|-- other_names: array
|    |-- element: struct
|    |    |-- lang: string
|    |    |-- note: string
|    |    |-- name: string
|-- sort_name: string
|-- images: array
|    |-- element: struct
|    |    |-- url: string
|-- given_name: string
|-- birth_date: string
|-- id: string
|-- contact_details: array
|    |-- element: struct
```

```
|    |    |-- type: string
|    |    |-- value: string
|-- death_date: string

Schema for the memberships DynamicFrame:
root
|-- area_id: string
|-- on_behalf_of_id: string
|-- organization_id: string
|-- role: string
|-- person_id: string
|-- legislative_period_id: string
|-- start_date: string
|-- end_date: string

Schema for the orgs DynamicFrame:
root
|-- identifiers: array
|    |-- element: struct
|    |    |-- scheme: string
|    |    |-- identifier: string
|-- other_names: array
|    |-- element: struct
|    |    |-- lang: string
|    |    |-- note: string
|    |    |-- name: string
|-- id: string
|-- classification: string
|-- name: string
|-- links: array
|    |-- element: struct
|    |    |-- note: string
|    |    |-- url: string
|-- image: string
|-- seats: int
|-- type: string

Schema for the new legislators_combined DynamicFrame:
root
|-- role: string
|-- seats: int
|-- org_name: string
|-- links: array
|    |-- element: struct
```

```
|    |    |-- note: string
|    |    |-- url: string
|-- type: string
|-- sort_name: string
|-- area_id: string
|-- images: array
|    |-- element: struct
|    |    |-- url: string
|-- on_behalf_of_id: string
|-- other_names: array
|    |-- element: struct
|    |    |-- note: string
|    |    |-- name: string
|    |    |-- lang: string
|-- contact_details: array
|    |-- element: struct
|    |    |-- type: string
|    |    |-- value: string
|-- name: string
|-- birth_date: string
|-- organization_id: string
|-- gender: string
|-- classification: string
|-- legislative_period_id: string
|-- identifiers: array
|    |-- element: struct
|    |    |-- scheme: string
|    |    |-- identifier: string
|-- image: string
|-- given_name: string
|-- start_date: string
|-- family_name: string
|-- id: string
|-- death_date: string
|-- end_date: string
```

マップ

map(f, transformation_ctx="", info="", stageThreshold=0, totalThreshold=0)

指定したマッピング関数を元の DynamicFrame のすべてのレコードに適用した結果の新しい DynamicFrame を返します。

- f - DynamicFrame 内のすべてのレコードに適用されるマッピング関数。この関数は、DynamicRecord を引数として取り、新しい DynamicRecord を返す必要があります (必須)。

  DynamicRecord は DynamicFrame 内の論理レコードを表します。これは、自己記述型であり、固定スキーマに準拠しないデータに使用できる点を除いて、Apache Spark DataFrame の行に似ています。
- transformation_ctx - 状態情報を識別するために使用される一意の文字列 (オプション)。
- info - 変換のエラーに関連付けられた文字列 (オプション)。
- stageThreshold – エラーを出力する前に、変換で発生する可能性のあるエラーの最大数 (オプション)。デフォルト値は 0 です。
- totalThreshold – エラーの出力を処理する前に、全体で発生する可能性のあるエラーの最大数 (オプション)。デフォルト値は 0 です。

例: マッピングを使用して、**DynamicFrame** のすべてのレコードに関数を適用する

この例は、map メソッドを使用して、関数を DynamicFrame のすべてのレコードに適用する方法を示しています。具体的には、この例では、複数のアドレスフィールドを単一の struct タイプに結合するため、MergeAddress という名前の関数を各レコードに適用します。

> **Note**
>
> この例で使用されているデータセットにアクセスするには、「コード例: ResolveChoice、Lambda、および ApplyMapping を使用したデータ準備」を参照し、「ステップ 1: Amazon S3 バケット内のデータをクロールする」の手順に従います。

```
# Example: Use map to combine fields in all records
# of a DynamicFrame

from pyspark.context import SparkContext
from awsglue.context import GlueContext

# Create GlueContext
sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

# Create a DynamicFrame and view its schema
```

```
medicare = glueContext.create_dynamic_frame.from_options(
        "s3",
        {"paths": ["s3://awsglue-datasets/examples/medicare/
Medicare_Hospital_Provider.csv"]},
        "csv",
        {"withHeader": True})
print("Schema for medicare DynamicFrame:")
medicare.printSchema()

# Define a function to supply to the map transform
# that merges address fields into a single field
def MergeAddress(rec):
  rec["Address"] = {}
  rec["Address"]["Street"] = rec["Provider Street Address"]
  rec["Address"]["City"] = rec["Provider City"]
  rec["Address"]["State"] = rec["Provider State"]
  rec["Address"]["Zip.Code"] = rec["Provider Zip Code"]
  rec["Address"]["Array"] = [rec["Provider Street Address"], rec["Provider City"],
 rec["Provider State"], rec["Provider Zip Code"]]
  del rec["Provider Street Address"]
  del rec["Provider City"]
  del rec["Provider State"]
  del rec["Provider Zip Code"]
  return rec


# Use map to apply MergeAddress to every record
mapped_medicare = medicare.map(f = MergeAddress)
print("Schema for mapped_medicare DynamicFrame:")
mapped_medicare.printSchema()
```

Output

```
Schema for medicare DynamicFrame:
root
|-- DRG Definition: string
|-- Provider Id: string
|-- Provider Name: string
|-- Provider Street Address: string
|-- Provider City: string
|-- Provider State: string
|-- Provider Zip Code: string
|-- Hospital Referral Region Description: string
```

```
|-- Total Discharges: string
|-- Average Covered Charges: string
|-- Average Total Payments: string
|-- Average Medicare Payments: string

Schema for mapped_medicare DynamicFrame:
root
|-- Average Total Payments: string
|-- Average Covered Charges: string
|-- DRG Definition: string
|-- Average Medicare Payments: string
|-- Hospital Referral Region Description: string
|-- Address: struct
|    |-- Zip.Code: string
|    |-- City: string
|    |-- Array: array
|    |    |-- element: string
|    |-- State: string
|    |-- Street: string
|-- Provider Id: string
|-- Total Discharges: string
|-- Provider Name: string
```

mergeDynamicFrame

mergeDynamicFrame(stage_dynamic_frame, primary_keys, transformation_ctx = "", options = {}, info = "", stageThreshold = 0, totalThreshold = 0)

レコードを識別するために、この DynamicFrame を指定されたプライマリキーに基づくステージング DynamicFrame とマージします。重複レコード (同じプライマリキーを持つレコード) は重複除外されません。ステージングフレームに一致するレコードがない場合、すべてのレコード (重複を含む) がソースから保持されます。ステージングフレームに一致するレコードがある場合、ステージングフレームのレコードによって、AWS Glue のソースのレコードが上書きされます。

- stage_dynamic_frame – マージするステージング DynamicFrame。
- primary_keys – ソースおよびステージング動的フレームからのレコードを照合する、プライマリキーフィールドのリスト。
- transformation_ctx – 現在の変換に関するメタデータを取得するために使用される一意の文字列 (オプション)。
- options – この変換に関する追加情報を提供する、JSON の名前と値のペアを示す文字列。この引数は現在使用されていません。

- info – String。この変換でのエラーに関連付けられる任意の文字列。
- stageThreshold – Long。指定された変換で処理がエラーアウトする必要があるエラーの数。
- totalThreshold – Long。この変換までに発生したエラーのうち、処理でエラーを出力する必要があるエラーの合計数。

このメソッドは、この DynamicFrame をステージング DynamicFrame とマージして取得した新しい DynamicFrame を返します。

以下の場合、返される DynamicFrame にはレコード A が含まれます。

- A がソースフレームとステージングフレームの両方に存在する場合、ステージングフレームの A が返されます。
- A がソーステーブルに存在し、A.primaryKeys が stagingDynamicFrame に存在しない場合、A はステージングテーブルで更新されていません。

ソースフレームとステージングフレームが、同じスキーマを持つ必要はありません。

例: mergeDynamicFrame を使用して、プライマリキーに基づいて 2 つの **DynamicFrames** をマージする

次のコード例は、mergeDynamicFrame メソッドを使用して、プライマリキー id に基づいて DynamicFrame を「ステージング」DynamicFrame とマージする方法を示しています。

データセットの例

この例では、split_rows_collection と呼ばれる DynamicFrameCollection からの 2 つの DynamicFrames を使用しています。次のリストに示しているのは、split_rows_collection のキーです。

```
dict_keys(['high', 'low'])
```

コードの例

```
# Example: Use mergeDynamicFrame to merge DynamicFrames
# based on a set of specified primary keys

from pyspark.context import SparkContext
from awsglue.context import GlueContext
```

```
from awsglue.transforms import SelectFromCollection

# Inspect the original DynamicFrames
frame_low = SelectFromCollection.apply(dfc=split_rows_collection, key="low")
print("Inspect the DynamicFrame that contains rows where ID < 10")
frame_low.toDF().show()

frame_high = SelectFromCollection.apply(dfc=split_rows_collection, key="high")
print("Inspect the DynamicFrame that contains rows where ID > 10")
frame_high.toDF().show()

# Merge the DynamicFrames based on the "id" primary key
merged_high_low = frame_high.mergeDynamicFrame(
    stage_dynamic_frame=frame_low, primary_keys=["id"]
)

# View the results where the ID is 1 or 20
print("Inspect the merged DynamicFrame that contains the combined rows")
merged_high_low.toDF().where("id = 1 or id= 20").orderBy("id").show()
```

Output

```
Inspect the DynamicFrame that contains rows where ID < 10
+---+-----+-----------------------+-----------------------+
| id|index|contact_details.val.type|contact_details.val.value|
+---+-----+-----------------------+-----------------------+
|  1|    0|                    fax|           202-225-3307|
|  1|    1|                  phone|           202-225-5731|
|  2|    0|                    fax|           202-225-3307|
|  2|    1|                  phone|           202-225-5731|
|  3|    0|                    fax|           202-225-3307|
|  3|    1|                  phone|           202-225-5731|
|  4|    0|                    fax|           202-225-3307|
|  4|    1|                  phone|           202-225-5731|
|  5|    0|                    fax|           202-225-3307|
|  5|    1|                  phone|           202-225-5731|
|  6|    0|                    fax|           202-225-3307|
|  6|    1|                  phone|           202-225-5731|
|  7|    0|                    fax|           202-225-3307|
|  7|    1|                  phone|           202-225-5731|
|  8|    0|                    fax|           202-225-3307|
|  8|    1|                  phone|           202-225-5731|
|  9|    0|                    fax|           202-225-3307|
```

```
|  9|    1|                   phone|             202-225-5731|
| 10|    0|                     fax|             202-225-6328|
| 10|    1|                   phone|             202-225-4576|
+---+-----+----------------------+-------------------------+
only showing top 20 rows

Inspect the DynamicFrame that contains rows where ID > 10
+---+-----+----------------------+-------------------------+
| id|index|contact_details.val.type|contact_details.val.value|
+---+-----+----------------------+-------------------------+
| 11|    0|                     fax|             202-225-6328|
| 11|    1|                   phone|             202-225-4576|
| 11|    2|                 twitter|           RepTrentFranks|
| 12|    0|                     fax|             202-225-6328|
| 12|    1|                   phone|             202-225-4576|
| 12|    2|                 twitter|           RepTrentFranks|
| 13|    0|                     fax|             202-225-6328|
| 13|    1|                   phone|             202-225-4576|
| 13|    2|                 twitter|           RepTrentFranks|
| 14|    0|                     fax|             202-225-6328|
| 14|    1|                   phone|             202-225-4576|
| 14|    2|                 twitter|           RepTrentFranks|
| 15|    0|                     fax|             202-225-6328|
| 15|    1|                   phone|             202-225-4576|
| 15|    2|                 twitter|           RepTrentFranks|
| 16|    0|                     fax|             202-225-6328|
| 16|    1|                   phone|             202-225-4576|
| 16|    2|                 twitter|           RepTrentFranks|
| 17|    0|                     fax|             202-225-6328|
| 17|    1|                   phone|             202-225-4576|
+---+-----+----------------------+-------------------------+
only showing top 20 rows

Inspect the merged DynamicFrame that contains the combined rows
+---+-----+----------------------+-------------------------+
| id|index|contact_details.val.type|contact_details.val.value|
+---+-----+----------------------+-------------------------+
|  1|    0|                     fax|             202-225-3307|
|  1|    1|                   phone|             202-225-5731|
| 20|    0|                     fax|             202-225-5604|
| 20|    1|                   phone|             202-225-6536|
| 20|    2|                 twitter|                USRepLong|
+---+-----+----------------------+-------------------------+
```

関係付け

relationalize(root_table_name, staging_path, options, transformation_ctx="", info="", stageThreshold=0, totalThreshold=0)

DynamicFrame をリレーショナルデータベースに適合する形式に変換します。DynamicFrame の関係付けは、DynamoDB などの NoSQL 環境から MySQL などのリレーショナルデータベースにデータを移動する場合に特に便利です。

この変換は、ネストされた列をネスト解除し、配列の列をピボットすることでフレームのリストを生成します。フェーズのネスト解除時に生成された結合キーを使用して、ピボットされた配列の列をルートテーブルに結合できます。

- root_table_name - ルートテーブルの名前。
- staging_path – このメソッドがピボットテーブルのパーティションを CSV 形式で保存する保存先のパス (オプション)。ピボットされたテーブルはこのパスから読み取ります。
- options - オプションのパラメータのディクショナリ。
- transformation_ctx - 状態情報を識別するために使用される一意の文字列 (オプション)。
- info - この変換のエラー報告に関連付ける文字列 (オプション)。
- stageThreshold - この変換中に発生した、プロセスがエラーを出力するエラーの数 (オプション)。デフォルトはゼロで、エラーを出力しないことを示します。
- totalThreshold - この変換までに発生したエラーのうち、プロセスがエラーを出力するエラーの数 (オプション)。デフォルトはゼロで、エラーを出力しないことを示します。

例: relationalize を使用して、**DynamicFrame** のネストされたスキーマをフラット化する

このコード例では、relationalize メソッドを使用して、ネストされたスキーマをリレーショナルデータベースに適合する形式にフラット化します。

データセットの例

この例では、次のスキーマを持つ legislators_combined と呼ばれる DynamicFrame を使用しています。legislators_combined には、relationalize 変換によってフラット化される、links、images、contact_details などのネストされた複数のフィールドがあります。

```
root
|-- role: string
|-- seats: int
|-- org_name: string
|-- links: array
|    |-- element: struct
|    |    |-- note: string
|    |    |-- url: string
|-- type: string
|-- sort_name: string
|-- area_id: string
|-- images: array
|    |-- element: struct
|    |    |-- url: string
|-- on_behalf_of_id: string
|-- other_names: array
|    |-- element: struct
|    |    |-- note: string
|    |    |-- name: string
|    |    |-- lang: string
|-- contact_details: array
|    |-- element: struct
|    |    |-- type: string
|    |    |-- value: string
|-- name: string
|-- birth_date: string
|-- organization_id: string
|-- gender: string
|-- classification: string
|-- legislative_period_id: string
|-- identifiers: array
|    |-- element: struct
|    |    |-- scheme: string
|    |    |-- identifier: string
|-- image: string
|-- given_name: string
|-- start_date: string
|-- family_name: string
|-- id: string
|-- death_date: string
|-- end_date: string
```

コードの例

```
# Example: Use relationalize to flatten
# a nested schema into a format that fits
# into a relational database.
# Replace DOC-EXAMPLE-S3-BUCKET/tmpDir with your own location.

from pyspark.context import SparkContext
from awsglue.context import GlueContext

# Create GlueContext
sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

# Apply relationalize and inspect new tables
legislators_relationalized = legislators_combined.relationalize(
    "l_root", "s3://DOC-EXAMPLE-BUCKET/tmpDir"
)
legislators_relationalized.keys()

# Compare the schema of the contact_details
# nested field to the new relationalized table that
# represents it
legislators_combined.select_fields("contact_details").printSchema()
legislators_relationalized.select("l_root_contact_details").toDF().where(
    "id = 10 or id = 75"
).orderBy(["id", "index"]).show()
```

Output

次の出力では、contact_details と呼ばれるネストされたフィールドのスキーマを、relationalize 変換によって作成されたテーブルと比較できます。テーブルのレコードが id と呼ばれる外部キーと配列の位置を表す index 列を使用してメインテーブルにリンクされていることに注意してください。

```
dict_keys(['l_root', 'l_root_images', 'l_root_links', 'l_root_other_names',
 'l_root_contact_details', 'l_root_identifiers'])

root
|-- contact_details: array
|    |-- element: struct
|    |    |-- type: string
|    |    |-- value: string
```

```
+---+-----+-----------------------+-----------------------+
| id|index|contact_details.val.type|contact_details.val.value|
+---+-----+-----------------------+-----------------------+
| 10|    0|                    fax|           202-225-4160|
| 10|    1|                  phone|           202-225-3436|
| 75|    0|                    fax|           202-225-6791|
| 75|    1|                  phone|           202-225-2861|
| 75|    2|                twitter|             RepSamFarr|
+---+-----+-----------------------+-----------------------+
```

rename_field

rename_field(oldName, newName, transformation_ctx="", info="", stageThreshold=0, totalThreshold=0)

この DynamicFrame のフィールドの名前を変更し、フィールドの名前が変更された新しい DynamicFrame を返します。

- oldName - 名前を変更するノードへのフルパス。

  古い名前にドットが含まれている場合、RenameField はバックティック (`) で囲まなければ機能しません。例えば、this.old.name を thisNewName に置き換えるには、rename_field を次のように呼び出します。

  ```
  newDyF = oldDyF.rename_field("`this.old.name`", "thisNewName")
  ```

- newName - 完全パスとしての新しい名前。
- transformation_ctx - 状態情報を識別するために使用される一意の文字列 (オプション)。
- info - この変換のエラー報告に関連付ける文字列 (オプション)。
- stageThreshold - この変換中に発生した、プロセスがエラーを出力するエラーの数 (オプション)。デフォルトはゼロで、エラーを出力しないことを示します。
- totalThreshold - この変換までに発生したエラーのうち、プロセスがエラーを出力するエラーの数 (オプション)。デフォルトはゼロで、エラーを出力しないことを示します。

例: rename_field を使用して、**DynamicFrame** のフィールドの名前を変更する

このコード例では、rename_field メソッドを使用して DynamicFrame のフィールドの名前を変更します。この例では、メソッドの連鎖を使用して、複数のフィールドの名前を同時に変更していることに注意してください。

> **Note**
>
> この例で使用されているデータセットにアクセスするには、「コード例: データの結合と関係付け」を参照し、「ステップ 1: Amazon S3 バケット内のデータをクロールする」の手順に従います。

コードの例

```
# Example: Use rename_field to rename fields
# in a DynamicFrame

from pyspark.context import SparkContext
from awsglue.context import GlueContext

# Create GlueContext
sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

# Inspect the original orgs schema
orgs = glueContext.create_dynamic_frame.from_catalog(
    database="legislators", table_name="organizations_json"
)
print("Original orgs schema: ")
orgs.printSchema()

# Rename fields and view the new schema
orgs = orgs.rename_field("id", "org_id").rename_field("name", "org_name")
print("New orgs schema with renamed fields: ")
orgs.printSchema()
```

Output

```
Original orgs schema:
root
|-- identifiers: array
|    |-- element: struct
|    |    |-- scheme: string
|    |    |-- identifier: string
|-- other_names: array
|    |-- element: struct
|    |    |-- lang: string
```

```
|    |    |-- note: string
|    |    |-- name: string
|-- id: string
|-- classification: string
|-- name: string
|-- links: array
|    |-- element: struct
|    |    |-- note: string
|    |    |-- url: string
|-- image: string
|-- seats: int
|-- type: string

New orgs schema with renamed fields:
root
|-- identifiers: array
|    |-- element: struct
|    |    |-- scheme: string
|    |    |-- identifier: string
|-- other_names: array
|    |-- element: struct
|    |    |-- lang: string
|    |    |-- note: string
|    |    |-- name: string
|-- classification: string
|-- org_id: string
|-- org_name: string
|-- links: array
|    |-- element: struct
|    |    |-- note: string
|    |    |-- url: string
|-- image: string
|-- seats: int
|-- type: string
```

resolveChoice

resolveChoice(specs = None, choice = "" , database = None , table_name = None , transformation_ctx="", info="", stageThreshold=0, totalThreshold=0, catalog_id = None)

この DynamicFrame 内で選択タイプを解決し、新しい DynamicFrame を返します。

- specs – それぞれがタプルの形式である、解決すべき特定の曖昧要素のリスト: (field_path, action)。

  resolveChoice を使用するには 2 つの方法があります。最初の方法では、specs 引数により、特定のフィールドのシーケンスと解決方法を指定します。resolveChoice のもう 1 つのモードでは、choice 引数を使用して、すべての ChoiceTypes に対して単一の解決策を指定します。

  specs の値は、(field_path, action) ペアで構成されたタプルとして指定されます。field_path 値は特定のあいまいな要素を識別し、action 値は対応する解決を識別します。以下のアクションを指定できます。

  - cast:*type* – すべての値について、指定した型へのキャストを試みます。例: cast:int。
  - make_cols – それぞれの異なるタイプを *columnName*_*type* という名前の列に変換します。データをフラット化することで潜在的なあいまいさを解消します。例えば、columnA が int または string の場合、解決策は、作成された DynamicFrame に columnA_int および columnA_string という名前の 2 つの列を生成することです。
  - make_struct – struct を使用してデータを表現することで、潜在的なあいまいさを解決します。例えば、列のデータが int または string となる可能性がある場合、make_struct アクションを使用すると、作成された DynamicFrame に構造体の列が生成されます。各構造体には、int と string の両方が含まれています。
  - project:*type* – 可能なデータ型の 1 つにすべてのデータを投影することで、潜在的なあいまいさを解消します。例えば、列のデータが int または string の場合、project:string アクションを使用すると、すべての int 値が文字列に変換されている、作成された DynamicFrame に列が生成されます。

  field_path で配列を識別する場合は、あいまいさを避けるために配列名の後に空の角括弧を置きます。例えば、使用しているデータが次のように構造化されているとします。

```
"myList": [
  { "price": 100.00 },
  { "price": "$100.00" }
]
```

  field_path を "myList[].price" に設定し、action を "cast:double" に設定すると、文字列バージョンではなく、数値バージョンの料金を選択できます。

> **Note**
>
> specs パラメータおよび choice パラメータのうち 1 つのみを使用できます。specs パラメータが None ではない場合、choice パラメータは空の文字列である必要があります。逆に、choice が空の文字列ではない場合、specs パラメータは None である必要があります。

- choice – すべての ChoiceTypes について、単一の解決方法を指定します。このモードは、ランタイム前に ChoiceTypes の完全なリストが不明な場合に使用できます。この引数では、上記の specs 用のアクションに加えて、以下のアクションもサポートされています。
  - match_catalog – 各 ChoiceType について、指定した Data Catalog テーブル内の対応する型へのキャストを試みます。
- database – match_catalog アクションで使用する Data Catalog データベース。
- table_name – match_catalog アクションで使用する Data Catalog テーブル。
- transformation_ctx - 状態情報を識別するために使用される一意の文字列 (オプション)。
- info - この変換のエラー報告に関連付ける文字列 (オプション)。
- stageThreshold - この変換中に発生した、プロセスがエラーを出力するエラーの数 (オプション)。デフォルトはゼロで、エラーを出力しないことを示します。
- totalThreshold – この変換までに発生したエラーのうち、プロセスでエラーを出力する必要があるエラーの数 (オプション)。デフォルトはゼロで、プロセスでエラーを出力しないことを示します。
- catalog_id – 現在アクセスされている Data Catalog のカタログ ID (Data Catalog のアカウント ID)。None を設定した場合(デフォルト値) では、呼び出し元アカウントのカタログ ID が使用されます。

例: resolveChoice を使用して、複数の型を含む列を処理する

このコード例では、resolveChoice メソッドを使用して、複数の型の値を含む DynamicFrame 列の処理方法を指定します。この例は、型の異なる列を処理する一般的な 2 つの方法を示しています。

- 列を単一のデータ型にキャストします。
- すべての型を別々の列に保持します。

データセットの例

> **Note**
>
> この例で使用されているデータセットにアクセスするには、「コード例: ResolveChoice、Lambda、および ApplyMapping を使用したデータ準備」を参照し、「ステップ 1: Amazon S3 バケット内のデータをクロールする」の手順に従います。

この例では、次のスキーマを持つ `medicare` と呼ばれる `DynamicFrame` を使用しています。

```
root
|-- drg definition: string
|-- provider id: choice
|    |-- long
|    |-- string
|-- provider name: string
|-- provider street address: string
|-- provider city: string
|-- provider state: string
|-- provider zip code: long
|-- hospital referral region description: string
|-- total discharges: long
|-- average covered charges: string
|-- average total payments: string
|-- average medicare payments: string
```

コードの例

```
# Example: Use resolveChoice to handle
# a column that contains multiple types

from pyspark.context import SparkContext
from awsglue.context import GlueContext

# Create GlueContext
sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

# Load the input data and inspect the "provider id" column
medicare = glueContext.create_dynamic_frame.from_catalog(
    database="payments", table_name="medicare_hospital_provider_csv"
```

```
)
print("Inspect the provider id column:")
medicare.toDF().select("provider id").show()

# Cast provider id to type long
medicare_resolved_long = medicare.resolveChoice(specs=[("provider id", "cast:long")])
print("Schema after casting provider id to type long:")
medicare_resolved_long.printSchema()
medicare_resolved_long.toDF().select("provider id").show()

# Create separate columns
# for each provider id type
medicare_resolved_cols = medicare.resolveChoice(choice="make_cols")
print("Schema after creating separate columns for each type:")
medicare_resolved_cols.printSchema()
medicare_resolved_cols.toDF().select("provider id_long", "provider id_string").show()
```

Output

```
Inspect the 'provider id' column:
+-----------+
|provider id|
+-----------+
|   [10001,]|
|   [10005,]|
|   [10006,]|
|   [10011,]|
|   [10016,]|
|   [10023,]|
|   [10029,]|
|   [10033,]|
|   [10039,]|
|   [10040,]|
|   [10046,]|
|   [10055,]|
|   [10056,]|
|   [10078,]|
|   [10083,]|
|   [10085,]|
|   [10090,]|
|   [10092,]|
|   [10100,]|
|   [10103,]|
```

```
+-----------+
only showing top 20 rows

Schema after casting 'provider id' to type long:
root
|-- drg definition: string
|-- provider id: long
|-- provider name: string
|-- provider street address: string
|-- provider city: string
|-- provider state: string
|-- provider zip code: long
|-- hospital referral region description: string
|-- total discharges: long
|-- average covered charges: string
|-- average total payments: string
|-- average medicare payments: string

+-----------+
|provider id|
+-----------+
|      10001|
|      10005|
|      10006|
|      10011|
|      10016|
|      10023|
|      10029|
|      10033|
|      10039|
|      10040|
|      10046|
|      10055|
|      10056|
|      10078|
|      10083|
|      10085|
|      10090|
|      10092|
|      10100|
|      10103|
+-----------+
only showing top 20 rows
```

```
Schema after creating separate columns for each type:
root
|-- drg definition: string
|-- provider id_string: string
|-- provider id_long: long
|-- provider name: string
|-- provider street address: string
|-- provider city: string
|-- provider state: string
|-- provider zip code: long
|-- hospital referral region description: string
|-- total discharges: long
|-- average covered charges: string
|-- average total payments: string
|-- average medicare payments: string

+----------------+------------------+
|provider id_long|provider id_string|
+----------------+------------------+
|           10001|              null|
|           10005|              null|
|           10006|              null|
|           10011|              null|
|           10016|              null|
|           10023|              null|
|           10029|              null|
|           10033|              null|
|           10039|              null|
|           10040|              null|
|           10046|              null|
|           10055|              null|
|           10056|              null|
|           10078|              null|
|           10083|              null|
|           10085|              null|
|           10090|              null|
|           10092|              null|
|           10100|              null|
|           10103|              null|
+----------------+------------------+
only showing top 20 rows
```

select_fields

`select_fields(paths, transformation_ctx="", info="", stageThreshold=0, totalThreshold=0)`

選択したフィールドを含む新しい DynamicFrame を返します。

- `paths` - 文字列のリスト。各文字列は、選択する最上位ノードへのパスです。
- `transformation_ctx` - 状態情報を識別するために使用される一意の文字列 (オプション)。
- `info` - この変換のエラー報告に関連付ける文字列 (オプション)。
- `stageThreshold` - この変換中に発生した、プロセスがエラーを出力するエラーの数 (オプション)。デフォルトはゼロで、エラーを出力しないことを示します。
- `totalThreshold` - この変換までに発生したエラーのうち、プロセスがエラーを出力するエラーの数 (オプション)。デフォルトはゼロで、エラーを出力しないことを示します。

例: select_fields を使用して、選択したフィールドで新しい **DynamicFrame** を作成する

次のコード例は、`select_fields` メソッドを使用して、既存の DynamicFrame から選択されたフィールドのリストを使用し、新しい DynamicFrame を作成する方法を示しています。

> **Note**
>
> この例で使用されているデータセットにアクセスするには、「コード例: データの結合と関係付け」を参照し、「ステップ 1: Amazon S3 バケット内のデータをクロールする」の手順に従います。

```
# Example: Use select_fields to select specific fields from a DynamicFrame

from pyspark.context import SparkContext
from awsglue.context import GlueContext

# Create GlueContext
sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

# Create a DynamicFrame and view its schema
persons = glueContext.create_dynamic_frame.from_catalog(
    database="legislators", table_name="persons_json"
```

```
)
print("Schema for the persons DynamicFrame:")
persons.printSchema()

# Create a new DynamicFrame with chosen fields
names = persons.select_fields(paths=["family_name", "given_name"])
print("Schema for the names DynamicFrame, created with select_fields:")
names.printSchema()
names.toDF().show()
```

Output

```
Schema for the persons DynamicFrame:
root
|-- family_name: string
|-- name: string
|-- links: array
|    |-- element: struct
|    |    |-- note: string
|    |    |-- url: string
|-- gender: string
|-- image: string
|-- identifiers: array
|    |-- element: struct
|    |    |-- scheme: string
|    |    |-- identifier: string
|-- other_names: array
|    |-- element: struct
|    |    |-- lang: string
|    |    |-- note: string
|    |    |-- name: string
|-- sort_name: string
|-- images: array
|    |-- element: struct
|    |    |-- url: string
|-- given_name: string
|-- birth_date: string
|-- id: string
|-- contact_details: array
|    |-- element: struct
|    |    |-- type: string
|    |    |-- value: string
|-- death_date: string
```

```
Schema for the names DynamicFrame:
root
|-- family_name: string
|-- given_name: string

+-----------+----------+
|family_name|given_name|
+-----------+----------+
|    Collins|   Michael|
|   Huizenga|      Bill|
|    Clawson|    Curtis|
|    Solomon|    Gerald|
|     Rigell|    Edward|
|      Crapo|   Michael|
|      Hutto|      Earl|
|      Ertel|     Allen|
|     Minish|    Joseph|
|    Andrews|    Robert|
|     Walden|      Greg|
|      Kazen|   Abraham|
|     Turner|   Michael|
|      Kolbe|     James|
|  Lowenthal|      Alan|
|    Capuano|   Michael|
|   Schrader|      Kurt|
|     Nadler|   Jerrold|
|     Graves|       Tom|
|   McMillan|      John|
+-----------+----------+
only showing top 20 rows
```

simplify_ddb_json

`simplify_ddb_json(): DynamicFrame`

特に DynamoDB JSON 構造にある DynamicFrame でネスト化された列を単純化し、新しい単純化された DynamicFrame を返します。リストタイプに複数タイプまたは Map タイプがある場合、リストの要素は単純化されません。これは通常の unnest 変換とは異なって動作する変換の特定タイプであり、データが既に DynamoDB JSON 構造に格納されている必要があることに注意してください。詳細については、「DynamoDB JSON」を参照してください。

例えば、DynamoDB JSON 構造体のあるエクスポートを読み取るスキーマは次のようになります。

```
root
|-- Item: struct
|    |-- parentMap: struct
|    |    |-- M: struct
|    |    |    |-- childMap: struct
|    |    |    |    |-- M: struct
|    |    |    |    |    |-- appName: struct
|    |    |    |    |    |    |-- S: string
|    |    |    |    |    |-- packageName: struct
|    |    |    |    |    |    |-- S: string
|    |    |    |    |    |-- updatedAt: struct
|    |    |    |    |    |    |-- N: string
|    |-- strings: struct
|    |    |-- SS: array
|    |    |    |-- element: string
|    |-- numbers: struct
|    |    |-- NS: array
|    |    |    |-- element: string
|    |-- binaries: struct
|    |    |-- BS: array
|    |    |    |-- element: string
|    |-- isDDBJson: struct
|    |    |-- BOOL: boolean
|    |-- nullValue: struct
|    |    |-- NULL: boolean
```

`simplify_ddb_json()` 変換は、以下のように変換します。

```
root
|-- parentMap: struct
|    |-- childMap: struct
|    |    |-- appName: string
|    |    |-- packageName: string
|    |    |-- updatedAt: string
|-- strings: array
|    |-- element: string
|-- numbers: array
|    |-- element: string
|-- binaries: array
|    |-- element: string
|-- isDDBJson: boolean
```

```
|-- nullValue: null
```

例: simplify_ddb_json を使用して DynamoDB JSON の単純化を呼び出す

このコード例では simplify_ddb_json メソッドを使用し、AWS Glue DynamoDB エクスポートコネクタを使用し、DynamoDB JSON 単純化を呼び出してパーティションの数を表示します。

コードの例

```
from pyspark.context import SparkContext
from awsglue.context import GlueContext

sc = SparkContext()
glueContext = GlueContext(sc)

dynamicFrame = glueContext.create_dynamic_frame.from_options(
    connection_type = "dynamodb",
    connection_options = {
        'dynamodb.export': 'ddb',
        'dynamodb.tableArn': '<table arn>',
        'dynamodb.s3.bucket': '<bucket name>',
        'dynamodb.s3.prefix': '<bucket prefix>',
        'dynamodb.s3.bucketOwner': '<account_id of bucket>'
    }
)
simplified = dynamicFrame.simplify_ddb_json()
print(simplified.getNumPartitions())
```

スピゴット

spigot(path, options={})

ジョブで実行された変換が確認しやすくなるように、サンプルレコードを指定した送信先に書き込みます。

- path – 書き込み先へのパス (必須)。
- options – オプションを指定するキーと値のペア (オプション)。"topk" オプションは、最初の k レコードを書き込むことを指定します。"prob" オプションは、指定されたレコードを選択する確率(10 進数)を指定します。これを使用して、書き込むレコードを選択できます。
- transformation_ctx - 状態情報を識別するために使用される一意の文字列 (オプション)。

例: spigot を使用して **DynamicFrame** のサンプルフィールドを Amazon S3 に書き込む

このコード例では、spigot メソッドを使用して、select_fields 変換を適用した後に Amazon S3 バケットにサンプルレコードを書き込みます。

データセットの例

> **Note**
>
> この例で使用されているデータセットにアクセスするには、「コード例: データの結合と関係付け」を参照し、「ステップ 1: Amazon S3 バケット内のデータをクロールする」の手順に従います。

この例では、次のスキーマを持つ persons と呼ばれる DynamicFrame を使用しています。

```
root
|-- family_name: string
|-- name: string
|-- links: array
|    |-- element: struct
|    |    |-- note: string
|    |    |-- url: string
|-- gender: string
|-- image: string
|-- identifiers: array
|    |-- element: struct
|    |    |-- scheme: string
|    |    |-- identifier: string
|-- other_names: array
|    |-- element: struct
|    |    |-- lang: string
|    |    |-- note: string
|    |    |-- name: string
|-- sort_name: string
|-- images: array
|    |-- element: struct
|    |    |-- url: string
|-- given_name: string
|-- birth_date: string
|-- id: string
|-- contact_details: array
```

```
|    |-- element: struct
|    |    |-- type: string
|    |    |-- value: string
|-- death_date: string
```

コードの例

```
# Example: Use spigot to write sample records
# to a destination during a transformation
# from pyspark.context import SparkContext.
# Replace DOC-EXAMPLE-BUCKET with your own location.

from pyspark.context import SparkContext
from awsglue.context import GlueContext

# Create GlueContext
sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

# Load table data into a DynamicFrame
persons = glueContext.create_dynamic_frame.from_catalog(
    database="legislators", table_name="persons_json"
)

# Perform the select_fields on the DynamicFrame
persons = persons.select_fields(paths=["family_name", "given_name", "birth_date"])

# Use spigot to write a sample of the transformed data
# (the first 10 records)
spigot_output = persons.spigot(
    path="s3://DOC-EXAMPLE-BUCKET", options={"topk": 10}
)
# Example: Use spigot to write sample records
# to a destination during a transformation
# from pyspark.context import SparkContext.
# Replace DOC-EXAMPLE-BUCKET with your own location.

from pyspark.context import SparkContext
from awsglue.context import GlueContext

# Create GlueContext
sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)
```

```
# Load table data into a DynamicFrame
persons = glueContext.create_dynamic_frame.from_catalog(
    database="legislators", table_name="persons_json"
)

# Perform the select_fields on the DynamicFrame
persons = persons.select_fields(paths=["family_name", "given_name", "birth_date"])

# Use spigot to write a sample of the transformed data
# (the first 10 records)
spigot_output = persons.spigot(
    path="s3://DOC-EXAMPLE-BUCKET", options={"topk": 10}
)
```

Output

spigot で Amazon S3 に書き込むデータの例を次に示します。サンプルコードで options={"topk": 10} を指定したため、サンプルデータには最初の 10 個のレコードが含まれます。

```
{"family_name":"Collins","given_name":"Michael","birth_date":"1944-10-15"}
{"family_name":"Huizenga","given_name":"Bill","birth_date":"1969-01-31"}
{"family_name":"Clawson","given_name":"Curtis","birth_date":"1959-09-28"}
{"family_name":"Solomon","given_name":"Gerald","birth_date":"1930-08-14"}
{"family_name":"Rigell","given_name":"Edward","birth_date":"1960-05-28"}
{"family_name":"Crapo","given_name":"Michael","birth_date":"1951-05-20"}
{"family_name":"Hutto","given_name":"Earl","birth_date":"1926-05-12"}
{"family_name":"Ertel","given_name":"Allen","birth_date":"1937-11-07"}
{"family_name":"Minish","given_name":"Joseph","birth_date":"1916-09-01"}
{"family_name":"Andrews","given_name":"Robert","birth_date":"1957-08-04"}
```

split_fields

split_fields(paths, name1, name2, transformation_ctx="", info="", stageThreshold=0, totalThreshold=0)

2 つの DynamicFrames を含む新しい DynamicFrameCollection を返します。1 つ目の DynamicFrame には分割されたすべてのノードが含まれ、2 つ目には残りのノードが含まれています。

- paths - 文字列のリスト。各文字列は新しい DynamicFrame に分割するノードのフルパスです。
- name1 - 分割された DynamicFrame の名前文字列。
- name2 - 指定されたノードが分割された後に残る DynamicFrame の名前文字列。
- transformation_ctx - 状態情報を識別するために使用される一意の文字列 (オプション)。
- info - この変換のエラー報告に関連付ける文字列 (オプション)。
- stageThreshold - この変換中に発生した、プロセスがエラーを出力するエラーの数 (オプション)。デフォルトはゼロで、エラーを出力しないことを示します。
- totalThreshold - この変換までに発生したエラーのうち、プロセスがエラーを出力するエラーの数 (オプション)。デフォルトはゼロで、エラーを出力しないことを示します。

例: split_fields を使用して、選択したフィールドを別の **DynamicFrame** に分割する

このコード例では、split_fields メソッドを使用して、指定したフィールドのリストを別の DynamicFrame に分割します。

データセットの例

この例では、legislators_relationalized という名前のコレクションからの l_root_contact_details と呼ばれる DynamicFrame が使用されています。

l_root_contact_details には、次のスキーマとエントリが含まれています。

```
root
|-- id: long
|-- index: int
|-- contact_details.val.type: string
|-- contact_details.val.value: string

+---+-----+------------------------+-------------------------+
| id|index|contact_details.val.type|contact_details.val.value|
+---+-----+------------------------+-------------------------+
|  1|    0|                   phone|             202-225-5265|
|  1|    1|                 twitter|              kathyhochul|
|  2|    0|                   phone|             202-225-3252|
|  2|    1|                 twitter|            repjackyrosen|
|  3|    0|                     fax|             202-225-1314|
|  3|    1|                   phone|             202-225-3772|
...
```

コードの例

```
# Example: Use split_fields to split selected
# fields into a separate DynamicFrame

from pyspark.context import SparkContext
from awsglue.context import GlueContext

# Create GlueContext
sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

# Load the input DynamicFrame and inspect its schema
frame_to_split = legislators_relationalized.select("l_root_contact_details")
print("Inspect the input DynamicFrame schema:")
frame_to_split.printSchema()

# Split id and index fields into a separate DynamicFrame
split_fields_collection = frame_to_split.split_fields(["id", "index"], "left", "right")

# Inspect the resulting DynamicFrames
print("Inspect the schemas of the DynamicFrames created with split_fields:")
split_fields_collection.select("left").printSchema()
split_fields_collection.select("right").printSchema()
```

Output

```
Inspect the input DynamicFrame's schema:
root
|-- id: long
|-- index: int
|-- contact_details.val.type: string
|-- contact_details.val.value: string

Inspect the schemas of the DynamicFrames created with split_fields:
root
|-- id: long
|-- index: int

root
|-- contact_details.val.type: string
|-- contact_details.val.value: string
```

split_rows

split_rows(comparison_dict, name1, name2, transformation_ctx="", info="", stageThreshold=0, totalThreshold=0)

DynamicFrame の 1 つ以上の行を、新しい DynamicFrame に分割します。

このメソッドは、2 つの DynamicFrames を含む新しい DynamicFrameCollection を返します。1 つ目の DynamicFrame には分割されたすべての行が含まれ、2 つ目には残りの行が含まれています。

- comparison_dict – 列へのパスを示すためのキーと、列の値の比較対象である値に対しコンパレータをマッピングする別のディクショナリを示すための値を持つ、ディクショナリ。例えば、{"age": {">": 10, "<": 20}} は、age 列の値が 10 より大きく 20 より小さいすべての行を分割します。
- name1 - 分割された DynamicFrame の名前文字列。
- name2 - 指定されたノードが分割された後に残る DynamicFrame の名前文字列。
- transformation_ctx - 状態情報を識別するために使用される一意の文字列 (オプション)。
- info - この変換のエラー報告に関連付ける文字列 (オプション)。
- stageThreshold - この変換中に発生した、プロセスがエラーを出力するエラーの数 (オプション)。デフォルトはゼロで、エラーを出力しないことを示します。
- totalThreshold - この変換までに発生したエラーのうち、プロセスがエラーを出力するエラーの数 (オプション)。デフォルトはゼロで、エラーを出力しないことを示します。

例: split_rows を使用して、**DynamicFrame** 内の行を分割する

このコード例では、split_rows メソッドを使用して、id フィールド値に基づいて DynamicFrame の行を分割します。

データセットの例

この例では、legislators_relationalized という名前のコレクションから選択された l_root_contact_details と呼ばれる DynamicFrame が使用されています。

l_root_contact_details には、次のスキーマとエントリが含まれています。

```
root
|-- id: long
```

```
|-- index: int
|-- contact_details.val.type: string
|-- contact_details.val.value: string

+---+-----+-----------------------+-----------------------+
| id|index|contact_details.val.type|contact_details.val.value|
+---+-----+-----------------------+-----------------------+
|  1|    0|                  phone|             202-225-5265|
|  1|    1|                twitter|              kathyhochul|
|  2|    0|                  phone|             202-225-3252|
|  2|    1|                twitter|            repjackyrosen|
|  3|    0|                    fax|             202-225-1314|
|  3|    1|                  phone|             202-225-3772|
|  3|    2|                twitter|          MikeRossUpdates|
|  4|    0|                    fax|             202-225-1314|
|  4|    1|                  phone|             202-225-3772|
|  4|    2|                twitter|          MikeRossUpdates|
|  5|    0|                    fax|             202-225-1314|
|  5|    1|                  phone|             202-225-3772|
|  5|    2|                twitter|          MikeRossUpdates|
|  6|    0|                    fax|             202-225-1314|
|  6|    1|                  phone|             202-225-3772|
|  6|    2|                twitter|          MikeRossUpdates|
|  7|    0|                    fax|             202-225-1314|
|  7|    1|                  phone|             202-225-3772|
|  7|    2|                twitter|          MikeRossUpdates|
|  8|    0|                    fax|             202-225-1314|
+---+-----+-----------------------+-----------------------+
```

コードの例

```
# Example: Use split_rows to split up
# rows in a DynamicFrame based on value

from pyspark.context import SparkContext
from awsglue.context import GlueContext

# Create GlueContext
sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

# Retrieve the DynamicFrame to split
frame_to_split = legislators_relationalized.select("l_root_contact_details")
```

```
# Split up rows by ID
split_rows_collection = frame_to_split.split_rows({"id": {">": 10}}, "high", "low")

# Inspect the resulting DynamicFrames
print("Inspect the DynamicFrame that contains IDs < 10")
split_rows_collection.select("low").toDF().show()
print("Inspect the DynamicFrame that contains IDs > 10")
split_rows_collection.select("high").toDF().show()
```

Output

```
Inspect the DynamicFrame that contains IDs < 10
+---+-----+-----------------------+-----------------------+
| id|index|contact_details.val.type|contact_details.val.value|
+---+-----+-----------------------+-----------------------+
|  1|    0|                  phone|             202-225-5265|
|  1|    1|                twitter|              kathyhochul|
|  2|    0|                  phone|             202-225-3252|
|  2|    1|                twitter|            repjackyrosen|
|  3|    0|                    fax|             202-225-1314|
|  3|    1|                  phone|             202-225-3772|
|  3|    2|                twitter|          MikeRossUpdates|
|  4|    0|                    fax|             202-225-1314|
|  4|    1|                  phone|             202-225-3772|
|  4|    2|                twitter|          MikeRossUpdates|
|  5|    0|                    fax|             202-225-1314|
|  5|    1|                  phone|             202-225-3772|
|  5|    2|                twitter|          MikeRossUpdates|
|  6|    0|                    fax|             202-225-1314|
|  6|    1|                  phone|             202-225-3772|
|  6|    2|                twitter|          MikeRossUpdates|
|  7|    0|                    fax|             202-225-1314|
|  7|    1|                  phone|             202-225-3772|
|  7|    2|                twitter|          MikeRossUpdates|
|  8|    0|                    fax|             202-225-1314|
+---+-----+-----------------------+-----------------------+
only showing top 20 rows

Inspect the DynamicFrame that contains IDs > 10
+---+-----+-----------------------+-----------------------+
| id|index|contact_details.val.type|contact_details.val.value|
+---+-----+-----------------------+-----------------------+
```

```
| 11|    0|                   phone|             202-225-5476|
| 11|    1|                 twitter|            RepDavidYoung|
| 12|    0|                   phone|             202-225-4035|
| 12|    1|                 twitter|          RepStephMurphy|
| 13|    0|                     fax|             202-226-0774|
| 13|    1|                   phone|             202-225-6335|
| 14|    0|                     fax|             202-226-0774|
| 14|    1|                   phone|             202-225-6335|
| 15|    0|                     fax|             202-226-0774|
| 15|    1|                   phone|             202-225-6335|
| 16|    0|                     fax|             202-226-0774|
| 16|    1|                   phone|             202-225-6335|
| 17|    0|                     fax|             202-226-0774|
| 17|    1|                   phone|             202-225-6335|
| 18|    0|                     fax|             202-226-0774|
| 18|    1|                   phone|             202-225-6335|
| 19|    0|                     fax|             202-226-0774|
| 19|    1|                   phone|             202-225-6335|
| 20|    0|                     fax|             202-226-0774|
| 20|    1|                   phone|             202-225-6335|
+---+-----+------------------------+------------------------+
only showing top 20 rows
```

アンボックス

unbox(path, format, transformation_ctx="", info="", stageThreshold=0, totalThreshold=0, **options)

DynamicFrame の文字列フィールドをアンボックス (再フォーマット) し、アンボックスされた DynamicRecords を含む新しい DynamicFrame を返します。

DynamicRecord は DynamicFrame 内の論理レコードを表します。これは、自己記述型であり、固定スキーマに準拠しないデータに使用できる点を除いて、Apache Spark DataFrame の行に似ています。

- path - アンボックスする文字列ノードへのフルパス。
- format – 形式の仕様 (オプション)。Amazon S3 や、複数の形式をサポートする AWS Glue 接続の場合に使用します。サポートされる形式については、「AWS Glue for Spark での入出力のデータ形式に関するオプション」を参照してください。
- transformation_ctx - 状態情報を識別するために使用される一意の文字列 (オプション)。

- info - この変換のエラー報告に関連付ける文字列 (オプション)。
- stageThreshold - この変換中に発生した、プロセスがエラーを出力するエラーの数 (オプション)。デフォルトはゼロで、エラーを出力しないことを示します。
- totalThreshold - この変換までに発生したエラーのうち、プロセスがエラーを出力するエラーの数 (オプション)。デフォルトはゼロで、エラーを出力しないことを示します。
- options - 次の 1 つ以上。
  - separator – 区切り文字を含む文字列。
  - escaper – エスケープ文字を含む文字列。
  - skipFirst – 最初のインスタンスをスキップするかどうかを示すブール値。
  - withSchema — ノードのスキーマの JSON 表現を含む文字列。スキーマの JSON 表現の形式は、StructType.json( ) の出力によって定義されます。
  - withHeader – ヘッダーが含まれているかどうかを示すブール値。

例: unbox を使用して、文字列フィールドを構造体にアンボックスする

このコード例では、unbox メソッドを使用して、DynamicFrame の文字列フィールドを struct 型のフィールドにアンボックスまたは再フォーマットします。

データセットの例

この例では、次のスキーマとエントリを持つ mapped_with_string と呼ばれる DynamicFrame を使用しています。

AddressString という名前のフィールドに注目してください。この例では、このフィールドを構造体にアンボックスしています。

```
root
|-- Average Total Payments: string
|-- AddressString: string
|-- Average Covered Charges: string
|-- DRG Definition: string
|-- Average Medicare Payments: string
|-- Hospital Referral Region Description: string
|-- Address: struct
|    |-- Zip.Code: string
|    |-- City: string
|    |-- Array: array
|    |    |-- element: string
```

```
|    |-- State: string
|    |-- Street: string
|-- Provider Id: string
|-- Total Discharges: string
|-- Provider Name: string

+----------------------+--------------------+----------------------
+--------------------+------------------------+-----------------------------------
+--------------------+----------+---------------+--------------------+
|Average Total Payments|       AddressString|Average Covered Charges|      DRG
 Definition|Average Medicare Payments|Hospital Referral Region Description|
 Address|Provider Id|Total Discharges|       Provider Name|
+----------------------+--------------------+----------------------
+--------------------+------------------------+-----------------------------------
+--------------------+----------+---------------+--------------------+
|              $5777.24|{"Street": "1108 ...|              $32963.07|039 -
 EXTRACRANIA...|                 $4763.73|                         AL - Dothan|[36301,
 DOTHAN, [...|      10001|              91|SOUTHEAST ALABAMA...|
|              $5787.57|{"Street": "2505 ...|              $15131.85|039 -
 EXTRACRANIA...|                 $4976.71|                     AL - Birmingham|[35957,
 BOAZ, [25...|      10005|              14|MARSHALL MEDICAL ...|
|              $5434.95|{"Street": "205 M...|              $37560.37|039 -
 EXTRACRANIA...|                 $4453.79|                     AL - Birmingham|[35631,
 FLORENCE,...|      10006|              24|ELIZA COFFEE MEMO...|
|              $5417.56|{"Street": "50 ME...|              $13998.28|039 -
 EXTRACRANIA...|                 $4129.16|                     AL - Birmingham|[35235,
 BIRMINGHA...|      10011|              25|   ST VINCENT'S EAST|
...
```

コードの例

```
# Example: Use unbox to unbox a string field
# into a struct in a DynamicFrame

from pyspark.context import SparkContext
from awsglue.context import GlueContext

# Create GlueContext
sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

unboxed = mapped_with_string.unbox("AddressString", "json")
```

```
unboxed.printSchema()
unboxed.toDF().show()
```

Output

```
root
|-- Average Total Payments: string
|-- AddressString: struct
|    |-- Street: string
|    |-- City: string
|    |-- State: string
|    |-- Zip.Code: string
|    |-- Array: array
|    |    |-- element: string
|-- Average Covered Charges: string
|-- DRG Definition: string
|-- Average Medicare Payments: string
|-- Hospital Referral Region Description: string
|-- Address: struct
|    |-- Zip.Code: string
|    |-- City: string
|    |-- Array: array
|    |    |-- element: string
|    |-- State: string
|    |-- Street: string
|-- Provider Id: string
|-- Total Discharges: string
|-- Provider Name: string

+----------------------+--------------------+-----------------------
+--------------------+-------------------------+-----------------------------------
+--------------------+-----------+---------------+--------------------+
|Average Total Payments|       AddressString|Average Covered Charges|      DRG
 Definition|Average Medicare Payments|Hospital Referral Region Description|
 Address|Provider Id|Total Discharges|       Provider Name|
+----------------------+--------------------+-----------------------
+--------------------+-------------------------+-----------------------------------
+--------------------+-----------+---------------+--------------------+
|              $5777.24|[1108 ROSS CLARK ...|             $32963.07|039 -
 EXTRACRANIA...|                $4763.73|                        AL - Dothan|[36301,
 DOTHAN, [...|      10001|              91|SOUTHEAST ALABAMA...|
```

```
|              $5787.57|[2505 U S HIGHWAY...|              $15131.85|039 -
 EXTRACRANIA...|                $4976.71|                    AL - Birmingham|[35957,
 BOAZ, [25...|      10005|              14|MARSHALL MEDICAL ...|
|              $5434.95|[205 MARENGO STRE...|              $37560.37|039 -
 EXTRACRANIA...|                $4453.79|                    AL - Birmingham|[35631,
 FLORENCE,...|      10006|              24|ELIZA COFFEE MEMO...|
|              $5417.56|[50 MEDICAL PARK ...|              $13998.28|039 -
 EXTRACRANIA...|                $4129.16|                    AL - Birmingham|[35235,
 BIRMINGHA...|      10011|              25|   ST VINCENT'S EAST|
|              $5658.33|[1000 FIRST STREE...|              $31633.27|039 -
 EXTRACRANIA...|                $4851.44|                    AL - Birmingham|[35007,
 ALABASTER...|      10016|              18|SHELBY BAPTIST ME...|
|              $6653.80|[2105 EAST SOUTH ...|              $16920.79|039 -
 EXTRACRANIA...|                $5374.14|                    AL - Montgomery|[36116,
 MONTGOMER...|      10023|              67|BAPTIST MEDICAL C...|
|              $5834.74|[2000 PEPPERELL P...|              $11977.13|039 -
 EXTRACRANIA...|                $4761.41|                    AL - Birmingham|[36801,
 OPELIKA, ...|      10029|              51|EAST ALABAMA MEDI...|
|              $8031.12|[619 SOUTH 19TH S...|              $35841.09|039 -
 EXTRACRANIA...|                $5858.50|                    AL - Birmingham|[35233,
 BIRMINGHA...|      10033|              32|UNIVERSITY OF ALA...|
|              $6113.38|[101 SIVLEY RD, H...|              $28523.39|039 -
 EXTRACRANIA...|                $5228.40|                    AL - Huntsville|[35801,
 HUNTSVILL...|      10039|             135| HUNTSVILLE HOSPITAL|
|              $5541.05|[1007 GOODYEAR AV...|              $75233.38|039 -
 EXTRACRANIA...|                $4386.94|                    AL - Birmingham|[35903,
 GADSDEN, ...|      10040|              34|GADSDEN REGIONAL ...|
|              $5461.57|[600 SOUTH THIRD ...|              $67327.92|039 -
 EXTRACRANIA...|                $4493.57|                    AL - Birmingham|[35901,
 GADSDEN, ...|      10046|              14|RIVERVIEW REGIONA...|
|              $5356.28|[4370 WEST MAIN S...|              $39607.28|039 -
 EXTRACRANIA...|                $4408.20|                        AL - Dothan|[36305,
 DOTHAN, [...|      10055|              45|    FLOWERS HOSPITAL|
|              $5374.65|[810 ST VINCENT'S...|              $22862.23|039 -
 EXTRACRANIA...|                $4186.02|                    AL - Birmingham|[35205,
 BIRMINGHA...|      10056|              43|ST VINCENT'S BIRM...|
|              $5366.23|[400 EAST 10TH ST...|              $31110.85|039 -
 EXTRACRANIA...|                $4376.23|                    AL - Birmingham|[36207,
 ANNISTON,...|      10078|              21|NORTHEAST ALABAMA...|
|              $5282.93|[1613 NORTH MCKEN...|              $25411.33|039 -
 EXTRACRANIA...|                $4383.73|                        AL - Mobile|[36535,
 FOLEY, [1...|      10083|              15|SOUTH BALDWIN REG...|
```

```
|              $5676.55|[1201 7TH STREET ...|               $9234.51|039 -
 EXTRACRANIA...|                $4509.11|                   AL - Huntsville|[35609,
 DECATUR, ...|      10085|              27|DECATUR GENERAL H...|
|              $5930.11|[6801 AIRPORT BOU...|              $15895.85|039 -
 EXTRACRANIA...|                $3972.85|                       AL - Mobile|[36608,
 MOBILE, [...|      10090|              27| PROVIDENCE HOSPITAL|
|              $6192.54|[809 UNIVERSITY B...|              $19721.16|039 -
 EXTRACRANIA...|                $5179.38|                   AL - Tuscaloosa|[35401,
 TUSCALOOS...|      10092|              31|D C H REGIONAL ME...|
|              $4968.00|[750 MORPHY AVENU...|              $10710.88|039 -
 EXTRACRANIA...|                $3898.88|                       AL - Mobile|[36532,
 FAIRHOPE,...|      10100|              18|     THOMAS HOSPITAL|
|              $5996.00|[701 PRINCETON AV...|              $51343.75|039 -
 EXTRACRANIA...|                $4962.45|                   AL - Birmingham|[35211,
 BIRMINGHA...|      10103|              33|BAPTIST MEDICAL C...|
+----------------------+--------------------+----------------------
+--------------------+------------------------+-----------------------------------
+--------------------+----------+---------------+--------------------+
only showing top 20 rows
```

union

union(frame1, frame2, transformation_ctx = "", info = "", stageThreshold = 0, totalThreshold = 0)

2 つの DynamicFrames を結合します。両方の入力 DynamicFrame からのすべてのレコードを含む DynamicFrames を返します。この変換は、2 つの DataFrames rame を同等のデータで結合した結果とは異なる結果を返す場合があります。Spark DataFrame ユニオンの動作が必要な場合は、toDF の使用を検討してください。

- frame1 — 最初に結合する DynamicFrame。
- frame2 — 2 番目に結合する DynamicFrame。
- transformation_ctx – (オプション) 統計/ステータス情報を識別するために使用される一意の文字列。
- info – (オプション) 変換のエラーに関連付けられる文字列。
- stageThreshold — (オプション) 処理がエラーになるまでの変換中の最大エラー数。
- totalThreshold — (オプション) 処理がエラーになるまでの変換中の最大エラー数。

ネスト解除

unnest(transformation_ctx="", info="", stageThreshold=0, totalThreshold=0)

DynamicFrame 内のネストされたオブジェクトをネスト解除して、最上位レベルのオブジェクトにし、新しくネスト解除された DynamicFrame を返します。

- transformation_ctx - 状態情報を識別するために使用される一意の文字列 (オプション)。
- info - この変換のエラー報告に関連付ける文字列 (オプション)。
- stageThreshold - この変換中に発生した、プロセスがエラーを出力するエラーの数 (オプション)。デフォルトはゼロで、エラーを出力しないことを示します。
- totalThreshold - この変換までに発生したエラーのうち、プロセスがエラーを出力するエラーの数 (オプション)。デフォルトはゼロで、エラーを出力しないことを示します。

例: unnest を使用して、ネストされたフィールドを最上位フィールドに変換する

このコード例では、unnest メソッドを使用して、DynamicFrame のネストされたすべてのフィールドを最上位フィールドにフラット化します。

データセットの例

この例では、次のスキーマを持つ mapped_medicare と呼ばれる DynamicFrame を使用しています。Address フィールドは、ネストされたデータを含む唯一のフィールドであることに注意してください。

```
root
|-- Average Total Payments: string
|-- Average Covered Charges: string
|-- DRG Definition: string
|-- Average Medicare Payments: string
|-- Hospital Referral Region Description: string
|-- Address: struct
|    |-- Zip.Code: string
|    |-- City: string
|    |-- Array: array
|    |    |-- element: string
|    |-- State: string
|    |-- Street: string
|-- Provider Id: string
|-- Total Discharges: string
```

```
|-- Provider Name: string
```

コードの例

```
# Example: Use unnest to unnest nested
# objects in a DynamicFrame

from pyspark.context import SparkContext
from awsglue.context import GlueContext

# Create GlueContext
sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

# Unnest all nested fields
unnested = mapped_medicare.unnest()
unnested.printSchema()
```

Output

```
root
|-- Average Total Payments: string
|-- Average Covered Charges: string
|-- DRG Definition: string
|-- Average Medicare Payments: string
|-- Hospital Referral Region Description: string
|-- Address.Zip.Code: string
|-- Address.City: string
|-- Address.Array: array
|    |-- element: string
|-- Address.State: string
|-- Address.Street: string
|-- Provider Id: string
|-- Total Discharges: string
|-- Provider Name: string
```

unnest_ddb_json

特に DynamoDB JSON 構造体にある DynamicFrame でネストされた列をネスト解除すると、新しくネスト解除された DynamicFrame が返されます。構造体型の配列のある列は、ネスト解除されません。これは、通常の unnest 変換とは異なる特殊なタイプのネスト解除変換で、データが

DynamoDB JSON 構造体に格納されている必要があることに注意してください。詳細については、「DynamoDB JSON」を参照してください。

unnest_ddb_json(transformation_ctx="", info="", stageThreshold=0, totalThreshold=0)

- transformation_ctx - 状態情報を識別するために使用される一意の文字列 (オプション)。
- info - この変換のエラー報告に関連付ける文字列 (オプション)。
- stageThreshold - この変換中にプロセスで発生するエラーの数 (オプション: デフォルトではゼロ、プロセスがエラーを出力しないことを示します)。
- totalThreshold - この変換までに発生したエラーのうち、プロセスでエラーとなるエラーの数 (オプション: デフォルトではゼロ、プロセスがエラーを出力しないことを示します)。

例えば、DynamoDB JSON 構造体のあるエクスポートを読み取るスキーマは次のようになります。

```
root
|-- Item: struct
|    |-- ColA: struct
|    |    |-- S: string
|    |-- ColB: struct
|    |    |-- S: string
|    |-- ColC: struct
|    |    |-- N: string
|    |-- ColD: struct
|    |    |-- L: array
|    |    |    |-- element: null
```

unnest_ddb_json( ) 変換は、以下のように変換します。

```
root
|-- ColA: string
|-- ColB: string
|-- ColC: string
|-- ColD: array
|    |-- element: null
```

次のコード例は、AWS Glue DynamoDB エクスポートコネクタを使用し、DynamoDB JSON unnest を呼び出し、パーティションの数を表示します。

```
import sys
```

```
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job
from awsglue.utils import getResolvedOptions

args = getResolvedOptions(sys.argv, ["JOB_NAME"])
glue_context= GlueContext(SparkContext.getOrCreate())
job = Job(glue_context)
job.init(args["JOB_NAME"], args)

dynamicFrame = glue_context.create_dynamic_frame.from_options(
    connection_type="dynamodb",
    connection_options={
        "dynamodb.export": "ddb",
        "dynamodb.tableArn": "<test_source>",
        "dynamodb.s3.bucket": "<bucket name>",
        "dynamodb.s3.prefix": "<bucket prefix>",
        "dynamodb.s3.bucketOwner": "<account_id>",
    }
)
unnested = dynamicFrame.unnest_ddb_json()
print(unnested.getNumPartitions())

job.commit()
```

書き込み

write(connection_type, connection_options, format, format_options, accumulator_size)

この DynamicFrame の GlueContext クラス から指定された接続タイプの DataSink (オブジェクト) を取得し、この DynamicFrame のコンテンツの書式設定および書き込みに使用します。指定されたとおりに書式設定され、書き込まれる新しい DynamicFrame を返します。

- connection_type - 使用する接続タイプ。有効な値には、s3、mysql、postgresql、redshift、sqlserver、および oracle があります。
- connection_options - 使用する接続オプション (オプション)。s3 の connection_type では、Amazon S3 パスが定義されています。

```
connection_options = {"path": "s3://aws-glue-target/temp"}
```

JDBC 接続の場合、いくつかのプロパティを定義する必要があります。データベース名は URL の一部である必要があることに注意してください。オプションで接続オプションに含めることができます。

> **Warning**
>
> スクリプトにパスワードを保存することはお勧めしません。AWS Secrets Manager または AWS Glue データカタログから取得する場合には、boto3 を使用することを検討してください。

```
connection_options = {"url": "jdbc-url/database", "user": "username",
 "password": passwordVariable,"dbtable": "table-name", "redshiftTmpDir": "s3-tempdir-path"}
```

- format – 形式の仕様 (オプション)。これは、Amazon Simple Storage Service (Amazon S3)、または複数の形式をサポートする AWS Glue 接続で使用されます。サポートされる形式については、「AWS Glue for Spark での入出力のデータ形式に関するオプション」を参照してください。
- format_options – 指定された形式についてのオプション。サポートされる形式については、「AWS Glue for Spark での入出力のデータ形式に関するオプション」を参照してください。
- accumulator_size - 使用するバイト単位の累積サイズ (オプション)。

— errors —

- assertErrorThreshold
- errorsAsDynamicFrame
- errorsCount
- stageErrorsCount

assertErrorThreshold

assertErrorThreshold( ) - この DynamicFrame を作成した変換エラーに対するアサーション。基盤になる DataFrame から Exception を返します。

errorsAsDynamicFrame

errorsAsDynamicFrame( ) - 内部にネストされたエラーレコードを持つ DynamicFrame を返します。

例: errorsAsDynamicFrame を使用してエラーレコードを表示する

次のコード例は、errorsAsDynamicFrame メソッドを使用して DynamicFrame のエラーレコードを表示する方法を示しています。

データセットの例

この例では、JSON として Amazon S3 にアップロードできる次のデータセットを使用します。2 つ目のレコードの形式に誤りがあることに注意してください。通常、SparkSQL を使用すると、不正な形式のデータによってファイルの解析が中断されます。ただし、DynamicFrame は、不正な形式の問題を認識し、不正な行を個別に処理できるエラーレコードに変換します。

```
{"id": 1, "name": "george", "surname": "washington", "height": 178}
{"id": 2, "name": "benjamin", "surname": "franklin",
{"id": 3, "name": "alexander", "surname": "hamilton", "height": 171}
{"id": 4, "name": "john", "surname": "jay", "height": 190}
```

コードの例

```
# Example: Use errorsAsDynamicFrame to view error records.
# Replace s3://DOC-EXAMPLE-S3-BUCKET/error_data.json with your location.

from pyspark.context import SparkContext
from awsglue.context import GlueContext

# Create GlueContext
sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

# Create errors DynamicFrame, view schema
errors = glueContext.create_dynamic_frame.from_options(
    "s3", {"paths": ["s3://DOC-EXAMPLE-S3-BUCKET/error_data.json"]}, "json"
)
print("Schema of errors DynamicFrame:")
errors.printSchema()

# Show that errors only contains valid entries from the dataset
```

```
print("errors contains only valid records from the input dataset (2 of 4 records)")
errors.toDF().show()

# View errors
print("Errors count:", str(errors.errorsCount()))
print("Errors:")
errors.errorsAsDynamicFrame().toDF().show()

# View error fields and error data
error_record = errors.errorsAsDynamicFrame().toDF().head()

error_fields = error_record["error"]
print("Error fields: ")
print(error_fields.asDict().keys())

print("\nError record data:")
for key in error_fields.asDict().keys():
    print("\n", key, ": ", str(error_fields[key]))
```

Output

```
Schema of errors DynamicFrame:
root
|-- id: int
|-- name: string
|-- surname: string
|-- height: int

errors contains only valid records from the input dataset (2 of 4 records)
+---+------+----------+------+
| id|  name|   surname|height|
+---+------+----------+------+
|  1|george|washington|   178|
|  4|  john|       jay|   190|
+---+------+----------+------+

Errors count: 1
Errors:
+--------------------+
|               error|
+--------------------+
|[[  File "/tmp/20...|
+--------------------+
```

```
Error fields:
dict_keys(['callsite', 'msg', 'stackTrace', 'input', 'bytesread', 'source',
 'dynamicRecord'])

Error record data:

 callsite :  Row(site='  File "/tmp/2060612586885849088", line 549, in <module>\n
 sys.exit(main())\n  File "/tmp/2060612586885849088", line 523, in main\n    response
 = handler(content)\n  File "/tmp/2060612586885849088", line 197, in execute_request
\n    result = node.execute()\n  File "/tmp/2060612586885849088", line 103, in
 execute\n    exec(code, global_dict)\n  File "<stdin>", line 10, in <module>\n
 File "/opt/amazon/lib/python3.6/site-packages/awsglue/dynamicframe.py", line 625, in
 from_options\n    format_options, transformation_ctx, push_down_predicate, **kwargs)\n
  File "/opt/amazon/lib/python3.6/site-packages/awsglue/context.py", line 233, in
 create_dynamic_frame_from_options\n    source.setFormat(format, **format_options)\n',
 info='')

 msg :  error in jackson reader

 stackTrace :  com.fasterxml.jackson.core.JsonParseException: Unexpected character
 ('{' (code 123)): was expecting either valid name character (for unquoted name) or
 double-quote (for quoted) to start field name
 at [Source: com.amazonaws.services.glue.readers.BufferedStream@73492578; line: 3,
 column: 2]
 at com.fasterxml.jackson.core.JsonParser._constructError(JsonParser.java:1581)
 at
 com.fasterxml.jackson.core.base.ParserMinimalBase._reportError(ParserMinimalBase.java:533)
 at
 com.fasterxml.jackson.core.base.ParserMinimalBase._reportUnexpectedChar(ParserMinimalBase.java
 at
 com.fasterxml.jackson.core.json.UTF8StreamJsonParser._handleOddName(UTF8StreamJsonParser.java:
 at
 com.fasterxml.jackson.core.json.UTF8StreamJsonParser._parseName(UTF8StreamJsonParser.java:1650
 at
 com.fasterxml.jackson.core.json.UTF8StreamJsonParser.nextToken(UTF8StreamJsonParser.java:740)
 at com.amazonaws.services.glue.readers.JacksonReader$$anonfun$hasNextGoodToken
$1.apply(JacksonReader.scala:57)
 at com.amazonaws.services.glue.readers.JacksonReader$$anonfun$hasNextGoodToken
$1.apply(JacksonReader.scala:57)
 at scala.collection.Iterator$$anon$9.next(Iterator.scala:162)
 at scala.collection.Iterator$$anon$16.hasNext(Iterator.scala:599)
 at scala.collection.Iterator$$anon$16.hasNext(Iterator.scala:598)
 at scala.collection.Iterator$class.foreach(Iterator.scala:891)
```

```
 at scala.collection.AbstractIterator.foreach(Iterator.scala:1334)
 at com.amazonaws.services.glue.readers.JacksonReader$$anonfun
$1.apply(JacksonReader.scala:120)
 at com.amazonaws.services.glue.readers.JacksonReader$$anonfun
$1.apply(JacksonReader.scala:116)
 at
 com.amazonaws.services.glue.DynamicRecordBuilder.handleErr(DynamicRecordBuilder.scala:209)
 at
 com.amazonaws.services.glue.DynamicRecordBuilder.handleErrorWithException(DynamicRecordBuilder
 at
 com.amazonaws.services.glue.readers.JacksonReader.nextFailSafe(JacksonReader.scala:116)
 at com.amazonaws.services.glue.readers.JacksonReader.next(JacksonReader.scala:109)
 at com.amazonaws.services.glue.readers.JSONReader.next(JSONReader.scala:247)
 at
 com.amazonaws.services.glue.hadoop.TapeHadoopRecordReaderSplittable.nextKeyValue(TapeHadoopRec
 at org.apache.spark.rdd.NewHadoopRDD$$anon$1.hasNext(NewHadoopRDD.scala:230)
 at org.apache.spark.InterruptibleIterator.hasNext(InterruptibleIterator.scala:37)
 at scala.collection.Iterator$$anon$11.hasNext(Iterator.scala:409)
 at scala.collection.Iterator$$anon$11.hasNext(Iterator.scala:409)
 at scala.collection.Iterator$$anon$13.hasNext(Iterator.scala:462)
 at scala.collection.Iterator$$anon$11.hasNext(Iterator.scala:409)
 at scala.collection.Iterator$$anon$11.hasNext(Iterator.scala:409)
 at scala.collection.Iterator$$anon$13.hasNext(Iterator.scala:462)
 at scala.collection.Iterator$$anon$11.hasNext(Iterator.scala:409)
 at scala.collection.Iterator$$anon$11.hasNext(Iterator.scala:409)
 at org.apache.spark.sql.execution.SparkPlan$$anonfun$2.apply(SparkPlan.scala:255)
 at org.apache.spark.sql.execution.SparkPlan$$anonfun$2.apply(SparkPlan.scala:247)
 at org.apache.spark.rdd.RDD$$anonfun$mapPartitionsInternal$1$$anonfun$apply
$24.apply(RDD.scala:836)
 at org.apache.spark.rdd.RDD$$anonfun$mapPartitionsInternal$1$$anonfun$apply
$24.apply(RDD.scala:836)
 at org.apache.spark.rdd.MapPartitionsRDD.compute(MapPartitionsRDD.scala:52)
 at org.apache.spark.rdd.RDD.computeOrReadCheckpoint(RDD.scala:324)
 at org.apache.spark.rdd.RDD.iterator(RDD.scala:288)
 at org.apache.spark.rdd.MapPartitionsRDD.compute(MapPartitionsRDD.scala:52)
 at org.apache.spark.rdd.RDD.computeOrReadCheckpoint(RDD.scala:324)
 at org.apache.spark.rdd.RDD.iterator(RDD.scala:288)
 at org.apache.spark.scheduler.ResultTask.runTask(ResultTask.scala:90)
 at org.apache.spark.scheduler.Task.run(Task.scala:121)
 at org.apache.spark.executor.Executor$TaskRunner$$anonfun$10.apply(Executor.scala:408)
 at org.apache.spark.util.Utils$.tryWithSafeFinally(Utils.scala:1360)
 at org.apache.spark.executor.Executor$TaskRunner.run(Executor.scala:414)
 at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1149)
 at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624)
```

```
at java.lang.Thread.run(Thread.java:750)


input :

bytesread :  252

source :

dynamicRecord :  Row(id=2, name='benjamin', surname='franklin')
```

errorsCount

`errorsCount( )` - DynamicFrame 内のエラーの総数を返します。

stageErrorsCount

`stageErrorsCount` - この DynamicFrame を生成するプロセスで発生したエラーの数を返します。

DynamicFrameCollection クラス

`DynamicFrameCollection` は DynamicFrame クラス オブジェクトのディクショナリで、そのキーは DynamicFrames の名前、値は DynamicFrame オブジェクトです。

__init__

`__init__(dynamic_frames, glue_ctx)`

- `dynamic_frames` – DynamicFrame クラス オブジェクトのディクショナリ。
- `glue_ctx` – GlueContext クラス オブジェクト。

キー

`keys( )` – このコレクション内のキーのリストを返します。これは一般的に、対応する DynamicFrame 値の名前で構成されます。

[値]

`values(key)` – このコレクション内の DynamicFrame 値のリストを返します。

選択

select(key)

指定されたキー (一般に DynamicFrame の名前) に対応する DynamicFrame を返します。

- key – DynamicFrameCollection 内のキー。通常は DynamicFrame の名前を表します。

マッピング

map(callable, transformation_ctx="")

渡された関数を使用して、このコレクション内の DynamicFrameCollection に基づいた新しい DynamicFrames を作成して返します。

- callable – DynamicFrame と指定された変換コンテキストをパラメータとして取り、DynamicFrame を返す関数。
- transformation_ctx – 呼び出し可能なものによって使用される変換コンテキスト (省略可能)。

Flatmap

flatmap(f, transformation_ctx="")

渡された関数を使用して、このコレクション内の DynamicFrameCollection に基づいた新しい DynamicFrames を作成して返します。

- f – DynamicFrame をパラメータとして取り、DynamicFrame または DynamicFrameCollection を返す関数。
- transformation_ctx – 関数で使用される変換コンテキスト (省略可能)。

DynamicFrameWriter クラス

方法

- __init__
- from_options
- from_catalog

- from_jdbc_conf

__init__

__init__(glue_context)

- glue_context - 使用する GlueContext クラス。

from_options

from_options(frame, connection_type, connection_options={}, format=None, format_options={}, transformation_ctx="")

指定された接続と形式を使用して DynamicFrame を書き込みます。

- frame - 書き込む DynamicFrame。
- connection_type - 接続タイプ。有効な値には、s3、mysql、postgresql、redshift、sqlserver、および oracle があります。
- connection_options - 接続オプション (パスやデータベーステーブルなど) (オプション)。s3 の connection_type では、Amazon S3 パスが定義されています。

```
connection_options = {"path": "s3://aws-glue-target/temp"}
```

JDBC 接続の場合、いくつかのプロパティを定義する必要があります。データベース名は URL の一部である必要があることに注意してください。オプションで接続オプションに含めることができます。

> **Warning**
> スクリプトにパスワードを保存することはお勧めしません。AWS Secrets Manager または AWS Glue データカタログから取得する場合には、boto3 を使用することを検討してください。

```
connection_options = {"url": "jdbc-url/database", "user": "username",
 "password": passwordVariable,"dbtable": "table-name", "redshiftTmpDir": "s3-tempdir-path"}
```

dbtable プロパティは JDBC テーブルの名前です。データベース内でスキーマをサポートする JDBC データストアの場合、schema.table-name を指定します。スキーマを指定しない場合、デフォルトの「パブリック」スキーマが使用されます。

詳しくは、「AWS Glue for Spark での ETL の接続タイプとオプション」を参照してください。

- format – 形式の仕様 (オプション)。これは、Amazon Simple Storage Service (Amazon S3)、または複数の形式をサポートする AWS Glue 接続で使用されます。サポートされる形式については、「AWS Glue for Spark での入出力のデータ形式に関するオプション」を参照してください。
- format_options – 指定された形式についてのオプション。サポートされる形式については、「AWS Glue for Spark での入出力のデータ形式に関するオプション」を参照してください。
- transformation_ctx - 使用する変換コンテキスト (オプション)。

from_catalog

from_catalog(frame, name_space, table_name, redshift_tmp_dir="", transformation_ctx="")

指定されたカタログデータベースとテーブル名を使用して DynamicFrame を書き込みます。

- frame - 書き込む DynamicFrame。
- name_space – 使用するデータベース。
- table_name - 使用する table_name。
- redshift_tmp_dir - 使用する Amazon Redshift の一時ディレクトリ (オプション)。
- transformation_ctx - 使用する変換コンテキスト (オプション)。
- additional_options – AWS Glue で使用する追加のオプション。

Lake Formation governed table に書き込むには、以下の追加オプションを使用することができます。

- transactionId — (文字列) governed table への書き込みを行うトランザクション ID。このトランザクションをコミットまたは中断することはできません。書き込みが失敗します。
- callDeleteObjectsOnCancel - (プール値、オプション) true (デフォルト) に設定すると、オブジェクトが Amazon S3 に書き込まれた後、AWS Glue で自動的に DeleteObjectsOnCancel APIを呼び出します。詳細については、AWS Lake Formation Developer Guide の「DeleteObjectsOnCancel」を参照してください。

Example 例: Lake Formation の governed table への書き込み

```
txId = glueContext.start_transaction(read_only=False)
glueContext.write_dynamic_frame.from_catalog(
    frame=dyf,
    database = db,
    table_name = tbl,
    transformation_ctx = "datasource0",
    additional_options={"transactionId":txId})
...
glueContext.commit_transaction(txId)
```

from_jdbc_conf

from_jdbc_conf(frame, catalog_connection, connection_options={}, redshift_tmp_dir = "", transformation_ctx="")

指定された JDBC 接続情報を使用して DynamicFrame を書き込みます。

- frame - 書き込む DynamicFrame。
- catalog_connection - 使用するカタログ接続。
- connection_options - 接続オプション (パスやデータベーステーブルなど) (オプション)。
- redshift_tmp_dir - 使用する Amazon Redshift の一時ディレクトリ (オプション)。
- transformation_ctx - 使用する変換コンテキスト (オプション)。

write_dynamic_frame の例

この例では、POSIX パス引数を持つ S3 のconnection_type を connection_options で使用して、ローカルストレージへの書き込みを許可し、出力をローカルに書き込みます。

```
glueContext.write_dynamic_frame.from_options(\
frame = dyf_splitFields,\
connection_options = {'path': '/home/glue/GlueLocalOutput/'},\
connection_type = 's3',\
format = 'json')
```

DynamicFrameReader クラス

— 方法 —

- __init__
- from_rdd
- from_options
- from_catalog

__init__

__init__(glue_context)

- glue_context - 使用する GlueContext クラス。

from_rdd

from_rdd(data, name, schema=None, sampleRatio=None)

Resilient Distributed Dataset (RDD) から DynamicFrame を読み取ります。

- data - 読み取り元のデータセット。
- name - 読み取り元の名前。
- schema - 読み取るスキーマ (オプション)。
- sampleRatio - サンプル比率 (オプション)。

from_options

from_options(connection_type, connection_options={}, format=None, format_options={}, transformation_ctx="")

指定された接続と形式を使用して DynamicFrame を読み込みます。

- connection_type - 接続タイプ。有効な値には、s3、mysql、postgresql、redshift、sqlserver、oracle、dynamodb、snowflake が含まれています。

- connection_options - 接続オプション (パスやデータベーステーブルなど) (オプション)。詳細については、「AWS Glue for Spark で ETL の接続タイプとオプション」を参照してください。s3 の connection_type の場合、Amazon S3 のパスは配列で定義されます。

```
connection_options = {"paths": [ "s3://amzn-s3-demo-bucket/object_a", "s3://amzn-s3-demo-bucket/object_b"]}
```

JDBC 接続の場合、いくつかのプロパティを定義する必要があります。データベース名は URL の一部である必要があることに注意してください。オプションで接続オプションに含めることができます。

> **Warning**
>
> スクリプトにパスワードを保存することはお勧めしません。AWS Secrets Manager または AWS Glue データカタログから取得する場合には、boto3 を使用することを検討してください。

```
connection_options = {"url": "jdbc-url/database", "user": "username",
 "password": passwordVariable,"dbtable": "table-name", "redshiftTmpDir": "s3-tempdir-path"}
```

並列読み込みを実行する JDBC 接続の場合、ハッシュフィールドオプションを設定できます。例:

```
connection_options = {"url": "jdbc-url/database", "user": "username",
 "password": passwordVariable,"dbtable": "table-name", "redshiftTmpDir": "s3-tempdir-path" , "hashfield": "month"}
```

詳しくは、「JDBC テーブルからの並列読み取り」を参照してください。

- format – 形式の仕様 (オプション)。これは、Amazon Simple Storage Service (Amazon S3)、または複数の形式をサポートする AWS Glue 接続で使用されます。サポートされる形式については、「AWS Glue for Spark での入出力のデータ形式に関するオプション」を参照してください。
- format_options – 指定された形式についてのオプション。サポートされる形式については、「AWS Glue for Spark での入出力のデータ形式に関するオプション」を参照してください。
- transformation_ctx - 使用する変換コンテキスト (オプション)。

- push_down_predicate – データセットのすべてのファイルをリストアップして読み取る必要がないフィルタパーティション。詳細については、「Pre-Filtering Using Pushdown Predicates」を参照してください。

from_catalog

from_catalog(database, table_name, redshift_tmp_dir="", transformation_ctx="", push_down_predicate="", additional_options={})

指定されたカタログの名前空間とテーブル名を使用して DynamicFrame を読み取ります。

- database - 読み取り元のデータベース。
- table_name - 読み取り元のテーブルの名前。
- redshift_tmp_dir – 使用する Amazon Redshift の一時ディレクトリ (Redshift からデータを読み取らない場合はオプション)。
- transformation_ctx - 使用する変換コンテキスト (オプション)。
- push_down_predicate – データセットのすべてのファイルをリストアップして読み取る必要がないフィルタパーティション。詳しくは、「プッシュダウン述語を使用した事前フィルタ処理」を参照してください。
- additional_options – AWS Glue で使用する追加のオプション。
  - 並列読み込みを実行する JDBC 接続を使用するには、hashfield、hashexpression、または hashpartitions オプションを設定できます。例:

    ```
    additional_options = {"hashfield": "month"}
    ```

    詳しくは、「JDBC テーブルからの並列読み取り」を参照してください。
  - インデックス列に基づいたフィルタリングのためにカタログ式を渡すには、catalogPartitionPredicate オプションを使用できます。

    catalogPartitionPredicate – カタログ式を渡して、インデックス列に基づいたフィルタリングができます。これにより、フィルタリングをサーバー側で処理できます。詳細については、「AWS Glue パーティションインデックス」を参照してください。push_down_predicate と catalogPartitionPredicate では、異なる構文が使用されることに注意してください。前者では Spark SQL の標準構文を使用し、後者では JSQL パーサーを使用します。

    詳細については、「AWS Glue での ETL 出力のパーティションの管理」を参照してください。

GlueContext クラス

Apache Spark の SparkContext オブジェクトをラップすることにより、Apache Spark プラットフォームとやり取りするためのメカニズムが提供されます。

__init__

__init__(sparkContext)

- sparkContext - 使用する Apache Spark のコンテキスト。

作成中

- __init__
- getSource
- create_dynamic_frame_from_rdd
- create_dynamic_frame_from_catalog
- create_dynamic_frame_from_options
- create_sample_dynamic_frame_from_catalog
- create_sample_dynamic_frame_from_options
- add_ingestion_time_columns
- create_data_frame_from_catalog
- create_data_frame_from_options
- forEachBatch

getSource

getSource(connection_type, transformation_ctx = "", **options)

外部ソースから DynamicFrames を読み取るために使用できる DataSource オブジェクトを作成します。

- connection_type – 使用する接続タイプ (Amazon Simple Storage Service (Amazon S3)、Amazon Redshift、JDBC など)。有効な値には、s3、mysql、postgresql、redshift、sqlserver、oracle および dynamodb があります。

- transformation_ctx - 使用する変換コンテキスト (オプション)。
- options - オプションの名前と値のペアのコレクション。詳しくは、「AWS Glue for Spark での ETL の接続タイプとオプション」を参照してください。

以下は、getSource の使用例です。

```
>>> data_source = context.getSource("file", paths=["/in/path"])
>>> data_source.setFormat("json")
>>> myFrame = data_source.getFrame()
```

create_dynamic_frame_from_rdd

create_dynamic_frame_from_rdd(data, name, schema=None, sample_ratio=None, transformation_ctx="")

Apache Spark Resilient Distributed Dataset (RDD) から作成された DynamicFrame を返します。

- data - 使用するデータソース。
- name - 使用するデータの名前。
- schema - 使用するスキーマ (オプション)。
- sample_ratio - 使用するサンプル比率 (オプション)。
- transformation_ctx - 使用する変換コンテキスト (オプション)。

create_dynamic_frame_from_catalog

create_dynamic_frame_from_catalog(database, table_name, redshift_tmp_dir, transformation_ctx = "", push_down_predicate= "", additional_options = {}, catalog_id = None)

データカタログデータベースとテーブル名を使用して作成された DynamicFrame を返します。このメソッドを使用するときは、指定した AWS Glue データカタログテーブルのテーブルプロパティを使用して format_options を指定し、additional_options 引数を使用して他のオプションを指定します。

- Database - 読み取り元のデータベース。
- table_name - 読み取り元のテーブルの名前。
- redshift_tmp_dir - 使用する Amazon Redshift の一時ディレクトリ (オプション)。

- transformation_ctx - 使用する変換コンテキスト (オプション)。
- push_down_predicate – データセットのすべてのファイルをリストアップして読み取る必要がないフィルタパーティション。サポートされているソースと制限については、「AWS Glue ETL の「プッシュダウンによる読み取りの最適化」を参照してください。詳細については、「プッシュダウン述語を使用した事前フィルタ処理」を参照してください。
- additional_options - オプションの名前と値のペアのコレクション。AWS Glue for Spark での ETL の接続タイプとオプション でリストされている使用可能なオプション (endpointUrl、streamName、bootstrap.servers、security.protocol、topicName、classi および delimiter を除く)。別のオプションとして、catalogPartitionPredicate もサポートされています。

  catalogPartitionPredicate – カタログ式を渡して、インデックス列に基づいたフィルタリングができます。これにより、フィルタリングをサーバー側で処理できます。詳細については、「AWS Glue パーティションインデックス」を参照してください。push_down_predicate と catalogPartitionPredicate では、異なる構文が使用されることに注意してください。前者では Spark SQL の標準構文を使用し、後者では JSQL パーサーを使用します。
- catalog_id – 現在アクセスされているデータカタログのカタログ ID (アカウント ID)。None の場合は、呼び出し元のアカウント ID のデフォルトが使用されます。

create_dynamic_frame_from_options

create_dynamic_frame_from_options(connection_type, connection_options={}, format=None, format_options={}, transformation_ctx = "")

指定された接続と形式で作成された DynamicFrame を返します。

- connection_type – 接続タイプ (Amazon S3、Amazon Redshift、JDBC など)。有効な値には、s3、mysql、postgresql、redshift、sqlserver、oracle および dynamodb があります。
- connection_options – パスやデータベーステーブルなど接続オプション (オプション)。s3 の connection_type に関しては、Amazon S3 パスのリストが定義されています。

  ```
  connection_options = {"paths": ["s3://aws-glue-target/temp"]}
  ```

  JDBC 接続の場合、いくつかのプロパティを定義する必要があります。データベース名は URL の一部である必要があることに注意してください。オプションで接続オプションに含めることができます。

> **Warning**
>
> スクリプトにパスワードを保存することはお勧めしません。AWS Secrets Manager または AWS Glue データカタログから取得する場合には、boto3 を使用することを検討してください。

```
connection_options = {"url": "jdbc-url/database", "user": "username",
 "password": passwordVariable,"dbtable": "table-name", "redshiftTmpDir": "s3-tempdir-path"}
```

dbtable プロパティは JDBC テーブルの名前です。データベース内でスキーマをサポートする JDBC データストアの場合、schema.table-name を指定します。スキーマを指定しない場合、デフォルトの「パブリック」スキーマが使用されます。

詳細については、「AWS Glue for Spark での ETL の接続タイプとオプション」を参照してください。

- format – フォーマット仕様 Amazon S3 や、複数のフォーマットをサポートする AWS Glue 接続の場合に使用します。サポートされる形式については、「AWS Glue for Spark での入出力のデータ形式に関するオプション」を参照してください。
- format_options – 指定された形式についてのオプション。サポートされる形式については、「AWS Glue for Spark での入出力のデータ形式に関するオプション」を参照してください。
- transformation_ctx - 使用する変換コンテキスト (オプション)。
- push_down_predicate – データセットのすべてのファイルをリストアップして読み取る必要がないフィルタパーティション。サポートされているソースと制限については、「AWS Glue ETL の「プッシュダウンによる読み取りの最適化」を参照してください。詳細については、「プッシュダウン述語を使用した事前フィルタ処理」を参照してください。

create_sample_dynamic_frame_from_catalog

create_sample_dynamic_frame_from_catalog(database, table_name, num, redshift_tmp_dir, transformation_ctx = "", push_down_predicate= "", additional_options = {}, sample_options = {}, catalog_id = None)

データカタログデータベースとテーブル名を使用して作成されたサンプル DynamicFrame を返します。DynamicFrame にはデータソースからの最初の num レコードのみが含まれます。

- database - 読み取り元のデータベース。
- table_name - 読み取り元のテーブルの名前。
- num — 返されるサンプル動的フレーム内のレコードの最大数。
- redshift_tmp_dir - 使用する Amazon Redshift の一時ディレクトリ (オプション)。
- transformation_ctx - 使用する変換コンテキスト (オプション)。
- push_down_predicate – データセットのすべてのファイルをリストアップして読み取る必要がないフィルタパーティション。詳しくは、「プッシュダウン述語を使用した事前フィルタ処理」を参照してください。
- additional_options - オプションの名前と値のペアのコレクション。AWS Glue for Spark での ETL の接続タイプとオプション でリストされている使用可能なオプション (endpointUrl、streamName、bootstrap.servers、security.protocol、topicName、classi および delimiter を除く)。
- sample_options— サンプリング動作を制御するパラメータ (オプション)。Simple Storage Service (Amazon S3) ソースで現在使用可能なパラメータ:
  - maxSamplePartitions — サンプリングが読み取るパーティションの最大数。デフォルト値は 10 です
  - maxSampleFilesPerPartition — サンプリングが 1 つのパーティションで読み取るファイルの最大数。デフォルト値は 10 です。

    これらのパラメータは、ファイル一覧で消費される時間を短縮するのに役立ちます。例えば、データセットに 1000 個のパーティションがあり、各パーティションには 10 個のファイルがあるとします。10,000 個のファイルをすべてリスト表示する代わりに、maxSamplePartitions = 10 および maxSampleFilesPerPartition = 10 と設定した場合、サンプリングでは、最初の 10 個のパーティションの最初の 10 個のファイルのみがリスト表示されて読み込まれ、合計で 10*10 = 100 個のファイルとなります。
- catalog_id – 現在アクセスされているデータカタログのカタログ ID (データカタログのアカウント ID)。デフォルトでは、None に設定されています。None のデフォルト値は、サービス内の呼び出し元アカウントのカタログ ID になります。

create_sample_dynamic_frame_from_options

create_sample_dynamic_frame_from_options(connection_type, connection_options={}, num, sample_options={}, format=None, format_options={}, transformation_ctx = "")

指定された接続と形式で作成されたサンプル DynamicFrame を返します。DynamicFrame にはデータソースからの最初の num レコードのみが含まれます。

- connection_type – 接続タイプ (Amazon S3、Amazon Redshift、JDBC など)。有効な値には、s3、mysql、postgresql、redshift、sqlserver、oracle および dynamodb があります。
- connection_options – パスやデータベーステーブルなど接続オプション (オプション)。詳しくは、「AWS Glue for Spark での ETL の接続タイプとオプション」を参照してください。
- num — 返されるサンプル動的フレーム内のレコードの最大数。
- sample_options— サンプリング動作を制御するパラメータ (オプション)。Simple Storage Service (Amazon S3) ソースで現在使用可能なパラメータ:
  - maxSamplePartitions — サンプリングが読み取るパーティションの最大数。デフォルト値は 10 です
  - maxSampleFilesPerPartition — サンプリングが 1 つのパーティションで読み取るファイルの最大数。デフォルト値は 10 です。

    これらのパラメータは、ファイル一覧で消費される時間を短縮するのに役立ちます。例えば、データセットに 1000 個のパーティションがあり、各パーティションには 10 個のファイルがあるとします。10,000 個のファイルをすべてリスト表示する代わりに、maxSamplePartitions = 10 および maxSampleFilesPerPartition = 10 と設定した場合、サンプリングでは、最初の 10 個のパーティションの最初の 10 個のファイルのみがリスト表示されて読み込まれ、合計で 10*10 = 100 個のファイルとなります。
- format – フォーマット仕様 Amazon S3 や、複数のフォーマットをサポートする AWS Glue 接続の場合に使用します。サポートされる形式については、「AWS Glue for Spark での入出力のデータ形式に関するオプション」を参照してください。
- format_options – 指定された形式についてのオプション。サポートされる形式については、「AWS Glue for Spark での入出力のデータ形式に関するオプション」を参照してください。
- transformation_ctx - 使用する変換コンテキスト (オプション)。

- push_down_predicate – データセットのすべてのファイルをリストアップして読み取る必要がないフィルタパーティション。詳しくは、「プッシュダウン述語を使用した事前フィルタ処理」を参照してください。

add_ingestion_time_columns

add_ingestion_time_columns(dataFrame, timeGranularity = "")

入力 DataFrame への取り込み時間列 (ingest_year、ingest_month、ingest_day、ingest_hour、ingest_minute など) を追加します。Amazon S3 のデータカタログテーブルをターゲットとして指定する場合、この関数は、AWS Glue により生成されたスクリプト内で自動的に生成されます。この関数は、出力テーブル上で、取り込み時間列があるパーティションを自動的に更新します。これにより、入力データに明示的な取り込み時間列を指定しなくても、取り込み時間において出力データの自動的なパーティション化が行えます。

- dataFrame – 取り込み時間列の追加先である dataFrame。
- timeGranularity – 時間列の詳細度。有効な値は「day」、「hour」、および「minute」です。例えば、関数に対し「hour」が渡された場合、元の dataFrame は「ingest_year」、「ingest_month」、「ingest_day」に加え「ingest_hour」の時間列を持つことになります。

時間の詳細度列を追加した後、そのデータフレームを返します。

例:

```
dynamic_frame = DynamicFrame.fromDF(glueContext.add_ingestion_time_columns(dataFrame,
 "hour"))
```

create_data_frame_from_catalog

create_data_frame_from_catalog(database, table_name, transformation_ctx = "", additional_options = {})

データカタログテーブルからの情報を使用して作成された DataFrame を返します。

- database – 読み取り元のデータカタログデータベース。
- table_name – 読み取り元のデータカタログテーブルの名前。

- `transformation_ctx` - 使用する変換コンテキスト (オプション)。
- `additional_options` - オプションの名前と値のペアのコレクション。可能なオプションには、`startingPosition`、`maxFetchTimeInMs`、および `startingOffsets` など、ストリーミングソース用として AWS Glue for Spark での ETL の接続タイプとオプション にリストされているものが含まれます。
  - `useSparkDataSource` – true に設定すると、AWS Glue はネイティブ Spark データソース API を使用してテーブルを読み取るようになります。Spark データソース API でサポートされている形式は、AVRO、バイナリ、CSV、JSON、ORC、Parquet、およびテキストです。データカタログテーブルでは、`classification` プロパティを使用して形式を指定します。Spark データソース API の詳細については、公式の「Apache Spark ドキュメント」を参照してください。

    `create_data_frame_from_catalog` を `useSparkDataSource` と併用すると次のようなメリットがあります。
    - `DataFrame` を直接返し、`create_dynamic_frame.from_catalog().toDF()` の代替手段になります。
    - ネイティブ形式の AWS Lake Formation テーブルレベルのアクセス許可制御をサポートします。
    - AWS Lake Formation テーブルレベルのアクセス許可制御なしでデータレイク形式の読み取りをサポートします。詳細については、「AWS Glue ETL ジョブでのデータレイクフレームワークの使用」を参照してください。

    `useSparkDataSource` を有効にすると、必要に応じて任意の Spark データソースオプションを `additional_options` に追加することもできます。AWSGlue はこれらのオプションを Spark リーダーに直接渡します。
  - `useCatalogSchema` – true に設定すると、AWS Glue は結果として得られた `DataFrame` にデータカタログスキーマを適用します。true に設定しなければ、リーダーはデータからスキーマを推測します。`useCatalogSchema` を有効にする場合は、`useSparkDataSource` も true に設定する必要もあります。

制約事項

`useSparkDataSource` オプションを使用する際には、次の制限事項を考慮してください。

- `useSparkDataSource` を使用すると、AWS Glue は元の Spark セッションとは異なる別の Spark セッションで新しい `DataFrame` を作成します。
- Spark DataFrame パーティションフィルタリングは、次の AWS Glue 機能では機能しません。

- ジョブのブックマーク
- Amazon S3 ストレージクラスの除外
- カタログパーティション述語

これらの機能でパーティションフィルタリングを使用するには、AWS Glue プッシュダウン述語を使用します。詳細については、「プッシュダウン述語を使用した事前フィルタ処理」を参照してください。分割されていない列のフィルタリングは影響を受けません。

次のスクリプト例は、excludeStorageClasses オプションを使用してパーティションフィルタリングを実行する誤った方法を示しています。

```
// Incorrect partition filtering using Spark filter with excludeStorageClasses
read_df = glueContext.create_data_frame.from_catalog(
    database=database_name,
    table_name=table_name,
    additional_options = {
      "useSparkDataSource": True,
      "excludeStorageClasses" : ["GLACIER", "DEEP_ARCHIVE"]
    }
)

//  Suppose year and month are partition keys.
//  Filtering on year and month won't work, the filtered_df will still
//  contain data with other year/month values.
filtered_df = read_df.filter("year == '2017 and month == '04' and 'state == 'CA'")
```

次のスクリプト例は、excludeStorageClasses オプションを使用してパーティションフィルタリングを実行するためにプッシュダウン述語を使用する正しい方法を示しています。

```
// Correct partition filtering using the AWS Glue pushdown predicate
// with excludeStorageClasses
read_df = glueContext.create_data_frame.from_catalog(
    database=database_name,
    table_name=table_name,
    //  Use AWS Glue pushdown predicate to perform partition filtering
    push_down_predicate = "(year=='2017' and month=='04')"
    additional_options = {
      "useSparkDataSource": True,
      "excludeStorageClasses" : ["GLACIER", "DEEP_ARCHIVE"]
    }
)
```

```
//  Use Spark filter only on non-partitioned columns
filtered_df = read_df.filter("state == 'CA'")
```

例: Spark データソースリーダーを使用した CSV テーブルの作成

```
//  Read a CSV table with '\t' as separator
read_df = glueContext.create_data_frame.from_catalog(
    database=<database_name>,
    table_name=<table_name>,
    additional_options = {"useSparkDataSource": True,  "sep": '\t'}
)
```

create_data_frame_from_options

create_data_frame_from_options(connection_type, connection_options={}, format=None, format_options={}, transformation_ctx = "")

この API は廃止されました。代わりに、getSource() API を使用してください。指定された接続と形式で作成された DataFrame を返します。この関数は、AWS Glue ストリーミングソースのみで使用してください。

- connection_type – ストリーミング接続タイプ。有効な値は、kinesis および kafka です。
- connection_options – 接続オプション。Kinesis と Kafka では異なります。各ストリーミングデータソースのすべての接続オプションの一覧は、AWS Glue for Spark での ETL の接続タイプとオプション で確認いただけます。ストリーミング接続オプションについては、以下の違いに注意してください。
  - Kinesis ストリーミングのソースには streamARN、startingPosition、inferSchema、および classification が必要です。
  - Kafka ストリーミングのソースには connectionName、topicName、startingOffsets、inferSchema、および classification が必要です。
- format – フォーマット仕様 Amazon S3 や、複数のフォーマットをサポートする AWS Glue 接続の場合に使用します。サポートされる形式については、「AWS Glue for Spark での入出力のデータ形式に関するオプション」を参照してください。

- format_options – 指定された形式についてのオプション。サポートされる形式オプションについては、「AWS Glue for Spark での入出力のデータ形式に関するオプション」を参照してください。
- transformation_ctx - 使用する変換コンテキスト (オプション)。

Amazon Kinesis ストリーミングソースの例:

```
kinesis_options =
   { "streamARN": "arn:aws:kinesis:us-east-2:777788889999:stream/fromOptionsStream",
     "startingPosition": "TRIM_HORIZON",
     "inferSchema": "true",
     "classification": "json"
   }
data_frame_datasource0 =
 glueContext.create_data_frame.from_options(connection_type="kinesis",
 connection_options=kinesis_options)
```

Kafka ストリーミングソースの例 :

```
kafka_options =
    { "connectionName": "ConfluentKafka",
      "topicName": "kafka-auth-topic",
      "startingOffsets": "earliest",
      "inferSchema": "true",
      "classification": "json"
    }
data_frame_datasource0 =
 glueContext.create_data_frame.from_options(connection_type="kafka",
 connection_options=kafka_options)
```

forEachBatch

forEachBatch(frame, batch_function, options)

ストリーミングソースから読み取られるすべてのマイクロバッチに渡される、batch_function を適用します。

- frame – 現在のマイクロバッチを含む DataFrame。
- batch_function – すべてのマイクロバッチに適用される関数。

- options – マイクロバッチの処理方法に関する情報を保持している、キーと値のペアの集合。以下のような必須オプションがあります。
  - windowSize – 各バッチの処理にかかる時間。
  - checkpointLocation – ストリーミング ETL ジョブ用に、チェックポイントが格納される場所。
  - batchMaxRetries – 失敗した場合にこのバッチを再試行する最大回数。デフォルト値は 3 です。このオプションは、Glue バージョン 2.0 以降でのみ設定可能です。

例:

```
glueContext.forEachBatch(
    frame = data_frame_datasource0,
    batch_function = processBatch,
    options = {
        "windowSize": "100 seconds",
        "checkpointLocation": "s3://kafka-auth-dataplane/confluent-test/output/
checkpoint/"
    }
)

def processBatch(data_frame, batchId):
    if (data_frame.count() > 0):
        datasource0 = DynamicFrame.fromDF(
          glueContext.add_ingestion_time_columns(data_frame, "hour"),
          glueContext, "from_data_frame"
        )
        additionalOptions_datasink1 = {"enableUpdateCatalog": True}
        additionalOptions_datasink1["partitionKeys"] = ["ingest_yr", "ingest_mo",
 "ingest_day"]
        datasink1 = glueContext.write_dynamic_frame.from_catalog(
          frame = datasource0,
          database = "tempdb",
          table_name = "kafka-auth-table-output",
          transformation_ctx = "datasink1",
          additional_options = additionalOptions_datasink1
        )
```

Simple Storage Service (Amazon S3) でのデータセットの操作

- purge_table

- purge_s3_path
- transition_table
- transition_s3_path

purge_table

purge_table(catalog_id=None, database="", table_name="", options={}, transformation_ctx="")

指定したカタログのデータベースとテーブルのファイルを Simple Storage Service (Amazon S3) から削除します。パーティション内のすべてのファイルが削除されると、そのパーティションもカタログから削除されます。Lake Formation に登録されたテーブルでは、purge_table アクションはサポートされていません。

削除したオブジェクトを回復できるようにするには、Amazon S3 バケットでオブジェクトのバージョニングを有効にします。オブジェクトバージョニングが有効になっていないバケットからオブジェクトが削除された場合、そのオブジェクトは復元できません。バージョニングが有効にされているバケットで削除されたオブジェクトを復元する方法の詳細については、AWS サポート ナレッジセンターで「バージョニングが有効なバケットで削除された Simple Storage Service (Amazon S3) オブジェクトを取得するにはどうすればよいですか?」を参照してください。

- catalog_id – 現在アクセスされているデータカタログのカタログ ID (データカタログのアカウント ID)。デフォルトでは、None に設定されています。None のデフォルト値は、サービス内の呼び出し元アカウントのカタログ ID になります。
- database – 使用するデータベース。
- table_name – 使用するテーブルの名前。
- options – 削除するファイルのフィルタリングと、マニフェストファイルの生成のためオプション。
  - retentionPeriod – ファイルを保持する期間を時間単位で指定します。保存期間より新しいファイルは保持されます。デフォルトでは 168 時間 (7 日) に設定されています。
  - partitionPredicate – この述語を満たすパーティションは削除されます。これらのパーティションの保存期間内のファイルは削除されません。"" を設定 – デフォルトでは空です。
  - excludeStorageClasses – excludeStorageClasses セット内のストレージクラスを持つファイルは削除されません。デフォルトは Set() – 空のセットです。

- manifestFilePath – マニフェストファイルを生成するためのオプションのパス。正常にパージされたすべてのファイルが Success.csv に記録され、失敗したファイルは Failed.csv に記録されます。
- transformation_ctx - 使用する変換コンテキスト (オプション)。マニフェストファイルパスで使用されます。

Example

```
glueContext.purge_table("database", "table", {"partitionPredicate": "(month=='march')",
 "retentionPeriod": 1, "excludeStorageClasses": ["STANDARD_IA"], "manifestFilePath":
 "s3://bucketmanifest/"})
```

purge_s3_path

purge_s3_path(s3_path, options={}, transformation_ctx="")

指定された Amazon S3 パスからファイルを再帰的に削除します。

削除したオブジェクトを回復できるようにするには、Amazon S3 バケットでオブジェクトのバージョニングを有効にします。オブジェクトバージョニングが有効になっていないバケットからオブジェクトが削除された場合、そのオブジェクトは復元できません。バージョニングされたバケットで削除されたオブジェクトを復元する方法の詳細については、サポート ナレッジセンターで「バージョニングが有効なバケットで削除された Simple Storage Service (Amazon S3) オブジェクトを取得するにはどうすればよいですか?」を参照してください。

- s3_path – s3://<*bucket*>/<*prefix*>/ 形式で削除するファイルの Amazon S3 でのパス
- options – 削除するファイルのフィルタリングと、マニフェストファイルの生成のためオプション。
  - retentionPeriod – ファイルを保持する期間を時間単位で指定します。保存期間より新しいファイルは保持されます。デフォルトでは 168 時間 (7 日) に設定されています。
  - excludeStorageClasses – excludeStorageClasses セット内のストレージクラスを持つファイルは削除されません。デフォルトは Set() – 空のセットです。
  - manifestFilePath – マニフェストファイルを生成するためのオプションのパス。正常にパージされたすべてのファイルが Success.csv に記録され、失敗したファイルは Failed.csv に記録されます。
- transformation_ctx - 使用する変換コンテキスト (オプション)。マニフェストファイルパスで使用されます。

Example

```
glueContext.purge_s3_path("s3://bucket/path/", {"retentionPeriod": 1,
 "excludeStorageClasses": ["STANDARD_IA"], "manifestFilePath": "s3://bucketmanifest/"})
```

transition_table

`transition_table(database, table_name, transition_to, options={}, transformation_ctx="", catalog_id=None)`

指定されたカタログのデータベースとテーブルのために、Simple Storage Service (Amazon S3) に格納されているファイルの、ストレージクラスを移行します。

任意の 2 つのストレージクラス間で移行できます。GLACIER と DEEP_ARCHIVE のストレージクラスでは、これらのクラスに移行できます。ただし、GLACIER と DEEP_ARCHIVE のストレージクラスからの移行には S3 RESTORE を使用します。

Amazon S3 からファイルまたはパーティションを読み取る AWS Glue ETL ジョブを実行している場合は、一部の Amazon S3 ストレージクラスタイプを除外できます。詳細については、「Excluding Amazon S3 Storage Classes」を参照してください。

- database – 使用するデータベース。
- table_name – 使用するテーブルの名前。
- transition_to – 移行する先の Amazon S3 ストレージクラス。
- options – 削除するファイルのフィルタリングと、マニフェストファイルの生成のためオプション。
  - retentionPeriod – ファイルを保持する期間を時間単位で指定します。保存期間より新しいファイルは保持されます。デフォルトでは 168 時間 (7 日) に設定されています。
  - partitionPredicate – この述語を満たすパーティションは移行されます。これらのパーティションの保存期間内のファイルは移行されません。"" を設定 – デフォルトでは空です。
  - excludeStorageClasses – excludeStorageClasses セット内のストレージクラスを持つファイルは移行されません。デフォルトは Set() – 空のセットです。
  - manifestFilePath – マニフェストファイルを生成するためのオプションのパス。正常に移行されたすべてのファイルが Success.csv に記録され、失敗したファイルは Failed.csv に記録されます。
  - accountId – 移行変換を実行する Amazon Web Services アカウント ID。この変換には必須です。

  - roleArn – 移行変換を実行する AWS ロール。この変換には必須です。
- transformation_ctx - 使用する変換コンテキスト (オプション)。マニフェストファイルパスで使用されます。
- catalog_id – 現在アクセスされているデータカタログのカタログ ID (データカタログのアカウント ID)。デフォルトでは、None に設定されています。None のデフォルト値は、サービス内の呼び出し元アカウントのカタログ ID になります。

Example

```
glueContext.transition_table("database", "table", "STANDARD_IA", {"retentionPeriod": 1,
 "excludeStorageClasses": ["STANDARD_IA"], "manifestFilePath": "s3://bucketmanifest/",
 "accountId": "12345678901", "roleArn": "arn:aws:iam::123456789012:user/example-
username"})
```

transition_s3_path

transition_s3_path(s3_path, transition_to, options={}, transformation_ctx="")

指定された Simple Storage Service (Amazon S3) パス内のファイルのストレージクラスを再帰的に移行します。

任意の 2 つのストレージクラス間で移行できます。GLACIER と DEEP_ARCHIVE のストレージクラスでは、これらのクラスに移行できます。ただし、GLACIER と DEEP_ARCHIVE のストレージクラスからの移行には S3 RESTORE を使用します。

Amazon S3 からファイルまたはパーティションを読み取る AWS Glue ETL ジョブを実行している場合は、一部の Amazon S3 ストレージクラスタイプを除外できます。詳細については、「Excluding Amazon S3 Storage Classes」を参照してください。

- s3_path – s3://<*bucket*>/<*prefix*>/ 形式で移行するファイルの Amazon S3 でのパス
- transition_to – 移行する先の Amazon S3 ストレージクラス。
- options – 削除するファイルのフィルタリングと、マニフェストファイルの生成のためオプション。
  - retentionPeriod – ファイルを保持する期間を時間単位で指定します。保存期間より新しいファイルは保持されます。デフォルトでは 168 時間 (7 日) に設定されています。
  - partitionPredicate – この述語を満たすパーティションは移行されます。これらのパーティションの保存期間内のファイルは移行されません。"" を設定 – デフォルトでは空です。

- excludeStorageClasses – excludeStorageClasses セット内のストレージクラスを持つファイルは移行されません。デフォルトは Set() – 空のセットです。
- manifestFilePath – マニフェストファイルを生成するためのオプションのパス。正常に移行されたすべてのファイルが Success.csv に記録され、失敗したファイルは Failed.csv に記録されます。
- accountId – 移行変換を実行する Amazon Web Services アカウント ID。この変換には必須です。
- roleArn – 移行変換を実行する AWS ロール。この変換には必須です。

- transformation_ctx - 使用する変換コンテキスト (オプション)。マニフェストファイルパスで使用されます。

Example

```
glueContext.transition_s3_path("s3://bucket/prefix/", "STANDARD_IA",
 {"retentionPeriod": 1, "excludeStorageClasses": ["STANDARD_IA"],
 "manifestFilePath": "s3://bucketmanifest/", "accountId": "12345678901", "roleArn":
 "arn:aws:iam::123456789012:user/example-username"})
```

抽出

- extract_jdbc_conf

extract_jdbc_conf

extract_jdbc_conf(connection_name, catalog_id = None)

データカタログの AWS Glue 接続オブジェクトから、設定プロパティを使用するキーを持つ dict を返します。

- user - データベースのユーザー名です。
- password - データベースのパスワードです。
- vendor - ベンダーを指定します (mysql、postgresql、oracle、sqlserver など)。
- enforceSSL - 安全な接続が必要かどうかを示すブール文字列です。
- customJDBCCert - 提示された Amazon S3 パスからの特定のクライアント証明書を使用します。

- skipCustomJDBCCertValidation - customJDBCCert が CA によって検証される必要があるかどうかを示すブール文字列です。
- customJDBCCertString - ドライバーの種類に固有のカスタム証明書に関する追加情報です。
- url - (廃止) プロトコル、サーバー、ポートのみを含む JDBC URL です。
- fullUrl - 接続の作成時に入力された JDBC URL です (AWS Glue バージョン 3.0 以降で使用可能)。

JDBC 設定の取得例:

```
jdbc_conf = glueContext.extract_jdbc_conf(connection_name="your_glue_connection_name")
print(jdbc_conf)
>>> {'enforceSSL': 'false', 'skipCustomJDBCCertValidation': 'false', 'url':
 'jdbc:mysql://myserver:3306', 'fullUrl': 'jdbc:mysql://myserver:3306/mydb',
 'customJDBCCertString': '', 'user': 'admin', 'customJDBCCert': '', 'password': '1234',
 'vendor': 'mysql'}
```

トランザクション

- start_transaction
- commit_transaction
- cancel_transaction

start_transaction

`start_transaction(read_only)`

新しいトランザクションの開始。Lake Formation startTransaction API を内部的に呼び出します。

- read_only — (ブール値) このトランザクションを読み取り専用にするか、または読み取りおよび書き込みを行うかを示します。読み取り専用のトランザクション ID を使用した書き込みは拒否されます。読み取り専用トランザクションはコミットする必要はありません。

トランザクション ID を返します。

commit_transaction

commit_transaction(transaction_id, wait_for_commit = True)

指定されたトランザクションをコミットしようとします。commit_transaction でトランザクションのコミットが完了する前に戻ることがあります。Lake Formation startTransaction API を内部的に呼び出します。

- transaction_id — (文字列) コミットするトランザクション。
- wait_for_commit — (ブール値) commit_transaction がすぐに戻るかどうか指定します。デフォルト値は True です。false の場合、commit_transaction はトランザクションがコミットされるまでポーリングし待機します。最大で 6 回の再試行でエクスポネンシャルバックオフを使用すると、待機時間は 1 分に制限されます。

コミットが完了したかどうかを示すブール値を返します。

cancel_transaction

cancel_transaction(transaction_id)

指定されたトランザクションをキャンセルしようとします。戻り値は、トランザクションが以前にコミットされた場合は TransactionCommittedException 例外です。Lake Formation startTransaction API を内部的に呼び出します。

- transaction_id — (文字列) キャンセルするトランザクション。

書き込み

- getSink
- write_dynamic_frame_from_options
- write_from_options
- write_dynamic_frame_from_catalog
- write_data_frame_from_catalog
- write_dynamic_frame_from_jdbc_conf
- write_from_jdbc_conf

getSink

getSink(connection_type, format = None, transformation_ctx = "", **options)

外部ソースに DynamicFrames を書き込むために使用できる DataSink オブジェクトを取得します。期待しているシンクを確実に取得するために、SparkSQL format を最初に確認します。

- connection_type – 使用する接続タイプ (Amazon S3、Amazon Redshift、JDBC など)。有効な値には、s3、mysql、postgresql、redshift、sqlserver、oracle、kinesis、kafka が含まれています。
- format – 使用する SparkSQL 形式 (オプション)。
- transformation_ctx - 使用する変換コンテキスト (オプション)。
- options — 接続オプションの指定に使用される名前と値のペアの集合。指定できる値は以下のとおりです。
  - user と password: 認可用
  - url: データストアのエンドポイント
  - dbtable: ターゲットテーブルの名前。
  - bulkSize: 挿入操作の並列度の度合い

指定できるオプションは、接続タイプによって異なります。追加の値と例については、「AWS Glue for Spark での ETL の接続タイプとオプション」を参照してください。

例:

```
>>> data_sink = context.getSink("s3")
>>> data_sink.setFormat("json"),
>>> data_sink.writeFrame(myFrame)
```

write_dynamic_frame_from_options

write_dynamic_frame_from_options(frame, connection_type, connection_options={}, format=None, format_options={}, transformation_ctx = "")

指定された接続と形式を使用して DynamicFrame を書き込み、返します。

- frame - 書き込む DynamicFrame。

- connection_type – 接続タイプ (Amazon S3、Amazon Redshift、JDBC など)。有効な値には、s3、mysql、postgresql、redshift、sqlserver、oracle、kinesis、kafka が含まれています。
- connection_options - 接続オプション (パスやデータベーステーブルなど) (オプション)。s3 の connection_type では、Amazon S3 パスが定義されています。

```
connection_options = {"path": "s3://aws-glue-target/temp"}
```

JDBC 接続の場合、いくつかのプロパティを定義する必要があります。データベース名は URL の一部である必要があることに注意してください。オプションで接続オプションに含めることができます。

> **Warning**
>
> スクリプトにパスワードを保存することはお勧めしません。AWS Secrets Manager または AWS Glue データカタログから取得する場合には、boto3 を使用することを検討してください。

```
connection_options = {"url": "jdbc-url/database", "user": "username",
 "password": passwordVariable,"dbtable": "table-name", "redshiftTmpDir": "s3-tempdir-
path"}
```

dbtable プロパティは JDBC テーブルの名前です。データベース内でスキーマをサポートする JDBC データストアの場合、schema.table-name を指定します。スキーマを指定しない場合、デフォルトの「パブリック」スキーマが使用されます。

詳細については、「AWS Glue for Spark での ETL の接続タイプとオプション」を参照してください。

- format – フォーマット仕様 Amazon S3 や、複数のフォーマットをサポートする AWS Glue 接続の場合に使用します。サポートされる形式については、「AWS Glue for Spark での入出力のデータ形式に関するオプション」を参照してください。
- format_options – 指定された形式についてのオプション。サポートされる形式については、「AWS Glue for Spark での入出力のデータ形式に関するオプション」を参照してください。
- transformation_ctx - 使用する変換コンテキスト (オプション)。

write_from_options

`write_from_options(frame_or_dfc, connection_type, connection_options={}, format={}, format_options={}, transformation_ctx = "")`

指定された接続および形式情報で作成された DynamicFrame または DynamicFrameCollection を書き込み、返します。

- `frame_or_dfc` - 書き込む DynamicFrame または DynamicFrameCollection。
- `connection_type` – 接続タイプ (Amazon S3、Amazon Redshift、JDBC など)。有効な値には、`s3`、`mysql`、`postgresql`、`redshift`、`sqlserver`、および `oracle` があります。
- `connection_options` - 接続オプション (パスやデータベーステーブルなど) (オプション)。`s3` の `connection_type` では、Amazon S3 パスが定義されています。

```
connection_options = {"path": "s3://aws-glue-target/temp"}
```

JDBC 接続の場合、いくつかのプロパティを定義する必要があります。データベース名は URL の一部である必要があることに注意してください。オプションで接続オプションに含めることができます。

> **Warning**
>
> スクリプトにパスワードを保存することはお勧めしません。AWS Secrets Manager または AWS Glue データカタログから取得する場合には、boto3 を使用することを検討してください。

```
connection_options = {"url": "jdbc-url/database", "user": "username",
 "password": passwordVariable,"dbtable": "table-name", "redshiftTmpDir": "s3-tempdir-path"}
```

`dbtable` プロパティは JDBC テーブルの名前です。データベース内でスキーマをサポートする JDBC データストアの場合、`schema.table-name` を指定します。スキーマを指定しない場合、デフォルトの「パブリック」スキーマが使用されます。

詳細については、「AWS Glue for Spark での ETL の接続タイプとオプション」を参照してください。

- format – フォーマット仕様 Amazon S3 や、複数のフォーマットをサポートする AWS Glue 接続の場合に使用します。サポートされる形式については、「AWS Glue for Spark での入出力のデータ形式に関するオプション」を参照してください。
- format_options – 指定された形式についてのオプション。サポートされる形式については、「AWS Glue for Spark での入出力のデータ形式に関するオプション」を参照してください。
- transformation_ctx - 使用する変換コンテキスト (オプション)。

write_dynamic_frame_from_catalog

write_dynamic_frame_from_catalog(frame, database, table_name, redshift_tmp_dir, transformation_ctx = "", additional_options = {}, catalog_id = None)

データカタログデータベースとテーブルからの情報を使用して、記述した DynamicFrame を返します。

- frame - 書き込む DynamicFrame。
- Database – テーブルを含むデータカタログデータベース。
- table_name – ターゲットに関連付けられたデータカタログテーブルの名前。
- redshift_tmp_dir - 使用する Amazon Redshift の一時ディレクトリ (オプション)。
- transformation_ctx - 使用する変換コンテキスト (オプション)。
- additional_options - オプションの名前と値のペアのコレクション。
- catalog_id – 現在アクセスされているデータカタログのカタログ ID (アカウント ID)。None の場合は、呼び出し元のアカウント ID のデフォルトが使用されます。

write_data_frame_from_catalog

write_data_frame_from_catalog(frame, database, table_name, redshift_tmp_dir, transformation_ctx = "", additional_options = {}, catalog_id = None)

データカタログデータベースとテーブルからの情報を使用して、記述した DataFrame を返します。このメソッドは、データレイク形式 (Hudi、Iceberg、および Delta Lake) への書き込みをサポートします。詳細については、「AWS Glue ETL ジョブでのデータレイクフレームワークの使用」を参照してください。

- frame - 書き込む DataFrame。
- Database – テーブルを含むデータカタログデータベース。
- table_name – ターゲットに関連付けられたデータカタログテーブルの名前。
- redshift_tmp_dir - 使用する Amazon Redshift の一時ディレクトリ (オプション)。
- transformation_ctx - 使用する変換コンテキスト (オプション)。
- additional_options - オプションの名前と値のペアのコレクション。
  - useSparkDataSink – true に設定すると、AWS Glue はネイティブ Spark データシンク API を使用してテーブルに書き込むようになります。このオプションを有効にすると、必要に応じて任意の Spark データソースオプションを additional_options に追加できます。AWSGlue はこれらのオプションを Spark ライターに直接渡します。
- catalog_id – 現在アクセスされているデータカタログのカタログ ID (アカウント ID)。値を指定しない場合、呼び出し元のアカウント ID が使用されます。

制約事項

useSparkDataSink オプションを使用する際には、次の制限事項を考慮してください。

- useSparkDataSink オプションを使用する場合、enableUpdateCatalog オプションはサポートされません。

例: Spark データソースライターを使用して Hudi テーブルに書き込む

```
hudi_options = {
    'useSparkDataSink': True,
    'hoodie.table.name': <table_name>,
    'hoodie.datasource.write.storage.type': 'COPY_ON_WRITE',
    'hoodie.datasource.write.recordkey.field': 'product_id',
    'hoodie.datasource.write.table.name': <table_name>,
    'hoodie.datasource.write.operation': 'upsert',
    'hoodie.datasource.write.precombine.field': 'updated_at',
    'hoodie.datasource.write.hive_style_partitioning': 'true',
    'hoodie.upsert.shuffle.parallelism': 2,
    'hoodie.insert.shuffle.parallelism': 2,
    'hoodie.datasource.hive_sync.enable': 'true',
    'hoodie.datasource.hive_sync.database': <database_name>,
    'hoodie.datasource.hive_sync.table': <table_name>,
    'hoodie.datasource.hive_sync.use_jdbc': 'false',
    'hoodie.datasource.hive_sync.mode': 'hms'}
```

```
glueContext.write_data_frame.from_catalog(
    frame = <df_product_inserts>,
    database = <database_name>,
    table_name = <table_name>,
    additional_options = hudi_options
)
```

write_dynamic_frame_from_jdbc_conf

write_dynamic_frame_from_jdbc_conf(frame, catalog_connection, connection_options={}, redshift_tmp_dir = "", transformation_ctx = "", catalog_id = None)

指定された JDBC 接続情報を使用して DynamicFrame を書き込み、返します。

- frame - 書き込む DynamicFrame。
- catalog_connection - 使用するカタログ接続。
- connection_options - 接続オプション (パスやデータベーステーブルなど) (オプション)。詳しくは、「AWS Glue for Spark での ETL の接続タイプとオプション」を参照してください。
- redshift_tmp_dir - 使用する Amazon Redshift の一時ディレクトリ (オプション)。
- transformation_ctx - 使用する変換コンテキスト (オプション)。
- catalog_id – 現在アクセスされているデータカタログのカタログ ID (アカウント ID)。None の場合は、呼び出し元のアカウント ID のデフォルトが使用されます。

write_from_jdbc_conf

write_from_jdbc_conf(frame_or_dfc, catalog_connection, connection_options={}, redshift_tmp_dir = "", transformation_ctx = "", catalog_id = None)

指定された JDBC 接続情報を使用して DynamicFrame または DynamicFrameCollection を書き込み、返します。

- frame_or_dfc - 書き込む DynamicFrame または DynamicFrameCollection。
- catalog_connection - 使用するカタログ接続。
- connection_options - 接続オプション (パスやデータベーステーブルなど) (オプション)。詳しくは、「AWS Glue for Spark での ETL の接続タイプとオプション」を参照してください。

- `redshift_tmp_dir` - 使用する Amazon Redshift の一時ディレクトリ (オプション)。
- `transformation_ctx` - 使用する変換コンテキスト (オプション)。
- `catalog_id` – 現在アクセスされているデータカタログのカタログ ID (アカウント ID)。None の場合は、呼び出し元のアカウント ID のデフォルトが使用されます。

AWS Glue PySpark 変換リファレンス

AWS Glue には、PySpark ETL オペレーションに使用できる、次の組み込み変換が用意されています。データは、変換から変換へと DynamicFrame というデータ構造で渡されます。これは、Apache Spark SQL `DataFrame` を拡張したものです。`DynamicFrame` にはデータが含まれており、データを処理するためにそのスキーマを参照します。

これらの変換のほとんどは、`DynamicFrame` クラスのメソッドとしても存在します。詳細については、「DynamicFrame transforms」(DynamicFrame 変換) を参照してください。

- GlueTransform 基本クラス
- ApplyMapping クラス
- DropFields クラス
- DropNullFields クラス
- ErrorsAsDynamicFrame クラス
- EvaluateDataQuality クラス
- FillMissingValues クラス
- フィルタクラス
- FindIncrementalMatches クラス
- FindMatches クラス
- FlatMap クラス
- Join クラス
- マップクラス
- MapToCollection クラス
- mergeDynamicFrame
- クラスの関連付け
- RenameField クラス
- ResolveChoice クラス

- SelectFields クラス
- SelectFromCollection クラス
- Simplify_ddb_json class
- スピゴットクラス
- SplitFields クラス
- SplitRows クラス
- Unbox クラス
- UnnestFrame クラス

GlueTransform 基本クラス

すべての `awsglue.transforms` クラスが継承する基本クラス。

クラスはすべて `__call__` メソッドを定義します。次のセクションにリストされている `GlueTransform` クラスのメソッドを上書きするか、デフォルトでクラス名を使用して呼び出されます。

方法

- apply(cls, *args, **kwargs)
- name(cls)
- describeArgs(cls)
- describeReturn(cls)
- describeTransform(cls)
- describeErrors(cls)
- describe(cls)

apply(cls, *args, **kwargs)

変換クラスを呼び出して変換を適用し、結果を返します。

- `cls` - `self` クラスオブジェクト。

name(cls)

派生変換クラスの名前を返します。

- `cls` - `self` クラスオブジェクト。

describeArgs(cls)

- `cls` - `self` クラスオブジェクト。

名前付き引数にそれぞれ対応する辞書のリストを次の形式で返します。

```
[
  {
    "name": "(name of argument)",
    "type": "(type of argument)",
    "description": "(description of argument)",
    "optional": "(Boolean, True if the argument is optional)",
    "defaultValue": "(Default value string, or None)(String; the default value, or
 None)"
  },
...
]
```

実装されていない派生変換で呼び出されたときに `NotImplementedError` 例外が発生します。

describeReturn(cls)

- `cls` - `self` クラスオブジェクト。

戻り型に関する情報を含む辞書を次の形式で返します。

```
{
  "type": "(return type)",
  "description": "(description of output)"
}
```

実装されていない派生変換で呼び出されたときに `NotImplementedError` 例外が発生します。

describeTransform(cls)

変換について説明する文字列を返します。

- `cls` - `self` クラスオブジェクト。

実装されていない派生変換で呼び出されたときに NotImplementedError 例外が発生します。

describeErrors(cls)

- cls - self クラスオブジェクト。

この変換によってスローされる可能性のある例外をそれぞれ説明する辞書のリストを、次の形式で返します。

```
[
  {
    "type": "(type of error)",
    "description": "(description of error)"
  },
...
]
```

describe(cls)

- cls - self クラスオブジェクト。

次の形式のオブジェクトを返します。

```
{
  "transform" : {
    "name" : cls.name( ),
    "args" : cls.describeArgs( ),
    "returns" : cls.describeReturn( ),
    "raises" : cls.describeErrors( ),
    "location" : "internal"
  }
}
```

ApplyMapping クラス

DynamicFrame でマッピングを適用します。

例

DynamicFrame.apply_mapping() メソッドを使用して、DynamicFrame でマッピングを適用することをお勧めします。コード例については、「例: apply_mapping を使用してフィールドの名前を変更し、フィールドタイプを変更する」を参照してください。

方法

- __call__
- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- 説明

__call__(frame, mappings, transformation_ctx = "", info = "", stageThreshold = 0, totalThreshold = 0)

指定された DynamicFrame に宣言型のマッピングを適用します。

- frame – マッピングを適用するための DynamicFrame (必須)。
- mappings — マッピングタプルのリスト (必須)。それぞれが (ソース列、ソースタイプ、ターゲット列、ターゲットタイプ) で構成されます。

  ソース列で、名前にドット「.」が含まれている場合、バックティック「``」で囲む必要があります。例えば、this.old.name (文字列) を thisNewName にマッピングするには、次のタプルを使用します。

  ```
  ("`this.old.name`", "string", "thisNewName", "string")
  ```
- transformation_ctx - 状態情報を識別するために使用される一意の文字列 (オプション)。
- info - 変換のエラーに関連付けられた文字列 (オプション)。
- stageThreshold – エラーを出力する前に、変換で発生する可能性のあるエラーの最大数 (オプション)。デフォルト値は 0 です。
- totalThreshold – エラーの出力を処理する前に、全体で発生する可能性のあるエラーの最大数 (オプション)。デフォルト値は 0 です。

「マッピング」タプルで指定された DynamicFrame のフィールドのみを返します。

apply(cls, *args, **kwargs)

継承元は GlueTransform apply。

name(cls)

継承元は GlueTransform name。

describeArgs(cls)

継承元は GlueTransform describeArgs。

describeReturn(cls)

継承元は GlueTransform describeReturn。

describeTransform(cls)

継承元は GlueTransform describeTransform。

describeErrors(cls)

継承元は GlueTransform describeErrors。

describe(cls)

継承元は GlueTransform 説明。

DropFields クラス

DynamicFrame 内のフィールドを削除します。

例

DynamicFrame.drop_fields() メソッドを使用して、DynamicFrame からフィールドを削除することをお勧めします。コード例については、「例: drop_fields を使用して DynamicFrame からフィールドを削除する」を参照してください。

方法

- __call__
- apply

- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- 説明

__call__(frame, paths, transformation_ctx = "", info = "", stageThreshold = 0, totalThreshold = 0)

DynamicFrame 内のノードを削除します。

- frame – ノードを削除する DynamicFrame (必須)。
- paths – 削除するノードへの完全パスのリスト (必須)。
- transformation_ctx - 状態情報を識別するために使用される一意の文字列 (オプション)。
- info - 変換のエラーに関連付けられた文字列 (オプション)。
- stageThreshold – エラーを出力する前に、変換で発生する可能性のあるエラーの最大数 (オプション)。デフォルト値は 0 です。
- totalThreshold – エラーの出力を処理する前に、全体で発生する可能性のあるエラーの最大数 (オプション)。デフォルト値は 0 です。

指定したフィールドを除く新しい DynamicFrame を返します。

apply(cls, *args, **kwargs)

継承元は GlueTransform apply。

name(cls)

継承元は GlueTransform name。

describeArgs(cls)

継承元は GlueTransform describeArgs。

describeReturn(cls)

継承元は GlueTransform describeReturn。

describeTransform(cls)

継承元は GlueTransform describeTransform。

describeErrors(cls)

継承元は GlueTransform describeErrors。

describe(cls)

継承元は GlueTransform 説明。

DropNullFields クラス

DynamicFrame でタイプが NullType のすべての null フィールドを削除します。これらは、DynamicFrame データセット内のすべてのレコードに、値がないか null 値があるフィールドです。

例

この例では、DropNullFields を使用して、タイプ NullType のフィールドが削除された新しい DynamicFrame を作成します。DropNullFields を示すため、null 型の empty_column という名前の新しい列を、ロード済みの persons データセットに追加します。

> **Note**
>
> この例で使用されているデータセットにアクセスするには、「コード例: データの結合と関係付け」を参照し、「ステップ 1: Amazon S3 バケット内のデータをクロールする」の手順に従います。

```
# Example: Use DropNullFields to create a new DynamicFrame without NullType fields

from pyspark.context import SparkContext
from awsglue.context import GlueContext
from pyspark.sql.functions import lit
from pyspark.sql.types import NullType
from awsglue.dynamicframe import DynamicFrame
from awsglue.transforms import DropNullFields

# Create GlueContext
sc = SparkContext.getOrCreate()
```

```
glueContext = GlueContext(sc)

# Create DynamicFrame
persons = glueContext.create_dynamic_frame.from_catalog(
    database="legislators", table_name="persons_json"
)
print("Schema for the persons DynamicFrame:")
persons.printSchema()

# Add new column "empty_column" with NullType
persons_with_nulls = persons.toDF().withColumn("empty_column",
 lit(None).cast(NullType()))
persons_with_nulls_dyf = DynamicFrame.fromDF(persons_with_nulls, glueContext,
 "persons_with_nulls")
print("Schema for the persons_with_nulls_dyf DynamicFrame:")
persons_with_nulls_dyf.printSchema()

# Remove the NullType field
persons_no_nulls = DropNullFields.apply(persons_with_nulls_dyf)
print("Schema for the persons_no_nulls DynamicFrame:")
persons_no_nulls.printSchema()
```

出力

```
Schema for the persons DynamicFrame:
root
|-- family_name: string
|-- name: string
|-- links: array
|    |-- element: struct
|    |    |-- note: string
|    |    |-- url: string
|-- gender: string
|-- image: string
|-- identifiers: array
|    |-- element: struct
|    |    |-- scheme: string
|    |    |-- identifier: string
|-- other_names: array
|    |-- element: struct
|    |    |-- lang: string
|    |    |-- note: string
|    |    |-- name: string
```

```
|-- sort_name: string
|-- images: array
|    |-- element: struct
|    |    |-- url: string
|-- given_name: string
|-- birth_date: string
|-- id: string
|-- contact_details: array
|    |-- element: struct
|    |    |-- type: string
|    |    |-- value: string
|-- death_date: string

Schema for the persons_with_nulls_dyf DynamicFrame:
root
|-- family_name: string
|-- name: string
|-- links: array
|    |-- element: struct
|    |    |-- note: string
|    |    |-- url: string
|-- gender: string
|-- image: string
|-- identifiers: array
|    |-- element: struct
|    |    |-- scheme: string
|    |    |-- identifier: string
|-- other_names: array
|    |-- element: struct
|    |    |-- lang: string
|    |    |-- note: string
|    |    |-- name: string
|-- sort_name: string
|-- images: array
|    |-- element: struct
|    |    |-- url: string
|-- given_name: string
|-- birth_date: string
|-- id: string
|-- contact_details: array
|    |-- element: struct
|    |    |-- type: string
|    |    |-- value: string
|-- death_date: string
```

```
|-- empty_column: null

null_fields ['empty_column']
Schema for the persons_no_nulls DynamicFrame:
root
|-- family_name: string
|-- name: string
|-- links: array
|    |-- element: struct
|    |    |-- note: string
|    |    |-- url: string
|-- gender: string
|-- image: string
|-- identifiers: array
|    |-- element: struct
|    |    |-- scheme: string
|    |    |-- identifier: string
|-- other_names: array
|    |-- element: struct
|    |    |-- lang: string
|    |    |-- note: string
|    |    |-- name: string
|-- sort_name: string
|-- images: array
|    |-- element: struct
|    |    |-- url: string
|-- given_name: string
|-- birth_date: string
|-- id: string
|-- contact_details: array
|    |-- element: struct
|    |    |-- type: string
|    |    |-- value: string
|-- death_date: string
```

方法

- __call__
- apply
- name
- describeArgs

- describeReturn
- describeTransform
- describeErrors
- 説明

__call__(frame, transformation_ctx = "", info = "", stageThreshold = 0, totalThreshold = 0)

DynamicFrame でタイプが NullType のすべての null フィールドを削除します。これらは、DynamicFrame データセット内のすべてのレコードに、値がないか null 値があるフィールドです。

- frame – null フィールドを削除する DynamicFrame (必須)。
- transformation_ctx - 状態情報を識別するために使用される一意の文字列 (オプション)。
- info - 変換のエラーに関連付けられた文字列 (オプション)。
- stageThreshold – エラーを出力する前に、変換で発生する可能性のあるエラーの最大数 (オプション)。デフォルト値は 0 です。
- totalThreshold – エラーの出力を処理する前に、全体で発生する可能性のあるエラーの最大数 (オプション)。デフォルト値は 0 です。

null フィールドのない新しい DynamicFrame を返します。

apply(cls, *args, **kwargs)

- cls – cls

name(cls)

- cls – cls

describeArgs(cls)

- cls – cls

describeReturn(cls)

- cls – cls

describeTransform(cls)

- `cls` – cls

describeErrors(cls)

- `cls` – cls

describe(cls)

- `cls` – cls

ErrorsAsDynamicFrame クラス

ソース `DynamicFrame` の作成中に発生したエラーのネストされたレコードを含む `DynamicFrame` を返します。

例

`DynamicFrame.errorsAsDynamicFrame()` メソッドを使用して、エラーレコードを取得および表示することをお勧めします。コード例については、「例: errorsAsDynamicFrame を使用してエラーレコードを表示する」を参照してください。

方法

- __call__
- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- 説明

__call__(frame)

ソース DynamicFrame に関連する、ネストされたエラーレコードを含む DynamicFrame を返します。

- frame – ソース DynamicFrame (必須)。

apply(cls, *args, **kwargs)

- cls – cls

name(cls)

- cls – cls

describeArgs(cls)

- cls – cls

describeReturn(cls)

- cls – cls

describeTransform(cls)

- cls – cls

describeErrors(cls)

- cls – cls

describe(cls)

- cls – cls

EvaluateDataQuality クラス

DynamicFrame に対してデータ品質ルールセットを評価し、評価結果を含む新しい DynamicFrame を返します。

例

次のコード例は、DynamicFrame のデータ品質を評価し、データ品質結果を表示する方法を示しています。

```
from awsglue.transforms import *
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsgluedq.transforms import EvaluateDataQuality

#Create Glue context
sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

# Define DynamicFrame
legislatorsAreas = glueContext.create_dynamic_frame.from_catalog(
    database="legislators", table_name="areas_json")

# Create data quality ruleset
ruleset = """Rules = [ColumnExists "id", IsComplete "id"]"""

# Evaluate data quality
dqResults = EvaluateDataQuality.apply(
    frame=legislatorsAreas,
    ruleset=ruleset,
    publishing_options={
        "dataQualityEvaluationContext": "legislatorsAreas",
        "enableDataQualityCloudWatchMetrics": True,
        "enableDataQualityResultsPublishing": True,
        "resultsS3Prefix": "amzn-s3-demo-bucket1",
    },
)


# Inspect data quality results
dqResults.printSchema()
dqResults.toDF().show()
```

出力

```
root
|-- Rule: string
|-- Outcome: string
|-- FailureReason: string
|-- EvaluatedMetrics: map
|    |-- keyType: string
|    |-- valueType: double


+----------------------+-------+-------------+--------------------------------------+
|Rule                  |Outcome|FailureReason|EvaluatedMetrics                      |
+----------------------+-------+-------------+--------------------------------------+
|ColumnExists "id"     |Passed |null         |{}                                    |
|IsComplete "id"       |Passed |null         |{Column.first_name.Completeness -> 1.0}|
+----------------------+-------+-------------+--------------------------------------+
```

方法

- __call__
- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- 説明

__call__(frame, ruleset, publishing_options = {})

- `frame` – データ品質を評価したい DynamicFrame。
- `ruleset` – 文字列形式のデータ品質定義言語 (DQDL) ルールセット。DQDL の詳細については、データ品質定義言語 (DQDL) リファレンス のガイドを参照してください。
- `publishing_options` – 評価結果とメトリクスを発行する次のオプションを指定するディクショナリ。

- `dataQualityEvaluationContext` – AWS Glue が Amazon CloudWatch メトリクスとデータ品質結果を発行する名前空間を指定する文字列。集計されたメトリクスは CloudWatch に表示され、完全な結果は AWS Glue Studio インターフェイスに表示されます。
  - 必須：いいえ
  - デフォルト値: `default_context`
- `enableDataQualityCloudWatchMetrics`– データ品質評価の結果を CloudWatch に発行するかどうかを指定します。`dataQualityEvaluationContext` オプションを使用してメトリクスの名前空間を指定します。
  - 必須：いいえ
  - デフォルト値: False
- `enableDataQualityResultsPublishing` – データ品質結果を AWS Glue Studio インターフェイスの [Data Quality] (データ品質) タブに表示するかどうかを指定します。
  - 必須：いいえ
  - デフォルト値: True
- `resultsS3Prefix` – AWS Glue がデータ品質評価結果を書き込める Amazon S3 ロケーションを指定します。
  - 必須：いいえ
  - デフォルト値: "" (空の文字列)

apply(cls, *args, **kwargs)

継承元は `GlueTransform` apply。

name(cls)

継承元は `GlueTransform` name。

describeArgs(cls)

継承元は `GlueTransform` describeArgs。

describeReturn(cls)

継承元は `GlueTransform` describeReturn。

describeTransform(cls)

継承元は `GlueTransform` describeTransform。

describeErrors(cls)

継承元は GlueTransform describeErrors。

describe(cls)

継承元は GlueTransform 説明。

FillMissingValues クラス

FillMissingValues クラスでは、指定された DynamicFrame の中で Null 値や空の文字列を特定し、欠落した値を推定するために、線形回帰やランダムフォレストなどの機械学習手法を使用します。ETL ジョブは、入力データセットの値を使用して機械学習モデルをトレーニングします。その後、欠落した値が予測されます。

> **Tip**
>
> 増分データセットを使用する場合、各増分セットが機械学習モデルのトレーニングデータとして使用されます。したがって、結果から正確性が損なわれることがあります。

インポートの対象:

```
from awsglueml.transforms import FillMissingValues
```

方法

- 適用

apply(frame, missing_values_column, output_column ="", transformation_ctx ="", info ="", stageThreshold = 0, totalThreshold = 0)

指定された列の動的フレームの欠落値を埋め、推定値を含む新しい列を持つ新しいフレームを返します。欠落した値がない行の場合、指定した列の値が新しい列に複製されます。

- frame – 欠落値を埋める DynamicFrame 。必須。
- missing_values_column – 欠落値を含む列 (null 値や空の文字列)。必須。
- output_column – 値が欠落しているすべての行に推定値が埋められた新しい列の名前。デフォルトは、末尾に "_filled" が付いた missing_values_column の名前です。

- transformation_ctx - 状態情報を識別するために使用される一意の文字列 (オプション)。
- info - 変換のエラーに関連付けられた文字列 (オプション)。
- stageThreshold - エラーが発生する前に変換で発生する可能性のあるエラーの最大数 (オプション、デフォルト値は 0)。
- totalThreshold - 処理がエラーを出す前に全体的に発生する可能性のあるエラーの最大数 (オプション、デフォルト値は 0)。

欠落値を持つ行の推定値と、他の行の現在値を含む、追加的な列を持つ新しい DynamicFrame を返します。

フィルタクラス

指定された述語関数を満たす、入力 DynamicFrame からのレコードが含まれた新しい DynamicFrame を構築します。

例

DynamicFrame.filter() メソッドを使用して、DynamicFrame のレコードをフィルタリングすることをお勧めします。コード例については、「例: フィルターを使用して、フィールドのフィルタリングされた選択内容を取得する」を参照してください。

方法

- __call__
- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- 説明

__call__(frame, f, transformation_ctx="", info="", stageThreshold=0, totalThreshold=0))

指定された述語関数を満たす入力 DynamicFrame からのレコードを選択することにより構築された新しい DynamicFrame を返します。

- frame – 指定されたフィルタ関数を適用する先のソース DynamicFrame (必須)。
- f – DynamicFrame のそれぞれの DynamicRecord に適用する述語関数。この関数は、引数として DynamicRecord を取り、DynamicRecord がフィルタリングの要件を満たす場合は True を返し、そうでない場合は False を返す必要があります (必須)。

  DynamicRecord は DynamicFrame 内の論理レコードを表します。これは、自己記述型であり、固定スキーマに準拠しないデータに使用できる点を除いて、Spark DataFrame の行に似ています。
- transformation_ctx - 状態情報を識別するために使用される一意の文字列 (オプション)。
- info - 変換のエラーに関連付けられた文字列 (オプション)。
- stageThreshold – エラーを出力する前に、変換で発生する可能性のあるエラーの最大数 (オプション)。デフォルト値は 0 です。
- totalThreshold – エラーの出力を処理する前に、全体で発生する可能性のあるエラーの最大数 (オプション)。デフォルト値は 0 です。

apply(cls, *args, **kwargs)

継承元は GlueTransform apply。

name(cls)

継承元は GlueTransform name。

describeArgs(cls)

継承元は GlueTransform describeArgs。

describeReturn(cls)

継承元は GlueTransform describeReturn。

describeTransform(cls)

継承元は GlueTransform describeTransform。

describeErrors(cls)

継承元は GlueTransform describeErrors。

describe(cls)

継承元は GlueTransform 説明。

FindIncrementalMatches クラス

既存と増分の DynamicFrame 内で一致するレコードを識別します。さらに、一致するレコードの各グループに割り当てられる一意の識別子を使用して新しい DynamicFrame を作成します。

インポートの対象:

```
from awsglueml.transforms import FindIncrementalMatches
```

方法

- 適用

apply(existingFrame, incrementalFrame, transformId, transformation_ctx = "", info = "", stageThreshold = 0, totalThreshold = 0, enforcedMatches = None, computeMatchConfidenceScores = 0)

入力の DynamicFrame 内で一致するレコードを特定し、そのレコードの各グループに割り当てられている一意の識別子を含む新しい DynamicFrame を作成します。

- existingFrame – FindIncrementalMatches 変換を適用する、既存の、および事前一致済みの DynamicFrame 。必須。
- incrementalFrame – existingFrame に一致させるために FindIncrementalMatches 変換を適用する増分 DynamicFrame。必須。
- transformId – DynamicFrames 内のレコードに適用するために FindIncrementalMatches 変換に関連付けられた一意の ID。必須。
- transformation_ctx – 統計/ステータス情報を識別するために使用される一意の文字列。オプション。
- info – 変換のエラーに関連付けられる文字列。オプション。
- stageThreshold – エラーで終了する前に、変換で許容されるエラーの最大発生数。オプション。デフォルト値は 0 です。
- totalThreshold – エラーで終了する前に、処理で全体的に許容されるエラーの最大発生数。オプション。デフォルト値は 0 です。

- enforcedMatches – 一致を強制するために使用される DynamicFrame。オプション。デフォルトは [なし] です。
- computeMatchConfidenceScores — 一致するレコードの各グループの信頼スコアをコンピューティングするかどうかを示すブール値。オプション。デフォルトは False です。

一致するレコードの各グループに割り当てられている一意の識別子を含む、新しい DynamicFrame を返します。

FindMatches クラス

入力の DynamicFrame 内で一致するレコードを特定し、そのレコードの各グループに割り当てられている一意の識別子を含む新しい DynamicFrame を作成します。

インポートの対象:

```
from awsglueml.transforms import FindMatches
```

方法

- 適用

apply(frame, transformId, transformation_ctx = "", info = "", stageThreshold = 0, totalThreshold = 0, enforcedMatches = None, computeMatchConfidenceScores = 0)

入力の DynamicFrame 内で一致するレコードを特定し、そのレコードの各グループに割り当てられている一意の識別子を含む新しい DynamicFrame を作成します。

- frame – FindMatches 変換を適用する DynamicFrame。必須。
- transformId – DynamicFrame 内のレコードに適用するために FindMatches 変換に関連付けられた一意の ID 。必須。
- transformation_ctx – 統計/ステータス情報を識別するために使用される一意の文字列。オプション。
- info – 変換のエラーに関連付けられる文字列。オプション。
- stageThreshold – エラーで終了する前に、変換で許容されるエラーの最大発生数。オプション。デフォルト値は 0 です。
- totalThreshold – エラーで終了する前に、処理で全体的に許容されるエラーの最大発生数。オプション。デフォルト値は 0 です。

- enforcedMatches – 一致を強制するために使用される DynamicFrame。オプション。デフォルトは [なし] です。
- computeMatchConfidenceScores — 一致するレコードの各グループの信頼スコアをコンピューティングするかどうかを示すブール値。オプション。デフォルトは False です。

一致するレコードの各グループに割り当てられている一意の識別子を含む、新しい DynamicFrame を返します。

FlatMap クラス

コレクションの各 DynamicFrame に変換が適用されます。結果が単一の DynamicFrame にフラット化されるのではなく、コレクションとして保存されます。

FlatMap の例

次のスニペットの例は、FlatMap に適用したときに、ResolveChoice 変換を動的フレームのコレクションで使用する方法を示しています。入力に使用されるデータは、プレースホルダ Amazon S3 アドレス s3://bucket/path-for-data/sample.json にある JSON の中にあって、次のデータが含まれています。

JSON データの例

```
[{
    "firstname": "Arnav",
    "lastname": "Desai",
    "address": {
        "street": "6 Anyroad Avenue",
        "city": "London",
        "state": "England",
        "country": "UK"
    },
    "phone": 17235550101,
    "affiliations": [
        "General Anonymous Example Products",
        "Example Independent Research",
        "Government Department of Examples"
    ]
},
{
    "firstname": "Mary",
    "lastname": "Major",
```

```
    "address": {
        "street": "7821 Spot Place",
        "city": "Centerville",
        "state": "OK",
        "country": "US"
    },
    "phone": 19185550023,
    "affiliations": [
        "Example Dot Com",
        "Example Independent Research",
        "Example.io"
    ]
},
{
    "firstname": "Paulo",
    "lastname": "Santos",
    "address": {
        "street": "123 Maple Street",
        "city": "London",
        "state": "Ontario",
        "country": "CA"
    },
    "phone": 12175550181,
    "affiliations": [
        "General Anonymous Example Products",
        "Example Dot Com"
    ]
}]
```

Example ResolveChoice を DynamicFrameCollection に適用し、出力を表示します。

```
#Read DynamicFrame
datasource = glueContext.create_dynamic_frame_from_options("s3", connection_options =
 {"paths":["s3://bucket/path/to/file/mysamplejson.json"]}, format="json")
datasource.printSchema()
datasource.show()

## Split to create a DynamicFrameCollection
split_frame=datasource.split_fields(["firstname","lastname","address"],"personal_info","busines
split_frame.keys()
print("---")

## Use FlatMap to run ResolveChoice
```

```
kwargs = {"choice": "cast:string"}
flat = FlatMap.apply(split_frame, ResolveChoice, frame_name="frame",
 transformation_ctx='tcx', **kwargs)
flat.keys()

##Select one of the DynamicFrames
personal_info = flat.select("personal_info")
personal_info.printSchema()
personal_info.show()
print("---")

business_info = flat.select("business_info")
business_info.printSchema()
business_info.show()
```

> **Important**
>
> FlatMap.apply を呼び出すとき、frame_name パラメータは "frame" である必要があります。現在、他の値は受け付けられません。

出力例

```
root
|-- firstname: string
|-- lastname: string
|-- address: struct
|    |-- street: string
|    |-- city: string
|    |-- state: string
|    |-- country: string
|-- phone: long
|-- affiliations: array
|    |-- element: string
---
{
    "firstname": "Mary",
    "lastname": "Major",
    "address": {
        "street": "7821 Spot Place",
        "city": "Centerville",
        "state": "OK",
```

```
        "country": "US"
    },
    "phone": 19185550023,
    "affiliations": [
        "Example Dot Com",
        "Example Independent Research",
        "Example.io"
    ]
}

{
    "firstname": "Paulo",
    "lastname": "Santos",
    "address": {
        "street": "123 Maple Street",
        "city": "London",
        "state": "Ontario",
        "country": "CA"
    },
    "phone": 12175550181,
    "affiliations": [
        "General Anonymous Example Products",
        "Example Dot Com"
    ]
}
---
root
|-- firstname: string
|-- lastname: string
|-- address: struct
|    |-- street: string
|    |-- city: string
|    |-- state: string
|    |-- country: string

{
    "firstname": "Mary",
    "lastname": "Major",
    "address": {
        "street": "7821 Spot Place",
        "city": "Centerville",
        "state": "OK",
        "country": "US"
    }
```

```
}

{
    "firstname": "Paulo",
    "lastname": "Santos",
    "address": {
        "street": "123 Maple Street",
        "city": "London",
        "state": "Ontario",
        "country": "CA"
    }
}
---
root
|-- phone: long
|-- affiliations: array
|    |-- element: string

{
    "phone": 19185550023,
    "affiliations": [
        "Example Dot Com",
        "Example Independent Research",
        "Example.io"
    ]
}

{
    "phone": 12175550181,
    "affiliations": [
        "General Anonymous Example Products",
        "Example Dot Com"
    ]
}
```

方法

- __call__
- 適用
- 名前
- describeArgs
- describeReturn

- describeTransform
- describeErrors
- 説明

__call__(dfc, BaseTransform, frame_name, transformation_ctx = "", **base_kwargs)

変換をコレクション内の各 DynamicFrame に適用し、結果をフラット化します。

- dfc – フラットマップする DynamicFrameCollection (必須)。
- BaseTransform – コレクションの各メンバーに適用する GlueTransform から派生した変換 (必須)。
- frame_name – コレクションの要素を渡す引数名 (必須)。
- transformation_ctx - 状態情報を識別するために使用される一意の文字列 (オプション)。
- base_kwargs – ベース変換に渡す引数 (必須)。

ソース DynamicFrameCollection の各 DynamicFrame に変換を適用して作成された新しい DynamicFrameCollection を返します。

apply(cls, *args, **kwargs)

継承元は GlueTransform apply。

name(cls)

継承元は GlueTransform name。

describeArgs(cls)

継承元は GlueTransform describeArgs。

describeReturn(cls)

継承元は GlueTransform describeReturn。

describeTransform(cls)

継承元は GlueTransform describeTransform。

describeErrors(cls)

継承元は GlueTransform describeErrors。

describe(cls)

継承元は GlueTransform 説明。

Join クラス

2 つの DynamicFrames の等値結合を実行します。

例

DynamicFrames を結合するには、DynamicFrame.join( ) を使用することをお勧めします。コード例については、「例: 結合を使用して DynamicFrames を結合する」を参照してください。

方法

- __call__
- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- 説明

__call__(frame1, frame2, keys1, keys2, transformation_ctx = "")

2 つの DynamicFrames の等値結合を実行します。

- frame1 - 結合する最初の DynamicFrame (必須)。
- frame2 - 結合する 2 番目の DynamicFrame (必須)。
- keys1 - 最初のフレームで結合するキー (必須)。
- keys2 - 2 番目のフレームで結合するキー (必須)。
- transformation_ctx - 状態情報を識別するために使用される一意の文字列 (オプション)。

2 つの DynamicFrames を結合することで作成された新しい DynamicFrame を返します。

apply(cls, *args, **kwargs)

継承元は GlueTransform apply。

name(cls)

継承元は GlueTransform name。

describeArgs(cls)

継承元は GlueTransform describeArgs。

describeReturn(cls)

継承元は GlueTransform describeReturn。

describeTransform(cls)

継承元は GlueTransform describeTransform。

describeErrors(cls)

継承元は GlueTransform describeErrors。

describe(cls)

継承元は GlueTransform 説明。

マップクラス

入力 DynamicFrame ですべてのレコードに関数を適用して、新しい DynamicFrame をビルドします。

例

DynamicFrame.map() メソッドを使用して、DynamicFrame のすべてのレコードに関数を適用することをお勧めします。コード例については、「例: マッピングを使用して、DynamicFrame のすべてのレコードに関数を適用する」を参照してください。

方法

- __call__

- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- 説明

__call__(frame, f, transformation_ctx="", info="", stageThreshold=0, totalThreshold=0)

指定された関数を元の DynamicFrame で、すべての DynamicRecords に適用した結果の新しい DynamicFrame を返します。

- frame - マッピング関数を適用するための元の DynamicFrame (必須)。
- f – DynamicRecords 内のすべての DynamicFrame に適用する関数。この関数は、引数として DynamicRecord を取り、マッピングによって生成された新しい DynamicRecord を返す必要があります (必須)。

  DynamicRecord は DynamicFrame 内の論理レコードを表します。これは、自己記述型であり、固定スキーマに準拠しないデータに使用できる点を除いて、Apache Spark DataFrame の行に似ています。
- transformation_ctx - 状態情報を識別するために使用される一意の文字列 (オプション)。
- info - 変換のエラーに関連付けられた文字列 (オプション)。
- stageThreshold – エラーを出力する前に、変換で発生する可能性のあるエラーの最大数 (オプション)。デフォルト値は 0 です。
- totalThreshold – エラーの出力を処理する前に、全体で発生する可能性のあるエラーの最大数 (オプション)。デフォルト値は 0 です。

指定された関数を元の DynamicFrame で、すべての DynamicRecords に適用した結果の新しい DynamicFrame を返します。

apply(cls, *args, **kwargs)

継承元は GlueTransform apply。

name(cls)

継承元は GlueTransform name。

describeArgs(cls)

継承元は GlueTransform describeArgs。

describeReturn(cls)

継承元は GlueTransform describeReturn。

describeTransform(cls)

継承元は GlueTransform describeTransform。

describeErrors(cls)

継承元は GlueTransform describeErrors。

describe(cls)

継承元は GlueTransform 説明。

MapToCollection クラス

指定された DynamicFrameCollection の各 DynamicFrame に変換が適用されます。

方法

- __call__
- 適用
- 名前
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- 説明

__call__(dfc, BaseTransform, frame_name, transformation_ctx = "", **base_kwargs)

指定された DynamicFrameCollection の各 DynamicFrame に変換関数が適用されます。

- dfc - 変換関数を適用する DynamicFrameCollection (必須)。
- callable - コレクションの各メンバーに適用するコール可能な変換関数 (必須)。
- transformation_ctx - 状態情報を識別するために使用される一意の文字列 (オプション)。

ソース DynamicFrameCollection の各 DynamicFrame に変換を適用して作成された新しい DynamicFrameCollection を返します。

apply(cls, *args, **kwargs)

GlueTransform apply から継承されました。

name(cls)

継承元は GlueTransform name。

describeArgs(cls)

継承元は GlueTransform describeArgs。

describeReturn(cls)

継承元は GlueTransform describeReturn。

describeTransform(cls)

継承元は GlueTransform describeTransform。

describeErrors(cls)

継承元は GlueTransform describeErrors。

describe(cls)

継承元は GlueTransform 説明。

クラスの関連付け

DynamicFrame のネストされたスキーマをフラット化し、フラット化されたフレームから配列の列をピボットアウトします。

例

DynamicFrame.relationalize() メソッドを使用して、DynamicFrame を関係付けすることをお勧めします。コード例については、「例: relationalize を使用して、DynamicFrame のネストされたスキーマをフラット化する」を参照してください。

方法

- __call__
- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- 説明

__call__(frame, staging_path=None, name='roottable', options=None, transformation_ctx = "", info = "", stageThreshold = 0, totalThreshold = 0)

DynamicFrame を関係付け、ネストされた列をネスト解除し、配列の列をピボットすることによってフレームのリストを生成します。unnest のフェーズで生成された結合キーを使用して、ピボットされた配列の列をルートテーブルに結合できます。

- frame – 関連付ける DynamicFrame (必須)。
- staging_path – このメソッドを使用して、ピボットされたテーブルのパーティションを CSV 形式で保存する先を示すパス (オプション)。ピボットされたテーブルはこのパスから読み取ります。
- name – ルートテーブルの名前 (オプション)。
- options - オプションのパラメータのディクショナリ。現在使用されていません。
- transformation_ctx - 状態情報を識別するために使用される一意の文字列 (オプション)。
- info - 変換のエラーに関連付けられた文字列 (オプション)。
- stageThreshold – エラーを出力する前に、変換で発生する可能性のあるエラーの最大数 (オプション)。デフォルト値は 0 です。
- totalThreshold – エラーの出力を処理する前に、全体で発生する可能性のあるエラーの最大数 (オプション)。デフォルト値は 0 です。

apply(cls, *args, **kwargs)

継承元は GlueTransform apply。

name(cls)

継承元は GlueTransform name。

describeArgs(cls)

継承元は GlueTransform describeArgs。

describeReturn(cls)

継承元は GlueTransform describeReturn。

describeTransform(cls)

継承元は GlueTransform describeTransform。

describeErrors(cls)

継承元は GlueTransform describeErrors。

describe(cls)

継承元は GlueTransform 説明。

RenameField クラス

DynamicFrame 内のノードの名前を変更します。

例

DynamicFrame.rename_field() メソッドを使用して、DynamicFrame のフィールドの名前を変更することをお勧めします。コード例については、「例: rename_field を使用して、DynamicFrame のフィールドの名前を変更する」を参照してください。

方法

- __call__
- apply

- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- 説明

__call__(frame, old_name, new_name, transformation_ctx = "", info = "", stageThreshold = 0, totalThreshold = 0)

DynamicFrame 内のノードの名前を変更します。

- frame - ノードの名前を変更する DynamicFrame (必須)。
- old_name – 名前を変更するノードへのフルパス (必須)。

  古い名前にドットが含まれている場合、RenameField はバックティック (``) で囲まなければ機能しません。たとえば、this.old.name を thisNewName に置き換えるには、RenameField を次のように呼び出します。

  ```
  newDyF = RenameField(oldDyF, "`this.old.name`", "thisNewName")
  ```

- new_name – フルパスを含む新しい名前 (必須)。
- transformation_ctx - 状態情報を識別するために使用される一意の文字列 (オプション)。
- info - 変換のエラーに関連付けられた文字列 (オプション)。
- stageThreshold – エラーを出力する前に、変換で発生する可能性のあるエラーの最大数 (オプション)。デフォルト値は 0 です。
- totalThreshold – エラーの出力を処理する前に、全体で発生する可能性のあるエラーの最大数 (オプション)。デフォルト値は 0 です。

apply(cls, *args, **kwargs)

継承元は GlueTransform apply。

name(cls)

継承元は GlueTransform name。

describeArgs(cls)

継承元は GlueTransform describeArgs。

describeReturn(cls)

継承元は GlueTransform describeReturn。

describeTransform(cls)

継承元は GlueTransform describeTransform。

describeErrors(cls)

継承元は GlueTransform describeErrors。

describe(cls)

継承元は GlueTransform 説明。

ResolveChoice クラス

DynamicFrame 内で Choice 型を解決します。

例

DynamicFrame.resolveChoice() メソッドを使用して、DynamicFrame の複数の型を含むフィールドを処理することをお勧めします。コード例については、「例: resolveChoice を使用して、複数の型を含む列を処理する」を参照してください。

方法

- __call__
- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors

- 説明

__call__(frame, specs = None, choice = "", transformation_ctx = "", info = "", stageThreshold = 0, totalThreshold = 0)

DynamicFrame 内のあいまいな型を解決するための情報を提供します。作成された DynamicFrame を返します。

- frame – Choice 型を解決する DynamicFrame (必須)。
- specs – 解決する特定のあいまいさのリスト、それぞれがタプルの形式: (path, action)。path 値は特定のあいまいな要素を識別し、action 値は対応する解決を識別します。

  spec パラメータおよび choice パラメータのうち 1 つのみを使用できます。spec パラメータが None ではない場合、choice パラメータは空の文字列である必要があります。逆に、choice が空の文字列ではない場合、spec パラメータは None である必要があります。どちらのパラメータも指定されていない場合、AWS Glue はスキーマを解析し、それを使用してあいまいさを解決します。

  specs タプルの action 部分で、次の解決策のうちの 1 つを指定できます。

  - cast – キャスト先の型を指定できます (例: cast:int)。
  - make_cols – データをフラット化することで潜在的なあいまいさを解消します。たとえば、columnA が int または string の場合、解決策は、DynamicFrame に columnA_int および columnA_string という名前の 2 つの列を生成することです。
  - make_struct – 構造体を使用してデータを表現することで、潜在的なあいまいさを解決します。たとえば、列のデータが int または string の場合、make_struct アクションを使用すると、作成された DynamicFrame に int および string の両方を含む構造体の列が生成されます。
  - project – 生成された DynamicFrame で指定された型の値のみを保持して、潜在的なあいまいさを解決します。例えば、ChoiceType 列のデータが int または string となる可能性がある場合、project:string アクションを指定すると、生成された DynamicFrame から string 型以外の値が削除されます。

  path で配列を識別する場合は、あいまいさを避けるために配列名の後に空の角括弧を置きます。例えば、使用しているデータが次のように構造化されているとします。

```
"myList": [
  { "price": 100.00 },
```

```
  { "price": "$100.00" }
]
```

path を "myList[].price" に設定し、action を "cast:double" に設定すると、文字列バージョンではなく、数値バージョンの料金を選択できます。

- choice – specs パラメータが None の場合のデフォルトの解決アクション。specs パラメータが None ではない場合、空の文字列以外に設定することはできません。

  この引数では、上記の specs アクションに加えて、次のアクションもサポートされています。

  - MATCH_CATALOG – 各 ChoiceType について、指定した Data Catalog テーブル内の対応する型へのキャストを試みます。
- database – MATCH_CATALOG 選択肢で使用する AWS Glue Data Catalog データベース (MATCH_CATALOG に必要)。
- table_name – MATCH_CATALOG アクションで使用する AWS Glue Data Catalog のテーブル名 (MATCH_CATALOG に必要)。
- transformation_ctx - 状態情報を識別するために使用される一意の文字列 (オプション)。
- info - 変換のエラーに関連付けられた文字列 (オプション)。
- stageThreshold – エラーを出力する前に、変換で発生する可能性のあるエラーの最大数 (オプション)。デフォルト値は 0 です。
- totalThreshold – エラーの出力を処理する前に、全体で発生する可能性のあるエラーの最大数 (オプション)。デフォルト値は 0 です。

apply(cls, *args, **kwargs)

継承元は GlueTransform apply。

name(cls)

継承元は GlueTransform name。

describeArgs(cls)

継承元は GlueTransform describeArgs。

describeReturn(cls)

継承元は GlueTransform describeReturn。

describeTransform(cls)

継承元は GlueTransform describeTransform。

describeErrors(cls)

継承元は GlueTransform describeErrors。

describe(cls)

継承元は GlueTransform 説明。

SelectFields クラス

この SelectFields クラスは、既存の DynamicFrame から新しい DynamicFrame を作成し、指定したフィールドのみを保持します。SelectFields は、SQL SELECT ステートメントと同様の機能を提供します。

例

DynamicFrame.select_fields() メソッドを使用して、DynamicFrame からフィールドを選択することをお勧めします。コード例については、「例: select_fields を使用して、選択したフィールドで新しい DynamicFrame を作成する」を参照してください。

方法

- __call__
- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- 説明

__call__(frame, paths, transformation_ctx = "", info = "", stageThreshold = 0, totalThreshold = 0)

DynamicFrame のフィールド (ノード) を取得します。

- frame - フィールドを選択するための DynamicFrame (必須)。

- paths - 選択するフィールドへの完全パスのリスト (必須)。
- transformation_ctx - 状態情報を識別するために使用される一意の文字列 (オプション)。
- info - 変換のエラーに関連付けられた文字列 (オプション)。
- stageThreshold – エラーを出力する前に、変換で発生する可能性のあるエラーの最大数 (オプション)。デフォルト値は 0 です。
- totalThreshold – エラーの出力を処理する前に、全体で発生する可能性のあるエラーの最大数 (オプション)。デフォルト値は 0 です。

指定したフィールドのみを含む新しい DynamicFrame を返します。

apply(cls, *args, **kwargs)

継承元は GlueTransform apply。

name(cls)

継承元は GlueTransform name。

describeArgs(cls)

継承元は GlueTransform describeArgs。

describeReturn(cls)

継承元は GlueTransform describeReturn。

describeTransform(cls)

継承元は GlueTransform describeTransform。

describeErrors(cls)

継承元は GlueTransform describeErrors。

describe(cls)

継承元は GlueTransform 説明。

SelectFromCollection クラス

DynamicFrameCollection で DynamicFrame を 1 つ選択します。

例

この例では、SelectFromCollection を使用して、DynamicFrame を DynamicFrameCollection から選択します。

データセットの例

この例では、split_rows_collection と呼ばれる DynamicFrameCollection から 2 つの DynamicFrames を選択します。次のリストに示しているのは、split_rows_collection のキーです。

```
dict_keys(['high', 'low'])
```

コードの例

```
# Example: Use SelectFromCollection to select
# DynamicFrames from a DynamicFrameCollection

from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.transforms import SelectFromCollection

# Create GlueContext
sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

# Select frames and inspect entries
frame_low = SelectFromCollection.apply(dfc=split_rows_collection, key="low")
frame_low.toDF().show()

frame_high = SelectFromCollection.apply(dfc=split_rows_collection, key="high")
frame_high.toDF().show()
```

出力

```
+---+-----+-----------------------+-----------------------+
| id|index|contact_details.val.type|contact_details.val.value|
+---+-----+-----------------------+-----------------------+
|  1|    0|                    fax|           202-225-3307|
|  1|    1|                  phone|           202-225-5731|
|  2|    0|                    fax|           202-225-3307|
|  2|    1|                  phone|           202-225-5731|
```

```
|  3|    0|                   fax|             202-225-3307|
|  3|    1|                 phone|             202-225-5731|
|  4|    0|                   fax|             202-225-3307|
|  4|    1|                 phone|             202-225-5731|
|  5|    0|                   fax|             202-225-3307|
|  5|    1|                 phone|             202-225-5731|
|  6|    0|                   fax|             202-225-3307|
|  6|    1|                 phone|             202-225-5731|
|  7|    0|                   fax|             202-225-3307|
|  7|    1|                 phone|             202-225-5731|
|  8|    0|                   fax|             202-225-3307|
|  8|    1|                 phone|             202-225-5731|
|  9|    0|                   fax|             202-225-3307|
|  9|    1|                 phone|             202-225-5731|
| 10|    0|                   fax|             202-225-6328|
| 10|    1|                 phone|             202-225-4576|
+---+-----+----------------------+-------------------------+
only showing top 20 rows

+---+-----+----------------------+-------------------------+
| id|index|contact_details.val.type|contact_details.val.value|
+---+-----+----------------------+-------------------------+
| 11|    0|                   fax|             202-225-6328|
| 11|    1|                 phone|             202-225-4576|
| 11|    2|               twitter|           RepTrentFranks|
| 12|    0|                   fax|             202-225-6328|
| 12|    1|                 phone|             202-225-4576|
| 12|    2|               twitter|           RepTrentFranks|
| 13|    0|                   fax|             202-225-6328|
| 13|    1|                 phone|             202-225-4576|
| 13|    2|               twitter|           RepTrentFranks|
| 14|    0|                   fax|             202-225-6328|
| 14|    1|                 phone|             202-225-4576|
| 14|    2|               twitter|           RepTrentFranks|
| 15|    0|                   fax|             202-225-6328|
| 15|    1|                 phone|             202-225-4576|
| 15|    2|               twitter|           RepTrentFranks|
| 16|    0|                   fax|             202-225-6328|
| 16|    1|                 phone|             202-225-4576|
| 16|    2|               twitter|           RepTrentFranks|
| 17|    0|                   fax|             202-225-6328|
| 17|    1|                 phone|             202-225-4576|
+---+-----+----------------------+-------------------------+
only showing top 20 rows
```

方法

- __call__
- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- 説明

__call__(dfc, key, transformation_ctx = "")

DynamicFrameCollection から DynamicFrame を 1 つ取得します。

- dfc – 選択すべき DynamicFrame を含む DynamicFrameCollection (必須)。
- key – 選択する DynamicFrame のキー (必須)。
- transformation_ctx - 状態情報を識別するために使用される一意の文字列 (オプション)。

apply(cls, *args, **kwargs)

継承元は GlueTransform apply。

name(cls)

継承元は GlueTransform name。

describeArgs(cls)

継承元は GlueTransform describeArgs。

describeReturn(cls)

継承元は GlueTransform describeReturn。

describeTransform(cls)

継承元は GlueTransform describeTransform。

describeErrors(cls)

継承元は GlueTransform describeErrors。

describe(cls)

継承元は GlueTransform 説明。

Simplify_ddb_json class

特に DynamoDB JSON 構造にある DynamicFrame でネスト化された列を単純化し、新しい単純化された DynamicFrame を返します。

例

特に DynamoDB JSON 構造にある DynamicFrame でネスト化された列を単純化するには、DynamicFrame.simplify_ddb_json() メソッドを使用することをお勧めします。コード例については、「例: simplify_ddb_json を使用して DynamoDB JSON の単純化を呼び出す」を参照してください。

スピゴットクラス

AWS Glue ジョブで実行された変換が確認しやすくなるように、サンプルレコードを指定した送信先に書き込みます。

例

DynamicFrame.spigot() メソッドを使用して、レコードのサブセットを DynamicFrame から指定された送信先に書き込むことをお勧めします。コード例については、「例: spigot を使用して DynamicFrame のサンプルフィールドを Amazon S3 に書き込む」を参照してください。

方法

- __call__
- apply
- name
- describeArgs

- describeReturn
- describeTransform
- describeErrors
- 説明

__call__(frame, path, options, transformation_ctx = "")

変換中に指定した場所にサンプルレコードを書き込みます。

- `frame` - スピゴットする `DynamicFrame` (必須)。
- `path` – 書き込み先へのパス (必須)。
- `options` – オプションを指定する JSON のキーと値のペア (オプション)。`"topk"` オプションは、最初の k レコードを書き込む必要があることを指定します。`"prob"` オプションは、指定されたレコードを選択する確率を (10 進数で) 指定します。これを使用して、書き込むレコードを選択します。
- `transformation_ctx` - 状態情報を識別するために使用される一意の文字列 (オプション)。

apply(cls, *args, **kwargs)

`GlueTransform` apply から継承されました。

name(cls)

`GlueTransform` name から継承されました。

describeArgs(cls)

`GlueTransform` describeArgs から継承されました。

describeReturn(cls)

`GlueTransform` describeReturn から継承されました。

describeTransform(cls)

`GlueTransform` describeTransform から継承されました。

describeErrors(cls)

`GlueTransform` describeErrors から継承されました。

describe(cls)

GlueTransform 説明 から継承されました。

SplitFields クラス

指定されたフィールドで DynamicFrame を 2 つに新しく分割します。

例

DynamicFrame.split_fields() メソッドを使用して、DynamicFrame のフィールドを分割することをお勧めします。コード例については、「例: split_fields を使用して、選択したフィールドを別の DynamicFrame に分割する」を参照してください。

方法

- __call__
- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- 説明

__call__(frame, paths, name1 = None, name2 = None, transformation_ctx = "", info = "", stageThreshold = 0, totalThreshold = 0)

DynamicFrame の 1 つまたは複数のフィールドを新しい DynamicFrame に分割し、残りのフィールドを含む別の新しい DynamicFrame を作成します。

- frame – 2 つの新しい DynamicFrame に分割するためのソース (必須)。
- paths – 分割されるフィールドへの完全パスのリスト (必須)。
- name1 – 分割されるフィールドを含む DynamicFrame に割り当てる名前 (オプション)。名前が指定されていない場合、ソースフレームの名前に「1」を付加した名前が使用されます。
- name2 – 指定されたフィールドの分割後に残されるフィールドを含む、DynamicFrame に割り当てる名前 (オプション)。名前が指定されていない場合、ソースフレームの名前に「2」を付加した名前が使用されます。

- transformation_ctx - 状態情報を識別するために使用される一意の文字列 (オプション)。
- info - 変換のエラーに関連付けられた文字列 (オプション)。
- stageThreshold – エラーを出力する前に、変換で発生する可能性のあるエラーの最大数 (オプション)。デフォルト値は 0 です。
- totalThreshold – エラーの出力を処理する前に、全体で発生する可能性のあるエラーの最大数 (オプション)。デフォルト値は 0 です。

apply(cls, *args, **kwargs)

継承元は GlueTransform apply。

name(cls)

継承元は GlueTransform name。

describeArgs(cls)

継承元は GlueTransform describeArgs。

describeReturn(cls)

継承元は GlueTransform describeReturn。

describeTransform(cls)

継承元は GlueTransform describeTransform。

describeErrors(cls)

継承元は GlueTransform describeErrors。

describe(cls)

継承元は GlueTransform 説明。

SplitRows クラス

2 つの DynamicFrames を含む DynamicFrameCollection を作成します。1 つの DynamicFrame には分割するよう指定された行のみが含まれ、もう片方には残りの行すべてが含まれています。

例

DynamicFrame.split_rows() メソッドを使用して、DynamicFrame の行を分割することをお勧めします。コード例については、「例: split_rows を使用して、DynamicFrame 内の行を分割する」を参照してください。

方法

- __call__
- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- 説明

__call__(frame, comparison_dict, name1="frame1", name2="frame2", transformation_ctx = "", info = None, stageThreshold = 0, totalThreshold = 0)

DynamicFrame の 1 つ以上の行を、新しい DynamicFrame に分割します。

- frame – 2 つの新しい DynamicFrame に分割するためのソース (必須)。
- comparison_dict – 列へのフルパスを示すためのキーと、列の値の比較対象である値に対しコンパレータをマッピングする別のディクショナリを示すための値を持つ、ディクショナリ。たとえば、{"age": {">": 10, "<": 20}} は、"age" の値が 10 ~ 20 の行を分割します。ただし "age" の値がこの範囲外の行は除外されます (必須)。
- name1 – 分割される行を含む DynamicFrame に割り当てる名前 (省略可能)。
- name2 – 指定された行が分割された後に残る行を含む DynamicFrame に割り当てる名前 (省略可能)。
- transformation_ctx - 状態情報を識別するために使用される一意の文字列 (オプション)。
- info - 変換のエラーに関連付けられた文字列 (オプション)。
- stageThreshold – エラーを出力する前に、変換で発生する可能性のあるエラーの最大数 (オプション)。デフォルト値は 0 です。

- totalThreshold – エラーの出力を処理する前に、全体で発生する可能性のあるエラーの最大数 (オプション)。デフォルト値は 0 です。

apply(cls, *args, **kwargs)

継承元は GlueTransform apply。

name(cls)

継承元は GlueTransform name。

describeArgs(cls)

継承元は GlueTransform describeArgs。

describeReturn(cls)

継承元は GlueTransform describeReturn。

describeTransform(cls)

継承元は GlueTransform describeTransform。

describeErrors(cls)

継承元は GlueTransform describeErrors。

describe(cls)

継承元は GlueTransform 説明。

Unbox クラス

DynamicFrame の文字列フィールドをアンボックス (再フォーマット) します。

例

DynamicFrame.unbox() メソッドを使用して、DynamicFrame のフィールドをアンボックスすることをお勧めします。コード例については、「例: unbox を使用して、文字列フィールドを構造体にアンボックスする」を参照してください。

方法

- __call__

- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- 説明

__call__(frame, path, format, transformation_ctx = "", info="", stageThreshold=0, totalThreshold=0, **options)

DynamicFrame の文字列フィールドをアンボックスします。

- frame – フィールドをアンボックスする DynamicFrame (必須)。
- path – アンボックスする StringNode への完全パス (必須)。
- format – 形式の仕様 (オプション)。Amazon S3 や、複数の形式をサポートする AWS Glue 接続の場合に使用します。サポートされる形式については、「AWS Glue for Spark での入出力のデータ形式に関するオプション」を参照してください。
- transformation_ctx - 状態情報を識別するために使用される一意の文字列 (オプション)。
- info - 変換のエラーに関連付けられた文字列 (オプション)。
- stageThreshold – エラーを出力する前に、変換で発生する可能性のあるエラーの最大数 (オプション)。デフォルト値は 0 です。
- totalThreshold – エラーの出力を処理する前に、全体で発生する可能性のあるエラーの最大数 (オプション)。デフォルト値は 0 です。
- separator – 区切りトークン (省略可能)。
- escaper – エスケープトークン (省略可能)。
- skipFirst – データの最初の行をスキップする必要がある場合は True、スキップしない場合は False (省略可能)。
- withSchema – アンボックスされるデータのスキーマを含む文字列 (オプション)。これは常に StructType.json を使用して作成する必要があります。
- withHeader – 解凍されるデータにヘッダーが含まれている場合は True、そうでない場合は False (省略可能)。

apply(cls, *args, **kwargs)

継承元は GlueTransform apply。

name(cls)

継承元は GlueTransform name。

describeArgs(cls)

継承元は GlueTransform describeArgs。

describeReturn(cls)

継承元は GlueTransform describeReturn。

describeTransform(cls)

継承元は GlueTransform describeTransform。

describeErrors(cls)

継承元は GlueTransform describeErrors。

describe(cls)

継承元は GlueTransform 説明。

UnnestFrame クラス

DynamicFrame をネスト解除し、ネストされたオブジェクトを最上位の要素にフラット化して、配列オブジェクトの結合キーを生成します。

例

DynamicFrame.unnest() メソッドを使用して、DynamicFrame のネストされた構造をフラット化することをお勧めします。コード例については、「例: unnest を使用して、ネストされたフィールドを最上位フィールドに変換する」を参照してください。

方法

- __call__

- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- 説明

__call__(frame, transformation_ctx = "", info="", stageThreshold=0, totalThreshold=0)

DynamicFrame をネスト解除し、ネストされたオブジェクトを最上位の要素にフラット化して、配列オブジェクトの結合キーを生成します。

- frame - ネスト解除する DynamicFrame (必須)。
- transformation_ctx - 状態情報を識別するために使用される一意の文字列 (オプション)。
- info - 変換のエラーに関連付けられた文字列 (オプション)。
- stageThreshold – エラーを出力する前に、変換で発生する可能性のあるエラーの最大数 (オプション)。デフォルト値は 0 です。
- totalThreshold – エラーの出力を処理する前に、全体で発生する可能性のあるエラーの最大数 (オプション)。デフォルト値は 0 です。

apply(cls, *args, **kwargs)

継承元は GlueTransform apply。

name(cls)

継承元は GlueTransform name。

describeArgs(cls)

継承元は GlueTransform describeArgs。

describeReturn(cls)

継承元は GlueTransform describeReturn。

describeTransform(cls)

継承元は GlueTransform describeTransform。

describeErrors(cls)

継承元は GlueTransform describeErrors。

describe(cls)

継承元は GlueTransform 説明。

FlagDuplicatesInColumn class

FlagDuplicatesInColumn 変換は、各行に指定された値を持つ新しい列を返します。この列は、行のソース列の値がソース列の前の行の値と一致するかどうかを示します。一致が見つかった場合、重複としてフラグが付けられます。以前の行と一致しないため、最初の出現にはフラグが付けられません。

例

```
from pyspark.context import SparkContext
from pyspark.sql import SparkSession
from awsgluedi.transforms import *

sc = SparkContext()
spark = SparkSession(sc)

datasource1 = spark.read.json("s3://${BUCKET}/json/zips/raw/data")

try:
    df_output = column.FlagDuplicatesInColumn.apply(
        data_frame=datasource1,
        spark_context=sc,
        source_column="city",
        target_column="flag_col",
        true_string="True",
        false_string="False"
    )
except:
    print("Unexpected Error happened ")
    raise
```

出力

FlagDuplicatesInColumn 変換により、新しい列「flag_col」が「df_output」 DataFrame に追加されます。この列には、対応する行に「city」列に重複する値があるかどうかを示す文字列値が含まれます。行に重複する「city」値がある場合、「flag_col」には「true_string」値『True』が含まれます。行に一意の「city」値がある場合、「flag_col」には「false_string」値『False』が含まれます。

結果の「df_output」 DataFrame には、元の「datasource1」 DataFrame のすべての列に加えて、重複する「city」値を示す追加の「flag_col」列が含まれます。

方法

- __call__
- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- 説明

__call__(spark_context, data_frame, source_column, target_column, true_string=DEFAULT_TRUE_STRING, false_string=DEFAULT_FALSE_STRING)

FlagDuplicatesInColumn 変換は、各行に指定された値を持つ新しい列を返します。この列は、行のソース列の値がソース列の前の行の値と一致するかどうかを示します。一致が見つかった場合、重複としてフラグが付けられます。以前の行と一致しないため、最初の出現にはフラグが付けられません。

- source_column – ソース列の名前。
- target_column – ターゲット列の名前。
- true_string – ソース列の値がその列の以前の値と重複する場合に、ターゲット列に挿入される文字列。
- false_string – ソース列の値がその列の以前の値と異なる場合に、ターゲット列に挿入される文字列。

apply(cls, *args, **kwargs)

継承元は GlueTransform apply。

name(cls)

継承元は GlueTransform name。

describeArgs(cls)

継承元は GlueTransform describeArgs。

describeReturn(cls)

継承元は GlueTransform describeReturn。

describeTransform(cls)

継承元は GlueTransform describeTransform。

describeErrors(cls)

継承元は GlueTransform describeErrors。

describe(cls)

継承元は GlueTransform 説明。

FormatPhoneNumber class

FormatPhoneNumber 変換は、電話番号文字列がフォーマットされた値に変換される列を返します。

例

```
from pyspark.context import SparkContext
from pyspark.sql import SparkSession
from awsgluedi.transforms import *

sc = SparkContext()
spark = SparkSession(sc)

input_df = spark.createDataFrame(
    [
```

```
        ("408-341-5669",),
        ("4083415669",)
    ],
    ["phone"],
)

try:
    df_output = column_formatting.FormatPhoneNumber.apply(
        data_frame=input_df,
        spark_context=sc,
        source_column="phone",
        default_region="US"
    )
    df_output.show()
except:
    print("Unexpected Error happened ")
    raise
```

出力

出力は次のとおりです。

```
```
+---------------+
| phone|
+---------------+
|(408) 341-5669|
|(408) 341-5669|
+---------------+
```
```

FormatPhoneNumber 変換では、「source_column」を「『phone』」、「default_region」を「『US』」と指定します。

変換は、最初のフォーマットに関係なく、両方の電話番号を標準的な米国フォーマット「+1-408-341-5669」に正常にフォーマットします。

方法

- __call__
- apply

- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- 説明

__call__(spark_context, data_frame, source_column, phone_number_format=None, default_region=None, default_region_column=None)

FormatPhoneNumber 変換は、電話番号文字列がフォーマットされた値に変換される列を返します。

- source_column - 既存の列の名前。
- phone_number_format – 電話番号を変換するフォーマット。フォーマットが指定されていない場合、デフォルトは E.164です。これは国際的に認められている標準の電話番号フォーマットです。有効な値には次のようなものがあります。
  - E164 (E の後のピリオドは省略)
- default_region – 番号自体に国コードが存在しない場合に電話番号のリージョンを指定する 2 つまたは 3 つの大文字で構成される有効なリージョンコード。最大で、defaultRegion または defaultRegionColumn のいずれかを指定できます。
- default_region_column – アドバンストデータタイプ Country の列の名前。指定された列のリージョンコードは、電話番号自体に国コードがない場合に、電話番号の国コードを決定するために使用されます。最大で、defaultRegion または defaultRegionColumn のいずれかを指定できます。

apply(cls, *args, **kwargs)

継承元は GlueTransform apply。

name(cls)

継承元は GlueTransform name。

describeArgs(cls)

継承元は GlueTransform describeArgs。

describeReturn(cls)

継承元は GlueTransform describeReturn。

describeTransform(cls)

継承元は GlueTransform describeTransform。

describeErrors(cls)

継承元は GlueTransform describeErrors。

describe(cls)

継承元は GlueTransform 説明。

FormatCase class

FormatCase 変換により、列内の各文字列が指定されたケースタイプに変更されます。

例

```
from pyspark.context import SparkContext
from pyspark.sql import SparkSession
from awsgluedi.transforms import *

sc = SparkContext()
spark = SparkSession(sc)

datasource1 = spark.read.json("s3://${BUCKET}/json/zips/raw/data")

try:
    df_output = data_cleaning.FormatCase.apply(
        data_frame=datasource1,
        spark_context=sc,
        source_column="city",
        case_type="LOWER"
    )
except:
    print("Unexpected Error happened ")
    raise
```

出力

FormatCase 変換により、「case_type="LOWER"」 パラメータに基づいて、「city」列の値が小文字に変換されます。結果の「df_output」 DataFrame には、元の「datasource1」 DataFrame のすべての列が含まれますが、「city」列値は小文字です。

方法

- __call__
- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- 説明

__call__(spark_context、data_frame、source_column、case_type)

FormatCase 変換により、列内の各文字列が指定されたケースタイプに変更されます。

- source_column - 既存の列の名前。
- case_type – サポートされているケースタイプは CAPITAL、LOWER、UPPER、SENTENCE です。

apply(cls, *args, **kwargs)

継承元は GlueTransform apply。

name(cls)

継承元は GlueTransform name。

describeArgs(cls)

継承元は GlueTransform describeArgs。

describeReturn(cls)

継承元は GlueTransform describeReturn。

describeTransform(cls)

継承元は GlueTransform describeTransform。

describeErrors(cls)

継承元は GlueTransform describeErrors。

describe(cls)

継承元は GlueTransform 説明。

FillWithMode class

FillWithMode 変換は、指定した電話番号形式に従って列をフォーマットします。値の一部が同じ場合、タイブレーカーロジックを指定することもできます。例えば、次の値を考えてみます: 1 2 2 3 3 4

MINIMUM の modeType の場合、FillWithMode はモード値として 2 を返します。modeType が MAXIMUM の場合、モードは 3 です。AVERAGE の場合、モードは 2.5 です。

例

```
from awsglue.context import *
from pyspark.sql import SparkSession
from awsgluedi.transforms import *

sc = SparkContext()
spark = SparkSession(sc)

input_df = spark.createDataFrame(
    [
        (105.111, 13.12),
        (1055.123, 13.12),
        (None, 13.12),
        (13.12, 13.12),
        (None, 13.12),
    ],
    ["source_column_1", "source_column_2"],
```

```
)

try:
    df_output = data_quality.FillWithMode.apply(
        data_frame=input_df,
        spark_context=sc,
        source_column="source_column_1",
        mode_type="MAXIMUM"
    )
    df_output.show()
except:
    print("Unexpected Error happened ")
    raise
```

出力

コードの出力は次のようになります。

```
```
+---------------+---------------+
|source_column_1|source_column_2|
+---------------+---------------+
105.111	13.12
1055.123	13.12
1055.123	13.12
13.12	13.12
1055.123	13.12
+---------------+---------------+
```
```

「awsglue.data_quality」モジュールからの FillWithMode 変換は、「input_df」DataFrame に適用されます。これは、source_column_1 列の「null」値を、その列の非 null 値からの最大値 (「mode_type="MAXIMUM"」) に置き換えます。

この場合、source_column_1 列の最大値は「1055.123」です。したがって、source_column_1 の「null」値は、出力 DataFrame の「df_output」で「1055.123」に置き換えられます。

方法

- __call__
- apply

- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- 説明

__call__(spark_context、data_frame、source_column、mode_type)

FillWithMode 変換は、列内の文字列のケースをフォーマットします。

- source_column - 既存の列の名前。
- mode_type – データ内のタイ値を解決する方法。この値は、MINIMUM、NONE、AVERAGE、または MAXIMUM のいずれかである必要があります。

apply(cls, *args, **kwargs)

継承元は GlueTransform apply。

name(cls)

継承元は GlueTransform name。

describeArgs(cls)

継承元は GlueTransform describeArgs。

describeReturn(cls)

継承元は GlueTransform describeReturn。

describeTransform(cls)

継承元は GlueTransform describeTransform。

describeErrors(cls)

継承元は GlueTransform describeErrors。

describe(cls)

継承元は GlueTransform 説明。

FlagDuplicateRows class

FlagDuplicateRows 変換は、各行に指定された値を持つ新しい列を返します。この列は、その行がデータセット内の以前の行と完全に一致しているかどうかを示します。一致が見つかった場合、重複としてフラグが付けられます。以前の行と一致しないため、最初の出現にはフラグが付けられません。

例

```
from pyspark.context import SparkContext
from pyspark.sql import SparkSession
from awsgluedi.transforms import *

sc = SparkContext()
spark = SparkSession(sc)

input_df = spark.createDataFrame(
    [
        (105.111, 13.12),
        (13.12, 13.12),
        (None, 13.12),
        (13.12, 13.12),
        (None, 13.12),
    ],
    ["source_column_1", "source_column_2"],
)

try:
    df_output = data_quality.FlagDuplicateRows.apply(
        data_frame=input_df,
        spark_context=sc,
        target_column="flag_row",
        true_string="True",
        false_string="False",
        target_index=1
    )
except:
    print("Unexpected Error happened ")
    raise
```

出力

出力は、source_column_1 列に基づいて行が重複しているかどうかを示す追加の列 flag_row を含む PySpark DataFrame になります。結果の「df_output」 DataFrame には、次の行が含まれます。

```
+---------------+---------------+--------+
|source_column_1|source_column_2|flag_row|
+---------------+---------------+--------+
| 105.111| 13.12| False|
| 13.12| 13.12| True|
| null| 13.12| True|
| 13.12| 13.12| True|
| null| 13.12| True|
+---------------+---------------+--------+
```

flag_row 列は、行が重複しているかどうかを示します。「true_string」は「True」に設定され、「false_string」は「False」に設定されます。「target_index」は 1 に設定されています。つまり、flag_row 列は出力 DataFrame の 2 番目の位置 (インデックス 1) に挿入されます。

方法

- __call__
- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- 説明

__call__(spark_context, data_frame, target_column, true_string=DEFAULT_TRUE_STRING, false_string=DEFAULT_FALSE_STRING, target_index=None)

FlagDuplicateRows 変換は、各行に指定された値を持つ新しい列を返します。この列は、その行がデータセット内の以前の行と完全に一致しているかどうかを示します。一致が見つかった場合、重複としてフラグが付けられます。以前の行と一致しないため、最初の出現にはフラグが付けられません。

- true_string – 行が前の行と一致する場合に挿入される値。
- false_string – 行が一意である場合に挿入される値。
- target_column – データセットに挿入された新しい列の名前。

apply(cls, *args, **kwargs)

継承元は GlueTransform apply。

name(cls)

継承元は GlueTransform name。

describeArgs(cls)

継承元は GlueTransform describeArgs。

describeReturn(cls)

継承元は GlueTransform describeReturn。

describeTransform(cls)

継承元は GlueTransform describeTransform。

describeErrors(cls)

継承元は GlueTransform describeErrors。

describe(cls)

継承元は GlueTransform 説明。

RemoveDuplicates class

選択したソース列で重複する値が検出された場合、RemoveDuplicates 変換によって行全体が削除されます。

例

```
from pyspark.context import SparkContext
from pyspark.sql import SparkSession
from awsgluedi.transforms import *

sc = SparkContext()
spark = SparkSession(sc)

input_df = spark.createDataFrame(
    [
        (105.111, 13.12),
        (13.12, 13.12),
        (None, 13.12),
        (13.12, 13.12),
        (None, 13.12),
    ],
    ["source_column_1", "source_column_2"],
)

try:
    df_output = data_quality.RemoveDuplicates.apply(
        data_frame=input_df,
        spark_context=sc,
        source_column="source_column_1"
    )
except:
    print("Unexpected Error happened ")
    raise
```

出力

出力は、source_column_1 列に基づいて重複が削除された PySpark DataFrame になります。結果の「df_output」 DataFrame には、次の行が含まれます。

```
```

```
+---------------+---------------+
|source_column_1|source_column_2|
+---------------+---------------+
| 105.111| 13.12|
| 13.12| 13.12|
| null| 13.12|
+---------------+---------------+
```

「13.12」と「null」の source_column_1 値を持つ行は、source_column_1 列に基づいて重複が削除されているため、出力 DataFrame に 1 回だけ表示されることに注意してください。

方法

- __call__
- 適用
- 名前
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- 説明

__call__(spark_context, data_frame, source_column)

選択したソース列で重複する値が検出された場合、RemoveDuplicates 変換によって行全体が削除されます。

- source_column - 既存の列の名前。

apply(cls, *args, **kwargs)

継承元は GlueTransform apply。

name(cls)

継承元は GlueTransform name。

describeArgs(cls)

継承元は GlueTransform describeArgs。

describeReturn(cls)

継承元は GlueTransform describeReturn。

describeTransform(cls)

継承元は GlueTransform describeTransform。

describeErrors(cls)

継承元は GlueTransform describeErrors。

describe(cls)

継承元は GlueTransform 説明。

MonthName class

MonthName 変換により、日付を表す文字列から、月の名前を含む新しい列が作成されます。

例

```
from pyspark.context import SparkContext
from pyspark.sql import SparkSession
from awsgluedi.transforms import *

sc = SparkContext()
spark = SparkSession(sc)

spark.conf.set("spark.sql.legacy.timeParserPolicy", "LEGACY")

input_df = spark.createDataFrame(
    [
        ("20-2018-12",),
        ("2018-20-12",),
        ("20182012",),
        ("12202018",),
        ("20122018",),
        ("20-12-2018",),
```

```
        ("12/20/2018",),
        ("02/02/02",),
        ("02 02 2009",),
        ("02/02/2009",),
        ("August/02/2009",),
        ("02/june/2009",),
        ("02/2020/june",),
        ("2013-02-21 06:35:45.658505",),
        ("August 02 2009",),
        ("2013/02/21",),
        (None,),
    ],
    ["column_1"],
)

try:
    df_output = datetime_functions.MonthName.apply(
        data_frame=input_df,
        spark_context=sc,
        source_column="column_1",
        target_column="target_column"
    )
    df_output.show()
except:
    print("Unexpected Error happened ")
    raise
```

出力

出力は次のとおりです。

```
```
+------------+------------+
| column_1|target_column|
+------------+------------+
20-2018-12	December
2018-20-12	null
20182012	null
12202018	null
20122018	null
20-12-2018	December
12/20/2018	December
02/02/02	February
```
```

```
|02 02 2009 | February |
|02/02/2009 | February |
|August/02/2009| August |
|02/june/2009| null |
|02/2020/june| null |
|2013-02-21 06:35:45.658505| February |
|August 02 2009| August |
| 2013/02/21| February |
| null | null |
+------------+------------+
```

MonthName 変換では、「source_column」を『column_1』、「target_column」を『target_column』とします。「『column_1』」列の日付/時刻文字列から月名を抽出し、「『target_column』」列に配置します。日付/時刻文字列が認識されないフォーマットである、または解析できない場合、「『target_column』」値は「null」に設定されます。

変換は、「2018-12-20」、「2018/12/20」、「2009/02/02」、「2013-02-21 06:35:45.658505」、「2009 年 8 月 2 日」など、さまざまな日付/時刻フォーマットから月名を正常に抽出します。

方法

- __call__
- 適用
- 名前
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- 説明

__call__(spark_context, data_frame, target_column, source_column=None, value=None)

MonthName 変換により、日付を表す文字列から、月の名前を含む新しい列が作成されます。

- source_column - 既存の列の名前。
- value – 評価する文字列。

- target_column – 新しく作成された列の名前。

apply(cls, *args, **kwargs)

継承元は GlueTransform apply。

name(cls)

継承元は GlueTransform name。

describeArgs(cls)

継承元は GlueTransform describeArgs。

describeReturn(cls)

継承元は GlueTransform describeReturn。

describeTransform(cls)

継承元は GlueTransform describeTransform。

describeErrors(cls)

継承元は GlueTransform describeErrors。

describe(cls)

継承元は GlueTransform 説明。

IsEven クラス

IsEven 変換は、ソース列または値が偶数であるかどうかを示す新しい列にブール値を返します。ソース列または値が 小数の場合、結果は false です。

例

```
from pyspark.context import SparkContext
from pyspark.sql import SparkSession
from awsgluedi.transforms import *
```

```
sc = SparkContext()
spark = SparkSession(sc)

input_df = spark.createDataFrame(
    [(5,), (0,), (-1,), (2,), (None,)],
    ["source_column"],
)

try:
    df_output = math_functions.IsEven.apply(
        data_frame=input_df,
        spark_context=sc,
        source_column="source_column",
        target_column="target_column",
        value=None,
        true_string="Even",
        false_string="Not even",
    )
    df_output.show()
except:
    print("Unexpected Error happened ")
    raise
```

出力

出力は次のとおりです。

```
```
+------------+------------+
|source_column|target_column|
+------------+------------+
5	Not even
0	Even
-1	Not even
2	Even
null	null
+------------+------------+
```
```

IsEven 変換では、「source_column」を『source_column』、「target_column」を『target_column』とします。『source_column』の値が偶数かどうかをチェックします。値が偶数の場合、「『target_column』」値は「true_string」『Even』に設定します。値が奇数の場合、

「『target_column』」値は「false_string」『Not even』に設定します。「『source_column』」値が「null」の場合、「『target_column』」値は「null」に設定されます。

変換は偶数 (0 と 2) を正しく識別し、「『target_column』」値を『Even』に設定します。奇数 (5 と -1) の場合、「『target_column』」の値を『Not even』に設定します。「『source_column』」の「null」値の場合、「『target_column』」値は「null」に設定されます。

方法

- __call__
- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- 説明

__call__(spark_context, data_frame, target_column, source_column=None, true_string=DEFAULT_TRUE_STRING, false_string=DEFAULT_FALSE_STRING, value=None)

IsEven 変換は、ソース列または値が偶数であるかどうかを示す新しい列にブール値を返します。ソース列または値が 小数の場合、結果は false です。

- source_column - 既存の列の名前。
- target_column – 作成される新しい列の名前。
- true_string – 値が偶数かどうかを示す文字列。
- false_string – 値が偶数ではないかどうかを示す文字列。

apply(cls, *args, **kwargs)

継承元は GlueTransform apply。

name(cls)

継承元は GlueTransform name。

describeArgs(cls)

継承元は GlueTransform describeArgs。

describeReturn(cls)

継承元は GlueTransform describeReturn。

describeTransform(cls)

継承元は GlueTransform describeTransform。

describeErrors(cls)

継承元は GlueTransform describeErrors。

describe(cls)

継承元は GlueTransform 説明。

CryptographicHash クラス

CryptographicHash 変換は、列のハッシュ値にアルゴリズムを適用します。

例

```
from pyspark.context import SparkContext
from pyspark.sql import SparkSession
from awsgluedi.transforms import *

secret = "${SECRET}"
sc = SparkContext()
spark = SparkSession(sc)

input_df = spark.createDataFrame(
    [
        (1, "1234560000"),
        (2, "1234560001"),
        (3, "1234560002"),
        (4, "1234560003"),
        (5, "1234560004"),
        (6, "1234560005"),
```

```
        (7, "1234560006"),
        (8, "1234560007"),
        (9, "1234560008"),
        (10, "1234560009"),
    ],
    ["id", "phone"],
)

try:
    df_output = pii.CryptographicHash.apply(
        data_frame=input_df,
        spark_context=sc,
        source_columns=["id", "phone"],
        secret_id=secret,
        algorithm="HMAC_SHA256",
        output_format="BASE64",
    )
    df_output.show()
except:
    print("Unexpected Error happened ")
    raise
```

出力

出力は次のとおりです。

```
+---+------------+------------------+------------------+
| id| phone | id_hashed | phone_hashed |
+---+------------+------------------+------------------+
| 1| 1234560000 | QUI1zXTJiXmfIb... | juDBAmiRnnO3g... |
| 2| 1234560001 | ZAUWiZ3dVTzCo... | vC8lgUqBVDMNQ... |
| 3| 1234560002 | ZP4VvZWkqYifu... | Kl3QAkgswYpzB... |
| 4| 1234560003 | 3u8vO3wQ8EQfj... | CPBzK1P8PZZkV... |
| 5| 1234560004 | eWkQJk4zAOIzx... | aLf7+mHcXqbLs... |
| 6| 1234560005 | xtI9fZCJZCvsa... | dy2DFgdYWmr0p... |
| 7| 1234560006 | iW9hew7jnHuOf... | wwfGMCOEv6oOv... |
| 8| 1234560007 | H9V1pqvgkFhfS... | g9WKhagIXy9ht... |
| 9| 1234560008 | xDhEuHaxAUbU5... | b3uQLKPY+Q5vU... |
| 10| 1234560009 | GRN6nFXkxk349... | VJdsKt8VbxBbt... |
+---+------------+------------------+------------------+
```

変換は、指定されたアルゴリズムとシークレットキーを使用して、「id」列と「phone」列の値の暗号化ハッシュを計算し、Base64 形式でハッシュをエンコードします。結果の「df_output」DataFrame には、元の「input_df」 DataFrame のすべての列に加えて、計算ハッシュを含む追加の「id_hashed」列と「phone_hashed」列が含まれます。

方法

- __call__
- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- 説明

__call__(spark_context, data_frame, source_columns, secret_id, algorithm=None, secret_version=None, create_secret_if_missing=False, output_format=None, entity_type_filter=None)

`CryptographicHash` 変換は、列のハッシュ値にアルゴリズムを適用します。

- `source_columns` – 既存の列の配列。
- `secret_id` – Secrets Manager シークレットキーの ARN。ソース列を Hash-based Message Authentication Code (HMAC) プレフィックスアルゴリズムで使用されるキー。
- `secret_version` - オプション。デフォルトは最新のシークレットバージョンです。
- `entity_type_filter` – エンティティタイプのオプションの配列。検出された PII のみをフリーテキスト列で暗号化するために使用できます。
- `create_secret_if_missing` – オプションのブール値。true の場合、呼び出し元に代わってシークレットの作成を試みます。
- `algorithm` – データをハッシュするために使用されるアルゴリズム。有効な列挙値: MD5, SHA1, SHA256, SHA512, HMAC_MD5, HMAC_SHA1, HMAC_SHA256, HMAC_SHA512。

apply(cls, *args, **kwargs)

継承元は `GlueTransform` apply。

name(cls)

継承元は GlueTransform name。

describeArgs(cls)

継承元は GlueTransform describeArgs。

describeReturn(cls)

継承元は GlueTransform describeReturn。

describeTransform(cls)

継承元は GlueTransform describeTransform。

describeErrors(cls)

継承元は GlueTransform describeErrors。

describe(cls)

継承元は GlueTransform 説明。

Decrypt クラス

Decrypt 変換は AWS Glue 内で復号化されます。AWS Encryption SDK を使用して、AWS Glue の外部でデータを復号することもできます。提供された KMS キー ARN が列の暗号化に使用されたものと一致しない場合、復号オペレーションは失敗します。

例

```
from pyspark.context import SparkContext
from pyspark.sql import SparkSession
from awsgluedi.transforms import *

kms = "${KMS}"
sc = SparkContext()
spark = SparkSession(sc)

input_df = spark.createDataFrame(
    [
```

```
        (1, "1234560000"),
        (2, "1234560001"),
        (3, "1234560002"),
        (4, "1234560003"),
        (5, "1234560004"),
        (6, "1234560005"),
        (7, "1234560006"),
        (8, "1234560007"),
        (9, "1234560008"),
        (10, "1234560009"),
    ],
    ["id", "phone"],
)

try:
    df_encrypt = pii.Encrypt.apply(
        data_frame=input_df,
        spark_context=sc,
        source_columns=["phone"],
        kms_key_arn=kms
    )
    df_decrypt = pii.Decrypt.apply(
        data_frame=df_encrypt,
        spark_context=sc,
        source_columns=["phone"],
        kms_key_arn=kms
    )
    df_decrypt.show()
except:
    print("Unexpected Error happened ")
    raise
```

出力

出力は、元の「id」列と復号された「phone」列を持つ PySpark DataFrame になります。

```
```
+---+------------+
| id| phone|
+---+------------+
1	1234560000
2	1234560001
3	1234560002
```
```

```
| 4| 1234560003|
| 5| 1234560004|
| 6| 1234560005|
| 7| 1234560006|
| 8| 1234560007|
| 9| 1234560008|
| 10| 1234560009|
+---+------------+
```

Encrypt 変換では、「source_columns」を「[phone]」「kms_key_arn」を「${KMS}」環境変数の値として受け取ります。変換は、指定された KMS キーを使用して、「phone」列の値を暗号化します。次に、暗号化された DataFrame 「df_encrypt」が「awsglue.pii」モジュールからの Decrypt 変換に渡されます。「source_columns」を「[phone]」とし、「kms_key_arn」を「${KMS}」環境変数の値として扱います。変換は、同じ KMS キーを使用して、「phone」列の暗号化された値を復号します。結果の「df_decrypt」DataFrame には、元の「id」列と復号された「phone」列が含まれています。

方法

- __call__
- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- 説明

__call__(spark_context、data_frame、source_columns、kms_key_arn)

Decrypt 変換は Glue AWS 内で復号化されます。AWS Encryption SDK を使用して、AWS Glue の外部でデータを復号することもできます。提供された KMS キー ARN が列の暗号化に使用されたものと一致しない場合、復号オペレーションは失敗します。

- source_columns – 既存の列の配列。
- kms_key_arn – ソース列の復号に使用する AWS Key Management Service キーのキー ARN。

apply(cls, *args, **kwargs)

継承元は GlueTransform apply。

name(cls)

継承元は GlueTransform name。

describeArgs(cls)

継承元は GlueTransform describeArgs。

describeReturn(cls)

継承元は GlueTransform describeReturn。

describeTransform(cls)

継承元は GlueTransform describeTransform。

describeErrors(cls)

継承元は GlueTransform describeErrors。

describe(cls)

継承元は GlueTransform 説明。

暗号化クラス

Encrypt 変換は、AWS Key Management Service キーを使用してソース列を暗号化します。Encrypt 変換は、セルあたり最大 128 MiB を暗号化できます。復号時にフォーマットを保持しようとします。データ型を保持するには、データ型メタデータを 1KB 未満にシリアル化する必要があります。それ以外の場合は、preserve_data_type パラメータを false に設定する必要があります。データ型メタデータは、暗号化コンテキストのプレーンテキストで保存されます。

例

```
from pyspark.context import SparkContext
from pyspark.sql import SparkSession
from awsgluedi.transforms import *
```

```
kms = "${KMS}"
sc = SparkContext()
spark = SparkSession(sc)

input_df = spark.createDataFrame(
    [
        (1, "1234560000"),
        (2, "1234560001"),
        (3, "1234560002"),
        (4, "1234560003"),
        (5, "1234560004"),
        (6, "1234560005"),
        (7, "1234560006"),
        (8, "1234560007"),
        (9, "1234560008"),
        (10, "1234560009"),
    ],
    ["id", "phone"],
)

try:
    df_encrypt = pii.Encrypt.apply(
        data_frame=input_df,
        spark_context=sc,
        source_columns=["phone"],
        kms_key_arn=kms
    )
except:
    print("Unexpected Error happened ")
    raise
```

出力

出力は、元の「id」列と「phone」列の暗号化された値を含む追加の列を持つ PySpark DataFrame になります。

```
```
+---+------------+-------------------------+
| id| phone | phone_encrypted |
+---+------------+-------------------------+
| 1| 1234560000| EncryptedData1234...abc |
| 2| 1234560001| EncryptedData5678...def |
```
```

```
| 3| 1234560002| EncryptedData9012...ghi |
| 4| 1234560003| EncryptedData3456...jkl |
| 5| 1234560004| EncryptedData7890...mno |
| 6| 1234560005| EncryptedData1234...pqr |
| 7| 1234560006| EncryptedData5678...stu |
| 8| 1234560007| EncryptedData9012...vwx |
| 9| 1234560008| EncryptedData3456...yz0 |
| 10| 1234560009| EncryptedData7890...123 |
+---+------------+------------------------+
```

Encrypt 変換では、「source_columns」を「[phone]」「kms_key_arn」を「${KMS}」環境変数の値として受け取ります。変換は、指定された KMS キーを使用して、「phone」列の値を暗号化します。結果の「df_encrypt」 DataFrame には、元の「id」列、元の「phone」列、および「phone_encrypted」列の暗号化された値を含む「phone_encrypted」という名前の追加列が含まれます。

方法

- __call__
- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- 説明

__call__(spark_context, data_frame, source_columns, kms_key_arn, entity_type_filter=None, preserve_data_type=None)

Encrypt 変換は、AWS Key Management Service キーを使用してソース列を暗号化します。

- source_columns – 既存の列の配列。
- kms_key_arn – ソース列の暗号化に使用する AWS Key Management Service キーのキー ARN。
- entity_type_filter – エンティティタイプのオプションの配列。検出された PII のみをフリーテキスト列で暗号化するために使用できます。

- `preserve_data_type` – オプションのブール値。デフォルトは true です。false の場合、データ型は保存されません。

apply(cls, *args, **kwargs)

継承元は `GlueTransform` apply。

name(cls)

継承元は `GlueTransform` name。

describeArgs(cls)

継承元は `GlueTransform` describeArgs。

describeReturn(cls)

継承元は `GlueTransform` describeReturn。

describeTransform(cls)

継承元は `GlueTransform` describeTransform。

describeErrors(cls)

継承元は `GlueTransform` describeErrors。

describe(cls)

継承元は `GlueTransform` 説明。

IntToIp class

`IntToIp` 変換は、ソース列の整数値または他の値を、次にターゲット列の対応する IPv4 値に変換し、結果を新しい列に返します。

例

```
from pyspark.context import SparkContext
from pyspark.sql import SparkSession
from awsgluedi.transforms import *
```

```
sc = SparkContext()
spark = SparkSession(sc)

input_df = spark.createDataFrame(
    [
        (3221225473,),
        (0,),
        (1,),
        (100,),
        (168430090,),
        (4294967295,),
        (4294967294,),
        (4294967296,),
        (-1,),
        (None,),
    ],
    ["source_column_int"],
)

try:
    df_output = web_functions.IntToIp.apply(
        data_frame=input_df,
        spark_context=sc,
        source_column="source_column_int",
        target_column="target_column",
        value=None
    )
    df_output.show()
except:
    print("Unexpected Error happened ")
    raise
```

出力

出力は次のとおりです。

```
```
+---------------+---------------+
|source_column_int|target_column|
+---------------+---------------+
3221225473	192.0.0.1
0	0.0.0.0
1	0.0.0.1
```
```

```
| 100| 0.0.0.100|
| 168430090 | 10.0.0.10 |
| 4294967295| 255.255.255.255|
| 4294967294| 255.255.255.254|
| 4294967296| null |
| -1| null |
| null| null |
+---------------+---------------+
```

`IntToIp.apply` 変換では、「source_column」を『source_column_int』として、「target_column」を『target_column』として受け取り、「source_column_int」列の整数値を対応する IPv4 アドレス表現に変換し、結果を「target_column」列に保存します。

IPv4 アドレス (0 ~ 4294967295) の範囲内の有効な整数値の場合、変換は IPv4 アドレス表現 (192.0.0.1、0.0.0.0、10.0.0.10、255.255.255.255 など) に正常に変換します。

有効範囲外の整数値 (例: 4294967296、-1) の場合、「target_column」値は「null」に設定されます。「source_column_int」列の「null」値の場合、「target_column」値も「null」に設定されます。

方法

- __call__
- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- 説明

__call__(spark_context, data_frame, target_column, source_column=None, value=None)

`IntToIp` 変換は、ソース列の整数値または他の値を、次にターゲット列の対応する IPv4 値に変換し、結果を新しい列に返します。

- `sourceColumn` - 既存の列の名前。
- `value` – 評価する文字列。

- `targetColumn` – 作成される新しい列の名前。

apply(cls, *args, **kwargs)

継承元は `GlueTransform` apply。

name(cls)

継承元は `GlueTransform` name。

describeArgs(cls)

継承元は `GlueTransform` describeArgs。

describeReturn(cls)

継承元は `GlueTransform` describeReturn。

describeTransform(cls)

継承元は `GlueTransform` describeTransform。

describeErrors(cls)

継承元は `GlueTransform` describeErrors。

describe(cls)

継承元は `GlueTransform` 説明。

IpToInt class

`IpToInt` 変換はソース列またはその他の値の Internet Protocol version 4 (IPv4) 値をターゲット列の対応する整数値に変換し、結果を新しい列に返します。

例

AWS Glue 4.0 以降の場合、`key: --enable-glue-di-transforms, value: true` を使用してジョブ引数を作成または更新します。

```
from pyspark.context import SparkContext
```

```
from awsgluedi.transforms import *

sc = SparkContext()

input_df = spark.createDataFrame(
    [
        ("192.0.0.1",),
        ("10.10.10.10",),
        ("1.2.3.4",),
        ("1.2.3.6",),
        ("http://12.13.14.15",),
        ("https://16.17.18.19",),
        ("1.2.3.4",),
        (None,),
        ("abc",),
        ("abc.abc.abc.abc",),
        ("321.123.123.123",),
        ("244.4.4.4",),
        ("255.255.255.255",),
    ],
    ["source_column_ip"],
)

    df_output = web_functions.IpToInt.apply(
        data_frame=input_df,
        spark_context=sc,
        source_column="source_column_ip",
        target_column="target_column",
        value=None
    )
    df_output.show()
```

出力

出力は次のとおりです。

```
```
+----------------+---------------+
|source_column_ip| target_column|
+----------------+---------------+
192.0.0.1	3221225473
10.10.10.10	168427722
1.2.3.4	16909060
```
```

```
| 1.2.3.6| 16909062|
|http://12.13.14.15| null|
|https://16.17.18.19| null|
| 1.2.3.4| 16909060|
| null| null|
| abc| null|
|abc.abc.abc.abc| null|
| 321.123.123.123| null|
| 244.4.4.4| 4102444804|
| 255.255.255.255| 4294967295|
+---------------+--------------+
```

`IpToInt` 変換では、「source_column」を『source_column_ip』として、「target_column」を『target_column』として受け取り、「source_column_ip」列の有効な IPv4 アドレス文字列を対応する 32 ビット整数表現に変換し、結果を「target_column」列に格納します。

有効な IPv4 アドレス文字列 (『192.0.0.1』、『10.10.10.10』、『1.2.3.4』など) の場合、変換はそれらを整数表現 (3221225473、168427722、16909060 など) に正常に変換します。有効な IPv4 アドレスではない文字列 (URLs 、『abc』などの非 IP 文字列、『abc.abc.abc』などの無効な IP 形式など) の場合、「target_column」値は「null」に設定されます。「source_column_ip」列の「null」値の場合、「target_column」値も「null」に設定されます。

方法

- __call__
- apply
- name
- describeArgs
- describeReturn
- describeTransform
- describeErrors
- 説明

__call__(spark_context, data_frame, target_column, source_column=None, value=None)

`IpToInt` 変換はソース列またはその他の値の Internet Protocol version 4 (IPv4) 値をターゲット列の対応する整数値に変換し、結果を新しい列に返します。

- `sourceColumn` - 既存の列の名前。
- `value` – 評価する文字列。
- `targetColumn` – 作成される新しい列の名前。

apply(cls, *args, **kwargs)

継承元は `GlueTransform` apply。

name(cls)

継承元は `GlueTransform` name。

describeArgs(cls)

継承元は `GlueTransform` describeArgs。

describeReturn(cls)

継承元は `GlueTransform` describeReturn。

describeTransform(cls)

継承元は `GlueTransform` describeTransform。

describeErrors(cls)

継承元は `GlueTransform` describeErrors。

describe(cls)

継承元は `GlueTransform` 説明。

データ統合変換

AWS Glue 4.0 以降では、`key: --enable-glue-di-transforms, value: true` を使用してジョブ引数を作成または更新します。

ジョブスクリプトの例：

```
from pyspark.context import SparkContext
```

```
from awsgluedi.transforms import *
sc = SparkContext()

input_df = spark.createDataFrame(
    [(5,), (0,), (-1,), (2,), (None,)],
    ["source_column"],
)

try:
    df_output = math_functions.IsEven.apply(
        data_frame=input_df,
        spark_context=sc,
        source_column="source_column",
        target_column="target_column",
        value=None,
        true_string="Even",
        false_string="Not even",
    )
    df_output.show()
except:
    print("Unexpected Error happened ")
    raise
```

ノートブックを使用したセッションの例

```
%idle_timeout 2880
%glue_version 4.0
%worker_type G.1X
%number_of_workers 5
%region eu-west-1
```

```
%%configure
{
    "--enable-glue-di-transforms": "true"
}
```

```
from pyspark.context import SparkContext
from awsgluedi.transforms import *

sc = SparkContext()
```

```
input_df = spark.createDataFrame(
    [(5,), (0,), (-1,), (2,), (None,)],
    ["source_column"],
)

try:
    df_output = math_functions.IsEven.apply(
        data_frame=input_df,
        spark_context=sc,
        source_column="source_column",
        target_column="target_column",
        value=None,
        true_string="Even",
        false_string="Not even",
    )
    df_output.show()
except:
    print("Unexpected Error happened ")
    raise
```

AWS CLI を使用したセッションの例

```
aws glue create-session --default-arguments "--enable-glue-di-transforms=true"
```

DI トランスフォーム:

- FlagDuplicatesInColumn class
- FormatPhoneNumber class
- FormatCase class
- FillWithMode class
- FlagDuplicateRows class
- RemoveDuplicates class
- MonthName class
- IsEven クラス
- CryptographicHash クラス
- Decrypt クラス
- 暗号化クラス

- IntToIp class
- IpToInt class

Maven: Spark アプリケーションへのプラグインのバンドル

Spark アプリケーションをローカルで開発する際、Maven pom.xml に変換依存関係を追加することで、Spark アプリケーションと Spark ディストリビューション (バージョン 3.3) にプラグインをバンドルできます。

```
<repositories>
   ...
    <repository>
        <id>aws-glue-etl-artifacts</id>
        <url>https://aws-glue-etl-artifacts.s3.amazonaws.com/release/ </url>
    </repository>
</repositories>
...
<dependency>
    <groupId>com.amazonaws</groupId>
    <artifactId>AWSGlueTransforms</artifactId>
    <version>4.0.0</version>
</dependency>
```

別の方法として、以下のように AWS Glue Maven アーティファクトからバイナリを直接ダウンロードして、Spark アプリケーションにインクルードすることも可能です。

```
#!/bin/bash
sudo wget -v https://aws-glue-etl-artifacts.s3.amazonaws.com/release/com/amazonaws/
AWSGlueTransforms/4.0.0/AWSGlueTransforms-4.0.0.jar -P /usr/lib/spark/jars/
```

Scala での AWS Glue ETL スクリプトのプログラミング

AWS Glue 用の Scala コード例とユーティリティは、GitHub ウェブサイトの AWS Glueサンプルリポジトリで見つかります。

AWS Glue では、抽出、変換、ロード (ETL) ジョブを実行する PySpark Scala 言語のスクリプトのために、拡張機能がサポートされています。以下のセクションでは、AWS Glue Scala ライブラリと AWS Glue API を ETL スクリプトで使用する方法について説明します。また、ライブラリのリファレンス資料を提供します。

目次

- Scala を使用した AWS Glue ETL スクリプトのプログラミング
  - 開発エンドポイントでの Jupyter Notebook を使用した Scala ETL プログラムのテスト
  - Scala REPL での Scala ETL プログラムのテスト
- Scala スクリプトの例 - ストリーミング ETL
- AWS Glue Scala ライブラリの API
  - com.amazonaws.services.glue
  - com.amazonaws.services.glue.ml
  - com.amazonaws.services.glue.dq
  - com.amazonaws.services.glue.types
  - com.amazonaws.services.glue.util
  - AWS Glue Scala ChoiceOption API
    - ChoiceOption 特性
    - ChoiceOption オブジェクト
      - Def apply
    - ケースクラス ChoiceOptionWithResolver
    - ケースクラス MatchCatalogSchemaChoiceOption
  - Abstract DataSink クラス
    - Def writeDynamicFrame
    - Def pyWriteDynamicFrame
    - Def writeDataFrame
    - Def pyWriteDataFrame
    - Def setCatalogInfo
    - Def supportsFormat
    - Def setFormat
    - Def withFormat
    - Def setAccumulableSize
    - Def getOutputErrorRecordsAccumulable
    - Def errorsAsDynamicFrame
    - DataSink オブジェクト

        - Def recordMetrics
- AWS Glue Scala DataSource トレイト
- AWS Glue Scala DynamicFrame API
    - AWS Glue Scala DynamicFrame クラス
        - Val errorsCount
        - Def applyMapping
        - Def assertErrorThreshold
        - Def count
        - Def dropField
        - Def dropFields
        - Def dropNulls
        - Def errorsAsDynamicFrame
        - Def filter
        - Def getName
        - Def getNumPartitions
        - Def getSchemaIfComputed
        - Def isSchemaComputed
        - Def javaToPython
        - Def join
        - Def map
        - Def mergeDynamicFrames
        - Def printSchema
        - Def recomputeSchema
        - Def relationalize
        - Def renameField
        - Def repartition
        - Def resolveChoice
        - Def schema
        - Def selectField
        - Def selectFields

        - Def show
        - Def simplifyDDBJson
        - Def spigot
        - Def splitFields
        - Def splitRows
        - Def stageErrorsCount
        - Def toDF
        - Def unbox
        - Def unnest
        - Def unnestDDBJson
        - Def withFrameSchema
        - Def withName
        - Def withTransformationContext
    - DynamicFrame オブジェクト
        - Def apply
        - Def emptyDynamicFrame
        - Def fromPythonRDD
        - Def ignoreErrors
        - Def inlineErrors
        - Def newFrameWithErrors
- AWS Glue Scala DynamicRecord クラス
    - def addField
    - def dropField
    - def setError
    - def isError
    - def getError
    - def clearError
    - def write
    - def readFields
    - def clone

    - def schema
    - def getRoot
    - def toJson
    - def getFieldNode
    - def getField
    - def hashCode
    - def equals
    - DynamicRecord オブジェクト
        - def apply
    - RecordTraverser 特性
- AWS Glue Scala GlueContext API
    - def addIngestionTimeColumns
    - def createDataFrameFromOptions
    - forEachBatch
    - def getCatalogSink
    - def getCatalogSource
    - def getJDBCSink
    - def getSink
    - def getSinkWithFormat
    - def getSource
    - def getSourceWithFormat
    - def getSparkSession
    - def startTransaction
    - def commitTrac
    - def cancelTransaction
    - def this
    - def this
    - def this
- MappingSpec
    - MappingSpec ケースクラス

    - MappingSpec オブジェクト
    - Val orderingByTarget
    - Def apply
    - Def apply
    - Def apply
- AWS Glue Scala ResolveSpec API
    - ResolveSpec オブジェクト
        - def
        - def
    - ResolveSpec ケースクラス
        - ResolveSpec def メソッド
- AWS Glue Scala ArrayNode API
    - ArrayNode ケースクラス
        - ArrayNode def メソッド
- AWS Glue Scala BinaryNode API
    - BinaryNode ケースクラス
        - BinaryNode val フィールド
        - BinaryNode def メソッド
- AWS Glue Scala BooleanNode API
    - BooleanNode ケースクラス
        - BooleanNode val フィールド
        - BooleanNode def メソッド
- AWS Glue Scala ByteNode API
    - ByteNode ケースクラス
        - ByteNode val フィールド
        - ByteNode def メソッド
- AWS Glue Scala DateNode API
    - DateNode ケースクラス
        - DateNode val フィールド
        - DateNode def メソッド

- AWS Glue Scala DecimalNode API
  - DecimalNode ケースクラス
    - DecimalNode val フィールド
    - DecimalNode def メソッド
- AWS Glue Scala DoubleNode API
  - DoubleNode ケースクラス
    - DoubleNode val フィールド
    - DoubleNode def メソッド
- AWS Glue Scala DynamicNode API
  - DynamicNode クラス
    - DynamicNode def メソッド
  - DynamicNode オブジェクト
    - DynamicNode def メソッド
- EvaluateDataQuality クラス
  - def apply
  - 例
- AWS Glue Scala FloatNode API
  - FloatNode ケースクラス
    - FloatNode val フィールド
    - FloatNode def メソッド
- FillMissingValues クラス
  - def apply
- FindMatches クラス
  - def apply
- FindIncrementalMatches クラス
  - def apply
- AWS Glue Scala IntegerNode API
  - IntegerNode ケースクラス
    - IntegerNode val フィールド
    - IntegerNode def メソッド

- AWS Glue Scala LongNode API
  - LongNode ケースクラス
    - LongNode val フィールド
    - LongNode def メソッド
- AWS Glue Scala MapLikeNode API
  - MapLikeNode クラス
    - MapLikeNode def メソッド
- AWS Glue Scala MapNode API
  - MapNode ケースクラス
    - MapNode def メソッド
- AWS Glue Scala NullNode API
  - NullNode クラス
  - NullNode ケースオブジェクト
- AWS Glue Scala ObjectNode API
  - ObjectNode オブジェクト
    - ObjectNode def メソッド
  - ObjectNode ケースクラス
    - ObjectNode def メソッド
- AWS Glue Scala ScalarNode API
  - ScalarNode クラス
    - ScalarNode def メソッド
  - ScalarNode オブジェクト
    - ScalarNode def メソッド
- AWS Glue Scala ShortNode API
  - ShortNode ケースクラス
    - ShortNode val フィールド
    - ShortNode def メソッド
- AWS Glue Scala StringNode API
  - StringNode ケースクラス
    - StringNode val フィールド

        - StringNode def メソッド
- AWS Glue Scala TimestampNode API
    - TimestampNode ケースクラス
        - TimestampNode val フィールド
        - TimestampNode def メソッド
- AWS Glue Scala GlueArgParser API
    - GlueArgParser オブジェクト
        - GlueArgParser def メソッド
- AWS Glue Scala Job API
    - Job オブジェクト
        - Job def メソッド

Scala を使用した AWS Glue ETL スクリプトのプログラミング

AWS Glue コンソールを使用して Scala ETL (抽出、変換、ロード) プログラムを自動的に生成し、必要に応じて変更してからジョブに割り当てることができます。または、独自のプログラムを最初から記述することもできます。詳細については、「AWS Glue で Spark ジョブに関するジョブプロパティの構成」を参照してください。次に AWS Glue はサーバーで Scala プログラムをコンパイルし、その後、関連付けられているジョブを実行します。

プログラムをエラーなしでコンパイルして正常に実行するには、ジョブで実行する前にプログラムを REPL (Read-Eval-Print Loop) や Jupyter Notebook の開発エンドポイントにロードし、テストすることが重要です。コンパイルプロセスはサーバーで実行されるため、そこで問題が発生してもよく確認できません。

開発エンドポイントでの Jupyter Notebook を使用した Scala ETL プログラムのテスト

AWS Glue 開発エンドポイントで Scala プログラムをテストするには、「開発エンドポイントの追加」の説明に従って開発エンドポイントを設定します。

次に、コンピュータでローカルに実行されているか、Amazon EC2 ノートブックサーバーでリモートに実行されている Jupyter Notebook に開発エンドポイントを接続します。Jupyter Notebook のローカルバージョンをインストールするには、「チュートリアル： JupyterLab の Jupyter Notebook」の手順に従います。

Scala コードの実行とノートブックでの PySpark コードの実行が唯一異なる点は、ノートブックでは以下のコードを使用して各パラグラフを開始する必要があることです。

```
%spark
```

これにより、ノートブックサーバーがデフォルトで Spark インタープリタの PySpark フレーバーになるのを防止できます。

Scala REPL での Scala ETL プログラムのテスト

AWS Glue Scala REPL を使用して開発エンドポイントで Scala プログラムをテストできます。「チュートリアル: SageMaker AI ノートブックを使用する」の手順に従います。ただし、SSH-to-REPL コマンドの最後では、`-t gluepyspark` を `-t glue-spark-shell` に置き換えます。これにより、AWS Glue Scala REPL が呼び出されます。

完了後に REPL を閉じるには、「`sys.exit`」と入力します。

Scala スクリプトの例 - ストリーミング ETL

Example

次のスクリプト例では、Amazon Kinesis Data Streams に接続し、Data Catalog のスキーマを使用してデータストリームを解析した上で、そのストリームを Amazon S3 にある静的データセットに結合します。結合された結果は Parquet 形式で Amazon S3 に出力されます。

```
// This script connects to an Amazon Kinesis stream, uses a schema from the data
 catalog to parse the stream,
// joins the stream to a static dataset on Amazon S3, and outputs the joined results to
 Amazon S3 in parquet format.
import com.amazonaws.services.glue.GlueContext
import com.amazonaws.services.glue.util.GlueArgParser
import com.amazonaws.services.glue.util.Job
import java.util.Calendar
import org.apache.spark.SparkContext
import org.apache.spark.sql.Dataset
import org.apache.spark.sql.Row
import org.apache.spark.sql.SaveMode
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.functions.from_json
import org.apache.spark.sql.streaming.Trigger
import scala.collection.JavaConverters._

object streamJoiner {
  def main(sysArgs: Array[String]) {
    val spark: SparkContext = new SparkContext()
```

```
    val glueContext: GlueContext = new GlueContext(spark)
    val sparkSession: SparkSession = glueContext.getSparkSession
    import sparkSession.implicits._
    // @params: [JOB_NAME]
    val args = GlueArgParser.getResolvedOptions(sysArgs, Seq("JOB_NAME").toArray)
    Job.init(args("JOB_NAME"), glueContext, args.asJava)

    val staticData = sparkSession.read          // read() returns type DataFrameReader
      .format("csv")
      .option("header", "true")
      .load("s3://amzn-s3-demo-bucket/inputs/productsStatic.csv")  // load() returns a
DataFrame

    val datasource0 = sparkSession.readStream   // readstream() returns type
DataStreamReader
      .format("kinesis")
      .option("streamName", "stream-join-demo")
      .option("endpointUrl", "https://kinesis.us-east-1.amazonaws.com")
      .option("startingPosition", "TRIM_HORIZON")
      .load                                     // load() returns a DataFrame

    val selectfields1 = datasource0.select(from_json($"data".cast("string"),
glueContext.getCatalogSchemaAsSparkSchema("stream-demos", "stream-join-demo2")) as
"data").select("data.*")

    val datasink2 = selectfields1.writeStream.foreachBatch { (dataFrame: Dataset[Row],
batchId: Long) => {   //foreachBatch() returns type DataStreamWriter
      val joined = dataFrame.join(staticData, "product_id")
      val year: Int = Calendar.getInstance().get(Calendar.YEAR)
      val month :Int = Calendar.getInstance().get(Calendar.MONTH) + 1
      val day: Int = Calendar.getInstance().get(Calendar.DATE)
      val hour: Int = Calendar.getInstance().get(Calendar.HOUR_OF_DAY)

      if (dataFrame.count() > 0) {
        joined.write                            // joined.write returns type
DataFrameWriter
          .mode(SaveMode.Append)
          .format("parquet")
          .option("quote", " ")
          .save("s3://amzn-s3-demo-bucket/output/" + "/year=" + "%04d".format(year)
+ "/month=" + "%02d".format(month) + "/day=" + "%02d".format(day) + "/hour=" +
"%02d".format(hour) + "/")
      }
    }
```

```
    }  // end foreachBatch()
      .trigger(Trigger.ProcessingTime("100 seconds"))
      .option("checkpointLocation", "s3://amzn-s3-demo-bucket/checkpoint/")
      .start().awaitTermination()              // start() returns type StreamingQuery
    Job.commit()
  }
}
```

AWS Glue Scala ライブラリの API

AWS Glue では、抽出、変換、ロード (ETL) ジョブのスクリプト化に使用する PySpark Scala 言語の拡張機能がサポートされています。以下のセクションでは、AWS Glue Scala ライブラリの API について説明します。

com.amazonaws.services.glue

Scala ライブラリの com.amazonaws.services.glueAWS Glue パッケージには、以下の API が含まれています。

- ChoiceOption
- DataSink
- DataSource 特性
- DynamicFrame
- DynamicRecord
- GlueContext
- MappingSpec
- ResolveSpec

com.amazonaws.services.glue.ml

AWS Glue Scala ライブラリの com.amazonaws.services.glue.ml パッケージには、以下の API が含まれています。

- FillMissingValues
- FindIncrementalMatches
- FindMatches

com.amazonaws.services.glue.dq

AWS Glue Scala ライブラリの com.amazonaws.services.glue.dq パッケージには、以下の API が含まれています。

- EvaluateDataQuality

com.amazonaws.services.glue.types

Scala ライブラリの com.amazonaws.services.glue.typesAWS Glue パッケージには、以下の API が含まれています。

- ArrayNode
- BinaryNode
- BooleanNode
- ByteNode
- DateNode
- DecimalNode
- DoubleNode
- DynamicNode
- FloatNode
- IntegerNode
- LongNode
- MapLikeNode
- MapNode
- NullNode
- ObjectNode
- ScalarNode
- ShortNode
- StringNode
- TimestampNode

com.amazonaws.services.glue.util

Scala ライブラリの com.amazonaws.services.glue.utilAWS Glue パッケージには、以下の API が含まれています。

- GlueArgParser
- ジョブ

AWS Glue Scala ChoiceOption API

トピック

- ChoiceOption 特性
- ChoiceOption オブジェクト
- ケースクラス ChoiceOptionWithResolver
- ケースクラス MatchCatalogSchemaChoiceOption

パッケージ: com.amazonaws.services.glue

ChoiceOption 特性

```
trait ChoiceOption extends Serializable
```

ChoiceOption オブジェクト

ChoiceOption

```
object ChoiceOption
```

DynamicFrame のすべての ChoiceType ノードに適用可能な選択肢を解決するための一般的な戦略。

- val CAST
- val MAKE_COLS
- val MAKE_STRUCT
- val MATCH_CATALOG

- `val PROJECT`

Def apply

```
def apply(choice: String): ChoiceOption
```

ケースクラス ChoiceOptionWithResolver

```
case class ChoiceOptionWithResolver(name: String, choiceResolver: ChoiceResolver)
 extends ChoiceOption {}
```

ケースクラス MatchCatalogSchemaChoiceOption

```
case class MatchCatalogSchemaChoiceOption() extends ChoiceOption {}
```

Abstract DataSink クラス

トピック

- Def writeDynamicFrame
- Def pyWriteDynamicFrame
- Def writeDataFrame
- Def pyWriteDataFrame
- Def setCatalogInfo
- Def supportsFormat
- Def setFormat
- Def withFormat
- Def setAccumulableSize
- Def getOutputErrorRecordsAccumulable
- Def errorsAsDynamicFrame
- DataSink オブジェクト

パッケージ: com.amazonaws.services.glue

```
abstract class DataSink
```

ライターは DataSource に似ています。DataSink は、DynamicFrame が書き込まれる送信先と形式をカプセル化します。

Def writeDynamicFrame

```
def writeDynamicFrame( frame : DynamicFrame,
                       callSite : CallSite = CallSite("Not provided", "")
                     ) : DynamicFrame
```

Def pyWriteDynamicFrame

```
def pyWriteDynamicFrame( frame : DynamicFrame,
                         site : String = "Not provided",
                         info : String = "" )
```

Def writeDataFrame

```
def writeDataFrame(frame: DataFrame,
                   glueContext: GlueContext,
                   callSite: CallSite = CallSite("Not provided", "")
                   ): DataFrame
```

Def pyWriteDataFrame

```
def pyWriteDataFrame(frame: DataFrame,
                     glueContext: GlueContext,
                     site: String = "Not provided",
                     info: String = ""
                     ): DataFrame
```

Def setCatalogInfo

```
def setCatalogInfo(catalogDatabase: String,
                   catalogTableName : String,
```

```
                    catalogId : String = "")
```

Def supportsFormat

```
def supportsFormat( format : String ) : Boolean
```

Def setFormat

```
def setFormat( format : String,
               options : JsonOptions
             ) : Unit
```

Def withFormat

```
def withFormat( format : String,
                options : JsonOptions = JsonOptions.empty
              ) : DataSink
```

Def setAccumulableSize

```
def setAccumulableSize( size : Int ) : Unit
```

Def getOutputErrorRecordsAccumulable

```
def getOutputErrorRecordsAccumulable : Accumulable[List[OutputError], OutputError]
```

Def errorsAsDynamicFrame

```
def errorsAsDynamicFrame : DynamicFrame
```

DataSink オブジェクト

```
object DataSink
```

Def recordMetrics

```
def recordMetrics( frame : DynamicFrame,
                   ctxt : String
                 ) : DynamicFrame
```

AWS Glue Scala DataSource トレイト

パッケージ: com.amazonaws.services.glue

DynamicFrame を生成するための高水準インターフェイス。

```
trait DataSource {

  def getDynamicFrame : DynamicFrame

  def getDynamicFrame( minPartitions : Int,
                       targetPartitions : Int
                     ) : DynamicFrame
  def getDataFrame : DataFrame

  /** @param num: the number of records for sampling.
    * @param options: optional parameters to control sampling behavior. Current
 available parameter for Amazon S3 sources in options:
    *  1. maxSamplePartitions: the maximum number of partitions the sampling will
 read.
    *  2. maxSampleFilesPerPartition: the maximum number of files the sampling will
 read in one partition.
    */
  def getSampleDynamicFrame(num:Int, options: JsonOptions = JsonOptions.empty):
 DynamicFrame

  def glueContext : GlueContext

  def setFormat( format : String,
                 options : String
               ) : Unit

  def setFormat( format : String,
                 options : JsonOptions
               ) : Unit

  def supportsFormat( format : String ) : Boolean
```

```
  def withFormat( format : String,
                  options : JsonOptions = JsonOptions.empty
                ) : DataSource
}
```

AWS Glue Scala DynamicFrame API

パッケージ: com.amazonaws.services.glue

目次

- AWS Glue Scala DynamicFrame クラス
  - Val errorsCount
  - Def applyMapping
  - Def assertErrorThreshold
  - Def count
  - Def dropField
  - Def dropFields
  - Def dropNulls
  - Def errorsAsDynamicFrame
  - Def filter
  - Def getName
  - Def getNumPartitions
  - Def getSchemaIfComputed
  - Def isSchemaComputed
  - Def javaToPython
  - Def join
  - Def map
  - Def mergeDynamicFrames
  - Def printSchema
  - Def recomputeSchema
  - Def relationalize
  - Def renameField
  - Def repartition

- Def resolveChoice
- Def schema
- Def selectField
- Def selectFields
- Def show
- Def simplifyDDBJson
- Def spigot
- Def splitFields
- Def splitRows
- Def stageErrorsCount
- Def toDF
- Def unbox
- Def unnest
- Def unnestDDBJson
- Def withFrameSchema
- Def withName
- Def withTransformationContext

- DynamicFrame オブジェクト
  - Def apply
  - Def emptyDynamicFrame
  - Def fromPythonRDD
  - Def ignoreErrors
  - Def inlineErrors
  - Def newFrameWithErrors

AWS Glue Scala DynamicFrame クラス

パッケージ: com.amazonaws.services.glue

```
class DynamicFrame extends Serializable with Logging  (
           val glueContext : GlueContext,
           _records : RDD[DynamicRecord],
           val name : String = s"",
```

```
            val transformationContext : String = DynamicFrame.UNDEFINED,
            callSite : CallSite = CallSite("Not provided", ""),
            stageThreshold : Long = 0,
            totalThreshold : Long = 0,
            prevErrors : => Long = 0,
            errorExpr : => Unit = {} )
```

DynamicFrame は、自己記述型の DynamicRecord オブジェクトの分散コレクションです。

DynamicFrame は、ETL (抽出、変換、ロード) オペレーションの柔軟なデータモデルを提供するように設計されています。これらのオブジェクトを作成するのにスキーマは必要なく、乱雑または不整合な値や型を持つデータの読み取りと変換に使用できます。スキーマは、スキーマを必要とするオペレーションでオンデマンドで計算できます。

DynamicFrame は、データクリーニングと ETL 用の広範な変換を提供します。また、既存のコードと統合するための SparkSQL DataFrames との相互変換や、DataFrames が提供する多くの分析オペレーションをサポートしています。

以下のパラメータは、DynamicFrame を生成する AWS Glue 変換の多くで共有されます。

- transformationContext - この DynamicFrame の識別子。実行間で保持されるジョブのブックマーク状態のキーとして、transformationContext が使用されます。
- callSite – エラーレポートのコンテキスト情報を提供します。これらの値は、Python から呼び出すときに、自動的に設定されます。
- stageThreshold - この DynamicFrame の計算から例外がスローされるまでのエラーレコードの最大許容数。以前の DynamicFrame にあるレコードは除きます。
- totalThreshold - 例外がスローされるまでの合計エラーレコードの最大数。以前のフレームのレコードも含みます。

Val errorsCount

```
val errorsCount
```

この DynamicFrame のエラーレコードの数。以前のオペレーションのエラーも含みます。

Def applyMapping

```
def applyMapping( mappings : Seq[Product4[String, String, String, String]],
                  caseSensitive : Boolean = true,
```

```
    transformationContext : String = "",
    callSite : CallSite = CallSite("Not provided", ""),
    stageThreshold : Long = 0,
    totalThreshold : Long = 0
  ) : DynamicFrame
```

- mappings - 新しい DynamicFrame を生成するための一連のマッピング。
- caseSensitive – ソース列で大文字と小文字を区別して扱うかどうかを指定。このパラメータを false に設定すると、AWS Glue Data Catalog のように大文字と小文字を区別しないストアと統合する場合に役立ちます。

マッピングのシーケンスに基づく選択列、プロジェクト列、およびキャスト列。

各マッピングは、ソース列/タイプとターゲット列/タイプで構成されます。マッピングは、4 タプル (source_path、source_type、 target_path、target_type) として指定するか、同じ情報を含む MappingSpec オブジェクトとして指定できます。

マッピングは、シンプルなプロジェクションやキャストに使用できるだけなく、パスのコンポーネントを "." (ピリオド) で区切ることでフィールドのネストまたはネスト解除にも使用できます。

例えば、以下のスキーマを含む DynamicFrame があるとします。

```
{{{
  root
  |-- name: string
  |-- age: int
  |-- address: struct
  |    |-- state: string
  |    |-- zip: int
}}}
```

以下の呼び出しを行って、state フィールドと zip フィールドをネスト解除できます。

```
{{{
  df.applyMapping(
   Seq(("name", "string", "name", "string"),
       ("age", "int", "age", "int"),
       ("address.state", "string", "state", "string"),
       ("address.zip", "int", "zip", "int")))
}}}
```

その結果、スキーマは以下のようになります。

```
{{{
  root
  |-- name: string
  |-- age: int
  |-- state: string
  |-- zip: int
}}}
```

applyMapping を使用して列を再ネストすることもできます。以下の例では、前の変換を反転し、ターゲットに address という名前の構造体を作成します。

```
{{{
  df.applyMapping(
   Seq(("name", "string", "name", "string"),
       ("age", "int", "age", "int"),
       ("state", "string", "address.state", "string"),
       ("zip", "int", "address.zip", "int")))
}}}
```

"." (ピリオド) 文字を含むフィールド名は、バッククォート (` `) で囲むことで使用できます。

> **Note**
>
> 現在、applyMapping メソッドを使用して、配列の下にネストされた列をマップすることはできません。

Def assertErrorThreshold

```
def assertErrorThreshold : Unit
```

このアクションでは、計算を適用し、エラーレコード数が stageThreshold と totalThreshold を下回っていることを確認します。いずれかの条件が失敗すると、例外をスローします。

Def count

```
lazy
def count
```

この DynamicFrame の要素数を返します。

Def dropField

```
def dropField( path : String,
               transformationContext : String = "",
               callSite : CallSite = CallSite("Not provided", ""),
               stageThreshold : Long = 0,
               totalThreshold : Long = 0
             ) : DynamicFrame
```

指定した列が削除された新しい DynamicFrame を返します。

Def dropFields

```
def dropFields( fieldNames : Seq[String],   // The column names to drop.
                transformationContext : String = "",
                callSite : CallSite = CallSite("Not provided", ""),
                stageThreshold : Long = 0,
                totalThreshold : Long = 0
              ) : DynamicFrame
```

指定した列が削除された新しい DynamicFrame を返します。

このメソッドを使用して、ネストされた列 (配列内の列など) を削除できますが、特定の配列要素を削除することはできません。

Def dropNulls

```
def dropNulls( transformationContext : String = "",
               callSite : CallSite = CallSite("Not provided", ""),
               stageThreshold : Long = 0,
               totalThreshold : Long = 0 )
```

すべての NULL 列が削除された新しい DynamicFrame を返します。

> **Note**
>
> NullType タイプの列のみが削除されます。他の列にある個別の null 値は削除または変更されません。

Def errorsAsDynamicFrame

```
def errorsAsDynamicFrame
```

この DynamicFrame からのエラーレコードを含む新しい DynamicFrame を返します。

Def filter

```
def filter( f : DynamicRecord => Boolean,
            errorMsg : String = "",
            transformationContext : String = "",
            callSite : CallSite = CallSite("Not provided"),
            stageThreshold : Long = 0,
            totalThreshold : Long = 0
          ) : DynamicFrame
```

"f" 関数が true を返すレコードのみを含む新しい DynamicFrame を生成します。フィルター関数 "f" は、入力レコードを変更しないものとします。

Def getName

```
def getName : String
```

この DynamicFrame の名前を返します。

Def getNumPartitions

```
def getNumPartitions
```

この DynamicFrame のパーティション数を返します。

Def getSchemaIfComputed

```
def getSchemaIfComputed : Option[Schema]
```

計算済みのスキーマを返します。スキーマが計算済みでない場合は、データをスキャンしません。

Def isSchemaComputed

```
def isSchemaComputed : Boolean
```

この DynamicFrame に対してスキーマが計算された場合は true を返し、それ以外の場合は false を返します。このメソッドが false を返した場合、schema メソッドを呼び出すには、この DynamicFrame のレコードを再び渡す必要があります。

Def javaToPython

```
def javaToPython : JavaRDD[Array[Byte]]
```

Def join

```
def join( keys1 : Seq[String],
          keys2 : Seq[String],
          frame2 : DynamicFrame,
          transformationContext : String = "",
          callSite : CallSite = CallSite("Not provided", ""),
          stageThreshold : Long = 0,
          totalThreshold : Long = 0
        ) : DynamicFrame
```

- keys1 - 結合に使用するこの DynamicFrame の列。
- keys2 – 結合に使用する frame2 の列。keys1 と同じ長さにする必要があります。
- frame2 - 結合の対象になる DynamicFrame。

指定したキーを使用して frame2 との等結合を行った結果を返します。

Def map

```
def map( f : DynamicRecord => DynamicRecord,
         errorMsg : String = "",
         transformationContext : String = "",
         callSite : CallSite = CallSite("Not provided", ""),
         stageThreshold : Long = 0,
         totalThreshold : Long = 0
       ) : DynamicFrame
```

指定した関数 "f" をこの DynamicFrame の各レコードに適用することで生成された新しい DynamicFrame を返します。

このメソッドは、指定した関数を適用する前に各レコードをコピーするため、レコードを安全に変更できます。特定のレコードでマッピング関数から例外がスローされた場合、そのレコードはエラーとしてマークされ、スタックトレースがエラーレコードの列として保存されます。

Def mergeDynamicFrames

```
def mergeDynamicFrames( stageDynamicFrame: DynamicFrame,  primaryKeys: Seq[String],
 transformationContext: String = "",
                         options: JsonOptions = JsonOptions.empty, callSite: CallSite =
 CallSite("Not provided"),
                         stageThreshold: Long = 0, totalThreshold: Long = 0):
 DynamicFrame
```

- stageDynamicFrame – マージするステージング DynamicFrame。
- primaryKeys – ソースおよびステージング DynamicFrame からのレコードを照合するプライマリキーフィールドのリスト。
- transformationContext – 現在の変換に関するメタデータを取得するために使用される一意の文字列 (オプション)。
- options – この変換に関する追加情報を提供する JSON の名前と値のペアを示す文字列。
- callSite – エラーレポートのコンテキスト情報を提供するために使用します。
- stageThreshold — A Long。指定された変換で処理がエラーアウトする必要があるエラーの数。
- totalThreshold — A Long。この変換までで処理がエラーアウトする必要があるエラーの合計数。

レコードを識別するために、この DynamicFrame を指定されたプライマリキーに基づくステージング DynamicFrame とマージします。重複レコード (同じプライマリキーを持つレコード) は重複除外されません。ステージングフレームに一致するレコードがない場合、すべてのレコード (重複を含む) がソースから保持されます。ステージングフレームに一致するレコードがある場合、ステージングフレームのレコードによって、AWS Glue のソースのレコードが上書きされます。

以下の場合、返される DynamicFrame にはレコード A が含まれます。

1. A がソースフレームとステージングフレームの両方に存在する場合、ステージングフレームの A が返されます。
2. A がソーステーブルにあり、A.primaryKeys が stagingDynamicFrame にない場合 (これは、ステージングテーブルで A が更新されていないことを意味します)。

ソースフレームとステージングフレームが、同じスキーマを持つ必要はありません。

Example

```
val mergedFrame: DynamicFrame = srcFrame.mergeDynamicFrames(stageFrame, Seq("id1",
 "id2"))
```

Def printSchema

```
def printSchema : Unit
```

この DynamicFrame のスキーマを、人間が判読できる形式で stdout に出力します。

Def recomputeSchema

```
def recomputeSchema : Schema
```

スキーマの再計算を強制します。これには、データのスキャンが必要ですが、現在のスキーマの一部のフィールドがデータにない場合、スキーマが "強化" される場合があります。

再計算されたスキーマを返します。

Def relationalize

```
def relationalize( rootTableName : String,
                   stagingPath : String,
                   options : JsonOptions = JsonOptions.empty,
                   transformationContext : String = "",
                   callSite : CallSite = CallSite("Not provided"),
                   stageThreshold : Long = 0,
                   totalThreshold : Long = 0
                 ) : Seq[DynamicFrame]
```

- rootTableName – 出力の基本 DynamicFrame に使用する名前。ピボット配列によって作成される DynamicFrame は、この名前をプレフィックスとして使用します。
- stagingPath – 中間データを書き込む Amazon Simple Storage Service (Amazon S3) のパス。
- options – Relationalize のオプションと設定。現在使用されていません。

すべてのネストされた構造をフラット化し、配列を個別のテーブルにピボットします。

このオペレーションでは、リレーショナルデータベースに取り込むための深くネストされたデータを準備できます。ネストされた構造体は、ネスト解除 変換と同じ方法でフラット化されます。さらに、配列は個別のテーブルにピボットされ、各配列要素が行になります。例えば、以下のデータを含む DynamicFrame があるとします。

```
 {"name": "Nancy", "age": 47, "friends": ["Fred", "Lakshmi"]}
 {"name": "Stephanie", "age": 28, "friends": ["Yao", "Phil", "Alvin"]}
 {"name": "Nathan", "age": 54, "friends": ["Nicolai", "Karen"]}
```

以下のコードを実行します。

```
{{{
  df.relationalize("people", "s3:/my_bucket/my_path", JsonOptions.empty)
}}}
```

これにより、2 つのテーブルが生成されます。最初のテーブルは "people" という名前で、内容は以下のとおりです。

```
{{{
  {"name": "Nancy", "age": 47, "friends": 1}
  {"name": "Stephanie", "age": 28, "friends": 2}
  {"name": "Nathan", "age": 54, "friends": 3)
}}}
```

ここで、友人の配列は自動生成された結合キーに置き換えられています。別のテーブルは people.friends という名前で、以下の内容で作成されます。

```
{{{
  {"id": 1, "index": 0, "val": "Fred"}
  {"id": 1, "index": 1, "val": "Lakshmi"}
  {"id": 2, "index": 0, "val": "Yao"}
  {"id": 2, "index": 1, "val": "Phil"}
  {"id": 2, "index": 2, "val": "Alvin"}
  {"id": 3, "index": 0, "val": "Nicolai"}
  {"id": 3, "index": 1, "val": "Karen"}
}}}
```

このテーブルで、"id" は配列要素の元のレコードを識別する結合キーです。"index" は元の配列内の位置を参照します。"val" は実際の配列エントリです。

relationalize メソッドは、このプロセスをすべての配列に再帰的に適用することで作成した DynamicFrame のシーケンスを返します。

> **Note**
>
> AWS Glue ライブラリは、新しいテーブルの結合キーを自動的に生成します。ジョブの実行間で結合キーが必ず一意になるように、ジョブのブックマークを有効にする必要があります。

Def renameField

```
def renameField( oldName : String,
                 newName : String,
                 transformationContext : String = "",
                 callSite : CallSite = CallSite("Not provided", ""),
                 stageThreshold : Long = 0,
                 totalThreshold : Long = 0
               ) : DynamicFrame
```

- oldName – 列の元の名前。
- newName – 列の新しい名前。

指定したフィールドの名前が変更された新しい DynamicFrame を返します。

このメソッドを使用して、ネストされたフィールドの名前を変更できます。例えば、以下のコードはアドレス構造体内の state の名前を state_code に変更します。

```
{{{
  df.renameField("address.state", "address.state_code")
}}}
```

Def repartition

```
def repartition( numPartitions : Int,
                 transformationContext : String = "",
                 callSite : CallSite = CallSite("Not provided", ""),
                 stageThreshold : Long = 0,
                 totalThreshold : Long = 0
```

```
                ) : DynamicFrame
```

numPartitions パーティションを含む新しい DynamicFrame を返します。

Def resolveChoice

```
def resolveChoice( specs : Seq[Product2[String, String]] = Seq.empty[ResolveSpec],
                   choiceOption : Option[ChoiceOption] = None,
                   database : Option[String] = None,
                   tableName : Option[String] = None,
                   transformationContext : String = "",
                   callSite : CallSite = CallSite("Not provided", ""),
                   stageThreshold : Long = 0,
                   totalThreshold : Long = 0
                 ) : DynamicFrame
```

- choiceOption - 仕様シーケンスに列挙されていない ChoiceType 列に適用するアクション。
- database – match_catalog アクションで使用する Data Catalog データベース。
- tableName – match_catalog アクションで使用する Data Catalog テーブル。

1 つ以上の ChoiceType をより限定されたタイプに置き換えて新しい DynamicFrame を返します。

resolveChoice を使用するには 2 つの方法があります。最初の方法では、特定の列のシーケンスと解決方法を指定します。これらは (列、アクション) ペアで構成されたタプルとして指定します。

以下のアクションを指定できます。

- cast:type – すべての値を指定した型にキャストしようとします。
- make_cols – それぞれの異なるタイプを columnName_type という名前の列に変換します。
- make_struct – 列を各区別型のキーを持つ構造体に変換します。
- project:type - 指定した型の値のみを保持します。

resolveChoice のもう 1 つのモードでは、すべての ChoiceType に対して単一の解決策を指定します。このモードは、実行前に ChoiceType の完全なリストが不明な場合に使用できます。このモードでは、上記のアクションに加えて、以下のアクションもサポートされています。

- match_catalogChoiceType – 指定したカタログテーブルの対応するタイプへの各 のキャストを試行します。

例:

user.id 列を解決するために int にキャストし、address フィールドで構造体のみを保持します。

```
{{{
  df.resolveChoice(specs = Seq(("user.id", "cast:int"), ("address", "project:struct")))
}}}
```

すべての ChoiceType を解決するために、各選択肢を別個の列に変換します。

```
{{{
  df.resolveChoice(choiceOption = Some(ChoiceOption("make_cols")))
}}}
```

すべての ChoiceType を解決するために、指定したカタログテーブルのタイプにキャストします。

```
{{{
  df.resolveChoice(choiceOption = Some(ChoiceOption("match_catalog")),
                   database = Some("my_database"),
                   tableName = Some("my_table"))
}}}
```

Def schema

```
def schema : Schema
```

この DynamicFrame のスキーマを返します。

返されたスキーマには必ず、この DynamicFrame のレコードにあるすべてのフィールドが含まれます。しかし、その他のフィールドが含まれる場合がまれにあります。ネスト解除 メソッドを使用して、この DynamicFrame のレコードに基づいてスキーマを "強化" できます。

Def selectField

```
def selectField( fieldName : String,
```

```
                transformationContext : String = "",
                callSite : CallSite = CallSite("Not provided", ""),
                stageThreshold : Long = 0,
                totalThreshold : Long = 0
              ) : DynamicFrame
```

単一のフィールドを DynamicFrame として返します。

Def selectFields

```
def selectFields( paths : Seq[String],
                  transformationContext : String = "",
                  callSite : CallSite = CallSite("Not provided", ""),
                  stageThreshold : Long = 0,
                  totalThreshold : Long = 0
                ) : DynamicFrame
```

- paths - 選択する列名のシーケンス。

指定した列を含む新しい DynamicFrame を返します。

> **Note**
>
> selectFields メソッドでは、最上位の列のみを選択できます。applyMapping メソッドでは、ネストされた列を選択できます。

Def show

```
def show( numRows : Int = 20 ) : Unit
```

- numRows - 出力する行の数。

この DynamicFrame の行を JSON 形式で出力します。

Def simplifyDDBJson

DynamoDB は AWS Glue DynamoDB エクスポートコネクタを使用してエクスポートし、特殊なネスト構造の JSON ファイルが生成されます。詳細については、「データオブジェクト」を参照して

ください。simplifyDDBJsonこのタイプのデータの DynamicFrame にあるネスト化された列を単純化し、新しい単純化された DynamicFrame を返します。リストタイプに複数タイプまたは Map タイプが含まれている場合、リストの要素は単純化されません。このメソッドは、DynamoDB エクスポート JSON 形式のデータのみをサポートします。unnest が他の種類のデータにも同様の変更を加えることを考慮してください。

```
def simplifyDDBJson() : DynamicFrame
```

このメソッドにはパラメータはありません。

入力例

DynamoDB エクスポートによって生成された次のスキーマを考えてみましょう。

```
root
|-- Item: struct
|    |-- parentMap: struct
|    |    |-- M: struct
|    |    |    |-- childMap: struct
|    |    |    |    |-- M: struct
|    |    |    |    |    |-- appName: struct
|    |    |    |    |    |    |-- S: string
|    |    |    |    |    |-- packageName: struct
|    |    |    |    |    |    |-- S: string
|    |    |    |    |    |-- updatedAt: struct
|    |    |    |    |    |    |-- N: string
|    |-- strings: struct
|    |    |-- SS: array
|    |    |    |-- element: string
|    |-- numbers: struct
|    |    |-- NS: array
|    |    |    |-- element: string
|    |-- binaries: struct
|    |    |-- BS: array
|    |    |    |-- element: string
|    |-- isDDBJson: struct
|    |    |-- BOOL: boolean
|    |-- nullValue: struct
|    |    |-- NULL: boolean
```

コードの例

```
import com.amazonaws.services.glue.GlueContext
import com.amazonaws.services.glue.util.GlueArgParser
import com.amazonaws.services.glue.util.Job
import com.amazonaws.services.glue.util.JsonOptions
import com.amazonaws.services.glue.DynamoDbDataSink
import org.apache.spark.SparkContextimport scala.collection.JavaConverters._

object GlueApp {

  def main(sysArgs: Array[String]): Unit = {
    val glueContext = new GlueContext(SparkContext.getOrCreate())
    val args = GlueArgParser.getResolvedOptions(sysArgs, Seq("JOB_NAME").toArray)
    Job.init(args("JOB_NAME"), glueContext, args.asJava)

    val dynamicFrame = glueContext.getSourceWithFormat(
      connectionType = "dynamodb",
      options = JsonOptions(Map(
        "dynamodb.export" -> "ddb",
        "dynamodb.tableArn" -> "ddbTableARN",
        "dynamodb.s3.bucket" -> "exportBucketLocation",
        "dynamodb.s3.prefix" -> "exportBucketPrefix",
        "dynamodb.s3.bucketOwner" -> "exportBucketAccountID",
      ))
    ).getDynamicFrame()

    val simplified = dynamicFrame.simplifyDDBJson()
    simplified.printSchema()

    Job.commit()
  }

}
```

出力例

この simplifyDDBJson 変換により、以下のように簡略化されます。

```
root
|-- parentMap: struct
|    |-- childMap: struct
|    |    |-- appName: string
|    |    |-- packageName: string
|    |    |-- updatedAt: string
```

```
|-- strings: array
|    |-- element: string
|-- numbers: array
|    |-- element: string
|-- binaries: array
|    |-- element: string
|-- isDDBJson: boolean
|-- nullValue: null
```

Def spigot

```
def spigot( path : String,
            options : JsonOptions = new JsonOptions("{}"),
            transformationContext : String = "",
            callSite : CallSite = CallSite("Not provided"),
            stageThreshold : Long = 0,
            totalThreshold : Long = 0
          ) : DynamicFrame
```

同じレコードを返すが、副作用としてレコードのサブセットを書き出すパススルー変換。

- path – 出力を s3://bucket//path 形式で書き込む先の Amazon S3 のパス。
- options - サンプリング動作を記述するオプションの JsonOptions マップ。

このレコードと同じレコードを含む DynamicFrame を返します。

デフォルトでは、path で指定した場所に任意の 100 レコードを書き込みます。この動作は、options マップを使用してカスタマイズできます。有効なキーは以下のとおりです。

- topk – 書き出されるレコードの総数を指定します。デフォルトは 100 です。
- prob – 各レコードを含める確率を (小数として) 指定します。デフォルトは1です。

例えば、以下の呼び出しでは、データセットをサンプリングするために、20 パーセントの確率で各レコードを選択し、200 レコードを書き出した後で停止します。

```
{{{
  df.spigot("s3://my_bucket/my_path", JsonOptions(Map("topk" -&gt; 200, "prob" -&gt;
 0.2)))
}}}
```

Def splitFields

```
def splitFields( paths : Seq[String],
                 transformationContext : String = "",
                 callSite : CallSite = CallSite("Not provided", ""),
                 stageThreshold : Long = 0,
                 totalThreshold : Long = 0
               ) : Seq[DynamicFrame]
```

- paths - 最初の DynamicFrame に含めるパス。

2 つの DynamicFrame のシーケンスを返します。DynamicFrame の 1 つ目には指定したパスが含まれ、2 つ目には他のすべての列が含まれます。

例

この例では、persons Glue Data Catalog の legislators データベースの AWS テーブルから作成した DynamicFrame を 2 つに分割し、指定したフィールドを 1 つ目の DynamicFrame に、残りのフィールドを 2 つ目の DynamicFrame に配置しています。次に、結果から最初の DynamicFrame を選択します。

```
val InputFrame = glueContext.getCatalogSource(database="legislators",
 tableName="persons",
transformationContext="InputFrame").getDynamicFrame()

val SplitField_collection = InputFrame.splitFields(paths=Seq("family_name", "name",
 "links.note",
"links.url", "gender", "image", "identifiers.scheme", "identifiers.identifier",
 "other_names.lang",
"other_names.note", "other_names.name"), transformationContext="SplitField_collection")

val ResultFrame = SplitField_collection(0)
```

Def splitRows

```
def splitRows( paths : Seq[String],
               values : Seq[Any],
               operators : Seq[String],
               transformationContext : String,
               callSite : CallSite,
               stageThreshold : Long,
```

```
              totalThreshold : Long
            ) : Seq[DynamicFrame]
```

列と定数を比較する述語に基づいて行を分割します。

- paths - 比較に使用する列。
- values - 比較に使用する定数値。
- operators - 比較に使用する演算子。

2 つの DynamicFrame のシーケンスを返します。1 つ目には述語が true の行を含め、2 つ目には述語が false の行を含めます。

述語を指定するには 3 つのシーケンスを使用します。"paths" には、ネストされている可能性がある列の名前を含め、"'values' には、比較する定数値を含め、"operators" には、比較に使用する演算子を含めます。3 つすべてのシーケンスを同じ長さにする必要があります。n 演算子では、n 番目の列を n 番目の値と比較します。

各演算子は "!="、"="、"<="、"<"、">="、">" のいずれかにする必要があります。

以下の呼び出しの例では、DynamicFrame を分割し、1 つ目の出力フレームには米国の 65 才を超える人々のレコード、2 つ目には他のすべてのレコードを含めています。

```
{{{
  df.splitRows(Seq("age", "address.country"), Seq(65, "USA"), Seq("&gt;=", "="))
}}}
```

Def stageErrorsCount

```
def stageErrorsCount
```

この DynamicFrame の計算中に生じたエラーレコードの数を返します。この DynamicFrame に入力として渡した以前のオペレーションのエラーは含まれません。

Def toDF

```
def toDF( specs : Seq[ResolveSpec] = Seq.empty[ResolveSpec] ) : DataFrame
```

この DynamicFrame を、同じスキーマとレコードを含む Apache Spark SQL DataFrame に変換します。

> **Note**
>
> DataFrame は ChoiceType をサポートしていないため、このメソッドは ChoiceType 列を StructType に自動的に変換します。選択肢の解決の詳細とオプションについては、「resolveChoice」を参照してください。

Def unbox

```
def unbox( path : String,
           format : String,
           optionString : String = "{}",
           transformationContext : String = "",
           callSite : CallSite = CallSite("Not provided"),
           stageThreshold : Long = 0,
           totalThreshold : Long = 0
         ) : DynamicFrame
```

- path – 解析する列。文字列またはバイナリにする必要があります。
- format – 解析に使用する形式。
- optionString – 形式に渡すオプション (CSV 区切り記号など)。

指定した形式に従って、埋め込まれた文字列またはバイナリ列を解析します。解析された列は、元の列名で構造体の下にネストされます。

例えば、JSON 列が埋め込まれた CSV ファイルがあるとします。

```
name, age, address
Sally, 36, {"state": "NE", "city": "Omaha"}
...
```

最初の解析の後、以下のスキーマを含む DynamicFrame を取得します。

```
{{{
  root
  |-- name: string
  |-- age: int
  |-- address: string
}}}
```

アドレス列で unbox を呼び出して、特定のコンポーネントを解析できます。

```
{{{
  df.unbox("address", "json")
}}}
```

これにより、以下のスキーマを含む DynamicFrame を取得します。

```
{{{
  root
  |-- name: string
  |-- age: int
  |-- address: struct
  |    |-- state: string
  |    |-- city: string
}}}
```

Def unnest

```
def unnest( transformationContext : String = "",
            callSite : CallSite = CallSite("Not Provided"),
            stageThreshold : Long = 0,
            totalThreshold : Long = 0
          ) : DynamicFrame
```

すべてのネストされた構造体が平坦化された新しい DynamicFrame を返します。名前は "." (ピリオド) 文字を使用して生成されます。

例えば、以下のスキーマを含む DynamicFrame があるとします。

```
{{{
  root
  |-- name: string
  |-- age: int
  |-- address: struct
  |    |-- state: string
  |    |-- city: string
}}}
```

次の呼び出しでアドレス構造体がネスト解除されます。

```
{{{
  df.unnest()
}}}
```

その結果、スキーマは以下のようになります。

```
{{{
  root
  |-- name: string
  |-- age: int
  |-- address.state: string
  |-- address.city: string
}}}
```

このメソッドはまた、配列内でネストされた構造体をネスト解除しません。しかしこれまでの慣習上、そのようなフィールドの名前の前には、外側の配列の名前と ".val" が付けられます。

Def unnestDDBJson

```
unnestDDBJson(transformationContext : String = "",
            callSite : CallSite = CallSite("Not Provided"),
            stageThreshold : Long = 0,
            totalThreshold : Long = 0): DynamicFrame
```

特に DynamoDB JSON 構造体内にある DynamicFrame でネスト化された列をネスト解除すると、新しくネスト解除された DynamicFrame が返されます。構造体型の配列のある列は、ネスト解除されません。これは、通常の unnest 変換とは異なる特殊なタイプのネスト解除変換で、データが DynamoDB JSON 構造体に格納されている必要があることに注意してください。詳細については、「DynamoDB JSON」を参照してください。

例えば、DynamoDB JSON 構造体のある エクスポートを読み取るスキーマは次のようになります。

```
root
|-- Item: struct
|    |-- ColA: struct
|    |    |-- S: string
|    |-- ColB: struct
|    |    |-- S: string
|    |-- ColC: struct
|    |    |-- N: string
```

```
|    |-- ColD: struct
|    |    |-- L: array
|    |    |    |-- element: null
```

unnestDDBJson( ) 変換は、以下のように変換します。

```
root
|-- ColA: string
|-- ColB: string
|-- ColC: string
|-- ColD: array
|    |-- element: null
```

次のコード例は、AWS Glue DynamoDB エクスポートコネクタを使用し、DynamoDB JSON unnest を呼び出し、パーティションの数を表示します。

```
import com.amazonaws.services.glue.GlueContext
import com.amazonaws.services.glue.util.GlueArgParser
import com.amazonaws.services.glue.util.Job
import com.amazonaws.services.glue.util.JsonOptions
import com.amazonaws.services.glue.DynamoDbDataSink
import org.apache.spark.SparkContext
import scala.collection.JavaConverters._


object GlueApp {

  def main(sysArgs: Array[String]): Unit = {
    val glueContext = new GlueContext(SparkContext.getOrCreate())
    val args = GlueArgParser.getResolvedOptions(sysArgs, Seq("JOB_NAME").toArray)
    Job.init(args("JOB_NAME"), glueContext, args.asJava)

    val dynamicFrame = glueContext.getSourceWithFormat(
      connectionType = "dynamodb",
      options = JsonOptions(Map(
        "dynamodb.export" -> "ddb",
        "dynamodb.tableArn" -> "<test_source>",
        "dynamodb.s3.bucket" -> "<bucket name>",
        "dynamodb.s3.prefix" -> "<bucket prefix>",
        "dynamodb.s3.bucketOwner" -> "<account_id of bucket>",
      ))
    ).getDynamicFrame()
```

```
    val unnested = dynamicFrame.unnestDDBJson()
    print(unnested.getNumPartitions())

    Job.commit()
  }

}
```

Def withFrameSchema

```
def withFrameSchema( getSchema : () => Schema ) : DynamicFrame
```

- getSchema – 使用するスキーマを返す関数。高価である可能性が高い計算を延期するために、パラメータがゼロである関数として指定します。

この DynamicFrame のスキーマを、指定した値に設定します。高価なスキーマの再計算を回避するために主に内部で使用されます。渡すスキーマには、データ内に存在するすべての列を含める必要があります。

Def withName

```
def withName( name : String ) : DynamicFrame
```

- name - 使用する新しい名前。

この DynamicFrame のコピーを新しい名前で返します。

Def withTransformationContext

```
def withTransformationContext( ctx : String ) : DynamicFrame
```

この DynamicFrame のコピーを、指定した変換コンテキストで返します。

DynamicFrame オブジェクト

パッケージ: com.amazonaws.services.glue

```
object DynamicFrame
```

Def apply

```
def apply( df : DataFrame,
           glueContext : GlueContext
         ) : DynamicFrame
```

Def emptyDynamicFrame

```
def emptyDynamicFrame( glueContext : GlueContext ) : DynamicFrame
```

Def fromPythonRDD

```
def fromPythonRDD( rdd : JavaRDD[Array[Byte]],
                   glueContext : GlueContext
                 ) : DynamicFrame
```

Def ignoreErrors

```
def ignoreErrors( fn : DynamicRecord => DynamicRecord ) : DynamicRecord
```

Def inlineErrors

```
def inlineErrors( msg : String,
                  callSite : CallSite
                ) : (DynamicRecord => DynamicRecord)
```

Def newFrameWithErrors

```
def newFrameWithErrors( prevFrame : DynamicFrame,
                        rdd : RDD[DynamicRecord],
                        name : String = "",
                        transformationContext : String = "",
                        callSite : CallSite,
                        stageThreshold : Long,
                        totalThreshold : Long
                      ) : DynamicFrame
```

AWS Glue Scala DynamicRecord クラス

トピック

- def addField
- def dropField
- def setError
- def isError
- def getError
- def clearError
- def write
- def readFields
- def clone
- def schema
- def getRoot
- def toJson
- def getFieldNode
- def getField
- def hashCode
- def equals
- DynamicRecord オブジェクト
- RecordTraverser 特性

パッケージ: com.amazonaws.services.glue

```
class DynamicRecord extends Serializable with Writable with Cloneable
```

DynamicRecord は、処理対象のデータセット内のデータ行を表す自己記述型のデータ構造体です。自己記述型とは、DynamicRecord が表す行のスキーマを、レコード自体を検査することで取得できるという意味です。DynamicRecord は Apache Spark の Row に似ています。

def addField

```
def addField( path : String,
```

```
            dynamicNode : DynamicNode
          ) : Unit
```

指定したパスに DynamicNode を追加します。

- path - 追加するフィールドのパス。
- dynamicNode - 指定したパスに追加する DynamicNode。

def dropField

```
def dropField(path: String, underRename: Boolean = false): Option[DynamicNode]
```

指定したパスに配列がない場合は、指定したパスから DynamicNode を削除し、削除したノードを返します。

- path - 削除するフィールドへのパス。
- underRenamedropField – 名前変更の変換の一部として が呼び出された場合は true、それ以外の場合は false (デフォルトは false)。

scala.Option Option (DynamicNode) を返します。

def setError

```
def setError( error : Error )
```

error パラメータの指定に従って、このレコードをエラーレコードとして設定します。

戻り値は DynamicRecord。

def isError

```
def isError
```

このレコードがエラーレコードであるかどうかを確認します。

def getError

```
def getError
```

レコードがエラーレコードである場合、Error を受け取ります。このレコードがエラーレコードである場合は scala.Some Some (エラー) を返し、それ以外の場合は scala.None を返します。

def clearError

```
def clearError
```

Error を scala.None.None に設定します。

def write

```
override def write( out : DataOutput ) : Unit
```

def readFields

```
override def readFields( in : DataInput ) : Unit
```

def clone

```
override def clone : DynamicRecord
```

このレコードを新しい DynamicRecord に複製し、返します。

def schema

```
def schema
```

レコードを検査して Schema を取得します。

def getRoot

```
def getRoot : ObjectNode
```

レコードのルート ObjectNode を取得します。

def toJson

```
def toJson : String
```

レコードの JSON 文字列を取得します。

def getFieldNode

```
def getFieldNode( path : String ) : Option[DynamicNode]
```

DynamicNode のオプションとして指定した path でフィールドの値を取得します。

フィールドが存在する場合は scala.Some Some(DynamicNode) を返し、それ以外の場合は scala.None.None を返します。

def getField

```
def getField( path : String ) : Option[Any]
```

DynamicNode のオプションとして指定した path でフィールドの値を取得します。

scala.Some Some (値) を返します。

def hashCode

```
override def hashCode : Int
```

def equals

```
override def equals( other : Any )
```

DynamicRecord オブジェクト

```
object DynamicRecord
```

def apply

```
def apply( row : Row,
           schema : SparkStructType )
```

メソッドを適用して Apache Spark SQL Row を DynamicRecord に変換します。

- `row` – Spark SQL Row。
- `schema` – 行の `Schema`。

戻り値は `DynamicRecord`。

RecordTraverser 特性

```
trait RecordTraverser {
  def nullValue(): Unit
  def byteValue(value: Byte): Unit
  def binaryValue(value: Array[Byte]): Unit
  def booleanValue(value: Boolean): Unit
  def shortValue(value: Short) : Unit
  def intValue(value: Int) : Unit
  def longValue(value: Long) : Unit
  def floatValue(value: Float): Unit
  def doubleValue(value: Double): Unit
  def decimalValue(value: BigDecimal): Unit
  def stringValue(value: String): Unit
  def dateValue(value: Date): Unit
  def timestampValue(value: Timestamp): Unit
  def objectStart(length: Int): Unit
  def objectKey(key: String): Unit
  def objectEnd(): Unit
  def mapStart(length: Int): Unit
  def mapKey(key: String): Unit
  def mapEnd(): Unit
  def arrayStart(length: Int): Unit
  def arrayEnd(): Unit
}
```

AWS Glue Scala GlueContext API

パッケージ: com.amazonaws.services.glue

```
class GlueContext extends SQLContext(sc) (
           @transient val sc : SparkContext,
           val defaultSourcePartitioner : PartitioningStrategy )
```

`GlueContext` は、Amazon Simple Storage Service (Amazon S3) や、AWS Glue Data Catalog、JDBC その他との間で、DynamicFrame の読み取りと書き込みを行うためのエントリポ

イントです。このクラスが提供するユーティリティ関数によって DataSource 特性 オブジェクトと DataSink オブジェクトが作成され、これらのオブジェクトを使用して DynamicFrame を読み書きできます。

また、ソースから作成されたパーティションの数がパーティションの最小しきい値 (デフォルトは 10) 未満の場合は、DynamicFrame で GlueContext を使用してパーティションのターゲット数 (デフォルトは 20) を設定することもできます。

def addIngestionTimeColumns

```
def addIngestionTimeColumns(
         df : DataFrame,
         timeGranularity : String = "") : dataFrame
```

入力 DataFrame への取り込み時間列 (ingest_year、ingest_month、ingest_day、ingest_hour、ingest_minute) を追加します。Amazon S3 のデータカタログテーブルをターゲットとして指定する場合、この関数は、AWS Glue により生成されたスクリプト内で自動的に生成されます。この関数は、出力テーブル上で、取り込み時間列があるパーティションを自動的に更新します。これにより、入力データに明示的な取り込み時間列を指定しなくても、取り込み時間において出力データの自動的なパーティション化が行えます。

- dataFrame – 取り込み時間列の追加先である dataFrame。
- timeGranularity – 時間列の詳細度。有効な値は「day」、「hour」、および「minute」です。例えば、関数に対し「hour」が渡された場合、元の dataFrame は「ingest_year」、「ingest_month」、「ingest_day」に加え「ingest_hour」の時間列を持つことになります。

時間の詳細度列を追加した後、そのデータフレームを返します。

例：

```
glueContext.addIngestionTimeColumns(dataFrame, "hour")
```

def createDataFrameFromOptions

```
def createDataFrameFromOptions( connectionType : String,
                        connectionOptions : JsonOptions,
                        transformationContext : String = "",
```

```
                        format : String = null,
                        formatOptions : JsonOptions = JsonOptions.empty
                      ) : DataSource
```

指定された接続と形式で作成された DataFrame を返します。この関数は、AWS Glue のストリーミングソースのみで使用してください。

- connectionType – ストリーミング接続タイプ。有効な値は、kinesis および kafka です。
- connectionOptions – 接続オプション。これは、Kinesis と Kafka で異なります。各ストリーミングデータソースのすべての接続オプションの一覧は、AWS Glue for Spark での ETL の接続タイプとオプション で確認いただけます。ストリーミング接続オプションについては、以下の違いに注意してください。
  - Kinesis ストリーミングのソースには streamARN、startingPosition、inferSchema、および classification が必要です。
  - Kafka ストリーミングのソースには connectionName、topicName、startingOffsets、inferSchema、および classification が必要です。
- transformationContext – 使用する変換コンテキスト (オプション)。
- format – 形式の仕様 (オプション)。Amazon S3 や、複数のフォーマットをサポートする AWS Glue 接続の場合に使用します。サポートされる形式については、「AWS Glue for Spark での入出力のデータ形式に関するオプション」を参照してください。
- formatOptions – 指定された形式に関するオプション。サポートされる形式オプションについては、「データ形式に関するオプション」を参照してください。

Amazon Kinesis ストリーミングソースの例:

```
val data_frame_datasource0 =
glueContext.createDataFrameFromOptions(transformationContext = "datasource0",
 connectionType = "kinesis",
connectionOptions = JsonOptions("""{"streamName": "example_stream", "startingPosition":
 "TRIM_HORIZON", "inferSchema": "true", "classification": "json"}}""" ))
```

Kafka ストリーミングソースの例 :

```
val data_frame_datasource0 =
glueContext.createDataFrameFromOptions(transformationContext = "datasource0",
 connectionType = "kafka",
```

```
connectionOptions = JsonOptions("""{"connectionName": "example_connection",
 "topicName": "example_topic", "startingPosition": "earliest", "inferSchema": "false",
 "classification": "json", "schema":"`column1` STRING, `column2` STRING"}"""))
```

forEachBatch

forEachBatch(frame, batch_function, options)

ストリーミングソースから読み取られるすべてのマイクロバッチに渡される、batch_function を適用します。

- frame – 現在のマイクロバッチを含む DataFrame。
- batch_function – すべてのマイクロバッチに適用される関数。
- options – マイクロバッチの処理方法に関する情報を保持している、キーと値のペアの集合。以下のような必須オプションがあります。
  - windowSize – 各バッチの処理にかかる時間。
  - checkpointLocation – ストリーミング ETL ジョブ用に、チェックポイントが格納される場所。
  - batchMaxRetries – 失敗した場合にこのバッチを再試行する最大回数。デフォルト値は 3 です。このオプションは、Glue バージョン 2.0 以降でのみ設定可能です。

例:

```
glueContext.forEachBatch(data_frame_datasource0, (dataFrame: Dataset[Row], batchId:
 Long) =>
   {
      if (dataFrame.count() > 0)
        {
          val datasource0 = DynamicFrame(glueContext.addIngestionTimeColumns(dataFrame,
 "hour"), glueContext)
          // @type: DataSink
          // @args: [database = "tempdb", table_name = "fromoptionsoutput",
 stream_batch_time = "100 seconds",
          //      stream_checkpoint_location = "s3://from-options-testing-eu-central-1/
fromOptionsOutput/checkpoint/",
          //      transformation_ctx = "datasink1"]
          // @return: datasink1
          // @inputs: [frame = datasource0]
          val options_datasink1 = JsonOptions(
```

```
            Map("partitionKeys" -> Seq("ingest_year", "ingest_month","ingest_day",
 "ingest_hour"),
            "enableUpdateCatalog" -> true))
         val datasink1 = glueContext.getCatalogSink(
            database = "tempdb",
            tableName = "fromoptionsoutput",
            redshiftTmpDir = "",
            transformationContext = "datasink1",
            additionalOptions = options_datasink1).writeDynamicFrame(datasource0)
        }
   }, JsonOptions("""{"windowSize" : "100 seconds",
         "checkpointLocation" : "s3://from-options-testing-eu-central-1/
fromOptionsOutput/checkpoint/"}"""))
```

def getCatalogSink

```
def getCatalogSink( database : String,
        tableName : String,
        redshiftTmpDir : String = "",
        transformationContext : String = ""
        additionalOptions: JsonOptions = JsonOptions.empty,
        catalogId: String = null
) : DataSink
```

Data Catalog で定義されているテーブル内の指定した場所に書き込みを行う DataSink を作成します。

- `database` – Data Catalog 内のデータベース名。
- `tableName` – Data Catalog 内のテーブル名。
- `redshiftTmpDir` – 特定のデータシンクで使用する一時的なステージングディレクトリ。デフォルトでは空に設定されます。
- `transformationContext` – ジョブのブックマークで使用されるシンクに関連付けられた変換コンテキスト。デフォルトでは空に設定されます。
- `additionalOptions` – AWS Glue で使用する追加のオプション。
- `catalogId` – 現在アクセスされている Data Catalog のカタログ ID (アカウント ID)。null の場合は、呼び出し元のアカウント ID のデフォルトが使用されます。

戻り値は `DataSink`。

def getCatalogSource

```
def getCatalogSource( database : String,
                      tableName : String,
                      redshiftTmpDir : String = "",
                      transformationContext : String = ""
                      pushDownPredicate : String = " "
                      additionalOptions: JsonOptions = JsonOptions.empty,
                      catalogId: String = null
                    ) : DataSource
```

Data Catalog のテーブル定義からデータを読み取る DataSource 特性 を作成します。

- `database` – Data Catalog 内のデータベース名。
- `tableName` – Data Catalog 内のテーブル名。
- `redshiftTmpDir` – 特定のデータシンクで使用する一時的なステージングディレクトリ。デフォルトでは空に設定されます。
- `transformationContext` – ジョブのブックマークで使用されるシンクに関連付けられた変換コンテキスト。デフォルトでは空に設定されます。
- `pushDownPredicate` – データセットのすべてのファイルをリストアップして読み取る必要がないフィルタパーティション。詳しくは、「プッシュダウン述語を使用した事前フィルタ処理」を参照してください。
- `additionalOptions` - オプションの名前と値のペアのコレクション。AWS Glue for Spark での ETL の接続タイプとオプション でリストされている使用可能なオプション (`endpointUrl`、`streamName`、`bootstrap.servers`、`security.protocol`、`topicName`、`classi` および `delimiter` を除く)。別のオプションとして、`catalogPartitionPredicate` もサポートされています。

  `catalogPartitionPredicate` – カタログ式を渡して、インデックス列に基づいたフィルタリングができます。これにより、フィルタリングをサーバー側で処理できます。詳細については、「AWS Glue パーティションインデックス」を参照してください。`push_down_predicate` と `catalogPartitionPredicate` では、異なる構文が使用されることに注意してください。前者では Spark SQL の標準構文を使用し、後者では JSQL パーサーを使用します。
- `catalogId` – 現在アクセスされている Data Catalog のカタログ ID (アカウント ID)。null の場合は、呼び出し元のアカウント ID のデフォルトが使用されます。

戻り値は `DataSource`。

ストリーミングソースの例

```
val data_frame_datasource0 = glueContext.getCatalogSource(
    database = "tempdb",
    tableName = "test-stream-input",
    redshiftTmpDir = "",
    transformationContext = "datasource0",
    additionalOptions = JsonOptions("""{
        "startingPosition": "TRIM_HORIZON", "inferSchema": "false"}""")
    ).getDataFrame()
```

def getJDBCSink

```
def getJDBCSink( catalogConnection : String,
                 options : JsonOptions,
                 redshiftTmpDir : String = "",
                 transformationContext : String = "",
                 catalogId: String = null
               ) : DataSink
```

Data Catalog 内の Connection オブジェクトに指定されている、JDBC データベースに対し書き込みを行う DataSink を作成します。Connection オブジェクトには、URL、ユーザー名、パスワード、VPC、サブネット、セキュリティグループなど、JDBC シンクに接続するための情報があります。

- catalogConnection – 書き込み先の JDBC URL を含む Data Catalog の接続名。
- options – JDBC データストアへの書き込みに必要な追加情報を提供する、名前/値ペアを示す JSON 形式の文字列。これには、以下が含まれます。
  - dbtable (必須) - JDBC テーブルの名前。データベース内でスキーマをサポートする JDBC データストアの場合、schema.table-name を指定します。スキーマを指定しない場合、デフォルトの「パブリック」スキーマが使用されます。次の例では、test_db データベースの test という名前のスキーマおよび test_table という名前のテーブルを指すオプションパラメータを示します。

    ```
    options = JsonOptions("""{"dbtable": "test.test_table", "database": "test_db"}""")
    ```

  - database (必須) - JDBC データベースの名前。
  - その他のオプションはすべて SparkSQL JDBC ライターに直接渡されます。詳細については、「Redshift data source for Spark」を参照してください。

- redshiftTmpDir – 特定のデータシンクで使用する一時的なステージングディレクトリ。デフォルトでは空に設定されます。
- transformationContext – ジョブのブックマークで使用されるシンクに関連付けられた変換コンテキスト。デフォルトでは空に設定されます。
- catalogId – 現在アクセスされている Data Catalog のカタログ ID (アカウント ID)。null の場合は、呼び出し元のアカウント ID のデフォルトが使用されます。

コード例:

```
getJDBCSink(catalogConnection = "my-connection-name", options =
 JsonOptions("""{"dbtable": "my-jdbc-table", "database": "my-jdbc-db"}"""),
 redshiftTmpDir = "", transformationContext = "datasink4")
```

戻り値は DataSink。

def getSink

```
def getSink( connectionType : String,
             connectionOptions : JsonOptions,
             transformationContext : String = ""
           ) : DataSink
```

Amazon Simple Storage Service (Amazon S3)、JDBC、AWS Glue Data Catalog、Apache Kafka、Amazon Kinesis Data Streams などの書き込み先にデータを書き込む DataSink を作成します。

- connectionType - 接続のタイプ。「the section called “接続パラメータ”」を参照してください。
- connectionOptions – データシンクとの接続を確立するための追加情報を提供する JSON 形式の名前と値のペアの文字列。「the section called “接続パラメータ”」を参照してください。
- transformationContext – ジョブのブックマークで使用されるシンクに関連付けられた変換コンテキスト。デフォルトでは空に設定されます。

戻り値は DataSink。

def getSinkWithFormat

```
def getSinkWithFormat( connectionType : String,
```

```
                        options : JsonOptions,
                        transformationContext : String = "",
                        format : String = null,
                        formatOptions : JsonOptions = JsonOptions.empty
                      ) : DataSink
```

Amazon S3、JDBC、Data Catalog、Apache Kafka、Amazon Kinesis Data Streams などの書き込み先にデータを書き込む DataSink を作成します。書き込み先に書き込むデータの形式も設定します。

- connectionType - 接続のタイプ。「the section called “接続パラメータ”」を参照してください。
- options - データシンクとの接続を確立するための追加情報を提供する JSON 形式の名前/値ペアの文字列。「the section called “接続パラメータ”」を参照してください。
- transformationContext – ジョブのブックマークで使用されるシンクに関連付けられた変換コンテキスト。デフォルトでは空に設定されます。
- format - 書き込み先に書き込むデータの形式。
- formatOptions — 書き込み先でデータをフォーマットするための追加オプションを提供する JSON 形式の名前と値のペアの文字列。「データ形式に関するオプション」を参照してください。

戻り値は DataSink。

def getSource

```
def getSource( connectionType : String,
               connectionOptions : JsonOptions,
               transformationContext : String = ""
               pushDownPredicate
             ) : DataSource
```

Amazon S3、JDBC、AWS Glue Data Catalog などのソースからデータを読み取る DataSource 特性を作成します。Kafka および Kinesis ストリーミングデータソースもサポートしています。

- connectionType – データソースの型。「the section called “接続パラメータ”」を参照してください。
- connectionOptions – データソースとの接続を確立するための追加情報を提供する JSON 形式の名前と値のペアの文字列。詳細については、「the section called “接続パラメータ”」を参照してください。

Kinesis ストリーミングソースには、streamARN、startingPosition、inferSchema、および classification の接続オプションが必要です。

Kafka ストリーミングソースには、connectionName、topicName、startingOffsets、inferSchema、および classification の接続オプションが必要です。

- transformationContext – ジョブのブックマークで使用されるシンクに関連付けられた変換コンテキスト。デフォルトでは空に設定されます。
- pushDownPredicate — パーティション列に関する述語です。

戻り値は DataSource。

Amazon Kinesis ストリーミングソースの例:

```
val kinesisOptions = jsonOptions()
data_frame_datasource0 = glueContext.getSource("kinesis",
 kinesisOptions).getDataFrame()

private def jsonOptions(): JsonOptions = {
    new JsonOptions(
      s"""{"streamARN": "arn:aws:kinesis:eu-central-1:123456789012:stream/
fromOptionsStream",
         |"startingPosition": "TRIM_HORIZON",
         |"inferSchema": "true",
         |"classification": "json"}""".stripMargin)
}
```

Kafka ストリーミングソースの例 :

```
val kafkaOptions = jsonOptions()
val data_frame_datasource0 = glueContext.getSource("kafka",
 kafkaOptions).getDataFrame()

private def jsonOptions(): JsonOptions = {
    new JsonOptions(
      s"""{"connectionName": "ConfluentKafka",
         |"topicName": "kafka-auth-topic",
         |"startingOffsets": "earliest",
         |"inferSchema": "true",
         |"classification": "json"}""".stripMargin)
```

```
}
```

def getSourceWithFormat

```
def getSourceWithFormat( connectionType : String,
                         options : JsonOptions,
                         transformationContext : String = "",
                         format : String = null,
                         formatOptions : JsonOptions = JsonOptions.empty
                       ) : DataSource
```

Amazon S3、JDBC、AWS Glue Data Catalog などのソースからのデータ読み取りと、ソースに保存されているデータの形式指定を行うための DataSource 特性 を作成します。

- connectionType – データソースの型。「the section called “接続パラメータ”」を参照してください。
- options – データソースとの接続を確立するための追加情報を提供する、名前と値のペアを示す JSON 形式の文字列。「the section called “接続パラメータ”」を参照してください。
- transformationContext – ジョブのブックマークで使用するシンクに関連付けられている変換コンテキスト。デフォルトでは空に設定されます。
- format – ソースに保存されるデータの形式。connectionType が "s3" のときは、format を指定することもできます。"avro"、"csv"、"grokLog"、"ion"、"json"、"xml"、"parquet"、"orc" のいずれかになります。
- formatOptions – ソースでデータを解析するための追加オプションを提供する、名前と値ペアを示す JSON 形式の文字列。「データ形式に関するオプション」を参照してください。

戻り値は DataSource。

例

データソース (Amazon S3 上のコンマ区切り値 (CSV) ファイル) から DynamicFrame を作成します。

```
val datasource0 = glueContext.getSourceWithFormat(
    connectionType="s3",
    options =JsonOptions(s"""{"paths": [ "s3://csv/nycflights.csv"]}"""),
    transformationContext = "datasource0",
    format = "csv",
    formatOptions=JsonOptions(s"""{"withHeader":"true","separator": ","}""")
```

```
    ).getDynamicFrame()
```

データソース (JDBC 接続を使用している PostgreSQL) から DynamicFrame を作成します。

```
val datasource0 = glueContext.getSourceWithFormat(
    connectionType="postgresql",
    options =JsonOptions(s"""{
      "url":"jdbc:postgresql://databasePostgres-1.rds.amazonaws.com:5432/testdb",
      "dbtable": "public.company",
      "redshiftTmpDir":"",
      "user":"username",
      "password":"password123"
    }"""),
    transformationContext = "datasource0").getDynamicFrame()
```

データソース (JDBC 接続を使用している MySQL) からDynamicFrameを作成します。

```
 val datasource0 = glueContext.getSourceWithFormat(
    connectionType="mysql",
    options =JsonOptions(s"""{
      "url":"jdbc:mysql://databaseMysql-1.rds.amazonaws.com:3306/testdb",
      "dbtable": "athenatest_nycflights13_csv",
      "redshiftTmpDir":"",
      "user":"username",
      "password":"password123"
    }"""),
    transformationContext = "datasource0").getDynamicFrame()
```

def getSparkSession

```
def getSparkSession : SparkSession
```

この GlueContext に関連付けられている SparkSession オブジェクトを取得します。この SparkSession オブジェクトでは、DynamicFrames から作成した DataFrame で使用するテーブルと UDF を登録します。

SparkSession を返します。

def startTransaction

```
def startTransaction(readOnly: Boolean):String
```

新しいトランザクションの開始。Lake Formation startTransaction API を内部的に呼び出します。

- readOnly — (Boolean) このトランザクションを読み取り専用にするか、または読み取りおよび書き込みを行うかを示します。読み取り専用のトランザクション ID を使用した書き込みは拒否されます。読み取り専用トランザクションはコミットする必要はありません。

トランザクション ID を返します。

def commitTrac

```
def commitTransaction(transactionId: String, waitForCommit: Boolean): Boolean
```

指定されたトランザクションをコミットしようとします。commitTransaction トランザクションのコミットが完了する前に戻ることがあります。Lake Formation startTransaction API を内部的に呼び出します。

- transactionId — (文字列) コミットするトランザクション。
- waitForCommit — (ブール値) commitTransaction がすぐに戻るかどうか指定します。デフォルト値は True です。false の場合、commitTransaction はトランザクションがコミットされるまでポーリングし待機します。最大で 6 回の再試行でエクスポネンシャルバックオフを使用すると、待機時間は 1 分に制限されます。

コミットが完了したかどうかを示すブール値を返します。

def cancelTransaction

```
def cancelTransaction(transactionId: String): Unit
```

指定されたトランザクションをキャンセルしようとします。Lake Formation startTransaction API を内部的に呼び出します。

- transactionId — (文字列) キャンセルするトランザクション。

戻り値は、トランザクションが以前にコミットされた場合は TransactionCommittedException 例外です。

def this

```
def this( sc : SparkContext,
          minPartitions : Int,
          targetPartitions : Int )
```

指定した SparkContext、最小パーティション数、ターゲットパーティション数を使用して GlueContext オブジェクトを作成します。

- sc - SparkContext。
- minPartitions - 最小パーティション数。
- targetPartitions - ターゲットパーティション数。

戻り値は GlueContext。

def this

```
def this( sc : SparkContext )
```

渡された SparkContext で GlueContext オブジェクトを作成します。最小限のパーティション数を 10 に設定し、ターゲットパーティション数を 20 に設定します。

- sc - SparkContext。

戻り値は GlueContext。

def this

```
def this( sparkContext : JavaSparkContext )
```

渡された JavaSparkContext で GlueContext オブジェクトを作成します。最小限のパーティション数を 10 に設定し、ターゲットパーティション数を 20 に設定します。

- sparkContext - JavaSparkContext。

戻り値は GlueContext。

MappingSpec

パッケージ: com.amazonaws.services.glue

MappingSpec ケースクラス

```
case class MappingSpec( sourcePath: SchemaPath,
                        sourceType: DataType,
                        targetPath: SchemaPath,
                        targetType: DataTyp
                       ) extends Product4[String, String, String, String] {
  override def _1: String = sourcePath.toString
  override def _2: String = ExtendedTypeName.fromDataType(sourceType)
  override def _3: String = targetPath.toString
  override def _4: String = ExtendedTypeName.fromDataType(targetType)
}
```

- `sourcePath` - ソースフィールドの `SchemaPath`。
- `sourceType` - ソースフィールドの `DataType`。
- `targetPath` - ターゲットフィールドの `SchemaPath`。
- `targetType` - ターゲットフィールドの `DataType`。

`MappingSpec` は、ソースパスとソースデータ型から、ターゲットパスとターゲットデータ型へのマッピングを指定します。ソーフフレームのソースパスの値は、ターゲットフレームのターゲットパスに表示されます。ソースデータ型はターゲットデータ型にキャストされます。

`Product4` からの拡張であるため、`applyMapping` インターフェイスで `Product4` をすべて処理できます。

MappingSpec オブジェクト

```
object MappingSpec
```

`MappingSpec` オブジェクトには以下のメンバーがあります。

Val orderingByTarget

```
val orderingByTarget: Ordering[MappingSpec]
```

Def apply

```
def apply( sourcePath : String,
           sourceType : DataType,
           targetPath : String,
           targetType : DataType
         ) : MappingSpec
```

MappingSpec を作成します。

- sourcePath - ソースパスの文字列表現。
- sourceType - ソース DataType。
- targetPath - ターゲットパスの文字列表現。
- targetType - ターゲット DataType。

戻り値は MappingSpec。

Def apply

```
def apply( sourcePath : String,
           sourceTypeString : String,
           targetPath : String,
           targetTypeString : String
         ) : MappingSpec
```

MappingSpec を作成します。

- sourcePath - ソースパスの文字列表現。
- sourceType - ソースデータ型の文字列表現。
- targetPath - ターゲットパスの文字列表現。
- targetType - ターゲットデータ型の文字列表現。

MappingSpec を返します。

Def apply

```
def apply( product : Product4[String, String, String, String] ) : MappingSpec
```

MappingSpec を作成します。

- product - ソースパス、ソースデータ型、ターゲットパス、およびターゲットデータ型の Product4。

戻り値は MappingSpec。

AWS Glue Scala ResolveSpec API

トピック

- ResolveSpec オブジェクト
- ResolveSpec ケースクラス

パッケージ: com.amazonaws.services.glue

ResolveSpec オブジェクト

ResolveSpec

```
object ResolveSpec
```

def

```
def apply( path : String,
           action : String
         ) : ResolveSpec
```

ResolveSpec を作成します。

- path - 解決する必要がある選択肢フィールドの文字列表現。
- action — 解決のためのアクション。アクションは Project、KeepAsStruct、または Cast のいずれかになります。

戻り値は ResolveSpec。

def

```
def apply( product : Product2[String, String] ) : ResolveSpec
```

ResolveSpec を作成します。

- product - : ソースパスおよび解決アクションの Product2。

戻り値は ResolveSpec。

ResolveSpec ケースクラス

```
case class ResolveSpec extends Product2[String, String]  (
           path : SchemaPath,
           action : String )
```

ResolveSpec を作成します。

- path - 解決する必要がある選択肢フィールドの SchemaPath。
- action — 解決のためのアクション。アクションは Project、KeepAsStruct、または Cast のいずれかになります。

ResolveSpec def メソッド

```
def _1 : String
```

```
def _2 : String
```

AWS Glue Scala ArrayNode API

パッケージ: com.amazonaws.services.glue.types

ArrayNode ケースクラス

ArrayNode

```
case class ArrayNode extends DynamicNode  (
           value : ArrayBuffer[DynamicNode] )
```

ArrayNode def メソッド

```
def add( node : DynamicNode )
```

```
def clone
```

```
def equals( other : Any )
```

```
def get( index : Int ) : Option[DynamicNode]
```

```
def getValue
```

```
def hashCode : Int
```

```
def isEmpty : Boolean
```

```
def nodeType
```

```
def remove( index : Int )
```

```
def this
```

```
def toIterator : Iterator[DynamicNode]
```

```
def toJson : String
```

```
def update( index : Int,
            node : DynamicNode )
```

AWS Glue Scala BinaryNode API

パッケージ: com.amazonaws.services.glue.types

BinaryNode ケースクラス

BinaryNode

```
case class BinaryNode extends ScalarNode(value, TypeCode.BINARY)  (
           value : Array[Byte] )
```

BinaryNode val フィールド

- ordering

BinaryNode def メソッド

```
def clone
```

```
def equals( other : Any )
```

```
def hashCode : Int
```

AWS Glue Scala BooleanNode API

パッケージ: com.amazonaws.services.glue.types

BooleanNode ケースクラス

BooleanNode

```
case class BooleanNode extends ScalarNode(value, TypeCode.BOOLEAN)  (
           value : Boolean )
```

BooleanNode val フィールド

- ordering

BooleanNode def メソッド

```
def equals( other : Any )
```

AWS Glue Scala ByteNode API

パッケージ: com.amazonaws.services.glue.types

ByteNode ケースクラス

ByteNode

```
case class ByteNode extends ScalarNode(value, TypeCode.BYTE)  (
```

```
            value : Byte )
```

ByteNode val フィールド

- `ordering`

ByteNode def メソッド

```
def equals( other : Any )
```

AWS Glue Scala DateNode API

パッケージ: com.amazonaws.services.glue.types

DateNode ケースクラス

DateNode

```
case class DateNode extends ScalarNode(value, TypeCode.DATE)  (
            value : Date )
```

DateNode val フィールド

- `ordering`

DateNode def メソッド

```
def equals( other : Any )
```

```
def this( value : Int )
```

AWS Glue Scala DecimalNode API

パッケージ: com.amazonaws.services.glue.types

DecimalNode ケースクラス

DecimalNode

```
case class DecimalNode extends ScalarNode(value, TypeCode.DECIMAL)  (
```

```
            value : BigDecimal )
```

DecimalNode val フィールド

- `ordering`

DecimalNode def メソッド

```
def equals( other : Any )
```

```
def this( value : Decimal )
```

AWS Glue Scala DoubleNode API

パッケージ: com.amazonaws.services.glue.types

DoubleNode ケースクラス

DoubleNode

```
case class DoubleNode extends ScalarNode(value, TypeCode.DOUBLE)  (
            value : Double )
```

DoubleNode val フィールド

- `ordering`

DoubleNode def メソッド

```
def equals( other : Any )
```

AWS Glue Scala DynamicNode API

トピック

- DynamicNode クラス
- DynamicNode オブジェクト

パッケージ: com.amazonaws.services.glue.types

DynamicNode クラス

DynamicNode

```
class DynamicNode extends Serializable with Cloneable
```

DynamicNode def メソッド

```
def getValue : Any
```

平坦な値を取得し、現在のレコードにバインドします。

```
def nodeType : TypeCode
```

```
def toJson : String
```

デバッグのメソッド:

```
def toRow( schema : Schema,
           options : Map[String, ResolveOption]
         ) : Row
```

```
def typeName : String
```

DynamicNode オブジェクト

DynamicNode

```
object DynamicNode
```

DynamicNode def メソッド

```
def quote( field : String,
           useQuotes : Boolean
         ) : String
```

```
def quote( node : DynamicNode,
           useQuotes : Boolean
```

```
        ) : String
```

EvaluateDataQuality クラス

AWS Glue Data Quality は AWS Glue のプレビューリリースに含まれ、変更される可能性があります。

パッケージ: com.amazonaws.services.glue.dq

```
object EvaluateDataQuality
```

def apply

```
def apply(frame: DynamicFrame,
            ruleset: String,
            publishingOptions: JsonOptions = JsonOptions.empty): DynamicFrame
```

DynamicFrame に対してデータ品質ルールセットを評価し、評価の結果を含む新しい DynamicFrame を返します。AWS Glue Data Quality の詳細については、「AWS Glue Data Quality」を参照してください。

- frame – データ品質を評価する DynamicFrame。
- ruleset – 文字列形式のデータ品質定義言語 (DQDL) ルールセット。DQDL の詳細については、データ品質定義言語 (DQDL) リファレンス のガイドを参照してください。
- publishingOptions – 評価結果とメトリクスを発行する次のオプションを指定するディクショナリ。
  - dataQualityEvaluationContext – AWS Glue が Amazon CloudWatch メトリクスとデータ品質結果を発行する名前空間を指定する文字列。集計されたメトリクスは CloudWatch に表示され、完全な結果は AWS Glue Studio インターフェイスに表示されます。
    - 必須：いいえ
    - デフォルト値: default_context
  - enableDataQualityCloudWatchMetrics– データ品質評価の結果を CloudWatch に発行するかどうかを指定します。dataQualityEvaluationContext オプションを使用してメトリクスの名前空間を指定します。
    - 必須：いいえ

  - デフォルト値: False
- enableDataQualityResultsPublishing – データ品質結果を AWS Glue Studio インターフェイスの [Data Quality] (データ品質) タブに表示するかどうかを指定します。
  - 必須：いいえ
  - デフォルト値: True
- resultsS3Prefix – AWS Glue がデータ品質評価結果を書き込める Amazon S3 ロケーションを指定します。
  - 必須：いいえ
  - デフォルト値: "" (空の文字列)

例

次のコード例は、SelectFields 変換を実行する前に DynamicFrame のデータ品質を評価する方法を示しています。変換を試行する前に、すべてのデータ品質ルールに合格していることを、このスクリプトで検証します。

```
import com.amazonaws.services.glue.GlueContext
import com.amazonaws.services.glue.MappingSpec
import com.amazonaws.services.glue.errors.CallSite
import com.amazonaws.services.glue.util.GlueArgParser
import com.amazonaws.services.glue.util.Job
import com.amazonaws.services.glue.util.JsonOptions
import org.apache.spark.SparkContext
import scala.collection.JavaConverters._
import com.amazonaws.services.glue.dq.EvaluateDataQuality

object GlueApp {
  def main(sysArgs: Array[String]) {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)
    // @params: [JOB_NAME]
    val args = GlueArgParser.getResolvedOptions(sysArgs, Seq("JOB_NAME").toArray)
    Job.init(args("JOB_NAME"), glueContext, args.asJava)

    // Create DynamicFrame with data
    val Legislators_Area = glueContext.getCatalogSource(database="legislators",
 tableName="areas_json", transformationContext="S3bucket_node1").getDynamicFrame()

    // Define data quality ruleset
```

```
    val DQ_Ruleset = """
      Rules = [ColumnExists "id"]
    """

    // Evaluate data quality
    val DQ_Results = EvaluateDataQuality.apply(frame=Legislators_Area,
 ruleset=DQ_Ruleset, publishingOptions=JsonOptions("""{"dataQualityEvaluationContext":
 "Legislators_Area", "enableDataQualityMetrics": "true",
 "enableDataQualityResultsPublishing": "true"}"""))
    assert(DQ_Results.filter(_.getField("Outcome").contains("Failed")).count == 0,
 "Failing DQ rules for Legislators_Area caused the job to fail.")

    // Script generated for node Select Fields
    val SelectFields_Results = Legislators_Area.selectFields(paths=Seq("id", "name"),
 transformationContext="Legislators_Area")

    Job.commit()
  }
}
```

AWS Glue Scala FloatNode API

パッケージ: com.amazonaws.services.glue.types

FloatNode ケースクラス

FloatNode

```
case class FloatNode extends ScalarNode(value, TypeCode.FLOAT)  (
           value : Float )
```

FloatNode val フィールド

- `ordering`

FloatNode def メソッド

```
def equals( other : Any )
```

FillMissingValues クラス

パッケージ: com.amazonaws.services.glue.ml

```
object FillMissingValues
```

def apply

```
def apply(frame: DynamicFrame,
          missingValuesColumn: String,
          outputColumn: String = "",
          transformationContext: String = "",
          callSite: CallSite = CallSite("Not provided", ""),
          stageThreshold: Long = 0,
          totalThreshold: Long = 0): DynamicFrame
```

指定された列の動的フレームの欠落値を埋め、推定値を含む新しい列を持つ新しいフレームを返します。欠落した値がない行の場合、指定した列の値が新しい列に複製されます。

- `frame` — 欠落した値を埋めるための DynamicFrame。必須。
- `missingValuesColumn` — 欠落した値 (`null` 値および空の文字列) を含む列。必須。
- `outputColumn` — 値が欠落しているすべての行に推定値が挿入された新しい列の名前。デフォルトは、末尾に `"_filled"` が付加された `missingValuesColumn` の値です。
- `transformationContext` — 状態の情報を識別するために使用される一意の文字列 (オプション)。
- `callSite` — エラーレポートのコンテキスト情報を提供するために使用します (オプション)。
- `stageThreshold` — エラーで終了する前に変換で許容される、エラーの最大発生数 (オプション、デフォルト値は 0)。
- `totalThreshold` — エラーによる終了が処理される前に、全体で許容されるエラーの最大発生数 (オプション、デフォルト値は 0)。

値が欠落した行のための推定値を含む (追加された) 1 つの列、さらに他の行の現在値を含む、新しい動的フレームを返します。

FindMatches クラス

パッケージ: com.amazonaws.services.glue.ml

```
object FindMatches
```

def apply

```
def apply(frame: DynamicFrame,
            transformId: String,
            transformationContext: String = "",
            callSite: CallSite = CallSite("Not provided", ""),
            stageThreshold: Long = 0,
            totalThreshold: Long = 0,
            enforcedMatches: DynamicFrame = null): DynamicFrame,
   computeMatchConfidenceScores: Boolean
```

入力フレームで一致するものを検索し、一致したグループごとに、一意の ID を使用する新しい列を持つ新しいフレームを返します。

- `frame` — 検索の対象となる DynamicFrame。必須。
- `transformId` — 入力フレームに適用する FindMatches 変換に関連付けられた一意の ID。必須。
- `transformationContext` – この `DynamicFrame` の識別子。`transformationContext` は、実行全体で保持される、ジョブのブックマーク状態のキーとして使用されます。オプション。
- `callSite` – エラーレポートのコンテキスト情報を提供するために使用します。これらの値は、Python から呼び出すときに、自動的に設定されます。オプション。
- `stageThreshold` – この `DynamicFrame` の計算から例外がスローされるまでに許容される、エラーレコードの最大数 (以前の `DynamicFrame` に含まれているレコードは除きます)。オプション。デフォルト値は 0 です。
- `totalThreshold` – 例外がスローされるまでのエラーレコードの合計最大数。以前のフレームにあるレコードも含みます。オプション。デフォルト値は 0 です。
- `enforcedMatches` – 強制的な一致のためのフレーム。オプション。デフォルト: `null`。
- `computeMatchConfidenceScores` — 一致するレコードの各グループの信頼スコアをコンピューティングするかどうかを示すブール値。オプション。デフォルトは False です。

一致するレコードの各グループに割り当てられた一意の識別子を含む、新しい動的フレームを返します。

FindIncrementalMatches クラス

パッケージ: com.amazonaws.services.glue.ml

```
object FindIncrementalMatches
```

def apply

```
apply(existingFrame: DynamicFrame,
            incrementalFrame: DynamicFrame,
            transformId: String,
            transformationContext: String = "",
            callSite: CallSite = CallSite("Not provided", ""),
            stageThreshold: Long = 0,
            totalThreshold: Long = 0,
            enforcedMatches: DynamicFrame = null): DynamicFrame,
   computeMatchConfidenceScores: Boolean
```

既存ならびに増分フレームにわたって一致を検索し、一致グループごとに一意の ID を使用する列を含む、新しいフレームを返します。

- existingframe — グループごとに一致する ID が割り当てられている既存のフレーム。必須。
- incrementalframe — 既存のフレームに対する一致を検索するために使用される増分フレームです。必須。
- transformId — 入力フレームに適用する FindIncrementalMatches 変換に関連付けられた一意の ID。必須。
- transformationContext – この DynamicFrame の識別子。transformationContext は、実行全体で保持される、ジョブのブックマーク状態のキーとして使用されます。オプション。
- callSite – エラーレポートのコンテキスト情報を提供するために使用します。これらの値は、Python から呼び出すときに、自動的に設定されます。オプション。
- stageThreshold – この DynamicFrame の計算から例外がスローされるまでに許容される、エラーレコードの最大数 (以前の DynamicFrame に含まれているレコードは除きます)。オプション。デフォルト値は 0 です。
- totalThreshold – 例外がスローされるまでのエラーレコードの合計最大数。以前のフレームにあるレコードも含みます。オプション。デフォルト値は 0 です。
- enforcedMatches – 強制的な一致のためのフレーム。オプション。デフォルト: null。
- computeMatchConfidenceScores — 一致するレコードの各グループの信頼スコアをコンピューティングするかどうかを示すブール値。オプション。デフォルトは False です。

一致するレコードの各グループに割り当てられた一意の識別子を含む、新しい動的フレームを返します。

AWS Glue Scala IntegerNode API

パッケージ: com.amazonaws.services.glue.types

IntegerNode ケースクラス

IntegerNode

```
case class IntegerNode extends ScalarNode(value, TypeCode.INT)  (
           value : Int )
```

IntegerNode val フィールド

- `ordering`

IntegerNode def メソッド

```
def equals( other : Any )
```

AWS Glue Scala LongNode API

パッケージ: com.amazonaws.services.glue.types

LongNode ケースクラス

LongNode

```
case class LongNode extends ScalarNode(value, TypeCode.LONG)  (
           value : Long )
```

LongNode val フィールド

- `ordering`

LongNode def メソッド

```
def equals( other : Any )
```

AWS Glue Scala MapLikeNode API

パッケージ: com.amazonaws.services.glue.types

MapLikeNode クラス

MapLikeNode

```
class MapLikeNode extends DynamicNode  (
           value : mutable.Map[String, DynamicNode] )
```

MapLikeNode def メソッド

```
def clear : Unit
```

```
def get( name : String ) : Option[DynamicNode]
```

```
def getValue
```

```
def has( name : String ) : Boolean
```

```
def isEmpty : Boolean
```

```
def put( name : String,
         node : DynamicNode
       ) : Option[DynamicNode]
```

```
def remove( name : String ) : Option[DynamicNode]
```

```
def toIterator : Iterator[(String, DynamicNode)]
```

```
def toJson : String
```

```
def toJson( useQuotes : Boolean ) : String
```

例: この JSON の場合:

```
{"foo": "bar"}
```

useQuotes == true であれば、toJson は {"foo": "bar"} を生成します。useQuotes == false であれば、toJson は {foo: bar} @return を生成します。

AWS Glue Scala MapNode API

パッケージ: com.amazonaws.services.glue.types

MapNode ケースクラス

MapNode

```
case class MapNode extends MapLikeNode(value)  (
           value : mutable.Map[String, DynamicNode] )
```

MapNode def メソッド

```
def clone
```

```
def equals( other : Any )
```

```
def hashCode : Int
```

```
def nodeType
```

```
def this
```

AWS Glue Scala NullNode API

トピック

- NullNode クラス
- NullNode ケースオブジェクト

パッケージ: com.amazonaws.services.glue.types

NullNode クラス

NullNode

```
class NullNode
```

NullNode ケースオブジェクト

NullNode

```
case object NullNode extends NullNode
```

AWS Glue Scala ObjectNode API

トピック

- ObjectNode オブジェクト
- ObjectNode ケースクラス

パッケージ: com.amazonaws.services.glue.types

ObjectNode オブジェクト

ObjectNode

```
object ObjectNode
```

ObjectNode def メソッド

```
def apply( frameKeys : Set[String],
           v1 : mutable.Map[String, DynamicNode],
           v2 : mutable.Map[String, DynamicNode],
           resolveWith : String
         ) : ObjectNode
```

ObjectNode ケースクラス

ObjectNode

```
case class ObjectNode extends MapLikeNode(value)  (
           val value : mutable.Map[String, DynamicNode] )
```

ObjectNode def メソッド

```
def clone
```

```
def equals( other : Any )
```

```
def hashCode : Int
```

```
def nodeType
```

```
def this
```

AWS Glue Scala ScalarNode API

トピック

- ScalarNode クラス
- ScalarNode オブジェクト

パッケージ: com.amazonaws.services.glue.types

ScalarNode クラス

ScalarNode

```
class ScalarNode extends DynamicNode  (
           value : Any,
           scalarType : TypeCode )
```

ScalarNode def メソッド

```
def compare( other : Any,
             operator : String
           ) : Boolean
```

```
def getValue
```

```
def hashCode : Int
```

```
def nodeType
```

```
def toJson
```

ScalarNode オブジェクト

ScalarNode

```
object ScalarNode
```

ScalarNode def メソッド

```
def apply( v : Any ) : DynamicNode
```

```
def compare( tv : Ordered[T],
             other : T,
             operator : String
           ) : Boolean
```

```
def compareAny( v : Any,
                y : Any,
                o : String )
```

```
def withEscapedSpecialCharacters( jsonToEscape : String ) : String
```

AWS Glue Scala ShortNode API

パッケージ: com.amazonaws.services.glue.types

ShortNode ケースクラス

ShortNode

```
case class ShortNode extends ScalarNode(value, TypeCode.SHORT)  (
           value : Short )
```

ShortNode val フィールド

- `ordering`

ShortNode def メソッド

```
def equals( other : Any )
```

AWS Glue Scala StringNode API

パッケージ: com.amazonaws.services.glue.types

StringNode ケースクラス

StringNode

```
case class StringNode extends ScalarNode(value, TypeCode.STRING)  (
           value : String )
```

StringNode val フィールド

- ordering

StringNode def メソッド

```
def equals( other : Any )
```

```
def this( value : UTF8String )
```

AWS Glue Scala TimestampNode API

パッケージ: com.amazonaws.services.glue.types

TimestampNode ケースクラス

TimestampNode

```
case class TimestampNode extends ScalarNode(value, TypeCode.TIMESTAMP)  (
           value : Timestamp )
```

TimestampNode val フィールド

- ordering

TimestampNode def メソッド

```
def equals( other : Any )
```

```
def this( value : Long )
```

AWS Glue Scala GlueArgParser API

パッケージ: com.amazonaws.services.glue.util

GlueArgParser オブジェクト

GlueArgParser

```
object GlueArgParser
```

これは、AWSGlueDataplanePython パッケージ内の utils.getResolvedOptions の Python バージョンと厳密に一致しています。

GlueArgParser def メソッド

```
def getResolvedOptions( args : Array[String],
                        options : Array[String]
                      ) : Map[String, String]
```

```
def initParser( userOptionsSet : mutable.Set[String] ) : ArgumentParser
```

Example ジョブに渡された引数を取得する

ジョブ引数を取得するために、getResolvedOptions メソッドを使用できます。aws_region という名前のジョブ引数を取得する次の例を考えてみましょう。

```
val args = GlueArgParser.getResolvedOptions(sysArgs,
 Seq("JOB_NAME","aws_region").toArray)
Job.init(args("JOB_NAME"), glueContext, args.asJava)
val region = args("aws_region")
println(region)
```

AWS Glue Scala Job API

パッケージ: com.amazonaws.services.glue.util

Job オブジェクト

ジョブ

```
object Job
```

Job def メソッド

```
def commit
```

```
def init( jobName : String,
          glueContext : GlueContext,
          args : java.util.Map[String, String] = Map[String, String]().asJava
        ) : this.type
```

```
def init( jobName : String,
          glueContext : GlueContext,
          endpoint : String,
          args : java.util.Map[String, String]
        ) : this.type
```

```
def isInitialized
```

```
def reset
```

```
def runId
```

AWS Glue for Spark ETL スクリプトのプログラミングの機能と最適化

以下のセクションでは、言語を問わず、AWS Glue for Spark ETL (抽出、変換、ロード) プログラミングに適用される一般的な手法と値について説明します。

トピック

- AWS Glue for Spark での ETL の接続タイプとオプション
- AWS Glue for Spark での入出力のデータ形式に関するオプション
- Spark SQL ジョブ用の AWS Glue Data Catalog のサポート
- ジョブのブックマークを使用する
- AWS Glue Studio 外での機密データ検出の使用
- AWS Glue ビジュアルジョブ API

AWS Glue for Spark での ETL の接続タイプとオプション

AWS Glue for Spark では、PySpark と Scala のさまざまなメソッドと変換で、connectionType パラメータを使用しながら接続タイプを指定します。connectionOptions または options パラメータを使用して接続オプションを指定します。

connectionType パラメータには、次の表に示す値を指定できます。各タイプの関連する connectionOptions (または options) パラメータ値については、以下のセクションで説明します。特に明記されていない限り、パラメータは、接続がソースまたはシンクとして使用されるときに適用されます。

接続オプションの設定と使用方法を示すサンプルコードについては、「各接続タイプのホームページ」を参照してください。

connectionType	接続先
dynamodb	Amazon DynamoDB データベース
kinesis	Amazon Kinesis Data Streams
s3	Simple Storage Service (Amazon S3)
documentdb	Amazon DocumentDB (MongoDB 互換) データベース
opensearch	Amazon OpenSearch Service。
Redshift	Amazon Redshift データベース
kafka	Kafka または Amazon Managed Streaming for Apache Kafka
azurecosmos	Azure Cosmos for NoSQL。

connectionType	接続先
azuresql	Azure SQL。
bigquery	Google BigQuery。
mongodb	MongoDB データベース (MongoDB Atlas を含む)。
sqlserver	Microsoft SQL Server データベース (「JDBC 接続」を参照)
mysql	MySQL データベース (「JDBC 接続」を参照)
oracle	Oracle データベース (「JDBC 接続」を参照)
postgresql	PostgreSQL データベース (「JDBC 接続」を参照)
saphana	SAP HANA。
snowflake	Snowflake データレイク
teradata	Teradata Vantage。
vertica	Vertica。
custom.*	Spark、Athena、または JDBC データストア (「カスタムと AWS Marketplace での、connectionType の値」 を参照
marketplace.*	Spark、Athena、または JDBC データストア (「カスタムと AWS Marketplace での、connectionType の値」を参照)

DynamoDB 接続

Spark 用の AWS Glue を使用して AWS Glue の DynamoDB 内のテーブルに対する読み込みと書き込みを行うことができます。AWS Glue ジョブにアタッチされている IAM 権限を使用して DynamoDB に接続します。AWS Glue は、別の AWS アカウントの DynamoDB テーブルに対するデータの書き込みをサポートしています。詳細については、「the section called “DynamoDB テーブルへのクロスアカウントおよびクロスリージョンでのアクセス”」を参照してください。

AWS Glue DynamoDB ETL コネクターに加えて、DynamoDB エクスポートコネクターを使用して DynamoDB から読み取ることができます。このコネクターは、DynamoDB

`ExportTableToPointInTime` リクエストを呼び出し、これを指定した Amazon S3 の場所に DynamoDB JSON 形式で保存します。次に、AWS Glue は、Amazon S3 のエクスポート場所からデータを読み取ることによって DynamicFrame オブジェクトを作成します。

DynamoDB のライターは、AWS Glue バージョン 1.0 以降で利用可能です。AWS Glue DynamoDB のエクスポートコネクターは、AWS Glue バージョン 2.0 以降で利用可能です。

DynamoDB の詳細については、Amazon DynamoDB のドキュメントを参照してください。

> **Note**
>
> DynamoDB ETL リーダーでは、フィルタまたはプッシュダウン述語は使用できません。

DynamoDB 接続の設定

AWS Glue から DynamoDB に接続するには、AWS Glue ジョブに関連付けられた IAM ロールに DynamoDB と対話する権限を付与します。DynamoDB からの読み取りまたは書き込みに必要な権限の詳細については、「IAM ドキュメント」の DynamoDB のアクション、リソース、および条件キーを参照してください。

次の状況では、追加の設定が必要になる場合があります。

- DynamoDB エクスポートコネクターを使用するときは、ジョブが DynamoDB テーブルのエクスポートをリクエストできるように IAM を設定する必要があります。さらに、エクスポート用の Amazon S3 バケットを特定し、DynamoDB がそのバケットに書き込むための適切な権限と、AWS Glue ジョブがそこから読み取るための適切な権限を IAM に付与する必要があります。詳細については、「DynamoDB でテーブルのエクスポートをリクエストする」を参照してください。
- AWS Glue ジョブに特定の Amazon VPC 接続要件がある場合は、`NETWORK` AWS Glue 接続タイプを使用してネットワークオプションを指定します。DynamoDB へのアクセスは IAM によって承認されるため、AWS Glue DynamoDB 接続タイプは必要ありません。

DynamoDB からの読み込みと書き込み

次のコード例では、DynamoDB テーブルからの読み取り (ETL コネクタ経由) と、DynamoDB のテーブルへの書き込み方法を示します。ここでは、あるテーブルから読み取りを行い、別のテーブルに対して書き込みを行っています。

Python

```
import sys
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job
from awsglue.utils import getResolvedOptions

args = getResolvedOptions(sys.argv, ["JOB_NAME"])
glue_context= GlueContext(SparkContext.getOrCreate())
job = Job(glue_context)
job.init(args["JOB_NAME"], args)

dyf = glue_context.create_dynamic_frame.from_options(
    connection_type="dynamodb",
    connection_options={"dynamodb.input.tableName": test_source,
        "dynamodb.throughput.read.percent": "1.0",
        "dynamodb.splits": "100"
    }
)
print(dyf.getNumPartitions())

glue_context.write_dynamic_frame_from_options(
    frame=dyf,
    connection_type="dynamodb",
    connection_options={"dynamodb.output.tableName": test_sink,
        "dynamodb.throughput.write.percent": "1.0"
    }
)

job.commit()
```

Scala

```
import com.amazonaws.services.glue.GlueContext
import com.amazonaws.services.glue.util.GlueArgParser
import com.amazonaws.services.glue.util.Job
import com.amazonaws.services.glue.util.JsonOptions
import com.amazonaws.services.glue.DynamoDbDataSink
import org.apache.spark.SparkContext
import scala.collection.JavaConverters._
```

```
object GlueApp {

  def main(sysArgs: Array[String]): Unit = {
    val glueContext = new GlueContext(SparkContext.getOrCreate())
    val args = GlueArgParser.getResolvedOptions(sysArgs, Seq("JOB_NAME").toArray)
    Job.init(args("JOB_NAME"), glueContext, args.asJava)

    val dynamicFrame = glueContext.getSourceWithFormat(
      connectionType = "dynamodb",
      options = JsonOptions(Map(
        "dynamodb.input.tableName" -> test_source,
        "dynamodb.throughput.read.percent" -> "1.0",
        "dynamodb.splits" -> "100"
      ))
    ).getDynamicFrame()

    print(dynamicFrame.getNumPartitions())

    val dynamoDbSink: DynamoDbDataSink =  glueContext.getSinkWithFormat(
      connectionType = "dynamodb",
      options = JsonOptions(Map(
        "dynamodb.output.tableName" -> test_sink,
        "dynamodb.throughput.write.percent" -> "1.0"
      ))
    ).asInstanceOf[DynamoDbDataSink]

    dynamoDbSink.writeDynamicFrame(dynamicFrame)

    Job.commit()
  }

}
```

DynamoDB エクスポートコネクタを使用する

DynamoDB テーブルサイズが 80 GB を超える場合、エクスポートコネクタは ETL コネクタよりもパフォーマンスが向上します。さらに、エクスポートリクエストが AWS Glue ジョブで Spark プロセスの外部で実行される場合、AWS Glue ジョブの自動スケーリングを有効にして、エクスポートリクエスト中の DPU 使用量を節約できます。エクスポートコネクタでは、Spark エグゼキューターの並列処理のためのスプリット数や、DynamoDB スループットの読み取り率を設定する必要がありません。

> **Note**
>
> DynamoDBには、ExportTableToPointInTime リクエストを呼び出すための特定の要件があります。詳細については、「DynamoDB でテーブルのエクスポートをリクエストする」を参照してください。例えば、このコネクタを使用するには、テーブルでポイントインタイムリストア (PITR) を有効にする必要があります。DynamoDB コネクタは、Amazon S3 への DynamoDB エクスポートの AWS KMS 暗号化もサポートします。AWS Glue ジョブの設定でセキュリティ設定を指定することで、DynamoDB エクスポートの AWS KMS 暗号化が有効になります。KMS キーは、Amazon S3 バケットと同じリージョンにある必要があります。
>
> DynamoDB エクスポートと Amazon S3 ストレージのコストに追加料金が適用されることに注意してください。Amazon S3 でエクスポートされたデータは、ジョブ実行の終了後も保持されるため、DynamoDB を追加でエクスポートしなくても再利用できます。このコネクタを使用するための要件として、テーブルに対してポイントインタイムリカバリ (PITR) が有効になっていることが必要です。
>
> DynamoDB ETL コネクタまたはエクスポートコネクタは、DynamoDB ソースに適用されるフィルタまたはプッシュダウン述語をサポートしていません。

次のコード例は、(エクスポートコネクタを経由して) 読み取り、パーティションの数を出力する方法を示しています。

Python

```
import sys
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job
from awsglue.utils import getResolvedOptions

args = getResolvedOptions(sys.argv, ["JOB_NAME"])
glue_context= GlueContext(SparkContext.getOrCreate())
job = Job(glue_context)
job.init(args["JOB_NAME"], args)

dyf = glue_context.create_dynamic_frame.from_options(
    connection_type="dynamodb",
    connection_options={
        "dynamodb.export": "ddb",
        "dynamodb.tableArn": test_source,
```

```
        "dynamodb.s3.bucket": bucket_name,
        "dynamodb.s3.prefix": bucket_prefix,
        "dynamodb.s3.bucketOwner": account_id_of_bucket,
    }
)
print(dyf.getNumPartitions())

job.commit()
```

Scala

```
import com.amazonaws.services.glue.GlueContext
import com.amazonaws.services.glue.util.GlueArgParser
import com.amazonaws.services.glue.util.Job
import com.amazonaws.services.glue.util.JsonOptions
import com.amazonaws.services.glue.DynamoDbDataSink
import org.apache.spark.SparkContext
import scala.collection.JavaConverters._


object GlueApp {

  def main(sysArgs: Array[String]): Unit = {
    val glueContext = new GlueContext(SparkContext.getOrCreate())
    val args = GlueArgParser.getResolvedOptions(sysArgs, Seq("JOB_NAME").toArray)
    Job.init(args("JOB_NAME"), glueContext, args.asJava)

    val dynamicFrame = glueContext.getSourceWithFormat(
      connectionType = "dynamodb",
      options = JsonOptions(Map(
        "dynamodb.export" -> "ddb",
        "dynamodb.tableArn" -> test_source,
        "dynamodb.s3.bucket" -> bucket_name,
        "dynamodb.s3.prefix" -> bucket_prefix,
        "dynamodb.s3.bucketOwner" -> account_id_of_bucket,
      ))
    ).getDynamicFrame()

    print(dynamicFrame.getNumPartitions())

    Job.commit()
  }
```

```
}
```

これらの例は、(エクスポートコネクタを介して) 読み取りを行い、dynamodb 分類を持つ AWS Glue データカタログテーブルからパーティションの数を出力する方法を示しています。

Python

```
import sys
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job
from awsglue.utils import getResolvedOptions

args = getResolvedOptions(sys.argv, ["JOB_NAME"])
glue_context= GlueContext(SparkContext.getOrCreate())
job = Job(glue_context)
job.init(args["JOB_NAME"], args)

dynamicFrame = glue_context.create_dynamic_frame.from_catalog(
        database=catalog_database,
        table_name=catalog_table_name,
        additional_options={
            "dynamodb.export": "ddb",
            "dynamodb.s3.bucket": s3_bucket,
            "dynamodb.s3.prefix": s3_bucket_prefix
        }
    )
print(dynamicFrame.getNumPartitions())

job.commit()
```

Scala

```
import com.amazonaws.services.glue.GlueContext
import com.amazonaws.services.glue.util.GlueArgParser
import com.amazonaws.services.glue.util.Job
import com.amazonaws.services.glue.util.JsonOptions
import com.amazonaws.services.glue.DynamoDbDataSink
import org.apache.spark.SparkContext
import scala.collection.JavaConverters._
```

```
object GlueApp {

  def main(sysArgs: Array[String]): Unit = {
    val glueContext = new GlueContext(SparkContext.getOrCreate())
    val args = GlueArgParser.getResolvedOptions(sysArgs, Seq("JOB_NAME").toArray)
    Job.init(args("JOB_NAME"), glueContext, args.asJava)

    val dynamicFrame = glueContext.getCatalogSource(
        database = catalog_database,
        tableName = catalog_table_name,
        additionalOptions = JsonOptions(Map(
            "dynamodb.export" -> "ddb",
            "dynamodb.s3.bucket" -> s3_bucket,
            "dynamodb.s3.prefix" -> s3_bucket_prefix
        ))
    ).getDynamicFrame()
    print(dynamicFrame.getNumPartitions())
)
```

DynamoDB エクスポート JSON の使用を簡素化

AWS Glue DynamoDB エクスポートコネクタを使用した DynamoDB エクスポートでは、特殊なネスト構造の JSON ファイルが生成されます。詳細については、データオブジェクトを参照してください。 AWS Glue は DynamicFrame 変換を提供します。これを使用して、このような構造をダウンストリームアプリケーションで使いやすい形式に変換できます。

変換は 2 つの方法のいずれかで起動できます。。DynamoDB から読み取るメソッドを呼び出すときに、値 "`true`" を使用して接続オプション "`dynamodb.simplifyDDBJson`" を設定できます。また、AWS Glue ライブラリで独立して利用可能なメソッドとして変換を呼び出すこともできます。

DynamoDB エクスポートによって生成された次のスキーマを考えてみましょう。

```
root
|-- Item: struct
|    |-- parentMap: struct
|    |    |-- M: struct
|    |    |    |-- childMap: struct
|    |    |    |    |-- M: struct
|    |    |    |    |    |-- appName: struct
|    |    |    |    |    |    |-- S: string
|    |    |    |    |    |-- packageName: struct
```

```
|    |    |    |    |    |    |-- S: string
|    |    |    |    |    |-- updatedAt: struct
|    |    |    |    |    |    |-- N: string
|    |-- strings: struct
|    |    |-- SS: array
|    |    |    |-- element: string
|    |-- numbers: struct
|    |    |-- NS: array
|    |    |    |-- element: string
|    |-- binaries: struct
|    |    |-- BS: array
|    |    |    |-- element: string
|    |-- isDDBJson: struct
|    |    |-- BOOL: boolean
|    |-- nullValue: struct
|    |    |-- NULL: boolean
```

この simplifyDDBJson 変換により、以下のように簡略化されます。

```
root
|-- parentMap: struct
|    |-- childMap: struct
|    |    |-- appName: string
|    |    |-- packageName: string
|    |    |-- updatedAt: string
|-- strings: array
|    |-- element: string
|-- numbers: array
|    |-- element: string
|-- binaries: array
|    |-- element: string
|-- isDDBJson: boolean
|-- nullValue: null
```

> **Note**
>
> simplifyDDBJson は、AWS Glue バージョン 3.0 以降で有効です。unnestDDBJson トランスフォームを使用して DynamoDB のエクスポート JSON を簡略化することもできます。ユーザーには unnestDDBJson から simplifyDDBJson への移行をお勧めします。

DynamoDB オペレーションにおける並列処理の設定

パフォーマンスを向上させるために、DynamoDB コネクタで使用できる特定のパラメーターを調整できます。並列処理パラメーターを調整する際の目標は、プロビジョニングされた AWS Glue ワーカーを最大限に活用することです。そして、さらにパフォーマンスを向上させる必要がある場合は、DPU 数を増加してジョブをスケールアウトすることをお勧めします。

ETL コネクタを使用する場合、dynamodb.splits パラメータを使用して DynamoDB 読み取りオペレーションの並列処理を変更できます。　エクスポートコネクタで読み込みをする場合は、Spark エグゼキュータの並列処理の分割数を構成する必要がありません。DynamoDB 書き込みオペレーションの並列処理は、dynamodb.output.numParallelTasks を使用して変更できます。

DynamoDB ETL コネクターによる読み取り

dynamodb.splitsジョブ構成に設定されている最大ワーカー数と以下 numSlots の計算に基づいて計算することをお勧めします。自動スケーリングの場合、実際に利用可能なワーカー数は、その上限に基づいて変更される可能性があります。最大ワーカー数の設定についての詳細は、「the section called “Spark ジョブプロパティの構成”」の「[ワーカー数] (NumberOfWorkers)」を参照してください。

- numExecutors = NumberOfWorkers - 1

  1 つのエグゼキューターが Spark ドライバー専用になっている場合、他のエグゼキューターはデータの処理に使用されます。

- numSlotsPerExecutor =

  AWS Glue 3.0 and later versions

  - WorkerType が G.1X の場合は 4
  - WorkerType が G.2X の場合は 8
  - WorkerType が G.4X の場合は 16
  - WorkerType が G.8X の場合は 32

  AWS Glue 2.0 and legacy versions

  - WorkerType が G.1X の場合は 8
  - WorkerType が G.2X の場合は 16

- numSlots = numSlotsPerExecutor * numExecutors

dynamodb.splits を使用可能なスロット数、numSlots に設定することをおすすめします。

DynamoDB への書き込み

この dynamodb.output.numParallelTasks パラメータは、以下の計算を使用して Spark タスクごとの WCU を決定するために使用されます。

```
permittedWcuPerTask = ( TableWCU * dynamodb.throughput.write.percent ) /
dynamodb.output.numParallelTasks
```

DynamoDB ライターは、設定が DynamoDB に書き込まれる Spark タスクの数を正確に表している場合に最適に機能します。場合によっては、書き込みパフォーマンスを向上させるために、デフォルトの計算をオーバーライドする必要がある場合があります。このパラメータを指定しない場合、Spark タスクごとに許可される WCU は、次の式により自動的に計算されます。

-
  - `numPartitions = dynamicframe.getNumPartitions()`
  - `numSlots` (このセクションすでに定義したとおり)
  - `numParallelTasks = min(numPartitions, numSlots)`
- 例 1。DPU=10、WorkerType=Standard。入力 DynamicFrame には、100 個の RDD パーティションがあります。
  - `numPartitions = 100`
  - `numExecutors = (10 - 1) * 2 - 1 = 17`
  - `numSlots = 4 * 17 = 68`
  - `numParallelTasks = min(100, 68) = 68`
- 例 2。DPU=10、WorkerType=Standard。入力 DynamicFrame には、20 個の RDD パーティションがあります。
  - `numPartitions = 20`
  - `numExecutors = (10 - 1) * 2 - 1 = 17`
  - `numSlots = 4 * 17 = 68`
  - `numParallelTasks = min(20, 68) = 20`

> **Note**
>
> 従来の AWS Glue バージョンのジョブと Standard Worker を使用するジョブでは、スロット数を計算するために違った方法が必要となります。これらのジョブのパフォーマンスを調整

する必要がある場合は、サポートされている AWS Glue バージョンに移行することをおすすめします。

DynamoDB 接続オプションのリファレンス

Amazon DynamoDB への接続を指定します。

接続オプションは、ソース接続とシンク接続とで異なります。

"connectionType": ソースとして ETL コネクタを使用する "dynamodb"

AWS Glue DynamoDB ETL コネクタを使用している場合は、`"connectionType": "dynamodb"` をソースとして次の接続オプションを使用します。

- `"dynamodb.input.tableName"`: (必須) 読み取り元の DynamoDB テーブル。
- `"dynamodb.throughput.read.percent"`: (オプション) 使用する読み込みキャパシティーユニット (RCU) の割合。デフォルトでは、"0.5" に設定されます。許容値は "0.1" から "1.5" (これらの値を含む) です。
  - `0.5` ではデフォルトの読み込み速度を表し、AWS Glue はテーブルの読み込み容量の半分を消費しようとすることを意味します。上記の値を `0.5` より上に設定すると、AWS Glue は読み取りのリクエストレートを増加させ、`0.5` より低くした場合はそのリクエストレートを減少させます。(実際の読み込みレートは、DynamoDB テーブルに、統一ディストリビューションのキーがあるかどうかなどの要因によって変わります。)
  - DynamoDB テーブルがオンデマンドモードの場合、AWS Glue はテーブルの読み取り容量を 40000 として処理します。大きなテーブルをエクスポートする場合は、DynamoDB テーブルをオンデマンドモードに切り替えることをお勧めします。
- `"dynamodb.splits"`: (オプション) 読み取り中にこの DynamoDB テーブルを分割するパーティションの数を定義します。デフォルトでは、"1" に設定されます。許容値は "1" から "1,000,000" (これらの値を含む) です。

  1 は並列処理がないことを表します。より良いパフォーマンスを得るためには、より大きな値を (以下の式を使用して) 指定することを強くお勧めします。値を適切に設定する方法の詳細は、「the section called “DynamoDB 並列処理”」を参照してください。
- `"dynamodb.sts.roleArn"`: (オプション) クロスアカウントアクセスのために引き受ける IAM ロール の ARN。このパラメータは、AWS Glue 1.0 以降で使用可能です。

- "dynamodb.sts.roleSessionName": (任意) STS セッション名。デフォルトでは、「glue-dynamodb-read-sts-session」に設定されています。このパラメータは、AWS Glue 1.0 以降で使用可能です。

"connectionType" : AWS Glue DynamoDB エクスポートコネクタをソースとする "dynamodb"

AWS Glue バージョン 2.0 以降でのみ使用可能な AWS Glue DynamoDB エクスポートコネクタを使用する場合は、ソースとして "connectionType": "dynamodb" で次の接続オプションを使用します。

- "dynamodb.export": (必須) 文字列値:
  - ddb に設定すると、AWS Glue DynamoDB エクスポートコネクタが有効になり、AWS Glue ジョブ中に新しい ExportTableToPointInTimeRequest が呼び出されます。dynamodb.s3.bucket と dynamodb.s3.prefix から渡された場所で新しいエクスポートが生成されます。
  - s3 に設定すると、AWS Glue DynamoDB エクスポートコネクタが有効になりますが、新しい DynamoDB エクスポートの作成はスキップされ、代わりに dynamodb.s3.bucket と dynamodb.s3.prefix がそのテーブルの過去のエクスポートの Amazon S3 ロケーションとして使用されます。
- "dynamodb.tableArn": (必須) 読み取り元の DynamoDB テーブル。
- "dynamodb.unnestDDBJson": (オプション) デフォルト: false。 有効な値: ブール値 true に設定すると、エクスポートに存在する DynamoDB JSON 構造体のネスト解除の変換が実行されます。"dynamodb.unnestDDBJson" と "dynamodb.simplifyDDBJson" を同時に true に設定するとエラーになります。AWS Glue 3.0 以降のバージョンでは、DynamoDB マップタイプを簡略化するときの動作を改善するために "dynamodb.simplifyDDBJson" を使用することをおすすめします。詳細については、「the section called “DynamoDB エクスポート JSON の使用を簡素化”」を参照してください。
- "dynamodb.simplifyDDBJson": (オプション) デフォルト: false。 有効な値: ブール値 true に設定すると、エクスポートに存在する DynamoDB JSON 構造のスキーマを簡素化するための変換を実行します。これは "dynamodb.unnestDDBJson" オプションと同じ目的ですが、DynamoDB テーブル内の DynamoDB マップタイプやネストされたマップタイプをより適切にサポートします。このオプションは、AWS Glue バージョン 3.0 以降でのみ有効です。"dynamodb.unnestDDBJson" と "dynamodb.simplifyDDBJson" を同時に true に設定するとエラーになります。詳細については、「the section called “DynamoDB エクスポート JSON の使用を簡素化”」を参照してください。

- "dynamodb.s3.bucket": (オプション) DynamoDB ExportTableToPointInTime プロセスが実行される Simple Storage Service (Amazon S3) バケットの場所を示します。エクスポートファイル形式は DynamoDB JSON です。
  - "dynamodb.s3.prefix": (オプション) DynamoDB ExportTableToPointInTime の読み込みが保存される Simple Storage Service (Amazon S3) バケット内の Amazon S3 プレフィックスの場所を示します。dynamodb.s3.prefix と dynamodb.s3.bucket のどちらも指定しなかった場合、これらの値は AWS Glue ジョブの設定で指定された一時ディレクトリの場所がデフォルトとなります。詳細については、「AWS Glue で使用される特別なパラメータ」を参照してください。
  - "dynamodb.s3.bucketOwner": クロスアカウント Simple Storage Service (Amazon S3) アクセスのために必要なバケット所有者を示します。
- "dynamodb.sts.roleArn": (オプション) DynamoDB テーブルのクロスアカウントアクセスおよび/またはクロスリージョンアクセスのために割り当てる IAM ロール の ARN。注: 同じ IAM ロール のARN を使用して、ExportTableToPointInTime リクエストに指定された Simple Storage Service (Amazon S3) の場所にアクセスします。
- "dynamodb.sts.roleSessionName": (任意) STS セッション名。デフォルトでは、「glue-dynamodb-read-sts-session」に設定されています。
- "dynamodb.exportTime" (オプション) 有効な値: ISO-8601 インスタントを表す文字列。 エクスポートが実行されるべき時点。
- "dynamodb.sts.region": (リージョンのエンドポイントを使用してリージョン間通話を行う場合は必須) 読み取りたい DynamoDB テーブルをホストするリージョン。

"connectionType": ETLコネクタをシンクとして使用する "dynamodb"

"connectionType": "dynamodb" をシンクとして、次の接続オプションを使用します。

- "dynamodb.output.tableName": (必須) 書き込み先の DynamoDB テーブル。
- "dynamodb.throughput.write.percent": (オプション) 使用する書き込みキャパシティーユニット (WCU) の割合。デフォルトでは、"0.5" に設定されます。許容値は "0.1" から "1.5" (これらの値を含む) です。
  - 0.5 ではデフォルトの読み込み速度を表し、AWS Glue はテーブルの書き込み容量の半分を消費しようとすることを意味します。上記の値を 0.5 より上に設定すると、AWS Glue は書き込みリクエストレートを増加させ、0.5 より低くした場合はそのリクエストレートを減少させます。(実際の書き込みレートは、統一されたキーのディストリビューションが、DynamoDB テーブルにあるかどうかなどの要因によって変わります)。

  - DynamoDB テーブルがオンデマンドモードの場合、AWS Glue はテーブルの書き込み容量を 40000 として処理します。大きなテーブルをインポートする場合は、DynamoDB テーブルをオンデマンドモードに切り替えることをお勧めします。
- "dynamodb.output.numParallelTasks": (オプション) 同時に DynamoDB に書き込める並列タスクの数を定義します。Spark タスクごとに許容される WCU を計算するために使用されます。ほとんどの場合、AWS Glue はこの値の妥当なデフォルト値を計算します。 詳細については、「the section called “DynamoDB 並列処理”」を参照してください。
- "dynamodb.output.retry": (オプション) DynamoDB から ProvisionedThroughputExceededException が送られている場合の、再試行の実行回数を定義します。デフォルトでは、"10" に設定されています。
- "dynamodb.sts.roleArn": (オプション) クロスアカウントアクセスのために引き受ける IAM ロール の ARN。
- "dynamodb.sts.roleSessionName": (任意) STS セッション名。デフォルトでは、「glue-dynamodb-write-sts-session」に設定されています。

DynamoDB テーブルへのクロスアカウントおよびクロスリージョンでのアクセス

AWS Glue ETL ジョブでは、DynamoDB テーブルに対する、クロスリージョンならびにクロスアカウントでのアクセスを供にサポートしています。AWS GlueETL ジョブは、別の AWS アカウントの DynamoDB テーブルからのデータ読み取り、および別の AWS アカウントの DynamoDB テーブルへのデータ書き込みをサポートしています。加えて AWS Glue では、別のリージョンの DynamoDB テーブルからの読み取りと、別のリージョンの DynamoDB テーブルへの書き込みもサポートしています。このセクションでは、このアクセスを設定する手順と、スクリプトの例を示します。

このセクションに示す手順では、IAM のチュートリアルに則り、IAM ロールを作成しそのロールにアクセス許可を付与しています。このチュートリアルでは、ロールの引き受けについても解説されていますが、今回はその代わりに、AWS Glue でのロールの引き受けにはジョブ・スクリプトを使用します。また、このチュートリアルでは、クロスアカウントに関連した一般的な作業についても説明しています。詳しくは、IAM ユーザーガイドの「チュートリアル: IAMロールを使用した AWS アカウント間のアクセス権の委譲」を参照してください。

ロールを作成する

チュートリアルのステップ 1 に従い、アカウント A に IAM ロールを作成します。ロールのアクセス許可を定義する際に、AmazonDynamoDBReadOnlyAccess または AmazonDynamoDBFullAccess など既存のポリシーをアタッチして、ロールに対し DynamoDB の読み取り/書き込みを許

可します。次の例に、アクセス許可ポリシー AmazonDynamoDBFullAccess を使用する、DynamoDBCrossAccessRole という名前のロールを作成する方法を示します。

ロールにアクセス許可を付与する

IAM ユーザーガイドにあるチュートリアルのステップ 2 に従い、アカウント B が新しく作成されたロールに切り替えることを許可します。次の例では、ここで示されるステートメントを使用して新しいポリシーを作成しています。

```
{
    "Version": "2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": "sts:AssumeRole",
        "Resource": "<DynamoDBCrossAccessRole's ARN>"
    }
}
```

次に、DynamoDB へのアクセスに使用するグループ/ロール/ユーザーに、このポリシーをアタッチします。

AWS Glue Job スクリプトの中でロールを引き受ける

この段階で、アカウント B にログインして、AWS Glue ジョブを作成できるようになっています。ジョブを作成するには、AWS Glue で Spark ジョブに関するジョブプロパティの構成 にある手順を参照してください。

ジョブスクリプト内で DynamoDBCrossAccessRole ロールを引き受けるには、dynamodb.sts.roleArn パラメーターを使用する必要があります。このロールを引き受けることで、アカウント B の DynamoDB にアクセスするために必要な、一時的な認証情報を取得できます。このためのサンプルスクリプトを示します。

リージョン間でのクロスアカウント読み取りの場合 (ETL コネクタ):

```
import sys
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job
from awsglue.utils import getResolvedOptions

args = getResolvedOptions(sys.argv, ["JOB_NAME"])
```

```
glue_context= GlueContext(SparkContext.getOrCreate())
job = Job(glue_context)
job.init(args["JOB_NAME"], args)

dyf = glue_context.create_dynamic_frame_from_options(
    connection_type="dynamodb",
    connection_options={
    "dynamodb.region": "us-east-1",
    "dynamodb.input.tableName": "test_source",
    "dynamodb.sts.roleArn": "<DynamoDBCrossAccessRole's ARN>"
    }
)
dyf.show()
job.commit()
```

リージョン間でのクロスアカウント読み取りの場合 (ETL コネクタ):

```
import sys
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job
from awsglue.utils import getResolvedOptions

args = getResolvedOptions(sys.argv, ["JOB_NAME"])
glue_context= GlueContext(SparkContext.getOrCreate())
job = Job(glue_context)
job.init(args["JOB_NAME"], args)

dyf = glue_context.create_dynamic_frame_from_options(
    connection_type="dynamodb",
    connection_options={
        "dynamodb.export": "ddb",
        "dynamodb.tableArn": "<test_source ARN>",
        "dynamodb.sts.roleArn": "<DynamoDBCrossAccessRole's ARN>"
    }
)
dyf.show()
job.commit()
```

リージョンにまたがるクロスアカウント読み取りおよび書き込みの場合:

```
import sys
from pyspark.context import SparkContext
```

```
from awsglue.context import GlueContext
from awsglue.job import Job
from awsglue.utils import getResolvedOptions

args = getResolvedOptions(sys.argv, ["JOB_NAME"])
glue_context= GlueContext(SparkContext.getOrCreate())
job = Job(glue_context)
job.init(args["JOB_NAME"], args)

dyf = glue_context.create_dynamic_frame_from_options(
    connection_type="dynamodb",
    connection_options={
        "dynamodb.region": "us-east-1",
        "dynamodb.input.tableName": "test_source"
    }
)
dyf.show()

glue_context.write_dynamic_frame_from_options(
    frame=dyf,
    connection_type="dynamodb",
    connection_options={
        "dynamodb.region": "us-west-2",
        "dynamodb.output.tableName": "test_sink",
        "dynamodb.sts.roleArn": "<DynamoDBCrossAccessRole's ARN>"
    }
)

job.commit()
```

Kinesis 接続

Kinesis 接続を使用すると、データカタログ テーブルに保存されている情報を使用して、またはデータ ストリームに直接アクセスするための情報を提供することで、Amazon Kinesis データ ストリームの読み取りと書き込みを行うことができます。Kinesis から Spark DataFrame に情報を読み取り、それを AWS Glue DynamicFrame に変換できます。DynamicFrames は JSON 形式で Kinesis に書き込むことができます。データストリームに直接アクセスする場合は、これらのオプションを使用して、データストリームへのアクセス方法に関する情報を提供します。

`getCatalogSource` または `create_data_frame_from_catalog` を使用して Kinesis ストリーミングソースからレコードを消費する場合、ジョブはデータカタログデータベースとテーブル名の情報を持っており、それを使用して Kinesis ストリー

ミングソースから読み込むためのいくつかの基本パラメータを取得することができます。getSource、getSourceWithFormat、createDataFrameFromOptions、または create_data_frame_from_options を使用している場合、ここで説明する接続オプションを使用して、これらの基本パラメータを指定する必要があります。

Kinesis の接続オプションは、GlueContext クラス内の指定されたメソッドに対して以下の引数で指定することができます。

- Scala
  - connectionOptions: getSource、createDataFrameFromOptions、getSink で使用
  - additionalOptions: getCatalogSource、getCatalogSink で使用
  - options: getSourceWithFormat、getSinkWithFormat で使用
- Python
  - connection_options: create_data_frame_from_options、write_dynamic_frame_from_options で使用
  - additional_options: create_data_frame_from_catalog、write_dynamic_frame_from_catalog で使用
  - options: getSource、getSink で使用

ストリーミング ETL ジョブに関する注意事項と制限事項については、「the section called “ストリーミング ETL に関する注意と制限”」を参照してください。

Kinesis の設定

AWS Glue Spark ジョブで Kinesis データストリームに接続するには、いくつかの前提条件が必要です。

- 読み取る場合、AWS Glue ジョブには、Kinesis データストリームへの読み取りアクセスレベルの IAM 権限が必要です。
- 書き込みの場合、AWS Glue ジョブには、Kinesis データストリームへの書き込みアクセスレベルの IAM 権限が必要です。

場合によっては、追加の前提条件を設定する必要があります。

- AWS Glue ジョブが (通常は他のデータセットに接続するための) 追加のネットワーク接続で設定されていて、その接続のいずれかが Amazon VPC ネットワークオプションを提供している場合、

ジョブは Amazon VPC 経由で通信するように指示されます。この場合、Amazon VPC を介して通信するように Kinesis データストリームを設定する必要もあります。そのためには、Amazon VPC と Kinesis データストリームの間にインターフェイス VPC エンドポイントを作成します。詳細については、「インターフェイス VPC エンドポイントと Amazon Kinesis Data Streams の使用」を参照してください。

- 別のアカウントで Amazon Kinesis Data Streams を指定する場合は、クロスアカウントアクセスを許可するようにロールとポリシーを設定する必要があります。詳細については、「Example: Read From a Kinesis Stream in a Different Account」を参照してください。

ストリーミング ETL ジョブの前提条件の詳細については、「the section called “ストリーミング ETL ジョブ”」を参照してください。

例: Kinesis ストリームからの読み込み

例: Kinesis ストリームからの読み込み

the section called “forEachBatch” と組み合わせて使用します。

Amazon Kinesis ストリーミングソースの例:

```
kinesis_options =
   { "streamARN": "arn:aws:kinesis:us-east-2:777788889999:stream/fromOptionsStream",
     "startingPosition": "TRIM_HORIZON",
     "inferSchema": "true",
     "classification": "json"
   }
data_frame_datasource0 =
 glueContext.create_data_frame.from_options(connection_type="kinesis",
 connection_options=kinesis_options)
```

例: Kinesis ストリームへの書き込み

例: Kinesis ストリームからの読み込み

the section called “forEachBatch” と組み合わせて使用します。

Amazon Kinesis ストリーミングソースの例:

```
kinesis_options =
   { "streamARN": "arn:aws:kinesis:us-east-2:777788889999:stream/fromOptionsStream",
```

```
    "startingPosition": "TRIM_HORIZON",
    "inferSchema": "true",
    "classification": "json"
  }
data_frame_datasource0 =
 glueContext.create_data_frame.from_options(connection_type="kinesis",
 connection_options=kinesis_options)
```

Kinesis 接続オプションのリファレンス

Amazon Kinesis Data Streams への接続を指定します。

Kinesis ストリーミングデータソースには、次の接続オプションを使用します。

- "streamARN" (必須) 読み取り/書き込みに使用されます。Kinesis データストリームの ARN。
- "classification" (読み取りに必須) 読み取りで使用。レコード内のデータで使用されるファイル形式。データカタログを通じて提供されていない限り、必須です。
- "streamName" - (オプション) 読み取りで使用。読み取り対象/読み取り元の Kinesis データストリームの名前。endpointUrl で使用。
- "endpointUrl" - (オプション) 読み取りで使用。デフォルト: "https://kinesis.us-east-1.amazonaws.com"。Kinesis エンドポイントの AWS エンドポイント。特別なリージョンに接続する場合を除き、これを変更する必要はありません。
- "partitionKey" - (オプション) 書き込みに使用。レコードを作成する際に使用される Kinesis パーティションキー。
- "delimiter" (オプション) 読み取りに使用。classification が CSV の場合に使用される値の区切り文字。デフォルトは「,」です。
- "startingPosition": (オプション) 読み込みに使用。Kinesis データストリーム内の、データの読み取り開始位置。指定できる値は "latest"、"trim_horizon"、"earliest"、または UTC 形式のタイムスタンプ文字列であり、この文字列のパターンは yyyy-mm-ddTHH:MM:SSZ です (Z は UTC タイムゾーンのオフセットを +/- で表します。例:「2023-04-04T08:00:00-04:00」)。デフォルト値は "latest" です。注意: "startingPosition" の UTC 形式のタイムスタンプ文字列は AWS Glue バージョン 4.0 以降のみでサポートされます。
- "failOnDataLoss": (オプション) アクティブなシャードがないか、有効期限が切れている場合、ジョブは失敗します。デフォルト値は "false" です。
- "awsSTSRoleARN": (オプション) 読み取り/書き込みに使用。AWS Security Token Service (AWS STS) を使用して担うロールの Amazon リソースネーム (ARN)。このロールには、Kinesis

データストリームのレコードの説明操作または読み取り操作の権限が必要です。このパラメーターは、別のアカウントのデータストリームにアクセスするときに使用する必要があります。"awsSTSSessionName" と組み合わせて使用します。

- "awsSTSSessionName": (オプション) 読み取り/書き込みに使用。AWS STS を使って、ロールを担うセッションの識別子。このパラメーターは、別のアカウントのデータストリームにアクセスするときに使用する必要があります。"awsSTSRoleARN" と組み合わせて使用します。
- "awsSTSEndpoint": (オプション) 引き受けたロールで Kinesis に接続するときに使用する AWS STS エンドポイント。これにより、デフォルトのグローバルエンドポイントでは不可能な VPC 内のリージョナル AWS STS エンドポイントを使用できます。
- "maxFetchTimeInMs": (オプション) 読み込みに使用。ジョブエグゼキューターが Kinesis データストリームから現在のバッチのレコードを読み取るために費やした最大時間は、ミリ秒 (ms) 単位で指定されます。この時間内に複数の GetRecords API コールを行うことができます。デフォルト値は 1000 です。
- "maxFetchRecordsPerShard": (オプション) 読み込みに使用。1 マイクロバッチ当たりに Kinesis データストリームでシャードごとにフェッチするレコードの最大数。メモ: ストリーミングジョブが既に Kinesis (同じ get-records 呼び出しで) から余分なレコードを読み取っている場合、クライアントはこの制限を超えることができます。maxFetchRecordsPerShard が厳密である必要がある場合、maxRecordPerRead の倍数にする必要があります。デフォルト値は 100000 です。
- "maxRecordPerRead": (オプション) 読み込みに使用。getRecords オペレーションごとに、Kinesis データストリームからフェッチするレコードの最大数。デフォルト値は 10000 です。
- "addIdleTimeBetweenReads": (オプション) 読み込みに使用。2 つの連続する getRecords オペレーション間の遅延時間を追加します。デフォルト値は "False" です。このオプションは、Glue バージョン 2.0 以降でのみ設定可能です。
- "idleTimeBetweenReadsInMs": (オプション) 読み込みに使用。2 つの連続する getRecords オペレーション間での、最短の遅延時間 (ミリ秒単位で指定)。デフォルト値は 1000 です。このオプションは、Glue バージョン 2.0 以降でのみ設定可能です。
- "describeShardInterval": (オプション) 読み込みに使用。スクリプトが呼び出す 2 つの ListShards API 間での、再シャーディングを考慮すべき最小時間。詳細については、Amazon Kinesis Data Streams デベロッパーガイドの「リシャーディングのための戦略」を参照してください。デフォルト値は 1s です。
- "numRetries": (オプション) 読み込みに使用。Kinesis Data Streams API リクエストを再試行する最大の回数。デフォルト値は 3 です。

- "retryIntervalMs": (オプション) 読み込みに使用。Kinesis Data Streams API 呼び出しを再試行するまでのクールオフ期間 (ミリ秒単位で指定)。デフォルト値は 1000 です。
- "maxRetryIntervalMs": (オプション) 読み込みに使用。再試行で 2 つの Kinesis Data Streams API を呼び出す間の最大クールオフ期間 (ミリ秒単位で指定)。デフォルト値は 10000 です。
- "avoidEmptyBatches": (オプション) 読み込みに使用。バッチ処理を開始する前に、Kinesis データストリームで未読のデータをチェックすることで、空のマイクロバッチジョブを作成しないようにします。デフォルト値は "False" です。
- "schema": (inferSchema に false を設定した場合は必須) 読み取りに使用。ペイロードの処理に使用するスキーマ。分類が avro である場合、提供されるスキーマには Avro スキーマ形式を使用する必要があります。分類が avro 以外の場合、提供されるスキーマは DDL スキーマ形式である必要があります。

  以下に、スキーマの例を示します。

  Example in DDL schema format

  ```
  `column1` INT, `column2` STRING , `column3` FLOAT
  ```

  Example in Avro schema format

  ```
  {
    "type":"array",
    "items":
    {
      "type":"record",
      "name":"test",
      "fields":
      [
        {
          "name":"_id",
          "type":"string"
        },
        {
          "name":"index",
          "type":
          [
            "int",
            "string",
            "float"
          ]
        }
  ```

```
    ]
  }
}
```

- "inferSchema": (オプション) 読み込みに使用。デフォルト値は、「false」です。「true」に設定すると、実行時に、スキーマが foreachbatch 内のペイロードから検出されます。
- "avroSchema": (非推奨) 読み取りに使用。Avro 形式を使用する場合に、Avro データのスキーマを指定するために使用されるパラメータです。このパラメータは非推奨となりました。schema パラメータを使用します。
- "addRecordTimestamp": (オプション) 読み込みに使用。このオプションが「true」に設定されている場合、データ出力には、対応するレコードがストリームによって受信された時刻を表示する「__src_timestamp」という名前が付けられた追加の列が含まれます。デフォルト値は、「false」です。このオプションは AWS Glue バージョン 4.0 以降でサポートされています。
- "emitConsumerLagMetrics": (オプション) 読み込みに使用。このオプションを「true」に設定すると、バッチごとに、ストリームが受信した最も古いレコードと、それが AWS Glue で CloudWatch に到着した時間との間隔のメトリクスが出力されます。メトリクスの名前は「glue.driver.streaming.maxConsumerLagInMs」です。デフォルト値は、「false」です。このオプションは AWS Glue バージョン 4.0 以降でサポートされています。
- "fanoutConsumerARN": (オプション) 読み込みに使用。streamARN で指定されたストリームの Kinesis ストリームコンシューマの ARN。Kinesis 接続の拡張ファンアウトモードを有効にするために使用されます。拡張ファンアウトが使用された Kinesis ストリームの使用に関する詳細については、「the section called “Kinesis ストリーミングジョブでの拡張ファンアウトの使用”」を参照してください。
- "recordMaxBufferedTime" - (オプション) 書き込みに使用。デフォルト: 1000 (ミリ秒)。レコードが書き込まれるのを待っている間にバッファリングされる最大時間。
- "aggregationEnabled" - (オプション) 書き込みに使用。デフォルト: true。Kinesis に送信する前にレコードを集約するかどうかを指定します。
- "aggregationMaxSize" - (オプション) 書き込みに使用。デフォルト: 51200 (バイト) レコードがこの制限よりも大きい場合、そのレコードはアグリゲータをバイパスします。注: Kinesis では、レコードサイズに 50 KB の制限が適用されます。これを 50 KB を超えて設定すると、サイズ超過のレコードは Kinesis によって拒否されます。
- "aggregationMaxCount" - (オプション) 書き込みに使用。デフォルト: 4294967295。集計されたレコードにパックされる項目の最大数。

- "producerRateLimit" - (オプション) 書き込みに使用。デフォルト: 150 (%)。1 つのプロデューサー (ジョブなど) からの送信されるシャード単位のスループットをバックエンド制限のパーセンテージとして制限できます。
- "collectionMaxCount" - (オプション) 書き込みに使用。デフォルト: 500。PutRecords レコードにパックされる項目の最大数。
- "collectionMaxSize" - (オプション) 書き込みに使用。デフォルト: 5242880 (バイト)。PutRecords リクエストで送信するデータの最大量。

Kinesis ストリーミングジョブでの拡張ファンアウトの使用

拡張ファンアウトコンシューマは、通常のコンシューマよりも高い専用スループットで Kinesis ストリームからレコードを受信できます。これは、ジョブなどの Kinesis コンシューマにデータを提供するために使用される転送プロトコルを最適化することによって行われます。Kinesis 拡張ファンアウトの詳細については、Kinesis のドキュメントを参照してください。

拡張ファンアウトモードでは、maxRecordPerRead および idleTimeBetweenReadsInMs 接続オプションは適用されなくなりました。拡張ファンアウトを使用する場合、これらのパラメータは設定できないためです。リトライの設定オプションは説明どおりに機能します。

以下の手順に従って、ストリーミングジョブの拡張ファンアウトを有効または無効にします。ストリームのデータを消費するジョブごとに、ストリームコンシューマを登録する必要があります。

ジョブの拡張ファンアウト消費を有効にするには:

1. Kinesis API を使用してジョブのストリームコンシューマを登録します。手順に従って、Kinesis ドキュメントの Kinesis Data Streams API を使用して、拡張ファンアウトでコンシューマを登録します。必要なのは、最初のステップである RegisterStreamConsumer を呼び出すことだけです。リクエストは ARN、*consumerARN* を返す必要があります。
2. 接続メソッドの引数で接続オプション fanoutConsumerARN を *consumerARN* に設定します。
3. ジョブを再開してください。

ジョブの拡張ファンアウト消費を無効にするには:

1. メソッド呼び出しから接続オプション fanoutConsumerARN を削除します。
2. ジョブを再開してください。
3. Kinesis ドキュメントの指示に従ってコンシューマを登録解除します。これらの手順はコンソールに適用されますが、Kinesis API を使用して実行することもできます。Kinesis

API によるストリームコンシューマの登録解除の詳細については、Kinesis ドキュメントの「DeregisterStreamConsumer」を参照してください。

Amazon S3 接続

AWS Glue for Spark を使用して Amazon S3 内のファイルの読み込みと書き込みを行うことができます。AWSGlue for Spark は、CSV、Avro、JSON、Orc、Parquet など、Amazon S3 に保存されている多くの一般的なデータ形式をそのままでサポートしています。サポートされるデータ形式の詳細については、「the section called “データ形式に関するオプション”」を参照してください。各データ形式が、異なる AWS Glue の機能のセットをサポートする場合があります。機能のサポートの詳細については、使用しているデータ形式のページを参照してください。さらに、Hudi、Iceberg、Delta Lake のデータレイクフレームワークに保存されているバージョン対応ファイルを読み書きできます。データレイクフレームワークの詳細については、「the section called “データレイクフレームワーク”」を参照してください。

AWS Glue を使用すると、書き込み中に Amazon S3 オブジェクトをフォルダ構造に分割し、簡単な設定でパーティションごとに取得してパフォーマンスを向上させることができます。また、データを変換する際に小さなファイルをまとめてグループ化するように設定してパフォーマンスを向上させることもできます。Amazon S3 では、`bzip2` の読み込み、書き込み、`gzip` アーカイブを行うことができます。

トピック

- S3 接続の設定
- Amazon S3 接続のオプションのリファレンス
- 廃止されたデータ形式の接続構文
- Amazon S3 ストレージクラスの除外
- AWS Glue での ETL 出力のパーティションの管理
- 大きなグループの入力ファイルの読み取り
- Amazon S3 用の VPC エンドポイント

S3 接続の設定

AWS Glue with Spark ジョブで Amazon S3 に接続するには、いくつかの前提条件が必要です。

- AWS Glue ジョブには、関連する Amazon S3 バケットに対する IAM 権限が必要です。

場合によっては、追加の前提条件を設定する必要があります。

- クロスアカウントアクセスを設定するときは、Amazon S3 バケットで適切なアクセス制御を行います。
- セキュリティ上の理由から、Amazon S3 リクエストを Amazon VPC 経由でルーティングすることもできます。このアプローチでは、帯域幅や可用性の課題が生じる場合があります。詳細については、「the section called “Amazon S3 用の VPC エンドポイント”」を参照してください。

Amazon S3 接続のオプションのリファレンス

Simple Storage Service (Amazon S3) への接続を指定します。

Amazon S3 はテーブルではなくファイルを管理するため、このドキュメントに記載されている接続プロパティを指定することに加えて、ファイルタイプに関する追加の設定を指定する必要があります。この情報はデータ形式オプションを使用して指定します。形式オプションの詳細については、「the section called “データ形式に関するオプション”」を参照してください。AWS Glue データカタログと統合してこの情報を指定することもできます。

接続オプションとフォーマットオプションの違いの例として、the section called “create_dynamic_frame_from_options” メソッドが `connection_type`、`connection_options`、`format` および `format_options` をどのように使用するかを考えてみます。このセクションでは、`connection_options` に提供されるパラメータについて具体的に説明します。

`"connectionType": "s3"` では、次の接続オプションを使用します。

- `"paths"`: (必須) 読み取りのソースとなる Amazon S3 パスのリスト。
- `"exclusions"`: (オプション) 除外する Unix スタイルの glob パターンの JSON リストを含む文字列。例えば、`"[\"**.pdf\"]"` はすべての PDF ファイルを除外します。AWS Glue がサポートする glob 構文の詳細については、「包含パターンと除外パターンを使用する」を参照してください。
- `"compressionType"`: または「`compression`」: (オプション) データの圧縮方法を指定します。Simple Storage Service (Amazon S3) ソース用には `"compressionType"` を、Simple Storage Service (Amazon S3) ターゲット用には `"compression"` を使用します。データに標準のファイル拡張子が付いている場合、このオプションは一般的に不要です。指定できる値は `"gzip"` および `"bzip2"` です。特定の形式では、他の圧縮形式がサポートされている場合もあります。機能のサポートの詳細については、データ形式のページを参照してください。

- "groupFiles": (オプション) 入力ファイルが 50,000 個を超える場合、デフォルトでファイルのグループ化が有効化されます。入力ファイルが 50,000 個未満の場合にグループ化を有効化するには、このパラメータに "inPartition" を設定します。入力ファイルが 50,000 個を超える場合に、グループ化を無効にするには、このパラメータを "none" に設定します。
- "groupSize": (オプション) ターゲットグループのサイズ (バイト単位)。デフォルトは、入力データのサイズとクラスターのサイズに基づいて計算されます。入力ファイルが 50,000 個未満の場合、このオプションを有効にするには、"groupFiles" を "inPartition" に設定する必要があります。
- "recurse": (オプション) true に設定した場合は、指定したパスの下にあるすべてのサブディレクトリ内のファイルを再帰的に読み取ります。
- "maxBand": (オプション、詳細設定) このオプションでは、s3 リストの一貫性が認められるまでの期間をミリ秒単位で指定します。Amazon S3 の結果整合性を担保するために、直前の maxBand ミリ秒以内の変更タイムスタンプが付いたファイルが、特に JobBookmarks の使用時に追跡されます。ほとんどのユーザーはこのオプションを設定する必要はありません。デフォルトは 900000 ミリ秒 (15 分) です。
- "maxFilesInBand": (オプション、詳細設定) このオプションは、直前の maxBand 秒間に保存するファイルの最大数を指定します。この数を超えた場合、余分なファイルはスキップされ、次のジョブ実行時にのみ処理されます。ほとんどのユーザーはこのオプションを設定する必要はありません。
- "isFailFast": (オプション) このオプションにより、AWS Glue ETL ジョブがリーダー解析の例外をスローするかどうかを決定します。true に設定すると、Spark タスクが 4 回の再試行のうちにデータを正しく解析できなかった場合、ジョブは速やかに失敗します。
- "catalogPartitionPredicate": (オプション) 読み込みに使用。SQL WHERE 句の内容。非常に多数のパーティションを含むデータカタログテーブルから読み込むときに使用されます。データカタログインデックスから一致するパーティションを取得します。the section called “create_dynamic_frame_from_catalog” メソッド (および他の類似メソッド) のオプション push_down_predicate と共に使用します。詳細については、「the section called “カタログパーティション述語”」を参照してください。
- "partitionKeys": (オプション) 書き込みに使用。列ラベル文字列の配列。AWSGlue は、この設定で指定されたとおりにデータを分割します。詳細については、「the section called “パーティションの書き込み”」を参照してください。
- "excludeStorageClasses": (オプション) 読み込みに使用。Amazon S3 ストレージクラスを指定する文字列の配列。AWSGlue は、この設定に基づいて Amazon S3 オブジェクトを除外します。詳細については、「the section called “Amazon S3 ストレージクラスの除外”」を参照してください。

廃止されたデータ形式の接続構文

特定のデータ形式には、特定の接続タイプ構文を使用してアクセスできます。この構文は廃止されました。代わりに the section called “データ形式に関するオプション” で提供されている s3 接続タイプと形式オプションを使用して形式を指定することをお勧めします。

"connectionType": "orc"

Apache Hive Optimized Row Columnar (ORC) ファイル形式で、Amazon S3 に保存されるファイルへの接続を指定します。

`"connectionType": "orc"` では、次の接続オプションを使用します。

- `paths`: (必須) 読み取りのソースとなる Amazon S3 パスのリスト。
- (その他のオプション名/値ペア): 書式設定オプションなどのその他のオプションはすべて SparkSQL `DataSource` に直接渡されます。

"connectionType": "parquet"

Apache Parquet ファイル形式で、Amazon S3 に保存されるファイルへの接続を指定します。

`"connectionType": "parquet"` では、次の接続オプションを使用します。

- `paths`: (必須) 読み取りのソースとなる Amazon S3 パスのリスト。
- (その他のオプション名/値ペア): 書式設定オプションなどのその他のオプションはすべて SparkSQL `DataSource` に直接渡されます。

Amazon S3 ストレージクラスの除外

Amazon Simple Storage Service (Amazon S3) からファイルまたはパーティションを読み取る AWS Glue ETL ジョブを実行している場合は、一部の Amazon S3 ストレージクラスタイプを除外できます。

Amazon S3 には、次のストレージクラスがあります。

- `STANDARD` – 頻繁にアクセスされるデータの汎用ストレージ向け。
- `INTELLIGENT_TIERING` – アクセスパターンが不明または変化するデータ向け。
- `STANDARD_IA` および `ONEZONE_IA` – 保持期間が長くアクセス頻度の低いデータ向け。

- GLACIER、DEEP_ARCHIVE、および REDUCED_REDUNDANCY – 長期アーカイブおよびデジタル保存向け。

詳細については、Amazon S3 デベロッパーガイドの「Amazon S3 ストレージクラス」を参照してください。

このセクションの例では、GLACIER および DEEP_ARCHIVE ストレージクラスを除外する方法を示します。これらのクラスではファイルをリストできますが、復元されない限り、ファイルを読み取ることはできません (さらに詳細な情報については、Amazon S3 デベロッパーガイドの「Restoring Archived Objects」を参照してください。)

ストレージクラスの除外を使用することで、これらのストレージクラス層にまたがるパーティションを持つテーブルで AWS Glue ジョブを確実に動作させることができます。除外がない場合、これらの階層からデータを読み取るジョブは、次のエラーで失敗します。 AmazonS3Exception: オペレーションはオブジェクトのストレージクラスに対して有効ではありません。

AWS Glue で Amazon S3 ストレージクラスをフィルタリングするには、さまざまな方法があります 。

トピック

- 動的フレームの作成時の Amazon S3 ストレージクラスの除外
- Data Catalog テーブルでの Amazon S3 ストレージクラスの除外

動的フレームの作成時の Amazon S3 ストレージクラスの除外

ダイナミックフレームの作成時に Amazon S3 のストレージクラスを除外するには、excludeStorageClasses で additionalOptions を使います。AWS Glue では指定されたストレージクラスに対応するファイルを、独自の Amazon S3 Lister の実装を使用して自動的にリストアップし、除外します。

次に、動的フレームの作成時に GLACIER および DEEP_ARCHIVE ストレージクラスを除外する、Python および Scala の例を示します。

Python の例

```
glueContext.create_dynamic_frame.from_catalog(
    database = "my_database",
    tableName = "my_table_name",
    redshift_tmp_dir = "",
```

```
    transformation_ctx = "my_transformation_context",
    additional_options = {
        "excludeStorageClasses" : ["GLACIER", "DEEP_ARCHIVE"]
    }
)
```

Scala の例

```
val* *df = glueContext.getCatalogSource(
    nameSpace, tableName, "", "my_transformation_context",
    additionalOptions = JsonOptions(
        Map("excludeStorageClasses" -> List("GLACIER", "DEEP_ARCHIVE"))
    )
).getDynamicFrame()
```

Data Catalog テーブルでの Amazon S3 ストレージクラスの除外

AWS Glue ETL のジョブによるストレージクラスの除外は、AWS Glue Data Catalog のテーブルパラメータとして指定できます。このパラメータは、AWS Command Line Interface (AWS CLI) を使用するか、API を使用してプログラミングによって `CreateTable` オペレーションに含むことができます。詳細については、「テーブル構造」および「CreateTable」を参照してください。

AWS Glue コンソールで除外されるストレージクラスを指定することもできます。

Amazon S3 ストレージクラスを除外するには (コンソール)

1. AWS Management Consoleにサインインし、AWS Glue コンソール (https://console.aws.amazon.com/glue/) を開きます。
2. 左側のナビゲーションペインで、[Tables (テーブル)] を選択します。
3. リストでテーブル名を選択し、[Edit table (テーブルの編集)] を選択します。
4. [Table properties (テーブルプロパティ)] で、キーとして **`excludeStorageClasses`** を、値として **`[\"GLACIER\",\"DEEP_ARCHIVE\"]`** を追加します。
5. [Apply] (適用) を選択します。

AWS Glue での ETL 出力のパーティションの管理

パーティション分割は、データセットを整理して効率的にクエリを実行可能にする重要な手法です。1 つまたは複数の列の個別の値に基づいて、データを階層形式のディレクトリ構造に整理します。

たとえば、Amazon Simple Storage Service (Amazon S3) のアプリケーションログを、年、月、日で分類しながら、日付についてパーティション化できます。次に 1 日分のデータに対応するファイルを `s3://my_bucket/logs/year=2018/month=01/day=23/` などのプレフィックス別に配置します。Amazon Athena、Amazon Redshift Spectrum、そして AWS Glue などのシステムでは、基になるデータ全体を Amazon S3 から読み取ることなく、これらのパーティションを使用してパーティション値でデータをフィルタリングできます。

クローラは、ファイルタイプとスキーマを推定するだけでなく、AWS Glue Data Catalog を構成する際に、データセットのパーティション構造も自動的に特定します。これにより出力されるパーティション列に対しては、AWS Glue ETL ジョブやクエリエンジン (Amazon Athena など) からクエリを実行できます。

テーブルのクロールが完了すると、クローラが作成したパーティションを表示できます。AWS Glue コンソールの左のナビゲーションペインで、[Tables] (テーブル) をクリックします。クローラで作成されたテーブルを選択した後、[View Partitions] (パーティション) の表示をクリックします。

Apache Hive 形式のパーティション分割されたパス (`key=val` 形式) の場合、クローラはキー名を使用して自動的に列名を事前設定します。それ以外の場合は、`partition_0`、`partition_1` などのデフォルト名が使用されます。コンソールでデフォルトの名前を変更できます。そのためには、テーブルに移動します。[インデックス] タブにインデックスが存在するかどうかを確認します。ある場合は、削除してから続行する必要があります (後で新しい列名を使用して再作成できます)。次に、[スキーマの編集] を選択し、そこでパーティション列の名前を変更します。

次に、ETL スクリプトでパーティション列をフィルタリングできます。パーティション情報は Data Catalog に格納されるため、パーティション列を `DynamicFrame` に含めるには `from_catalog` API 呼び出しを使用します。例えば、`create_dynamic_frame.from_options` ではなく `create_dynamic_frame.from_catalog` を使用します。

パーティション化は、データスキャンを減らす最適化手法です。この手法が適切であることを特定するプロセスの詳細については、「AWS 規範的ガイダンス」の「Best practices for performance tuning AWS Glue for Apache Spark jobs」ガイドにある「Reduce the amount of data scan」を参照してください。

プッシュダウン述語を使用した事前フィルタ処理

多くの場合、プッシュダウン述語を使用してパーティションをフィルタリングできます。データセットのすべてのファイルをリストアップして読み取る必要はありません。データセット全体を読み取って DynamicFrame でフィルタリングする代わりに、Data Catalog 内でパーティションのメタデータに直接フィルターを適用できます。次に、実際に必要なものだけをリストアップして DynamicFrame 内に読み取ることができます。

たとえば、Python では以下のように記述できます。

```
glue_context.create_dynamic_frame.from_catalog(
    database = "my_S3_data_set",
    table_name = "catalog_data_table",
    push_down_predicate = my_partition_predicate)
```

これによって作成される DynamicFrame では、Data Catalog のパーティションのうち、述語式を満たすものだけがロードされます。ロードするデータのサブセットを絞り込む度合いに応じて、処理時間を大幅に短縮できる場合があります。

述語式として、Spark SQL でサポートされている任意のブール式を使用できます。Spark SQL クエリで WHERE 句に指定できる条件は、すべて正常に動作します。例えば述語式 `pushDownPredicate = "(year=='2017' and month=='04')"` では、Data Catalog 内で year が「2017」に等しく、また month が「04」に等しいパーティションのみロードされます。詳細については、Apache Spark SQL のドキュメントを参照してください。特に Scala SQL 関数リファレンスが参考になります。

カタログのパーティション述語を使用したサーバー側のフィルタリング

push_down_predicate オプションは、カタログからすべてのパーティションを一覧表示した後、Amazon S3 にあるそれらのパーティションからファイルをリストする前に適用されます。テーブルに多数のパーティションがある場合、カタログパーティションのリストに、時間的なオーバーヘッドが余計に発生する可能性があります。このオーバーヘッドに対処するには、catalogPartitionPredicate オプションを指定して AWS Glue Data Catalog のパーティションインデックスを使用しながら、サーバー側のパーティションでプルーニングを行います。1 つのテーブルに数百万のパーティションがある場合、これにより、パーティションのフィルタリングを大幅に高速化できます。カタログのパーティションインデックスではまだサポートされていない述語構文が、catalogPartitionPredicate で必要となる場合には、additional_options の中で push_down_predicate と catalogPartitionPredicate の両方を使用することもできます

Python:

```
dynamic_frame = glueContext.create_dynamic_frame.from_catalog(
    database=dbname,
    table_name=tablename,
    transformation_ctx="datasource0",
    push_down_predicate="day>=10 and customer_id like '10%'",
    additional_options={"catalogPartitionPredicate":"year='2021' and month='06'"}
```

```
)
```

Scala:

```
val dynamicFrame = glueContext.getCatalogSource(
    database = dbname,
    tableName = tablename,
    transformationContext = "datasource0",
    pushDownPredicate="day>=10 and customer_id like '10%'",
    additionalOptions = JsonOptions("""{
        "catalogPartitionPredicate": "year='2021' and month='06'"}""")
    ).getDynamicFrame()
```

> **Note**
>
> push_down_predicate と catalogPartitionPredicate では、使用される構文が異なります。前者では Spark SQL の標準構文を使用し、後者では JSQL パーサーを使用します。

パーティションの書き込み

デフォルトでは、DynamicFrame は書き込むときにパーティション分割されません。すべての出力ファイルは、指定した出力パスの最上位レベルに書き込まれます。最近まで、DynamicFrame をパーティションに書き込む唯一の方法は、書き込む前に Spark SQL DataFrame に変換することでした。

ただし、DynamicFrames ではキーのシーケンスを使用したネイティブのパーティション分割がサポートされるようになりました。この場合、シンクの作成時に partitionKeys オプションを使用します。例えば、次の Python コードではデータセットを Parquet 形式で Amazon S3 のディレクトリに書き込みます。これらのディレクトリでは、型フィールドごとにパーティション分割されています。これらのパーティションに対しては、他のシステム (Amazon Athena など) を使用しての処理が行えます。

```
glue_context.write_dynamic_frame.from_options(
    frame = projectedEvents,
    connection_type = "s3",
    connection_options = {"path": "$outpath", "partitionKeys": ["type"]},
    format = "parquet")
```

大きなグループの入力ファイルの読み取り

テーブルのプロパティを設定しておくと、Amazon S3 データストアからファイルが読み取られる際に、AWS Glue の ETL ジョブでそれらのファイルをグループ化させることができます。これらのプロパティを使用すると、各 ETL タスクでは入力ファイルのグループを単一のインメモリパーティションに読み取ることができます。これは、Amazon S3 データストアに多数の小さいファイルがある場合に便利です。特定のプロパティを定義する際には、Amazon S3 データパーティション内のファイルをグループ化し、そのグループの読み取り用のサイズを設定することを AWS Glue に指示します。また、`create_dynamic_frame.from_options` メソッドを使用した Amazon S3 データストアからの読み取りのために、これらのオプションを設定することもできます。

テーブルのファイルをグループ化するには、テーブル構造のパラメータフィールドにキーと値のペアを設定します。テーブルのパラメータフィールドに値を設定するには、JSON 表記を使用します。テーブルのプロパティを編集する詳しい方法については、「テーブルの詳細の表示と管理」を参照してください。

このメソッドを使用すると、Amazon S3 データストアにある Data Catalog でテーブルをグループ化できます。

groupFiles

Amazon S3 データパーティションにあるファイルのグループ化を有効にするには、groupFiles に `inPartition` を設定します。入力ファイル数が 50,000 を超える場合には、次の例のように AWS Glue がグループ化を自動的に有効にします。

```
'groupFiles': 'inPartition'
```

groupSize

groupSize をグループのターゲットサイズ (バイト単位) に設定します。groupSize プロパティはオプションです。指定しない場合、AWS Glue はクラスター内のすべての CPU コアを使用すると同時に、ETL タスクとメモリ内パーティションの総数を減らすようにサイズを計算します。

たとえば、次の例では、グループサイズを 1 MB に設定します。

```
'groupSize': '1048576'
```

groupsize は、計算の結果を使用して設定する必要があることに注意してください。例: 1024 * 1024 = 1048576。

recurse

recurse に Trueを設定し、paths によりパスの配列を指定すると、すべてのサブディレクトリのファイルを再帰的に読み取ることができます。次の例のように、paths が Amazon S3 のオブジェクトキーの配列である場合、または入力フォーマットが parquet/orc の場合、recurse を設定する必要はありません。

```
'recurse':True
```

create_dynamic_frame.from_options メソッドを使用して Amazon S3 から直接読み取る場合は、以下の接続オプションを追加します。たとえば、ファイルを 1 MB のグループにグループ化する方法は次のとおりです。

```
df = glueContext.create_dynamic_frame.from_options("s3", {'paths': ["s3://s3path/"],
 'recurse':True, 'groupFiles': 'inPartition', 'groupSize': '1048576'}, format="json")
```

> **Note**
>
> groupFiles は、csv、ion、grokLog、json、および xml のデータ形式から作成された DynamicFrames でサポートされています。このオプションは、avro、parquet、orc ではサポートされていません。

Amazon S3 用の VPC エンドポイント

セキュリティ上の理由から、多くの AWS ユーザーがアプリケーションを Amazon Virtual Private Cloud 環境 (Amazon VPC) 内で実行しています。Amazon VPC を使用すると、Amazon EC2 インスタンスを仮想プライベートクラウドで作成できます。そのため、パブリックインターネットなどの他のネットワークから論理的に分離されます。Amazon VPC を使用すると、IP アドレスの範囲、サブネット、ルーティングテーブル、ネットワークゲートウェイ、セキュリティ設定を適切に管理できます。

> **Note**
>
> 2013 年 12 月 4 日より後に AWS アカウントを作成した場合は、各 AWS リージョンにデフォルトで VPC が用意されています。追加設定なしにデフォルトの VPC をすぐに使用できます。
> デフォルト VPC の詳細については、Amazon VPC ユーザーガイドの「デフォルト VPC とデフォルトサブネット」を参照してください。

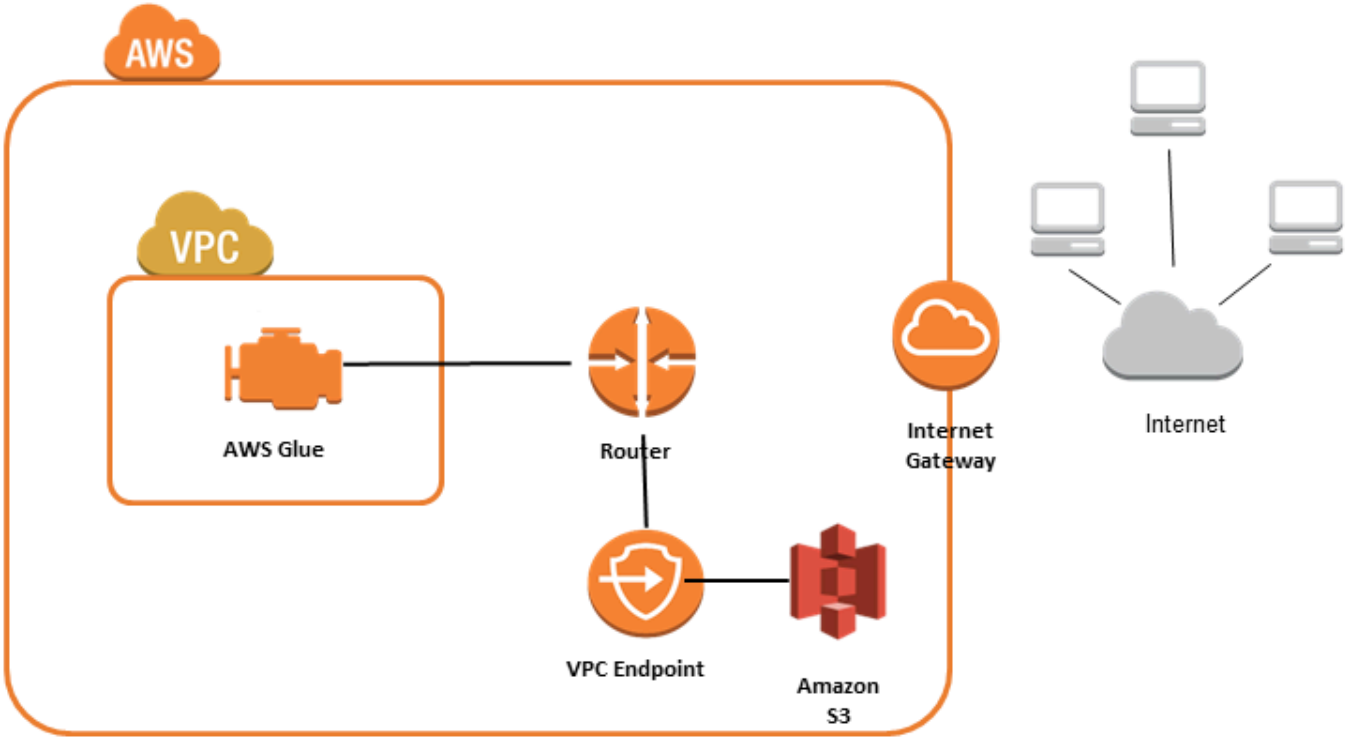
多くのお客様が、パブリックインターネット間のデータ送受信に関して、プライバシーとセキュリティに関する正当な懸念を抱いています。こういったお客様は、この懸念事項を解決するために、バーチャルプライベートネットワーク (VPN) を使用して、すべての Amazon S3 ネットワークトラフィックを、自社内の企業ネットワークのインフラストラクチャ経由でルーティングします。ただし、このアプローチでは、帯域幅や可用性の課題が生じる場合があります。

Amazon S3 用の VPC エンドポイントにより、これらの課題が軽減されます。Amazon S3 用 VPC エンドポイントを使用することで、AWS Glue はプライベート IP アドレスを使用して、パブリックインターネットに公開されることなく Amazon S3 にアクセスできるようになります。AWS Glue はパブリック IP アドレスを必要とせず、VPC のためのインターネットゲートウェイ、NAT デバイス、仮想プライベートゲートウェイは不要です。Amazon S3 へのアクセスを制御するには、エンドポイントのポリシーを使用します。VPC と AWS サービス間のトラフィックは、Amazon ネットワークを離れません。

Amazon S3 用に VPC エンドポイントを作成する際、リージョン内の Amazon S3 エンドポイント (例: s3.us-west-2.amazonaws.com) に対するリクエストはすべて、Amazon ネットワーク内のプライベートの Amazon S3 エンドポイントにルーティングされます。VPC の Amazon EC2 インスタンスで実行されているアプリケーションを変更する必要はありません。エンドポイント名は変わりませんが、Amazon S3 へのルーティングは完全に Amazon ネットワーク内で行われ、パブリックインターネットにアクセスすることはありません。

エンドポイントの詳細については、Amazon VPC ユーザーガイドの「VPC Endpoints」を参照してください。

AWS Glue が VPC エンドポイントを使用して Amazon S3 にアクセスする様子を、次の図に示します。



Amazon S3 のアクセスをセットアップするには

1. AWS Management Console にサインインして、Amazon VPC コンソール (https://console.aws.amazon.com/vpc/) を開きます。
2. 左のナビゲーションペインで [エンドポイント] を選択します。
3. [Create Endpoint] (エンドポイントの作成) をクリックし、ステップに従ってゲートウェイタイプの Amazon S3 VPC エンドポイントを作成します。

Amazon DocumentDB の接続

Spark 用の AWS Glue を使用して Amazon DocumentDB 内のテーブルに対する読み込みと書き込みを行うことができます。AWS Glue 接続を介して AWS Secrets Manager で保存されている認証情報を使用して Amazon DocumentDB に接続できます。

Amazon DocumentDB の詳細については、「Amazon DocumentDB のドキュメント」を参照してください。

> **Note**
>
> Amazon DocumentDB エラスティッククラスターは、現在 AWS Glue コネクタを使用する場合にはサポートされていません。エラスティッククラスターの詳細については、「Amazon DocumentDB エラスティッククラスターの使用」を参照してください。

Amazon DocumentDB コレクションへの読み取りと書き込み

> **Note**
>
> Amazon DocumentDB に接続する ETL ジョブを作成する際、`Connections` ジョブのプロパティにおいて、Amazon DocumentDB が実行されている Virtual Private Cloud (VPC) を特定するための、接続オブジェクトを指定する必要があります。接続オブジェクトの場合、接続タイプは `JDBC` で、`JDBC URL` は `mongo://<DocumentDB_host>:27017` である必要があります。

> **Note**
>
> これらのコードサンプルは AWS Glue 3.0 用に開発されました。AWS Glue 4.0 への移行については、the section called “MongoDB” を参照してください。`uri` パラメータが変更されました。

> **Note**
>
> Amazon DocumentDB を使用する場合、書かれた文書に `_id` が指定されている場合など、特定の状況では、`retryWrites` は false に設定する必要があります。詳細については、Amazon DocumentDB ドキュメントの「MongoDB との機能の違い」を参照してください。

次の Python スクリプトでは、Amazon DocumentDB に対する読み取りと書き込みのための、接続タイプと接続オプションの使用方法を示しています。

```
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext, SparkConf
from awsglue.context import GlueContext
from awsglue.job import Job
import time

## @params: [JOB_NAME]
args = getResolvedOptions(sys.argv, ['JOB_NAME'])

sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session

job = Job(glueContext)
job.init(args['JOB_NAME'], args)

output_path = "s3://some_bucket/output/" + str(time.time()) + "/"
documentdb_uri = "mongodb://<mongo-instanced-ip-address>:27017"
documentdb_write_uri = "mongodb://<mongo-instanced-ip-address>:27017"

read_docdb_options = {
    "uri": documentdb_uri,
    "database": "test",
    "collection": "coll",
    "username": "username",
    "password": "1234567890",
    "ssl": "true",
    "ssl.domain_match": "false",
    "partitioner": "MongoSamplePartitioner",
    "partitionerOptions.partitionSizeMB": "10",
    "partitionerOptions.partitionKey": "_id"
}

write_documentdb_options = {
    "retryWrites": "false",
    "uri": documentdb_write_uri,
    "database": "test",
    "collection": "coll",
    "username": "username",
    "password": "pwd"
}
```

```
# Get DynamicFrame from  DocumentDB
dynamic_frame2 =
 glueContext.create_dynamic_frame.from_options(connection_type="documentdb",

 connection_options=read_docdb_options)

# Write DynamicFrame to MongoDB and DocumentDB
glueContext.write_dynamic_frame.from_options(dynamic_frame2,
 connection_type="documentdb",

 connection_options=write_documentdb_options)

job.commit()
```

次の Scala スクリプトでは、Amazon DocumentDB に対する読み取りと書き込みのための、接続タイプと接続オプションの使用方法を示しています。

```
import com.amazonaws.services.glue.GlueContext
import com.amazonaws.services.glue.MappingSpec
import com.amazonaws.services.glue.errors.CallSite
import com.amazonaws.services.glue.util.GlueArgParser
import com.amazonaws.services.glue.util.Job
import com.amazonaws.services.glue.util.JsonOptions
import com.amazonaws.services.glue.DynamicFrame
import org.apache.spark.SparkContext
import scala.collection.JavaConverters._

object GlueApp {
  val DOC_URI: String = "mongodb://<mongo-instanced-ip-address>:27017"
  val DOC_WRITE_URI: String = "mongodb://<mongo-instanced-ip-address>:27017"
  lazy val documentDBJsonOption = jsonOptions(DOC_URI)
  lazy val writeDocumentDBJsonOption = jsonOptions(DOC_WRITE_URI)
  def main(sysArgs: Array[String]): Unit = {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)
    val args = GlueArgParser.getResolvedOptions(sysArgs, Seq("JOB_NAME").toArray)
    Job.init(args("JOB_NAME"), glueContext, args.asJava)

    // Get DynamicFrame from DocumentDB
    val resultFrame2: DynamicFrame = glueContext.getSource("documentdb",
 documentDBJsonOption).getDynamicFrame()
```

```
    // Write DynamicFrame to DocumentDB
    glueContext.getSink("documentdb", writeJsonOption).writeDynamicFrame(resultFrame2)

    Job.commit()
  }

  private def jsonOptions(uri: String): JsonOptions = {
    new JsonOptions(
      s"""{"uri": "${uri}",
          |"database":"test",
          |"collection":"coll",
          |"username": "username",
          |"password": "pwd",
          |"ssl":"true",
          |"ssl.domain_match":"false",
          |"partitioner": "MongoSamplePartitioner",
          |"partitionerOptions.partitionSizeMB": "10",
          |"partitionerOptions.partitionKey": "_id"}""".stripMargin)
  }
}
```

Amazon DocumentDB 接続のオプションのリファレンス

Amazon DocumentDB (MongoDB 互換) への接続を指定します。

接続オプションは、ソース接続とシンク接続とで異なります。

"connectionType": "Documentdb" ソースとする

`"connectionType": "documentdb"` をソースとして、次の接続オプションを使用します。

- `"uri"`: (必須) 読み取り元の Amazon DocumentDB ホスト (`mongodb://<host>:<port>` 形式)。
- `"database"`: (必須) 読み取り元の Amazon DocumentDB データベース。
- `"collection"`: (必須) 読み取り元の Amazon DocumentDB コレクション。
- `"username"`: (必須) Amazon DocumentDB のユーザー名。
- `"password"`: (必須) Amazon DocumentDB のパスワード。
- `"ssl"`: (SSL を使用する場合は必須) SSL を使用して接続する場合は、このオプションの値を `"true"` に設定する必要があります。
- `"ssl.domain_match"`: (SSL を使用する場合は必須) SSL を使用して接続する場合は、このオプションの値を `"false"` に設定する必要があります。

- "batchSize": (オプション): 内部バッチのカーソル内で使用される、バッチごとに返されるドキュメントの数。
- "partitioner": (オプション) Amazon DocumentDB から入力データを読み取るためのパーティショナーのクラス名。コネクタには、次のパーティショナーがあります。
  - MongoDefaultPartitioner (デフォルト) (AWS Glue 4.0 ではサポートされていません)
  - MongoSamplePartitioner (AWS Glue 4.0 ではサポートされていません)
  - MongoShardedPartitioner
  - MongoSplitVectorPartitioner
  - MongoPaginateByCountPartitioner
  - MongoPaginateBySizePartitioner (AWS Glue 4.0 ではサポートされていません)
- "partitionerOptions" (オプション): 指定されたパーティショナーのオプション。各パーティショナーでは、次のオプションがサポートされています。
  - MongoSamplePartitioner: partitionKey, partitionSizeMB, samplesPerPartition
  - MongoShardedPartitioner: shardkey
  - MongoSplitVectorPartitioner: partitionKey、partitionSizeMB
  - MongoPaginateByCountPartitioner: partitionKey, numberOfPartitions
  - MongoPaginateBySizePartitioner: partitionKey、partitionSizeMB

  これらのオプションの詳細については、MongoDB のドキュメントの「Partitioner Configuration」を参照してください。

"connectionType": "documentdb" as Sink

"connectionType": "documentdb" をシンクとして、次の接続オプションを使用します。

- "uri": (必須) 書き込み先の Amazon DocumentDB ホスト (mongodb://<host>:<port> 形式)。
- "database": (必須) 書き込み先の Amazon DocumentDB データベース。
- "collection": (必須) 書き込み先の Amazon DocumentDB コレクション。
- "username": (必須) Amazon DocumentDB のユーザー名。
- "password": (必須) Amazon DocumentDB のパスワード。
- "extendedBsonTypes": (オプション) true が指定されている場合、Amazon DocumentDB へのデータ書き込み時に拡張 BSON 型を使用できます。デフォルト: true。

- "replaceDocument": (オプション) true の場合、_id フィールドを含むデータセットを保存するときに、ドキュメント全体を置き換えます。false の場合、データセットのフィールドと一致するドキュメントのフィールドのみが更新されます。デフォルト: true。
- "maxBatchSize": (オプション): データを保存するときの一括オペレーションの最大バッチサイズ。デフォルトは 512 です。
- "retryWrites": (オプション): AWS Glue でネットワークエラーが発生した場合、特定の書き込みオペレーションを 1 回自動的に再試行します。

OpenSearch Service 接続

AWS Glue for Spark を使用して、AWS Glue 4.0 以降のバージョンの OpenSearch Service のテーブルからの読み取りとテーブルへの書き込みを行うことができます。OpenSearch クエリを使用して、OpenSearch Service から何を読み取るかを定義できます。AWS Glue 接続を通じて AWS Secrets Manager で保存されている HTTP 基本認証情報を使用して OpenSearch Service に接続します。この機能は OpenSearch サービスのサーバーレスとは互換性がありません。

Amazon OpenSearch Service の詳細については、Amazon OpenSearch Service ドキュメントを参照してください。

OpenSearch Service 接続の設定

AWS Glue から OpenSearch Service に接続するには、OpenSearch Service 認証情報を作成して AWS Secrets Manager シークレットに保存し、そのシークレットを OpenSearch Service AWS Glue 接続に関連付ける必要があります。

前提条件:

- 読み取り元とするドメインエンドポイント *aosEndpoint* とポート *aosPort* を特定するか、または Amazon OpenSearch Service ドキュメントの手順に従ってリソースを作成します。ドメインの作成の詳細については、Amazon OpenSearch Service ドキュメントの「Amazon OpenSearch Service ドメインの作成と管理」を参照してください。

  Amazon OpenSearch Service ドメインエンドポイントのデフォルト形式は、https://search-*domainName*-*unstructuredIdContent*.*region*.es.amazonaws.com です。ドメインエンドポイントの識別の詳細については、Amazon OpenSearch Service ドキュメントの「Amazon OpenSearch Service ドメインの作成と管理」を参照してください。

  ドメインの HTTP 基本認証情報、*aosUser*、および *aosPassword* を特定または生成します。

OpenSearch Service に対する接続を設定するには:

1. AWS Secrets Manager で、OpenSearch Service 認証情報を使用してシークレットを作成します。Secrets Manager でシークレットを作成するには、AWS Secrets Manager ドキュメントの「AWS Secrets Manager シークレットを作成する」にあるチュートリアルに従ってください。シークレットを作成したら、次のステップのためにシークレット名 *secretName* を保存しておきます。

   - [key/value ペア] を選択する際に、*aosUser* という値を持つキー USERNAME のペアを作成します。
   - [key/value ペア] を選択する際に、*aosPassword* という値を持つキー PASSWORD のペアを作成します。

2. AWS Glue コンソールで、「the section called “AWS Glue 接続の追加”」にあるステップに従って接続を作成します。接続を作成したら、将来的に AWS Glue で使用するために、接続名 *connectionName* を維持します。

   - [接続タイプ] を選択する場合は、[OpenSearch Service] を選択します。
   - ドメインエンドポイントを選択する場合は、*aosEndpoint* を入力します。
   - ポートを選択する場合は、*aosPort* を入力します。
   - [AWS Secret] をクリックして、*secretName* を入力します。

AWS Glue OpenSearch Service 接続を作成した後、AWS Glue ジョブを実行する前に次のステップを実行する必要があります。

- AWS Glue ジョブに関連付けられている IAM ロールに *secretName* を読み取るアクセス許可を付与します。
- AWS Glue ジョブ設定で、追加のネットワーク接続として *connectionName* を指定します。

OpenSearch Service のインデックスからの読み取り

前提条件:

- 読み取り元とする OpenSearch Service インデックスである *aosIndex*。
- 認証およびネットワーク位置情報を提供するように設定された AWS Glue OpenSearch Service 接続。これを取得するには、前の手順の「OpenSearch Service に対する接続を設定するには」のステップを実行します。AWS Glue 接続、*connectionName* の名前が必要になります。

この例では、Amazon OpenSearch Service からインデックスを読み取ります。pushdown パラメータを入力する必要があります。

例：

```
opensearch_read = glueContext.create_dynamic_frame.from_options(
    connection_type="opensearch",
    connection_options={
        "connectionName": "connectionName",
        "opensearch.resource": "aosIndex",
        "pushdown": "true",
    }
)
```

クエリ文字列を指定して、DynamicFrame で返される結果をフィルタリングすることもできます。opensearch.query を設定する必要があります。

opensearch.query は、URL クエリパラメータ文字列である *queryString* またはクエリ DSL JSON オブジェクトである *queryObject* を受け取ることができます。クエリ DSL の詳細については、OpenSearch ドキュメントの「クエリ DSL」を参照してください。URL クエリパラメータ文字列を入力するには、完全修飾 URL の場合と同様に、クエリの前に ?q= を付加します。クエリ DSL オブジェクトを入力するには、入力する前に JSON オブジェクトを文字列エスケープします。

例：

```
            queryObject = "{ "query": { "multi_match": { "query": "Sample", "fields":
 [ "sample" ] } } }"
            queryString = "?q=queryString"

            opensearch_read_query = glueContext.create_dynamic_frame.from_options(
    connection_type="opensearch",
    connection_options={
        "connectionName": "connectionName",
        "opensearch.resource": "aosIndex",
        "opensearch.query": queryString,
        "pushdown": "true",
    }
)
```

特定の構文以外でクエリを作成する方法の詳細については、OpenSearch ドキュメントの「クエリ文字列構文」を参照してください。

配列型データを含む OpenSearch コレクションから読み取る場合は、opensearch.read.field.as.array.include パラメーターを使用してメソッド呼び出しでどのフィールドが配列型であるかを指定する必要があります。

例えば、次のドキュメントを読み取ると、genre と actor 配列フィールドが表示されます。

```
{
    "_index": "movies",
    "_id": "2",
    "_version": 1,
    "_seq_no": 0,
    "_primary_term": 1,
    "found": true,
    "_source": {
        "director": "Frankenheimer, John",
        "genre": [
            "Drama",
            "Mystery",
            "Thriller",
            "Crime"
        ],
        "year": 1962,
        "actor": [
            "Lansbury, Angela",
            "Sinatra, Frank",
            "Leigh, Janet",
            "Harvey, Laurence",
            "Silva, Henry",
            "Frees, Paul",
            "Gregory, James",
            "Bissell, Whit",
            "McGiver, John",
            "Parrish, Leslie",
            "Edwards, James",
            "Flowers, Bess",
            "Dhiegh, Khigh",
            "Payne, Julie",
            "Kleeb, Helen",
            "Gray, Joe",
            "Nalder, Reggie",
            "Stevens, Bert",
            "Masters, Michael",
            "Lowell, Tom"
```

```
        ],
        "title": "The Manchurian Candidate"
    }
}
```

この場合、メソッド呼び出しにそれらのフィールド名を含めます。例：

```
"opensearch.read.field.as.array.include": "genre,actor"
```

配列フィールドがドキュメント構造内にネストされている場合は、ドット表記を使用して参照してください: "genre,actor,foo.bar.baz"。これにより、埋め込みドキュメント bar を含む埋め込みドキュメント foo を介してソースドキュメントに含まれる配列 baz が指定されます。

OpenSearch Service テーブルへの書き込み

この例では、既存の DynamicFrame である *dynamicFrame* から OpenSearch Service に情報を書き込みます。インデックスに既に情報がある場合、AWS Glue は DynamicFrame からのデータを付加します。pushdown パラメータを入力する必要があります。

前提条件:

- 書き込み先とする OpenSearch Service テーブル。テーブルの識別情報が必要になります。これに *tableName* と名前を付けます。
- 認証およびネットワークの場所の情報を提供するように設定された AWS Glue OpenSearch Service 接続。これを取得するには、前の手順の「OpenSearch Service に対する接続を設定するには」のステップを実行します。AWS Glue 接続、*connectionName* の名前が必要になります。

例：

```
glueContext.write_dynamic_frame.from_options(
    frame=dynamicFrame,
    connection_type="opensearch",
    connection_options={
      "connectionName": "connectionName",
      "opensearch.resource": "aosIndex",
    },
)
```

OpenSearch Service 接続のオプションのリファレンス

- `connectionName` — 必須。読み込み/書き込みに使用されます。認証およびネットワークの場所の情報を接続方法に提供するように設定された AWS Glue OpenSearch Service 接続の名前。
- `opensearch.resource` — 必須。読み込み/書き込みに使用されます。有効な値: OpenSearch インデックス名。接続メソッドがインタラクションするインデックスの名前。
- `opensearch.query` — 読み取りに使用。有効な値: JSON をエスケープした文字列、またはこの文字列が ? で始まる場合は URL の検索部分。読み取り時に取得する内容をフィルタリングする OpenSearch クエリ。このパラメータの使用方法の詳細については、前のセクション「the section called “OpenSearch Service から読み取る”」を参照してください。
- `pushdown` — 次の場合は必須です。読み込みに使用されます。有効な値: ブール値。Spark に読み取りクエリを OpenSearch に渡し、データベースが関連ドキュメントのみを返すように指示します。
- `opensearch.read.field.as.array.include` — 配列タイプのデータを読み取る場合は必須。読み込みに使用されます。有効値: カンマで区切られた、フィールド名のリスト。OpenSearch ドキュメントから配列として読み取るフィールドを指定します。このパラメータの使用方法の詳細については、前のセクション「the section called “OpenSearch Service から読み取る”」を参照してください。

Redshift 接続

AWS Glue for Spark を使用して Amazon Redshift データベース内のテーブルに対する読み込みと書き込みを行うことができます。AWS Glue は Amazon Redshift SQL の `COPY` および `UNLOAD` コマンドを使用して、Amazon S3 を介してデータを移動し、スループットを最大化します。AWS Glue 4.0 以降では、Apache Spark 用の Amazon Redshift インテグレーションを使用して、Amazon Redshift 固有の最適化と機能を使用して読み込みと書き込みを行うことができます。また、以前のバージョンで接続する場合には利用できなかった機能もあります。

AWS Glue によって Amazon Redshift ユーザーがサーバーレスデータ統合と ETL のための AWS Glue への移行がこれまでになく簡単になった方法をご覧ください。

Redshift 接続の設定

AWS Glue で Amazon Redshift クラスターを使用するには、いくつかの前提条件が必要です。

- データベースの読み込みおよび書き込み時に一時的なストレージとして使用する Amazon S3 ディレクトリがあること。

- Amazon Redshift クラスター、AWS Glue ジョブ、Amazon S3 ディレクトリ間の通信を可能にする Amazon VPC があること。
- AWS Glue ジョブと Amazon Redshift クラスターに対する適切な IAM アクセス許可があること。

IAM ロールを設定する

Amazon Redshift クラスターのロールを設定する

AWS Glue ジョブと統合するため、Amazon Redshift クラスターによる Amazon S3 への読み取りおよび書き込みが可能である必要があります。これを可能にするには、接続する Amazon Redshift クラスターに IAM ロールを関連付けます。ロールには、Amazon S3 の一時ディレクトリからの読み取り、および一時ディレクトリへの書き込みを許可するポリシーが必要です。ロールには、`redshift.amazonaws.com` サービスで `AssumeRole` を許可する信頼関係が必要です。

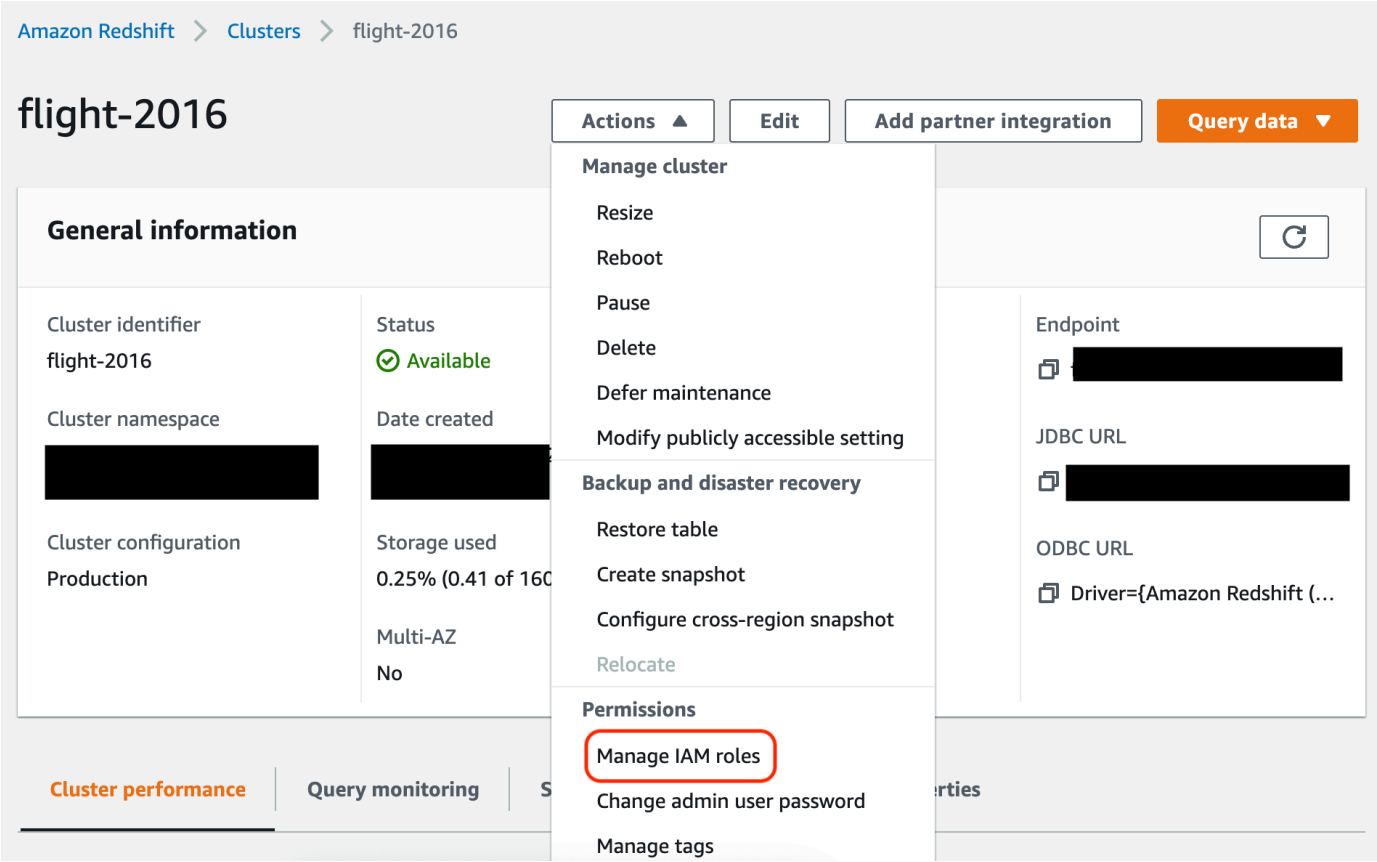
IAM ロールを Amazon Redshift に関連付けるには

1. 前提条件: ファイルの一時的なストレージとして使用する Amazon S3 バケットまたはディレクトリがあること。
2. Amazon Redshift クラスターに必要な Amazon S3 のアクセス許可を特定します。Amazon Redshift クラスターとの間でデータを移動する場合、AWS Glue ジョブは Amazon Redshift に対して COPY および UNLOAD ステートメントを発行します。ジョブによって Amazon Redshift のテーブルが変更された場合、AWS Glue は CREATE LIBRARY ステートメントも発行します。Amazon Redshift がこれらのステートメントを実行するために必要な特定の Amazon S3 のアクセス許可については、Amazon Redshift のドキュメント「Amazon Redshift: 他の AWS リソースにアクセスするアクセス許可」を参照してください。
3. IAM コンソールで、必要なアクセス許可を持つ IAM ポリシーを作成します。ポリシーの作成についての詳細は、「IAM ポリシーの作成」を参照してください。
4. IAM コンソールで、ロールと信頼関係を作成し、Amazon Redshift がロールを継承できるようにします。これについては、IAM のドキュメントにある「AWS のサービス用ロールの作成 (コンソール)」の手順に従ってください。
   - AWS サービスのユースケースを選択するように求められた場合、[Redshift - カスタマイズ可能] を選択します。
   - ポリシーをアタッチするように求められた場合、以前に定義したポリシーを選択します。

> **Note**
>
> Amazon Redshiftのロールの詳細については、Amazon Redshift のドキュメントの「ユーザーに代わって Amazon Redshift が他の AWS サービスにアクセスすることを許可する」を参照してください。

5. Amazon Redshift コンソールで、ロールを Amazon Redshift クラスターに関連付けます。これについては、Amazon Redshift のドキュメントに記載されている手順に従ってください。

   Amazon Redshift コンソールでハイライト表示されているオプションを選択して、この設定を行えます。



> **Note**
>
> AWS Glue はデフォルトで、ジョブを実行するために指定したロールを使用して作成された、Amazon Redshift の一時的な認証情報を渡します。これらの認証情報を使用することは

お勧めしません。これらの認証情報は、セキュリティ上の理由により 1 時間後に失効します。

AWS Glue ジョブのロールを設定する

AWS Glue ジョブには Amazon S3 バケットにアクセスするためのロールが必要です。Amazon Redshift クラスターの IAM アクセス許可は必要ありません。アクセスは Amazon VPC の接続とデータベースの認証情報によって制御されます。

Amazon VPC を設定する

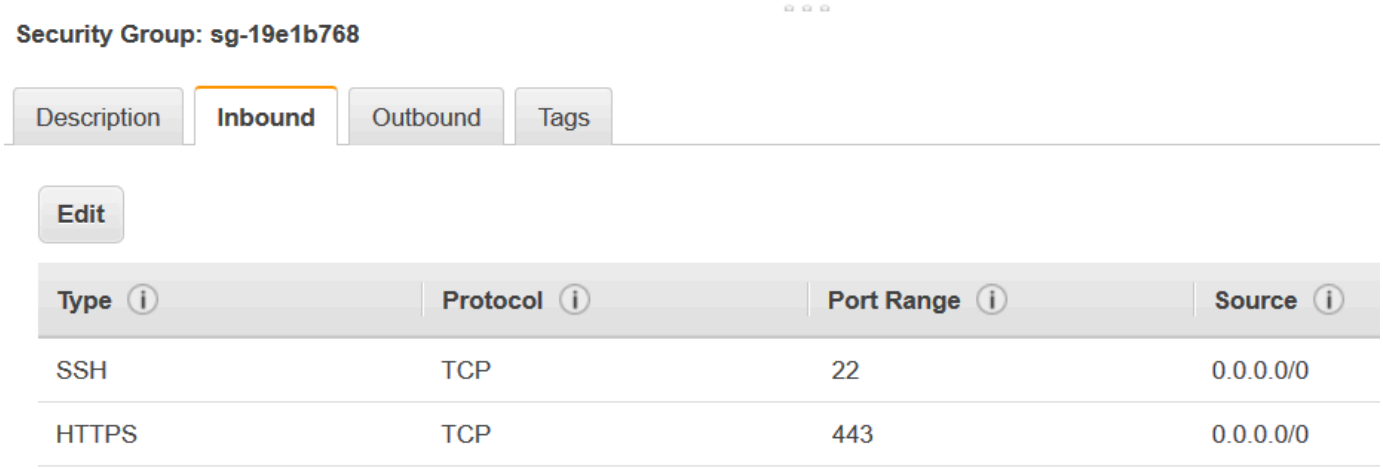
Amazon Redshift データストアへのアクセスをセットアップするには

1. AWS Management Consoleにサインインして、https://console.aws.amazon.com/redshiftv2/ で Amazon Redshift コンソールを開きます。
2. 左のナビゲーションペインで [Clusters] (クラスター) を選択してください。
3. AWS Glue からアクセスするクラスターの名前を選択します。
4. [Cluster Properties (クラスターのプロパティ)] セクションで、[VPC security groups (VPC セキュリティグループ)] 内のセキュリティグループを選択し、AWS Glue が使用できるようにします。今後の参照用に選択したセキュリティグループの名前を記録します。Amazon EC2 コンソールでセキュリティグループを選択すると、[Security Groups] (セキュリティグループ) の一覧が開きます。
5. 変更するセキュリティグループを選択し、[Inbound (インバウンド)] タブに移動します。
6. 自己参照ルールを追加して、AWS Glue コンポーネントが通信できるようにします。具体的には、[Type (タイプ)] `All TCP`、[Protocol (プロトコル)] は `TCP`、[Port Range (ポート範囲)] にはすべてのポートが含まれ、[Source (ソース)] は [Group ID (グループ ID)] と同じセキュリティグループ名であるというルールを追加または確認します。

   インバウンドルールは以下のようになります。

タイプ	プロトコル	ポート範囲	ソース
すべての TCP	TCP	0 ~ 65535	database-security-group

   例:



7. アウトバウンドトラフィックのルールも追加します。すべてのポートへのアウトバウンドトラフィックを開きます。以下に例を示します。

タイプ	プロトコル	ポート範囲	デスティネーション
すべてのトラフィック	すべて	すべて	0.0.0.0/0

または、[Type (タイプ)] は `All TCP`、[Protocol (プロトコル)] は `TCP`、[Port Range (ポート範囲)] にすべてのポートが含まれ、[Destination (宛先)] は [Group ID (グループ ID)] と同じセキュリティグループ名の自己参照ルールを作成します。Amazon S3 VPC エンドポイントを使用している場合は、Amazon S3 にアクセスするための HTTPS ルールも追加します。セキュリティグループルールには、VPC から Amazon S3 VPC エンドポイントへのトラフィックを許可するために、*`s3-prefix-list-id`* が必要です。

例:

タイプ	プロトコル	ポート範囲	デスティネーション
すべての TCP	TCP	0 ~ 65535	*`security-group`*
HTTPS	TCP	443	*`s3-prefix-list-id`*

AWS Glue をセットアップ

Amazon VPC 接続情報を提供する AWS Glue データカタログ接続を作成する必要があります。

コンソールで AWS Glue への Amazon Redshift Amazon VPC 接続を設定するには

1. 「the section called “AWS Glue 接続の追加”」のステップに従ってデータカタログ接続を作成します。接続を作成したら、次のステップのために接続名 *connectionName* を保存しておきます。

   - 接続タイプを選択するときは、Amazon Redshift を選択します。
   - Redshift クラスターを選択するときは、クラスターを名前で選択します。
   - クラスターの Amazon Redshift ユーザーのデフォルト接続情報を指定します。
   - Amazon VPC 設定は自動的に設定されます。

   > **Note**
   >
   > AWS SDK を使用して Amazon Redshift 接続を作成する場合は、Amazon VPC の `PhysicalConnectionRequirements` を手動で指定する必要があります。

2. AWS Glue ジョブ設定で、追加のネットワーク接続として *connectionName* を指定します。

例: Amazon Redshift テーブルからの読み取り

Amazon Redshift クラスターと Amazon Redshift Serverless 環境から読み込みを行うことができます。

前提条件: 読み込む目的の Amazon Redshift テーブルがあること。前のセクション the section called “Redshift の設定” の手順を実行してください。実行後に、一時ディレクトリ *temp-s3-dir* および IAM ロール *rs-role-name* 用の Amazon S3 URI が用意されるはずです (*role-account-id* アカウント内)。

Using the Data Catalog

その他の前提条件: 読み取る目的の Amazon Redshift テーブル用のテーブルとデータカタログのデータベースがあること。データカタログの詳細については、「データ検出とカタログ化」を参照してください Amazon Redshift テーブルのエントリを作成したら、*redshift-dc-database-name* および *redshift-table-name* を使用して接続を識別します。

設定: 関数オプションで、database および table_name パラメータを使用してデータカタログテーブルを識別します。redshift_tmp_dir を使用して、Amazon S3 の一時ディレクトリを識別します。また、additional_options パラメータの aws_iam_role キーを使用して、*rs-role-name* を指定します。

```
 glueContext.create_dynamic_frame.from_catalog(
    database = "redshift-dc-database-name",
    table_name = "redshift-table-name",
    redshift_tmp_dir = args["temp-s3-dir"],
    additional_options = {"aws_iam_role": "arn:aws:iam::role-account-id:role/rs-role-name"})
```

Connecting directly

その他の前提条件: Amazon Redshift テーブルの名前 (*redshift-table-name*) が必要です。そのテーブルを格納する Amazon Redshift クラスターの JDBC 接続情報が必要です。*host*、*port*、*redshift-database-name*、*username*、*password* で接続情報を入力します。

Amazon Redshift クラスターを操作するときは、Amazon Redshift コンソールから接続情報を取得できます。Amazon Redshift サーバーレスを使用する場合は、Amazon Redshift ドキュメントの「Amazon Redshift Serverless への接続」を参照してください。

設定: 関数オプションでは、url、dbtable、user および password を使用して接続パラメータを識別します。redshift_tmp_dir を使用して、Amazon S3 の一時ディレクトリを識別します。from_options を使用する場合は IAM ロールを指定できます。この構文はデータカタログを介して接続するのと似ていますが、connection_options マップにパラメータを配置します。

パスワードを AWS Glue のスクリプトにハードコードする方法は、好ましくありません。SDK for Python (Boto3) を使用して AWS Secrets Manager にパスワードを保存し、スクリプトで取得することを検討してください。

```
my_conn_options = {
    "url": "jdbc:redshift://host:port/redshift-database-name",
    "dbtable": "redshift-table-name",
    "user": "username",
```

```
    "password": "password",
    "redshiftTmpDir": args["temp-s3-dir"],
    "aws_iam_role": "arn:aws:iam::account id:role/rs-role-name"
}

df = glueContext.create_dynamic_frame.from_options("redshift", my_conn_options)
```

例: Amazon Redshift テーブルへの書き込み

Amazon Redshift クラスターと Amazon Redshift Serverless 環境に書き込みを行うことができます。

前提条件: Amazon Redshift クラスターがあり、前のセクション「the section called “Redshift の設定”」の手順を実行すること。実行後に、一時ディレクトリ *temp-s3-dir* および IAM ロール *rs-role-name* 用の Amazon S3 URI が用意されるはずです (*role-account-id* アカウント内)。また、データベースに書き込むコンテンツがある DynamicFrame も必要です。

Using the Data Catalog

その他の前提条件 Amazon Redshift クラスター用のデータカタログのデータベースと、読み取る目的のテーブルがあること。データカタログの詳細については、「データ検出とカタログ化」を参照してください *redshift-dc-database-name* を使用して接続を識別し、*redshift-table-name* を使用してターゲットテーブルを識別します。

設定: 関数オプションで、database パラメータを使用してデータカタログのデータベースを識別し、table_name を使用してテーブルを指定します。redshift_tmp_dir を使用して、Amazon S3 の一時ディレクトリを識別します。また、additional_options パラメータの aws_iam_role キーを使用して、*rs-role-name* を指定します。

```
 glueContext.write_dynamic_frame.from_catalog(
    frame = input dynamic frame,
    database = "redshift-dc-database-name",
    table_name = "redshift-table-name",
    redshift_tmp_dir = args["temp-s3-dir"],
    additional_options = {"aws_iam_role": "arn:aws:iam::account-id:role/rs-role-name"})
```

Connecting through a AWS Glue connection

write_dynamic_frame.from_options メソッドを使用して、Amazon Redshift に直接接続できます。ただし、接続の詳細をスクリプトに直接挿入するのではなく、from_jdbc_conf メソッドを使用してデータカタログの接続に保存されている接続の詳細を参照します。これは、データベースのデータカタログテーブルをクロールまたは作成することなく実行できます。データカタログの接続ついての詳細は、「データへの接続」を参照してください。

その他の前提条件: データベース用のデータカタログの接続と、読み取る目的の Amazon Redshift テーブルがあること。

設定: *dc-connection-name* を使用して、データカタログの接続を識別します。*redshift-table-name* および *redshift-database-name* を使用して、Amazon Redshift のデータベースおよびテーブルを識別します。catalog_connection を使用してデータカタログの接続情報を指定し、dbtable および database を使用して Amazon Redshift の情報を指定します。この構文はデータカタログを介して接続するのと似ていますが、connection_options マップにパラメータを配置します。

```
my_conn_options = {
    "dbtable": "redshift-table-name",
    "database": "redshift-database-name",
    "aws_iam_role": "arn:aws:iam::role-account-id:role/rs-role-name"
}

glueContext.write_dynamic_frame.from_jdbc_conf(
    frame = input dynamic frame,
    catalog_connection = "dc-connection-name",
    connection_options = my_conn_options,
    redshift_tmp_dir = args["temp-s3-dir"])
```

Amazon Redshift 接続のオプションに関するリファレンス

url、user および password などの情報を設定するため、AWS Glue の JDBC 接続に使用される基本的な接続オプションはすべて、全 JDBC タイプで共通しています。JDBC のパラメータの詳細については、「the section called “JDBC 接続パラメータ”」を参照してください。

Amazon Redshift 接続のタイプには、その他の接続オプションがいくつかあります。

- `"redshiftTmpDir"`: (必須) データベースからコピーする際に一時的なデータをステージングできる Amazon S3 パス。
- `"aws_iam_role"`: (オプション) IAM ロールの ARN。AWS Glue ジョブはこのロールを Amazon Redshift クラスターに渡し、ジョブの手順を完了するために必要なアクセス許可をクラスターに付与します。

AWS Glue 4.0 以降で利用可能なその他の接続オプション

新しい Amazon Redshift コネクタのオプションを、AWS Glue の接続オプションを介して渡すこともできます。サポートされているコネクタオプションの詳細なリストについては、「Apache Spark 用の Amazon Redshift の統合」の「Spark SQL パラメータ」セクションを参照してください。

参考までに、いくつかの新しいオプションをここで再度確認します。

名前	必須	デフォルト	説明
autopushdown	いいえ	正	SQL オペレーションの Spark 論理プランをキャプチャして分析することにより、述語とクエリのプッシュダウンを適用します。オペレーションは SQL クエリに変換され、Amazon Redshift で実行されることでパフォーマンスが向上します。
autopushdown.s3_result_cache	いいえ	誤	SQL クエリをキャッシュして Amazon S3 パスマッピングのデータをメモリにアンロードします。これにより、同じクエリを同じ Spark セッションで再度実行す

名前	必須	デフォルト	説明
			る必要がなくなります。autopushdown が有効な場合のみサポートされます。
unload_s3_format	いいえ	[PARQUET]	PARQUET - クエリ結果をパーケット形式でアンロードします。 TEXT - クエリ結果をパイプ区切りのテキスト形式でアンロードします。
sse_kms_key	いいえ	該当なし	AWS のデフォルトの暗号化の代わりに UNLOAD オペレーション中の暗号化に使用する AWS SSE-KMS キー。

名前	必須	デフォルト	説明
extracopyoptions	いいえ	該当なし	データロード時に Amazon Redshift COPY コマンドに付加する追加オプション (TRUNCATECOLUMNS 、MAXERROR n など) のリスト (その他のオプションについては、「COPY: 任意指定のパラメータ」を参照してください)。 これらのオプションはCOPYコマンドの最後に付加されるため、使用できるのはコマンドの最後で意味のあるオプションのみであることに注意してください。これで、考えられるほとんどのユースケースが力バーされるはずです。
csvnullstring (実験的)	いいえ	NULL	CSV tempformat を使用するときに NULL として書き込む文字列値。これは実際のデータには出現しない値でなければなりません。

これらの新しいパラメータは、次の方法で使用できます。

パフォーマンス向上のための新しいオプション

新しいコネクタには、パフォーマンス向上のための新しいオプションがいくつか導入されています。

- autopushdown: デフォルトでは有効になっています。
- autopushdown.s3_result_cache: デフォルトでは無効になっています。
- unload_s3_format: デフォルトでは PARQUET になっています。

これらのオプションの使用方法については、「Apache Spark 用の Amazon Redshift の統合」を参照してください。キャッシュされた結果には古い情報が含まれている可能性があるため、読み取りと書き込みのオペレーションが混在している場合は autopushdown.s3_result_cache をオンにしないことをお勧めします。パフォーマンスを向上させ、ストレージコストを削減するために、UNLOAD コマンドの unload_s3_format オプションはデフォルトで PARQUET に設定されています。UNLOAD コマンドのデフォルト動作を使用するには、オプションを TEXT にリセットします。

読み取り用の新しい暗号化オプション

デフォルトでは、Amazon Redshift テーブルからデータを読み取る際に AWS Glue が使用する一時フォルダ内のデータは、SSE-S3 暗号化を使用して暗号化されます。AWS Key Management Service (AWS KMS) のカスタマーマネージドキーを使用してデータを暗号化するには、AWS Glue バージョン 3.0 の従来の設定オプション ("extraunloadoptions" # s"ENCRYPTED KMS_KEY_ID '$kmsKey'") ではなく、ksmKey が AWS KMS からのキー ID になるように ("sse_kms_key" # kmsKey) を設定します。

```
datasource0 = glueContext.create_dynamic_frame.from_catalog(
  database = "database-name",
  table_name = "table-name",
  redshift_tmp_dir = args["TempDir"],
  additional_options = {"sse_kms_key":"<KMS_KEY_ID>"},
  transformation_ctx = "datasource0"
)
```

IAM ベースの JDBC URL をサポート

新しいコネクタは IAM ベースの JDBC URL をサポートしているため、ユーザーやパスワード、またはシークレットを渡す必要はありません。IAM ベースの JDBC URL では、コネクタはジョブのランタイムロールを使用して Amazon Redshift データソースにアクセスします。

ステップ 1: 次の最低限必要なポリシーを AWS Glue ジョブランタイムロールにアタッチします。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "redshift:GetClusterCredentials",
            "Resource": [
                "arn:aws:redshift:<region>:<account>:dbgroup:<cluster name>/*",
                "arn:aws:redshift:*:<account>:dbuser:*/*",
                "arn:aws:redshift:<region>:<account>:dbname:<cluster name>/<database
 name>"
            ]
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": "redshift:DescribeClusters",
            "Resource": "*"
        }
    ]
}
```

ステップ 2: IAM ベースの JDBC URL を次のように使用します。接続している Amazon Redshift ユーザー名で新しいオプション DbUser を指定します。

```
conn_options = {
     // IAM-based JDBC URL
    "url": "jdbc:redshift:iam://<cluster name>:<region>/<database name>",
    "dbtable": dbtable,
    "redshiftTmpDir": redshiftTmpDir,
    "aws_iam_role": aws_iam_role,
    "DbUser": "<Redshift User name>" // required for IAM-based JDBC URL
    }

redshift_write = glueContext.write_dynamic_frame.from_options(
    frame=dyf,
    connection_type="redshift",
    connection_options=conn_options
)
```

```
redshift_read = glueContext.create_dynamic_frame.from_options(
    connection_type="redshift",
    connection_options=conn_options
)
```

> **Note**
>
> DynamicFrame は現在、GlueContext.create_dynamic_frame.from_options ワークフローに DbUser が含まれている IAM ベースの JDBC URL のみをサポートしています。

AWS Glue バージョン 3.0 からバージョン 4.0 への移行

AWS Glue 4.0 では、さまざまなオプションと設定を搭載した新しい JDBC ドライバーや Amazon Redshift Spark コネクタに ETL ジョブでアクセスできます。新しい Amazon Redshift コネクタとドライバーはパフォーマンスを念頭に置いて作成されており、データのトランザクションの一貫性が保たれます。これらの製品については、Amazon Redshift のドキュメントに記載されています。詳細については、以下を参照してください。

- Apache Spark 用の Amazon Redshift インテグレーション
- Amazon Redshift JDBC ドライバー、バージョン 2.1

テーブル/列名と識別子の制限

新しい Amazon Redshift Spark コネクタとドライバーで、Redshift テーブル名の要件が厳しくなっています。Amazon Redshift のテーブル名定義の詳細については、「名前と識別子」を参照してください。ジョブブックマークのワークフローは、ルールに一致しないテーブル名やスペースなどの特定の文字があると機能しない場合があります。

名前と識別子のルールに適合しない名前のレガシーテーブルがあり、ブックマーク (古い Amazon Redshift テーブルデータを再処理するジョブ) に問題がある場合は、テーブル名を変更することをお勧めします。詳細については、「ALTER TABLE の例」を参照してください。

データフレームのデフォルトの一時形式の変更

AWS Glue バージョン 3.0 の Spark コネクタでは、Amazon Redshift への書き込み時にデフォルトで tempformat が CSV に設定されます。一貫性を保つために、AWS Glue バージョン 3.0 でも、 DynamicFrame ではデフォルトで tempformat に CSV を使用するようになっています。

以前に Amazon Redshift Spark コネクタで Spark データフレーム API を直接使用していた場合は、DataframeReader/Writer オプションで tempformat を CSV に明示的に設定できます。それ以外の場合は、新しい Spark コネクタで tempformat はデフォルトで AVRO になります。

動作の変更: Amazon Redshift データ型 REAL を Spark データ型 DOUBLE ではなく FLOAT にマッピングする

AWS Glue バージョン 3.0 では、Amazon Redshift の REAL は Spark の DOUBLE 型に変換されます。新しい Amazon Redshift Spark コネクタでは、動作が更新され、Amazon Redshift の REAL 型が Spark の FLOAT 型に変換されたり、その逆の変換が行われたりするようになりました。従来のユースケースで引き続き Amazon Redshift の REAL 型を Spark の DOUBLE 型にマッピングする場合は、次の回避策を使用できます。

- DynamicFrame の場合、DynamicFrame.ApplyMapping で Float 型を Double 型にマッピングします。Dataframe の場合、cast を使用する必要があります。

コードサンプル:

```
dyf_cast = dyf.apply_mapping([('a', 'long', 'a', 'long'), ('b', 'float', 'b',
 'double')])
```

VARBYTE データ型の処理

AWS Glue 3.0 および Amazon Redshift データ型を使用する場合、AWS Glue 3.0 は Amazon Redshift VARBYTE を Spark STRING 型に変換します。ただし、最新の Amazon Redshift Spark コネクタは VARBYTE データ型をサポートしていません。この制限を回避するには、VARBYTE 列をサポートされているデータ型に変換する Redshift ビューを作成できます。次に、新しいコネクタを使用して、元のテーブルではなくこのビューからデータをロードします。これにより、VARBYTE データへのアクセスを維持しながら互換性が確保されます。

Redshift クエリの例:

```
CREATE VIEW view_name AS SELECT FROM_VARBYTE(varbyte_column, 'hex') FROM table_name
```

Kafka 接続

Kafka 接続を使用すると、Data Catalog テーブルに格納されている情報を使用するか、データストリームに直接アクセスするための情報を指定することにより、Kafka データストリームへ読み込む

または書き込むことができます。接続は、Kafka クラスターまたは Amazon Managed Streaming for Apache Kafka クラスターをサポートします。Kafka から Spark DataFrame に情報を読み込み、AWS Glue DynamicFrame に変換できます。DynamicFrames を JSON 形式で Kafka に書き込むことができます。データストリームに直接アクセスする場合は、これらのオプションを使用して、データストリームへのアクセス方法に関する情報を提供します。

`getCatalogSource` または `create_data_frame_from_catalog` を使用して Kafka ストリーミングソースからレコードを消費するか、`getCatalogSink` または `write_dynamic_frame_from_catalog` を使用して Kafka にレコードを書き込み、ジョブに Data Catalog データベースおよびテーブル名の情報があり、それを使用して Kafka ストリーミングソースから読み込むためのいくつかの基本パラメータを取得できる場合。`getSource`、`getCatalogSink`、`getSourceWithFormat`、`getSinkWithFormat`、`createDataF` を使用する場合、ここで説明する接続オプションを使用し、これらの基本パラメータを指定する必要があります。

`GlueContext` クラスで指定されたメソッドの次の引数を使用し、Kafka の接続オプションを指定できます。

- Scala
  - `connectionOptions`: `getSource`、`createDataFrameFromOptions`、`getSink` で使用
  - `additionalOptions`: `getCatalogSource`、`getCatalogSink` で使用
  - `options`: `getSourceWithFormat`、`getSinkWithFormat` で使用
- Python
  - `connection_options`: `create_data_frame_from_options`、`write_dynamic_frame_from_options` で使用
  - `additional_options`: `create_data_frame_from_catalog`、`write_dynamic_frame_from_catalog` で使用
  - `options`: `getSource`、`getSink` で使用

ストリーミング ETL ジョブに関する注意事項と制限事項については、「the section called “ストリーミング ETL に関する注意と制限”」を参照してください。

Kafka の設定

インターネット経由で利用可能な Kafka ストリームに接続するための AWS の前提条件はありません。

AWS Glue Kafka 接続を作成して、接続認証情報を管理できます。詳細については、「the section called “Kafka データストリームの接続の作成”」を参照してください。AWS Glue ジョブ設定で、*connectionName* を追加ネットワーク接続として指定し、メソッド呼び出しで connectionName パラメータに *connectionName* を指定します。

場合によっては、追加の前提条件を設定する必要があります。

- IAM 認証で Amazon Managed Streaming for Apache Kafka を使用する場合は、適切な IAM 設定が必要になります。
- Amazon VPC で Amazon Managed Streaming for Apache Kafka を使用する場合は、適切な Amazon VPC 設定が必要になります。Amazon VPC 接続情報を提供する AWS Glue 接続を作成する必要があります。AWS Glue 接続を追加ネットワーク接続として含めるには、ジョブ設定が必要です。

ストリーミング ETL ジョブの前提条件の詳細については、「the section called “ストリーミング ETL ジョブ”」を参照してください。

例: Kafka ストリームからの読み込み

the section called “forEachBatch” と組み合わせて使用します。

Kafka ストリーミングソースの例 :

```
kafka_options =
    { "connectionName": "ConfluentKafka",
      "topicName": "kafka-auth-topic",
      "startingOffsets": "earliest",
      "inferSchema": "true",
      "classification": "json"
    }
data_frame_datasource0 =
 glueContext.create_data_frame.from_options(connection_type="kafka",
 connection_options=kafka_options)
```

例: Kafka ストリームへの書き込み

Kafka への書き込み例:

getSink メソッドを使った例:

```
data_frame_datasource0 =
```

```
glueContext.getSink(
 connectionType="kafka",
 connectionOptions={
  JsonOptions("""{
   "connectionName": "ConfluentKafka",
   "classification": "json",
   "topic": "kafka-auth-topic",
   "typeOfData": "kafka"}
  """)},
 transformationContext="dataframe_ApacheKafka_node1711729173428")
 .getDataFrame()
```

write_dynamic_frame.from_options メソッドを使った例:

```
kafka_options =
    { "connectionName": "ConfluentKafka",
      "topicName": "kafka-auth-topic",
      "classification": "json"
    }
data_frame_datasource0 =
 glueContext.write_dynamic_frame.from_options(connection_type="kafka",
 connection_options=kafka_options)
```

Kafka 接続オプションのリファレンス

読み込むとき、"connectionType": "kafka" に次の接続オプションを使用します。

- "bootstrap.servers" (必須) ブートストラップサーバーの URL のリスト (例: b-1.vpc-test-2.o4q88o.c6.kafka.us-east-1.amazonaws.com:9094)。このオプションは API 呼び出しで指定するか、データカタログ内のテーブルメタデータで定義する必要があります。
- "security.protocol" (必須) ブローカーと通信するために使用されるプロトコル。使用できる値は、"SSL" または "PLAINTEXT" です。
- "topicName" (必須) サブスクライブするトピックのカンマ区切りリスト。"topicName"、"assign"、または "subscribePattern" の中から、いずれか 1 つのみを指定する必要があります。
- "assign": (必須) 消費する特定の TopicPartitions を指定する JSON 文字列。"topicName"、"assign"、または "subscribePattern" の中から、いずれか 1 つのみを指定する必要があります。

  例: '{"topicA":[0,1],"topicB":[2,4]}'

- "subscribePattern": (必須) サブスクライブする先のトピックリストを識別する Java の正規表現文字列。"topicName"、"assign"、または "subscribePattern" の中から、いずれか 1 つのみを指定する必要があります。

  例: 'topic.*'

- "classification" (必須) レコード内のデータで使用されるファイル形式。データカタログを通じて提供されていない限り、必須です。
- "delimiter" (オプション) classification が CSV の場合に使用される値の区切り文字。デフォルトは「,」です。
- "startingOffsets": (オプション) Kafka トピック内で、データの読み取りを開始する位置 使用できる値は、"earliest" または "latest" です。デフォルト値は "latest" です。
- "startingTimestamp": (オプション、AWS Glue バージョン 4.0 以降でのみサポート) Kafka トピック内で、データの読み込みを開始するレコードのタイムスタンプ。指定できる値は UTC 形式 (yyyy-mm-ddTHH:MM:SSZ のパターン) のタイムスタンプ文字列です (Z は UTC タイムゾーンのオフセットを +/- で表します。例: 「2023-04-04T08:00:00-04:00」)。

  注: AWS Glue ストリーミングスクリプトの接続オプションリストには、「startingOffsets」または「startingTimestamp」のいずれかしか表示できません。これらのプロパティの両方を含めると、ジョブが失敗します。

- "endingOffsets": (オプション) バッチクエリの終了位置。設定が可能な値は、"latest" または、各 TopicPartition の終了オフセットを指定する JSON 文字列のいずれかです。

  JSON 文字列の場合、{"topicA":{"0":23,"1":-1},"topicB":{"0":-1}} の形式を使用します。オフセットとして値 -1 を設定する場合、"latest" の意味になります。

- "pollTimeoutMs": (オプション) Spark ジョブエグゼキュータで、Kafka からデータをポーリングする際のタイムアウト値 (ミリ秒単位)。デフォルト値は 600000 です。
- "numRetries": (オプション) Kafka オフセットのフェッチが失敗したと判断される前の再試行回数。デフォルト値は 3 です。
- "retryIntervalMs": (オプション) Kafka オフセットのフェッチを開始するまでの待機時間 (ミリ秒)。デフォルト値は 10 です。
- "maxOffsetsPerTrigger": (オプション) 処理されるオフセットの最大数を、トリガー間隔ごとのレート上限で指定する値。指定されたオフセットの合計数は、異なるボリュームの topicPartitions 間で均等に分割されます。デフォルト値は「null」です。この場合、コンシューマーは既知の最新のオフセットまで、すべてのオフセットを読み取ります。

- "minPartitions": (オプション) Kafka から読み取ることを想定する、最小のパーティション数。デフォルト値は「null」です。これは、Spark パーティションの数が Kafka パーティションの数に等しいことを意味します。
- "includeHeaders": (オプション) Kafka ヘッダーを含めるかどうかを決定します。このオプションが「true」に設定されている場合、データ出力には、「glue_streaming_kafka_headers」という名前で Array[Struct(key: String, value: String)] 型の列が追加されます。デフォルト値は『false』です。このオプションは、AWS Glue バージョン 3.0 以降でのみ使用可能です。
- "schema": (inferSchema に false を設定した場合は必須) ペイロードの処理に使用するスキーマ。分類が avro である場合、提供されるスキーマには Avro スキーマ形式を使用する必要があります。分類が avro 以外の場合、提供されるスキーマは DDL スキーマ形式である必要があります。

  以下に、スキーマの例を示します。

  Example in DDL schema format

  ```
  'column1' INT, 'column2' STRING , 'column3' FLOAT
  ```

  Example in Avro schema format

  ```
  {
  "type":"array",
  "items":
  {
  "type":"record",
  "name":"test",
  "fields":
  [
    {
      "name":"_id",
      "type":"string"
    },
    {
      "name":"index",
      "type":
      [
        "int",
        "string",
        "float"
      ]
    }
  ]
  ```

```
}
}
```

- "inferSchema": (オプション) デフォルト値は「false」です。「true」に設定すると、実行時に、スキーマが foreachbatch 内のペイロードから検出されます。
- "avroSchema": (非推奨) Avro 形式を使用する場合に、Avro データのスキーマを指定するために使用されるパラメータです。このパラメータは非推奨となりました。schema パラメータを使用します。
- "addRecordTimestamp": (オプション) このオプションを「true」に設定すると、トピックが対応するレコードを受信した時刻を表示する「__src_timestamp」という列が、データ出力に追加で表示されます。デフォルト値は、「false」です。このオプションは AWS Glue のバージョン 4.0 以降でサポートされています。
- "emitConsumerLagMetrics": (オプション) このオプションを「true」に設定すると、バッチごとに、トピックが受信した最も古いレコードと、それが AWS Glue で CloudWatch に到着した時間との、間隔のメトリクスが出力されます。このメトリクスの名前は「glue.driver.streaming.maxConsumerLagInMs」です。デフォルト値は、「false」です。このオプションは AWS Glue バージョン 4.0 以降でサポートされています。

書き込むとき、"connectionType": "kafka" に次の接続オプションを使用します。

- "connectionName" (必須) Kafka クラスターへの接続に使用される AWS Glue 接続の名前 (Kafka ソースと同様)。
- "topic" (必須) トピック列が存在する場合、トピック設定オプションが設定されていない限り、指定された行を Kafka に書き込む際にトピック列の値がトピックとして使用されます。つまり、topic 設定オプションはトピック列をオーバーライドします。
- "partition" (オプション) 有効なパーティション番号が指定された場合、レコードの送信時にその partition が使用されます。

  パーティションが指定されずに key が存在する場合、キーのハッシュを使用してパーティションが選択されます。

  key と partition のどちらも存在しない場合、そのパーティションに少なくとも batch.size バイトが生成された際に変更内容のスティッキーパーティション分割に基づいてパーティションが選択されます。
- "key" (オプション) partition が null の場合、パーティション分割に使用されます。

- "classification" (オプション) レコードのデータに使用されるファイル形式。JSON、CSV、Avro のみをサポートしています。

  Avro 形式を使用すると、シリアル化するためにカスタムの AvroSchema を提供できますが、シリアル化解除する場合にもソースに提供する必要があることに注意してください。それ以外の場合、デフォルトで Apache AvroSchema を使用してシリアル化します。

さらに、Kafka プロデューサーの設定パラメーターを更新することにより、必要に応じて Kafka シンクを微調整できます。接続オプションには許可リストがないことに注意してください。すべてのキー値のペアはそのままシンクに保持されます。

ただし、有効にならないオプションの小さな拒否リストがあります。詳細については、「Apache 固有の設定」を参照してください。

Azure Cosmos DB 接続

AWS Glue for Spark を使用すると、AWS Glue 4.0 以降のバージョンの NoSQL API を使用して、Azure Cosmos DB の既存のコンテナとの間で読み取りと書き込みを行うことができます。SQL クエリを使用して、Azure Cosmos DB から何を読み取るかを定義できます。AWS Glue 接続を通じて AWS Secrets Manager に保存されている Azure Cosmos DB キーを使用して Azure Cosmos DB に接続します。

Azure Cosmos DB for NoSQL の詳細については、Azure のドキュメントを参照してください。

Azure Cosmos DB 接続の設定

AWS Glue から Azure Cosmos DB に接続するには、Azure Cosmos DB キーを作成して AWS Secrets Manager シークレットに保存し、そのシークレットを Azure Cosmos DB AWS Glue 接続に関連付ける必要があります。

前提条件:

- Azure では、AWS Glue で使用する Azure Cosmos DB キー (cosmosKey) を特定または生成する必要があります。詳細については、Azure ドキュメントの「Azure Cosmos DB のデータへのアクセスをセキュリティで保護する」を参照してください。

Azure Cosmos DB に対する接続を設定するには:

1. AWS Secrets Manager で、Azure Cosmos DB キーを使用してシークレットを作成します。Secrets Manager でシークレットを作成するには、AWS Secrets Manager ドキュメントの

「AWS Secrets Manager シークレットを作成する」にあるチュートリアルに従ってください。シークレットを作成したら、次のステップのためにシークレット名 *secretName* を保存しておきます。

- [key/value ペア] を選択する際に、*cosmosKey* という値を持つキー `spark.cosmos.accountKey` のペアを作成します。

2. AWS Glue コンソールで、「the section called “AWS Glue 接続の追加”」にあるステップに従って接続を作成します。接続を作成したら、将来的に AWS Glue で使用するために、接続名 *connectionName* を維持します。

   - [接続タイプ] を選択する際に、[Azure Cosmos DB] を選択します。
   - [AWS Secret] をクリックして、*secretName* を入力します。

AWS Glue Azure Cosmos DB 接続を作成した後、AWS Glue ジョブを実行する前に次のステップを実行する必要があります。

- AWS Glue ジョブに関連付けられている IAM ロールに *secretName* を読み取るアクセス許可を付与します。
- AWS Glue ジョブ設定で、追加のネットワーク接続として *connectionName* を指定します。

Azure Cosmos DB for NoSQL コンテナからの読み取り

前提条件:

- 読み取り元とする Azure Cosmos DB for NoSQL コンテナ。コンテナの識別情報が必要になります。

  Azure Cosmos for NoSQL コンテナは、データベースとコンテナによって識別されます。Azure Cosmos for NoSQL API に接続する際には、データベース *cosmosDBName* とコンテナ *cosmosContainerName* の名前を指定する必要があります。
- 認証およびネットワークの場所の情報を提供するように設定された AWS Glue Azure Cosmos DB 接続。これを取得するには、前の手順の「Azure Cosmos DB に対する接続を設定するには」のステップを実行します。AWS Glue 接続、*connectionName* の名前が必要になります。

例:

```
azurecosmos_read = glueContext.create_dynamic_frame.from_options(
```

```
    connection_type="azurecosmos",
    connection_options={
    "connectionName": connectionName,
    "spark.cosmos.database": cosmosDBName,
    "spark.cosmos.container": cosmosContainerName,
    }
)
```

SELECT SQL クエリを指定して、DynamicFrame に返される結果をフィルタリングすることもできます。query を設定する必要があります。

例:

```
azurecosmos_read_query = glueContext.create_dynamic_frame.from_options(
    connection_type="azurecosmos",
    connection_options={
        "connectionName": "connectionName",
        "spark.cosmos.database": cosmosDBName,
        "spark.cosmos.container": cosmosContainerName,
        "spark.cosmos.read.customQuery": "query"
    }
)
```

Azure Cosmos DB for NoSQL コンテナへの書き込み

この例では、既存の DynamicFrame である *dynamicFrame* から Azure Cosmos DB に情報を書き込みます。コンテナに既に情報がある場合、AWS Glue は DynamicFrame からのデータを付加します。コンテナ内の情報のスキーマが、書き込む情報と異なる場合、エラーが発生します。

前提条件:

- 書き込み先とする Azure Cosmos DB テーブル。コンテナの識別情報が必要になります。接続メソッドを呼び出す前にコンテナを作成する必要があります。

  Azure Cosmos for NoSQL コンテナは、データベースとコンテナによって識別されます。Azure Cosmos for NoSQL API に接続する際には、データベース *cosmosDBName* とコンテナ *cosmosContainerName* の名前を指定する必要があります。
- 認証およびネットワークの場所の情報を提供するように設定された AWS Glue Azure Cosmos DB 接続。これを取得するには、前の手順の「Azure Cosmos DB に対する接続を設定するには」のステップを実行します。AWS Glue 接続、*connectionName* の名前が必要になります。

例:

```
azurecosmos_write = glueContext.write_dynamic_frame.from_options(
    frame=dynamicFrame,
    connection_type="azurecosmos",
    connection_options={
    "connectionName": connectionName,
    "spark.cosmos.database": cosmosDBName,
    "spark.cosmos.container": cosmosContainerName
)
```

Azure Cosmos DB 接続オプションのリファレンス

- connectionName — 必須。読み込み/書き込みに使用されます。認証およびネットワークの場所の情報を接続方法に提供するように設定された AWS Glue Azure Cosmos DB 接続の名前。
- spark.cosmos.database — 必須。読み込み/書き込みに使用されます。有効な値: データベース名。Azure Cosmos DB for NoSQL のデータベース名。
- spark.cosmos.container — 必須。読み込み/書き込みに使用されます。有効な値: コンテナ名。Azure Cosmos DB for NoSQL のコンテナ名。
- spark.cosmos.read.customQuery — 読み取りに使用。有効な値: SELECT SQL クエリ。読み取るドキュメントを選択するためのカスタムクエリ。

Azure SQL 接続

AWS Glue for Spark を使用して、AWS Glue 4.0 以降のバージョンの Azure SQL Managed Instances のテーブルからの読み取りとテーブルへの書き込みを行うことができます。SQL クエリを使用して、Azure SQL から何を読み取るかを定義できます。AWS Glue 接続を介して AWS Secrets Manager で保存されているユーザーおよびパスワードの認証情報を使用して Azure SQL に接続します。

Azure SQL の詳細については、Azure SQL のドキュメントを参照してください。

Azure SQL 接続の設定

AWS Glue から Azure SQL に接続するには、Azure SQL 認証情報を作成して AWS Secrets Manager シークレットに保存し、そのシークレットを Azure SQL AWS Glue 接続に関連付ける必要があります。

Azure SQL に対する接続を設定するには:

1. AWS Secrets Manager で、Azure SQL 認証情報を使用してシークレットを作成します。Secrets Manager でシークレットを作成するには、AWS Secrets Manager ドキュメントの「AWS Secrets Manager シークレットを作成する」にあるチュートリアルに従ってください。シークレットを作成したら、次のステップのためにシークレット名 *secretName* を保存しておきます。

   - [key/value ペア] を選択する際に、*azuresqlUsername* という値を持つキー user のペアを作成します。
   - [key/value ペア] を選択する際に、*azuresqlPassword* という値を持つキー password のペアを作成します。

2. AWS Glue コンソールで、「the section called “AWS Glue 接続の追加”」にあるステップに従って接続を作成します。接続を作成したら、将来的に AWS Glue で使用するために、接続名 *connectionName* を維持します。

   - [接続タイプ] を選択する際に、[Azure SQL] を選択します。
   - Azure SQL URL を指定する場合は、JDBC エンドポイント URL を入力します。

     URL は、次のような形式になります:
     `jdbc:sqlserver://`*databaseServerName*`:`*databasePort*`;databaseName=`*azuresqlDBnam*

     AWS Glue には次の URL プロパティが必要です。

     - `databaseName` – 接続先の Azure SQL のデフォルトデータベース。

     Azure SQL Managed Instances の JDBC URL の詳細については、Microsoft のドキュメントを参照してください。

   - [AWS Secret] をクリックして、*secretName* を入力します。

AWS Glue Azure SQL 接続を作成した後、AWS Glue ジョブを実行する前に次のステップを実行する必要があります。

- AWS Glue ジョブに関連付けられている IAM ロールに *secretName* を読み取るアクセス許可を付与します。
- AWS Glue ジョブ設定で、追加のネットワーク接続として *connectionName* を指定します。

Azure SQL テーブルからの読み取り

前提条件:

- 読み取り元とする Azure SQL テーブル。テーブルの識別情報である *databaseName* および *tableIdentifier* が必要になります。

  Azure SQL テーブルは、データベース、スキーマ、テーブル名によって識別されます。Azure SQL に接続する際には、データベース名とテーブル名を指定する必要があります。スキーマがデフォルトの「public」でない場合は、スキーマも指定する必要があります。データベースは、*connectionName* の URL プロパティ、dbtable を通じたスキーマおよびテーブル名を介して指定されます。
- 認証情報を提供するように設定された AWS Glue Azure SQL 接続。認証情報を設定するには、前の手順「Azure SQL に対する接続を設定するには」のステップを実行します。AWS Glue 接続、*connectionName* の名前が必要になります。

例:

```
azuresql_read_table = glueContext.create_dynamic_frame.from_options(
    connection_type="azuresql",
    connection_options={
        "connectionName": "connectionName",
        "dbtable": "tableIdentifier"
    }
)
```

SELECT SQL クエリを指定して、DynamicFrame に返される結果をフィルタリングすることもできます。query を設定する必要があります。

例:

```
azuresql_read_query = glueContext.create_dynamic_frame.from_options(
    connection_type="azuresql",
    connection_options={
        "connectionName": "connectionName",
        "query": "query"
    }
)
```

Azure SQL テーブルへの書き込み

この例では、既存の DynamicFrame である *dynamicFrame* から Azure SQL に情報を書き込みます。テーブルに既に情報がある場合、AWS Glue は DynamicFrame からのデータを付加します。

前提条件:

- 書き込み先とする Azure SQL テーブル。テーブルの識別情報である *databaseName* および *tableIdentifier* が必要になります。

  Azure SQL テーブルは、データベース、スキーマ、テーブル名によって識別されます。Azure SQL に接続する際には、データベース名とテーブル名を指定する必要があります。スキーマがデフォルトの「public」でない場合は、スキーマも指定する必要があります。データベースは、*connectionName* の URL プロパティ、dbtable を通じたスキーマおよびテーブル名を介して指定されます。
- Azure SQL 認証情報。認証情報を設定するには、前の手順「Azure SQL に対する接続を設定するには」のステップを実行します。AWS Glue 接続、*connectionName* の名前が必要になります。

例:

```
azuresql_write = glueContext.write_dynamic_frame.from_options(
    connection_type="azuresql",
    connection_options={
        "connectionName": "connectionName",
        "dbtable": "tableIdentifier"
    }
)
```

Azure SQL 接続オプションのリファレンス

- connectionName — 必須。読み込み/書き込みに使用されます。認証情報を接続方法に提供するように設定された AWS Glue Azure SQL 接続の名前。
- databaseName — 読み込み/書き込みに使用されます。有効な値: Azure SQL データベース名。接続先の Azure SQL のデータベースの名前。
- dbtable — 書き込みの場合は必須。query が指定されていない限り、読み取りの場合は必須。読み込み/書き込みに使用されます。有効な値: Azure SQL テーブルの名前、またはピリオドで区切られたスキーマ/テーブル名の組み合わせ。接続先のテーブルを識別するテーブルとスキーマを指定するために使用されます。デフォルトのスキーマは「public」です。テーブルがデフォルト以外のスキーマにある場合は、この情報を *schemaName*.*tableName* の形式で入力します。

- `query` — 読み取りに使用。Azure SQL から読み取るときに何を取得するかを定義する Transact-SQL SELECT クエリ。詳細については、Microsoft のドキュメントを参照してください。

BigQuery 接続

AWS Glue for Spark を使用して、AWS Glue 4.0 以降のバージョンの Google BigQuery のテーブルからの読み取りとテーブルへの書き込みを行うことができます。Google SQL クエリを使用して BigQuery からの読み取りを行うことができます。AWS Glue 接続を介して AWS Secrets Manager で保存されている認証情報を使用して BigQuery に接続します。

Google BigQuery の詳細については、Google Cloud BigQuery のウェブサイトを参照してください。

BigQuery 接続を設定する

AWS Glue から Google BigQuery へ接続するには、Google Cloud Platform の認証情報を作成して AWS Secrets Manager シークレットに保存し、そのシークレットを Google BigQuery AWS Glue 接続に関連付ける必要があります。

BigQuery への接続を設定するには:

1. Google Cloud Platform で、関連するリソースを作成して特定します。
   - 接続したい BigQuery テーブルを含む GCP プロジェクトを作成または特定します。
   - BigQuery API を有効にします。詳細については、「Use the BigQuery Storage Read API to read table data」を参照してください。
2. Google Cloud Platform で、サービスアカウントの認証情報を作成してエクスポートします。

   BigQuery 認証情報ウィザードを使用すると、「認証情報の作成」というステップを迅速に実行できます。

   GCP でサービスアカウントを作成するには、「サービス アカウントを作成する」にあるチュートリアルに従ってください。
   - プロジェクトを選択するときは、BigQuery テーブルを含むプロジェクトを選択します。
   - サービスアカウントの GCP IAM ロールを選択するときは、BigQuery テーブルの読み取り、書き込み、作成を行う BigQuery ジョブを実行するための適切な権限を付与するロールを追加または作成します。

サービスアカウントの認証情報を作成するには、「サービス アカウント キーを作成する」にあるチュートリアルに従ってください。

- キータイプを選択するときは、[JSON] を選択します。

これで、サービスアカウントの認証情報が記載された JSON ファイルがダウンロードされたはずです。これは次のように表示されます。

```
{
  "type": "service_account",
  "project_id": "*****",
  "private_key_id": "*****",
  "private_key": "*****",
  "client_email": "*****",
  "client_id": "*****",
  "auth_uri": "https://accounts.google.com/o/oauth2/auth",
  "token_uri": "https://oauth2.googleapis.com/token",
  "auth_provider_x509_cert_url": "https://www.googleapis.com/oauth2/v1/certs",
  "client_x509_cert_url": "*****",
  "universe_domain": "googleapis.com"
}
```

3. ダウンロードした認証情報ファイルを base64 でエンコードします。AWS CloudShell セッションなどでは、コマンドラインから `cat` *`credentialsFile.json`* `| base64 -w 0` コマンドを実行してこれを実行できます。このコマンドの出力 *`credentialString`* を保持してください。
4. AWS Secrets Manager で、Google Cloud Platform の認証情報を使用してシークレットを作成します。Secrets Manager でシークレットを作成するには、AWS Secrets Manager ドキュメントの「AWS Secrets Manager シークレットを作成する」にあるチュートリアルに従ってください。シークレットを作成したら、次のステップのためにシークレット名 *`secretName`* を保存しておきます。

   - キーと値のペアを選択するときは、*`credentialString`* という値を持つキー `credentials` のペアを作成します。
5. AWS Glue データカタログで、「the section called “AWS Glue 接続の追加”」にある手順に従って接続を作成します。接続を作成したら、次のステップのために接続名 *`connectionName`* を保存しておきます。

- [接続タイプ] を選択するときは、Google BigQuery を選択してください。
- [AWS Secret] をクリックして、*secretName* を入力します。

6. AWS Glue ジョブに関連付けられている IAM ロールに *secretName* を読み取るアクセス許可を付与します。
7. AWS Glue ジョブ設定で、追加のネットワーク接続として *connectionName* を指定します。

BigQuery テーブルから読み取る

前提条件:

- 読み取る対象の BigQuery テーブル。フォーム `[dataset].[table]` には BigQuery のテーブル名とデータセット名が必要です。これに *tableName* と名前を付けます。
- BigQuery テーブルの請求プロジェクト。そのためには、プロジェクトの名前、*parentProject* が必要です。請求元の親プロジェクトがない場合は、テーブルを含むプロジェクトを使用してください。
- BigQuery の認証情報。「AWS Glue で接続認証情報を管理するには」の手順を実行して、認証情報を設定します。AWS Glue 接続、*connectionName* の名前が必要になります。

例:

```
bigquery_read = glueContext.create_dynamic_frame.from_options(
    connection_type="bigquery",
    connection_options={
        "connectionName": "connectionName",
        "parentProject": "parentProject",
        "sourceType": "table",
        "table": "tableName",
    }
```

クエリを指定して、DynamicFrame に返される結果をフィルタリングすることもできます。`query`、`sourceType`、`viewsEnabled` および `materializationDataset` を設定する必要があります。

例:

追加の前提条件:

BigQuery がクエリのマテリアライズドビューを書き込むことのできる BigQuery データセット、*materializationDataset* を作成または特定する必要があります。

materializationDataset にテーブルを作成するには、サービスアカウントに適切な GCP IAM アクセス許可を付与する必要があります。

```
glueContext.create_dynamic_frame.from_options(
            connection_type="bigquery",
            connection_options={
                "connectionName": "connectionName",
                "materializationDataset": materializationDataset,
                "parentProject": "parentProject",
                "viewsEnabled": "true",
                "sourceType": "query",
                "query": "select * from bqtest.test"
            }
        )
```

BigQuery テーブルへ書き込む

この例では BigQuery サービスに直接書き込みます。BigQuery は「間接」書き込み方法もサポートしています。間接書き込みの設定の詳細については、「the section called “Google BigQuery での間接書き込みを使用する”」を参照してください。

前提条件:

- 書き込む対象の BigQuery テーブル。フォーム [dataset].[table] には BigQuery のテーブル名とデータセット名が必要です。自動的に作成される新しいテーブル名を指定することもできます。これに *tableName* と名前を付けます。
- BigQuery テーブルの請求プロジェクト。そのためには、プロジェクトの名前、*parentProject* が必要です。請求元の親プロジェクトがない場合は、テーブルを含むプロジェクトを使用してください。
- BigQuery の認証情報。「AWS Glue で接続認証情報を管理するには」の手順を実行して、認証情報を設定します。AWS Glue 接続、*connectionName* の名前が必要になります。

例:

```
bigquery_write = glueContext.write_dynamic_frame.from_options(
    frame=frameToWrite,
```

```
    connection_type="bigquery",
    connection_options={
        "connectionName": "connectionName",
        "parentProject": "parentProject",
        "writeMethod": "direct",
        "table": "tableName",
    }
)
```

BigQuery 接続オプションのリファレンス

- `project` — デフォルト: Google Cloud サービスアカウントのデフォルト。読み込み/書き込みに使用されます。テーブルに関連付けられている Google Cloud プロジェクトの名前。
- `table` — (必須) 読み込み/書き込みに使用されます。`[[project:]dataset.]` フォーマット内の BigQuery テーブルの名前。
- `dataset` — `table` オプションで定義されていない場合は必須。読み込み/書き込みに使用されます。BigQuery テーブルを含むデータセットの名前。
- `parentProject` — デフォルト: Google Cloud サービスアカウントのデフォルト。読み込み/書き込みに使用されます。請求に使用される `project` に関連付けられた Google Cloud プロジェクトの名前。
- `sourceType` — 読み取りに使用。読み取り時には必須です。有効値: `table`、`query` AWS Glue にテーブルで読み取るかクエリで読み取るかを通知します。
- `materializationDataset` — 読み取りに使用。有効な値: 文字列。ビューのマテリアライゼーションの保存に使用される BigQuery データセットの名前。
- `viewsEnabled` — 読み取りに使用。デフォルト: false。有効な値: true、false。BigQuery がビューを使用するかどうかを設定します。
- `query` — 読み取りに使用。`viewsEnabled` が true のときに使用されます。GoogleSQL DQL クエリ。
- `temporaryGcsBucket` — 書き込みに使用。`writeMethod` がデフォルト (`indirect`) に設定されている場合は必須です。BigQuery への書き込み中に中間形式のデータを保存するために使用される Google Cloud Storage バケットの名前。
- `writeMethod` - デフォルト: `indirect`。有効な値: `direct`、`indirect`。書き込みに使用。データの書き込みに使用する方法を指定します。
  - `direct` に設定すると、コネクタは BigQuery Storage Write API を使用して書き込みを行います。

- `indirect` に設定すると、コネクタは Google Cloud Storage に書き込み、読み取り操作を使用して BigQuery に転送します。Google Cloud サービスアカウントには適切な GCS アクセス許可が必要です。

Google BigQuery での間接書き込みを使用する

この例では、Google Cloud ストレージにデータを書き込み、それを Google BigQuery にコピーする間接書き込みを使用しています。

前提条件:

Google Cloud Storage の一時的なバケット、*`temporaryBucket`* が必要になります。

AWS Glue の GCP サービスアカウントの GCP IAM ロールには、*`temporaryBucket`* にアクセスするための適切な GCS アクセス許可が必要です。

追加設定:

BigQuery による間接書き込みを設定するには:

1. the section called “BigQuery を設定する” にアクセスして検索するか、GCP 認証情報の JSON ファイルを再ダウンロードします。ジョブで使用する Google BigQuery AWS Glue 接続の AWS Secrets Manager シークレットである *`secretName`* を特定します。
2. 認証情報 JSON ファイルを、適切に安全な Amazon S3 の場所にアップロードします。ファイルへのパス、*`s3secretpath`* は、今後の手順に備えて保持してください。
3. *`secretName`* を編集し、`spark.hadoop.google.cloud.auth.service.account.json.keyfile` キーを追加します。*`s3secretpath`* に値を設定します。
4. AWS Glue ジョブに、*`s3secretpath`* にアクセスするための Amazon S3 IAM アクセス許可を付与します。

一時的な GCS バケットの場所を書き込みメソッドに指定できるようになりました。`indirect` がこれまでデフォルトだったように、`writeMethod` を指定する必要はありません。

```
bigquery_write = glueContext.write_dynamic_frame.from_options(
    frame=frameToWrite,
    connection_type="bigquery",
    connection_options={
```

```
        "connectionName": "connectionName",
        "parentProject": "parentProject",
        "temporaryGcsBucket": "temporaryBucket",
        "table": "tableName",
    }
)
```

JDBC 接続

特定の (通常はリレーショナル) データベースタイプは、JDBC 標準による接続をサポートします。JDBC の詳細については、Java JDBC API に関するドキュメントを参照してください。AWSGlue は JDBC コネクタを介した特定のデータベースへの接続をネイティブにサポートしています。JDBC ライブラリは AWS Glue Spark ジョブで提供されます。AWS Glue ライブラリを使用してこれらのデータベースタイプに接続すると、標準のオプションセットにアクセスできます。

JDBC connectionType の値には次のようなものがあります。

- `"connectionType": "sqlserver"`: Microsoft SQL Server データベースへの接続を指定します。
- `"connectionType": "mysql"`: MySQL データベースへの接続を指定します。
- `"connectionType": "oracle"`: Oracle データベースへの接続を指定します。
- `"connectionType": "postgresql"`: PostgreSQL データベースへの接続を指定します。
- `"connectionType": "redshift"`: Amazon Redshift データベースへの接続を指定します。詳細については、「the section called “Redshift 接続”」を参照してください。

次の表に、AWS Glue がサポートする JDBC ドライバーのバージョンを示します。

製品	Glue 5.0 の JDBC ドライバーのバージョン	Glue 4.0 の JDBC ドライバーのバージョン	Glue 3.0 の JDBC ドライバのバージョン	Glue 0.9、1.0、2.0 の JDBC ドライバのバージョン
Microsoft SQL Server	10.2.0	9.4.0	7.x	6.x
MySQL	8.0.33	8.0.23	8.0.23	5.1

製品	Glue 5.0 の JDBC ドライバーのバージョン	Glue 4.0 の JDBC ドライバーのバージョン	Glue 3.0 の JDBC ドライバのバージョン	Glue 0.9、1.0、2.0 の JDBC ドライバのバージョン
Oracle Database	23.3.0.23.09	21.7	21.1	11.2
PostgreSQL	42.7.3	42.3.6	42.2.18	42.1.x
Amazon Redshift *	redshift-jdbc42-2.1.0.29	redshift-jdbc42-2.1.0.16	redshift-jdbc41-1.2.12.1017	redshift-jdbc41-1.2.12.1017

* Amazon Redshift 接続タイプの場合は、書式設定オプションを含め、JDBC 接続用に接続オプションに含まれるその他のオプション名/値ペアはすべて、基になる SparkSQL DataSource に直接渡されます。AWS Glue 4.0 以降のバージョンの AWS Glue with Spark ジョブでは、Amazon Redshift の AWS Glue ネイティブコネクタは Apache Spark の Amazon Redshift インテグレーションを使用します。詳細については、「Apache Spark 用の Amazon Redshift の統合」を参照してください。以前のバージョンでは、「Spark 用の Amazon Redshift データソース」を参照してください。

JDBC を使用して Amazon RDS データストアに接続するように Amazon VPC を設定するには、「the section called “Amazon RDS データストアに接続するための VPC の 設定”」を参照してください。

> **Note**
>
> AWS Glue ジョブは、1 回の実行で 1 つのサブネットにのみ関連付けられます。これは、同じジョブで複数のデータソースに接続する能力に影響する可能性があります。この動作は JDBC ソースに限定されません。

トピック

- JDBC 接続オプションのリファレンス
- sampleQuery を使用
- カスタム JDBC ドライバーを使用
- JDBC テーブルからの並列読み取り

- AWS Glue から Amazon RDS データストアに JDBC 接続するための Amazon VPC の設定

JDBC 接続オプションのリファレンス

既に JDBC AWS Glue 接続を定義している場合、URL、ユーザー、パスワードなど、その中で定義されている設定プロパティを再利用できるため、コードで接続オプションとして指定する必要はありません。この機能は、AWS Glue バージョン 3.0 以降でのみ有効です。これを行うには、次の接続プロパティを使用します。

- `"useConnectionProperties"`: この設定を接続から使用するときは、「true」に設定します。
- `"connectionName"`: この設定を取得する接続名を入力します。接続は、ジョブと同じリージョンで定義されている必要があります。

JDBC 接続では、次の接続オプションを使用します。

- `"url"`: (必須) データベースの JDBC URL。
- `"dbtable"`: (必須) 読み込み元のデータベーステーブル。データベース内でスキーマをサポートする JDBC データストアの場合、`schema.table-name` を指定します。スキーマを指定しない場合、デフォルトの「パブリック」スキーマが使用されます。
- `"user"`: (必須) 接続時に使用するユーザー名。
- `"password"`: (必須) 接続時に使用するパスワード。
- (オプション) 次のオプションを使用すると、カスタム JDBC ドライバーを指定できます。AWS Glue がネイティブでサポートしていないドライバーを使用する必要がある場合は、これらのオプションを使用します。

  ソースおよびターゲットが同じデータベース製品の場合でも、ETL ジョブは、データソースおよびターゲットに対して、異なるバージョンの JDBC ドライバーを使用することができます。これにより、異なるバージョンのソースデータベースとターゲットデータベース間でデータを移行できます。これらのオプションを使用するには、最初に JDBC ドライバーの JAR ファイルを Amazon S3 にアップロードする必要があります。

  - `"customJdbcDriverS3Path"`: カスタム JDBC ドライバーの Amazon S3 パス。
  - `"customJdbcDriverClassName"`: JDBC ドライバーのクラス名。
- `"bulkSize"`: (オプション) JDBC ターゲットへのバルクロードを高速化するためのパラレル挿入を構成するために使用します。データの書き込みまたは挿入時に使用する並列度の整数値を指定します。このオプションは、Arch User Repository (AUR) などのデータベースへの書き込みのパフォーマンスを向上させるのに役立ちます。

- "hashfield" (オプション) JDBC テーブルから並列で読み込むときにデータをパーティションに分割するために使用される JDBC テーブル内の列の名前を指定するために使用される文字列。「hashfield」または「hashexpression」を指定します。詳細については、「the section called “JDBC からの並列読み取り”」を参照してください。
- "hashexpression" (オプション) 整数を返す SQL SELECT 句。JDBC テーブルから並列で読み込むときに、JDBC テーブル内のデータをパーティションに分割するために使用されます。「hashfield」または「hashexpression」を指定します。詳細については、「the section called “JDBC からの並列読み取り”」を参照してください。
- "hashpartitions" (オプション) 正の整数。JDBC テーブルから並列で読み込むときに、JDBC テーブルの並列読み込み数を指定するために使用されます。デフォルト: 7。詳細については、「the section called “JDBC からの並列読み取り”」を参照してください。
- "sampleQuery": (オプション) カスタム SQL クエリステートメント。テーブル内の情報のサブセットを指定して、テーブルの内容のサンプルを取得するために使用されます。データを考慮せずに設定すると、DynamicFrame メソッドよりも効率が悪くなり、タイムアウトやメモリ不足エラーが発生する可能性があります。詳細については、「the section called “sampleQuery を使用”」を参照してください。
- "enablePartitioningForSampleQuery": (オプション) ブール値。デフォルト: false。sampleQuery の指定時に、JDBC テーブルから並列で読み込むことを可能にするために使用されます。true に設定すると、AWS Glue でパーティショニング条件を追加するためには、**sampleQuery** が「where」または「and」で終わる必要があります。詳細については、「the section called “sampleQuery を使用”」を参照してください。
- "sampleSize": (オプション) 正の整数。サンプルクエリによって返される行数を制限します。enablePartitioningForSampleQuery が true の場合にのみ動作します。パーティショニングが有効になっていない場合は、代わりに sampleQuery で "limit x" を追加してサイズを制限します。詳細については、「the section called “sampleQuery を使用”」を参照してください。

sampleQuery を使用

このセクションでは、sampleQuery、sampleSize、enablePartitioningForSampleQuery を使用する方法について説明します。

sampleQuery ではデータセットのいくつかの行を効率的にサンプリングできます。デフォルトでは、クエリは単一のエグゼキューターによって実行されます。データを考慮せずに設定すると、DynamicFrame メソッドよりも効率が悪くなり、タイムアウトやメモリ不足エラーが発生する可能性があります。ETL パイプラインの一部として基盤となるデータベースで SQL を実行することは、通常、パフォーマンス上の目的でのみ必要になります。データセットの数行をプレビューする

場合は、the section called “show” を使用することを検討してください。SQL を使用してデータセットを変換する場合は、the section called “toDF” を使用して DataFrame フォーム内のデータに対して SparkSQL 変換を定義することを検討してください。

クエリではさまざまなテーブルを操作できますが、dbtable はやはり必須です。

sampleQuery を使用してテーブルのサンプルを取得する

デフォルトの sampleQuery の動作を使用してデータのサンプルを取得する場合、AWS Glue は大きなスループットを期待しないため、クエリは 1 つの工グゼキューターで実行されます。提供するデータを制限し、パフォーマンス上の問題を引き起こさないようにするには、SQL に LIMIT 句を指定することをお勧めします。

Example パーティション化せずに sampleQuery を使用する

次のコード例は、パーティション化せずに sampleQuery を使用する方法を示しています。

```
//A full sql query statement.
val query = "select name from $tableName where age > 0 limit 1"
val connectionOptions = JsonOptions(Map(
    "url" -> url,
    "dbtable" -> tableName,
    "user" -> user,
    "password" -> password,
    "sampleQuery" -> query ))
val dyf = glueContext.getSource("mysql", connectionOptions)
          .getDynamicFrame()
```

sampleQuery を大きなデータセットに対して使用する

大きなデータセットを読み取る場合は、JDBC パーティション化を有効にして、テーブルを並列にクエリする必要がある場合があります。詳細については、「the section called “JDBC からの並列読み取り”」を参照してください。sampleQuery を JDBC パーティショニングで使用する場合は、enablePartitioningForSampleQuery を true に設定します。この機能を有効にするには、sampleQuery にいくつかの変更を加える必要があります。

sampleQuery で JDBC パーティショニングを使用するときに、パーティショニング条件を追加するためには、AWS Glue でクエリが「where」または「and」で終わる必要があります。

JDBC テーブルから並列で読み込むときに sampleQuery の結果を制限したい場合は、LIMIT 句を指定するのではなく、"sampleSize" パラメータを設定してください。

Example JDBC パーティション化で sampleQuery を使用する

次のコード例は、JDBC パーティション化で sampleQuery を使用する方法を示しています。

```
//note that the query should end with "where" or "and" if use with JDBC partitioning.
val query = "select name from $tableName where age > 0 and"

//Enable JDBC partitioning by setting hashfield.
//to use sampleQuery with partitioning, set enablePartitioningForSampleQuery.
//use sampleSize to limit the size of returned data.
val connectionOptions = JsonOptions(Map(
    "url" -> url,
    "dbtable" -> tableName,
    "user" -> user,
    "password" -> password,
    "hashfield" -> primaryKey,
    "sampleQuery" -> query,
    "enablePartitioningForSampleQuery" -> true,
    "sampleSize" -> "1" ))
val dyf = glueContext.getSource("mysql", connectionOptions)
          .getDynamicFrame()
```

注意と制限

サンプルクエリはジョブブックマークと一緒に使用することはできません。両方の設定を指定しても、ブックマークの状態は無視されます。

カスタム JDBC ドライバーを使用

次のコード例は、カスタム JDBC ドライバーを使用して JDBC データベースから読み書きする方法を示しています。データベース製品の 1 つのバージョンから読み取り、同じ製品の新しいバージョンに書き込んでいます。

Python

```
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext, SparkConf
from awsglue.context import GlueContext
from awsglue.job import Job
import time
from pyspark.sql.types import StructType, StructField, IntegerType, StringType
```

```
sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session

# Construct JDBC connection options
connection_mysql5_options = {
    "url": "jdbc:mysql://<jdbc-host-name>:3306/db",
    "dbtable": "test",
    "user": "admin",
    "password": "pwd"}

connection_mysql8_options = {
    "url": "jdbc:mysql://<jdbc-host-name>:3306/db",
    "dbtable": "test",
    "user": "admin",
    "password": "pwd",
    "customJdbcDriverS3Path": "s3://amzn-s3-demo-bucket/mysql-connector-
java-8.0.17.jar",
    "customJdbcDriverClassName": "com.mysql.cj.jdbc.Driver"}

connection_oracle11_options = {
    "url": "jdbc:oracle:thin:@//<jdbc-host-name>:1521/ORCL",
    "dbtable": "test",
    "user": "admin",
    "password": "pwd"}

connection_oracle18_options = {
    "url": "jdbc:oracle:thin:@//<jdbc-host-name>:1521/ORCL",
    "dbtable": "test",
    "user": "admin",
    "password": "pwd",
    "customJdbcDriverS3Path": "s3://amzn-s3-demo-bucket/ojdbc10.jar",
    "customJdbcDriverClassName": "oracle.jdbc.OracleDriver"}

# Read from JDBC databases with custom driver
df_mysql8 = glueContext.create_dynamic_frame.from_options(connection_type="mysql",

 connection_options=connection_mysql8_options)

# Read DynamicFrame from MySQL 5 and write to MySQL 8
df_mysql5 = glueContext.create_dynamic_frame.from_options(connection_type="mysql",

 connection_options=connection_mysql5_options)
```

```
glueContext.write_from_options(frame_or_dfc=df_mysql5, connection_type="mysql",
                               connection_options=connection_mysql8_options)

# Read DynamicFrame from Oracle 11 and write to Oracle 18
df_oracle11 =
 glueContext.create_dynamic_frame.from_options(connection_type="oracle",

 connection_options=connection_oracle11_options)
glueContext.write_from_options(frame_or_dfc=df_oracle11, connection_type="oracle",
                               connection_options=connection_oracle18_options)
```

Scala

```
import com.amazonaws.services.glue.GlueContext
import com.amazonaws.services.glue.MappingSpec
import com.amazonaws.services.glue.errors.CallSite
import com.amazonaws.services.glue.util.GlueArgParser
import com.amazonaws.services.glue.util.Job
import com.amazonaws.services.glue.util.JsonOptions
import com.amazonaws.services.glue.DynamicFrame
import org.apache.spark.SparkContext
import scala.collection.JavaConverters._


object GlueApp {
  val MYSQL_5_URI: String = "jdbc:mysql://<jdbc-host-name>:3306/db"
  val MYSQL_8_URI: String = "jdbc:mysql://<jdbc-host-name>:3306/db"
  val ORACLE_11_URI: String = "jdbc:oracle:thin:@//<jdbc-host-name>:1521/ORCL"
  val ORACLE_18_URI: String = "jdbc:oracle:thin:@//<jdbc-host-name>:1521/ORCL"

  // Construct JDBC connection options
  lazy val mysql5JsonOption = jsonOptions(MYSQL_5_URI)
  lazy val mysql8JsonOption = customJDBCDriverJsonOptions(MYSQL_8_URI, "s3://amzn-
s3-demo-bucket/mysql-connector-java-8.0.17.jar", "com.mysql.cj.jdbc.Driver")
  lazy val oracle11JsonOption = jsonOptions(ORACLE_11_URI)
  lazy val oracle18JsonOption = customJDBCDriverJsonOptions(ORACLE_18_URI, "s3://
amzn-s3-demo-bucket/ojdbc10.jar", "oracle.jdbc.OracleDriver")

  def main(sysArgs: Array[String]): Unit = {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)
    val args = GlueArgParser.getResolvedOptions(sysArgs, Seq("JOB_NAME").toArray)
```

```
    Job.init(args("JOB_NAME"), glueContext, args.asJava)

    // Read from JDBC database with custom driver
    val df_mysql8: DynamicFrame = glueContext.getSource("mysql",
 mysql8JsonOption).getDynamicFrame()

    // Read DynamicFrame from MySQL 5 and write to MySQL 8
    val df_mysql5: DynamicFrame = glueContext.getSource("mysql",
 mysql5JsonOption).getDynamicFrame()
    glueContext.getSink("mysql", mysql8JsonOption).writeDynamicFrame(df_mysql5)

    // Read DynamicFrame from Oracle 11 and write to Oracle 18
    val df_oracle11: DynamicFrame = glueContext.getSource("oracle",
 oracle11JsonOption).getDynamicFrame()
    glueContext.getSink("oracle", oracle18JsonOption).writeDynamicFrame(df_oracle11)

    Job.commit()
  }

  private def jsonOptions(url: String): JsonOptions = {
    new JsonOptions(
      s"""{"url": "${url}",
         |"dbtable":"test",
         |"user": "admin",
         |"password": "pwd"}""".stripMargin)
  }

  private def customJDBCDriverJsonOptions(url: String, customJdbcDriverS3Path:
 String, customJdbcDriverClassName: String): JsonOptions = {
    new JsonOptions(
      s"""{"url": "${url}",
         |"dbtable":"test",
         |"user": "admin",
         |"password": "pwd",
         |"customJdbcDriverS3Path": "${customJdbcDriverS3Path}",
         |"customJdbcDriverClassName" :
 "${customJdbcDriverClassName}"}""".stripMargin)
  }
}
```

JDBC テーブルからの並列読み取り

AWS Glue がデータを並列に読み込むように、JDBC テーブルのプロパティを設定できます。特定のプロパティを設定するときに、データの論理パーティションに対して、並列 SQL クエリを実行するように AWS Glue に指示します。ハッシュフィールド、またはハッシュ式を設定して、パーティション分割を制御できます。データにアクセスするために使用される、並列読み込みの数を制御することもできます。

JDBC テーブルからの並列読み取りは、パフォーマンスを向上させる最適化手法です。この手法が適切であることを特定するプロセスの詳細については、「AWS 規範的ガイダンス」の「Best practices for performance tuning AWS Glue for Apache Spark jobs」ガイドにある「Reduce the amount of data scan」を参照してください。

並列読み込みするに、テーブル構造のパラメータフィールドにキーと値のペアを設定できます。テーブルのパラメータフィールドに値を設定するには、JSON 表記を使用します。テーブルのプロパティを編集する詳しい方法については、「テーブルの詳細の表示と管理」を参照してください。ETL (抽出、変換、ロード) メソッド `create_dynamic_frame_from_options` および `create_dynamic_frame_from_catalog` を呼び出すときに、並列読み込みを有効にすることもできます。これらのメソッドのオプションの指定の詳細については、「from_options」および「from_catalog」を参照してください。

JDBC テーブル、つまり、ほとんどの基本データが JDBC データストアのテーブルで、このメソッドを使用できます。Amazon Redshift および Amazon S3 テーブルの読み取り時は、これらのプロパティは無視されます。

hashfield

> データをパーティションに分割するために使用する JDBC テーブルの列の名前を `hashfield` に設定します。最良の結果を得るために、この列の値は、パーティション間にデータを分散する、値の均等分散である必要があります。この列には任意のデータ型を使用できます。AWS Glue は、この列によってパーティション分割されたデータの並列読み込みを実行する、重複しないクエリを生成します。たとえば、データが毎月均等に分散される場合、`month` 列を使用して、各月のデータを並行して読み込むことができます。

```
'hashfield': 'month'
```

AWS Glue は、フィールド値をパーティション番号にハッシュするクエリを作成して、すべてのパーティションにクエリを並行して実行します。パーティションのテーブル読み込みに独自のクエリを使用するには、hashfield の代わりに hashexpression を指定します。

hashexpression

hashexpression を、すべての番号を返す SQL 式 (JDBC データベースエンジンの文法に準拠する) に設定します。単純式は、テーブル内の任意の数値列の名前です。AWS Glue は、パーティションデータの WHERE 句で、hashexpression を使用して並行して JDBC データを読み込む SQL クエリを生成します。

たとえば、数値列 customerID を使用して、パーティション分割されたデータを顧客番号を使用して読み込みます。

```
'hashexpression': 'customerID'
```

AWS Glue がパーティション分割を制御するようにするには、hashexpression の代わりに hashfield を指定します。

hashpartitions

JDBC テーブルの並列読み込みの数を hashpartitions に設定します。このプロパティが設定されていない場合、デフォルト値は、7 です。

たとえば、並列読み込みの数に 5 を設定すると、AWS Glue は 5 (以下) のクエリを使用して、データを読み込みます。

```
'hashpartitions': '5'
```

AWS Glue から Amazon RDS データストアに JDBC 接続するための Amazon VPC の設定

JDBC を使用して Amazon RDS のデータベースに接続すると、追加の設定を行う必要があります。AWS Glue コンポーネントが Amazon RDS を通信できるようにするには、Amazon VPC で Amazon RDS データストアへのアクセスを設定する必要があります。AWS Glue がコンポーネント間で通信できるようにするには、すべての TCP ポートに対して自己参照のインバウンドルールを持つセキュリティグループを指定します。自己参照ルールを作成することにより、ソースを VPC の

同じセキュリティグループに制限することができます。自己参照ルールは、VPC をすべてのネットワークに開放しません。VPC のデフォルトのセキュリティグループには、すでに ALL Traffic (すべてのトラフィック) の自己参照インバウンドルールがある場合があります。

AWS Glue と Amazon RDS データストア間のアクセスを設定する方法

1. AWS Management Console にサインインし、Amazon RDS コンソール https://console.aws.amazon.com/rds/ を開きます。
2. Amazon RDS コンソールで、Amazon RDS データベースへのアクセスを管理するために使用するセキュリティグループを特定します。

   左側のナビゲーションペインで [データベース] を選択し、メインペインのリストから接続するインスタンスを選択します。

   データベースの詳細ページで、[接続とセキュリティ] タブで [VPC セキュリティグループ] を見つけます。
3. ネットワークアーキテクチャに基づいて、AWS Glue サービスにアクセスを許可するための変更が最適な関連セキュリティグループを特定します。今後の参照のため、*database-security-group* の名前を保存します。適切なセキュリティグループがない場合、Amazon RDS ドキュメントの「セキュリティグループを作成して VPC 内の DB インスタンスへのアクセスを提供する」の指示に従ってください。
4. AWS Management Console にサインインして、Amazon VPC コンソール (https://console.aws.amazon.com/vpc/) を開きます。
5. Amazon VPC コンソールで、*database-security-group* を更新する方法を特定します。

   左側のナビゲーションペインで [セキュリティグループ] を選択し、メインペインのリストから *database-security-group* を選択します。
6. *database-security-group* および *database-sg-id* のセキュリティグループ ID を特定します。今後の参照のために保存します。

   セキュリティグループの詳細ページで、[セキュリティグループ ID] を探します。
7. *database-security-group* のインバウンドルールを変更し、AWS Glue コンポーネントが通信できるように自己参照ルールを追加します。具体的には、[タイプ] は `All TCP`、[プロトコル] は TCP、[ポート範囲] にすべてのポートが含まれ、[ソース] は *database-sg-id* であるルールを追加または存在することを確認します。[ソース] に入力したセキュリティグループが、編集中のセキュリティグループと同じであることを確認します。

   セキュリティグループの詳細ページで、[インバウンドルールの編集] を選択します。

インバウンドルールは次のようになります。

タイプ	プロトコル	ポート範囲	ソース
すべての TCP	TCP	0 ~ 65535	*database-sg-id*

8. アウトバウンドトラフィックのルールを追加します。

   セキュリティグループの詳細ページで、[アウトバウンドルールの編集] を選択します。

   セキュリティグループがすべてのアウトバウンドトラフィックを許可する場合、個別のルールは必要ありません。例:

タイプ	プロトコル	ポート範囲	デスティネーション
すべてのトラフィック	すべて	すべて	0.0.0.0/0

   ネットワークアーキテクチャがアウトバウンドトラフィックを制限するために設計されている場合、次のアウトバウンドルールを作成してください。

   [タイプ] は `All TCP`、[プロトコル] は `TCP`、[ポート範囲] にすべてのポートが含まれ、[送信先] は *database-sg-id* である自己参照ルールを作成します。[送信先] に入力したセキュリティグループが、編集中のセキュリティグループと同じであることを確認します。

   Amazon S3 VPC エンドポイントを使用している場合、VPC から Amazon S3 へのトラフィックを許可する HTTPS ルールを追加します。[タイプ] は `HTTPS`、[プロトコル] は `TCP`、[ポート範囲] は `443`、[送信先] は Amazon S3 ゲートウェイエンドポイントのマネージドプレフィックスリストの ID である *s3-prefix-list-id* となるルールを作成します。プレフィックスのリストと Amazon S3 ゲートウェイエンドポイントの詳細については、Amazon VPC ドキュメントの「Gateway endpoints for Amazon S3」をご参照ください。

   例:

タイプ	プロトコル	ポート範囲	デスティネーション
すべての TCP	TCP	0 ~ 65535	*database-sg-id*

タイプ	プロトコル	ポート範囲	デスティネーション
HTTPS	TCP	443	*s3-prefix-list-id*

MongoDB 接続

AWS Glue for Spark を使用して、AWS Glue 4.0 以降のバージョンの MongoDB および MongoDB Atlas のテーブルからの読み取りとテーブルへの書き込みを行うことができます。AWS Glue 接続を介して AWS Secrets Manager で保存されているユーザー名およびパスワードの認証情報を使用して MongoDB に接続できます。

MongoDB の詳細については、MongoDB のドキュメントを参照してください。

MongoDB 接続の設定

AWS Glue から MongoDB に接続するには、MongoDB 認証情報、*mongodbUser* および *mongodbPass* が必要です。

AWS Glue から MongoDB に接続するには、いくつかの前提条件を満たす必要がある場合があります。

- MongoDB インスタンスが Amazon VPC 内にある場合は、トラフィックがパブリックインターネットを経由することなく、AWS Glue ジョブが MongoDB インスタンスと通信できるように Amazon VPC を設定します。

  Amazon VPC で、AWS Glue がジョブの実行中に使用する [VPC]、[サブネット]、および [セキュリティグループ] を特定または作成します。さらに、MongoDB インスタンスとこの場所の間のネットワークトラフィックを許可するように Amazon VPC が設定されているようにする必要があります。ネットワークレイアウトに基づいて、セキュリティグループルール、ネットワーク ACL、NAT ゲートウェイ、およびピアリング接続の変更が必要になる場合があります。

その後、MongoDB で使用できるように AWS Glue を設定する作業に進むことができます。

MongoDB に対する接続を設定するには:

1. 必要に応じて、AWS Secrets Manager で、MongoDB 認証情報を使用してシークレットを作成します。Secrets Manager でシークレットを作成するには、AWS Secrets Manager ドキュメントの「AWS Secrets Manager シークレットを作成する」にあるチュートリアルに従ってくださ

い。シークレットを作成したら、次のステップのためにシークレット名 *secretName* を保存しておきます。

- [key/value ペア] を選択する際に、*mongodbUser* という値を持つキー username のペアを作成します。

  [key/value ペア] を選択する際に、*mongodbPass* という値を持つキー password のペアを作成します。

2. AWS Glue コンソールで、「the section called “AWS Glue 接続の追加”」にあるステップに従って接続を作成します。接続を作成したら、将来的に AWS Glue で使用するために、接続名 *connectionName* を維持します。

   - [接続タイプ] を選択する際には、[MongoDB] または [MongoDB Atlas] を選択します。
   - [MongoDB URL] または [MongoDB Atlas URL] を選択する場合は、MongoDB インスタンスのホスト名を入力します。

     MongoDB URL は、mongodb://*mongoHost*:*mongoPort*/*mongoDBname* の形式で指定されます。

     MongoDB Atlas URL は、mongodb+srv://*mongoHost*:*mongoPort*/*mongoDBname* の形式で指定されます。

     接続にデフォルトのデータベースを指定する場合、*mongoDBname* はオプションです。
   - Secrets Manager シークレットを作成することを選択した場合は、AWS Secrets Manager の [認証情報タイプ] を選択します。

     その後、[AWS シークレット] で *secretName* を入力します。
   - [ユーザー名とパスワード] を入力することを選択した場合は、*mongodbUser* および *mongodbPass* を入力します。
3. 次の状況では、追加の設定が必要になる場合があります。

   - Amazon VPC の AWS でホストされている MongoDB インスタンスの場合
     - MongoDB セキュリティ認証情報を定義する AWS Glue 接続に、Amazon VPC 接続に関する情報を提供する必要があります。接続を作成または更新する際に、[ネットワークオプション] で [VPC]、[サブネット]、および [セキュリティグループ] を設定します。

AWS Glue MongoDB 接続を作成した後、接続メソッドを呼び出す前に次のアクションを実行する必要があります。

- Secrets Manager シークレットを作成することを選択した場合は、AWS Glue ジョブに関連付けられた IAM ロールに *secretName* を読み取るための許可を付与します。
- AWS Glue ジョブ設定で、追加のネットワーク接続として *connectionName* を指定します。

AWS Glue for Spark で AWS Glue MongoDB 接続を使用するには、接続メソッド呼び出しで connectionName オプションを指定します。あるいは、the section called “MongoDB との統合” のステップに従って、AWS Glue データカタログと組み合わせて接続を使用することもできます。

AWS Glue 接続を使用した MongoDB からの読み取り

前提条件:

- 読み取り元とする MongoDB コレクション。コレクションの識別情報が必要になります。

  MongoDB コレクションは、データベース名とコレクション名である *mongodbName* および *mongodbCollection* によって識別されます。
- 認証情報を提供するように設定された AWS Glue MongoDB 接続。認証情報を設定するには、前の手順「MongoDB に対する接続を設定するには」のステップを実行します。AWS Glue 接続、*connectionName* の名前が必要になります。

例:

```
mongodb_read = glueContext.create_dynamic_frame.from_options(
    connection_type="mongodb",
    connection_options={
        "connectionName": "connectionName",
        "database": "mongodbName",
        "collection": "mongodbCollection",
        "partitioner":
 "com.mongodb.spark.sql.connector.read.partitioner.SinglePartitionPartitioner",
        "partitionerOptions.partitionSizeMB": "10",
        "partitionerOptions.partitionKey": "_id",
        "disableUpdateUri": "false",
    }
)
```

MongoDB テーブルへの書き込み

この例では、既存の DynamicFrame である *dynamicFrame* から MongoDB に情報を書き込みます。

前提条件:

- 書き込み先とする MongoDB コレクション。コレクションの識別情報が必要になります。

  MongoDB コレクションは、データベース名とコレクション名である *mongodbName* および *mongodbCollection* によって識別されます。
- 認証情報を提供するように設定された AWS Glue MongoDB 接続。認証情報を設定するには、前の手順「MongoDB に対する接続を設定するには」のステップを実行します。AWS Glue 接続、*connectionName* の名前が必要になります。

例:

```
glueContext.write_dynamic_frame.from_options(
    frame=dynamicFrame,
    connection_type="mongodb",
    connection_options={
        "connectionName": "connectionName",
        "database": "mongodbName",
        "collection": "mongodbCollection",
        "disableUpdateUri": "false",
        "retryWrites": "false",
    },
)
```

MongoDB テーブルに対する読み書き

この例では、既存の DynamicFrame である *dynamicFrame* から MongoDB に情報を書き込みます。

前提条件:

- 読み取り元とする MongoDB コレクション。コレクションの識別情報が必要になります。

  書き込み先とする MongoDB コレクション。コレクションの識別情報が必要になります。

  MongoDB コレクションは、データベース名とコレクション名である *mongodbName* および *mongodbCollection* によって識別されます。
- MongoDB 認証情報である *mongodbUser* および *mongodbPassword*。

例:

Python

```
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext, SparkConf
from awsglue.context import GlueContext
from awsglue.job import Job
import time

## @params: [JOB_NAME]
args = getResolvedOptions(sys.argv, ['JOB_NAME'])

sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session

job = Job(glueContext)
job.init(args['JOB_NAME'], args)

output_path = "s3://some_bucket/output/" + str(time.time()) + "/"
mongo_uri = "mongodb://<mongo-instanced-ip-address>:27017"
mongo_ssl_uri = "mongodb://<mongo-instanced-ip-address>:27017"
write_uri = "mongodb://<mongo-instanced-ip-address>:27017"

read_mongo_options = {
    "uri": mongo_uri,
    "database": "mongodbName",
    "collection": "mongodbCollection",
    "username": "mongodbUsername",
    "password": "mongodbPassword",
    "partitioner": "MongoSamplePartitioner",
    "partitionerOptions.partitionSizeMB": "10",
    "partitionerOptions.partitionKey": "_id"}

ssl_mongo_options = {
    "uri": mongo_ssl_uri,
    "database": "mongodbName",
    "collection": "mongodbCollection",
    "ssl": "true",
    "ssl.domain_match": "false"
}

write_mongo_options = {
```

```
    "uri": write_uri,
    "database": "mongodbName",
    "collection": "mongodbCollection",
    "username": "mongodbUsername",
    "password": "mongodbPassword",
}

# Get DynamicFrame from MongoDB
dynamic_frame =
 glueContext.create_dynamic_frame.from_options(connection_type="mongodb",

 connection_options=read_mongo_options)

# Write DynamicFrame to MongoDB
glueContext.write_dynamic_frame.from_options(dynamicFrame,
 connection_type="mongodb", connection_options=write_mongo_options)

job.commit()
```

Scala

```
import com.amazonaws.services.glue.GlueContext
import com.amazonaws.services.glue.MappingSpec
import com.amazonaws.services.glue.errors.CallSite
import com.amazonaws.services.glue.util.GlueArgParser
import com.amazonaws.services.glue.util.Job
import com.amazonaws.services.glue.util.JsonOptions
import com.amazonaws.services.glue.DynamicFrame
import org.apache.spark.SparkContext
import scala.collection.JavaConverters._

object GlueApp {
  val DEFAULT_URI: String = "mongodb://<mongo-instanced-ip-address>:27017"
  val WRITE_URI: String = "mongodb://<mongo-instanced-ip-address>:27017"
  lazy val defaultJsonOption = jsonOptions(DEFAULT_URI)
  lazy val writeJsonOption = jsonOptions(WRITE_URI)
  def main(sysArgs: Array[String]): Unit = {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)
    val args = GlueArgParser.getResolvedOptions(sysArgs, Seq("JOB_NAME").toArray)
    Job.init(args("JOB_NAME"), glueContext, args.asJava)

    // Get DynamicFrame from MongoDB
```

```
    val dynamicFrame: DynamicFrame = glueContext.getSource("mongodb",
 defaultJsonOption).getDynamicFrame()

    // Write DynamicFrame to MongoDB
    glueContext.getSink("mongodb", writeJsonOption).writeDynamicFrame(dynamicFrame)

    Job.commit()
  }

  private def jsonOptions(uri: String): JsonOptions = {
    new JsonOptions(
      s"""{"uri": "${uri}",
         |"database":"mongodbName",
         |"collection":"mongodbCollection",
         |"username": "mongodbUsername",
         |"password": "mongodbPassword",
         |"ssl":"true",
         |"ssl.domain_match":"false",
         |"partitioner": "MongoSamplePartitioner",
         |"partitionerOptions.partitionSizeMB": "10",
         |"partitionerOptions.partitionKey": "_id"}""".stripMargin)
  }
}
```

MongoDB 接続オプションのリファレンス

MongoDB への接続を指定します。接続オプションは、ソース接続とシンク接続とで異なります。

これらの接続プロパティは、ソース接続とシンク接続の間で共有されます。

- connectionName — 読み込み/書き込みに使用されます。認証およびネットワークの情報を接続方法に提供するように設定された AWS Glue MongoDB 接続の名前。前のセクション「the section called “MongoDB の設定”」で説明したように AWS Glue 接続が設定されている場合、connectionName を入力することで、"uri"、"username"、および "password" 接続オプションを入力する必要がなくなります。
- "uri": (必須) 読み込み元の MongoDB ホスト (形式: mongodb://<host>:<port>)。AWS Glue 4.0 より前のバージョンの AWS Glue で使用されます。
- "connection.uri": (必須) 読み込み元の MongoDB ホスト (形式: mongodb://<host>:<port>)。AWS Glue 4.0 以降のバージョンで使用されます。
- "username": (必須) MongoDB のユーザー名。

- "password": (必須) MongoDB のパスワード。
- "database": (必須) 読み込み元の MongoDB データベース。このオプションは、ジョブスクリプトで glue_context.create_dynamic_frame_from_catalog を呼び出す際に、additional_options を介して渡すことも可能です。
- "collection": (必須) 読み込み元の MongoDB コレクション。このオプションは、ジョブスクリプトで glue_context.create_dynamic_frame_from_catalog を呼び出す際に、additional_options を介して渡すことも可能です。

"connectionType": "mongodb" ソースとする

"connectionType": "mongodb" をソースとして、次の接続オプションを使用します。

- "ssl": (オプション) true の場合、SSL 接続を開始します。デフォルト: false。
- "ssl.domain_match": (任意) true と ssl が true の場合 、ドメイン一致チェックが実行されます。デフォルト: true。
- "batchSize": (オプション): 内部バッチのカーソル内で使用される、バッチごとに返されるドキュメントの数。
- "partitioner": (オプション): MongoDB から入力データを読み取るためのパーティショナーのクラス名。コネクタには、次のパーティショナーがあります。
  - MongoDefaultPartitioner (デフォルト) (AWS Glue 4.0 ではサポートされていません)
  - MongoSamplePartitioner (MongoDB 3.2 以降が必要) (AWS Glue 4.0 ではサポートされていません)
  - MongoShardedPartitioner (AWS Glue 4.0 ではサポートされていません)
  - MongoSplitVectorPartitioner (AWS Glue 4.0 ではサポートされていません)
  - MongoPaginateByCountPartitioner (AWS Glue 4.0 ではサポートされていません)
  - MongoPaginateBySizePartitioner (AWS Glue 4.0 ではサポートされていません)
  - com.mongodb.spark.sql.connector.read.partitioner.SinglePartitionPartitioner
  - com.mongodb.spark.sql.connector.read.partitioner.ShardedPartitioner
  - com.mongodb.spark.sql.connector.read.partitioner.PaginateIntoPartitionsPartit
- "partitionerOptions" (オプション): 指定されたパーティショナーのオプション。各パーティショナーでは、次のオプションがサポートされています。
  - MongoSamplePartitioner: partitionKey, partitionSizeMB, samplesPerPartition
  - MongoShardedPartitioner: shardkey

- MongoSplitVectorPartitioner: partitionKey, partitionSizeMB
- MongoPaginateByCountPartitioner: partitionKey, numberOfPartitions
- MongoPaginateBySizePartitioner: partitionKey, partitionSizeMB

これらのオプションの詳細については、MongoDB のドキュメントの「Partitioner Configuration」を参照してください。

"connectionType": "mongodb" as Sink

"connectionType": "mongodb" をシンクとして、次の接続オプションを使用します。

- "ssl": (オプション) true の場合、SSL 接続を開始します。デフォルト: false。
- "ssl.domain_match": (任意) true と ssl が true の場合 、ドメイン一致チェックが実行されます。デフォルト: true。
- "extendedBsonTypes": (オプション) true が設定されている場合、MongoDB にデータを書き込む際に拡張 BSON 型を使用することを許可します。デフォルト: true。
- "replaceDocument": (オプション) true の場合、_id フィールドを含むデータセットを保存するときに、ドキュメント全体を置き換えます。false の場合、データセットのフィールドと一致するドキュメントのフィールドのみが更新されます。デフォルト: true。
- "maxBatchSize": (オプション): データを保存するときの一括オペレーションの最大バッチサイズ。デフォルトは 512 です。
- "retryWrites": (オプション): AWS Glue でネットワークエラーが発生した場合、特定の書き込みオペレーションを 1 回自動的に再試行します。

SAP HANA 接続

AWS Glue for Spark を使用して、AWS Glue 4.0 以降のバージョンの SAP HANA のテーブルからの読み取りとテーブルへの書き込みを行うことができます。SQL クエリを使用して、SAP HANA から何を読み取るかを定義できます。AWS Glue SAP HANA 接続を通じて AWS Secrets Manager に保存されている JDBC 認証情報を使用して SAP HANA に接続します。

SAP HANA JDBC の詳細については、SAP HANA のドキュメントを参照してください。

SAP HANA 接続の設定

AWS Glue から SAP HANA に接続するには、SAP HANA 認証情報を作成して AWS Secrets Manager シークレットに保存し、そのシークレットを SAP HANA AWS Glue 接続に関連付ける必

要があります。SAP HANA サービスと AWS Glue の間のネットワーク接続を設定する必要があります。

SAP HANA に接続するには、いくつかの前提条件を満たす必要がある場合があります。

- SAP HANA サービスが Amazon VPC 内にある場合は、トラフィックがパブリックインターネットを経由することなく、AWS Glue ジョブが SAP HANA サービスと通信できるように Amazon VPC を設定します。

  Amazon VPC で、AWS Glue がジョブの実行中に使用する [VPC]、[サブネット]、および [セキュリティグループ] を特定または作成します。さらに、SAP HANA エンドポイントとこの場所の間のネットワークトラフィックを許可するように Amazon VPC が設定されているようにする必要があります。ジョブでは、SAP HANA JDBC ポートとの TCP 接続を確立する必要があります。SAP HANA ポートの詳細については、SAP HANA のドキュメントを参照してください。ネットワークレイアウトに基づいて、セキュリティグループルール、ネットワーク ACL、NAT ゲートウェイ、およびピアリング接続の変更が必要になる場合があります。
- SAP HANA エンドポイントがインターネットにアクセスできる場合、追加の前提条件はありません。

SAP HANA に対する接続を設定するには:

1. AWS Secrets Manager で、SAP HANA 認証情報を使用してシークレットを作成します。Secrets Manager でシークレットを作成するには、AWS Secrets Manager ドキュメントの「AWS Secrets Manager シークレットを作成する」にあるチュートリアルに従ってください。シークレットを作成したら、次のステップのためにシークレット名 *secretName* を保存しておきます。

   - [key/value ペア] を選択する際に、*saphanaUsername* という値を持つキー `user` のペアを作成します。
   - [key/value ペア] を選択する際に、*saphanaPassword* という値を持つキー `password` のペアを作成します。
2. AWS Glue コンソールで、「the section called “AWS Glue 接続の追加”」にあるステップに従って接続を作成します。接続を作成したら、将来的に AWS Glue で使用するために、接続名 *connectionName* を維持します。

   - [接続タイプ] を選択する際に、[SAP HANA] を選択します。
   - [SAP HANA URL] を入力する場合は、インスタンスの URL を入力します。

SAP HANA JDBC URL の形式は
`jdbc:sap://saphanaHostname:saphanaPort/?databaseName=saphanaDBname,Paramete`
です

AWS Glue には次の JDBC URL パラメータが必要です。

- `databaseName` – 接続先の SAP HANA のデフォルトデータベース。

- [AWS Secret] をクリックして、*secretName* を入力します。

AWS Glue SAP HANA 接続を作成した後、AWS Glue ジョブを実行する前に次のステップを実行する必要があります。

- AWS Glue ジョブに関連付けられている IAM ロールに *secretName* を読み取るアクセス許可を付与します。
- AWS Glue ジョブ設定で、追加のネットワーク接続として *connectionName* を指定します。

SAP HANA テーブルからの読み取り

前提条件:

- 読み取り元とする SAP HANA テーブル。テーブルの識別情報が必要になります。

  テーブルは、SAP HANA テーブル名とスキーマ名 (形式: *schemaName*.*tableName*) を使用して指定できます。テーブルがデフォルトのスキーマ「public」にある場合、スキーマ名と「.」区切り文字は必要ありません。この *tableIdentifier* を呼び出します。データベースは `connectionName` の JDBC URL パラメータとして指定されることに注意してください。
- 認証情報を提供するように設定された AWS Glue SAP HANA 接続。認証情報を設定するには、前の手順「SAP HANA に対する接続を設定するには」のステップを実行します。AWS Glue 接続、*connectionName* の名前が必要になります。

例:

```
saphana_read_table = glueContext.create_dynamic_frame.from_options(
    connection_type="saphana",
    connection_options={
        "connectionName": "connectionName",
        "dbtable": "tableIdentifier",
    }
```

```
)
```

SELECT SQL クエリを指定して、DynamicFrame に返される結果をフィルタリングすることもできます。query を設定する必要があります。

例:

```
saphana_read_query = glueContext.create_dynamic_frame.from_options(
    connection_type="saphana",
    connection_options={
        "connectionName": "connectionName",
        "query": "query"
    }
)
```

SAP HANA テーブルへの書き込み

この例では、既存の DynamicFrame である *dynamicFrame* から SAP HANA に情報を書き込みます。テーブルに既に情報がある場合、AWS Glue はエラーになります。

前提条件:

- 書き込み先とする SAP HANA テーブル。

  テーブルは、SAP HANA テーブル名とスキーマ名 (形式: *schemaName*.*tableName*) を使用して指定できます。テーブルがデフォルトのスキーマ「public」にある場合、スキーマ名と「.」区切り文字は必要ありません。この *tableIdentifier* を呼び出します。データベースは connectionName の JDBC URL パラメータとして指定されることに注意してください。
- SAP HANA 認証情報。認証情報を設定するには、前の手順「SAP HANA に対する接続を設定するには」のステップを実行します。AWS Glue 接続、*connectionName* の名前が必要になります。

例:

```
options = {
    "connectionName": "connectionName",
    "dbtable": 'tableIdentifier'
}

    saphana_write = glueContext.write_dynamic_frame.from_options(
        frame=dynamicFrame,
```

```
        connection_type="saphana",
        connection_options=options
)
```

SAP HANA 接続オプションのリファレンス

- connectionName — 必須。読み込み/書き込みに使用されます。認証およびネットワークの情報を接続方法に提供するように設定された AWS Glue SAP HANA 接続の名前。
- databaseName — 読み込み/書き込みに使用されます。有効な値: SAP HANA のデータベースの名前。接続先のデータベースの名前。
- dbtable — 書き込みの場合は必須。query が指定されていない限り、読み取りの場合は必須。読み込み/書き込みに使用されます。有効な値: SAP HANA SQL FROM 句の内容。接続先の SAP HANA 内のテーブルを識別します。サブクエリなど、テーブル名以外の他の SQL を入力することもできます。詳細については、SAP HANA ドキュメントの「From 句」を参照してください。
- query — 読み取りに使用。SAP HANA から読み取るときに何を取得するかを定義する SAP HANA SQL SELECT クエリ。

Snowflake 接続

AWS Glue for Spark を使用して、AWS Glue 4.0 以降のバージョンの Snowflake のテーブルからの読み込みとテーブルへの書き込みを行うことができます。Snowflake からの読み込みは SQL クエリで行うことができます。ユーザーとパスワードを使用して Snowflake に接続できます。AWS Glue データカタログを使用して AWS Secrets Manager に保存されている Snowflake 認証情報を参照できます。AWS Glue for Spark のデータカタログ Snowflake 認証情報は、クローラー用のデータカタログ Snowflake 認証情報とは別に保存されます。Snowflake に接続するように設定された JDBC タイプの接続ではなく、SNOWFLAKE タイプの接続を選択する必要があります。

Snowflake の詳細については、「Snowflake ウェブサイト」を参照してください。AWS 上の Snowflake の詳細については、「Snowflake Data Warehouse on Amazon Web Services」を参照してください。

Snowflake 接続の設定

インターネット経由で利用できる Snowflake データベースに接続するための AWS の前提条件はありません。

オプションで、次の設定を実行して AWS Glue で接続認証情報を管理できます。

AWS Glue で接続認証情報を管理するには

1. Snowflake でユーザー *snowflakeUser* とパスワード *snowflakePassword* を作成します。
2. AWS Secrets Manager で、Snowflake の認証情報を使用してシークレットを作成します。Secrets Manager でシークレットを作成するには、AWS Secrets Manager ドキュメントの「AWS Secrets Manager シークレットを作成する」にあるチュートリアルに従ってください。シークレットを作成したら、次のステップのためにシークレット名 *secretName* を保存しておきます。
   - [キー/値のペア] をクリックして、USERNAME キーを使用して *snowflakeUser* のペアを作成します。
   - [キー/値のペア] をクリックして、PASSWORD キーを使用して *snowflakePassword* のペアを作成します。
   - キーと値のペアを選択するときに、Snowflake ウェアハウスにキー sfWarehouse を提供できます。
3. AWS Glue データカタログで、[接続]、[接続を作成] の順に選択して、接続を作成します。次の接続ウィザードの手順に従って、プロセスを完了します。
   - [データソース] を選択する際には、[Snowflake] を選択し、[次へ] を選択します。
   - ホストやポートなどの接続の詳細を入力します。ホストである [Snowflake URL] を入力して、Snowflake インスタンスの URL を指定します。通常、URL では *account_identifier*.snowflakecomputing.com という形式のホスト名を使用します。ただし、URL 形式は Snowflake アカウントタイプ (AWS、Azure、Snowflake-hosted など) によって異なる場合があります。
   - IAM サービスロールを選択する際には、ドロップダウンメニューから 選択します。これは、VPC が指定されている場合、AWS Secrets Manager にアクセスして IP を割り当てるために使用されるアカウントの IAM ロールです。
   - [AWS Secret] をクリックして、*secretName* を入力します。
4. ウィザードの次のステップで、Snowflake 接続のプロパティを設定します。
5. ウィザードの最後のステップで、設定を確認し、接続を作成するプロセスを完了します。

次のような状況では、以下が必要になることがあります。

- Amazon VPC の AWS でホストされている Snowflake の場合

- Snowflake には適切な Amazon VPC 設定が必要です。Amazon VPC の設定方法の詳細については、Snowflake ドキュメントの「AWS PrivateLink & Snowflake」を参照してください。
- AWS Glue には適切な Amazon VPC 設定が必要です。the section called “AWS Glue のインターフェイス VPC エンドポイント (AWS PrivateLink) の設定 ”。
- (Snowflake のセキュリティ認証情報を定義する AWS Secrets Manager シークレットの ID に加えて) Amazon VPC 接続情報を提供する AWS Glue データカタログ接続を作成する必要があります。前の項目でリンクされている Snowflake ドキュメントで説明されているように、AWS PrivateLink の使用時に URL が変更されます。
- ジョブ設定には、データカタログ接続を追加ネットワーク接続として含める必要があります。

Snowflake テーブルからの読み込み

前提条件: 読み込む目的の Snowflake テーブルがあること。Snowflake のテーブル名 *tableName* が必要になります。Snowflake の URL *snowflakeUrl*、ユーザー名 *snowflakeUser*、パスワード *snowflakePassword* が必要です。Snowflake ユーザーにデフォルトの名前空間が設定されていない場合は、Snowflake データベース名 *databaseName*、スキーマ名 *schemaName* が必要になります。さらに、Snowflake ユーザーにデフォルトのウェアハウスセットがない場合は、ウェアハウス名 *warehouseName* が必要になります。

以下に例を示します。

その他の前提条件: 「AWS Glue で接続認証情報を管理するには」の手順を実行して、*snowflakeUrl*、*snowflakeUsername*、および *snowflakePassword* を設定します。これらのステップを確認するには、前のセクション「the section called “Snowflake の設定”」を参照してください。接続するその他のネットワーク接続を選択するには、connectionName パラメータを使用します。

```
snowflake_read = glueContext.create_dynamic_frame.from_options(
  connection_type="snowflake",
  connection_options={
        "connectionName": "connectionName",
        "dbtable": "tableName",
        "sfDatabase": "databaseName",
        "sfSchema": "schemaName",
        "sfWarehouse": "warehouseName",
    }
)
```

さらに、autopushdown および query パラメータを使用して Snowflake テーブルの一部を読み込むことができます。これは、Spark に読み込まれた後に結果をフィルタリングするよりもはるかに効率的です。すべての売上が同じテーブルに格納されているが、分析する必要があるのは休日の特定の店舗の売上だけだという例を考えてみましょう。その情報がテーブルに格納されている場合は、述語プッシュダウンを使用して次のように結果を取得できます。

```
snowflake_node = glueContext.create_dynamic_frame.from_options(
    connection_type="snowflake",
    connection_options={
        "autopushdown": "on",
        "query": "select * from sales where store='1' and IsHoliday='TRUE'",
        "connectionName": "snowflake-glue-conn",
        "sfDatabase": "databaseName",
        "sfSchema": "schemaName",
        "sfWarehouse": "warehouseName",
    }
)
```

Snowflake テーブルへの書き込み

前提条件: 書き込み先の Snowflake データベース。現在の、または希望するテーブル名 *tableName* が必要です。Snowflake の URL *snowflakeUrl*、ユーザー名 *snowflakeUser*、パスワード *snowflakePassword* が必要です。Snowflake ユーザーにデフォルトの名前空間が設定されていない場合は、Snowflake データベース名 *databaseName*、スキーマ名 *schemaName* が必要になります。さらに、Snowflake ユーザーにデフォルトのウェアハウスセットがない場合は、ウェアハウス名 *warehouseName* が必要になります。

以下に例を示します。

その他の前提条件: 「AWS Glue で接続認証情報を管理するには」の手順を実行して、*snowflakeUrl*、*snowflakeUsername*、および *snowflakePassword* を設定します。これらのステップを確認するには、前のセクション「the section called “Snowflake の設定”」を参照してください。接続するその他のネットワーク接続を選択するには、connectionName パラメータを使用します。

```
glueContext.write_dynamic_frame.from_options(
    connection_type="snowflake",
    connection_options={
        "connectionName": "connectionName",
        "dbtable": "tableName",
```

```
        "sfDatabase": "databaseName",
        "sfSchema": "schemaName",
        "sfWarehouse": "warehouseName",
    },
)
```

Snowflake 接続オプションのリファレンス

Snowflake 接続タイプには次の接続オプションがあります。

このセクションのパラメータの一部はデータカタログ接続 (sfUrl、sfUser、sfPassword) から取得できます。その場合は指定する必要はありません。これを行うには、connectionName パラメータを含めます。

このセクションのパラメータの一部は、AWS Secrets Manager シークレット (sfUser、sfPassword) から取得できます。その場合は指定する必要はありません。シークレットは、sfUser および sfPassword キーの下にコンテンツを提供する必要があります。これを行うには、secretId パラメータを含めます。

以下のパラメータは、Snowflake に接続する際に一般的に使用されます。

- sfDatabase — Snowflake でユーザーデフォルトが設定されていない場合は必須です。読み込み/書き込みに使用されます。接続後にセッションに使用するデータベース。
- sfSchema — Snowflake でユーザーデフォルトが設定されていない場合は必須です。読み込み/書き込みに使用されます。接続後にセッションに使用するスキーマ。
- sfWarehouse — Snowflake でユーザーデフォルトが設定されていない場合は必須です。読み込み/書き込みに使用されます。接続後にセッションに使用するデフォルトの仮想ウェアハウス。
- sfRole — Snowflake でユーザーデフォルトが設定されていない場合は必須です。読み込み/書き込みに使用されます。接続後にセッションに使用するデフォルトのセキュリティロール。
- sfUrl — (必須) 読み込み/書き込みに使用されます。アカウントのホスト名を *account_identifier*.snowflakecomputing.com の形式で指定します。アカウント識別子の詳細については、Snowflake ドキュメントの「アカウント識別子」を参照してください。
- sfUser — (必須) 読み込み/書き込みに使用されます。Snowflake ユーザーのログイン名。
- sfPassword - (pem_private_key 指定がない場合は必須) 読み取り/書き込みに使用します。Snowflake ユーザーのパスワード。
- dbtable — フルテーブルで作業する場合は必須です。読み込み/書き込みに使用されます。読み込まれるテーブルまたはデータが書き込まれるテーブルの名前。読み込み時には、すべての列とレコードが取得されます。

- pem_private_key — 読み込み/書き込みに使用されます。暗号化されていない b64 でエンコードされたプライベートキーの文字列。Snowflake ユーザーのプライベートキー。 これを PEM ファイルからコピーするのが一般的です。詳細については、Snowflake ドキュメントの「キーペア認証とキーペアローテーション」を参照してください。
- query — クエリで読み込みを行う場合は必須です。読み込みに使用されます。実行する正確なクエリ (SELECT ステートメント)

以下のオプションを使用して、Snowflake への接続プロセス中の特定の動作を設定します。

- preactions — 読み込み/書き込みに使用されます。有効値: セミコロンで区切られた文字列形式の SQL ステートメントのリスト。SQL ステートメントは、AWS Glue と Snowflake の間でデータが転送される前に実行されます。ステートメントに %s が含まれる場合、%s は操作で参照されるテーブル名に置き換えられます。
- postactions — 読み込み/書き込みに使用されます。SQL ステートメントは、AWS Glue と Snowflake 間でデータが転送された後に実行されます。ステートメントに %s が含まれる場合、%s は操作で参照されるテーブル名に置き換えられます。
- autopushdown - デフォルト: "on"。有効な値: "on"、"off"。このパラメータは、自動クエリプッシュダウンを有効にするかどうかを制御します。プッシュダウンが有効になっている場合、Spark でクエリを実行すると、クエリの一部が Snowflake サーバーに「プッシュダウン」できる場合にクエリがプッシュダウンされます。これにより、一部のクエリのパフォーマンスが向上します。クエリをプッシュダウンできるかどうかについては、Snowflake ドキュメントの「プッシュダウン」を参照してください。

さらに、Snowflake Spark コネクタで使用できるオプションの一部が AWS Glue でサポートされている場合があります。Snowflake Spark コネクタで使用できるオプションの詳細については、Snowflake ドキュメントの「コネクタの構成オプションの設定」を参照してください。

Snowflake コネクタの制限事項

AWS Glue for Spark を使用してスノーフレークに接続することには、以下の制限があります。

- このコネクタはジョブブックマークをサポートしていません。ブックマークの詳細については、「the section called “ジョブのブックマークを使用した処理済みデータの追跡”」を参照してください。
- このコネクタは、create_dynamic_frame.from_catalog および write_dynamic_frame.from_catalog メソッドを使用した AWS Glue データカタログのテーブルを介した Snowflake の読み込みと書き込みをサポートしていません。

- このコネクタは、ユーザーとパスワード以外の認証情報による Snowflake への接続をサポートしていません。
- このコネクタはストリーミングジョブではサポートされていません。
- このコネクタは、情報を取得する (query パラメータを使用してなど) 際の SELECT ステートメントベースのクエリをサポートします。他の種類のクエリ (SHOW、DESC、DML ステートメントなど) はサポートされていません。
- Snowflake は、Snowflake クライアントを介して送信されるクエリテキスト (つまり、SQL ステートメント) のサイズをステートメントあたり 1 MB に制限しています。詳細については、「クエリテキストサイズの制限」を参照してください。

Teradata Vantage 接続

AWS Glue for Spark を使用して、AWS Glue 4.0 以降のバージョンの Teradata Vantage の既存のテーブルからの読み取りとテーブルへの書き込みを行うことができます。SQL クエリを使用して、Teradata から何を読み取るかを定義できます。AWS Glue 接続を介して AWS Secrets Manager で保存されているユーザー名およびパスワードの認証情報を使用して Teradata に接続できます。

Teradata の詳細については、Teradata ドキュメントを参照してください。

Teradata 接続の設定

AWS Glue から Teradata に接続するには、Teradata 認証情報を作成して AWS Secrets Manager シークレットに保存し、そのシークレットを Teradata AWS Glue 接続に関連付ける必要があります。Teradata インスタンスが Amazon VPC 内にある場合は、AWS Glue Teradata 接続にネットワークオプションを提供する必要もあります。

AWS Glue から Teradata に接続するには、いくつかの前提条件を満たす必要がある場合があります。

- Amazon VPC を通じて Teradata 環境にアクセスしている場合は、AWS Glue ジョブが Teradata 環境と通信できるように Amazon VPC を設定します。パブリックインターネット経由で Teradata 環境にアクセスすることは推奨されていません。

  Amazon VPC で、AWS Glue がジョブの実行中に使用する [VPC]、[サブネット]、および [セキュリティグループ] を特定または作成します。さらに、Teradata インスタンスとこの場所の間のネットワークトラフィックを許可するように Amazon VPC が設定されているようにする必要があります。ジョブでは、Teradata クライアントポートとの TCP 接続を確立する必要があります。Teradata ポートの詳細については、Teradata のドキュメントを参照してください。

ネットワークレイアウトに応じて、安全な VPC 接続を実現するには、Amazon VPC および他のネットワークサービスの変更が必要な場合があります。AWS 接続の詳細については、Teradata ドキュメントの「AWS 接続オプション」を参照してください。

AWS Glue Teradata 接続を設定するには:

1. Teradata 設定で、AWS Glue が接続するユーザーとパスワード (*teradataUser* および *teradataPassword*) を識別または作成します。詳細については、Teradata ドキュメントの「Vantage セキュリティの概要」を参照してください。
2. AWS Secrets Manager で、Teradata 認証情報を使用してシークレットを作成します。Secrets Manager でシークレットを作成するには、AWS Secrets Manager ドキュメントの「AWS Secrets Manager シークレットを作成する」にあるチュートリアルに従ってください。シークレットを作成したら、次のステップのためにシークレット名 *secretName* を保存しておきます。

   - [key/value ペア] を選択する際に、*teradataUsername* という値を持つキー `user` のペアを作成します。
   - [key/value ペア] を選択する際に、*teradataPassword* という値を持つキー `password` のペアを作成します。
3. AWS Glue コンソールで、「the section called “AWS Glue 接続の追加”」にあるステップに従って接続を作成します。接続を作成したら、次のステップのために接続名 *connectionName* を保存しておきます。

   - [接続タイプ] を選択する際に、[Teradata] を選択します。
   - [JDBC URL] を入力する場合は、インスタンスの URL を入力します。JDBC URL に特定のカンマ区切りの接続パラメータをハードコーディングすることもできます。URL は次の形式に準拠する必要があります:
     `jdbc:teradata://`*teradataHostname*`/`*ParameterName*`=`*ParameterValue*`,`*ParameterName*

     サポートされる URL パラメータには次が含まれます。

     - `DATABASE` – デフォルトでアクセスするホスト上のデータベースの名前。
     - `DBS_PORT` – データベースポート。非標準ポートで実行する場合に使用されます。
   - [認証情報タイプ] を選択する場合は、[AWS Secrets Manager] を選択し、[AWS シークレット] を *secretName* に設定します。
4. 次の状況では、追加の設定が必要になる場合があります。

- Amazon VPC の AWS でホストされている Teradata インスタンスの場合
  - Teradata セキュリティ認証情報を定義する AWS Glue 接続に、Amazon VPC 接続に関する情報を提供する必要があります。接続を作成または更新する際に、[ネットワークオプション] で [VPC]、[サブネット]、および [セキュリティグループ] を設定します。

AWS Glue Teradata 接続を作成した後、接続メソッドを呼び出す前に次のステップを実行する必要があります。

- AWS Glue ジョブに関連付けられている IAM ロールに *secretName* を読み取るアクセス許可を付与します。
- AWS Glue ジョブ設定で、追加のネットワーク接続として *connectionName* を指定します。

Teradata からの読み取り

前提条件:

- 読み取り元とする Teradata テーブル。テーブル名 *tableName* が必要になります。
- 認証情報を提供するように設定された AWS Glue Teradata 接続。認証情報を設定するには、「Teradata に対する接続を設定するには」のステップを実行します。AWS Glue 接続、*connectionName* の名前が必要になります。

例:

```
teradata_read_table = glueContext.create_dynamic_frame.from_options(
    connection_type="teradata",
    connection_options={
        "connectionName": "connectionName",
        "dbtable": "tableName"
    }
)
```

SELECT SQL クエリを指定して、DynamicFrame に返される結果をフィルタリングすることもできます。query を設定する必要があります。

例:

```
teradata_read_query = glueContext.create_dynamic_frame.from_options(
```

```
    connection_type="teradata",
    connection_options={
        "connectionName": "connectionName",
        "query": "query"
    }
)
```

Teradata テーブルへの書き込み

前提条件: 書き込み先とする Teradata テーブルである *tableName*。接続メソッドを呼び出す前にテーブルを作成する必要があります。

例:

```
teradata_write = glueContext.write_dynamic_frame.from_options(
    connection_type="teradata",
    connection_options={
        "connectionName": "connectionName",
        "dbtable": "tableName"
    }
)
```

Teradata 接続オプションのリファレンス

- connectionName — 必須。読み込み/書き込みに使用されます。認証およびネットワークの情報を接続方法に提供するように設定された AWS Glue Teradata 接続の名前。
- dbtable — 書き込みの場合は必須。query が指定されていない限り、読み取りの場合は必須。読み込み/書き込みに使用されます。接続メソッドがインタラクションするテーブルの名前。
- query — 読み取りに使用。Teradata から読み取るときに何を取得するかを定義する SELECT SQL クエリ。

Teradata Vantage NOS 接続

Teradata NOS (Native Object Store) 接続は、Teradata Vantage の新しい接続です。これは、Teradata WRITE_NOS クエリを活用して既存のテーブルから読み取り、READ_NOS クエリを使用してテーブルに書き込みます。これらのクエリはステージングディレクトリとして Amazon S3 を使用するため、Teradata NOS コネクタは、既存の Teradata コネクタ (JDBC ベース) よりも高速で、特に大量のデータを処理できます。

Spark 向けの AWS Glue の Teradata NOS 接続を使用して、AWS Glue 5.0 以降のバージョンの Teradata Vantage の既存テーブルからの読み取りやテーブルへの書き込みを行うことができます。SQL クエリを使用して、Teradata から何を読み取るかを定義できます。AWS Glue 接続を介して AWS Secrets Manager に保存されているユーザー名およびパスワードの認証情報を使用して Teradata に接続できます。

Teradata の詳細については、Teradata のドキュメントを参照してください。

トピック

- Teradata NOS 接続の作成
- Teradata テーブルからの読み取り
- Teradata テーブルへの書き込み
- Teradata 接続オプションのリファレンス
- AWS Glue ビジュアル ETL UI のオプションを指定する

Teradata NOS 接続の作成

AWS Glue から Teradata NOS に接続するには、Teradata 認証情報を作成して AWS Secrets Manager シークレットに保存し、そのシークレットを AWS Glue Teradata NOS 接続に関連付ける必要があります。Teradata インスタンスが Amazon VPC 内にある場合は、AWS Glue Teradata NOS 接続にネットワークオプションを提供する必要もあります。

前提条件:

- Amazon VPC を通じて Teradata 環境にアクセスしている場合は、AWS Glue ジョブが Teradata 環境と通信できるように Amazon VPC を設定します。パブリックインターネット経由で Teradata 環境にアクセスすることは推奨されていません。
- Amazon VPC で、AWS Glue がジョブの実行中に使用する [VPC]、[サブネット]、および [セキュリティグループ] を特定または作成します。さらに、Teradata インスタンスとこの場所の間のネットワークトラフィックを許可するように Amazon VPC が設定されているようにする必要があります。ジョブでは、Teradata クライアントポートとの TCP 接続を確立する必要があります。Teradata ポートの詳細については、「Teradata Vantage のセキュリティグループ」を参照してください。
- ネットワークレイアウトに応じて、安全な VPC 接続を実現するには、Amazon VPC および他のネットワークサービスの変更が必要な場合があります。AWS 接続の詳細については、Teradata ドキュメントの「AWS 接続オプション」を参照してください。

AWS Glue Teradata NOS 接続を設定するには:

1. Teradata 設定で、AWS Glue が接続する *teradataUsername* と *teradataPassword* を識別または作成します。詳細については、Teradata ドキュメントの「Vantage セキュリティの概要を 参照してください」を参照してください。
2. AWS Secrets Manager で、Teradata 認証情報を使用してシークレットを作成します。AWS Secrets Manager でシークレットを作成するには、AWS Secrets Manager ドキュメントの「AWS Secrets Manager シークレットを作成する」にあるチュートリアルに従ってください。シークレットを作成したら、次のステップのためにシークレット名 *secretName* を保存しておきます。
   - [Key/value ペア] を選択する際に、キー USERNAME と値 *teradataUsername* のペアを作成します。
   - [Key/value ペア] を選択する際に、キー PASSWORD と値 *teradataPassword* のペアを作成します。
3. AWS Glue コンソールで、「AWS Glue 接続の追加」にあるステップに従って接続を作成します。接続を作成したら、次のステップのために接続名 *connectionName* を保存しておきます。
   - [接続タイプ] を選択する際に、[Teradata Vantage NOS] を選択します。
   - [JDBC URL] を入力する場合は、インスタンスの URL を入力します。JDBC URL に特定のカンマ区切りの接続パラメータをハードコーディングすることもできます。URL は次の形式に準拠する必要があります: `jdbc:teradata://teradataHostname/ParameterName=ParameterValue,ParameterName=ParameterValue` 。
   - サポートされる URL パラメータには次が含まれます。
     - `DATABASE` – デフォルトでアクセスするホスト上のデータベースの名前。
     - `DBS_PORT` – データベースポート。非標準ポートで実行する場合に使用されます。
   - [認証情報タイプ] を選択する際は、[AWS Secrets Manager] を選択し、[AWS シークレット] を *secretName* に設定します。
4. 次の状況では、追加の設定が必要になる場合があります。
   - Amazon VPC 内の AWS でホストされている Teradata インスタンスの場合、Teradata セキュリティ認証情報を定義する AWS Glue 接続に、Amazon VPC 接続に関する情報を提供する必要があります。接続を作成または更新する際に、[ネットワークオプション] で [VPC]、[サブネット]、および [セキュリティグループ] を設定します。

AWS Glue Teradata Vantage NOS 接続を作成したら、接続メソッドを呼び出す前に次のステップを実行する必要があります。

1. AWS Glue ジョブに関連付けられている IAM ロールに *secretName* を読み取るアクセス許可を付与します。
2. AWS Glue ジョブ設定では、[接続] で [Additional network connection] (追加のネットワーク接続) として *connectionName* を指定します。

Teradata テーブルからの読み取り

前提条件:

- 読み取り元とする Teradata テーブル。テーブル名 *tableName* が必要になります。
- Teradata 環境には、`staging_fs_url` オプションで指定された Amazon S3 パスである *stagingFsUrl* への書き込みアクセス権があります。
- AWS Glue ジョブに関連付けられた IAM ロールには、`staging_fs_url` オプションで指定された Amazon S3 ロケーションへの書き込みアクセス権があります。
- 認証情報を提供するように設定された AWS Glue Teradata NOS 接続。認証情報を設定するステップ AWS Glue Teradata NOS 接続を設定するには: を完了します。AWS Glue 接続の名前、*connectionName* が必要になります。

例:

```
teradata_read_table = glueContext.create_dynamic_frame.from_options(
    connection_type= "teradatanos",
    connection_options={
        "connectionName": "connectionName",
        "dbtable": "tableName",
        "staging_fs_url": "stagingFsUrl"
    }
)
```

SELECT SQL クエリを指定して、DynamicFrame に返される結果をフィルタリングすることもできます。クエリを設定する必要があります。dbTable とクエリの両方を設定すると、コネクタはデータの読み取りに失敗します。例 :

```
teradata_read_query = glueContext.create_dynamic_frame.from_options(
    connection_type="teradatanos",
```

```
    connection_options={
        "connectionName": "connectionName",
        "query": "query",
        "staging_fs_url": "stagingFsUrl"
    }
)
```

さらに、Spark DataFrame API を使用して Teradata テーブルから読み取ることができます。例：

```
options = {
    "url": "JDBC_URL",
    "dbtable": "tableName",
    "user": "teradataUsername", # or use "username" as key here
    "password": "teradataPassword",
    "staging_fs_url": "stagingFsUrl"
}
teradata_read_table = spark.read.format("teradatanos").option(**options).load()
```

Teradata テーブルへの書き込み

前提条件

- 書き込み先とする Teradata テーブル: *tableName*。
- Teradata 環境には、staging_fs_url オプションで指定された Amazon S3 の場所である *stagingFsUrl* への読み取りアクセス権があります。
- AWS Glue ジョブに関連付けられた IAM ロールには、staging_fs_url オプションで指定された Amazon S3 ロケーションへの書き込みアクセス権があります。
- 認証情報を提供するように設定された AWS Glue Teradata 接続。認証情報を設定するには、「AWS Glue Teradata NOS 接続を設定するには:」の手順を実行します。AWS Glue 接続の名前、*connectionName* が必要になります。

  例：

```
teradata_write = glueContext.write_dynamic_frame.from_options(
    frame=dynamicFrame,
    connection_type= "teradatanos",
    connection_options={
        "connectionName": "connectionName",
        "dbtable": "tableName",
        "staging_fs_url": "stagingFsUrl"
```

```
    }
)
```

Teradata 接続オプションのリファレンス

接続とオペレーションのオプション：

- connectionName — 必須。読み込み/書き込みに使用されます。認証およびネットワークの情報をお使いの接続方法に提供するように設定された AWS Glue Teradata 接続の名前。
- staging_fs_url — 必須。読み込み/書き込みに使用されます。Amazon S3 の書き込み可能な場所。Teradata から読み取るときにはアンロードされたデータに使用され、Teradata に書き込むときには Parquet データが Redshift にロードされます。S3 バケットは、AWS Glue ジョブのリージョンと同じリージョンにある必要があります。
- dbtable — 書き込みの場合は必須。query が指定されていない限り、読み取りの場合は必須。読み込み/書き込みに使用されます。接続メソッドがインタラクションするテーブルの名前。
- query — 読み取りに使用。Teradata から読み取るときに何を取得するかを定義する SELECT SQL クエリ。dbtable オプションが指定されている場合、パスすることはできません。
- clean_staging_s3_dir — オプション。読み込み/書き込みに使用されます。true の場合は、読み取りまたは書き込み後にステージング Amazon S3 オブジェクトをクリーンアップします。デフォルト値は True です。
- pre_actions — オプション。書き込みに使用。Spark と Teradata Vantage の間でデータが転送される前に実行される SQL コマンドのセミコロン区切りリスト。
- post_actions — オプション。書き込みに使用。Spark と Teradata Vantage の間でデータが転送された後に実行される SQL コマンドのセミコロン区切りリスト。
- truncate — オプション。書き込みに使用。true の場合、コネクタは上書きモードで書き込むときにテーブルを切り捨てます。false の場合、コネクタは上書きモードで書き込むときにテーブルをドロップします。デフォルト値は false です。
- create_table_script — オプション。書き込みに使用。Teradata Vantage に書き込むときにテーブルを作成する SQL ステートメント。カスタムメタデータ (CREATE MULTISET や SET テーブル、プライマリインデックスの変更など) を使用してテーブルを作成する場合に便利です。テーブル作成スクリプトで使用されるテーブル名は、dbtable オプションで指定されたテーブル名と一致する必要があることに注意してください。
- partition_size_in_mb — オプション。読み込みに使用されます。ステージング Amazon S3 オブジェクトの読み取り中の Spark パーティションの最大サイズ (メガバイト)。デフォルト値は 128 です。

Teradata ノードを作成する際に、高度なオプションを指定できます。これらのオプションは Spark AWS Glue スクリプトのプログラミング時に使用できるオプションと同じです。

「the section called “Teradata Vantage 接続”」を参照してください。

認可オプション

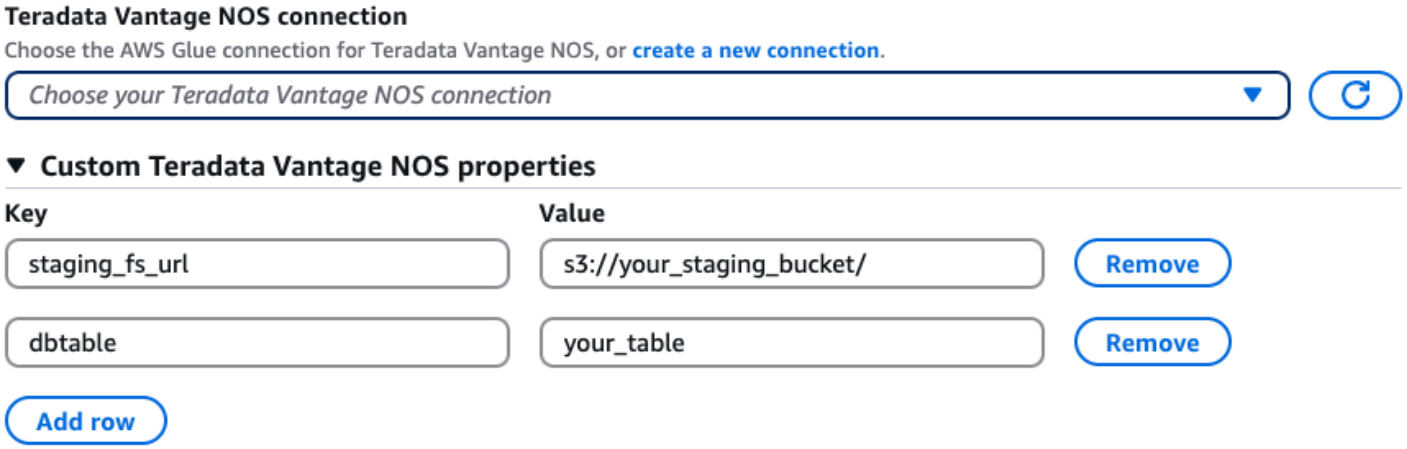
以下は、コネクタがステージング Amazon S3 バケットにアクセスするために使用する AWS アカウント認証情報を提供するために使用されるオプションです。(1) 認証オプションを一切提供せず、AWS Glue 実行ロールから生成された一時的な認証情報を使用するか、(2) 作成した認証オブジェクト `auth_object` を提供するか、(3) 長期認証情報を使用する場合は `aws_access_key_id and aws_secret_access_key` を提供するか、または一時的な認証情報を使用する場合は `aws_access_key`、`aws_secret_access_key`、および `aws_session_token` を提供するかを選択できます。

- `auth_object` - オプション。ステージング Amazon S3 バケットへのアクセスに使用されます。Teradata インスタンスで作成された認証オブジェクト文字列。指定すると、コネクタはこの認証オブジェクトを使用してステージング Amazon S3 バケットにアクセスします。指定がなく、`aws_secret_access_key` と AWS Glue も指定されていない場合、一時的な認証情報が `aws_access_key_id` 実行ロールから取得され、コネクタによって使用されます。この認証オブジェクトに関連付けられた AWS アカウントは、AWS Glue ジョブおよびステージング Amazon S3 バケットと同じリージョンにあるか、クロスアカウント信頼が設定されている必要があります。
- `aws_access_key_id` - オプション。ステージング Amazon S3 バケットへのアクセスに使用されます。AWS アカウントセキュリティ認証情報の一部。`auth_object` の指定がなく、`aws_access_key_id` が `aws_secret_access_key` に提供されている場合、コネクタはそれらを使用してステージング Amazon S3 バケットにアクセスします。このアクセスキーに関連付けられた AWS アカウントは、AWS Glue ジョブおよびステージング Amazon S3 バケットと同じリージョンにあるか、クロスアカウント信頼が設定されている必要があります。
- `aws_secret_access_key` - オプション。ステージング Amazon S3 バケットへのアクセスに使用されます。AWS アカウントセキュリティ認証情報の一部。`auth_object` の指定がなく、`aws_secret_access_key` が `aws_access_key_id` に設定されている場合、コネクタはそれらを使用してステージング Amazon S3 バケットにアクセスします。このシークレットキーに関連付けられた AWS アカウントは、AWS Glue ジョブとステージング Amazon S3 バケットと同じリージョンにあるか、クロスアカウント信頼が設定されている必要があります。

- aws_session_token - オプション。ステージング Amazon S3 バケットへのアクセスに使用されます。一時的な AWS アカウントセキュリティ認証情報の一部。aws_access_key_id と aws_secret_access_key を指定する必要があります。

AWS Glue ビジュアル ETL UI のオプションを指定する

ビジュアル ETL ジョブ UI で上記のすべてのオプションを指定できます。connectionName オプションは、Teradata Vantage NOS 接続のドロップダウンリストから選択します。他のすべてのオプションについては、カスタム Teradata Vantage NOS プロパティをキーと値のペアとして指定する必要があります。



Vertica 接続

AWS Glue for Spark を使用して、AWS Glue 4.0 以降のバージョンの Vertica のテーブルからの読み取りとテーブルへの書き込みを行うことができます。SQL クエリを使用して、Vertica から何を読み取るかを定義できます。AWS Glue 接続を介して AWS Secrets Manager で保存されているユーザー名およびパスワードの認証情報を使用して Vertica に接続します。

Vertica の詳細については、Vertica ドキュメントを参照してください。

Vertica 接続の設定

AWS Glue から Vertica に接続するには、Vertica 認証情報を作成して AWS Secrets Manager シークレットに保存し、そのシークレットを Vertica AWS Glue 接続に関連付ける必要があります。Vertica インスタンスが Amazon VPC 内にある場合は、AWS Glue Vertica 接続にネットワークオプションを提供する必要もあります。データベースの読み取りおよび書き込み時に一時ストレージとして使用する Amazon S3 バケットまたはフォルダが必要です。

AWS Glue から Vertica に接続するには、いくつかの前提条件を満たす必要があります。

- データベースの読み取りおよび書き込み時に一時ストレージとして使用する Amazon S3 バケットまたはフォルダ。*tempS3Path* によって参照されます。

> **Note**
>
> AWS Glue ジョブデータプレビューで Vertica を使用する場合、一時ファイルは *tempS3Path* から自動的に削除されない場合があります。一時ファイルを確実に削除するには、[データプレビュー] ペインで [セッションの終了] を選択して、データプレビューセッションを直接終了します。
> データプレビューセッションが直接終了することを保証できない場合は、Amazon S3 ライフサイクル構成を設定して古いデータを削除することを検討してください。ジョブの最大実行時間にマージンを加えた値に基づいて、49 時間を超える時間が経過しているデータを削除することをお勧めします。Amazon S3 ライフサイクルの設定の詳細については、Amazon S3 ドキュメントの「ストレージのライフサイクルの管理」を参照してください。

- AWS Glue ジョブロールに関連付けることができる、Amazon S3 パスに対する適切な許可を持つ IAM ポリシー。
- Vertica インスタンスが Amazon VPC 内にある場合は、トラフィックがパブリックインターネットを経由することなく、AWS Glue ジョブが Vertica インスタンスと通信できるように Amazon VPC を設定します。

  Amazon VPC で、AWS Glue がジョブの実行中に使用する [VPC]、[サブネット]、および [セキュリティグループ] を特定または作成します。さらに、Vertica インスタンスとこの場所の間のネットワークトラフィックを許可するように Amazon VPC が設定されているようにする必要があります。ジョブは、Vertica クライアントポート (デフォルトは 5433) との TCP 接続を確立する必要があります。ネットワークレイアウトに基づいて、セキュリティグループルール、ネットワーク ACL、NAT ゲートウェイ、およびピアリング接続の変更が必要になる場合があります。

その後、Vertica で使用できるように AWS Glue を設定する作業に進むことができます。

Vertica に対する接続を設定するには:

1. AWS Secrets Manager で、Vertica 認証情報、*verticaUsername* および *verticaPassword* を使用してシークレットを作成します。Secrets Manager でシークレットを作成するには、AWS Secrets Manager ドキュメントの「AWS Secrets Manager シークレットを作成する」

にあるチュートリアルに従ってください。シークレットを作成したら、次のステップのためにシークレット名 *secretName* を保存しておきます。

- [key/value ペア] を選択する際に、*verticaUsername* という値を持つキー user のペアを作成します。
- [key/value ペア] を選択する際に、*verticaPassword* という値を持つキー password のペアを作成します。

2. AWS Glue コンソールで、「the section called “AWS Glue 接続の追加”」にあるステップに従って接続を作成します。接続を作成したら、次のステップのために接続名 *connectionName* を保存しておきます。

   - [接続タイプ] を選択する際に、[Vertica] を選択します。
   - [Vertica ホスト] を選択する場合は、Vertica インストールのホスト名を入力します。
   - [Vertica ポート] を選択すると、Vertica インストールを使用できるポートが選択されます。
   - [AWS Secret] をクリックして、*secretName* を入力します。

3. 次の状況では、追加の設定が必要になる場合があります。

   - Amazon VPC の AWS でホストされている Vertica インスタンスの場合
     - Vertica セキュリティ認証情報を定義する AWS Glue 接続に、Amazon VPC 接続に関する情報を提供します。接続を作成または更新する際に、[ネットワークオプション] で [VPC]、[サブネット]、および [セキュリティグループ] を設定します。

AWS Glue Vertica 接続を作成した後、接続メソッドを呼び出す前に次のステップを実行する必要があります。

- AWS Glue ジョブの許可に関連付けられた IAM ロールを *tempS3Path* に付与します。
- AWS Glue ジョブに関連付けられている IAM ロールに *secretName* を読み取るアクセス許可を付与します。
- AWS Glue ジョブ設定で、追加のネットワーク接続として *connectionName* を指定します。

Vertica からの読み取り

前提条件:

- 読み取り元とする Vertica テーブル。Vertica データベース名である *dbName* とテーブル名である *tableName* が必要になります。

- 認証情報を提供するように設定された AWS Glue Vertica 接続。認証情報を設定するには、前の手順「Vertica に対する接続を設定するには」のステップを実行します。AWS Glue 接続、*connectionName* の名前が必要になります。
- 前述した一時ストレージに使用する Amazon S3 バケットまたはフォルダ。*tempS3Path* という名前が必要になります。s3a プロトコルを使用してこの場所に接続する必要があります。

例:

```
dynamicFrame = glueContext.create_dynamic_frame.from_options(
    connection_type="vertica",
    connection_options={
        "connectionName": "connectionName",
        "staging_fs_url": "s3a://tempS3Path",
        "db": "dbName",
        "table": "tableName",
    }
)
```

SELECT SQL クエリを入力して、DynamicFrame に返された結果をフィルタリングしたり、複数のテーブルからデータセットにアクセスしたりすることもできます。

例:

```
dynamicFrame = glueContext.create_dynamic_frame.from_options(
    connection_type="vertica",
    connection_options={
        "connectionName": "connectionName",
        "staging_fs_url": "s3a://tempS3Path",
        "db": "dbName",
        "query": "select * FROM tableName",
    },
)
```

Vertica テーブルへの書き込み

この例では、既存の DynamicFrame である *dynamicFrame* から Vertica に情報を書き込みます。テーブルに既に情報がある場合、AWS Glue は DynamicFrame からのデータを付加します。

前提条件:

- 書き込み先とする現在または希望のテーブル名である *tableName*。対応する Vertica データベース名である *dbName* も必要になります。
- 認証情報を提供するように設定された AWS Glue Vertica 接続。認証情報を設定するには、前の手順「Vertica に対する接続を設定するには」のステップを実行します。AWS Glue 接続、*connectionName* の名前が必要になります。
- 前述した一時ストレージに使用する Amazon S3 バケットまたはフォルダ。*tempS3Path* という名前が必要になります。s3a プロトコルを使用してこの場所に接続する必要があります。

例:

```
glueContext.write_dynamic_frame.from_options(
    frame=dynamicFrame,
    connection_type="vertica",
    connection_options={
        "connectionName": "connectionName",
        "staging_fs_url": "s3a://tempS3Path",
        "db": "dbName",
        "table": "tableName",
    }
)
```

Vertica 接続オプションのリファレンス

- connectionName — 必須。読み込み/書き込みに使用されます。認証およびネットワークの情報を接続方法に提供するように設定された AWS Glue Vertica 接続の名前。
- db — 必須。読み込み/書き込みに使用されます。接続方法がインタラクションする Vertica 内のデータベースの名前。
- dbSchema — テーブルを識別する必要がある場合は必須です。読み込み/書き込みに使用されます。デフォルト: public。接続メソッドがインタラクションするスキーマの名前。
- table — 書き込みの場合は必須。query が指定されていない限り、読み取りの場合は必須。読み込み/書き込みに使用されます。接続メソッドがインタラクションするテーブルの名前。
- query — 読み取りに使用。Teradata から読み取るときに何を取得するかを定義する SELECT SQL クエリ。
- staging_fs_url — 必須。読み込み/書き込みに使用されます。有効な値: s3a URL。一時ストレージに使用する Amazon S3 バケットまたはフォルダの URL。

AWS Glue 5.0 for Spark の ETL の DataFrame オプション

DataFrame は、名前の付いた列にまとめられた、テーブルに似たデータセットで、機能スタイル (マップ/リデュース/フィルター/その他) 操作と SQL 操作 (選択、プロジェクト、集計) をサポートしています。

Glue でサポートされているデータソースの DataFrame を作成するには、以下が必要です。

- データソースコネクタ `ClassName`
- データソース接続 `Options`

同様に、Glue でサポートされているデータシンクに DataFrame を書き込むときも、同じものが必要です。

- データシンクコネクタ `ClassName`
- データシンク接続 `Options`

ジョブのブックマークなどの AWS Glue 機能や、`connectionName` などの DynamicFrame オプションは DataFrame ではサポートされていないことに注意してください。DataFrame とサポートされているオペレーションの詳細については、DataFrame の Spark のドキュメントを参照してください。

コネクタ ClassName の指定

データソース/シンクの `ClassName` を指定するには、`.format` オプションを使用して、データソース/シンクを定義するための対応するコネクタ `ClassName` を指定します。

JDBC コネクタ

JDBC コネクタの場合は、`.format` オプションの値として `jdbc` を指定し、`driver` オプションで JDBC ドライバー `ClassName` を指定します。

```
df = spark.read.format("jdbc").option("driver", "<DATA SOURCE JDBC DRIVER
 CLASSNAME>")...

df.write.format("jdbc").option("driver", "<DATA SINK JDBC DRIVER CLASSNAME>")...
```

次の表は、AWS Glue for DataFrames でサポートされているデータソースの JDBC ドライバー `ClassName` の一覧です。

データソース	ドライバーの ClassName
PostgreSQL	org.postgresql.Driver
Oracle	oracle.jdbc.driver.OracleDriver
SQLServer	com.microsoft.sqlserver.jdbc.SQLServerDriver
MySQL	com.mysql.jdbc.Driver
SAPHana	com.sap.db.jdbc.Driver
Teradata	com.teradata.jdbc.TeraDriver

スパークコネクタ

Spark コネクタの場合は、コネクタの `ClassName` を `.format` オプションの値として指定します。

```
df = spark.read.format("<DATA SOURCE CONNECTOR CLASSNAME>")...

df.write.format("<DATA SINK CONNECTOR CLASSNAME>")...
```

次の表は、AWS Glue for DataFrames でサポートされているデータソースの Spark コネクタ `ClassName` の一覧です。

データソース	ClassName
MongoDB/DocumentDB	glue.spark.mongodb
Redshift	io.github.spark_redshift_community.spark.redshift
AzureCosmos	cosmos.oltp
AzureSQL	com.microsoft.sqlserver.jdbc.spark
BigQuery	com.google.cloud.spark.bigquery
OpenSearch	org.opensearch.spark.sql

Snowflake	net.snowflake.spark.snowflake
Vertica	com.vertica.spark.datasource.VerticaSource

接続オプションの指定

データソース/シンクへの接続の Options を指定するには、.option(<KEY>, <VALUE>) を使用して個々のオプションを指定するか、.options(<MAP>) を使用してキーと値のマップとして複数のオプションを指定します。

各データソース/シンクは、接続 Options の独自のセットをサポートします。利用可能な Options の詳細については、次の表に記載されている特定のデータソース/シンクの Spark コネクタの公開ドキュメントを参照してください。

- JDBC
- MongoDB/DocumentDB
- Redshift
- AzureCosmos
- AzureSQL
- BigQuery
- OpenSearch
- Snowflake
- Vertica

例

次の例では、PostgreSQL から読み取り、SnowFlake に書き込みます。

Python

例:

```
from awsglue.context import GlueContext
from pyspark.sql import SparkSession

spark = SparkSession.builder.getOrCreate()

dataSourceClassName = "jdbc"
```

```
dataSourceOptions = {
  "driver": "org.postgresql.Driver",
  "url": "<url>",
  "user": "<user>",
  "password": "<password>",
  "dbtable": "<dbtable>",
}

dataframe = spark.read.format(className).options(**options).load()

dataSinkClassName = "net.snowflake.spark.snowflake"
dataSinkOptions = {
  "sfUrl": "<url>",
  "sfUsername": "<username>",
  "sfPassword": "<password>",
  "sfDatabase" -> "<database>",
  "sfSchema" -> "<schema>",
  "sfWarehouse" -> "<warehouse>"
}

dataframe.write.format(dataSinkClassName).options(**dataSinkOptions).save()
```

Scala

例:

```
import org.apache.spark.sql.SparkSession

val spark = SparkSession.builder().getOrCreate()

val dataSourceClassName = "jdbc"
val dataSourceOptions = Map(
  "driver" -> "org.postgresql.Driver",
  "url" -> "<url>",
  "user" -> "<user>",
  "password" -> "<password>",
  "dbtable" -> "<dbtable>"
)

val dataframe =
 spark.read.format(dataSourceClassName).options(dataSourceOptions).load()

val dataSinkClassName = "net.snowflake.spark.snowflake"
val dataSinkOptions = Map(
```

```
  "sfUrl" -> "<url>",
  "sfUsername" -> "<username>",
  "sfPassword" -> "<password>",
  "sfDatabase" -> "<database>",
  "sfSchema" -> "<schema>",
  "sfWarehouse" -> "<warehouse>"
)

dataframe.write.format(dataSinkClassName).options(dataSinkOptions).save()
```

カスタムと AWS Marketplace での、connectionType の値

これには以下が含まれます。

- `"connectionType": "marketplace.athena"`: Amazon Athena データストアへの接続を指定します。この接続では、AWS Marketplace が提供するコネクタを使用します。
- `"connectionType": "marketplace.spark"`: Apache Spark データストアへの接続を指定します。この接続では、AWS Marketplace が提供するコネクタを使用します。
- `"connectionType": "marketplace.jdbc"`: JDBC データストアへの接続を指定します。この接続では、AWS Marketplace が提供するコネクタを使用します。
- `"connectionType": "custom.athena"`: Amazon Athena データストアへの接続を指定します。接続には、AWS Glue Studio にアップロードしたカスタムコネクタを使用します。
- `"connectionType": "custom.spark"`: Apache Spark データストアへの接続を指定します。接続には、AWS Glue Studio にアップロードしたカスタムコネクタを使用します。
- `"connectionType": "custom.jdbc"`: JDBC データストアへの接続を指定します。接続には、AWS Glue Studio にアップロードしたカスタムコネクタを使用します。

custom.jdbc または marketplace.jdbc 型での接続オプション

- `className` – (必須) ドライバクラス名を示す文字列。
- `connectionName` – (必須) コネクタに関連付けられている接続の名前を示す文字列。
- `url` – (必須) データソースへの接続を構築するために使用される、プレースホルダを含む JDBC URL (`${}`) を示す文字列。プレースホルダー `${secretKey}` は、AWS Secrets Manager 内にある同じ名前のシークレットにより置き換えられます。URL の構築の詳細については、データストアのドキュメントを参照してください。
- `secretId` または `user/password` – (必須) URL の認証情報を取得するために使用される文字列。

- dbTableまたはquery – (必須) データを取得するテーブルまたは SQL クエリを示す文字列。dbTable または query を指定できます。両方を指定することはできません。
- partitionColumn – (オプション) パーティション化に使用される整数カラムの名前を示す文字列。このオプションは、lowerBound、upperBound、および numPartitions に含まれている場合にのみ機能します。このオプションの機能は、Spark SQL JDBC リーダーのものと同様です。詳細については、Apache Spark SQL, DataFrames and Datasets Guide の「JDBC To Other Databases」を参照してください。

  lowerBound および upperBound 値は、パーティションのストライドを決定するために使用されます (テーブル内の行のフィルタリングには使用しません)。返されるテーブル内のすべての行は、パーティション化されています。

  > **Note**
  >
  > テーブル名の代わりにクエリを使用する場合は、指定されたパーティショニング条件でクエリが動作することを確認する必要があります。例:
  >
  > - "SELECT col1 FROM table1" の形式のクエリでパーティション列を使用する場合、末尾に WHERE 句を追加してそのクエリをテストします。
  > - クエリ形式が SELECT col1 FROM table1 WHERE col2=val" の場合は、WHERE 句を AND とパーティション列を使用する式で拡張することで、そのクエリをテストします。

- lowerBound – パーティションストライドを決定するために使用される partitionColumn の最小値を示す整数 (オプション)。
- upperBound – パーティションストライドを決定するために使用される partitionColumn の最大値を示す整数 (オプション)。
- numPartitions – パーティション数を示す整数 (オプション)。この値は、(範囲に含まれる) lowerBound と (範囲に含まれない) upperBound とともに使用され、partitionColumn の分割で使用するために生成された WHERE 句の式のための、パーティションストライドを形成します。

  > **Important**
  >
  > パーティションが多すぎると、外部データベースシステムで問題が発生する可能性があるため、パーティションの数には注意を払ってください。

- `filterPredicate` – ソースからのデータをフィルタリングする、追加の条件句を示す文字列 (オプション)。例:

```
BillingCity='Mountain View'
```

  table 名の代わりに query を使用した場合は、指定された `filterPredicate` でクエリが動作することを確認します。例:

  - クエリの形式が `"SELECT col1 FROM table1"` の場合は、フィルタ述語を使用するクエリの末尾に `WHERE` 句を追加して、そのクエリをテストします。
  - クエリ形式が `"SELECT col1 FROM table1 WHERE col2=val"` の場合は、`WHERE` 句を `AND` およびフィルター述語を使用する式で拡張して、そのクエリをテストします。

- `dataTypeMapping` – (ディクショナリ、オプション) JDBC データ型 から Glue データ型 に対するマッピングを構築する、カスタムのデータ型マッピング。例えば、オプション `"dataTypeMapping":{"FLOAT":"STRING"}` では、ドライバーの `ResultSet.getString()` メソッドを呼び出すことで、JDBC タイプ `FLOAT` のデータフィールドが Java `String` 型にマッピングされ、それを使用して AWS Glue レコードが構築されます。`ResultSet` オブジェクトは各ドライバによって実装されるため、その動作は使用するドライバにより決定されます。ドライバによる変換の実行方法については、JDBC ドライバのドキュメントを参照してください。

- AWS Glue で現在サポートされているデータ型は次のとおりです。
  - DATE
  - STRING
  - TIMESTAMP
  - INT
  - FLOAT
  - LONG
  - BIGDECIMAL
  - BYTE
  - SHORT
  - DOUBLE

  JDBC データ型としては、Java8 java.sql.types がサポートされています。

  デフォルトの (JDBC から AWS Glue への) データ型マッピングは以下のとおりです。

  DATE -> DATE

- VARCHAR -> STRING
- CHAR -> STRING
- LONGNVARCHAR -> STRING
- TIMESTAMP -> TIMESTAMP
- INTEGER -> INT
- FLOAT -> FLOAT
- REAL -> FLOAT
- BIT -> BOOLEAN
- BOOLEAN -> BOOLEAN
- BIGINT -> LONG
- DECIMAL -> BIGDECIMAL
- NUMERIC -> BIGDECIMAL
- TINYINT -> SHORT
- SMALLINT -> SHORT
- DOUBLE -> DOUBLE

カスタムのデータタイプマッピングで dataTypeMapping オプションを使用すると、デフォルトのデータ型マッピングをオーバーライドできます。この影響を受けるのは、dataTypeMapping オプションでリストされた JDBC データ型のみです。他のすべての JDBC データ型に対しては、デフォルトのマッピングが使用されます。必要に応じて、別の JDBC データ型のマッピングを追加することも可能です。デフォルトまたはカスタムのマッピングのいずれにも JDBC データ型が含まれていない場合、データ型はデフォルトで AWS GlueSTRING データ型に変換されます。

次の Python コード例は、AWS Marketplace のJDBC ドライバーを使用して、JDBC データベースからの読み取りを実行する方法を示しています。ここでは、データベースからの読み取りと、S3 ロケーションへの書き込みの方法を知ることができます。

```
    import sys
    from awsglue.transforms import *
    from awsglue.utils import getResolvedOptions
    from pyspark.context import SparkContext
    from awsglue.context import GlueContext
    from awsglue.job import Job

    ## @params: [JOB_NAME]
```

```
    args = getResolvedOptions(sys.argv, ['JOB_NAME'])

    sc = SparkContext()
    glueContext = GlueContext(sc)
    spark = glueContext.spark_session
    job = Job(glueContext)
    job.init(args['JOB_NAME'], args)
    ## @type: DataSource
    ## @args: [connection_type = "marketplace.jdbc", connection_options =
     {"dataTypeMapping":{"INTEGER":"STRING"},"upperBound":"200","query":"select id,
       name, department from department where id < 200","numPartitions":"4",
       "partitionColumn":"id","lowerBound":"0","connectionName":"test-connection-
jdbc"},
        transformation_ctx = "DataSource0"]
    ## @return: DataSource0
    ## @inputs: []
    DataSource0 = glueContext.create_dynamic_frame.from_options(connection_type =
      "marketplace.jdbc", connection_options = {"dataTypeMapping":{"INTEGER":"STRING"},
      "upperBound":"200","query":"select id, name, department from department where
       id < 200","numPartitions":"4","partitionColumn":"id","lowerBound":"0",
       "connectionName":"test-connection-jdbc"}, transformation_ctx = "DataSource0")
    ## @type: ApplyMapping
    ## @args: [mappings = [("department", "string", "department", "string"), ("name",
 "string",
     "name", "string"), ("id", "int", "id", "int")], transformation_ctx =
 "Transform0"]
    ## @return: Transform0
    ## @inputs: [frame = DataSource0]
    Transform0 = ApplyMapping.apply(frame = DataSource0, mappings = [("department",
 "string",
     "department", "string"), ("name", "string", "name", "string"), ("id", "int",
 "id", "int")],
      transformation_ctx = "Transform0")
    ## @type: DataSink
    ## @args: [connection_type = "s3", format = "json", connection_options = {"path":
     "s3://<S3 path>/", "partitionKeys": []}, transformation_ctx = "DataSink0"]
    ## @return: DataSink0
    ## @inputs: [frame = Transform0]
    DataSink0 = glueContext.write_dynamic_frame.from_options(frame = Transform0,
      connection_type = "s3", format = "json", connection_options = {"path":
      "s3://<S3 path>/", "partitionKeys": []}, transformation_ctx = "DataSink0")
    job.commit()
```

custom.athena または marketplace.athena タイプ用の接続オプション

- `className` – (必須) ドライバクラス名を示す文字列。Athena-CloudWatch コネクタを使用している場合、このパラメータ値はクラス名にプレフィックスされます (例: `"com.amazonaws.athena.connectors"`)。Athena-CloudWatch コネクタは、メタデータハンドラーとレコードハンドラーの 2 つのクラスで構成されています。共通のプレフィックスを指定することで、API がそのプレフィックスに基づいた適切なクラスをロードします。
- `tableName` – (必須) 読み込む CloudWatch ログストリームの名前を示す文字列。このコードスニペットでは、ビューに特別な名前 `all_log_streams` を使用しています。この場合、返された動的データフレームには、ロググループ内のすべてのログストリームからのデータが含まれます。
- `schemaName` – (必須) 読み取りのソースとなる CloudWatch ロググループの名前を示す文字列。例えば、`/aws-glue/jobs/output` と指定します。
- `connectionName` – (必須) コネクタに関連付けられている接続の名前を示す文字列。

このコネクタの追加オプションについては、GitHub の Amazon Athena CloudWatch Connector README ファイルを参照してください。

次の Python コード例は、AWS Marketplace コネクタを使用しながら、Athena データストアからの読み取りを実行する方法を示しています。ここでは、Athena からの読み取りと、S3 ロケーションへの書き込みを行う方法を知ることができます。

```
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job

## @params: [JOB_NAME]
args = getResolvedOptions(sys.argv, ['JOB_NAME'])

sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
job.init(args['JOB_NAME'], args)
## @type: DataSource
## @args: [connection_type = "marketplace.athena", connection_options =
 {"tableName":"all_log_streams","schemaName":"/aws-glue/jobs/output",
```

```
      "connectionName":"test-connection-athena"}, transformation_ctx = "DataSource0"]
   ## @return: DataSource0
   ## @inputs: []
   DataSource0 = glueContext.create_dynamic_frame.from_options(connection_type =
     "marketplace.athena", connection_options = {"tableName":"all_log_streams",,
     "schemaName":"/aws-glue/jobs/output","connectionName":
     "test-connection-athena"}, transformation_ctx = "DataSource0")
   ## @type: ApplyMapping
   ## @args: [mappings = [("department", "string", "department", "string"), ("name",
"string",
     "name", "string"), ("id", "int", "id", "int")], transformation_ctx =
"Transform0"]
   ## @return: Transform0
   ## @inputs: [frame = DataSource0]
   Transform0 = ApplyMapping.apply(frame = DataSource0, mappings = [("department",
"string",
     "department", "string"), ("name", "string", "name", "string"), ("id", "int",
"id", "int")],
      transformation_ctx = "Transform0")
   ## @type: DataSink
   ## @args: [connection_type = "s3", format = "json", connection_options = {"path":
    "s3://<S3 path>/", "partitionKeys": []}, transformation_ctx = "DataSink0"]
   ## @return: DataSink0
   ## @inputs: [frame = Transform0]
   DataSink0 = glueContext.write_dynamic_frame.from_options(frame = Transform0,
     connection_type = "s3", format = "json", connection_options = {"path":
     "s3://<S3 path>/", "partitionKeys": []}, transformation_ctx = "DataSink0")
   job.commit()
```

custom.spark または marketplace.spark タイプ用の接続オプション

- `className` – (必須) コネクタのクラス名を支援す文字列。
- `secretId` – (オプション) コネクタ接続の認証情報を取得するために使用される文字列。
- `connectionName` – (必須) コネクタに関連付けられている接続の名前を示す文字列。
- その他のオプションは、データストアによって異なります。例えば、Elasticsearch for Apache Hadoop ドキュメントの説明にあるように、OpenSearch の設定オプションは「`es`」でプレフィックスされます。Spark から Snowflake への接続では、Connecting to Snowflake ガイドの「Using the Spark Connector」で説明されているように、`sfUser` および `sfPassword` のオプションを使用します。

次の Python コード例に、marketplace.spark 接続を使用して、OpenSearch のデータストアからの読み取りを実行する方法を示します。

```
   import sys
   from awsglue.transforms import *
   from awsglue.utils import getResolvedOptions
   from pyspark.context import SparkContext
   from awsglue.context import GlueContext
   from awsglue.job import Job

   ## @params: [JOB_NAME]
   args = getResolvedOptions(sys.argv, ['JOB_NAME'])

   sc = SparkContext()
   glueContext = GlueContext(sc)
   spark = glueContext.spark_session
   job = Job(glueContext)
   job.init(args['JOB_NAME'], args)
   ## @type: DataSource
   ## @args: [connection_type = "marketplace.spark", connection_options =
{"path":"test",
     "es.nodes.wan.only":"true","es.nodes":"https://<AWS endpoint>",
     "connectionName":"test-spark-es","es.port":"443"}, transformation_ctx =
"DataSource0"]
   ## @return: DataSource0
   ## @inputs: []
   DataSource0 = glueContext.create_dynamic_frame.from_options(connection_type =
     "marketplace.spark", connection_options = {"path":"test","es.nodes.wan.only":
     "true","es.nodes":"https://<AWS endpoint>","connectionName":
     "test-spark-es","es.port":"443"}, transformation_ctx = "DataSource0")
   ## @type: DataSink
   ## @args: [connection_type = "s3", format = "json", connection_options = {"path":
        "s3://<S3 path>/", "partitionKeys": []}, transformation_ctx = "DataSink0"]
   ## @return: DataSink0
   ## @inputs: [frame = DataSource0]
   DataSink0 = glueContext.write_dynamic_frame.from_options(frame = DataSource0,
      connection_type = "s3", format = "json", connection_options = {"path":
      "s3://<S3 path>/", "partitionKeys": []}, transformation_ctx = "DataSink0")
   job.commit()
```

汎用オプション

このセクションのオプションは connection_options として提供されていますが、特定のコネクタには当てはまりません。

以下のパラメータは、ブックマークを設定する際に一般的に使用されます。Amazon S3 または JDBC ワークフローに適用される場合があります。詳細については、「the section called “ジョブのブックマークを使用する”」を参照してください。

- jobBookmarkKeys - 列名の配列。
- jobBookmarkKeysSortOrder — ソート順序に基づいて値を比較する方法を定義する文字列。有効な値: "asc"、"desc"。
- useS3ListImplementation — Amazon S3 バケットの内容を一覧表示する際のメモリパフォーマンスの管理に使用されます。詳しくは、「Optimize memory management in AWS Glue」を参照してください。

AWS Glue for Spark での入出力のデータ形式に関するオプション

これらのページでは、AWS Glue for Spark でサポートされているデータ形式の機能のサポートと設定パラメータについて説明します。この情報の使用と適用可能性の説明については、以下を参照してください。

AWS Glue でのデータ形式間の機能のサポート

各データ形式が、異なる AWS Glue の機能をサポートする場合があります。次の一般的な機能は、形式のタイプによってサポートされる場合とサポートされない場合があります。データ形式についてのドキュメントを参照し、当社の機能を活用して要件を満たす方法を理解してください。

読み取り	AWS Glue では、コネクタなどのリソースを追加することなく、データ形式を認識し解釈できます。
書き込み	AWS Glue では、リソースを追加することなく、この形式でデータを書き込むことができます。他の Spark 環境と同様に、サードパーティライブラリをジョブに含め、標準の Apache Spark 関数を使用してデータを書き込むことが

	できます。ライブラリを含める方法の詳細については、「the section called “Python ライブラリ”」を参照してください。
ストリーミングの読み取り	AWS Glue は、Apache Kafka、Amazon Managed Streaming for Apache Kafka、または Amazon Kinesis メッセージストリームから、このデータ形式を認識して解釈できます。ストリームは一貫した形式でデータを表示することを想定しているため、`DataFrames` として読み込まれます。
小さなファイルのグループ化	AWS Glue は、AWS Glue 変換を実行する際にファイルをグループ化し、各ノードに送信されるバッチを処理できます。これにより、大量の小さなファイルを扱うワークロードのパフォーマンスを大幅に向上できます。詳細については、「the section called “入力ファイルのグループ化”」を参照してください。
ジョブのブックマーク	AWS Glue は、ジョブブックマークを使用して、ジョブの実行間で同じデータセットに対して同じ作業を実行する変換の進行状況を追跡できます。これにより、最後のジョブ実行以降の新しいデータに対してのみ作業を行う必要があるデータセットを含むワークロードのパフォーマンスを向上させることができます。詳細については、「the section called “ジョブのブックマークを使用した処理済みデータの追跡”」を参照してください。

AWS Glue でデータ形式を操作するために使用されるパラメータ

特定の AWS Glue 接続タイプは複数の `format` タイプをサポートしており、`GlueContext.write_dynamic_frame.from_options` のようなメソッドを使用する場合は、`format_options` オブジェクトを使用してデータ形式に関する情報を指定する必要があります。

- `s3` - 詳細については、AWS Glue: S3 接続パラメータ の「Connection types and options for ETL」(ETL の接続タイプとオプション) を参照してください。また、Python でのこの接続タイプを容易にするメソッド the section called “create_dynamic_frame_from_options” および the section called “write_dynamic_frame_from_options” と、対応する Scala メソッド the section called “getSourceWithFormat” および the section called “getSinkWithFormat” についてのドキュメントもご覧ください。

- `kinesis` - 詳細については、AWS Glue: Kinesis 接続パラメータ の「Connection types and options for ETL」(ETL の接続タイプとオプション) を参照してください。また、この接続タイプを容易にするメソッドについてのドキュメント: the section called “create_data_frame_from_options” と、対応する Scala メソッド the section called “createDataFrameFromOptions” についてのドキュメントもご覧ください。
- `kafka` - 詳細については、AWS Glue: Kafka 接続パラメータ の「Connection types and options for ETL」(ETL の接続タイプとオプション) を参照してください。また、この接続タイプを容易にするメソッドについてのドキュメント: the section called “create_data_frame_from_options” と、対応する Scala メソッド the section called “createDataFrameFromOptions” についてのドキュメントもご覧ください。

一部の接続タイプでは、`format_options` は必要ありません。例えば、通常の使用においては、リレーショナルデータベースへの JDBC 接続では整合性のある表形式のデータ形式でデータを取得します。したがって、JDBC 接続からの読み取りでは、`format_options` は必要ありません。

Glue でデータを読み書きする一部のメソッドでは、`format_options` が必要です。例えば、AWS Glue クローラーで `GlueContext.create_dynamic_frame.from_catalog` を使用します。クローラーによってデータの形式が決定されます。クローラーを使用する場合、AWS Glue 分類子がデータを調べ、データ形式を表す方法について賢く判断します。次に、データの表現を AWS Glue Data Catalog に保存します。これを AWS Glue ETL スクリプト内で使用し、`GlueContext.create_dynamic_frame.from_catalog` メソッドによってデータを取得できます。クローラーを使用すると、データ形式に関する情報を手動で指定する必要がなくなります。

AWS Lake Formation government table にアクセスするジョブの場合、AWS Glue は Lake Formation のgovernment table でサポートされているすべてのフォーマットの読み書きをサポートします。現在サポートされている AWS Lake Formation government table のフォーマット一覧は、AWS Lake Formation デベロッパーガイドの「ガバメントテーブルの注意点と制限事項」を参照してください。

> **Note**
>
> Apache Parquetを書くために、AWS Glue ETL では、ダイナミックフレーム用に最適化されたカスタム Parquet ライタータイプのオプションを指定することで、管理されたテーブルへの書き込みのみをサポートします。`parquet` フォーマットでガバメントテーブルに書き込む場合は、テーブルパラメータの `useGlueParquetWriter` の値でキー `true` に追加する必要があります。

トピック

- AWS Glue で CSV 形式を使用する
- AWS Glue で Parquet 形式を使用する
- AWS Glue で XML 形式を使用する
- AWS Glue で Avro 形式を使用する
- AWS Glue で grokLog 形式を使用する
- AWS Glue で Ion 形式を使用する
- AWS Glue で JSON 形式を使用する
- AWS Glue で ORC 形式を使用する
- AWS Glue ETL ジョブでのデータレイクフレームワークの使用
- 共有設定リファレンス

AWS Glue で CSV 形式を使用する

AWS Glue はソースからデータを取得し、さまざまなデータ形式で保存および転送されたターゲットにデータを書き込みます。このドキュメントでは、データが CSV データ形式で保存または転送される場合に、AWS Glue でデータを使用する際に利用できる機能について説明します。

AWS Glue は、カンマ区切り値 (CSV) ファイル形式をサポートしています。この形式は、最小限の行をベースにしたデータ形式です。CSV は厳密に基準に準拠していないことがよくありますが、詳細については RFC 4180 および RFC 7111 を参照してください。

AWS Glue を使用して、Amazon S3 およびストリーミングソースから CSV を読み取ることができ、Amazon S3 に CSV を書き込むこともできます。S3 から、CSV ファイルが含まれる `bzip` および `gzip` アーカイブを読み書きすることができます。このページで説明する設定ではなく、S3 接続パラメータ 上で圧縮動作を設定します。

次の表は、CSV 形式オプションをサポートする一般的な AWS Glue 機能を示しています。

読み込み	書き込み	ストリーミングの読み取り	小さなファイルのグループ化	ジョブのブックマーク
サポート	サポート対象	サポート対象	サポート対象	サポート

例: S3 から CSV ファイルまたはフォルダを読み取る

前提条件: 読み取りたい CSV ファイルまたはフォルダへの S3 パス (s3path) が必要です。

設定: 関数オプションで format="csv" を指定します。connection_options で、paths キーを使用して s3path を指定します。リーダーが S3 とやり取りする方法は、connection_options で設定できます。詳細については、AWS Glue: S3 接続パラメータ の「ETL の接続タイプとオプション」を参照してください。リーダーが CSV ファイルを解釈する方法は、format_options で設定できます。詳細については、「CSV 設定リファレンス」を参照してください。

次の AWS Glue ETL スクリプトは、S3 から CSV ファイルまたはフォルダを読み取るプロセスを示しています。

optimizePerformance 設定キーを使用して、一般的なワークフローのパフォーマンスを最適化するカスタム CSV リーダーを提供します。このリーダーがワークロードに適しているかどうかを判断するには、「the section called “最適化された CSV リーダーを使用する”」を参照してください。

Python

この例では create_dynamic_frame.from_options メソッドを使用します。

```
# Example: Read CSV from S3
# For show, we handle a CSV with a header row.  Set the withHeader option.
# Consider whether optimizePerformance is right for your workflow.

from pyspark.context import SparkContext
from awsglue.context import GlueContext

sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)
spark = glueContext.spark_session

dynamicFrame = glueContext.create_dynamic_frame.from_options(
    connection_type="s3",
```

```
    connection_options={"paths": ["s3://s3path"]},
    format="csv",
    format_options={
        "withHeader": True,
        # "optimizePerformance": True,
    },
)
```

スクリプト (pyspark.sql.DataFrame) では DataFrame を使用することもできます。

```
dataFrame = spark.read\
    .format("csv")\
    .option("header", "true")\
    .load("s3://s3path")
```

Scala

この例では getSourceWithFormat 操作を使用します。

```
// Example: Read CSV from S3
// For show, we handle a CSV with a header row.  Set the withHeader option.
// Consider whether optimizePerformance is right for your workflow.

import com.amazonaws.services.glue.util.JsonOptions
import com.amazonaws.services.glue.{DynamicFrame, GlueContext}
import org.apache.spark.SparkContext

object GlueApp {
  def main(sysArgs: Array[String]): Unit = {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)

    val dynamicFrame = glueContext.getSourceWithFormat(
      formatOptions=JsonOptions("""{"withHeader": true}"""),
      connectionType="s3",
      format="csv",
      options=JsonOptions("""{"paths": ["s3://s3path"], "recurse": true}""")
    ).getDynamicFrame()
  }
}
```

スクリプト (org.apache.spark.sql.DataFrame) では DataFrame を使用することもできます。

```
val dataFrame = spark.read
  .option("header","true")
  .format("csv")
  .load("s3://s3path“)
```

例: CSV ファイルおよびフォルダを S3 に書き込む

前提条件: 初期化された DataFrame (dataFrame) または DynamicFrame (dynamicFrame) が必要です。また、予想される S3 出力パスである s3path も必要になります。

設定: 関数オプションで format="csv" を指定します。connection_options で、paths キーを使用して s3path を指定します。ライターが S3 とやり取りする方法は、connection_options で設定できます。詳細については、AWS Glue: S3 接続パラメータ の「ETL の接続タイプとオプション」を参照してください。操作によってファイルの内容が format_options にどのように書き込まれるかを設定できます。詳細については、「CSV 設定リファレンス」を参照してください。次の AWS Glue ETL スクリプトは、S3 に CSV ファイルとフォルダを書き込むプロセスを示しています。

Python

この例では write_dynamic_frame.from_options メソッドを使用します。

```
# Example: Write CSV to S3
# For show, customize how we write string type values.  Set quoteChar to -1 so our
 values are not quoted.

from pyspark.context import SparkContext
from awsglue.context import GlueContext

sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

glueContext.write_dynamic_frame.from_options(
    frame=dynamicFrame,
    connection_type="s3",
    connection_options={"path": "s3://s3path"},
    format="csv",
    format_options={
        "quoteChar": -1,
    },
```

```
)
```

スクリプト (pyspark.sql.DataFrame) では DataFrame を使用することもできます。

```
dataFrame.write\
    .format("csv")\
    .option("quote", None)\
    .mode("append")\
    .save("s3://s3path")
```

Scala

この例では getSinkWithFormat メソッドを使用します。

```
// Example: Write CSV to S3
// For show, customize how we write string type values. Set quoteChar to -1 so our
 values are not quoted.

import com.amazonaws.services.glue.util.JsonOptions
import com.amazonaws.services.glue.{DynamicFrame, GlueContext}
import org.apache.spark.SparkContext

object GlueApp {
  def main(sysArgs: Array[String]): Unit = {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)

    glueContext.getSinkWithFormat(
        connectionType="s3",
        options=JsonOptions("""{"path": "s3://s3path"}"""),
        format="csv"
    ).writeDynamicFrame(dynamicFrame)
  }
}
```

スクリプト (org.apache.spark.sql.DataFrame) では DataFrame を使用することもできます。

```
dataFrame.write
    .format("csv")
    .option("quote", null)
```

```
    .mode("Append")
    .save("s3://s3path")
```

CSV 設定リファレンス

AWS Glue ライブラリが format="csv" を指定している場合には、以下の format_options を使用することができます。

- separator - 区切り文字を指定します。デフォルトはカンマですが、他の任意の文字を指定できます。
  - タイプ: テキスト、デフォルト: ","
- escaper - エスケープに使用する文字を指定します。このオプションは、CSV ファイルを書き込む場合ではなく、読み取る場合にのみ使用します。有効にすると、直後の文字はそのまま使用されます。ただし、よく知られている小さいエスケープセット (\n、\r、\t、\0) を除きます。
  - タイプ: テキスト、デフォルト: なし
- quoteChar - 引用に使用する文字を指定します。デフォルト値は二重引用符です。これに -1 を設定すると、全体的に引用が無効になります。
  - タイプ: テキスト、デフォルト: '"'
- multiLine - 単一のレコードが複数行にまたがることができるかどうかを指定します。これが発生するのは、フィールドに引用符で囲まれた改行文字がある場合などです。複数行にまたがるレコードがある場合は、このオプションを True に設定する必要があります。multiLine を有効にすると、解析の際にファイル分割をより慎重に行う必要があるため、パフォーマンスが低下する可能性があります。
  - タイプ: ブール値、デフォルト: false
- withHeader - 最初の行をヘッダーとして扱うかどうかを指定します。このオプションは DynamicFrameReader クラスで使用できます。
  - タイプ: ブール値、デフォルト: false
- writeHeader - 出力にヘッダーを書き込むかどうかを指定します。このオプションは DynamicFrameWriter クラスで使用できます。
  - タイプ: ブール値、デフォルト: true
- skipFirst - 最初のデータ行をスキップするかどうかを指定します。
  - タイプ: ブール値、デフォルト: false
- optimizePerformance - 高度な SIMD CSV リーダーで、Apache Arrow ベースの列指向メモリ形式を使用するかどうかを指定します。これは、AWS Glue 3.0 以降でのみ使用できます。

  - タイプ: ブール値、デフォルト: `false`
- `strictCheckForQuoting` – CSV を作成する際、Glue は文字列と解釈した値に引用符を追加することがあります。これは、書き出される内容があいまいにならないようにするためです。Glue は、書き出す内容を決める時間を節約するために、特定の状況では引用符が不要でも引用符を追加することがあります。厳密なチェックを有効にすると、より強力な計算が行われ、厳密に必要な場合にのみ引用符が追加されます。これは、AWS Glue 3.0 以降でのみ使用できます。
  - タイプ: ブール値、デフォルト: `false`

ベクトル化された SIMD CSV リーダーで読み取りパフォーマンスを最適化する

AWS Glue バージョン 3.0 では、行ベースの CSV リーダーと比較して、全体的なジョブパフォーマンスを大幅に高速化できる最適化された CSV リーダーが追加されています。

最適化されたリーダーでは:

- CPU SIMD 命令を使用してディスクから読み取ります
- レコードを列形式 (Apache Arrow) で即座にメモリに書き込みます
- レコードをバッチに分割します

これにより、レコードがバッチ処理されるときや、後で列形式に変換されるときの処理時間が節約されます。一例としては、スキーマを変更したり、列ごとにデータを取得したりする場合などが挙げられます。

最適化されたリーダーを使用するには、`format_options` またはテーブルプロパティで `"optimizePerformance"` を `true` に設定します。

```
glueContext.create_dynamic_frame.from_options(
    frame = datasource1,
    connection_type = "s3",
    connection_options = {"paths": ["s3://s3path"]},
    format = "csv",
    format_options={
        "optimizePerformance": True,
        "separator": ","
        },
    transformation_ctx = "datasink2")
```

ベクトル化された CSV リーダーでの制限事項

ベクトル化された CSV リーダーには、次の制限があるので注意が必要です。

- `multiLine` および `escaper` 形式オプションはサポートされません。デフォルトの `escaper` として二重引用符文字 `'"'` を使用します。これらのオプションを設定すると、AWS Glue は自動的にフォールバックし、行ベースの CSV リーダーを使用するようになります。
- ChoiceType の DynamicFrame の作成はサポートされません。
- error records の DynamicFrame の作成はサポートされません。
- 日本語や中国語など、マルチバイト文字を含む CSV ファイルからの読み取りはサポートされません。

AWS Glue で Parquet 形式を使用する

AWS Glue はソースからデータを取得し、さまざまなデータ形式で保存および転送されたターゲットにデータを書き込みます。このドキュメントでは、データが Parquet データ形式で保存または転送される場合に、AWS Glue でデータを使用する際に利用できる機能について説明します。

AWS Glue は Parquet 形式の使用をサポートしています。この形式は、パフォーマンス指向の列ベースのデータ形式です。標準局から発行されている形式の概要については、「Apache Parquet ドキュメントの概要」を参照してください。

AWS Glue を使用して、Amazon S3 およびストリーミングソースから Parquet ファイルを読み取ることができ、Amazon S3 に Parquet ファイルを書き込むこともできます。S3 から、Parquet ファイルが含まれる `bzip` および `gzip` アーカイブを読み書きすることができます。このページで説明する設定ではなく、S3 接続パラメータ 上で圧縮動作を設定します。

次の表は、Parquet 形式オプションをサポートする一般的な AWS Glue 機能を示しています。

読み込み	書き込み	ストリーミングの読み取り	小さなファイルのグループ化	ジョブのブックマーク
サポート	サポート対象	サポート	サポートされていません	サポート対象*

* AWS Glue バージョン 1.0+ でサポート

例: S3 から Parquet ファイルまたはフォルダを読み取る

前提条件: 読み取りたい Parquet ファイルまたはフォルダへの S3 パス (`s3path`) が必要です。

設定: 関数オプションで format="parquet" を指定します。connection_options で、paths キーを使用して s3path を指定します。

リーダーが S3 とやり取りする方法は、connection_options で設定できます。詳細については、AWS Glue: S3 接続パラメータ の「ETL の接続タイプとオプション」を参照してください。

リーダーが Parquet ファイルを解釈する方法は、format_options で設定できます。詳細については、「Parquet 設定リファレンス」を参照してください。

次の AWS Glue ETL スクリプトは、S3 から Parquet ファイルまたはフォルダを読み取るプロセスを示しています。

Python

この例では create_dynamic_frame.from_options メソッドを使用します。

```
# Example: Read Parquet from S3

from pyspark.context import SparkContext
from awsglue.context import GlueContext

sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)
spark = glueContext.spark_session

dynamicFrame = glueContext.create_dynamic_frame.from_options(
    connection_type = "s3",
    connection_options = {"paths": ["s3://s3path/"]},
    format = "parquet"
)
```

スクリプト (pyspark.sql.DataFrame) では DataFrame を使用することもできます。

```
dataFrame = spark.read.parquet("s3://s3path/")
```

Scala

この例では getSourceWithFormat メソッドを使用します。

```
// Example: Read Parquet from S3
```

```
import com.amazonaws.services.glue.util.JsonOptions
import com.amazonaws.services.glue.{DynamicFrame, GlueContext}
import org.apache.spark.SparkContext

object GlueApp {
  def main(sysArgs: Array[String]): Unit = {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)

    val dynamicFrame = glueContext.getSourceWithFormat(
      connectionType="s3",
      format="parquet",
      options=JsonOptions("""{"paths": ["s3://s3path"]}""")
    ).getDynamicFrame()
  }
}
```

スクリプト (`org.apache.spark.sql.DataFrame`) では DataFrame を使用することもできます。

```
spark.read.parquet("s3://s3path/")
```

例: Parquet ファイルおよびフォルダを S3 に書き込む

前提条件: 初期化された DataFrame (`dataFrame`) または DynamicFrame (`dynamicFrame`) が必要です。また、予想される S3 出力パスである `s3path` も必要になります。

設定: 関数オプションで `format="parquet"` を指定します。`connection_options` で、`paths` キーを使用して `s3path` を指定します。

ライターが S3 と対話する方法を、`connection_options` でさらに詳しく変更することができます。詳細については、AWS Glue: S3 接続パラメータ の「ETL の接続タイプとオプション」を参照してください。操作によってファイルの内容が `format_options` にどのように書き込まれるかを設定できます。詳細については、「Parquet 設定リファレンス」を参照してください。

次の AWS Glue ETL スクリプトは、S3 に Parquet ファイルとフォルダを書き込むプロセスを示しています。

`useGlueParquetWriter` 設定キーを通して、DynamicFrame のパフォーマンスを最適化するカスタム Parquet ライターを提供します。このライターがワークロードに適しているかどうかを判断するには、「Glue Parquet Writer」を参照してください。

Python

この例では write_dynamic_frame.from_options メソッドを使用します。

```
# Example: Write Parquet to S3
# Consider whether useGlueParquetWriter is right for your workflow.

from pyspark.context import SparkContext
from awsglue.context import GlueContext

sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

glueContext.write_dynamic_frame.from_options(
    frame=dynamicFrame,
    connection_type="s3",
    format="parquet",
    connection_options={
        "path": "s3://s3path",
    },
    format_options={
        # "useGlueParquetWriter": True,
    },
)
```

スクリプト (pyspark.sql.DataFrame) では DataFrame を使用することもできます。

```
df.write.parquet("s3://s3path/")
```

Scala

この例では getSinkWithFormat メソッドを使用します。

```
// Example: Write Parquet to S3
// Consider whether useGlueParquetWriter is right for your workflow.

import com.amazonaws.services.glue.util.JsonOptions
import com.amazonaws.services.glue.{DynamicFrame, GlueContext}
import org.apache.spark.SparkContext

object GlueApp {
  def main(sysArgs: Array[String]): Unit = {
    val spark: SparkContext = new SparkContext()
```

```
    val glueContext: GlueContext = new GlueContext(spark)

    glueContext.getSinkWithFormat(
        connectionType="s3",
        options=JsonOptions("""{"path": "s3://s3path"}"""),
        format="parquet"
    ).writeDynamicFrame(dynamicFrame)
  }
}
```

スクリプト (org.apache.spark.sql.DataFrame) では DataFrame を使用することもできます。

```
df.write.parquet("s3://s3path/")
```

Parquet 設定リファレンス

AWS Glue ライブラリが format="parquet" を指定している場合には、以下の format_options を使用することができます。

- useGlueParquetWriter - DynamicFrame ワークフローのパフォーマンスを最適化するカスタム Parquet ライターの使用を指定します。使用方法の詳細については、「Glue Parquet Writer」を参照してください。
  - タイプ: ブール値、デフォルト: false
- compression - 使用する圧縮コーデックを指定します。値は org.apache.parquet.hadoop.metadata.CompressionCodecName との完全な互換性があります。
  - タイプ: 列挙型テキスト、デフォルト: "snappy"
  - 値: "uncompressed"、"snappy"、"gzip"、"lzo"
- blockSize - メモリにバッファされる行グループのサイズを、バイト数で指定します。これはパフォーマンスのチューニングに使用します。サイズは正確に、メガバイト数で分割する必要があります。
  - タイプ: 数値、デフォルト: 134217728
  - デフォルト値は 128 MB です。
- pageSize - ページのサイズをバイト数で指定します。これはパフォーマンスのチューニングに使用します。ページは、単一のレコードにアクセスするために完全に読み取らなければならない最小単位です。

- タイプ: 数値、デフォルト: 1048576
- デフォルト値は 1 MB です。

> **Note**
>
> 加えて、基盤となる SparkSQL コードで受け入れられるすべてのオプションは、connection_options マップパラメータを介し、この形式に渡されます。例えば、mergeSchema などの Spark 設定を AWS Glue Spark リーダーに行うことで、すべてのファイルのスキーマをマージすることができます。

書き込みパフォーマンスを AWS Glue Parquet ライターで最適化する

> **Note**
>
> AWS Glue Parquet ライターには、これまで glueparquet 形式タイプでアクセスしていました。このアクセス方法は非推奨となりました。代わりに、useGlueParquetWriter を有効にして parquet タイプを使用してください。

AWS Glue Parquet ライターには、Parquet ファイルの書き込みを高速化できるパフォーマンス強化が組み込まれています。これまでのライターでは、書き込む前にスキーマを計算していました。Parquet 形式では、すぐに取得できる形でスキーマを保存しないため、時間がかかることがありました。AWS Glue Parquet ライターを使用すれば、事前に計算済みのスキーマは必要なくなります。データが到着次第、ライターがスキーマを動的に計算して変更します。

useGlueParquetWriter を指定する場合、以下の制限に注意してください。

- ライターは、列の追加や削除などのスキーマの進化のみをサポートし、ResolveChoice のように列タイプの変更はサポートしません。
- ライターは、空の DataFrame の書き込み (スキーマのみのファイルを書き込む場合など) をサポートしていません。enableUpdateCatalog=True の設定によって AWS Glue データカタログと統合する場合、空の DataFrame を書き込もうとしてもデータカタログは更新されません。これにより、スキーマなしでデータカタログにテーブルが作成されます。

変換時にこれらの制限が必要とならない場合、AWS Glue Parquet ライターをオンにすることで、パフォーマンスが向上するはずです。

AWS Glue で XML 形式を使用する

AWS Glue はソースからデータを取得し、さまざまなデータ形式で保存および転送されたターゲットにデータを書き込みます。このドキュメントでは、データが XML データ形式で保存または転送される場合に、AWS Glue でデータを使用する際に利用できる機能について説明します。

AWS Glue は XML 形式の使用をサポートしています。この形式は、行ベースでも列ベースでもない、非常に設定可能性が高く、厳密に定義されたデータ構造を表します。XML は高度に標準化されています。標準局から発行されている形式の概要については、「XML エッセンシャル」を参照してください。

AWS Glue を使用して、Amazon S3 から XML ファイルを読み込んだり、XML ファイルが含まれる bzip および gzip アーカイブを読み込んだりすることができます。このページで説明する設定ではなく、S3 接続パラメータ 上で圧縮動作を設定します。

次の表は、XML 形式オプションをサポートする一般的な AWS Glue 機能を示しています。

読み込み	書き込み	ストリーミングの読み取り	小さなファイルのグループ化	ジョブのブックマーク
サポート	サポートされていません	サポートされていません	サポート	サポート

例: S3 から XML を読み込む

XML リーダーは XML タグ名を取得します。入力内にあるそのタグを持つ要素を調べてスキーマを推測し、対応する値を DynamicFrame に入力します。AWS Glue XML の機能は、Apache Spark の XML データソースに類似した動作をします。このリーダーをそのプロジェクトのドキュメントと比較することで、基本的な動作に関する洞察が得られる可能性があります。

前提条件: 読み取りたい XML ファイルまたはフォルダへの S3 パス (s3path) と、XML ファイルに関するいくつかの情報が必要です。また、読み込みたい XML 要素のタグである xmlTag が必要です。

設定: 関数オプションで format="xml" を指定します。connection_options で、paths キーを使用して s3path を指定します。リーダーが S3 とやり取りする方法は、connection_options でさらに詳しく設定できます。詳細については、AWS Glue: S3 接続パラメータ の「ETL の接続タイプとオプション」を参照してください。format_options で、rowTag キーを使用して xmlTag

を指定します。リーダーが XML ファイルを解釈する方法は、format_options でさらに詳しく設定できます。詳細については、「XML 設定リファレンス」を参照してください。

次の AWS Glue ETL スクリプトは、S3 から XML ファイルまたはフォルダを読み取るプロセスを示しています。

Python

この例では create_dynamic_frame.from_options メソッドを使用します。

```
# Example: Read XML from S3
# Set the rowTag option to configure the reader.

from awsglue.context import GlueContext
from pyspark.context import SparkContext

sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

dynamicFrame = glueContext.create_dynamic_frame.from_options(
    connection_type="s3",
    connection_options={"paths": ["s3://s3path"]},
    format="xml",
    format_options={"rowTag": "xmlTag"},
)
```

スクリプト (pyspark.sql.DataFrame) では DataFrame を使用することもできます。

```
dataFrame = spark.read\
    .format("xml")\
    .option("rowTag", "xmlTag")\
    .load("s3://s3path")
```

Scala

この例では getSourceWithFormat 操作を使用します。

```
// Example: Read XML from S3
// Set the rowTag option to configure the reader.

import com.amazonaws.services.glue.util.JsonOptions
```

```
import com.amazonaws.services.glue.GlueContext
import org.apache.spark.sql.SparkSession

val glueContext = new GlueContext(SparkContext.getOrCreate())
val sparkSession: SparkSession = glueContext.getSparkSession

object GlueApp {
  def main(sysArgs: Array[String]): Unit = {
    val dynamicFrame = glueContext.getSourceWithFormat(
      formatOptions=JsonOptions("""{"rowTag": "xmlTag"}"""),
      connectionType="s3",
      format="xml",
      options=JsonOptions("""{"paths": ["s3://s3path"], "recurse": true}""")
    ).getDynamicFrame()
}
```

スクリプト (org.apache.spark.sql.DataFrame) では DataFrame を使用することもできます。

```
val dataFrame = spark.read
  .option("rowTag", "xmlTag")
  .format("xml")
  .load("s3://s3path“)
```

XML 設定リファレンス

AWS Glue ライブラリが format="xml" を指定している場合には、以下の format_options を使用することができます。

- rowTag - 行として扱うファイル内の XML タグを指定します。行のタグを自己終了型にすることはできません。
  - タイプ: テキスト、必須
- encoding - 文字エンコードを指定します。これは、ランタイム環境がサポートする Charset の名前またはエイリアスにすることができます。エンコーディングのサポートに関しては特に保証していませんが、主要なエンコーディングは動作するはずです。
  - タイプ: テキスト、デフォルト: "UTF-8"
- excludeAttribute - 要素の属性を除外するかどうかを指定します。
  - タイプ: ブール値、デフォルト: false

- treatEmptyValuesAsNulls - 空白文字を null 値として扱うかどうかをします。
  - タイプ: ブール値、デフォルト: false
- attributePrefix - 子要素テキストから区別するために属性に付加するプレフィックス。このプレフィックスをフィールド名として使用します。
  - タイプ: テキスト、デフォルト: "_"
- valueTag - 要素内に子を持たない属性がある場合、値に使用するタグ。
  - タイプ: テキスト、デフォルト: "_VALUE"
- ignoreSurroundingSpaces - 値を囲む空白文字を無視するかどうかを指定します。
  - タイプ: ブール値、デフォルト: false
- withSchema - 推論されたスキーマを上書きしたい場合に、期待されるスキーマを含みます。このオプションを使用しない場合は、AWS Glue で XML データからスキーマを推定します。
  - タイプ: テキスト、デフォルト: 該当なし
  - 値は StructType を表す JSON オブジェクトである必要があります。

XML スキーマを手動で指定する

手動 XML スキーマの例

これは、XML データ用にスキーマを指定するために、withSchema 形式オプションを使用する場合の例です。

```
from awsglue.gluetypes import *

schema = StructType([
  Field("id", IntegerType()),
  Field("name", StringType()),
  Field("nested", StructType([
    Field("x", IntegerType()),
    Field("y", StringType()),
    Field("z", ChoiceType([IntegerType(), StringType()]))
  ]))
])

datasource0 = create_dynamic_frame_from_options(
    connection_type,
    connection_options={"paths": ["s3://xml_bucket/someprefix"]},
    format="xml",
    format_options={"withSchema": json.dumps(schema.jsonValue())},
```

```
    transformation_ctx = ""
)
```

AWS Glue で Avro 形式を使用する

AWS Glue はソースからデータを取得し、さまざまなデータ形式で保存および転送されたターゲットにデータを書き込みます。このドキュメントでは、データが Avro データ形式で保存または転送される場合に、AWS Glue でデータを使用する際に利用できる機能について説明します。

AWS Glue は Avro 形式の使用をサポートしています。この形式は、パフォーマンス指向の行ベースのデータ形式です。標準局による形式の概要については、「Apache Avro 1.8.2 Documentation」(Apache Avro ドキュメント 1.8.2) を参照してください。

AWS Glue を使用して、Amazon S3 およびストリーミングソースから Avro ファイルを読み取り、Avro ファイルを Amazon S3 に書き込むことができます。S3 から、Avro ファイルが含まれる `bzip2` および `gzip` アーカイブを読み書きすることができます。さらに、Avro ファイルが含まれる `deflate`、`snappy`、`xz` アーカイブを書き込むこともできます。このページで説明する設定ではなく、S3 接続パラメータ 上で圧縮動作を設定します。

次の表は、Avro 形式のオプションをサポートする一般的な AWS Glue オペレーションを示しています。

読み込み	書き込み	ストリーミングの読み取り	小さなファイルのグループ化	ジョブのブックマーク
サポート	サポート	サポート対象*	サポートされていません	サポート

制限付きで、* がサポートされています。詳細については、「the section called “Avro ストリーミングソースに関する注意事項と制約事項”」を参照してください。

例: S3 から Avro ファイルまたはフォルダを読み取る

前提条件: 読み取る Avro ファイルまたはフォルダへの S3 パス (`s3path`) が必要です。

設定: 関数オプションで `format="avro"` を指定します。`connection_options` で、`paths` キーを使用して `s3path` を指定します。リーダーが S3 とやり取りする方法は、`connection_options` で設定できます。詳細については、AWS Glue: the section called “S3 接続パラメータ” の「ETL の入

力および出力のデータ形式オプション」を参照してください。リーダーが Avro ファイルを解釈する方法は、format_options で設定できます。詳細については、Avro 設定リファレンスを参照してください。

次の AWS Glue ETL スクリプトは、S3 から Avro ファイルまたはフォルダを読み取るプロセスを示しています。

Python

この例では create_dynamic_frame.from_options メソッドを使用します。

```
from pyspark.context import SparkContext
from awsglue.context import GlueContext

sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

dynamicFrame = glueContext.create_dynamic_frame.from_options(
    connection_type="s3",
    connection_options={"paths": ["s3://s3path"]},
    format="avro"
)
```

Scala

この例では getSourceWithFormat 操作を使用します。

```
import com.amazonaws.services.glue.util.JsonOptions
import com.amazonaws.services.glue.GlueContext
import org.apache.spark.sql.SparkContext

object GlueApp {
  def main(sysArgs: Array[String]): Unit = {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)

    val dynamicFrame = glueContext.getSourceWithFormat(
      connectionType="s3",
      format="avro",
      options=JsonOptions("""{"paths": ["s3://s3path"]}""")
    ).getDynamicFrame()
  }
```

例: Avro ファイルおよびフォルダを S3 に書き込む

前提条件: 初期化された DataFrame (dataFrame) または DynamicFrame (dynamicFrame) が必要です。また、予想される S3 出力パスである s3path も必要になります。

設定: 関数オプションで format="avro" を指定します。connection_options で、paths キーを使用して s3path を指定します。ライターが S3 と対話する方法を、connection_options でさらに詳しく変更することができます。詳細については、AWS Glue: the section called “S3 接続パラメータ” の「ETL の入力および出力のデータ形式オプション」を参照してください。リーダーが Avro ファイルを解釈する方法は、format_options で変更できます。詳細については、Avro 設定リファレンスを参照してください。

次の AWS Glue ETL スクリプトは、S3 に Avro ファイルまたはフォルダを書き込むプロセスを示しています。

Python

この例では write_dynamic_frame.from_options メソッドを使用します。

```
from pyspark.context import SparkContext
from awsglue.context import GlueContext

sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

glueContext.write_dynamic_frame.from_options(
    frame=dynamicFrame,
    connection_type="s3",
    format="avro",
    connection_options={
        "path": "s3://s3path"
    }
)
```

Scala

この例では getSinkWithFormat メソッドを使用します。

```
import com.amazonaws.services.glue.util.JsonOptions
import com.amazonaws.services.glue.{DynamicFrame, GlueContext}
import org.apache.spark.SparkContext

object GlueApp {
```

```
  def main(sysArgs: Array[String]): Unit = {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)

    glueContext.getSinkWithFormat(
      connectionType="s3",
      options=JsonOptions("""{"path": "s3://s3path"}"""),
      format="avro"
    ).writeDynamicFrame(dynamicFrame)
  }
}
```

Avro 設定リファレンス

AWS Glue ライブラリが `format="avro"` を指定する場合は、以下の `format_options` を使用することができます。

- `version` – サポートする Apache Avro リーダー/ライター形式のバージョンを指定します。デフォルト値は「1.7」です。Avro 論理型の読み取りと書き込みを有効にするには、`format_options={"version": “1.8”}` を指定します。詳細については、「Apache Avro 1.7.7 の仕様」および「Apache Avro 1.8.2 の仕様」を参照してください。

  Apache Avro 1.8 コネクタは、次の論理型変換をサポートしています。

リーダーの場合: この表に、Avro リーダー 1.7 と 1.8 について、Avro データ型 (論理型と Avro プリミティブ型) と AWS GlueDynamicFrame データ型の間での型変換を示します。

Avro データ型: 論理型	Avro データ型: Avro プリミティブ型	GlueDynamicFrame データ型: Avro リーダー 1.7	GlueDynamicFrame データ型: Avro リーダー 1.8
10 進数	bytes	BINARY	10 進数
10 進数	固定	BINARY	10 進数
日付	整数	INT	日付
時間 (ミリ秒)	整数	INT	INT

Avro データ型: 論理型	Avro データ型: Avro プリミティブ型	GlueDynamicFrame データ型: Avro リーダー 1.7	GlueDynamicFrame データ型: Avro リーダー 1.8
時間 (マイクロ秒)	long	LONG	LONG
タイムスタンプ (ミリ秒)	long	LONG	タイムスタンプ
タイムスタンプ (マイクロ秒)	long	LONG	LONG
所要時間 (論理型ではない)	12 で固定	BINARY	BINARY

ライターの場合: この表に、Avro ライター 1.7 と 1.8 について、データ型と Avro データ型 AWS GlueDynamicFrame の間での型変換を示します。

AWS Glue**DynamicFrame** データ型	Avro データ型: Avro ライター 1.7	Avro データ型: Avro タイター 1.8
10 進数	文字列	decimal
日付	文字列	date
タイムスタンプ	文字列	timestamp-micros

Avro Spark DataFrame サポート

Spark DataFrame API から Avro を使用するには、対応する Spark のバージョン用の Spark Avro プラグインをインストールする必要があります。ジョブで使用できる Spark のバージョンは、AWS Glue のバージョンによって決まります。Spark のバージョンの詳細については、「the section called “AWS Glue バージョン”」を参照してください。このプラグインは Apache によって管理されており、特定のサポートを保証するものではありません。

AWS Glue 2.0 - Spark Avro プラグインのバージョン 2.4.3 を使用します。この JAR は Maven Central にあります。org.apache.spark:spark-avro_2.12:2.4.3 を参照してください。

AWS Glue 3.0 - Spark Avro プラグインのバージョン 3.1.1 を使用します。この JAR は Maven Central にあります。org.apache.spark:spark-avro_2.12:3.1.1 を参照してください。

AWS Glue の ETL ジョブに追加の JAR を含めるには、`--extra-jars` ジョブパラメータを使用します。ジョブパラメータについては、「the section called “ジョブのパラメータ”」を参照してください。AWS Management Console で、このパラメータを設定することもできます。

AWS Glue で grokLog 形式を使用する

AWS Glue はソースからデータを取得し、さまざまなデータ形式で保存および転送されたターゲットにデータを書き込みます。このドキュメントでは、データが大まかに構造化されたプレーンテキスト形式で保存または転送される場合に、AWS Glue でデータを使用する際に利用できる機能を紹介します。

AWS Glue は、Grok パターンの使用をサポートしています。Grok パターンは、正規表現のキャプチャグループに似ています。プレーンテキストファイル内の文字シーケンスのパターンを認識し、タイプと目的を与えます。AWS Glue での主な目的はログを読み取ることです。作成者による Grok の概要については、「Logstash Reference: Grok filter plugin」(Logstash リファレンス: Grok フィルタープラグイン) を参照してください。

読み込み	書き込み	ストリーミングの読み取り	小さなファイルのグループ化	ジョブのブックマーク
サポート	該当しません	サポート	サポート	サポートされていません

grokLog 設定リファレンス

`format="grokLog"` には、以下の `format_options` 値を使用できます。

- `logFormat` – ログの形式と一致する Grok パターンを指定します。
- `customPatterns` – ここで使用する追加の Grok パターンを指定します。
- `MISSING` – 欠落した値の識別に使用するシグナルを指定します。デフォルト: `'-'`。
- `LineCount` – 各ログレコードの行数を指定します。デフォルト値は `'1'` です。現在 1 行のレコードのみがサポートされています。

- `StrictMode` – strict モードを有効にするかどうかを指定するブール値。厳格モードでは、リーダーは自動的な型変換や復旧を行いません。デフォルト値は `"false"` です。

AWS Glue で Ion 形式を使用する

AWS Glue はソースからデータを取得し、さまざまなデータ形式で保存および転送されたターゲットにデータを書き込みます。このドキュメントでは、データが Ion データ形式で保存または転送される場合に、AWS Glue でデータを使用する際に利用できる機能を紹介します。

AWS Glue は Ion 形式の使用をサポートしています。この形式は、データ構造 (行または列ベースではない) を交換可能なバイナリおよびプレーンテキスト形式で表します。作成者による形式の概要については、「Amazon Ion」を参照してください。詳細については、Amazon Ion の仕様に関するドキュメントを参照してください。

AWS Glue を使用して、Amazon S3 から Ion ファイルを読み取ることができます。S3 から、Ion ファイルを含む `bzip` および `gzip` アーカイブを読み取ることができます。このページで説明する設定ではなく、S3 接続パラメータ 上で圧縮動作を設定します。

次の表は、Ion 形式のオプションをサポートする一般的な AWS Glue オペレーションを示しています。

読み込み	書き込み	ストリーミングの読み取り	小さなファイルのグループ化	ジョブのブックマーク
サポート	サポートされていません	サポートされていません	サポート	サポートされていません

例: S3 から Ion ファイルとフォルダを読み取る

前提条件: 読み取る Ion ファイルまたはフォルダへの S3 パス (`s3path`) が必要です。

設定: 関数オプションで `format="json"` を指定します。`connection_options` で、`paths` キーを使用して `s3path` を指定します。リーダーが S3 とやり取りする方法は、`connection_options` で設定できます。詳細については、AWS Glue: the section called “S3 接続パラメータ” の「ETL の接続タイプとオプション」を参照してください。

次の AWS Glue ETL スクリプトは、S3 から Ion ファイルまたはフォルダを読み取るプロセスを示しています。

Python

この例では create_dynamic_frame.from_options メソッドを使用します。

```
# Example: Read ION from S3

from pyspark.context import SparkContext
from awsglue.context import GlueContext

sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

dynamicFrame = glueContext.create_dynamic_frame.from_options(
    connection_type="s3",
    connection_options={"paths": ["s3://s3path"]},
    format="ion"
)
```

Scala

この例では getSourceWithFormat 操作を使用します。

```
// Example: Read ION from S3

import com.amazonaws.services.glue.util.JsonOptions
import com.amazonaws.services.glue.GlueContext
import org.apache.spark.SparkContext

object GlueApp {
  def main(sysArgs: Array[String]): Unit = {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)

    val dynamicFrame = glueContext.getSourceWithFormat(
      connectionType="s3",
      format="ion",
      options=JsonOptions("""{"paths": ["s3://s3path"], "recurse": true}""")
    ).getDynamicFrame()
  }
}
```

Ion 設定リファレンス

`format="ion"` の `format_options` 値はありません。

AWS Glue で JSON 形式を使用する

AWS Glue はソースからデータを取得し、さまざまなデータ形式で保存および転送されたターゲットにデータを書き込みます。このドキュメントでは、JSON データ形式で保存または転送されているデータを、AWS Glue で使用するために用意されている機能について説明します。

AWS Glue は、JSON 形式の使用をサポートしています。この形式は、行ベースまたは列ベースではなく、形状は一貫しているが内容は柔軟なデータ構造を表します。JSON は、複数の機関が発行する並列した規格で定義されており、そのうちの 1 つが ECMA-404 です。一般的に参照されるソースによる形式の概要については、「Introducing JSON」(JSON の概要) を参照してください。

AWS Glue を使用して、JSON ファイルや `bzip` および `gzip` などの圧縮された JSON ファイルを Amazon S3 から 読み取ることができます。このページで説明する設定ではなく、S3 接続パラメータ 上で圧縮動作を設定します。

読み込み	書き込み	ストリーミングの読み取り	小さなファイルのグループ化	ジョブのブックマーク	
サポート	サポート対象	サポート対象	サポート対象	サポート	

例: S3 から JSON ファイルまたはフォルダを読み取る

前提条件: 読み取る JSON ファイルまたはフォルダへの S3 パス (`s3path`) が必要です。

設定: 関数オプションで `format="json"` を指定します。`connection_options` で、`paths` キーを使用して `s3path` を指定します。さらに、接続オプションで、読み取り操作がS3 を通過する方法を変更することができます。詳細については、「the section called “S3 接続パラメータ”」を参照してください。リーダーが JSON ファイルを解釈する方法は、`format_options` で設定できます。詳細については、JSON 設定リファレンスを参照してください。

次の AWS Glue ETL スクリプトは、S3 から JSON ファイルまたはフォルダを読み取るプロセスを示しています。

Python

この例では create_dynamic_frame.from_options メソッドを使用します。

```
# Example: Read JSON from S3
# For show, we handle a nested JSON file that we can limit with the JsonPath
 parameter
# For show, we also handle a JSON where a single entry spans multiple lines
# Consider whether optimizePerformance is right for your workflow.

from pyspark.context import SparkContext
from awsglue.context import GlueContext

sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)
spark = glueContext.spark_session

dynamicFrame = glueContext.create_dynamic_frame.from_options(
    connection_type="s3",
    connection_options={"paths": ["s3://s3path"]},
    format="json",
    format_options={
        "jsonPath": "$.id",
        "multiline": True,
        # "optimizePerformance": True, -> not compatible with jsonPath, multiline
    }
)
```

スクリプト (pyspark.sql.DataFrame) では DataFrame を使用することもできます。

```
dataFrame = spark.read\
    .option("multiline", "true")\
    .json("s3://s3path")
```

Scala

この例では getSourceWithFormat 操作を使用します。

```
// Example: Read JSON from S3
// For show, we handle a nested JSON file that we can limit with the JsonPath
 parameter
// For show, we also handle a JSON where a single entry spans multiple lines
```

```
// Consider whether optimizePerformance is right for your workflow.

import com.amazonaws.services.glue.util.JsonOptions
import com.amazonaws.services.glue.{DynamicFrame, GlueContext}
import org.apache.spark.SparkContext

object GlueApp {
  def main(sysArgs: Array[String]): Unit = {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)

    val dynamicFrame = glueContext.getSourceWithFormat(
      formatOptions=JsonOptions("""{"jsonPath": "$.id", "multiline": true,
 "optimizePerformance":false}"""),
      connectionType="s3",
      format="json",
      options=JsonOptions("""{"paths": ["s3://s3path"], "recurse": true}""")
    ).getDynamicFrame()
  }
}
```

スクリプト (pyspark.sql.DataFrame) では DataFrame を使用することもできます。

```
val dataFrame = spark.read
    .option("multiline", "true")
    .json("s3://s3path")
```

例: JSON ファイルおよびフォルダを S3 に書き込む

前提条件: 初期化された DataFrame (dataFrame) または DynamicFrame (dynamicFrame) が必要です。また、予想される S3 出力パスである s3path も必要になります。

設定: 関数オプションで format="json" を指定します。connection_options で、paths キーを使用して s3path を指定します。ライターが S3 と対話する方法を、connection_options でさらに詳しく変更することができます。詳細については、AWS Glue: the section called “S3 接続パラメータ” の「ETL の入力および出力のデータ形式オプション」を参照してください。ライターが JSON ファイルを解釈する方法は、format_options で設定できます。詳細については、JSON 設定リファレンスを参照してください。

次の AWS Glue ETL スクリプトは、S3 から JSON ファイルまたはフォルダを書き込むプロセスを示しています。

Python

この例では write_dynamic_frame.from_options メソッドを使用します。

```
# Example: Write JSON to S3

from pyspark.context import SparkContext
from awsglue.context import GlueContext

sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

glueContext.write_dynamic_frame.from_options(
    frame=dynamicFrame,
    connection_type="s3",
    connection_options={"path": "s3://s3path"},
    format="json"
)
```

スクリプト (pyspark.sql.DataFrame) では DataFrame を使用することもできます。

```
df.write.json("s3://s3path/")
```

Scala

この例では getSinkWithFormat メソッドを使用します。

```
// Example: Write JSON to S3

import com.amazonaws.services.glue.util.JsonOptions
import com.amazonaws.services.glue.{DynamicFrame, GlueContext}
import org.apache.spark.SparkContext

object GlueApp {
  def main(sysArgs: Array[String]): Unit = {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)

    glueContext.getSinkWithFormat(
        connectionType="s3",
        options=JsonOptions("""{"path": "s3://s3path"}"""),
        format="json"
    ).writeDynamicFrame(dynamicFrame)
```

```
  }
}
```

スクリプト (pyspark.sql.DataFrame) では DataFrame を使用することもできます。

```
df.write.json("s3://s3path")
```

JSON 設定リファレンス

format="json" には、以下の format_options 値を使用できます。

- jsonPath – レコードに読み込むオブジェクトを特定する JsonPath 式。これは、ファイルが外部配列内にネストされたレコードを含む場合に役立ちます。たとえば、次の JsonPath 式のターゲットは、JSON オブジェクトの id フィールドです。

```
format="json", format_options={"jsonPath": "$.id"}
```

- multiline — 単一のレコードが複数行にまたがることができるかどうかを指定するブール値。これが発生するのは、フィールドに引用符で囲まれた改行文字がある場合などです。複数行にまたがるレコードがある場合は、このオプションを "true" に設定する必要があります。デフォルト値は "false" であり、解析時によりアグレッシブなファイル分割を可能にします。
- optimizePerformance – 高度な SIMD JSON リーダーで、Apache Arrow ベースの列指向メモリ形式を使用するかどうかを指定するブール値。これは、AWS Glue 3.0 でのみ使用できます。multiline または jsonPath と互換性がありません。これらのオプションのいずれかを指定すると、AWS Glue は標準のリーダーにフォールバックするように指示されます。
- withSchema — 「the section called “XML スキーマを指定する”」で説明されている形式でテーブルスキーマを指定する文字列の値。非カタログ接続から読み取る場合にのみ、optimizePerformance で使用されます。

Apache Arrow 列指向形式によりベクトル化された SIMD JSON リーダーの使用

AWS Glue バージョン 3.0 では、JSON データ用のベクター化されたリーダーが追加されています。特定の条件下では、標準のリーダーと比較して 2 倍高速に動作します。このリーダーには、このセクションに記載されているように、ユーザーが使用前に知っておく必要のある特定の制限があります。

最適化されたリーダーを使用するには、format_options またはテーブルプロパティで "optimizePerformance" を True に設定します。カタログを読み取らない限り、withSchema を

指定する必要もあります。withSchema は、「the section called “XML スキーマを指定する”」で説明されている入力を期待します

```
// Read from S3 data source
glueContext.create_dynamic_frame.from_options(
    connection_type = "s3",
    connection_options = {"paths": ["s3://s3path"]},
    format = "json",
    format_options={
        "optimizePerformance": True,
        "withSchema": SchemaString
        })

// Read from catalog table
glueContext.create_dynamic_frame.from_catalog(
    database = database,
    table_name = table,
    additional_options = {
    // The vectorized reader for JSON can read your schema from a catalog table
 property.
        "optimizePerformance": True,
        })
```

AWS Glue ライブラリで *SchemaString* を構築する方法の詳細については、「the section called “型”」を参照してください。

ベクトル化された CSV リーダーでの制限事項

以下の制限事項に留意してください。

- ネストされたオブジェクトまたは配列値を持つ JSON 要素はサポートされていません。指定されている場合、AWS Glue は標準のリーダーにフォールバックします。
- カタログから、または withSchema で、スキーマを指定する必要があります。
- multiline または jsonPath と互換性がありません。これらのオプションのいずれかを指定すると、AWS Glue は標準のリーダーにフォールバックするように指示されます。
- 入力スキーマと一致しない入力レコードを指定すると、リーダーは失敗します。
- エラーレコードは作成されません。
- マルチバイト文字 (日本語や中国語の文字など) を含む JSON ファイルはサポートされていません。

AWS Glue で ORC 形式を使用する

AWS Glue はソースからデータを取得し、さまざまなデータ形式で保存および転送されたターゲットにデータを書き込みます。このドキュメントでは、データが ORC データ形式で保存または転送される場合に、AWS Glue でデータを使用する際に利用できる機能について説明します。

AWS Glue は、ORC 形式の使用をサポートしています。この形式は、パフォーマンス指向の列ベースのデータ形式です。標準局による形式の概要については、「Apache Orc」を参照してください。

AWS Glue を使用して、Amazon S3 およびストリーミングソースから ORC ファイルを読み取り、Amazon S3 に ORC ファイルを書き込むことができます。S3 から、ORC ファイルを含む bzip および gzip アーカイブを読み書きすることができます。このページで説明する設定ではなく、S3 接続パラメータ 上で圧縮動作を設定します。

次の表は、ORC 形式のオプションをサポートする一般的な AWS Glue の機能を示しています。

読み込み	書き込み	ストリーミングの読み取り	小さなファイルのグループ化	ジョブのブックマーク
サポート	サポート対象	サポート	サポートされていません	サポート対象*

*AWS Glue バージョン 1.0+ でサポート

例: S3 から ORC ファイルまたはフォルダを読み取る

前提条件: 読み取る ORC ファイルまたはフォルダへの S3 パス (s3path) が必要です。

設定: 関数オプションで format="orc" を指定します。connection_options で、paths キーを使用して s3path を指定します。リーダーが S3 とやり取りする方法は、connection_options で設定できます。詳細については、AWS Glue: the section called “S3 接続パラメータ” の「ETL の接続タイプとオプション」を参照してください。

次の AWS Glue ETL スクリプトは、S3 から ORC ファイルまたはフォルダを読み取るプロセスを示しています。

Python

この例では create_dynamic_frame.from_options メソッドを使用します。

```
from pyspark.context import SparkContext
```

```
from awsglue.context import GlueContext

sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

dynamicFrame = glueContext.create_dynamic_frame.from_options(
    connection_type="s3",
    connection_options={"paths": ["s3://s3path"]},
    format="orc"
)
```

スクリプト (pyspark.sql.DataFrame) では DataFrame を使用することもできます。

```
dataFrame = spark.read\
    .orc("s3://s3path")
```

Scala

この例では getSourceWithFormat 操作を使用します。

```
import com.amazonaws.services.glue.util.JsonOptions
import com.amazonaws.services.glue.GlueContext
import org.apache.spark.sql.SparkContext

object GlueApp {
  def main(sysArgs: Array[String]): Unit = {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)

    val dynamicFrame = glueContext.getSourceWithFormat(
      connectionType="s3",
      format="orc",
      options=JsonOptions("""{"paths": ["s3://s3path"]}""")
    ).getDynamicFrame()
  }
}
```

スクリプト (pyspark.sql.DataFrame) では DataFrame を使用することもできます。

```
val dataFrame = spark.read
    .orc("s3://s3path")
```

例: ORC ファイルおよびフォルダを S3 に書き込む

前提条件: 初期化された DataFrame (`dataFrame`) または DynamicFrame (`dynamicFrame`) が必要です。また、予想される S3 出力パスである `s3path` も必要になります。

設定: 関数オプションで `format="orc"` を指定します。接続オプションでは、`s3path` を指定するための `paths` キーを使用します。ライターが S3 と対話する方法を、`connection_options` でさらに詳しく変更することができます。詳細については、AWS Glue: the section called “S3 接続パラメータ” の「ETL の入力および出力のデータ形式オプション」を参照してください。次のコード例は、プロセスを示しています。

Python

この例では write_dynamic_frame.from_options メソッドを使用します。

```
from pyspark.context import SparkContext
from awsglue.context import GlueContext

sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

glueContext.write_dynamic_frame.from_options(
    frame=dynamicFrame,
    connection_type="s3",
    format="orc",
    connection_options={
        "path": "s3://s3path"
    }
)
```

スクリプト (`pyspark.sql.DataFrame`) では DataFrame を使用することもできます。

```
df.write.orc("s3://s3path/")
```

Scala

この例では getSinkWithFormat メソッドを使用します。

```
import com.amazonaws.services.glue.util.JsonOptions
import com.amazonaws.services.glue.{DynamicFrame, GlueContext}
import org.apache.spark.SparkContext
```

```
object GlueApp {
  def main(sysArgs: Array[String]): Unit = {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)

    glueContext.getSinkWithFormat(
      connectionType="s3",
      options=JsonOptions("""{"path": "s3://s3path"}"""),
      format="orc"
    ).writeDynamicFrame(dynamicFrame)
  }
}
```

スクリプト (pyspark.sql.DataFrame) では DataFrame を使用することもできます。

```
df.write.orc("s3://s3path/")
```

ORC 設定リファレンス

format="orc" の format_options 値はありません。ただし、基になる SparkSQL コードで受け入れられるオプションは、connection_options マップパラメータを介して渡すことができます。

AWS Glue ETL ジョブでのデータレイクフレームワークの使用

オープンソースのデータレイクフレームワークを使用するいことで、Amazon S3 上に構築されたデータレイクに保存するファイルの増分データ処理を簡素化できます。AWS Glue 3.0 以降で、以下のオープンソースデータレイクフレームワークがサポートされています。

- Apache Hudi
- Linux Foundation Delta Lake
- Apache Iceberg

これらのフレームワークはネイティブにサポートされているので、Amazon S3 に保存するデータに対する読み書きが、一貫性のあるトランザクションにより実行できます。これらのフレームワークを AWS Glue ETLジョブで使用する場合にも、別のコネクタをインストールしたり、追加の構成手順を実行したりする必要はありません。

AWS Glue Data Catalog を介してデータセットを管理していれば、Spark DataFrames を使用してデータレイクテーブルを読み書きする場合に、AWS Glue メソッドを利用できます。また、Spark DataFrame API を使用して、Amazon S3 データを読み書きすることもできます。

このビデオでは、Apache Hudi、Apache Iceberg、Delta Lake の仕組みの基本について学ぶことができます。データレイクにデータを挿入、更新、削除する方法と、これらの各フレームワークの仕組みについて説明します。

トピック

- 制限
- AWS Glue での Hudi フレームワークの使用
- AWS Glue での Delta Lake フレームワークの使用
- AWS Glue での Iceberg フレームワークの使用

制限

AWS Glue でデータレイクフレームワークを使用する際には、以下の制限を考慮してください。

- DynamicFrame の次の AWS Glue `GlueContext` メソッドは、データレイクフレームワークテーブルの読み取りと書き込みをサポートしていません。代わりに、DataFrame または Spark DataFrame API については `GlueContext` メソッドを使用します。
  - `create_dynamic_frame.from_catalog`
  - `write_dynamic_frame.from_catalog`
  - `getDynamicFrame`
  - `writeDynamicFrame`
- DataFrame の次の `GlueContext` メソッドは、Lake Formation の許可のコントロールでサポートされています。
  - `create_data_frame.from_catalog`
  - `write_data_frame.from_catalog`
  - `getDataFrame`
  - `writeDataFrame`
- 小さなファイルのグループ化は、サポートされません。
- ジョブのブックマークはサポートされません。

- Apache Hudi 0.10.1 for AWS Glue 3.0 では、Hudi Merge on Read (MoR) テーブルはサポートされていません。
- Apache Iceberg 0.13.1 for AWS Glue 3.0 では、`ALTER TABLE … RENAME TO` を使用できません。

Lake Formation の許可によって管理されるデータレイク形式のテーブルの制限事項

データレイク形式は、Lake Formation の許可を介して AWS Glue ETL と統合されます。`create_dynamic_frame` を使用した DynamicFrame の作成はサポートされていません。詳細については、次の例を参照してください。

- 例: Lake Formation の許可のコントロールを使用した Iceberg テーブルの読み取りおよび書き込み
- 例: Lake Formation の許可のコントロールを使用した Hudi テーブルの読み取りおよび書き込み
- 例: Lake Formation の許可のコントロールを使用した Delta Lake テーブルの読み取りおよび書き込み

> **Note**
>
> Apache Hudi、Apache Iceberg、Delta Lake の Lake Formation 許可による AWS Glue ETLとの統合は、AWS Glue バージョン 4.0 でのみサポートされています。

Apache Iceberg は、Lake Formation の許可を介して AWS Glue ETL と最適に統合されています。ほぼすべてのオペレーションをサポートしており、SQL サポートも含まれています。

Hudi は、管理オペレーションを除くほとんどの基本オペレーションをサポートします。これは、これらのオプションが通常、データフレームの書き込みを介して実行され、`additional_options` を介して指定されるためです。SparkSQL はサポートされていないため、オペレーションのために AWS Glue API を使用して DataFrame を作成する必要があります。

Delta Lake は、テーブルデータの読み取り、付加、および上書きのみをサポートします。Delta Lake では、更新などのさまざまなタスクを実行できるように独自のライブラリを使用する必要があります。

次の機能は、Lake Formation の許可によって管理される Iceberg テーブルでは使用できません。

- AWS Glue ETL を使用した圧縮

- AWS Glue ETL を介した Spark SQL サポート

Lake Formation の許可によって管理される Hudi テーブルの制限は次のとおりです。

- 孤立したファイルの削除

Lake Formation の許可によって管理される Delta Lake テーブルの制限は次のとおりです。

- Delta Lake テーブルの挿入と読み取り以外のすべての機能。

AWS Glue での Hudi フレームワークの使用

AWS Glue 3.0 以降では、データレイク向けに Apache Hudi フレームワークが利用できます。Hudi は、増分データ処理とデータパイプラインの開発を簡素化する、オープンソースのデータレイク用ストレージフレームワークです。このトピックでは、AWS Glue 内でデータを Hudi テーブルに転送または保存する際に利用可能な、各機能について説明します。Hudi の詳細については、公式のApache Hudi ドキュメントを参照してください。

AWS Glue により、Amazon S3 内にある Hudi テーブルの読み取りおよび書き込み操作を実行できます。あるいは、AWS Glue データカタログを使用して、Hudi テーブルを操作することも可能です。挿入、更新、および、すべての Apache Spark オペレーションを含む操作も、追加でサポートされています。

> **Note**
>
> AWS Glue 5.0 の Apache Hudi 0.15.0 実装は HUDI-7001 を内部的に元に戻します。レコードキーが 1 つのフィールドで構成されている場合、Complex Key 生成に関連するリグレッションは表示されません。ただし、この動作は OSS Apache Hudi 0.15.0 とは異なります。
> Apache Hudi 0.10.1 for AWS Glue 3.0 では、Hudi Merge on Read (MoR) テーブルはサポートされません。

次の表に、AWS Glue の各バージョンに含まれている、Hudi のバージョンを一覧で示します。

AWS Glue のバージョン	サポートされる Hudi バージョン
5.0	0.15.0

AWS Glue のバージョン	サポートされる Hudi バージョン
4.0	0.12.1
3.0	0.10.1

AWS Glue がサポートするデータレイクフレームワークの詳細については、「AWS Glue ETL ジョブでのデータレイクフレームワークの使用」を参照してください。

Hudi の有効化

AWS Glue で Hudi を有効化するには、以下のタスクを実行します。

- hudi を `--datalake-formats` のジョブパラメータの値として指定します。詳細については、「AWS Glue ジョブでジョブパラメータを使用する」を参照してください。
- AWS Glue ジョブ用に、`--conf` という名前でキーを作成し、それに次の値を設定します。または、スクリプトで `SparkConf` を使用して、次の構成を設定することもできます。これらの設定は、Apache Spark が Hudi テーブルを適切に処理するために役立ちます。

```
spark.serializer=org.apache.spark.serializer.KryoSerializer
```

- Hudi に関する Lake Formation の許可のサポートは、AWS Glue 4.0 のためにデフォルトで有効になっています。Lake Formation に登録された Hudi テーブルの読み取り/書き込みに追加の設定は必要ありません。登録された Hudi テーブルを読み取るには、AWS Glue ジョブの IAM ロールに SELECT 許可が必要です。登録された Hudi テーブルに書き込むには、AWS Glue ジョブの IAM ロールに SUPER 許可が必要です。Lake Formation の許可の管理の詳細については、「Data Catalog リソースに対する許可の付与と取り消し」を参照してください。

別の Hudi バージョンの使用

AWS Glue でサポートされないバージョンの Hudi を使用するには、`--extra-jars` job パラメーターにより独自の Hudi JAR ファイルを指定します。`--datalake-formats` ジョブパラメータの値として、hudi は含めないでください。AWS Glue 5.0 を使用する場合、`--user-jars-first true` ジョブパラメータを設定する必要があります。

例: Hudi テーブルを Amazon S3 に書き込み、そのテーブルを AWS Glue データカタログに登録する

このスクリプト例では、Hudi テーブルを Amazon S3 に書き込み、そのテーブルを AWS Glue データカタログに登録する方法を示します。この例では、テーブルを登録するために、Hudi Hive Sync ツールを使用します。

> **Note**
>
> この例では、AWS Glue データカタログを Apache Spark Hive のメタストアとして使用するために、`--enable-glue-datacatalog` ジョブパラメータの設定が必要となります。詳細については AWS Glue ジョブでジョブパラメータを使用する を参照してください。

Python

```
# Example: Create a Hudi table from a DataFrame
# and register the table to Glue Data Catalog

additional_options={
    "hoodie.table.name": "<your_table_name>",
    "hoodie.database.name": "<your_database_name>",
    "hoodie.datasource.write.storage.type": "COPY_ON_WRITE",
    "hoodie.datasource.write.operation": "upsert",
    "hoodie.datasource.write.recordkey.field": "<your_recordkey_field>",
    "hoodie.datasource.write.precombine.field": "<your_precombine_field>",
    "hoodie.datasource.write.partitionpath.field": "<your_partitionkey_field>",
    "hoodie.datasource.write.hive_style_partitioning": "true",
    "hoodie.datasource.hive_sync.enable": "true",
    "hoodie.datasource.hive_sync.database": "<your_database_name>",
    "hoodie.datasource.hive_sync.table": "<your_table_name>",
    "hoodie.datasource.hive_sync.partition_fields": "<your_partitionkey_field>",
    "hoodie.datasource.hive_sync.partition_extractor_class":
 "org.apache.hudi.hive.MultiPartKeysValueExtractor",
    "hoodie.datasource.hive_sync.use_jdbc": "false",
    "hoodie.datasource.hive_sync.mode": "hms",
    "path": "s3://<s3Path/>"
}

dataFrame.write.format("hudi") \
    .options(**additional_options) \
    .mode("overwrite") \
    .save()
```

Scala

```
// Example: Example: Create a Hudi table from a DataFrame
// and register the table to Glue Data Catalog

val additionalOptions = Map(
  "hoodie.table.name" -> "<your_table_name>",
  "hoodie.database.name" -> "<your_database_name>",
  "hoodie.datasource.write.storage.type" -> "COPY_ON_WRITE",
  "hoodie.datasource.write.operation" -> "upsert",
  "hoodie.datasource.write.recordkey.field" -> "<your_recordkey_field>",
  "hoodie.datasource.write.precombine.field" -> "<your_precombine_field>",
  "hoodie.datasource.write.partitionpath.field" -> "<your_partitionkey_field>",
  "hoodie.datasource.write.hive_style_partitioning" -> "true",
  "hoodie.datasource.hive_sync.enable" -> "true",
  "hoodie.datasource.hive_sync.database" -> "<your_database_name>",
  "hoodie.datasource.hive_sync.table" -> "<your_table_name>",
  "hoodie.datasource.hive_sync.partition_fields" -> "<your_partitionkey_field>",
  "hoodie.datasource.hive_sync.partition_extractor_class" ->
 "org.apache.hudi.hive.MultiPartKeysValueExtractor",
  "hoodie.datasource.hive_sync.use_jdbc" -> "false",
  "hoodie.datasource.hive_sync.mode" -> "hms",
  "path" -> "s3://<s3Path/>")

dataFrame.write.format("hudi")
  .options(additionalOptions)
  .mode("append")
  .save()
```

例: AWS Glue データカタログを使用した Amazon S3 からの Hudi テーブルの読み取り

この例では、「例: Hudi テーブルを Amazon S3 に書き込み、そのテーブルを AWS Glue データカタログに登録する」で作成した Hudi テーブルを Amazon S3 から読み取ります。

> **Note**
>
> この例では、AWS Glue データカタログを Apache Spark Hive のメタストアとして使用するために、`--enable-glue-datacatalog` ジョブパラメータの設定が必要となります。詳細については AWS Glue ジョブでジョブパラメータを使用するを参照してください。

Python

この例では、GlueContext.create_data_frame.from_catalog() メソッドを使用します。

```
# Example: Read a Hudi table from Glue Data Catalog

from awsglue.context import GlueContext
from pyspark.context import SparkContext

sc = SparkContext()
glueContext = GlueContext(sc)

dataFrame = glueContext.create_data_frame.from_catalog(
    database = "<your_database_name>",
    table_name = "<your_table_name>"
)
```

Scala

この例では getCatalogSource メソッドを使用します。

```
// Example: Read a Hudi table from Glue Data Catalog

import com.amazonaws.services.glue.GlueContext
import org.apache.spark.SparkContext

object GlueApp {
  def main(sysArgs: Array[String]): Unit = {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)

    val dataFrame = glueContext.getCatalogSource(
      database = "<your_database_name>",
      tableName = "<your_table_name>"
    ).getDataFrame()
  }
}
```

例: **DataFrame** を更新して、Amazon S3 内にある Hudi テーブルに挿入する

この例では、「例: Hudi テーブルを Amazon S3 に書き込み、そのテーブルを AWS Glue データカタログに登録する」で作成した Hudi テーブルに対し、AWS Glue データカタログを使用して DataFrame を挿入します。

> **Note**
>
> この例では、AWS Glue データカタログを Apache Spark Hive のメタストアとして使用するために、--enable-glue-datacatalog ジョブパラメータの設定が必要となります。詳細についてはAWS Glue ジョブでジョブパラメータを使用するを参照してください。

Python

この例では、GlueContext.write_data_frame.from_catalog() メソッドを使用します。

```
# Example: Upsert a Hudi table from Glue Data Catalog

from awsglue.context import GlueContext
from pyspark.context import SparkContext

sc = SparkContext()
glueContext = GlueContext(sc)

glueContext.write_data_frame.from_catalog(
    frame = dataFrame,
    database = "<your_database_name>",
    table_name = "<your_table_name>",
    additional_options={
        "hoodie.table.name": "<your_table_name>",
        "hoodie.database.name": "<your_database_name>",
        "hoodie.datasource.write.storage.type": "COPY_ON_WRITE",
        "hoodie.datasource.write.operation": "upsert",
        "hoodie.datasource.write.recordkey.field": "<your_recordkey_field>",
        "hoodie.datasource.write.precombine.field": "<your_precombine_field>",
        "hoodie.datasource.write.partitionpath.field": "<your_partitionkey_field>",
        "hoodie.datasource.write.hive_style_partitioning": "true",
        "hoodie.datasource.hive_sync.enable": "true",
        "hoodie.datasource.hive_sync.database": "<your_database_name>",
        "hoodie.datasource.hive_sync.table": "<your_table_name>",
        "hoodie.datasource.hive_sync.partition_fields": "<your_partitionkey_field>",
```

```
        "hoodie.datasource.hive_sync.partition_extractor_class":
 "org.apache.hudi.hive.MultiPartKeysValueExtractor",
        "hoodie.datasource.hive_sync.use_jdbc": "false",
        "hoodie.datasource.hive_sync.mode": "hms"
    }
)
```

Scala

この例では getCatalogSink メソッドを使用します。

```
// Example: Upsert a Hudi table from Glue Data Catalog

import com.amazonaws.services.glue.GlueContext
import com.amazonaws.services.glue.util.JsonOptions
import org.apacke.spark.SparkContext

object GlueApp {
  def main(sysArgs: Array[String]): Unit = {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)
    glueContext.getCatalogSink("<your_database_name>", "<your_table_name>",
      additionalOptions = JsonOptions(Map(
        "hoodie.table.name" -> "<your_table_name>",
        "hoodie.database.name" -> "<your_database_name>",
        "hoodie.datasource.write.storage.type" -> "COPY_ON_WRITE",
        "hoodie.datasource.write.operation" -> "upsert",
        "hoodie.datasource.write.recordkey.field" -> "<your_recordkey_field>",
        "hoodie.datasource.write.precombine.field" -> "<your_precombine_field>",
        "hoodie.datasource.write.partitionpath.field" ->
 "<your_partitionkey_field>",
        "hoodie.datasource.write.hive_style_partitioning" -> "true",
        "hoodie.datasource.hive_sync.enable" -> "true",
        "hoodie.datasource.hive_sync.database" -> "<your_database_name>",
        "hoodie.datasource.hive_sync.table" -> "<your_table_name>",
        "hoodie.datasource.hive_sync.partition_fields" ->
 "<your_partitionkey_field>",
        "hoodie.datasource.hive_sync.partition_extractor_class" ->
 "org.apache.hudi.hive.MultiPartKeysValueExtractor",
        "hoodie.datasource.hive_sync.use_jdbc" -> "false",
        "hoodie.datasource.hive_sync.mode" -> "hms"
      )))
      .writeDataFrame(dataFrame, glueContext)
  }
```

```
}
```

例: Spark を使用した Amazon S3 からの Hudi テーブルの読み取り

この例では、Spark DataFrame API を使用して Amazon S3 から Hudi テーブルを読み取ります。

Python

```
# Example: Read a Hudi table from S3 using a Spark DataFrame

dataFrame = spark.read.format("hudi").load("s3://<s3path/>")
```

Scala

```
// Example: Read a Hudi table from S3 using a Spark DataFrame

val dataFrame = spark.read.format("hudi").load("s3://<s3path/>")
```

例: スパークを使用した Amazon S3 への Hudi テーブルの書き込み

この例では、Spark を使用して Amazon S3 に Hudi テーブルを書き込みます。

Python

```
# Example: Write a Hudi table to S3 using a Spark DataFrame

dataFrame.write.format("hudi") \
    .options(**additional_options) \
    .mode("overwrite") \
    .save("s3://<s3Path/>")
```

Scala

```
// Example: Write a Hudi table to S3 using a Spark DataFrame

dataFrame.write.format("hudi")
  .options(additionalOptions)
  .mode("overwrite")
  .save("s3://<s3path/>")
```

例: Lake Formation の許可のコントロールを使用した Hudi テーブルの読み取りおよび書き込み

この例では、Lake Formation の許可のコントロールを使用して Hudi テーブルの読み取りと書き込みを行います。

1. Hudi テーブルを作成して Lake Formation に登録します。
   a. Lake Formation の許可のコントロールを有効にするには、まずテーブルの Amazon S3 パスを Lake Formation に登録する必要があります。詳細については、「Registering an Amazon S3 location」を参照してください。Lake Formation コンソールから、または AWS CLI を使用して登録できます。

      ```
      aws lakeformation register-resource --resource-arn arn:aws:s3:::<s3-bucket>/<s3-folder> --use-service-linked-role --region <REGION>
      ```

      Amazon S3 の場所が登録されると、その場所 (またはその子である場所) をポイントするすべての AWS Glue テーブルが、`GetTable` 呼び出しで `IsRegisteredWithLakeFormation` パラメータの値を true として返します。
   b. Spark DataFrame API を介して登録された Amazon S3 パスをポイントする Hudi テーブルを作成します。

      ```
      hudi_options = {
          'hoodie.table.name': table_name,
          'hoodie.database.name': database_name,
          'hoodie.datasource.write.storage.type': 'COPY_ON_WRITE',
          'hoodie.datasource.write.recordkey.field': 'product_id',
          'hoodie.datasource.write.table.name': table_name,
          'hoodie.datasource.write.operation': 'upsert',
          'hoodie.datasource.write.precombine.field': 'updated_at',
          'hoodie.datasource.write.hive_style_partitioning': 'true',
          'hoodie.upsert.shuffle.parallelism': 2,
          'hoodie.insert.shuffle.parallelism': 2,
          'path': <S3_TABLE_LOCATION>,
          'hoodie.datasource.hive_sync.enable': 'true',
          'hoodie.datasource.hive_sync.database': database_name,
          'hoodie.datasource.hive_sync.table': table_name,
          'hoodie.datasource.hive_sync.use_jdbc': 'false',
          'hoodie.datasource.hive_sync.mode': 'hms'
      }

      df_products.write.format("hudi")  \
          .options(**hudi_options)  \
      ```

```
    .mode("overwrite")  \
    .save()
```

2. AWS Glue ジョブ IAM ロールに Lake Formation の許可を付与します。Lake Formation コンソールから、または AWS CLI を使用して許可を付与できます。詳細については、「Lake Formation コンソールと名前付きリソース方式を使用したテーブル許可の付与」を参照してください。

3. Lake Formation に登録されている Hudi テーブルを読み取ります。このコードは、未登録の Hudi テーブルを読み取る場合と同じです。読み取りを成功させるには、AWS Glue ジョブの IAM ロールに SELECT 許可が必要であることに留意してください。

```
val dataFrame = glueContext.getCatalogSource(
     database = "<your_database_name>",
     tableName = "<your_table_name>"
   ).getDataFrame()
```

4. Lake Formation に登録されている Hudi テーブルに書き込みます。このコードは、未登録の Hudi テーブルに書き込む場合と同じです。書き込みを成功させるには、AWS Glue ジョブの IAM ロールに SUPER 許可が必要であることに留意してください。

```
glueContext.getCatalogSink("<your_database_name>", "<your_table_name>",
      additionalOptions = JsonOptions(Map(
        "hoodie.table.name" -> "<your_table_name>",
        "hoodie.database.name" -> "<your_database_name>",
        "hoodie.datasource.write.storage.type" -> "COPY_ON_WRITE",
        "hoodie.datasource.write.operation" -> "<write_operation>",
        "hoodie.datasource.write.recordkey.field" -> "<your_recordkey_field>",
        "hoodie.datasource.write.precombine.field" -> "<your_precombine_field>",
        "hoodie.datasource.write.partitionpath.field" -> "<your_partitionkey_field>",
        "hoodie.datasource.write.hive_style_partitioning" -> "true",
        "hoodie.datasource.hive_sync.enable" -> "true",
        "hoodie.datasource.hive_sync.database" -> "<your_database_name>",
        "hoodie.datasource.hive_sync.table" -> "<your_table_name>",
        "hoodie.datasource.hive_sync.partition_fields" ->
 "<your_partitionkey_field>",
        "hoodie.datasource.hive_sync.partition_extractor_class" ->
 "org.apache.hudi.hive.MultiPartKeysValueExtractor",
        "hoodie.datasource.hive_sync.use_jdbc" -> "false",
        "hoodie.datasource.hive_sync.mode" -> "hms"
      )))
      .writeDataFrame(dataFrame, glueContext)
```

AWS Glue での Delta Lake フレームワークの使用

AWS Glue 3.0 以降では、Linux Foundation Delta Lake フレームワークを利用できます。Delta Lake は、ACID トランザクションの実行、メタデータ処理のスケーリング、さらにストリーミングとバッチデータ処理の統合を支援する、オープンソースのデータレイクストレージフレームワークです。このトピックでは、Delta Lake テーブルに転送または保存するデータに対して、AWS Glue 内で利用可能な機能について説明します。Delta Lake の詳細については、公式の Delta Lake のドキュメントを参照してください。

Amazon S3 内にある Delta Lake テーブルに対する読み取りおよび書き込みの操作は、AWS Glue を使用して実行できます。あるいは、AWS Glue データカタログにより Delta Lake テーブルを操作することもできます。挿入、更新、およびテーブルに対する一括での読み取りと書き込みなど、その他の操作もサポートされています。Delta Lake テーブルを使用する場合、Delta Lake Python ライブラリに備わったメソッド (例えば、`DeltaTable.forPath`) も使用可能です。Delta Lake Python ライブラリの詳細については、「Delta Lake の Python ドキュメント」を参照してください。

次の表に、AWS Glue の各バージョンに含まれている Delta Lake のバージョンを一覧で示します。

AWS Glue のバージョン	サポートされている Delta Lake バージョン
5.0	3.3.0
4.0	2.1.0
3.0	1.0.0

AWS Glue がサポートするデータレイクフレームワークの詳細については、「AWS Glue ETL ジョブでのデータレイクフレームワークの使用」を参照してください。

AWS Glue のための Delta Lake の有効化

AWS Glue で Delta Lake を有効化するには、以下のタスクを実行します。

- `delta` を `--datalake-formats` のジョブパラメータの値として指定します。詳細については、「AWS Glue ジョブでジョブパラメータを使用する」を参照してください。
- AWS Glue ジョブ用に、`--conf` という名前でキーを作成し、それに次の値を設定します。または、スクリプトで `SparkConf` を使用して、次の構成を設定することもできます。これらの設定は、Apache Spark が Delta Lake テーブルを適切に処理するために役立ちます。

```
spark.sql.extensions=io.delta.sql.DeltaSparkSessionExtension --conf
 spark.sql.catalog.spark_catalog=org.apache.spark.sql.delta.catalog.DeltaCatalog --
conf
 spark.delta.logStore.class=org.apache.spark.sql.delta.storage.S3SingleDriverLogStore
```

- Delta テーブルに関する Lake Formation の許可のサポートは、AWS Glue 4.0 のためにデフォルトで有効になっています。Lake Formation に登録された Delta テーブルの読み取り/書き込みに追加の設定は必要ありません。登録された Delta テーブルを読み取るには、AWS Glue ジョブの IAM ロールに SELECT 許可が必要です。登録された Delta テーブルに書き込むには、AWS Glue ジョブの IAM ロールに SUPER 許可が必要です。Lake Formation の許可の管理の詳細については、「Data Catalog リソースに対する許可の付与と取り消し」を参照してください。

別のバージョンの Delta Lake を使用する

AWS Glue でサポートされないバージョンの Delta Lake を使用するには、`--extra-jars` ジョブパラメータにより、独自の Delta Lake JAR ファイルを指定します。`--datalake-formats` ジョブパラメータの値として、`delta` は含めないでください。AWS Glue 5.0 を使用する場合、`--user-jars-first true` ジョブパラメータを設定する必要があります。この場合、Delta Lake Python ライブラリを使用するには、`--extra-py-files` ジョブパラメータにより、ライブラリの JAR ファイルを指定する必要があります。Python ライブラリは、Delta Lake JAR ファイルにパッケージ化されています。

例: Delta Lake テーブルを Amazon S3 に書き込んで、そのテーブルを AWS Glue データカタログに登録する

次の AWS Glue ETL スクリプトは、Delta Lake テーブルを Amazon S3 に書き込み、そのテーブルを AWS Glue データカタログに登録する方法を示しています。

Python

```
# Example: Create a Delta Lake table from a DataFrame
# and register the table to Glue Data Catalog

additional_options = {
    "path": "s3://<s3Path>"
}
dataFrame.write \
    .format("delta") \
    .options(**additional_options) \
```

```
    .mode("append") \
    .partitionBy("<your_partitionkey_field>") \
    .saveAsTable("<your_database_name>.<your_table_name>")
```

Scala

```
// Example: Example: Create a Delta Lake table from a DataFrame
// and register the table to Glue Data Catalog

val additional_options = Map(
  "path" -> "s3://<s3Path>"
)
dataFrame.write.format("delta")
  .options(additional_options)
  .mode("append")
  .partitionBy("<your_partitionkey_field>")
  .saveAsTable("<your_database_name>.<your_table_name>")
```

例: AWS Glue データカタログを使用して Amazon S3 から Delta Lake テーブルを読み取る

次の AWS Glue ETL スクリプトでは、「例: Delta Lake テーブルを Amazon S3 に書き込んで、そのテーブルを AWS Glue データカタログに登録する」で作成した Delta Lake テーブルの読み取りを実行します。

Python

この例では create_data_frame.from_catalog メソッドを使用します。

```
# Example: Read a Delta Lake table from Glue Data Catalog

from awsglue.context import GlueContext
from pyspark.context import SparkContext

sc = SparkContext()
glueContext = GlueContext(sc)

df = glueContext.create_data_frame.from_catalog(
    database="<your_database_name>",
    table_name="<your_table_name>",
    additional_options=additional_options
)
```

Scala

この例では getCatalogSource メソッドを使用します。

```
// Example: Read a Delta Lake table from Glue Data Catalog

import com.amazonaws.services.glue.GlueContext
import org.apacke.spark.SparkContext

object GlueApp {
  def main(sysArgs: Array[String]): Unit = {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)
    val df = glueContext.getCatalogSource("<your_database_name>",
 "<your_table_name>",
      additionalOptions = additionalOptions)
      .getDataFrame()
  }
}
```

例: AWS Glue データカタログを使用して Amazon S3 内にある Delta Lake テーブルに **DataFrame** を挿入する

この例では、「例: Delta Lake テーブルを Amazon S3 に書き込んで、そのテーブルを AWS Glue データカタログに登録する」で作成した Delta Lake テーブルにデータを挿入します。

> **Note**
>
> この例では、AWS Glue データカタログを Apache Spark Hive のメタストアとして使用するために、`--enable-glue-datacatalog` ジョブパラメータの設定が必要となります。詳細については AWS Glue ジョブでジョブパラメータを使用するを参照してください。

Python

この例では、write_data_frame.from_catalog メソッドを使用します。

```
# Example: Insert into a Delta Lake table in S3 using Glue Data Catalog

from awsglue.context import GlueContext
```

```
from pyspark.context import SparkContext

sc = SparkContext()
glueContext = GlueContext(sc)

glueContext.write_data_frame.from_catalog(
    frame=dataFrame,
    database="<your_database_name>",
    table_name="<your_table_name>",
    additional_options=additional_options
)
```

Scala

この例では getCatalogSink メソッドを使用します。

```
// Example: Insert into a Delta Lake table in S3 using Glue Data Catalog

import com.amazonaws.services.glue.GlueContext
import org.apacke.spark.SparkContext

object GlueApp {
  def main(sysArgs: Array[String]): Unit = {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)
    glueContext.getCatalogSink("<your_database_name>", "<your_table_name>",
      additionalOptions = additionalOptions)
      .writeDataFrame(dataFrame, glueContext)
  }
}
```

例: Spark API を使用して Amazon S3 から Delta Lake テーブルを読み取る

この例では、Spark API を使用して Amazon S3 から Delta Lake を読み取ります。

Python

```
# Example: Read a Delta Lake table from S3 using a Spark DataFrame

dataFrame = spark.read.format("delta").load("s3://<s3path/>")
```

Scala

```
// Example: Read a Delta Lake table from S3 using a Spark DataFrame

val dataFrame = spark.read.format("delta").load("s3://<s3path/>")
```

例: Spark を使用した Amazon S3 への Delta Lake テーブルの書き込み

この例では、Spark を使用して Amazon S3 に Delta Lake テーブルを書き込みます。

Python

```
# Example: Write a Delta Lake table to S3 using a Spark DataFrame

dataFrame.write.format("delta") \
    .options(**additional_options) \
    .mode("overwrite") \
    .partitionBy("<your_partitionkey_field>") \
    .save("s3://<s3Path>")
```

Scala

```
// Example: Write a Delta Lake table to S3 using a Spark DataFrame

dataFrame.write.format("delta")
  .options(additionalOptions)
  .mode("overwrite")
  .partitionBy("<your_partitionkey_field>")
  .save("s3://<s3path/>")
```

例: Lake Formation の許可のコントロールを使用した Delta Lake テーブルの読み取りおよび書き込み

この例では、Lake Formation の許可のコントロールを使用して Delta Lake テーブルの読み取りと書き込みを行います。

1. Delta テーブルを作成して Lake Formation に登録する
   a. Lake Formation の許可のコントロールを有効にするには、まずテーブルの Amazon S3 パスを Lake Formation に登録する必要があります。詳細については、「Registering an Amazon S3

location」を参照してください。Lake Formation コンソールから、または AWS CLI を使用して登録できます。

```
aws lakeformation register-resource --resource-arn arn:aws:s3:::<s3-bucket>/<s3-folder> --use-service-linked-role --region <REGION>
```

Amazon S3 の場所が登録されると、その場所 (またはその子である場所) をポイントするすべての AWS Glue テーブルが、`GetTable` 呼び出しで `IsRegisteredWithLakeFormation` パラメータの値を true として返します。

b. Spark を介して登録された Amazon S3 パスをポイントする Delta テーブルを作成します。

> **Note**
>
> 次に Python の例を示します。

```
dataFrame.write \
 .format("delta") \
 .mode("overwrite") \
 .partitionBy("<your_partitionkey_field>") \
 .save("s3://<the_s3_path>")
```

データが Amazon S3 に書き込まれた後、AWS Glue クローラーを使用して新しい Delta カタログテーブルを作成します。詳細については、「AWS Glue クローラーによるネイティブデータレイクテーブルサポートの紹介」を参照してください。

AWS Glue `CreateTable` API を通じてテーブルを手動で作成することもできます。

2. AWS Glue ジョブ IAM ロールに Lake Formation の許可を付与します。Lake Formation コンソールから、または AWS CLI を使用して許可を付与できます。詳細については、「Lake Formation コンソールと名前付きリソース方式を使用したテーブル許可の付与」を参照してください。
3. Lake Formation に登録されている Delta テーブルを読み取ります。このコードは、未登録の Delta テーブルを読み取る場合と同じです。読み取りを成功させるには、AWS Glue ジョブの IAM ロールに SELECT 許可が必要であることに留意してください。

```
# Example: Read a Delta Lake table from Glue Data Catalog

df = glueContext.create_data_frame.from_catalog(
    database="<your_database_name>",
```

```
    table_name="<your_table_name>",
    additional_options=additional_options
)
```

4. Lake Formation に登録されている Delta テーブルに書き込みます。このコードは、未登録の Delta テーブルに書き込む場合と同じです。書き込みを成功させるには、AWS Glue ジョブの IAM ロールに SUPER 許可が必要であることに留意してください。

   デフォルトでは、AWS Glue は saveMode として Append を使用します。これを変更するには、additional_options で saveMode オプションを設定します。Delta テーブルでの saveMode サポートの詳細については、「テーブルに書き込む」を参照してください。

```
glueContext.write_data_frame.from_catalog(
    frame=dataFrame,
    database="<your_database_name>",
    table_name="<your_table_name>",
    additional_options=additional_options
)
```

AWS Glue での Iceberg フレームワークの使用

AWS Glue 3.0 以降では、データレイク向けに Apache Iceberg フレームワークがサポートされています。Iceberg により、SQL テーブルと同様な機能を持つ、高性能なテーブルフォーマットが提供されます。このトピックでは、AWS Glue 内でデータを Iceberg テーブルに転送または保存する際に利用可能な、機能について説明します。Iceberg の詳細については、公式のApache Iceberg ドキュメントを参照してください。

Amazon S3 内の Iceberg テーブルに対する読み取りと書き込みの操作は、AWS Glue を使用して実行することができます。あるいは、AWS Glue データカタログを使用して Iceberg テーブルを操作することも可能です。挿入に加え、「Spark クエリ」 や 「Spark 書き込み」 のすべての操作も、追加でサポートされています。更新は Iceberg テーブルではサポートされていません。

> **Note**
>
> Apache Iceberg 0.13.1 for AWS Glue 3.0 では、ALTER TABLE … RENAME TO はサポートされません。

次の表に、AWS Glue の各バージョンに含まれる Iceberg のバージョンを一覧で示します。

AWS Glue のバージョン	サポートされる Iceberg バージョン
5.0	1.7.1
4.0	1.0.0
3.0	0.13.1

AWS Glue がサポートするデータレイクフレームワークの詳細については、「AWS Glue ETL ジョブでのデータレイクフレームワークの使用」を参照してください。

Iceberg フレームワークの有効化

AWS Glue で Iceberg を有効化するには、以下のタスクを実行します。

- `iceberg` を `--datalake-formats` のジョブパラメータの値として指定します。詳細については、「AWS Glue ジョブでジョブパラメータを使用する」を参照してください。
- AWS Glue ジョブ用に、`--conf` という名前でキーを作成し、それに次の値を設定します。または、スクリプトで `SparkConf` を使用して、次の構成を設定することもできます。これらの設定は、Apache Spark が Iceberg テーブルを適切に処理する際に役立ちます。

```
spark.sql.extensions=org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions
--conf spark.sql.catalog.glue_catalog=org.apache.iceberg.spark.SparkCatalog
--conf spark.sql.catalog.glue_catalog.warehouse=s3://<your-warehouse-dir>/
--conf spark.sql.catalog.glue_catalog.catalog-
impl=org.apache.iceberg.aws.glue.GlueCatalog
--conf spark.sql.catalog.glue_catalog.io-impl=org.apache.iceberg.aws.s3.S3FileIO
```

  Lake Formation に登録されている Iceberg テーブルを読み書きする場合は、AWS Glue 5.0 以降の the section called “FGAC のための Lake Formation” のガイダンスに従ってください。AWS Glue 4.0 で、次の設定を追加して Lake Formation のサポートを有効にします。

```
--conf spark.sql.catalog.glue_catalog.glue.lakeformation-enabled=true
--conf spark.sql.catalog.glue_catalog.glue.id=<table-catalog-id>
```

  AWS Glue 3.0 で Iceberg 0.13.1 を使用する場合には、Amazon DynamoDB ロックマネージャーを使用して確実なアトミックトランザクションを行うために、次の追加事項を設定する必要がありま

す。 AWSGlue 4.0 以降では、デフォルトで楽観的ロックが使用されます。詳細については、公式の Apache Iceberg ドキュメントで「Iceberg と AWS の統合」を参照してください。

```
--conf spark.sql.catalog.glue_catalog.lock-
impl=org.apache.iceberg.aws.glue.DynamoLockManager
--conf spark.sql.catalog.glue_catalog.lock.table=<your-dynamodb-table-name>
```

別バージョンの Iceberg の使用

AWS Glue でサポートされないバージョンの Iceberg を使用するには、`--extra-jars` ジョブパラメータを使用して独自の Iceberg JAR ファイルを指定します。`--datalake-formats` パラメータの値として、`iceberg` を含めないようにしてください。AWS Glue 5.0 を使用する場合、`--user-jars-first true` ジョブパラメータを設定する必要があります。

Iceberg テーブルの暗号化を有効にする

> **Note**
>
> Iceberg テーブルにはサーバー側の暗号化を有効にする独自のメカニズムがあります。AWS Glue のセキュリティ設定に加えて、これらの設定を有効にする必要があります。

Iceberg テーブルでサーバー側の暗号化を有効にするには、「Iceberg ドキュメント」のガイダンスを確認してください。

Iceberg クロスリージョンに Spark 設定を追加する

AWS Glue Data Catalog および AWS Lake Formation を使用して Iceberg クロスリージョンテーブルアクセスの Spark 設定を追加するには、以下の手順に従います。

1. マルチリージョンアクセスポイントを作成します。
2. 以下の Spark パラメータを設定します。

```
-----
    --conf spark.sql.catalog.my_catalog.s3.use-arn-region-enabled=true \
    --conf spark.sql.catalog.{CATALOG}.s3.access-points.bucket1",
 "arn:aws:s3::<account-id>:accesspoint/<mrap-id>.mrap \
    --conf spark.sql.catalog.{CATALOG}.s3.access-points.bucket2",
 "arn:aws:s3::<account-id>:accesspoint/<mrap-id>.mrap
```

```
-----
```

例: Amazon S3 に Iceberg テーブルを書き込み、そのテーブルを AWS Glue データカタログに登録する

このスクリプト例は、Amazon S3 に Iceberg テーブルを書き込む方法を示しています。この例では、Iceberg と AWS の統合を使用して、テーブルを AWS Glue データカタログに登録しています。

Python

```
# Example: Create an Iceberg table from a DataFrame
# and register the table to Glue Data Catalog

dataFrame.createOrReplaceTempView("tmp_<your_table_name>")

query = f"""
CREATE TABLE glue_catalog.<your_database_name>.<your_table_name>
USING iceberg
TBLPROPERTIES ("format-version"="2")
AS SELECT * FROM tmp_<your_table_name>
"""
spark.sql(query)
```

Scala

```
// Example: Example: Create an Iceberg table from a DataFrame
// and register the table to Glue Data Catalog

dataFrame.createOrReplaceTempView("tmp_<your_table_name>")

val query = """CREATE TABLE glue_catalog.<your_database_name>.<your_table_name>
USING iceberg
TBLPROPERTIES ("format-version"="2")
AS SELECT * FROM tmp_<your_table_name>
"""
spark.sql(query)
```

または、Spark メソッドを使用して Amazon S3 とデータカタログに Iceberg テーブルを書き込むこともできます。

前提条件: Iceberg ライブラリが使用するカタログを用意する必要があります。AWS Glue データカタログを使う場合、AWS Glue を使えば簡単です。AWS Glue データカタログは、glue_catalog として Spark ライブラリで使用できるように事前に設定されています。データカタログテーブルは、*atabaseName* と*tableName* で識別されます。AWS Glue データカタログの詳細については、「データ検出とカタログ化」を参照してください

AWS Glue データカタログを使用していない場合は、Spark API を使用してカタログをプロビジョニングする必要があります。詳細については、Spark ドキュメントの「Spark の設定」を参照してください。

この例では、Spark を使用して Amazon S3 と データカタログに Iceberg テーブルを書き込みます。

Python

```
# Example: Write an Iceberg table to S3 on the Glue Data Catalog

# Create (equivalent to CREATE TABLE AS SELECT)
dataFrame.writeTo("glue_catalog.databaseName.tableName") \
    .tableProperty("format-version", "2") \
    .create()

# Append (equivalent to INSERT INTO)
dataFrame.writeTo("glue_catalog.databaseName.tableName") \
    .tableProperty("format-version", "2") \
    .append()
```

Scala

```
// Example: Write an Iceberg table to S3 on the Glue Data Catalog

// Create (equivalent to CREATE TABLE AS SELECT)
dataFrame.writeTo("glue_catalog.databaseName.tableName")
    .tableProperty("format-version", "2")
    .create()

// Append (equivalent to INSERT INTO)
dataFrame.writeTo("glue_catalog.databaseName.tableName")
    .tableProperty("format-version", "2")
    .append()
```

例: AWS Glue データカタログを使用した Amazon S3 からの Iceberg テーブルの読み込み

この例では、「例: Amazon S3 に Iceberg テーブルを書き込み、そのテーブルを AWS Glue データカタログに登録する」で作成した Iceberg テーブルを読み取っています。

Python

この例では、`GlueContext.create_data_frame.from_catalog()` メソッドを使用します。

```
# Example: Read an Iceberg table from Glue Data Catalog

from awsglue.context import GlueContext
from pyspark.context import SparkContext

sc = SparkContext()
glueContext = GlueContext(sc)

df = glueContext.create_data_frame.from_catalog(
    database="<your_database_name>",
    table_name="<your_table_name>",
    additional_options=additional_options
)
```

Scala

この例では getCatalogSource メソッドを使用します。

```
// Example: Read an Iceberg table from Glue Data Catalog

import com.amazonaws.services.glue.GlueContext
import org.apacke.spark.SparkContext

object GlueApp {
  def main(sysArgs: Array[String]): Unit = {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)
    val df = glueContext.getCatalogSource("<your_database_name>",
 "<your_table_name>",
      additionalOptions = additionalOptions)
      .getDataFrame()
  }
}
```

例: AWS Glue データカタログを使用した Amazon S3 にある Iceberg テーブルへの **DataFrame** の挿入

この例では、「例: Amazon S3 に Iceberg テーブルを書き込み、そのテーブルを AWS Glue データカタログに登録する」で作成した Iceberg テーブルにデータを挿入します。

> **Note**
>
> この例では、AWS Glue データカタログを Apache Spark Hive のメタストアとして使用するために、`--enable-glue-datacatalog` ジョブパラメータの設定が必要となります。詳細については AWS Glue ジョブでジョブパラメータを使用するを参照してください。

Python

この例では、`GlueContext.write_data_frame.from_catalog()` メソッドを使用します。

```
# Example: Insert into an Iceberg table from Glue Data Catalog

from awsglue.context import GlueContext
from pyspark.context import SparkContext

sc = SparkContext()
glueContext = GlueContext(sc)

glueContext.write_data_frame.from_catalog(
    frame=dataFrame,
    database="<your_database_name>",
    table_name="<your_table_name>",
    additional_options=additional_options
)
```

Scala

この例では getCatalogSink メソッドを使用します。

```
// Example: Insert into an Iceberg table from Glue Data Catalog

import com.amazonaws.services.glue.GlueContext
import org.apacke.spark.SparkContext
```

```
object GlueApp {
  def main(sysArgs: Array[String]): Unit = {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)
    glueContext.getCatalogSink("<your_database_name>", "<your_table_name>",
      additionalOptions = additionalOptions)
      .writeDataFrame(dataFrame, glueContext)
  }
}
```

例: Spark を使用した Iceberg テーブルの Amazon S3 からの読み取り

前提条件: Iceberg ライブラリが使用するカタログを用意する必要があります。AWS Glue データカタログを使う場合、AWS Glue を使えば簡単です。AWS Glue データカタログは、glue_catalog として Spark ライブラリで使用できるように事前に設定されています。データカタログテーブルは、*atabaseName* と*tableName* で識別されます。AWS Glue データカタログの詳細については、「データ検出とカタログ化」を参照してください

AWS Glue データカタログを使用していない場合は、Spark API を使用してカタログをプロビジョニングする必要があります。詳細については、Spark ドキュメントの「Spark の設定」を参照してください。

この例では、Spark を使用してデータカタログから Amazon S3 の Iceberg テーブルを読み取ります。

Python

```
# Example: Read an Iceberg table on S3 as a DataFrame from the Glue Data Catalog

dataFrame = spark.read.format("iceberg").load("glue_catalog.databaseName.tableName")
```

Scala

```
// Example: Read an Iceberg table on S3 as a DataFrame from the Glue Data Catalog

val dataFrame =
 spark.read.format("iceberg").load("glue_catalog.databaseName.tableName")
```

例: Lake Formation の許可のコントロールを使用した Iceberg テーブルの読み取りおよび書き込み

この例では、Lake Formation の許可のコントロールを使用して Iceberg テーブルの読み取りと書き込みを行います。

> **Note**
>
> この例は、AWS Glue 4.0 でのみ動作します。AWS Glue 5.0 以降では、「the section called "FGAC のための Lake Formation"」のガイダンスに従ってください。

1. Iceberg テーブルを作成して Lake Formation に登録します:

   a. Lake Formation の許可のコントロールを有効にするには、まずテーブルの Amazon S3 パスを Lake Formation に登録する必要があります。詳細については、「Registering an Amazon S3 location」を参照してください。Lake Formation コンソールから、または AWS CLI を使用して登録できます。

   ```
   aws lakeformation register-resource --resource-arn arn:aws:s3:::<s3-bucket>/<s3-folder> --use-service-linked-role --region <REGION>
   ```

   Amazon S3 の場所が登録されると、その場所 (またはその子である場所) をポイントするすべての AWS Glue テーブルが、`GetTable` 呼び出しで `IsRegisteredWithLakeFormation` パラメータの値を true として返します。

   b. Spark SQL を介して登録されたパスをポイントする Iceberg テーブルを作成します。

   > **Note**
   >
   > 次に Python の例を示します。

   ```
   dataFrame.createOrReplaceTempView("tmp_<your_table_name>")

   query = f"""
   CREATE TABLE glue_catalog.<your_database_name>.<your_table_name>
   USING iceberg
   AS SELECT * FROM tmp_<your_table_name>
   """
   spark.sql(query)
   ```

AWS Glue `CreateTable` API を通じてテーブルを手動で作成することもできます。詳細については、「Apache Iceberg テーブルの作成」を参照してください。

> **Note**
>
> `UpdateTable` API は現在、 オペレーションへの入力として Iceberg テーブル形式をサポートしていません。

2. ジョブ IAM ロールに Lake Formation の許可を付与します。Lake Formation コンソールから、または AWS CLI を使用して許可を付与できます。詳細については、次を参照してください: https://docs.aws.amazon.com/lake-formation/latest/dg/granting-table-permissions.html
3. Lake Formation に登録されている Iceberg テーブルを読み取ります。このコードは、未登録の Iceberg テーブルを読み取る場合と同じです。読み取りを成功させるには、AWS Glue ジョブの IAM ロールに SELECT 許可が必要であることに留意してください。

```
# Example: Read an Iceberg table from the AWS Glue Data Catalog
from awsglue.context import GlueContextfrom pyspark.context import SparkContext

sc = SparkContext()
glueContext = GlueContext(sc)

df = glueContext.create_data_frame.from_catalog(
    database="<your_database_name>",
    table_name="<your_table_name>",
    additional_options=additional_options
)
```

4. Lake Formation に登録されている Iceberg テーブルに書き込みます。このコードは、未登録の Iceberg テーブルに書き込む場合と同じです。書き込みを成功させるには、AWS Glue ジョブの IAM ロールに SUPER 許可が必要であることに留意してください。

```
glueContext.write_data_frame.from_catalog(
    frame=dataFrame,
    database="<your_database_name>",
    table_name="<your_table_name>",
    additional_options=additional_options
)
```

共有設定リファレンス

形式タイプには次の format_options 値を使用できます。

- attachFilename — 列名として使用する適切な形式の文字列。このオプションを指定すると、レコードのソースファイルの名前がレコードに追加されます。パラメータ値は列名として使用されます。
- attachTimestamp — 列名として使用する適切な形式の文字列。このオプションを指定すると、レコードのソースファイルの変更時刻がレコードに追加されます。パラメータ値は列名として使用されます。

Spark SQL ジョブ用の AWS Glue Data Catalog のサポート

AWS Glue の Data Catalog は、Apache Hive メタストア互換のカタログです。Data Catalog を外部の Apache Hive メタストアとして使用するように、ジョブと開発エンドポイント AWS Glueを設定できます。その後、Data Catalog に格納されたテーブルに対して Apache Spark SQL クエリを直接実行することができます。AWS Glue ダイナミックフレームは、デフォルトで Data Catalogと 統合されています。また、この機能により Spark SQL ジョブは、Data Catalog を外部の Hive メタストアとして使用できるようになります。

この機能には、AWS Glue API エンドポイントへのネットワークアクセスが必要です。ネットワークアクセスを提供する場合、プライベートサブネットに接続がある AWS Glue ジョブでは VPC エンドポイントまたは NAT ゲートウェイを構成する必要があります。VPCエンドポイントの設定については、「データストアへのネットワークアクセスを設定する」を参照してください。NAT ゲートウェイを作成するには、Amazon VPC ユーザーガイドの「NAT ゲートウェイ」を参照してください。

ジョブ引数と開発エンドポイント引数にそれぞれ "--enable-glue-datacatalog": "" 引数を追加することで、AWS Glue ジョブと開発エンドポイントを設定できます。この引数を渡すことで、Spark が外部 Hive メタストアとして Data Catalog にアクセスできるようにする、特定の設定が構成されます。同時に、SparkSession オブジェクトにある AWS Glue ジョブまたは開発エンドポイント内で作成された Hive のサポートを有効化します。

データ カタログへのアクセスを有効にするには、コンソールの [ジョブの追加] または [エンドポイントの追加] ページのカタログオプション グループで [Hive メタストアとして AWS Glue データ カタログを使用] のチェックボックスをオンにします。ジョブまたは開発エンドポイントに使用される IAM ロールには glue:CreateDatabase アクセス許可が必要です。存在しない場合、「default」というデータベースが Data Catalog に作成されます。

Spark SQL ジョブでこの機能の使用方法の例を見てみましょう。次の例では、s3://awsglue-datasets/examples/us-legislators にある米国国会議員のデータセットをクロールしたと想定しています。

AWS Glue Data Catalog で定義されたテーブルからデータをシリアル化/逆シリアル化するには、Spark ジョブのクラスパスにある AWS Glue Data Catalog で定義された形式の Hive SerDe クラスが、Spark SQL に必要です。

特定の一般的なフォーマットの SerDes は AWS Glue によって配布されています。以下はこれらへの Amazon S3 リンクです。

- JSON
- XML
- Grok

JSON SerDe を追加の JAR として開発エンドポイントに追加します。ジョブでは、引数フィールドで --extra-jars 引数を使用して SerDe を追加できます。詳細については、 を参照してください。AWS Glue ジョブでジョブパラメータを使用する

ここに、Spark SQL 用に Data Catalog を有効にして開発エンドポイントを作成するための入力 JSON の例を示します。

```
{
    "EndpointName": "Name",
    "RoleArn": "role_ARN",
    "PublicKey": "public_key_contents",
    "NumberOfNodes": 2,
    "Arguments": {
      "--enable-glue-datacatalog": ""
    },
    "ExtraJarsS3Path": "s3://crawler-public/json/serde/json-serde.jar"
}
```

Spark SQL を使用して、米国国会議員のデータセットから作成されたテーブルにクエリを実行します。

```
>>> spark.sql("use legislators")
DataFrame[]
```

```
>>> spark.sql("show tables").show()
+-----------+-----------------+-----------+
|   database|        tableName|isTemporary|
+-----------+-----------------+-----------+
|legislators|       areas_json|      false|
|legislators|   countries_json|      false|
|legislators|      events_json|      false|
|legislators| memberships_json|      false|
|legislators|organizations_json|      false|
|legislators|     persons_json|      false|
+-----------+-----------------+-----------+
>>> spark.sql("describe memberships_json").show()
+--------------------+---------+-----------------+
|            col_name|data_type|          comment|
+--------------------+---------+-----------------+
|             area_id|   string|from deserializer|
|     on_behalf_of_id|   string|from deserializer|
|     organization_id|   string|from deserializer|
|                role|   string|from deserializer|
|           person_id|   string|from deserializer|
|legislative_perio...|   string|from deserializer|
|          start_date|   string|from deserializer|
|            end_date|   string|from deserializer|
+--------------------+---------+-----------------+
```

形式の SerDe クラスがジョブのクラスパスで使用できない場合は、以下に示すようなエラーが表示されます。

```
>>> spark.sql("describe memberships_json").show()

Caused by: MetaException(message:java.lang.ClassNotFoundException Class
 org.openx.data.jsonserde.JsonSerDe not found)
    at
 org.apache.hadoop.hive.metastore.MetaStoreUtils.getDeserializer(MetaStoreUtils.java:399)
    at
 org.apache.hadoop.hive.ql.metadata.Table.getDeserializerFromMetaStore(Table.java:276)
    ... 64 more
```

memberships テーブルとは異なる organization_id だけを表示するには、次の SQL クエリを実行します。

```
>>> spark.sql("select distinct organization_id from memberships_json").show()
```

```
+--------------------+
|     organization_id|
+--------------------+
|d56acebe-8fdc-47b...|
|8fa6c3d2-71dc-478...|
+--------------------+
```

動的フレームで同じ手順を実行する必要がある場合は、以下を実行してください。

```
>>> memberships = glueContext.create_dynamic_frame.from_catalog(database="legislators",
 table_name="memberships_json")
>>> memberships.toDF().createOrReplaceTempView("memberships")
>>> spark.sql("select distinct organization_id from memberships").show()
+--------------------+
|     organization_id|
+--------------------+
|d56acebe-8fdc-47b...|
|8fa6c3d2-71dc-478...|
+--------------------+
```

DynamicFrame は ETL オペレーション用に最適化されていますが、Spark SQL が Data Catalog に直接アクセスできるようにすることで、複雑な SQL ステートメントの実行や、既存のアプリケーションの移植が簡素化されます。

ジョブのブックマークを使用する

Spark 用の AWS Glue はジョブのブックマークを使用して処理済みのデータを追跡します。ジョブのブックマーク機能の概要とサポート内容については、「the section called “ジョブのブックマークを使用した処理済みデータの追跡”」を参照してください。ブックマークを使用して AWS Glue ジョブをプログラミングすると、ビジュアルジョブでは実現できない柔軟性が得られます。

- JDBC から読み込む場合、AWS Glue スクリプトのブックマークキーとして使用する列を指定できます。
- 各メソッド呼び出しに適用する `transformation_ctx` を選択できます。

常に適切に設定されたパラメータでスクリプトの先頭に `job.init` を呼び出して、末尾に `job.commit` を呼び出します。これら 2 つの関数は、ブックマークサービスを初期化し、サービスの状態変化を更新します。ブックマークは呼び出さないと機能しません。

ブックマークキーの指定

JDBC ワークフローの場合、ブックマークはキーフィールドの値をブックマークされた値と比較することで、ジョブがどの行を読み込んだかを追跡します。これは Amazon S3 ワークフローには必要ありませんし、適用可能でもありません。ビジュアルエディタを使用せずに AWS Glue スクリプトを記述する場合、どの列をブックマークで追跡するかを指定できます。複数の列を指定することもできます。ユーザー定義のブックマークキーを指定する場合、値の順序にギャップがあってもかまいません。

> **Warning**
>
> ユーザー定義のブックマークキーを使用する場合、キーはそれぞれ厳密に一定間隔で増減する必要があります。複合キーに追加のフィールドを選択しても、「マイナーバージョン」や「リビジョン番号」などの概念のフィールドは、値がデータセット全体で再利用されるため、この基準を満たしません。

`jobBookmarkKeys` および `jobBookmarkKeysSortOrder` は以下の方法で指定できます。

- `create_dynamic_frame.from_catalog` — `additional_options` を使用する。
- `create_dynamic_frame.from_options` — `connection_options` を使用する。

変換コンテキスト

AWS Glue PySpark の動的フレームの多くのメソッドには、`transformation_ctx` というオプションのパラメータが含まれています。このパラメータは ETL 演算子インスタンスの一意の識別子です。`transformation_ctx` パラメータは指定された演算子に対するジョブのブックマーク内の状態情報を識別するために使用されます。具体的には、AWS Glue では `transformation_ctx` を使用してブックマーク状態に対するキーにインデックスを付けます。

> **Warning**
>
> `transformation_ctx` は、スクリプト内の特定のソースのブックマーク状態を検索するためのキーとして機能します。ブックマークを正しく機能させるためには、ソースと関連する `transformation_ctx` の一貫性を常に保つ必要があります。ソースプロパティを変更したり、`transformation_ctx` の名前を変更したりすると、前のブックマークが無効になり、タイムスタンプベースのフィルタリングで正しい結果が得られない場合があります。

ジョブのブックマークが正しく機能するように、ジョブのブックマークのパラメータを有効にし、transformation_ctx パラメータを設定します。transformation_ctx パラメータを渡さない場合、メソッドで使用されている動的フレームやテーブルに対してジョブのブックマークは有効になりません。例えば、ETL ジョブで 2 つの Amazon S3 ソースを読み取って結合する場合、ブックマークを有効にするメソッドに対してのみ transformation_ctx パラメータを渡すことができます。1 つのジョブについてジョブのブックマークをリセットした場合、transformation_ctx の使用には関係なく、このジョブに関連付けられているすべての変換がリセットされます。

DynamicFrameReader クラスの詳細については、「DynamicFrameReader クラス」を参照してください。PySpark 拡張の詳細については、「AWS Glue PySpark 拡張機能リファレンス」を参照してください

例

Example

Amazon S3 データソース用に生成されたスクリプトの例を次に示します。ジョブのブックマークを使用するために必要なスクリプトの部分は、斜体で表示されています。これらの要素の詳細については、GlueContext クラス API および DynamicFrameWriter クラス API を参照してください。

```
# Sample Script
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job

args = getResolvedOptions(sys.argv, ['JOB_NAME'])
sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
job.init(args['JOB_NAME'], args)

datasource0 = glueContext.create_dynamic_frame.from_catalog(
    database = "database",
    table_name = "relatedqueries_csv",
    transformation_ctx = "datasource0"
)

applymapping1 = ApplyMapping.apply(
```

```
    frame = datasource0,
    mappings = [("col0", "string", "name", "string"), ("col1", "string", "number",
 "string")],
    transformation_ctx = "applymapping1"
)

datasink2 = glueContext.write_dynamic_frame.from_options(
    frame = applymapping1,
    connection_type = "s3",
    connection_options = {"path": "s3://input_path"},
    format = "json",
    transformation_ctx = "datasink2"
)


job.commit()
```

Example

JDBC ソース用に生成されたスクリプトの例を以下に示します。ソーステーブルは、empno 列がプライマリキーである従業員テーブルです。デフォルトでは、ブックマークキーが指定されていない場合、ジョブはブックマークキーとしてシーケンシャルプライマリキーを使用しますが、empno は必ずしもシーケンシャルではなく、値にギャップがある可能性があるため、デフォルトのブックマークキーとして適切ではありません。したがって、スクリプトは empno を明示的にブックマークキーとして指定します。コードのその部分は、斜体で表示されます。

```
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job

args = getResolvedOptions(sys.argv, ['JOB_NAME'])

sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
job.init(args['JOB_NAME'], args)

datasource0 = glueContext.create_dynamic_frame.from_catalog(
    database = "hr",
```

```
    table_name = "emp",
    transformation_ctx = "datasource0",
    additional_options = {"jobBookmarkKeys":["empno"],"jobBookmarkKeysSortOrder":"asc"}
)

applymapping1 = ApplyMapping.apply(
    frame = datasource0,
    mappings = [("ename", "string", "ename", "string"), ("hrly_rate", "decimal(38,0)",
 "hrly_rate", "decimal(38,0)"), ("comm", "decimal(7,2)", "comm", "decimal(7,2)"),
 ("hiredate", "timestamp", "hiredate", "timestamp"), ("empno", "decimal(5,0)", "empno",
 "decimal(5,0)"), ("mgr", "decimal(5,0)", "mgr", "decimal(5,0)"), ("photo", "string",
 "photo", "string"), ("job", "string", "job", "string"), ("deptno", "decimal(3,0)",
 "deptno", "decimal(3,0)"), ("ssn", "decimal(9,0)", "ssn", "decimal(9,0)"), ("sal",
 "decimal(7,2)", "sal", "decimal(7,2)")],
    transformation_ctx = "applymapping1"
)

datasink2 = glueContext.write_dynamic_frame.from_options(
    frame = applymapping1,
    connection_type = "s3",
    connection_options = {"path": "s3://hr/employees"},
    format = "csv",
    transformation_ctx = "datasink2"
)

job.commit()
```

AWS Glue Studio 外での機密データ検出の使用

AWS Glue Studio では機密データを検出することができますが、AWS Glue Studio 外でこの機密データ検出機能を使用することも可能です。

マネージド機密データタイプの全リストについては、「Managed data types」を参照してください。

AWS マネージド PII タイプを使用した機密データの検出

AWS Glue は、AWS Glue ETL ジョブで 2 つの API を提供します。これらは、`detect()` および `classifyColumns()` です。

```
  detect(frame: DynamicFrame,
      entityTypesToDetect: Seq[String],
```

```
      outputColumnName: String = "DetectedEntities",
      detectionSensitivity: String = "LOW"): DynamicFrame

  detect(frame: DynamicFrame,
      detectionParameters: JsonOptions,
      outputColumnName: String = "DetectedEntities",
      detectionSensitivity: String = "LOW"): DynamicFrame

   classifyColumns(frame: DynamicFrame,
      entityTypesToDetect: Seq[String],
      sampleFraction: Double = 0.1,
      thresholdFraction: Double = 0.1,
      detectionSensitivity: String = "LOW")
```

AWS マネージド PII タイプとカスタムエンティティタイプを識別するには、`detect()` API を使用できます。検出結果が含まれる新しい列が自動的に作成されます。`classifyColumns()` API は、キーが列名で値が検出されたエンティティタイプのリストになっているマップを返します。`SampleFraction` は、PII エンティティをスキャンしているときにサンプリングするためのデータの割合を示し、`ThresholdFraction` は、列が PII データとして識別されるために満たされる必要があるデータの割合を示します。

行レベルの検出

この例では、ジョブが `detect()` と `classifyColumns()` API を使用して以下のアクションを実行しています。

- Amazon S3 バケットからデータを読み込み、dynamicFrame に変換する
- dynamicFrame 内にある「Email」と「Credit Card」のインスタンスを検出する
- 元の値と、各行の検出結果が網羅された 1 つの列が含まれる dynamicFrame を返す
- 返された dynamicFrame を別の Amazon S3 パスに書き込む

```
  import com.amazonaws.services.glue.GlueContext
  import com.amazonaws.services.glue.MappingSpec
  import com.amazonaws.services.glue.errors.CallSite
  import com.amazonaws.services.glue.util.GlueArgParser
  import com.amazonaws.services.glue.util.Job
  import com.amazonaws.services.glue.util.JsonOptions
  import org.apache.spark.SparkContext
```

```
 import scala.collection.JavaConverters._
 import com.amazonaws.services.glue.ml.EntityDetector

 object GlueApp {
   def main(sysArgs: Array[String]) {
     val spark: SparkContext = new SparkContext()
     val glueContext: GlueContext = new GlueContext(spark)
     val args = GlueArgParser.getResolvedOptions(sysArgs, Seq("JOB_NAME").toArray)
     Job.init(args("JOB_NAME"), glueContext, args.asJava)
     val frame=
glueContext.getSourceWithFormat(formatOptions=JsonOptions("""{"quoteChar": "\"",
"withHeader": true, "separator": ","}"""), connectionType="s3", format="csv",
options=JsonOptions("""{"paths": ["s3://pathToSource"], "recurse": true}"""),
transformationContext="AmazonS3_node1650160158526").getDynamicFrame()

     val frameWithDetectedPII = EntityDetector.detect(frame, Seq("EMAIL",
"CREDIT_CARD"))

     glueContext.getSinkWithFormat(connectionType="s3",
options=JsonOptions("""{"path": "s3://pathToOutput/", "partitionKeys": []}"""),
transformationContext="someCtx",
format="json").writeDynamicFrame(frameWithDetectedPII)

     Job.commit()
   }
 }
```

詳細なアクションによる行レベルの検出

この例では、ジョブが `detect()` API を使用して次のアクションを実行しています。

- Amazon S3 バケットからデータを読み取り、dynamicFrame に変換する
- dynamicFrame 内の「USA_PTIN」、「BANK_ACCOUNT]、「USA_SSN」、「USA_PASSPORT_NUMBER」、および「PHONE_NUMBER」の機密データタイプの検出
- 変更およびマスキングされた値と、各行の検出結果が網羅された 1 つの列が含まれる dynamicFrame を返す
- 返された dynamicFrame を別の Amazon S3 パスに書き込む

上記の detect() API とは対照的に、これはエンティティタイプを検出するための詳細なアクションを使用します。詳細については、「detect() を使用する検出パラメータ」を参照してください。

```
import com.amazonaws.services.glue.GlueContext
import com.amazonaws.services.glue.MappingSpec
import com.amazonaws.services.glue.errors.CallSite
import com.amazonaws.services.glue.util.GlueArgParser
import com.amazonaws.services.glue.util.Job
import com.amazonaws.services.glue.util.JsonOptions
import org.apache.spark.SparkContext
import scala.collection.JavaConverters._
import com.amazonaws.services.glue.ml.EntityDetector

object GlueApp {
  def main(sysArgs: Array[String]) {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)
    val args = GlueArgParser.getResolvedOptions(sysArgs, Seq("JOB_NAME").toArray)
    Job.init(args("JOB_NAME"), glueContext, args.asJava)
    val frame =
 glueContext.getSourceWithFormat(formatOptions=JsonOptions("""{"quoteChar": "\"",
 "withHeader": true, "separator": ","}"""), connectionType="s3", format="csv",
 options=JsonOptions("""{"paths": ["s3://pathToSource"], "recurse": true}"""),
 transformationContext="AmazonS3_node_source").getDynamicFrame()

    val detectionParameters = JsonOptions(
      """
        {
          "USA_DRIVING_LICENSE": [{
            "action": "PARTIAL_REDACT",
            "sourceColumns": ["Driving License"],
            "actionOptions": {
              "matchPattern": "[0-9]",
              "redactChar": "*"
            }
          }],
          "BANK_ACCOUNT": [{
            "action": "DETECT",
            "sourceColumns": ["*"]
          }],
          "USA_SSN": [{
            "action": "SHA256_HASH",
```

```
            "sourceColumns": ["SSN"]
          }],
          "IP_ADDRESS": [{
            "action": "REDACT",
            "sourceColumns": ["IP Address"],
            "actionOptions": {"redactText": "*****"}
          }],
          "PHONE_NUMBER": [{
            "action": "PARTIAL_REDACT",
            "sourceColumns": ["Phone Number"],
            "actionOptions": {
              "numLeftCharsToExclude": 1,
              "numRightCharsToExclude": 0,
              "redactChar": "*"
            }
          }]
        }
      """
    )

    val frameWithDetectedPII = EntityDetector.detect(frame, detectionParameters,
 "DetectedEntities", "HIGH")

    glueContext.getSinkWithFormat(connectionType="s3", options=JsonOptions("""{"path":
 "s3://pathToOutput/", "partitionKeys": []}"""),
 transformationContext="AmazonS3_node_target",
 format="json").writeDynamicFrame(frameWithDetectedPII)

    Job.commit()
  }
}
```

列レベルの検出

この例では、ジョブが `classifyColumns()` API を使用して次のアクションを実行しています。

- Amazon S3 バケットからデータを読み取り、dynamicFrame に変換する
- dynamicFrame 内にある「Email」と「Credit Card」のインスタンスを検出する
- 列の 100% をサンプリングし、エンティティがセルの 10% にある場合に検出済みとしてマークするとともに、感度が [低] になるようにパラメータを設定します

- キーが列名、値が検出されたエンティティタイプのリストであるマップを返します
- 返された dynamicFrame を別の Amazon S3 パスに書き込む

```
import com.amazonaws.services.glue.GlueContext
import com.amazonaws.services.glue.MappingSpec
import com.amazonaws.services.glue.errors.CallSite
import com.amazonaws.services.glue.util.GlueArgParser
import com.amazonaws.services.glue.util.Job
import com.amazonaws.services.glue.util.JsonOptions
import org.apache.spark.SparkContext
import scala.collection.JavaConverters._
import com.amazonaws.services.glue.DynamicFrame
import com.amazonaws.services.glue.ml.EntityDetector

object GlueApp {
  def main(sysArgs: Array[String]) {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)
    val args = GlueArgParser.getResolvedOptions(sysArgs, Seq("JOB_NAME").toArray)
    Job.init(args("JOB_NAME"), glueContext, args.asJava)
    val frame =
 glueContext.getSourceWithFormat(formatOptions=JsonOptions("""{"quoteChar":
 "\"", "withHeader": true, "separator": ",", "optimizePerformance": false}"""),
 connectionType="s3", format="csv", options=JsonOptions("""{"paths": ["s3://
pathToSource"], "recurse": true}"""), transformationContext="frame").getDynamicFrame()

    import glueContext.sparkSession.implicits._

    val detectedDataFrame = EntityDetector.classifyColumns(
        frame,
        entityTypesToDetect = Seq("CREDIT_CARD", "PHONE_NUMBER"),
        sampleFraction = 1.0,
        thresholdFraction = 0.1,
        detectionSensitivity = "LOW"
    )
    val detectedDF = (detectedDataFrame).toSeq.toDF("columnName", "entityTypes")
    val DetectSensitiveData_node = DynamicFrame(detectedDF, glueContext)

    glueContext.getSinkWithFormat(connectionType="s3", options=JsonOptions("""{"path":
 "s3://pathToOutput", "partitionKeys": []}"""), transformationContext="someCtx",
 format="json").writeDynamicFrame(DetectSensitiveData_node)
```

```
    Job.commit()
  }
}
```

AWS CustomEntityType PII タイプを使用した機密データの検出

カスタムエンティティは AWS Studio を使用して定義できますが、AWS Studio 外でこの機能を使用するには、まずカスタムエンティティタイプを定義してから、定義されたカスタムエンティティタイプを entityTypesToDetect のリストに追加する必要があります。

データ内に特定の機密データタイプ (「Employee Id」など) がある場合は、CreateCustomEntityType( ) API を呼び出してカスタムエンティティを作成することができます。以下の例では、リクエストパラメータを使用して CreateCustomEntityType( ) API にカスタムエンティティタイプ「EMPLOYEE_ID」を定義します。

```
{
    "name": "EMPLOYEE_ID",
    "regexString": "\d{4}-\d{3}",
    "contextWords": ["employee"]
}
```

次に、カスタムエンティティタイプ (EMPLOYEE_ID) をEntityDetector( ) API に追加することで、新しいカスタム機密データタイプを使用するようにジョブを変更します。

```
import com.amazonaws.services.glue.GlueContext
import com.amazonaws.services.glue.MappingSpec
import com.amazonaws.services.glue.errors.CallSite
import com.amazonaws.services.glue.util.GlueArgParser
import com.amazonaws.services.glue.util.Job
import com.amazonaws.services.glue.util.JsonOptions
import org.apache.spark.SparkContext
import scala.collection.JavaConverters._
import com.amazonaws.services.glue.ml.EntityDetector

object GlueApp {
  def main(sysArgs: Array[String]) {
    val spark: SparkContext = new SparkContext()
```

```
      val glueContext: GlueContext = new GlueContext(spark)
      val args = GlueArgParser.getResolvedOptions(sysArgs, Seq("JOB_NAME").toArray)
      Job.init(args("JOB_NAME"), glueContext, args.asJava)
      val frame=
glueContext.getSourceWithFormat(formatOptions=JsonOptions("""{"quoteChar": "\"",
"withHeader": true, "separator": ","}"""), connectionType="s3", format="csv",
options=JsonOptions("""{"paths": ["s3://pathToSource"], "recurse": true}"""),
transformationContext="AmazonS3_node1650160158526").getDynamicFrame()

      val frameWithDetectedPII = EntityDetector.detect(frame, Seq("EMAIL",
"CREDIT_CARD", "EMPLOYEE_ID"))

      glueContext.getSinkWithFormat(connectionType="s3",
options=JsonOptions("""{"path": "s3://pathToOutput/", "partitionKeys": []}"""),
transformationContext="someCtx",
format="json").writeDynamicFrame(frameWithDetectedPII)

      Job.commit()
    }
 }
```

> **Note**
>
> カスタム機密データタイプが既存マネージドエンティティタイプと同じ名前で定義されている場合は、カスタム機密データタイプが優先され、マネージドエンティティタイプのロジックが上書きされます。

detect() を使用する検出パラメータ

このメソッドは、DynamicFrame 内のエンティティを検出するために使用されます。元の値を含む新しい DataFrame と、PII 検出メタデータを含む追加の列の outputColumnName を返します。この DynamicFrame が AWS Glue スクリプト内で返された後にカスタムマスキングを実行することも、詳細なアクション API を備えた detect() を代わりに使用することもできます。

```
detect(frame: DynamicFrame,
       entityTypesToDetect: Seq[String],
       outputColumnName: String = "DetectedEntities",
       detectionSensitivity: String = "LOW"): DynamicFrame
```

[パラメータ:]

- frame – (タイプ: DynamicFrame) 処理されるデータを含む入力 DynamicFrame。
- entityTypesToDetect – (タイプ: [Seq[String]) 検出するエンティティタイプのリスト。マネージドエンティティタイプまたはカスタムエンティティタイプのいずれかになります。
- outputColumnName – (タイプ: String、デフォルト:「DetectedEntities」) 検出されたエンティティが保存される列の名前。指定しない場合、デフォルトの列名は「DetectedEntities」となります。
- detectionSensitivity – (タイプ: String、オプション: [低] または [高]、デフォルト: [低]) 検出プロセスの感度を指定します。有効なオプションは [低] または [高] です。指定しない場合、デフォルトの感度は [低] に設定されます。

outputColumnName の設定:

検出されたエンティティが保存される列の名前。指定しない場合、デフォルトの列名は「DetectedEntities」となります。出力列の各行について、補足列には、次の key-value ペアを含む、検出されたエンティティメタデータに対する列名のマップが含まれます。

- entityType – 検出されたエンティティタイプ。
- start – 元のデータ内の検出されたエンティティの開始場所。
- end – 元のデータ内の検出されたエンティティの終了場所。
- actionUsed – 検出されたエンティティに対して実行されたアクション (例:「DETECT」、「REDACT」、「PARTIAL_REDACT」、「SHA256_HASH」)。

例:

```
{
   "DetectedEntities":{
      "SSN Col":[
         {
            "entityType":"USA_SSN",
            "actionUsed":"DETECT",
            "start":4,
            "end":15
         }
      ],
      "Random Data col":[
```

```
         {
            "entityType":"BANK_ACCOUNT",
            "actionUsed":"PARTIAL_REDACT",
            "start":4,
            "end":13
         },
         {
            "entityType":"IP_ADDRESS",
            "actionUsed":"REDACT",
            "start":4,
            "end":13
         }
      ]
   }
}
```

詳細なアクションの **detect()** の検出パラメータ

このメソッドは、指定されたパラメータを使用して DynamicFrame 内のエンティティを検出するために使用されます。元の値がマスキングされた機密データに置き換えられた新しい DataFrame と、PII 検出メタデータを含む追加の列 `outputColumnName` が返されます。

```
detect(frame: DynamicFrame,
       detectionParameters: JsonOptions,
       outputColumnName: String = "DetectedEntities",
       detectionSensitivity: String = "LOW"): DynamicFrame
```

[パラメータ:]

- frame – (タイプ: `DynamicFrame`): 処理されるデータを含む入力 DynamicFrame。
- detectionParameters – (タイプ: `JsonOptions`): 検出プロセスのパラメータを指定する JSON オプション。
- outputColumnName – (タイプ: `String`、デフォルト:「DetectedEntities」): 検出されたエンティティが保存される列の名前。指定しない場合、デフォルトの列名は「DetectedEntities」となります。
- detectionSensitivity – (タイプ: `String`、オプション: [低] または [高]、デフォルト: [低]): 検出プロセスの感度を指定します。有効なオプションは [低] または [高] です。指定しない場合、デフォルトの感度は [低] に設定されます。

`detectionParameters` の設定

設定が含まれていない場合は、デフォルト値が使用されます。

- action – (タイプ: `String`、オプション:「DETECT」、「REDACT」、「PARTIAL_REDACT」、「SHA256_HASH」) エンティティに対して実行するアクションを指定します。必須。マスキングを実行するアクション (「DETECT」を除くすべて) は、列ごとに 1 つのアクションしか実行できないことに留意してください。これは、合体したエンティティをマスキングするための予防措置です。
- sourceColumns – (タイプ: `List[String]`、デフォルト: [“*”]) エンティティの検出を実行するソース列名のリスト。存在しない場合は、デフォルトで [“*”] に設定されます。無効な列名が使用された場合に `IllegalArgumentException` を発生させます。
- sourceColumnsToExclude – (タイプ: `List[String]`) エンティティの検出を実行するソース列名のリスト。`sourceColumns` または `sourceColumnsToExclude` のいずれかを使用します。無効な列名が使用された場合に `IllegalArgumentException` を発生させます。
- actionOptions – 指定されたアクションに基づく追加オプション:
  - 「DETECT」および「SHA256_HASH」の場合、いかなるオプションも使用できません。
  - 「REDACT」の場合:
    - redactText – (タイプ: `String`、デフォルト:「*****」) 検出されたエンティティを置き換えるテキスト。
  - 「PARTIAL_REDACT」の場合:
    - redactChar – (タイプ: `String`、デフォルト:「*」) エンティティ内で検出された各文字を置き換える文字。
    - matchPattern – (タイプ: `String`) 部分的なマスキングの正規表現パターン。numLeftCharsToExclude または `numRightCharsToExclude` と組み合わせることはできません。
    - numLeftCharsToExclude – (タイプ: `String, integer`) 除外する左側の文字の数。matchPattern と組み合わせることはできませんが、`numRightCharsToExclude` と組み合わせて使用することはできます。
    - numRightCharsToExclude – (タイプ: `String, integer`) 除外する右側の文字の数。matchPattern と組み合わせることはできませんが、`numRightCharsToExclude` と組み合わせて使用することはできます。

`outputColumnName` の設定

outputColumnName の設定を参照してください

`classifyColumns()` の検出パラメータ

このメソッドは、DynamicFrame 内のエンティティを検出するために使用されます。キーが列名、値が検出されたエンティティタイプのリストであるマップを返します。これが AWS Glue スクリプト内で返された後、カスタムマスキングを実行できます。

```
classifyColumns(frame: DynamicFrame,
                entityTypesToDetect: Seq[String],
                sampleFraction: Double = 0.1,
                thresholdFraction: Double = 0.1,
                detectionSensitivity: String = "LOW")
```

[パラメータ:]

- frame – (タイプ: `DynamicFrame`) 処理されるデータを含む入力 DynamicFrame。
- entityTypesToDetect – (タイプ: `Seq[String]`) 検出するエンティティタイプのリスト。マネージドエンティティタイプまたはカスタムエンティティタイプのいずれかになります。
- sampleFraction – (タイプ: `Double`、デフォルト: 10%) PII エンティティをスキャンする際にサンプリングするデータの割合。
- thresholdFraction – (タイプ: `Double`、デフォルト: 10%): 列が PII データとして識別されるために満たす必要があるデータの割合。
- detectionSensitivity – (タイプ: `String`、オプション: [低] または [高]、デフォルト: [低]) 検出プロセスの感度を指定します。有効なオプションは [低] または [高] です。指定しない場合、デフォルトの感度は [低] に設定されます。

管理対象の機密データタイプ

グローバルエンティティ

データタイプ	カテゴリ	説明
PERSON_NAME	Universal	その人物の名前
EMAIL	個人	E メールアドレス

データタイプ	カテゴリ	説明
IP_ADDRESS	[コンピュータ]	IP アドレス
MAC_ADDRESS	個人	MAC アドレス

米国のデータタイプ

データタイプ	説明
BANK_ACCOUNT	銀行口座番号。国や地域に固有のものではありませんが、米国とカナダの口座形式のみが検出されます。
CREDIT_CARD	クレジットカード番号。
PHONE_NUMBER	電話番号。国や地域に固有のものではありませんが、現時点では米国とカナダの電話番号のみが検出されます。
USA_ATIN	内国歳入庁 (IRS、Internal Revenue Service) が発行する米国養子縁組納税者番号 (ATIN、Adoption Taxpayer Identification Number)。
USA_CPT_CODE	CPT コード (米国のみ)。
USA_DEA_NUMBER	DEA 番号 (米国のみ)。
USA_DRIVING_LICENSE	運転免許証番号 (米国のみ)。
USA_HCPCS_CODE	HCPCS コード (米国のみ)。
USA_HEALTH_INSURANCE_CLAIM_NUMBER	健康保険請求番号 (米国のみ)。
USA_ITIN	ITIN (米国人または法人の場合)。
USA_MEDICARE_BENEFICIARY_IDENTIFIER	メディケア受益者識別子 (米国のみ)。

データタイプ	説明
USA_NATIONAL_DRUG_CODE	NDC コード (米国のみ)。
USA_NATIONAL_PROVIDER_IDENTIFIER	米国医療従事者識別子番号 (米国のみ)。
USA_PASSPORT_NUMBER	パスポート番号 (米国人の場合)。
USA_PTIN	内国歳入庁が発行する米国申告書作成者番号 (PTIN、Preparer Tax Identification Number)。
USA_SSN	社会保障番号 (米国人の場合)。

アルゼンチンのデータタイプ

データタイプ	説明
ARGENTINA_TAX_IDENTIFICATION_NUMBER	アルゼンチンの納税者番号。CUIT または CUIL とも呼ばれます。

オーストラリアのデータタイプ

データタイプ	説明
AUSTRALIA_BUSINESS_NUMBER	オーストラリア事業者番号 (ABN、Australia Business Number)。オーストラリア事業者登記所 (ABR、Australian Business Register) が発行する固有の識別番号で、政府や地域社会が事業者を識別します。
AUSTRALIA_COMPANY_NUMBER	オーストラリア企業番号 (ACN、Australia Company Number)。オーストラリア証券投資委員会 (ASIC、Australian Securities and Investments Commission) が発行する固有の識別番号。

データタイプ	説明
AUSTRALIA_DRIVING_LICENSE	オーストラリアの運転免許証番号。
AUSTRALIA_MEDICARE_NUMBER	オーストラリアの健康保険番号。オーストラリア健康保険委員会 (Australian Health Insurance Commission) が発行する個人識別番号。
AUSTRALIA_PASSPORT_NUMBER	オーストラリアのパスポート番号。
AUSTRALIA_TAX_FILE_NUMBER	オーストラリアのタックスファイルナンバー (TFN、Tax File Number)。オーストラリア税務局 (ATO、Australian Taxation Office) が納税者 (個人、会社など) に税務取引を行うために発行します。

オーストリアのデータタイプ

データタイプ	説明
AUSTRIA_DRIVING_LICENSE	運転免許証番号 (オーストリアのみ)。
AUSTRIA_PASSPORT_NUMBER	パスポート番号 (オーストリアのみ)。
AUSTRIA_SSN	社会保障番号 (オーストリア人の場合)。
AUSTRIA_TAX_IDENTIFICATION_NUMBER	納税者番号 (オーストリアのみ)。
AUSTRIA_VALUE_ADDED_TAX	付加価値税 (オーストリアのみ)。

バルカン諸国のデータタイプ

データタイプ	説明
BOSNIA_UNIQUE_MASTER_CITIZEN_NUMBER	ボスニア・ヘルツェゴビナ市民の固有マスター市民番号 (JMBG、Unique master citizen number)。

データタイプ	説明
KOSOVO_UNIQUE_MASTER_CITIZEN_NUMBER	コソボの固有マスター市民番号 (JMBG)。
MACEDONIA_UNIQUE_MASTER_CITIZEN_NUMBER	マケドニアの固有マスター市民番号。
MONTENEGRO_UNIQUE_MASTER_CITIZEN_NUMBER	モンテネグロの固有マスター市民番号 (JMBG)。
SERBIA_UNIQUE_MASTER_CITIZEN_NUMBER	セルビアの固有マスター市民番号 (JMBG)。
SERBIA_VALUE_ADDED_TAX	付加価値税 (セルビアのみ)。
VOJVODINA_UNIQUE_MASTER_CITIZEN_NUMBER	ヴォイヴォディナの固有マスター市民番号 (JMBG)。

ベルギーのデータタイプ

データタイプ	説明
BELGIUM_DRIVING_LICENSE	運転免許証番号 (ベルギーのみ)。
BELGIUM_NATIONAL_IDENTIFICATION_NUMBER	ベルギー国民番号 (BNN、Belgian National Number)。
BELGIUM_PASSPORT_NUMBER	パスポート番号 (ベルギーのみ)。
BELGIUM_TAX_IDENTIFICATION_NUMBER	納税者番号 (ベルギーのみ)。
BELGIUM_VALUE_ADDED_TAX	付加価値税 (ベルギーのみ)。

ブラジルのデータタイプ

データタイプ	説明
BRAZIL_BANK_ACCOUNT	銀行口座番号 (ブラジルのみ)。
BRAZIL_NATIONAL_IDENTIFICATION_NUMBER	国民識別番号 (ブラジルのみ)。
BRAZIL_NATIONAL_REGISTRY_OF_LEGAL_ENTITIES_NUMBER	企業に発行される識別番号 (ブラジルのみ)。CNPJ とも呼ばれます。
BRAZIL_NATURAL_PERSON_REGISTRY_NUMBER	CPF とも呼ばれる自然人登録番号 (Natural Person Registry Number)。

ブルガリアのデータタイプ

データタイプ	説明
BULGARIA_DRIVING_LICENSE	運転免許証番号 (ブルガリアのみ)。
BULGARIA_UNIFORM_CIVIL_NUMBER	国民識別番号として機能する統一市民番号 (EGN、Unified Civil Number)。
BULGARIA_VALUE_ADDED_TAX	付加価値税 (ブルガリアのみ)。

カナダのデータタイプ

データタイプ	説明
CANADA_DRIVING_LICENSE	運転免許証番号 (カナダのみ)。
CANADA_GOVERNMENT_IDENTIFICATION_CARD_NUMBER	国民識別番号 (カナダのみ)。
CANADA_PASSPORT_NUMBER	パスポート番号 (カナダのみ)。
CANADA_PERMANENT_RESIDENCE_NUMBER	永住者番号 (PR カード番号)。

データタイプ	説明
CANADA_PERSONAL_HEALTH_NUMBER	ヘルスケアの固有識別番号 (PHN 番号)。
CANADA_SOCIAL_INSURANCE_NUMBER	カナダの社会保険番号 (SIN)。

チリのデータタイプ

データタイプ	説明
CHILE_DRIVING_LICENSE	運転免許証番号 (チリのみ)。
CHILE_NATIONAL_IDENTIFICATION_NUMBER	チリの国民識別番号。RUT または RUN とも呼ばれます。

中国、香港、マカオ、台湾のデータタイプ

データタイプ	説明
CHINA_IDENTIFICATION	中国の識別子。
CHINA_LICENSE_PLATE_NUMBER	運転免許証番号 (中国のみ)。
CHINA_MAINLAND_TRAVEL_PERMIT_ID_HONG_KONG_MACAU	香港とマカオの居住者向けの本土旅行許可証。
CHINA_MAINLAND_TRAVEL_PERMIT_ID_TAIWAN	中華人民共和国 (PRC) 政府が発行する台湾居住者向けの本土旅行許可証。
CHINA_PASSPORT_NUMBER	パスポート番号 (中国のみ)。
CHINA_PHONE_NUMBER	電話番号 (中国のみ)。
HONG_KONG_IDENTITY_CARD	香港入国管理局が発行する公式の身分証明書。
MACAU_RESIDENT_IDENTITY_CARD	マカオ居住者 ID カード (BIR) は、マカオの ID サービス局 (Identification Services Bureau) が発行する公式の ID カードです。

データタイプ	説明
TAIWAN_NATIONAL_IDENTIFICATION_NUMBER	国民識別番号 (台湾のみ)。
TAIWAN_PASSPORT_NUMBER	パスポート番号 (台湾のみ)。

コロンビアのデータタイプ

データタイプ	説明
COLOMBIA_PERSONAL_IDENTIFICATION_NUMBER	出生時にコロンビア人に割り当てられる固有の識別番号。
COLOMBIA_TAX_IDENTIFICATION_NUMBER	納税者番号 (コロンビアのみ)。

クロアチアのデータタイプ

データタイプ	説明
CROATIA_DRIVING_LICENSE	運転免許証番号 (クロアチアのみ)。
CROATIA_IDENTITY_NUMBER	国民識別番号 (クロアチアのみ)。
CROATIA_PASSPORT_NUMBER	パスポート番号 (クロアチアのみ)。
CROATIA_PERSONAL_IDENTIFICATION_NUMBER	個人識別番号 (OIB)。

キプロスのデータタイプ

データタイプ	説明
CYPRUS_DRIVING_LICENSE	運転免許証番号 (キプロスのみ)。

データタイプ	説明
CYPRUS_NATIONAL_IDENTIFICATION_NUMBER	キプロスの身分証明書。
CYPRUS_PASSPORT_NUMBER	パスポート番号 (キプロスのみ)。
CYPRUS_TAX_IDENTIFICATION_NUMBER	納税者番号 (キプロスのみ)。
CYPRUS_VALUE_ADDED_TAX	付加価値税 (キプロスのみ)。

チェコのデータタイプ

データタイプ	説明
CZECHIA_DRIVING_LICENSE	運転免許証番号 (チェコのみ)。
CZECHIA_PERSONAL_IDENTIFICATION_NUMBER	個人識別番号 (チェコのみ)。
CZECHIA_VALUE_ADDED_TAX	付加価値税 (チェコのみ)。

デンマークのデータタイプ

データタイプ	説明
DENMARK_DRIVING_LICENSE	運転免許証番号 (デンマークのみ)。
DENMARK_PERSONAL_IDENTIFICATION_NUMBER	個人識別番号 (デンマークのみ)。
DENMARK_TAX_IDENTIFICATION_NUMBER	納税者番号 (デンマークのみ)。
DENMARK_VALUE_ADDED_TAX	付加価値税 (デンマークのみ)。

エストニアのデータタイプ

データタイプ	説明
ESTONIA_DRIVING_LICENSE	運転免許証番号 (エストニアのみ)。
ESTONIA_PASSPORT_NUMBER	パスポート番号 (エストニアのみ)。
ESTONIA_PERSONAL_IDENTIFICATION_CODE	個人識別番号 (エストニアのみ)。
ESTONIA_VALUE_ADDED_TAX	付加価値税 (エストニアのみ)。

フィンランドのデータタイプ

データタイプ	説明
FINLAND_DRIVING_LICENSE	運転免許証番号 (フィンランドのみ)。
FINLAND_HEALTH_INSURANCE_NUMBER	健康保険番号 (フィンランドのみ)。
FINLAND_NATIONAL_IDENTIFICATION_NUMBER	国民識別番号 (フィンランドのみ)。
FINLAND_PASSPORT_NUMBER	パスポート番号 (フィンランドのみ)。
FINLAND_VALUE_ADDED_TAX	付加価値税 (フィンランドのみ)。

フランスのデータタイプ

データタイプ	説明
FRANCE_BANK_ACCOUNT	銀行口座番号 (フランスのみ)。
FRANCE_DRIVING_LICENSE	運転免許証番号 (フランスのみ)。
FRANCE_HEALTH_INSURANCE_NUMBER	フランスの健康保険番号。
FRANCE_INSEE_CODE	フランスの社会保障 (SSN または NIR) 番号。

データタイプ	説明
FRANCE_NATIONAL_IDENTIFICATION_NUMBER	フランス国民識別番号 (CNI)。
FRANCE_PASSPORT_NUMBER	パスポート番号 (フランスのみ)。
FRANCE_TAX_IDENTIFICATION_NUMBER	納税者番号 (フランスのみ)。
FRANCE_VALUE_ADDED_TAX	付加価値税 (フランスのみ)。

ドイツのデータタイプ

データタイプ	説明
GERMANY_BANK_ACCOUNT	銀行口座番号 (ドイツのみ)。
GERMANY_DRIVING_LICENSE	運転免許証番号 (ドイツのみ)。
GERMANY_PASSPORT_NUMBER	パスポート番号 (ドイツのみ)。
GERMANY_PERSONAL_IDENTIFICATION_NUMBER	個人識別番号 (ドイツのみ)。
GERMANY_TAX_IDENTIFICATION_NUMBER	納税者番号 (ドイツのみ)。
GERMANY_VALUE_ADDED_TAX	付加価値税 (ドイツのみ)。

ギリシャのデータタイプ

データタイプ	説明
GREECE_DRIVING_LICENSE	運転免許証番号 (ギリシャのみ)。
GREECE_PASSPORT_NUMBER	パスポート番号 (ギリシャのみ)。
GREECE_SSN	社会保障番号 (ギリシャ人の場合)。

データタイプ	説明
GREECE_TAX_IDENTIFICATION_NUMBER	納税者番号 (ギリシャのみ)。
GREECE_VALUE_ADDED_TAX	付加価値税 (ギリシャのみ)。

ハンガリーのデータタイプ

データタイプ	説明
HUNGARY_DRIVING_LICENSE	運転免許証番号 (ハンガリーのみ)。
HUNGARY_PASSPORT_NUMBER	パスポート番号 (ハンガリーのみ)。
HUNGARY_SSN	社会保障番号 (ハンガリー人の場合)。
HUNGARY_TAX_IDENTIFICATION_NUMBER	納税者番号 (ハンガリーのみ)。
HUNGARY_VALUE_ADDED_TAX	付加価値税 (ハンガリーのみ)。

アイスランドのデータタイプ

データタイプ	説明
ICELAND_NATIONAL_IDENTIFICATION_NUMBER	国民識別番号 (アイスランドのみ)。
ICELAND_PASSPORT_NUMBER	パスポート番号 (アイスランドのみ)。
ICELAND_VALUE_ADDED_TAX	付加価値税 (アイスランドのみ)。

インドのデータタイプ

データタイプ	説明
INDIA_AADHAAR_NUMBER	インドの固有 ID 局 (Unique Identification Authority) が発行する Aadhaar 識別番号。

データタイプ	説明
INDIA_PERMANENT_ACCOUNT_NUMBER	インドの納税者番号 (PAN、Permanent Account Number)。

インドネシアのデータタイプ

データタイプ	説明
INDONESIA_IDENTITY_CARD_NUMBER	国民識別番号 (インドネシアのみ)。

アイルランドのデータタイプ

データタイプ	説明
IRELAND_DRIVING_LICENSE	運転免許証番号 (アイルランドのみ)。
IRELAND_PASSPORT_NUMBER	パスポート番号 (アイルランドのみ)。
IRELAND_PERSONAL_PUBLIC_SERVICE_NUMBER	アイルランド個人公共サービス (PPS、Personal Public Service) 番号。
IRELAND_TAX_IDENTIFICATION_NUMBER	納税者番号 (アイルランドのみ)。
IRELAND_VALUE_ADDED_TAX	付加価値税 (アイルランドのみ)。

イスラエルのデータタイプ

データタイプ	説明
ISRAEL_IDENTIFICATION_NUMBER	国民識別番号 (イスラエルのみ)。

イタリアのデータタイプ

データタイプ	説明
ITALY_BANK_ACCOUNT	銀行口座番号 (イタリアのみ)。
ITALY_DRIVING_LICENSE	運転免許証番号 (イタリアのみ)。
ITALY_FISCAL_CODE	識別番号。Italian Codice Fiscale (イタリアの納税者コード) としても知られています。
ITALY_PASSPORT_NUMBER	パスポート番号 (イタリアのみ)。
ITALY_VALUE_ADDED_TAX	付加価値税 (イタリアのみ)。

日本のデータタイプ

データタイプ	説明
JAPAN_BANK_ACCOUNT	日本の銀行口座。
JAPAN_DRIVING_LICENSE	日本の運転免許証番号。
JAPAN_MY_NUMBER	国税行政、年金行政、災害対応に使用される日本の国民または法人のための固有の識別子
JAPAN_PASSPORT_NUMBER	日本のパスポート番号。

韓国のデータタイプ

データタイプ	説明
KOREA_PASSPORT_NUMBER	パスポート番号 (韓国のみ)。
KOREA_RESIDENCE_REGISTRATION_NUMBER_FOR_CITIZENS	居住者の韓国居住登録者番号。
KOREA_RESIDENCE_REGISTRATION_NUMBER_FOR_FOREIGNERS	外国人の韓国居住登録者番号。

ラトビアのデータタイプ

データタイプ	説明
LATVIA_DRIVING_LICENSE	運転免許証番号 (ラトビアのみ)。
LATVIA_PASSPORT_NUMBER	パスポート番号 (ラトビアのみ)。
LATVIA_PERSONAL_IDENTIFICATION_NUMBER	個人識別番号 (ラトビアのみ)。
LATVIA_VALUE_ADDED_TAX	付加価値税 (ラトビアのみ)。

リヒテンシュタインのデータタイプ

データタイプ	説明
LIECHTENSTEIN_NATIONAL_IDENTIFICATION_NUMBER	国民識別番号 (リヒテンシュタインのみ)。
LIECHTENSTEIN_PASSPORT_NUMBER	パスポート番号 (リヒテンシュタインのみ)。
LIECHTENSTEIN_TAX_IDENTIFICATION_NUMBER	納税者番号 (リヒテンシュタインのみ)。

リトアニアのデータタイプ

データタイプ	説明
LITHUANIA_DRIVING_LICENSE	運転免許証番号 (リトアニアのみ)。
LITHUANIA_PERSONAL_IDENTIFICATION_NUMBER	個人識別番号 (リトアニアのみ)。
LITHUANIA_TAX_IDENTIFICATION_NUMBER	納税者番号 (リトアニアのみ)。
LITHUANIA_VALUE_ADDED_TAX	付加価値税 (リトアニアのみ)。

ルクセンブルクのデータタイプ

データタイプ	説明
LUXEMBOURG_DRIVING_LICENSE	運転免許証番号 (ルクセンブルクのみ)。
LUXEMBOURG_NATIONAL_INDIVIDUAL_NUMBER	国民識別番号 (ルクセンブルクのみ)。
LUXEMBOURG_PASSPORT_NUMBER	パスポート番号 (ルクセンブルクのみ)。
LUXEMBOURG_TAX_IDENTIFICATION_NUMBER	納税者番号 (ルクセンブルクのみ)。
LUXEMBOURG_VALUE_ADDED_TAX	付加価値税 (ルクセンブルクのみ)。

マレーシアのデータタイプ

データタイプ	説明
MALAYSIA_MYKAD_NUMBER	国民識別番号 (マレーシアのみ)。
MALAYSIA_PASSPORT_NUMBER	パスポート番号 (マレーシアのみ)。

マルタのデータタイプ

データタイプ	説明
MALTA_DRIVING_LICENSE	運転免許証番号 (マルタのみ)。
MALTA_NATIONAL_IDENTIFICATION_NUMBER	国民識別番号 (マルタのみ)。
MALTA_TAX_IDENTIFICATION_NUMBER	納税者番号 (マルタのみ)。
MALTA_VALUE_ADDED_TAX	付加価値税 (マルタのみ)。

メキシコのデータタイプ

データタイプ	説明
MEXICO_CLABE_NUMBER	メキシコ CLABE (Clave Bancaria Estandarizada) 銀行番号。
MEXICO_DRIVING_LICENSE	運転免許証番号 (メキシコのみ)。
MEXICO_PASSPORT_NUMBER	パスポート番号 (メキシコのみ)。
MEXICO_TAX_IDENTIFICATION_NUMBER	納税者番号 (メキシコのみ)。
MEXICO_UNIQUE_POPULATION_REGISTRY_CODE	メキシコの Clave Única de Registro de Población (CURP) 固有識別コード。

オランダのデータタイプ

データタイプ	説明
NETHERLANDS_CITIZEN_SERVICE_NUMBER	オランダ市民番号 (BSN、burgerservicenummer)。
NETHERLANDS_DRIVING_LICENSE	運転免許証番号 (オランダのみ)。
NETHERLANDS_PASSPORT_NUMBER	パスポート番号 (オランダのみ)。
NETHERLANDS_TAX_IDENTIFICATION_NUMBER	納税者番号 (オランダのみ)。
NETHERLANDS_VALUE_ADDED_TAX	付加価値税 (オランダのみ)。
NETHERLANDS_BANK_ACCOUNT	銀行口座番号 (オランダのみ)。

ニュージーランドのデータタイプ

データタイプ	説明
NEW_ZEALAND_DRIVING_LICENSE	運転免許証番号 (ニュージーランドのみ)。
NEW_ZEALAND_NATIONAL_HEALTH_INDEX_NUMBER	ニュージーランドの国民健康指標番号 (National Health Index Number)。
NEW_ZEALAND_TAX_IDENTIFICATION_NUMBER	納税者番号。内国歳入番号 (Inland Revenue Number) とも呼ばれます (ニュージーランドのみ)。

ノルウェーのデータタイプ

データタイプ	説明
NORWAY_BIRTH_NUMBER	ノルウェーの国民識別番号。
NORWAY_DRIVING_LICENSE	運転免許証番号 (ノルウェーのみ)。
NORWAY_HEALTH_INSURANCE_NUMBER	ノルウェーの健康保険番号。
NORWAY_NATIONAL_IDENTIFICATION_NUMBER	国民識別番号 (ノルウェーのみ)。
NORWAY_VALUE_ADDED_TAX	付加価値税 (ノルウェーのみ)。

フィリピンのデータタイプ

データタイプ	説明
PHILIPPINES_DRIVING_LICENSE	運転免許証番号 (フィリピンのみ)。
PHILIPPINES_PASSPORT_NUMBER	パスポート番号 (フィリピンのみ)。

ポーランドのデータタイプ

データタイプ	説明
POLAND_DRIVING_LICENSE	運転免許証番号 (ポーランドのみ)。
POLAND_IDENTIFICATION_NUMBER	ポーランドの識別番号。
POLAND_PASSPORT_NUMBER	パスポート番号 (ポーランドのみ)。
POLAND_REGON_NUMBER	REGON 識別番号。統計識別番号 (Statistical Identification Number) とも呼ばれます。
POLAND_SSN	社会保障番号 (ポーランド人の場合)。
POLAND_TAX_IDENTIFICATION_NUMBER	納税者番号 (ポーランドのみ)。
POLAND_VALUE_ADDED_TAX	付加価値税 (ポーランドのみ)。

ポルトガルのデータタイプ

データタイプ	説明
PORTUGAL_DRIVING_LICENSE	運転免許証番号 (ポルトガルのみ)。
PORTUGAL_NATIONAL_IDENTIFICATION_NUMBER	国民識別番号 (ポルトガルのみ)。
PORTUGAL_PASSPORT_NUMBER	パスポート番号 (ポルトガルのみ)。
PORTUGAL_TAX_IDENTIFICATION_NUMBER	納税者番号 (ポルトガルのみ)。
PORTUGAL_VALUE_ADDED_TAX	付加価値税 (ポルトガルのみ)。

ルーマニアのデータタイプ

データタイプ	説明
ROMANIA_DRIVING_LICENSE	運転免許証番号 (ルーマニアのみ)。

データタイプ	説明
ROMANIA_NUMERICAL_PERSONAL_CODE	個人識別番号 (ルーマニアのみ)。
ROMANIA_PASSPORT_NUMBER	パスポート番号 (ルーマニアのみ)。
ROMANIA_VALUE_ADDED_TAX	付加価値税 (ルーマニアのみ)。

シンガポールのデータタイプ

データタイプ	説明
SINGAPORE_DRIVING_LICENSE	運転免許証番号 (シンガポールのみ)。
SINGAPORE_NATIONAL_REGISTRY_IDENTIFICATION_NUMBER	シンガポールの国民登録 ID カード。
SINGAPORE_PASSPORT_NUMBER	パスポート番号 (シンガポールのみ)。
SINGAPORE_UNIQUE_ENTITY_NUMBER	シンガポールの個別企業登録番号 (UEN)。

スロバキアのデータタイプ

データタイプ	説明
SLOVAKIA_DRIVING_LICENSE	運転免許証番号 (スロバキアのみ)。
SLOVAKIA_NATIONAL_IDENTIFICATION_NUMBER	国民識別番号 (スロバキアのみ)。
SLOVAKIA_PASSPORT_NUMBER	パスポート番号 (スロバキアのみ)。
SLOVAKIA_VALUE_ADDED_TAX	付加価値税 (スロバキアのみ)。

スロベニアのデータタイプ

データタイプ	説明
SLOVENIA_DRIVING_LICENSE	運転免許証番号 (スロベニアのみ)。
SLOVENIA_PASSPORT_NUMBER	パスポート番号 (スロベニアのみ)。
SLOVENIA_TAX_IDENTIFICATION_NUMBER	納税者番号 (スロベニアのみ)。
SLOVENIA_UNIQUE_MASTER_CITIZEN_NUMBER	スロベニア市民の固有マスター市民番号 (JMBG、Unique master citizen number)。
SLOVENIA_VALUE_ADDED_TAX	付加価値税 (スロベニアのみ)。

南アフリカのデータタイプ

データタイプ	説明
SOUTH_AFRICA_PERSONAL_IDENTIFICATION_NUMBER	個人識別番号 (南アフリカのみ)。

スペインのデータタイプ

データタイプ	説明
SPAIN_BANK_ACCOUNT	銀行口座番号 (スペインのみ)。
SPAIN_DNI	スペインの国民 ID カード (Documento Nacional de Identidad)。
SPAIN_DRIVING_LICENSE	運転免許証番号 (スペインのみ)。
SPAIN_NIE	NIE とも呼ばれる外国人 ID 番号 (スペインのみ)。
SPAIN_NIF	NIF とも呼ばれる納税者番号 (スペインのみ)。
SPAIN_PASSPORT_NUMBER	パスポート番号 (スペインのみ)。

データタイプ	説明
SPAIN_SSN	社会保障番号 (スペイン人の場合)。
SPAIN_VALUE_ADDED_TAX	付加価値税 (スペインのみ)。

スリランカのデータタイプ

データタイプ	説明
SRI_LANKA_NATIONAL_IDENTIFICATION_NUMBER	国民識別番号 (スリランカのみ)。

スウェーデンのデータタイプ

データタイプ	説明
SWEDEN_DRIVING_LICENSE	運転免許証番号 (スウェーデンのみ)。
SWEDEN_PASSPORT_NUMBER	パスポート番号 (スウェーデンのみ)。
SWEDEN_PERSONAL_IDENTIFICATION_NUMBER	国民識別番号 (スウェーデンのみ)。
SWEDEN_TAX_IDENTIFICATION_NUMBER	スウェーデンの納税者番号 (personnummer)。
SWEDEN_VALUE_ADDED_TAX	付加価値税 (スウェーデンのみ)。

スイスのデータタイプ

データタイプ	説明
SWITZERLAND_AHV	スイス人の社会保障番号 (AHV)。
SWITZERLAND_HEALTH_INSURANCE_NUMBER	スイスの健康保険番号。

データタイプ	説明
SWITZERLAND_PASSPORT_NUMBER	パスポート番号 (スイスのみ)。
SWITZERLAND_VALUE_ADDED_TAX	付加価値税 (スイスのみ)。

タイのデータタイプ

データタイプ	説明
THAILAND_PASSPORT_NUMBER	パスポート番号 (タイのみ)。
THAILAND_PERSONAL_IDENTIFICATION_NUMBER	個人識別番号 (タイのみ)。

トルコのデータタイプ

データタイプ	説明
TURKEY_NATIONAL_IDENTIFICATION_NUMBER	国民識別番号 (トルコのみ)。
TURKEY_PASSPORT_NUMBER	パスポート番号 (トルコのみ)。
TURKEY_VALUE_ADDED_TAX	付加価値税 (トルコのみ)。

ウクライナのデータタイプ

データタイプ	説明
UKRAINE_INDIVIDUAL_IDENTIFICATION_NUMBER	固有識別番号 (ウクライナのみ)。
UKRAINE_PASSPORT_NUMBER_DOMESTIC	国内パスポート番号 (ウクライナのみ)。

データタイプ	説明
UKRAINE_PASSPORT_NUMBER_INTERNATIONAL	国際パスポート番号 (ウクライナのみ)。

アラブ首長国連邦 (UAE) のデータタイプ

データタイプ	説明
UNITED_ARAB_EMIRATES_PERSONAL_NUMBER	個人識別番号 (UAE のみ)。

英国のデータタイプ

データタイプ	説明
UK_BANK_ACCOUNT	英国 (UK) の銀行口座。
UK_BANK_SORT_CODE	英国 (UK) の銀行ソートコード。ソートコードは、各国の銀行間で、それぞれの決済機関を通じて送金をルーティングするために使用される銀行コードです。
UK_DRIVING_LICENSE	グレートブリテンおよび北アイルランド連合王国の運転免許証番号 (英国固有)
UK_ELECTORAL_ROLL_NUMBER	選挙人名簿番号 (ERN) は、英国の選挙登録のために個人に発行される識別番号です。この番号の形式は、英国内閣府の英国政府基準によって定められています。
UK_NATIONAL_HEALTH_SERVICE_NUMBER	国民保険サービス (NHS) 番号は、英国の公衆衛生サービスの登録ユーザーに割り当てられる固有の番号です。
UK_NATIONAL_INSURANCE_NUMBER	国民保険番号 (NINO) は、国民保険プログラムまたは社会保障システムで個人を識別するため

データタイプ	説明
	に英国で使用される番号です。この番号は、NI No または NINO と呼ばれることもあります。
UK_PASSPORT_NUMBER	英国 (UK) パスポート番号。
UK_UNIQUE_TAXPAYER_REFERENCE_NUMBER	英国 (UK) 納税者固有番号 (UTR) 番号。英国政府が課税システムを管理するために使用する識別子。
UK_VALUE_ADDED_TAX	VAT は最終消費者が負担する消費税です。VAT は、製造および流通プロセスの各取引に対して支払われます。英国の場合、VAT 番号は事業が設立された地域の VAT 事務所によって発行されます。
UK_PHONE_NUMBER	英国 (UK) の電話番号。

ベネズエラのデータタイプ

データタイプ	説明
VENEZUELA_DRIVING_LICENSE	運転免許証番号 (ベネズエラのみ)。
VENEZUELA_NATIONAL_IDENTIFICATION_NUMBER	国民識別番号 (ベネズエラのみ)。
VENEZUELA_VALUE_ADDED_TAX	付加価値税 (ベネズエラのみ)。

詳細な機密データ検出の使用

> **Note**
>
> 詳細なアクションは AWS Glue 3.0 および 4.0 でのみ使用できます。これには AWS Glue Studio エクスペリエンスも含まれます。永続的な監査ログの変更も 2.0 では使用できません。

すべての AWS Glue Studio 3.0 および 4.0 のビジュアルジョブでは、詳細なアクション API を自動的に使用するスクリプトが作成されます。

Detect Sensitive Data は、定義したエンティティ、または AWS Glue によって事前定義されたエンティティを検出、マスキング、削除する機能を提供します。さらに、詳細なアクションを使用することで、エンティティごとに特定のアクションを適用できます。その他の利点には次が含まれます。

- データが検出されるとすぐにアクションが適用されるようになり、パフォーマンスが改善されました。
- 特定の列を含めるか、または除外するオプション。
- 部分的なマスキングを使用する機能。これにより、文字列全体をマスキングするのではなく、検出された機密データエンティティを部分的にマスキングできます。オフセットを含むシンプルなパラメータと正規表現の両方がサポートされます。

次のセクションで参照するサンプルジョブで使用される機密データ検出 API と詳細なアクションのコードスニペットを次に示します。

detect API – 詳細なアクションは、新しい `detectionParameters` パラメータを使用します。

```
def detect(
    frame: DynamicFrame,
    detectionParameters: JsonOptions,
    outputColumnName: String = "DetectedEntities",
    detectionSensitivity: String = "LOW"
): DynamicFrame = {}
```

詳細なアクションで機密データ検出 API を使用する

detect を使用する機密データ検出 API は、指定されたデータを分析し、行または列が機密データエンティティタイプであるかどうかを判断して、エンティティタイプごとにユーザーが指定したアクションを実行します。

詳細なアクションでの detect API の使用

detect API を使用して、`outputColumnName` と `detectionParameters` を指定します。

```
    object GlueApp {
```

```
      def main(sysArgs: Array[String]) {

        val spark: SparkContext = new SparkContext()
        val glueContext: GlueContext = new GlueContext(spark)

        // @params: [JOB_NAME]
        val args = GlueArgParser.getResolvedOptions(sysArgs, Seq("JOB_NAME").toArray)
        Job.init(args("JOB_NAME"), glueContext, args.asJava)

        // Script generated for node S3 bucket. Creates DataFrame from data stored in
S3.
        val S3bucket_node1 =
glueContext.getSourceWithFormat(formatOptions=JsonOptions("""{"quoteChar":
"\"", "withHeader": true, "separator": ",", "optimizePerformance": false}"""),
connectionType="s3", format="csv", options=JsonOptions("""{"paths":
["s3://189657479688-ddevansh-pii-test-bucket/tiny_pii.csv"], "recurse": true}"""),
transformationContext="S3bucket_node1").getDynamicFrame()

        // Script generated for node Detect Sensitive Data. Will run detect API for the
DataFrame
        // detectionParameter contains information on which EntityType are being
detected
        // and what actions are being applied to them when detected.
        val DetectSensitiveData_node2 = EntityDetector.detect(
            frame = S3bucket_node1,
            detectionParameters = JsonOptions(
             """
                {
                    "PHONE_NUMBER": [
                        {
                            "action": "PARTIAL_REDACT",
                            "actionOptions": {
                                "numLeftCharsToExclude": "3",
                                "numRightCharsToExclude": "4",
                                "redactChar": "#"
                            },
                            "sourceColumnsToExclude": [ "Passport No", "DL NO#" ]
                        }
                    ],
                    "USA_PASSPORT_NUMBER": [
                        {
                            "action": "SHA256_HASH",
                            "sourceColumns": [ "Passport No" ]
                        }
```

```
                    ],
                    "USA_DRIVING_LICENSE": [
                        {
                            "action": "REDACT",
                            "actionOptions": {
                                "redactText": "USA_DL"
                            },
                            "sourceColumns": [ "DL NO#" ]
                        }
                    ]

                }
            """
            ),
            outputColumnName = "DetectedEntities"
        )

        // Script generated for node S3 bucket. Store Results of detect to S3 location
        val S3bucket_node3 = glueContext.getSinkWithFormat(connectionType="s3",
options=JsonOptions("""{"path": "s3://amzn-s3-demo-bucket/test-output/",
"partitionKeys": []}"""), transformationContext="S3bucket_node3",
format="json").writeDynamicFrame(DetectSensitiveData_node2)

        Job.commit()
      }
```

上記のスクリプトは、Amazon S3 内の場所から DataFrame を作成し、detect API を実行します。detect API ではフィールド detectionParameters (エンティティに使用されるすべてのアクション設定のリストに対するエンティティ名のマップ) が AWS Glue の JsonOptions オブジェクトによって表されるため、API の機能を拡張することもできます。

エンティティごとに指定された各アクションについて、エンティティ/アクションの組み合わせを適用するすべての列名のリストを入力します。これにより、エンティティをカスタマイズしてデータセット内のすべての列を検出したり、特定の列に存在しないことがわかっているエンティティをスキップしたりできます。また、これらのエンティティを呼び出す不要な検出を実行しないことでジョブのパフォーマンスが改善し、各列とエンティティの組み合わせに固有のアクションを実行できるようになります。

detectionParameters を詳しく見てみると、サンプルジョブには 3 つのエンティティタイプがあることがわかります。これらは、Phone

Number、USA_PASSPORT_NUMBER、USA_DRIVING_LICENSE です。これらの各エンティティタイプについて、AWS Glue は、異なるアクション (PARTIAL_REDACT、SHA256_HASH、REDACT、DETECT のいずれか) を実行します。各エンティティタイプには、適用する sourceColumns および/または sourceColumnsToExclude (検出された場合) があります。

> **Note**
>
> 列ごとに使用できるその場編集アクション (PARTIAL_REDACT、SHA256_HASH、または REDACT) は 1 つだけですが、DETECT アクションはこれらのアクションのいずれかと併用できます。

detectionParameters フィールドのレイアウトは以下のとおりです。

```
    ENTITY_NAME -> List[Actions]
    {
     "ENTITY_NAME": [{
      Action, // required
      ColumnSpecs,
      ActionOptionsMap
        }],
        "ENTITY_NAME2": [{
      ...
        }]
    }
```

タイプ actions および actionOptions は以下のとおりです。

```
DETECT
{
    # Required
    "action": "DETECT",
    # Optional, depending on action chosen
    "actionOptions": {
        // There are no actionOptions for DETECT
    },
    # 1 of below required, both can also used
    "sourceColumns": [
```

```
        "COL_1", "COL_2", ..., "COL_N"
    ],
    "sourceColumnsToExclude": [
        "COL_5"
    ]
}

SHA256_HASH
{
    # Required
    "action": "SHA256_HASH",
    # Required or optional, depending on action chosen
    "actionOptions": {
        // There are no actionOptions for SHA256_HASH
    },

    # 1 of below required, both can also used
    "sourceColumns": [
        "COL_1", "COL_2", ..., "COL_N"
    ],
    "sourceColumnsToExclude": [
        "COL_5"
    ]
}

REDACT
{
    # Required
    "action": "REDACT",
    # Required or optional, depending on action chosen
    "actionOptions": {
        // The text that is being replaced
        "redactText": "USA_DL"
    },

    # 1 of below required, both can also used
    "sourceColumns": [
        "COL_1", "COL_2", ..., "COL_N"
    ],
    "sourceColumnsToExclude": [
        "COL_5"
    ]
}
```

```
PARTIAL_REDACT
{
    # Required
    "action": "PARTIAL_REDACT",
    # Required or optional, depending on action chosen
    "actionOptions": {
        // number of characters to not redact from the left side
        "numLeftCharsToExclude": "3",
        // number of characters to not redact from the right side
        "numRightCharsToExclude": "4",
        // the partial redact will be made with this redacted character
        "redactChar": "#",
        // regex pattern for partial redaction
        "matchPattern": "[0-9]"
    },

    # 1 of below required, both can also used
    "sourceColumns": [
        "COL_1", "COL_2", ..., "COL_N"
    ],
    "sourceColumnsToExclude": [
        "COL_5"
    ]
}
```

スクリプトが実行されると、指定された Amazon S3 の場所に結果が出力されます。Amazon S3 でデータを表示することはできますが、選択したアクションに基づいて、選択したエンティティタイプの感度が高くなります。この場合、行は次のようになります。

```
{
    "Name": "Colby Schuster",
    "Address": "39041 Antonietta Vista, South Rodgerside, Nebraska 24151",
    "Car Owned": "Fiat",
    "Email": "Kitty46@gmail.com",
    "Company": "O'Reilly Group",
    "Job Title": "Dynamic Functionality Facilitator",
    "ITIN": "991-22-2906",
    "Username": "Cassandre.Kub43",
    "SSN": "914-22-2906",
    "DOB": "2020-08-27",
    "Phone Number": "1-2#######1718",
    "Bank Account No": "69741187",
```

```
    "Credit Card Number": "6441-6289-6867-2162-2711",
    "Passport No": "94f311e93a623c72ccb6fc46cf5f5b0265ccb42c517498a0f27fd4c43b47111e",
    "DL NO#": "USA_DL"
}
```

上記のスクリプトでは、Phone Number は部分的に # でマスキングされています。Passport No は SHA256 ハッシュに変更されました。DL NO# は米国の運転免許証番号として検出され、detectionParameters で記述されているように「USA_DL」でマスキングされました。

> **Note**
>
> classifyColumns API は、API の性質上、詳細なアクションでは使用できません。この API は、列サンプリング (ユーザーによる調整が可能ですが、デフォルト値が設定されています) を実行して、検出をより迅速に実行します。このため、詳細なアクションでは、すべての値に対してイテレーションを実行する必要があります。

永続的な監査ログ

詳細なアクションで導入された新機能 (ただし、通常の API を使用する場合にも使用可能) は、永続的な監査ログです。現在、detect API を実行すると、PII 検出メタデータを含む追加の列 (DetectedEntities にデフォルト設定されていますが、outputColumnName を通じてカスタマイズ可能です) パラメータが追加されます。これには、「actionUsed」メタデータキーが追加されました。これは、DETECT、PARTIAL_REDACT、SHA256_HASH、REDACT のいずれかです。

```
"DetectedEntities": {
    "Credit Card Number": [
        {
            "entityType": "CREDIT_CARD",
            "actionUsed": "DETECT",
            "start": 0,
            "end": 19
        }
    ],
    "Phone Number": [
        {
            "entityType": "PHONE_NUMBER",
            "actionUsed": "REDACT",
            "start": 0,
```

```
            "end": 14
        }
    ]
}
```

detect(entityTypesToDetect, outputColumnName) などの詳細なアクションなしで API を使用しているお客様でも、結果として得られるデータフレームでこの永続的な監査ログを確認できます。

詳細なアクションが設定されている API を使用しているお客様は、マスキングされているかどうかにかかわらず、すべてのアクションを表示できます。例:

```
+---------------------+---------------
+-------------------------------------------------------------------------------------------------------
+
| Credit Card Number  |  Phone Number  |
                                              DetectedEntities
                                                                    |
+---------------------+---------------
+-------------------------------------------------------------------------------------------------------
+
| 622126741306XXXX    | +12#####7890   | {"Credit Card Number":
[{"entityType":"CREDIT_CARD","actionUsed":"PARTIAL_REDACT","start":0,"end":16}],"Phone
 Number":
[{"entityType":"PHONE_NUMBER","actionUsed":"PARTIAL_REDACT","start":0,"end":12}]}} |
| 6221 2674 1306 XXXX | +12#######7890 | {"Credit Card Number":
[{"entityType":"CREDIT_CARD","actionUsed":"PARTIAL_REDACT","start":0,"end":19}],"Phone
 Number":
[{"entityType":"PHONE_NUMBER","actionUsed":"PARTIAL_REDACT","start":0,"end":14}]}} |
| 6221-2674-1306-XXXX | 22#######7890  | {"Credit Card Number":
[{"entityType":"CREDIT_CARD","actionUsed":"PARTIAL_REDACT","start":0,"end":19}],"Phone
 Number":
[{"entityType":"PHONE_NUMBER","actionUsed":"PARTIAL_REDACT","start":0,"end":14}]}} |
+---------------------+---------------
+-------------------------------------------------------------------------------------------------------
+
```

[DetectedEntities] 列を表示したくない場合、必要なのは、カスタムスクリプトに追加の列をドロップすることだけです。

AWS Glue ビジュアルジョブ API

AWS Glue は ビジュアルステップワークフローを表す JSON オブジェクトから AWS Glue APIを使用して、お客様がデータ統合ジョブを作成できるようにする API を提供します。　お客様は AWS Glue Studio のビジュアルエディターを使って、これらのジョブを処理することができます。

Visual Job API データ型の詳細については、「Visual Job API」(ビジュアル Job API) を参照してください。

トピック

- API 設計と CRUD API
- 使用開始
- ビジュアルジョブの制限

API 設計と CRUD API

CreateJobAPI と UpdateJob の API は、追加のオプションパラメータとして codeGenConfigurationNodes をサポートするようになりました。このフィールドに空でない JSON 構造を指定すると、作成されたジョブの AWS Glue Studio に対して DAG が登録され、関連するコードが生成されます。ジョブ作成時のこのフィールドの NULL 値または空の文字列は無視されます。

codeGenConfigurationNodes フィールドの更新は、CreateJob と同様の方法で UpdateJob AWS Glue API で行われます。DAG が必要に応じて変更された UpdateJob でフィールド全体を指定する必要があります。指定された NULL 値は無視され、DAG への更新は実行されません。空の構造体または文字列を指定すると、CodeGenConfigurationNodes が空に設定され、以前の DAG が削除されます。GetJob API は DAG が存在する場合、DAG を返します。DeleteJob API では、関連する DAG も削除されます。

使用開始

ジョブを作成するには、CreateJob アクションを使用します。`CreateJob` リクエストの入力には、JSON で DAG オブジェクトを指定できる追加のフィールド 「CodeGenConfigurationNodes」があります。

留意すべきこと:

- 「CodeGenConfigurationNodes」フィールドは、ノードへの nodeID のマップです。

- 各ノードは、ノードの種類を識別するキーで始まります。
- ノードには 1 つのタイプしか指定できないため、キーは 1 つしか指定できません。
- 入力フィールドには、現在のノードの親ノードが含まれます。

以下は、CreateJob 入力の JSON 表現です。

```
{
  "node-1": {
    "S3CatalogSource": {
      "Table": "csvFormattedTable",
      "PartitionPredicate": "",
      "Name": "S3 bucket",
      "AdditionalOptions": {},
      "Database": "myDatabase"
    }
  },
  "node-3": {
    "S3DirectTarget": {
      "Inputs": ["node-2"],
      "PartitionKeys": [],
      "Compression": "none",
      "Format": "json",
      "SchemaChangePolicy": { "EnableUpdateCatalog": false },
      "Path": "",
      "Name": "S3 bucket"
    }
  },
  "node-2": {
    "ApplyMapping": {
      "Inputs": ["node-1"],
      "Name": "ApplyMapping",
      "Mapping": [
        {
          "FromType": "long",
          "ToType": "long",
          "Dropped": false,
          "ToKey": "myheader1",
          "FromPath": ["myheader1"]
        },
        {
          "FromType": "long",
          "ToType": "long",
```

```
          "Dropped": false,
          "ToKey": "myheader2",
          "FromPath": ["myheader2"]
        },
        {
          "FromType": "long",
          "ToType": "long",
          "Dropped": false,
          "ToKey": "myheader3",
          "FromPath": ["myheader3"]
        }
      ]
    }
  }
}
```

ジョブの更新と取得

[Updating jobs] (ジョブの更新) に 'codeGenConfigurationNodes'フィールドがあるため、入力形式は同じになります。「UpdateJob」アクションを参照してください。

GetJob アクションは、同じ形式の「CodeGenConfigurationNodes」フィールドも返します。「GetJob」アクションを参照してください。

ビジュアルジョブの制限

既存の API に「CodeGenConfigurationNodes」パラメータが追加されているため、これらの API の制限はすべて継承されます。さらに、CodeGenConfigurationNodes および一部のノードのサイズは制限されます。詳細については、次の「ジョブ構造」を参照してください。

Ray スクリプトのプログラミング

AWS Glue を使用すると、Ray スクリプトの記述と実行が簡単になります。このセクションでは、AWS Glue for Ray でサポートされている Ray の機能について説明します。Ray スクリプトは、Python でプログラムします。

カスタムスクリプトは、ジョブ定義内の `Runtime` フィールドで定義されている Ray のバージョンと互換性がある必要があります。Jobs API の `Runtime` の詳細については、「the section called “ジョブ”」を参照してください。各ランタイム環境の詳細については、「the section called “サポートされている Ray のランタイム環境”」を参照してください。

トピック

- チュートリアル: AWS Glue for Ray で ETL スクリプトを記述する
- AWS Glue for Ray での Ray Core と Ray Data の使用
- ファイルと Python ライブラリを Ray ジョブに提供する
- Ray ジョブのデータに接続する

チュートリアル: AWS Glue for Ray で ETL スクリプトを記述する

Ray を使用すると、分散したタスクを Python でネイティブに記述し、スケールすることができます。AWS Glue for Ray には、ジョブとインタラクティブセッションの両方からアクセスできるサーバーレスの Ray 環境が用意されています (Ray のインタラクティブセッションは現在プレビュー中です)。AWS Glue のジョブシステムを使用すると、スケジュールに基づいて、トリガーにより、または AWS Glue コンソールを使用して、一貫した方法でタスクを管理し、実行できます。

これら AWS Glue のツールを組み合わせると、AWS Glue の一般的なユースケースである、抽出、変換、ロード (ETL) ワークロードに使用できる強力なツールチェーンができあがります。このチュートリアルでは、このソリューションの組み立ての基礎を学びます。

また、ETL ワークロードで AWS Glue for Spark を使用する方法についても学習します。AWS Glue for Spark スクリプトの記述方法に関するチュートリアルは、「the section called “チュートリアル: Spark スクリプトの記述”」を参照してください。利用可能なエンジンの詳細については、「the section called “AWS Glue for Spark と AWS Glue for Ray”」を参照してください。Ray は、分析、機械学習 (ML)、アプリケーション開発におけるさまざまな種類のタスクに対応できます。

このチュートリアルでは、Amazon Simple Storage Service (Amazon S3) でホストされている CSV データセットの抽出、変換、ロードを行います。まず、パブリックな Amazon S3 バケットに保存されている New York City Taxi and Limousine Commission (TLC) の乗車記録データのデータセットから始めます。このデータセットの詳細については、「Registry of Open Data on AWS」を参照してください。

Ray Data ライブラリで利用できる事前定義された変換を使用して、データを変換していきます。Ray Data は Ray が設計したデータセットの準備ライブラリで、Ray の環境用に AWS Glue にデフォルトで含まれています。デフォルトで含まれているライブラリの詳細については、「the section called “Ray ジョブで提供されるモジュール”」を参照してください。次に、変換したデータを、自分が管理する Amazon S3 バケットに書き込みます。

前提条件 — このチュートリアルを行うには、AWS Glue と Amazon S3 へのアクセス権限を持つ AWS アカウントが必要です。

ステップ 1: Amazon S3 に出力データを保存するバケットを作成する

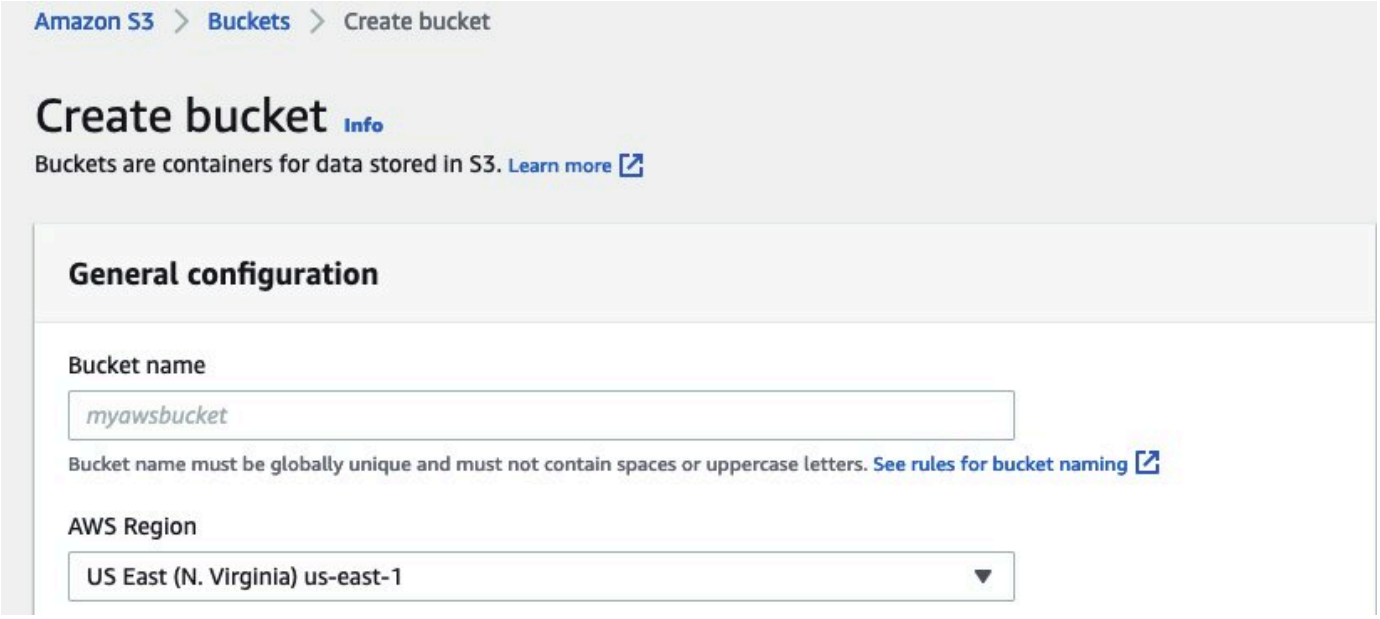
このチュートリアルで作成したデータのシンクとして機能する、ユーザーが管理する Amazon S3 バケットが必要です。このバケットは、以下の手順で作成します。

> **Note**
>
> 自分で管理している既存のバケットにデータを書き込む場合は、このステップは飛ばしてかまいません。その場合は、既存のバケット名である *yourBucketName* をメモしておきます。こちらは後のステップで使用します。

Ray ジョブの出力用バケットを作成するには

- 「Amazon S3 ユーザーガイド」の「バケットの作成」の手順に従ってバケットを作成します。
  - バケット名を選択ぶときは *yourBucketName* をメモしておきます。こちらは後のステップで使用します。
  - 他の設定については、Amazon S3 コンソールに表示された推奨の設定を使えばチュートリアルを問題なく進められるはずです。

  例えば、Amazon S3 コンソールのバケット作成のダイアログボックスは、次のように表示されます。



ステップ 2: Ray ジョブ用の IAM ロールを作成する

ジョブには、以下を持つ AWS Identity and Access Management (IAM) ロールが必要です。

- `AWSGlueServiceRole` マネージドポリシーが付与したアクセス権限。これらは、AWS Glue ジョブを実行するために必要な、基本的な権限です。
- `nyc-tlc/*` Amazon S3 リソースの `Read` のアクセスレベル権限。
- *`yourBucketName`*`/*` Amazon S3 リソースの `Write` のアクセスレベル権限。
- `glue.amazonaws.com` プリンシパルにロールの引き受けを許可する信頼関係。

このロールは、以下の手順で作成します。

AWS Glue for Ray ジョブ用に IAM ロールを作成するには

> **Note**
>
> IAM ロールは、さまざまな手順で作成できます。IAM リソースのプロビジョニング方法の詳細またはオプションについては、AWS Identity and Access Management のドキュメントを参照してください。

1. 上記の Amazon S3 アクセス権限を定義するポリシーを、「IAM ユーザーガイド」の「ビジュアルエディタを使用してポリシーを作成するには」の手順に従って作成します。
   - サービスを選ぶときは、Amazon S3 を選択します。
   - ポリシーのアクセス権限を選ぶときは、以下のリソースの、以下に挙げるアクションのセットを添付します (上記のとおり)。
     - `nyc-tlc/*` Amazon S3 リソースの Read のアクセスレベル権限。
     - *`yourBucketName`*`/*` Amazon S3 リソースの Write のアクセスレベル権限。
   - ポリシー名を選ぶときは、*`YourPolicyName`* をメモしておきます。こちらは後のステップで使用します。
2. AWS Glue for Ray ジョブ用のロールを、「IAM ユーザーガイド」の「AWS サービスのロールを作成するには (コンソール)」の手順に従って作成します。
   - 信頼された AWS サービスエンティティを選ぶときは、`Glue` を選択します。これにより、ジョブに必要な信頼関係が自動的に設定されます。
   - アクセス権限ポリシーのポリシーを選ぶときは、次のポリシーを添付します。

- AWSGlueServiceRole
- *YourPolicyName*

- ロール名を選ぶときは、*YourRoleName* をメモしておきます。こちらは後のステップで使用します。

ステップ 3: AWS Glue for Ray ジョブを作成して実行する

このステップでは、AWS Management Console を使用して AWS Glue ジョブを作成し、これにサンプルスクリプトを提供して、ジョブを実行します。ジョブを作成すると、Ray スクリプトを保存、設定、編集するための場所がコンソールに作成されます。ジョブの作成の詳細については、「the section called “コンソールにサインインする”」をご参照ください。

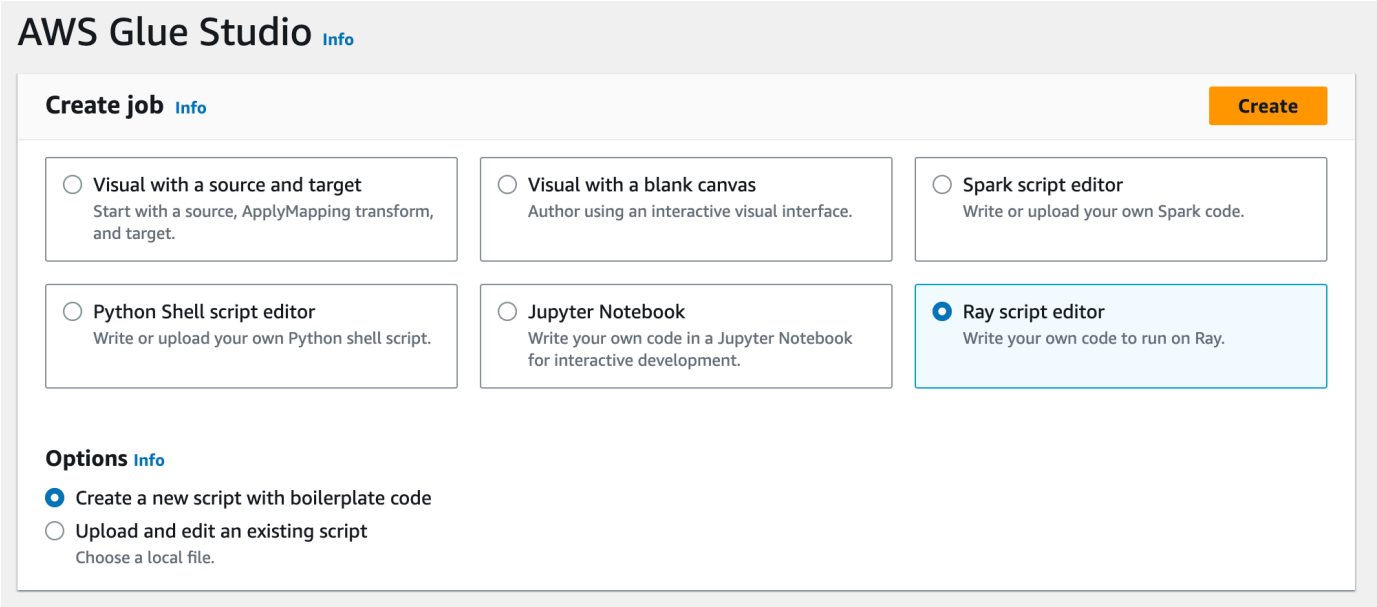
このチュートリアルでは、次の ETL シナリオを取り上げます。New York City TLC の乗車記録データセットから 2022 年 1 月のレコードを読み取り、既存の列のデータを組み合わせてデータセットに新しい列 (tip_rate) を追加し、現在の分析に無関係のいくつかの列を削除してから、結果を *yourBucketName* に書き込みます。次の Ray スクリプトは、以下のステップを実行します。

```
import ray
import pandas
from ray import data

ray.init('auto')

ds = ray.data.read_csv("s3://nyc-tlc/opendata_repo/opendata_webconvert/yellow/
yellow_tripdata_2022-01.csv")

# Add the given new column to the dataset and show the sample record after adding a new
 column
ds = ds.add_column( "tip_rate", lambda df: df["tip_amount"] / df["total_amount"])

# Dropping few columns from the underlying Dataset
ds = ds.drop_columns(["payment_type", "fare_amount", "extra", "tolls_amount",
 "improvement_surcharge"])

ds.write_parquet("s3://yourBucketName/ray/tutorial/output/")
```

AWS Glue for Ray ジョブを作成して実行するには

1. AWS Management Console で、AWS Glue のランディングページに進みます。

2. ナビゲーションペインで、[ETL Jobs] を選択します。
3. [ジョブを作成する] で、[Ray script editor] を選択し、[作成] をクリックします。以下の図を参照してください。



4. スクリプトの全文を [スクリプト] ペインに貼り付け、既存のテキストがあれば置き換えます。
5. [ジョブの詳細] に進み、[IAM ロール] のプロパティを *YourRoleName* に設定します。
6. [保存] をクリックし、[実行] をクリックします。

ステップ 4: 出力を確認する

AWS Glue ジョブを実行したら、出力がこのシナリオの期待値と一致していることを確認する必要があります。確認は、次の手順で行います。

Ray ジョブが正常に実行されたことを確認するには

1. ジョブの詳細ページから [実行] に移動します。
2. 数分後、実行の [実行ステータス] が [成功] と表示されるはずです。
3. Amazon S3 コンソール (https://console.aws.amazon.com/s3/) に移動し、*yourBucketName* をチェックします。出力バケットに書き込まれたファイルが表示されるはずです。
4. Parquet ファイルを読み、内容を確認します。これは既存のツールで実行できます。Parquet ファイルを確認するプロセスがない場合は、Spark または Ray を使用して、AWS Glue コンソールで AWS Glue インタラクティブセッションを実行します (現在プレビュー中)。

インタラクティブセッションでは、Ray Data、Spark、pandas のいずれかのライブラリにアクセスします。これは (選択したエンジンに応じて) デフォルトで指定されています。ファイルの内容の確認には、これらのライブラリで利用できる一般的な確認方法、例えば、`count`、`schema`、`show` などを使用できます。コンソールのインタラクティブセッションの詳細については、「AWS Glue Studio と AWS Glue を用いてノートブックの使用」を参照してください。

ファイルがバケットに書き込まれていることを確認できたので、出力に問題があったとしても、それは IAM の設定とは無関係であると確実に言えます。*`yourRoleName`* を使用してセッションを設定するのは、関連するファイルにアクセスできるようにするためです。

期待した結果が得られない場合は、このガイドのトラブルシューティングのページを参照し、エラーの原因を特定して修正します。ジョブの実行エラー状態を解釈するには、「the section called "ジョブ実行ステータス"」を参照してください。トラブルシューティングのページは「AWS Glue のトラブルシューティング」にあります。Ray ジョブに関連する特定のエラーについては、トラブルシューティング内の「the section called "Ray エラーをトラブルシューティングする"」を参照してください。

次のステップ

これで、AWS Glue for Ray を使用した ETL のプロセスの全体を確認し、実行できました。次のリソースを使用すると、AWS Glue for Ray がデータを大規模に変換し解釈するためにどのようなツールを提供しているのかを理解できます。

- Ray のタスクのモデルに関する詳細は、「the section called "AWS Glue for Ray での Ray Core と Ray Data の使用"」を参照してください。Ray のタスクの使用に関する詳細は、Ray Core ドキュメントにある例を参照してください。Ray のドキュメントの「Ray Core: Ray Tutorials and Examples (2.4.0)」を参照してください。
- AWS Glue for Ray で利用可能なデータ管理ライブラリのガイダンスは、「the section called "データへの接続"」を参照してください。Ray Data を使用してデータセットを変換し記述する方法の詳細については、Ray Data ドキュメントにある例を参照してください。「Ray Data: Examples (2.4.0)」を参照してください。
- AWS Glue for Ray ジョブの設定に関する詳細は、「the section called "Ray ジョブの使用"」を参照してください。
- AWS Glue for Ray のスクリプトの記述に関する詳細は、本セクション内のドキュメントを引き続きお読みください。

AWS Glue for Ray での Ray Core と Ray Data の使用

Ray は、クラスター全体に作業を分散することで Python スクリプトをスケールアップするためのフレームワークです。Ray はさまざまな問題の解決策として利用できます。つまり、特定のタスクを最適化するためのライブラリを提供しています。AWS Glue では、Ray を使用して大規模なデータセットを変換することに重点を置いています。AWS Glue は Ray Core の一部と Ray Data をサポートしているため、このタスクが容易になります。

Ray Core とは

分散アプリケーションを構築する最初のステップは、同時に実行できる作業を特定して定義することです。Ray Core には、同時に実行できるタスクを定義するために使用する Ray の機能が一部含まれています。Ray では、提供されるツールを学習するのに役立つリファレンスおよびクイックスタート情報を入手できます。詳細については、「What is Ray Core?」と「Ray Core Quick Start」を参照してください。Ray で同時タスクを効果的に定義する方法の詳細については、「Tips for first-time users」を参照してください。

> **Ray のタスクとアクター**
>
> AWS Glue for Ray ドキュメントでは、Ray の中核的な概念であるタスクとアクターについて言及することがあります。
> Ray では、分散コンピューティングシステムの構成要素として Python の関数とクラスを使用します。Python の関数や変数をクラスで使用すると「メソッド」や「属性」になるのと同様に、ワーカーにコードを送信するために Ray で使用すると、関数は「タスク」になり、クラスは「アクター」になります。Ray で使用される可能性のある関数やクラスは、`@ray.remote` アノテーションで識別できます。
> タスクとアクターは設定可能で、ライフサイクルがあり、そのライフサイクルを通じてコンピューティングリソースを消費します。エラーをスローするコードは、問題の根本原因を見つける場合に、タスクまたはアクターまでさかのぼることができます。したがって、AWS Glue for Ray ジョブの設定、モニタリング、デバッグの方法を学習するときに、これらの用語が出現する可能性があります。
> タスクとアクターを効果的に使用して分散アプリケーションを構築する方法を学習するには、Ray のドキュメントの「Key Concepts」を参照してください。

AWS Glue for Ray での Ray Core

AWS Glue for Ray 環境では、ログの収集と視覚化だけでなく、クラスターの形成とスケーリングも管理します。こうした懸念事項を管理しているため、当社は、オープンソースのクラスターでこれらの対処に使用される、Ray Core の API へのアクセスとサポートを制限しています。

マネージド `Ray2.4` ランタイム環境では、以下はサポートされていません。

- Ray Core CLI
- Ray State CLI
- `ray.util.metrics` Prometheus メトリクスユーティリティメソッド:
  - Counter
  - Gauge
  - Histogram
- その他のデバッグツール
  - ray.util.pdb.set_trace
  - ray.util.inspect_serializability
  - ray.timeline

Ray Data とは

データソースや送信先に接続する場合、データセットを処理する場合、一般的な変換を開始する場合などに Ray Data を使用すれば、Ray データセットの変換に関する問題を Ray で簡単に解決できます。Ray Data の使用方法の詳細については、「Ray Datasets: Distributed Data Preprocessing」を参照してください。

Ray Data やその他のツールを使用すると、データにアクセスできます。Ray 内におけるデータへのアクセスの詳細については、「the section called “データへの接続”」を参照してください。

AWS Glue for Ray での Ray Data

Ray Data では、マネージド `Ray2.4` ランタイム環境がデフォルトでサポートおよび提供されています。提供されるモジュールの詳細については、「the section called “Ray ジョブで提供されるモジュール”」を参照してください。

ファイルと Python ライブラリを Ray ジョブに提供する

このセクションでは、AWS Glue の Ray ジョブで Python ライブラリを使用するために必要な情報を提供します。すべての Ray ジョブにデフォルトで含まれている、特定の共通ライブラリを使用できます。また、Ray ジョブに独自の Python ライブラリを指定することもできます。

Ray ジョブで提供されるモジュール

次のパッケージを使用して、Ray ジョブでデータ統合ワークフローを実行できます。これらのパッケージは Ray ジョブで、デフォルトで使用できます。

AWS Glue version 4.0

AWS Glue 4.0 では、Ray (`Ray2.4` ランタイム) 環境で次のパッケージが利用可能です。

- boto3 == 1.26.133
- ray == 2.4.0
- pyarrow == 11.0.0
- pandas == 1.5.3
- numpy == 1.24.3
- fsspec == 2023.4.0

このリストには、`ray[data] == 2.4.0` でインストールされるすべてのパッケージが含まれます。Ray データは追加設定なしにサポートされます。

Ray ジョブへのファイルの提供

`--working-dir` パラメータを使用して Ray ジョブにファイルを提供できます。このパラメータには、Amazon S3 でホストされている .zip ファイルへのパスを指定します。.zip ファイル内では、ファイルは最上位の 1 つのディレクトリに含まれている必要があります。最上位には、他のファイルがないようにします。

ファイルは、スクリプトの実行を開始する前に各 Ray ノードに配布されます。それにより、各 Ray ノードが利用できるディスク容量にどのような影響が出るかを考えます。使用できるディスク容量は、ジョブ構成で設定された WorkerType により決まります。ジョブに大量のデータを提供したい場合、このメカニズムは適切なソリューションではありません。データをジョブに提供する方法の詳細については、「the section called “データへの接続”」を参照してください。

あたかもディレクトリが working_dir パラメータで Ray に提供されたかのように、ファイルにアクセスできるようになります。例えば、.zip ファイルの最上位ディレクトリにある sample.txt という名前のファイルを読み取るには、次のように呼び出すことができます。

```
@ray.remote
def do_work():
    f = open("sample.txt", "r")
    print(f.read())
```

working_dir の詳細については、Ray のドキュメントを参照してください。この機能は、Ray のネイティブの機能と同じように動作します。

Ray ジョブ用の Python モジュールを追加する

PyPI の追加モジュール

Ray ジョブでは、Python Package Installer (pip3) を使用して、Ray スクリプトで使用するモジュールを追加でインストールします。--pip-install パラメータでコンマ区切りの Python モジュールのリストを指定することで、新しいモジュールを追加したり、既存のモジュールのバージョンを変更したりできます。

例えば、更新や新しい scikit-learn モジュールの追加には、次のキー値ペアを使用します。

```
"--pip-install", "scikit-learn==0.21.3"
```

カスタムモジュールまたはカスタムパッチがある場合は、--s3-py-modules パラメータを使用して Amazon S3 から自分のライブラリをディストリビューションできます。ディストリビューションは、アップロード前に再パッケージ化と構築が必要になる場合があります。the section called “Ray ジョブに Python コードを含める” のガイドラインに従います。

Amazon S3 のカスタムディストリビューション

カスタムディストリビューションは、依存関係に関する Ray パッケージのガイドラインに従う必要があります。これらのディストリビューションの構築方法については、次のセクションをご覧ください。Ray での依存関係の設定方法に関する詳細は、Ray のドキュメントの「Environment Dependencies」(環境の依存関係) を参照してください。

コンテンツを評価した後にカスタムディストリビューションを含めるには、ジョブの IAM ロールで利用できるバケットにディストリビューションをアップロードします。パラメータの設定

で、Python zip アーカイブへの Amazon S3 パスを指定します。複数のディストリビューションを指定する場合は、カンマで区切ります。例:

```
"--s3-py-modules", "s3://s3bucket/pythonPackage.zip"
```

制約事項

Ray ジョブは、ジョブ環境でのネイティブコードのコンパイルをサポートしていません。Python の依存関係がネイティブのコンパイル済みコードに推移的に依存している場合、これにより制限される可能性があります。Ray ジョブは提供されたバイナリを実行できますが、Linux on ARM64 用にコンパイルする必要があります。つまり、`aarch64manylinux` wheel の内容を使用できる場合があるということです。Ray の基準に合わせて wheel を再パッケージ化することで、ネイティブの依存関係をコンパイルされた形式で提供できます。通常、これは `dist-info` フォルダを削除して、アーカイブのルートにあるフォルダが 1 つだけになるようにすることを意味します。

このパラメータを使用して `ray` または `ray[data]` のバージョンをアップグレードすることはできません。Ray の新しいバージョンを使用するには、ジョブのランタイムフィールドを、そのサポートをリリースした後に変更する必要があります。サポートされている Ray バージョンの詳細については、「the section called “AWS Glue バージョン”」を参照してください。

Ray ジョブに Python コードを含める

Python Software Foundation は、さまざまなランタイムで使用できるよう、Python ファイルをパッケージ化するための標準化された機能を提供しています。Ray では、注意すべきパッケージ基準に制限が設けられています。AWS Glue が、Ray に指定された以外のパッケージ基準を指定することはありません。次の手順では、シンプルな Python パッケージのパッケージ化に関する標準的なガイダンスを提供します。

ファイルを `.zip` アーカイブにパッケージ化します。1 つのディレクトリがアーカイブのルートにある必要があります。アーカイブのルートレベルにその他のファイルがあると、予期しない動作が発生する可能性があります。ルートディレクトリはパッケージで、インポート時に Python コードを参照するためにその名前が使用されます。

`--s3-py-modules` を使用してこの形式のディストリビューションを Ray ジョブに提供すると、Ray スクリプト内のパッケージから Python コードをインポートできます。

パッケージは、単一の Python モジュールにいくつかの Python ファイルを提供することも、多数のモジュールを一緒にパッケージ化することもできます。PyPI のライブラリなどの依存関係を再パッケージ化するときは、それらのパッケージ内の隠しファイルやメタデータディレクトリを確認してください。

> **Warning**
>
> OS の動作によっては、これらのパッケージ化手順に正しく従うことが難しくなります。
>
> - OSX では、__MACOSX などの隠しファイルが zip ファイルの最上位に追加されることがあります。
> - Windows では、ファイルが zip 内のフォルダに自動的に追加され、意図せずにネストされたフォルダが作成されることがあります。

以下の手順は、Info-Zip zip および zipinfo ユーティリティのディストリビューションを提供する Amazon Linux 2 または類似の OS でファイルを操作していることを前提としています 予期しない動作を防ぐため、これらのツールを使用することをお勧めします。

Ray で使用する Python ファイルをパッケージ化するには

1. パッケージ名で一時ディレクトリを作成し、作業ディレクトリが親ディレクトリであることを確認します。これを行うには、次のコマンドを使用します。

```
cd parent_directory
mkdir temp_dir
```

2. ファイルを一時ディレクトリにコピーし、ディレクトリ構造を確認します。このディレクトリの内容は、Python モジュールとして直接アクセスされます。これを行うには、次の コマンドを使用します。

```
ls -AR temp_dir
# my_file_1.py
# my_file_2.py
```

3. zip を使用して一時フォルダを圧縮します。これを行うには、次のコマンドを使用します。

```
zip -r zip_file.zip temp_dir
```

4. ファイルが適切にパッケージ化されていることを確認します。*zip_file*.zip は作業ディレクトリにあるはずです。次のコマンドで検査できます。

```
zipinfo -1 zip_file.zip
# temp_dir/
```

```
# temp_dir/my_file_1.py
# temp_dir/my_file_2.py
```

Python パッケージを Ray で使用できるように再パッケージ化するには。

1. パッケージ名で一時ディレクトリを作成し、作業ディレクトリが親ディレクトリであることを確認します。これを行うには、次のコマンドを使用します。

```
cd parent_directory
mkdir temp_dir
```

2. パッケージを解凍し、その内容を一時ディレクトリにコピーします。以前のパッケージング標準に関連するファイルを削除し、モジュールの内容のみを残します。以下のコマンドを実行して、ファイル構造が正しいことを確認します。

```
ls -AR temp_dir
# my_module
# my_module/__init__.py
# my_module/my_file_1.py
# my_module/my_submodule/__init__.py
# my_module/my_submodule/my_file_2.py
# my_module/my_submodule/my_file_3.py
```

3. zip を使用して一時フォルダを圧縮します。これを行うには、次のコマンドを使用します。

```
zip -r zip_file.zip temp_dir
```

4. ファイルが適切にパッケージ化されていることを確認します。*zip_file*.zip は作業ディレクトリにあるはずです。次のコマンドで検査できます。

```
zipinfo -1 zip_file.zip
# temp_dir/my_module/
# temp_dir/my_module/__init__.py
# temp_dir/my_module/my_file_1.py
# temp_dir/my_module/my_submodule/
# temp_dir/my_module/my_submodule/__init__.py
# temp_dir/my_module/my_submodule/my_file_2.py
# temp_dir/my_module/my_submodule/my_file_3.py
```

Ray ジョブのデータに接続する

AWS Glue Ray ジョブでは、データをすばやく統合するために設計されたさまざまな Python パッケージを使用できます。環境が混乱した状態にならないよう、最低限の依存関係のみが提供されています。デフォルトで含まれる内容の詳細については、「the section called “Ray ジョブで提供されるモジュール”」を参照してください。

> **Note**
>
> AWS Glue 抽出、変換、ロード (ETL) は、DynamicFrame の抽象化を提供し、データセット内の行間におけるスキーマの違いを解決する ETL ワークフローを効率化します。AWS GlueETL には、ジョブのブックマークや入力ファイルのグループ化など、追加機能があります。Ray ジョブには現在、対応する機能は提供されていません。
> AWS Glue for Spark では、特定のデータ形式、ソース、シンクへの接続を直接サポートしています。Ray では、AWS SDK for pandas と現在のサードパーティーライブラリがそれらのニーズを実質的にカバーしています。どのような機能を利用できるか理解するには、これらのライブラリを調べる必要があります。

AWS Glue for Ray の Amazon VPC との統合は、現在ご利用いただけません。Amazon VPC 内のリソースには、パブリックルートがないとアクセスできません。Amazon VPC での AWS Glue の使用に関する詳細は、「the section called “AWS Glue のインターフェイス VPC エンドポイント (AWS PrivateLink) の設定 ”」を参照してください。

Ray でデータを使用するための共通ライブラリ

Ray Data — Ray Data は、一般的なデータ形式、ソース、シンクを処理する方法を提供します。Ray Data でサポートされている形式とソースの詳細については、Ray Data のドキュメントの「Input/Output」を参照してください。Ray Data は、データセットを扱うための汎用のライブラリではなく特化型のライブラリです。

Ray では、Ray Data がユーザーの作業に最適なソリューションとなるユースケースについて、特定のガイダンスを提供します。詳細については、Ray のドキュメントの「Ray の使用例」を参照してください。

AWS SDK for pandas (awswrangler) – AWS SDK for pandas は、pandas の DataFrames を使用して、変換によりデータを管理する際に、AWS サービスでの読み取りと書き込みを行うためのクリーンなテスト済みのソリューションを提供する AWS の製品です。AWS SDK for pandas でサ

ポートされている形式とソースの詳細については、AWS SDK for pandas のドキュメントの「API Reference」を参照してください。

AWS SDK for pandas を使用してデータを読み書きする方法の例については、AWS SDK for pandas のドキュメントの「Quick Start」を参照してください。AWS SDK for pandas では、データの変換は行いません。ソースからの読み取りと書き込みのみサポートしています。

Modin — Modin は、一般的な pandas のオペレーションを配布可能な方法で実装している Python ライブラリです。Modin の詳細については、Modin のドキュメントを参照してください。Modin 自体は、ソースからの読み取りと書き込みをサポートしていません。一般的な変換の分散実装を行います。Modin は AWS SDK for pandas でサポートされています。

Modin を実行し、AWS SDK for pandas と共に Ray 環境で使用すると、高いパフォーマンスで一般的な ETL タスクを実行できます。Modin を AWS SDK for pandas と併用する方法の詳細については、AWS SDK for pandas のドキュメントの「At scale」を参照してください。

その他のフレームワーク — Ray がサポートしているフレームワークの詳細については、Ray のドキュメントの「Ray エコシステム」を参照してください。AWS Glue for Ray はその他のフレームワークをサポートしていません。

データカタログを介したデータへの接続

Ray ジョブと組み合わせたデータカタログによるデータ管理は、AWS SDK for pandas でサポートされています。詳細については、AWS SDK for pandas ウェブサイトの「Glue Catalog」(Glue カタログ) を参照してください。

このサービスを AWS SDK で使用する

AWS ソフトウェア開発キット (SDK) は、多くの一般的なプログラミング言語で使用できます。各 SDK には、デベロッパーが好みの言語でアプリケーションを簡単に構築できるようにする API、コード例、およびドキュメントが提供されています。

SDK ドキュメント	コードの例
AWS SDK for C++	AWS SDK for C++ コードの例
AWS CLI	AWS CLI コードの例
AWS SDK for Go	AWS SDK for Go コードの例
AWS SDK for Java	AWS SDK for Java コードの例
AWS SDK for JavaScript	AWS SDK for JavaScript コードの例
AWS SDK for Kotlin	AWS SDK for Kotlin コードの例
AWS SDK for .NET	AWS SDK for .NET コードの例
AWS SDK for PHP	AWS SDK for PHP コードの例
AWS Tools for PowerShell	AWS Tools for PowerShell コードの例
AWS SDK for Python (Boto3)	AWS SDK for Python (Boto3) コードの例
AWS SDK for Ruby	AWS SDK for Ruby コードの例
AWS SDK for Rust	AWS SDK for Rust コードの例
AWS SDK for SAP ABAP	AWS SDK for SAP ABAP コードの例
AWS SDK for Swift	AWS SDK for Swift コードの例

このサービスに固有の例については、「AWS SDK を使用した AWS Glue API コード例」を参照してください。

可用性の例

必要なものが見つからなかった場合。このページの下側にある [Provide feedback (フィードバックを送信)] リンクから、コードの例をリクエストしてください。

AWS Glue API

このセクションでは、AWS Glue SDK およびツールで使用されるデータ型とプリミティブについて説明します。AWS Management Console の外部で、プログラムによって AWS Glue を操作する一般的な方法は 3 つあり、それぞれ独自のドキュメントがあります。

- 言語 SDK ライブラリを使用すれば、一般的なプログラミング言語で AWS リソースにアクセスできます。詳細については、「AWS での構築ツール」をご覧ください。
- AWS CLI を使用すると、コマンドラインで AWS リソースにアクセスできます。詳細については、「AWS CLI コマンドリファレンス」を参照してください。
- AWS CloudFormation を使用すると、一貫して一緒にプロビジョンされる一連の AWS リソースを定義できます。詳細については、「AWS CloudFormation: AWS Glue リソースタイプリファレンス」を参照してください。

このセクションでは、これらの SDK およびツールから独立した共有プリミティブについて説明します。ツールは、AWS Glue ウェブ API リファレンスを使用して AWS と通信します。

目次

- AWS Glue のセキュリティ API
  - データ型
  - DataCatalogEncryptionSettings 構造
  - EncryptionAtRest 構造
  - ConnectionPasswordEncryption 構造
  - EncryptionConfiguration 構造
  - S3Encryption 構造
  - CloudWatchEncryption 構造
  - JobBookmarksEncryption 構造
  - SecurityConfiguration 構造
  - GluePolicy の構造
  - DataQualityEncryption 構造
  - 操作
  - GetDataCatalogEncryptionSettings アクション (Python: get_data_catalog_encryption_settings)
  - PutDataCatalogEncryptionSettings アクション (Python: put_data_catalog_encryption_settings)

- PutResourcePolicy アクション (Python: put_resource_policy)
- GetResourcePolicy アクション (Python: get_resource_policy)
- DeleteResourcePolicy アクション (Python: delete_resource_policy)
- CreateSecurityConfiguration アクション (Python: create_security_configuration)
- DeleteSecurityConfiguration アクション (Python: delete_security_configuration)
- GetSecurityConfiguration アクション (Python: get_security_configuration)
- GetSecurityConfigurations アクション (Python: get_security_configurations)
- GetResourcePolicies アクション (Python: get_resource_policies)
- カタログオブジェクト API
  - カタログ API
    - データ型
    - カタログの構造
    - CatalogInput 構造
    - TargetRedshiftCatalog 構造
    - CatalogProperties 構造
    - CatalogPropertiesOutput 構造
    - DataLakeAccessProperties 構造
    - DataLakeAccessPropertiesOutput 構造
    - FederatedCatalog 構造
    - 操作
    - CreateCatalog アクション (Python: create_catalog)
    - UpdateCatalog アクション (Python: update_catalog)
    - DeleteCatalog アクション (Python: delete_catalog)
    - GetCatalog アクション (Python: get_catalog)
    - GetCatalogs アクション (Python: get_catalogs)
  - データベース API
    - データ型
    - データベース構造
    - DatabaseInput 構造
    - PrincipalPermissions 構造

- DataLakePrincipal 構造
- DatabaseIdentifier 構造
- FederatedDatabase 構造
- 操作
- CreateDatabase アクション (Python: create_database)
- UpdateDatabase アクション (Python: update_database)
- DeleteDatabase アクション (Python: delete_database)
- GetDatabase アクション (Python: get_database)
- GetDatabases アクション (Python: get_databases)

- Table API
  - データ型
  - Table 構造
  - TableInput 構造
  - FederatedTable 構造
  - 列の構造
  - StorageDescriptor 構造
  - SchemaReference 構造
  - SerDeInfo 構造
  - Order 構造
  - SkewedInfo 構造
  - TableVersion 構造
  - TableError 構造
  - TableVersionError 構造
  - SortCriterion 構造
  - TableIdentifier 構造
  - KeySchemaElement 構造
  - PartitionIndex 構造
  - PartitionIndexDescriptor 構造
  - BackfillError 構造
  - IcebergInput 構造

- OpenTableFormatInput 構造
- ViewDefinition 構造
- ViewDefinitionInput 構造
- ViewRepresentation 構造
- ViewRepresentationInput 構造
- UpdateOpenTableFormatInput 構造
- UpdateIcebergInput 構造
- CreateIcebergTableInput 構造
- UpdateIcebergTableInput 構造
- IcebergSortOrder 構造
- IcebergSortField 構造
- IcebergPartitionSpec 構造
- IcebergPartitionField 構造
- IcebergSchema 構造
- IcebergStructField 構造
- IcebergTableUpdate 構造
- 操作
- CreateTable アクション (Python: create_table)
- UpdateTable アクション (Python: update_table)
- DeleteTable アクション (Python: delete_table)
- BatchDeleteTable アクション (Python: batch_delete_table)
- GetTable アクション (Python:get_table)
- GetTables アクション (Python: get_tables)
- GetTableVersion アクション (Python: get_table_version)
- GetTableVersions アクション (Python: get_table_versions)
- DeleteTableVersion アクション (Python: delete_table_version)
- BatchDeleteTableVersion アクション (Python: batch_delete_table_version)
- SearchTables アクション (Python: search_tables)
- GetPartitionIndexes アクション (Python: get_partition_indexes)
- CreatePartitionIndex アクション (Python: create_partition_index)

- DeletePartitionIndex アクション (Python: delete_partition_index)
- GetColumnStatisticsForTable アクション (Python: get_column_statistics_for_table)
- UpdateColumnStatisticsForTable アクション (Python: update_column_statistics_for_table)
- DeleteColumnStatisticsForTable アクション (Python: delete_column_statistics_for_table)

- パーティション API
  - データ型
  - パーティションの構造
  - PartitionInput の構造
  - PartitionSpecWithSharedStorageDescriptor 構造
  - PartitionListComposingSpec 構造
  - PartitionSpecProxy 構造
  - PartitionValueList 構造
  - セグメント構造
  - PartitionError 構造
  - BatchUpdatePartitionFailureEntry 構造
  - BatchUpdatePartitionRequestEntry 構造
  - StorageDescriptor 構造
  - SchemaReference 構造
  - SerDeInfo 構造
  - SkewedInfo 構造
  - 操作
  - CreatePartition アクション (Python: create_partition)
  - BatchCreatePartition アクション (Python: batch_create_partition)
  - UpdatePartition アクション (Python: update_partition)
  - DeletePartition アクション (Python: delete_partition)
  - BatchDeletePartition アクション (Python: batch_delete_partition)
  - GetPartition アクション (Python: get_partition)
  - GetPartitions アクション (Python: get_partitions)
  - BatchGetPartition アクション (Python: batch_get_partition)
  - BatchUpdatePartition アクション (Python: batch_update_partition)

- GetColumnStatisticsForPartition アクション (Python: get_column_statistics_for_partition)
- UpdateColumnStatisticsForPartition アクション (Python: update_column_statistics_for_partition)
- DeleteColumnStatisticsForPartition アクション (Python: delete_column_statistics_for_partition)

- 接続 API
  - 接続 API
    - データ型
    - 接続構造
    - ConnectionInput 構造
    - TestConnectionInput 構造
    - PhysicalConnectionRequirements 構造
    - GetConnectionsFilter 構造
    - AuthenticationConfiguration 構造
    - AuthenticationConfigurationInput 構造
    - OAuth2Properties 構造
    - OAuth2PropertiesInput 構造
    - OAuth2ClientApplication 構造
    - AuthorizationCodeProperties 構造
    - BasicAuthenticationCredentials 構造
    - OAuth2Credentials 構造
    - 操作
    - CreateConnection アクション (Python: create_connection)
    - DeleteConnection アクション (Python: delete_connection)
    - GetConnection アクション (Python: get_connection)
    - GetConnections アクション (Python: get_connections)
    - UpdateConnection アクション (Python: update_connection)
    - TestConnection アクション (Python: test_connection)
    - BatchDeleteConnection アクション (Python: batch_delete_connection)
  - 接続タイプ API
    - 接続管理 API

        - DescribeConnectionType アクション (Python: describe_connection_type)
        - ListConnectionTypes アクション (Python: list_connection_types)
        - ConnectionTypeBrief 構造
        - ConnectionTypeVariant の構造
        - データ型
        - 検証構造
        - AuthConfiguration 構造
        - Capabilities 構造
        - Property 構造
        - AllowedValue 構造
        - ComputeEnvironmentConfiguration 構造
    - 接続メタデータとプレビュー API
        - データ型
        - エンティティ構造
        - フィールド構造
        - 操作
        - ListEntities アクション (Python: list_entities)
        - DescribeEntity アクション (Python: describe_entity)
        - GetEntityRecords アクション (Python: get_entity_records)
- ユーザー定義関数 API
    - データ型
    - UserDefinedFunction 構造
    - UserDefinedFunctionInput 構造
    - 操作
    - CreateUserDefinedFunction アクション (Python: create_user_defined_function)
    - UpdateUserDefinedFunction アクション (Python: update_user_defined_function)
    - DeleteUserDefinedFunction アクション (Python: delete_user_defined_function)
    - GetUserDefinedFunction アクション (Python: get_user_defined_function)
    - GetUserDefinedFunctions アクション (Python: get_user_defined_functions)

- Athena カタログを AWS Glue にインポートする

    - データ型
    - CatalogImportStatus 構造
    - 操作
    - ImportCatalogToGlue アクション (Python: import_catalog_to_glue)
    - GetCatalogImportStatus アクション (Python: get_catalog_import_status)
- テーブルオプティマイザー API
    - データ型
    - TableOptimizer の構造
    - TableOptimizerConfiguration の構造
    - TableOptimizerVpcConfiguration 構造
    - CompactionConfiguration 構造
    - IcebergCompactionConfiguration 構造
    - TableOptimizerRun の構造
    - BatchGetTableOptimizerEntry の構造
    - BatchTableOptimizer の構造
    - BatchGetTableOptimizerError の構造
    - RetentionConfiguration 構造
    - IcebergRetentionConfiguration 構造
    - OrphanFileDeletionConfiguration 構造
    - IcebergOrphanFileDeletionConfiguration 構造
    - CompactionMetrics 構造
    - RetentionMetrics 構造
    - OrphanFileDeletionMetrics 構造
    - IcebergCompactionMetrics 構造
    - IcebergRetentionMetrics 構造
    - IcebergOrphanFileDeletionMetrics 構造
    - RunMetrics の構造
    - 操作
    - GetTableOptimizer アクション (Python: get_table_optimizer)
    - BatchGetTableOptimizer アクション (Python: batch_get_table_optimizer)

- ListTableOptimizerRuns アクション (Python: list_table_optimizer_runs)
- CreateTableOptimizer アクション (Python: create_table_optimizer)
- DeleteTableOptimizer アクション (Python: delete_table_optimizer)
- UpdateTableOptimizer アクション (Python: update_table_optimizer)
- クローラーおよび分類子 API
  - 分類子 API
    - データ型
    - 分類子の構造
    - GrokClassifier の構造
    - XMLClassifier の構造
    - JsonClassifier の構造
    - CsvClassifier の構造
    - CreateGrokClassifierRequest の構造
    - UpdateGrokClassifierRequest の構造
    - CreateXMLClassifierRequest の構造
    - UpdateXMLClassifierRequest の構造
    - CreateJsonClassifierRequest の構造
    - UpdateJsonClassifierRequest の構造
    - CreateCsvClassifierRequest の構造
    - UpdateCsvClassifierRequest の構造
    - 操作
    - CreateClassifier アクション (Python: create_classifier)
    - DeleteClassifier アクション (Python: delete_classifier)
    - GetClassifier アクション (Python: get_classifier)
    - GetClassifiers アクション (Python: get_classifiers)
    - UpdateClassifier アクション (Python: update_classifier)
  - クローラー API
    - データ型
    - Crawler 構造
    - Schedule 構造

- CrawlerTargets 構造
- S3Target 構造
- S3DeltaCatalogTarget 構造
- S3DeltaDirectTarget 構造
- JdbcTarget 構造
- MongoDBTarget 構造
- DynamoDBTarget 構造
- DeltaTarget 構造
- IcebergTarget 構造
- HudiTarget 構造
- CatalogTarget 構造
- CrawlerMetrics 構造
- CrawlerHistory 構造
- CrawlsFilter 構造
- SchemaChangePolicy 構造
- LastCrawlInfo 構造
- RecrawlPolicy 構造
- LineageConfiguration 構造
- LakeFormationConfiguration 構造
- 操作
- CreateCrawler アクション (Python: create_crawler)
- DeleteCrawler アクション (Python: delete_crawler)
- GetCrawler アクション (Python: get_crawler)
- GetCrawlers アクション (Python: get_crawlers)
- GetCrawlerMetrics アクション (Python: get_crawler_metrics)
- UpdateCrawler アクション (Python: update_crawler)
- StartCrawler アクション (Python: start_crawler)
- StopCrawler アクション (Python: stop_crawler)
- BatchGetCrawlers Action (Python: batch_get_crawlers)
- ListCrawlers アクション (Python: list_crawlers)

- ListCrawls アクション (Python: list_crawls)
- 列統計 API
  - データ型
  - ColumnStatisticsTaskRun の構造
  - ColumnStatisticsTaskSettings 構造
  - ExecutionAttempt 構造
  - 操作
  - StartColumnStatisticsTaskRun アクション (Python: start_column_statistics_task_run)
  - GetColumnStatisticsTaskRun アクション (Python: get_column_statistics_task_run)
  - GetColumnStatisticsTaskRuns アクション (Python: get_column_statistics_task_runs)
  - ListColumnStatisticsTaskRuns アクション (Python: list_column_statistics_task_runs)
  - StopColumnStatisticsTaskRun アクション (Python: stop_column_statistics_task_run)
  - CreateColumnStatisticsTaskSettings アクション (Python: create_column_statistics_task_settings)
  - UpdateColumnStatisticsTaskSettings アクション (Python: update_column_statistics_task_settings)
  - GetColumnStatisticsTaskSettings アクション (Python: get_column_statistics_task_settings)
  - DeleteColumnStatisticsTaskSettings アクション (Python: delete_column_statistics_task_settings)
  - StartColumnStatisticsTaskRunSchedule アクション (Python: start_column_statistics_task_run_schedule)
  - StopColumnStatisticsTaskRunSchedule アクション (Python: stop_column_statistics_task_run_schedule)
  - 例外
  - ColumnStatisticsTaskRunningException の構造
  - ColumnStatisticsTaskNotRunningException の構造
  - ColumnStatisticsTaskStoppingException の構造
  - ColumnStatisticsTaskAutoConcurrencyLimitException 構造
  - InvalidCatalogSettingException 構造
- クローラースケジューラ API
  - データ型

    - Schedule 構造
    - 操作
    - UpdateCrawlerSchedule アクション (Python: update_crawler_schedule)
    - StartCrawlerSchedule アクション (Python: start_crawler_schedule)
    - StopCrawlerSchedule アクション (Python: stop_crawler_schedule)
- ETL スクリプト API の自動生成
    - データ型
    - CodeGenNode 構造
    - CodeGenNodeArg 構造
    - CodeGenEdge 構造
    - 場所の構造
    - CatalogEntry 構造
    - MappingEntry 構造
    - 操作
    - CreateScript アクション (Python: create_script)
    - GetDataflowGraph アクション (Python: get_dataflow_graph)
    - GetMapping アクション (Python: get_mapping)
    - GetPlan アクション (Python: get_plan)
- ビジュアルジョブ API
    - データ型
    - CodeGenConfiguration 構造
    - JDBCConnectorOptions 構造
    - StreamingDataProviewOptions 構造
    - AthenaConnectorSource 構造
    - JDBCConnectorSource 構造
    - SparkConnectorSource 構造
    - CatalogSource 構造
    - MySQLCatalogSource 構造
    - PostgreSQLCatalogSource 構造
    - OracleSQLCatalogSource 構造

- MicrosoftSQLServerCatalogSource 構造
- CatalogKinesisSource 構造
- DirectKinesisSource 構造
- KinesisStreamingSourceOptions 構造
- CatalogKafkaSource 構造
- DirectKafkaSource 構造
- KafkaStreamingSourceOptions 構造
- RedShiftSource 構造
- AmazonRedshiftSource 構造
- AmazonRedshiftNodeData 構造
- AmazonRedshiftAdvancedOption 構造
- Option 構造
- S3CatalogSource 構造
- S3SourceAdditionalOptions 構造
- S3CsvSource 構造
- DirectJDBCSource 構造
- S3DirectSourceAdditionalOptions 構造
- S3JsonSource 構造
- S3ParquetSource 構造
- S3DeltaSource 構造
- S3CatalogDeltaSource 構造
- CatalogDeltaSource 構造
- S3HudiSource 構造
- S3CatalogHudiSource 構造
- S3ExcelSource の構造
- CatalogHudiSource 構造
- DynamoDBCatalogSource 構造
- RelationalCatalogSource 構造
- JDBCConnectorTarget 構造
- SparkConnectorTerg 構造

- BasicCatalogTerg 構造
- MySQLCatalogTarget 構造
- PostgreSQLCatalogTarget 構造
- OracleSQLCatalogTarget 構造
- MicrosoftSQLServerCatalogTarget 構造
- RedShiftTarget 構造
- AmazonRedshiftTarget 構造
- UpsertRedshiftTargetOptions 構造
- S3CatalogTarget 構造
- S3GlueParquetTarget 構造
- CatalogSchemaChangePolicy 構造
- S3directTarget 構造
- S3HudiCatalogTarget 構造
- S3HudiDirectTarget 構造
- S3DeltaCatalogTarget 構造
- S3DeltaDirectTarget 構造
- S3HyperDirectTarget 構造
- S3IcebergDirectTarget 構造
- DirectSchemaChangePolicy 構造
- ApplyMapping 構造
- Mapping 構造
- SelectFields 構造
- DropFields 構造
- RenameField 構造
- スピゴット構造
- Join 構造
- JoinColumn 構造
- SplitFields 構造
- SelectFromCollection 構造
- FillMissingValues 構造

- Filter 構造
- FilterExpression 構造
- FilterValue 構造
- CustomCode 構造
- SparkSQL 構造
- SqlAlias 構造
- DropNullFields 構造
- NullCheckboxList 構造
- NullValueField 構造
- Datatype 構造
- Merge 構造
- Union 構造
- PIIDetection 構造
- Aggregate 構造
- DropDuplicates 構造
- GovernedCatalogTarget 構造
- GovernedCatalogSource 構造
- AggregateOperation 構造
- GlueSchema 構造
- GlueStudioSchemaColumn 構造
- GlueStudioColumn 構造
- DynamicTransform 構造
- TransformConfigParameter 構造
- EvaluateDataQuality 構造
- DQResultsPublishingOptions 構造
- DQStopJobOnFailureOptions 構造
- EvaluateDataQualityMultiFrame 構造
- Recipe 構造
- RecipeReference 構造
- SnowflakeNodeData 構造

- SnowflakeSource 構造
- SnowflakeTarget 構造
- ConnectorDataSource の構造
- ConnectorDataTarget の構造
- RecipeStep 構造
- RecipeAction 構造
- ConditionExpression 構造

- ジョブ API
  - ジョブ
    - データ型
    - Job 構造
    - ExecutionProperty 構造
    - NotificationProperty 構造
    - JobCommand 構造
    - ConnectionsList 構造
    - JobUpdate 構造
    - SourceControlDetails の構造
    - 操作
    - CreateJob アクション (Python: create_job)
    - UpdateJob アクション (Python: update_job)
    - GetJob アクション (Python: get_job)
    - GetJobs アクション (Python: get_jobs)
    - DeleteJob アクション (Python: delete_job)
    - ListJobs アクション (Python: list_jobs)
    - BatchGetJobs アクション (Python: batch_get_jobs)
  - ジョブ実行
    - データ型
    - JobRun の構造
    - Predecessor 構造
    - JobBookmarkEntry 構造

- BatchStopJobRunSuccessfulSubmission 構造
- BatchStopJobRunError 構造
- NotificationProperty 構造
- 操作
- StartJobRun アクション (Python: start_job_run)
- BatchStopJobRun アクション (Python: batch_stop_job_run)
- GetJobRun アクション (Python: get_job_run)
- GetJobRuns アクション (Python: get_job_runs)
- GetJobBookmark アクション (Python: get_job_bookmark)
- GetJobBookmarks アクション (Python: get_job_bookmarks)
- ResetJobBookmark アクション (Python: reset_job_bookmark)

- トリガー
  - データ型
  - トリガー構造
  - TriggerUpdate 構造
  - 述語構造
  - 条件の構造
  - アクション構造
  - EventBatchingCondition 構造
  - 操作
  - CreateTrigger アクション (Python: create_trigger)
  - StartTrigger アクション (Python: start_trigger)
  - GetTrigger アクション (Python: get_trigger)
  - GetTriggers アクション (Python: get_triggers)
  - UpdateTrigger アクション (Python: update_trigger)
  - StopTrigger アクション (Python: stop_trigger)
  - DeleteTrigger アクション (Python: delete_trigger)
  - ListTriggers アクション (Python: list_triggers)
  - BatchGetTriggers アクション (Python: batch_get_triggers)

- AWS Glue の統合 API

- データ型
- 統合構造
- IntegrationConfig 構造
- IntegrationPartition 構造
- IntegrationError 構造
- IntegrationFilter 構造
- InboundIntegration 構造
- SourceProcessingProperties 構造
- TargetProcessingProperties 構造
- SourceTableConfig 構造
- TargetTableConfig 構造
- 操作
- CreateIntegration アクション (Python: create_integration)
- ModifyIntegration アクション (Python: modify_integration)
- DescribeIntegrations アクション (Python: describe_integrations)
- DeleteIntegration アクション (Python: delete_integration)
- DescribeInboundIntegrations アクション (Python: describe_inbound_integrations)
- CreateIntegrationTableProperties アクション (Python: create_integration_table_properties)
- UpdateIntegrationTableProperties アクション (Python: update_integration_table_properties)
- GetIntegrationTableProperties アクション (Python: get_integration_table_properties)
- DeleteIntegrationTableProperties アクション (Python: delete_integration_table_properties)
- CreateIntegrationResourceProperty アクション (Python: create_integration_resource_property)
- UpdateIntegrationResourceProperty アクション (Python: update_integration_resource_property)
- GetIntegrationResourceProperty アクション (Python: get_integration_resource_property)
- UntagResource アクション (Python: untag_resource)
- ListTagsForResource アクション (Python: list_tags_for_resource)
- 例外
- ResourceNotFoundException structure
- InternalServerException 構造
- IntegrationAlreadyExistsFault 構造

- IntegrationConflictOperationFault 構造
- IntegrationQuotaExceededFault 構造
- KMSKeyNotAccessibleFault (構造)
- IntegrationNotFoundFault 構造
- TargetResourceNotFound 構造
- InvalidIntegrationStateFault 構造
- インタラクティブセッション API
  - データ型
  - セッション構造
  - SessionCommand の構造
  - Statement 構造
  - StatementOutput 構造
  - StatementOutputData 構造
  - ConnectionsList 構造
  - 操作
  - CreateSession アクション (Python: create_session)
  - StopSession アクション (Python: stop_session)
  - DeleteSession アクション (Python: delete_session)
  - GetSession アクション (Python: get_session)
  - ListSessions アクション (Python: list_sessions)
  - RunStatement アクション (Python: run_statement)
  - CancelStatement アクション (Python: cancel_statement)
  - GetStatement アクション (Python: get_statement)
  - ListStatements アクション (Python: list_statements)
- 開発エンドポイント API
  - データ型
  - DevEndpoint 構造
  - DevEndpointCustomLibraries 構造
  - 操作
  - CreateDevEndpoint アクション (Python: create_dev_endpoint)

- UpdateDevEndpoint アクション (Python: update_dev_endpoint)
- DeleteDevEndpoint アクション (Python: delete_dev_endpoint)
- GetDevEndpoint アクション (Python: get_dev_endpoint)
- GetDevEndpoints アクション (Python: get_dev_endpoints)
- BatchGetDevEndpoints アクション (Python: batch_get_dev_endpoints)
- ListDevEndpoints アクション (Python: list_dev_endpoints)
- スキーマレジストリ
  - データ型
  - RegistryId 構造
  - RegistryListItem 構造
  - MetadataInfo 構造
  - OtherMetadataValueListItem 構造
  - SchemaListItem 構造
  - SchemaVersionListItem 構造
  - MetadataKeyValuePair 構造
  - SchemaVersionErrorItem 構造
  - ErrorDetails 構造
  - SchemaVersionNumber 構造
  - SchemaId 構造
  - 操作
  - CreateRegistry アクション (Python: create_registry)
  - CreateSchema アクション (Python: create_schema)
  - GetSchema アクション (Python: get_schema)
  - ListSchemaVersions アクション (Python: list_schema_versions)
  - GetSchemaVersion アクション (Python: get_schema_version)
  - GetSchemaVersionsDiff アクション (Python: get_schema_versions_diff)
  - ListRegistries アクション (Python: list_registries)
  - ListSchemas アクション (Python: list_schemas)
  - RegisterSchemaVersion アクション (Python: register_schema_version)
  - UpdateSchema アクション (Python: update_schema)

- CheckSchemaVersionValidity アクション (Python: check_schema_version_validity)
- UpdateRegistry アクション (Python: update_registry)
- GetSchemaByDefinition アクション (Python: get_schema_by_definition)
- GetRegistry アクション (Python: get_registry)
- PutSchemaVersionMetadata アクション (Python: put_schema_version_metadata)
- QuerySchemaVersionMetadata アクション (Python: query_schema_version_metadata)
- RemoveSchemaVersionMetadata アクション (Python: remove_schema_version_metadata)
- DeleteRegistry アクション (Python: delete_registry)
- DeleteSchema アクション (Python: delete_schema)
- DeleteSchemaVersions アクション (Python: delete_schema_versions)

- ワークフロー
  - データ型
  - JobNodeDetails の構造
  - CrawlerNodeDetails 構造
  - TriggerNodeDetails 構造
  - Crawl 構造
  - Node 構造
  - Edge 構造
  - Workflow 構造
  - WorkflowGraph 構造
  - WorkflowRun 構造
  - WorkflowRunStatistics 構造
  - StartingEventBatchCondition 構造
  - Blueprint 構造
  - BlueprintDetails 構造
  - LastActiveDefinition 構造
  - BlueprintRun 構造
  - 操作
  - CreateWorkflow アクション (Python: create_workflow)
  - UpdateWorkflow アクション (Python: update_workflow)

- DeleteWorkflow アクション (Python: delete_workflow)
- GetWorkflow アクション (Python: get_workflow)
- ListWorkflows アクション (Python: list_workflows)
- BatchGetWorkflows アクション (Python: batch_get_workflows)
- GetWorkflowRun アクション (Python: get_workflow_run)
- GetWorkflowRuns アクション (Python: get_workflow_runs)
- GetWorkflowRunProperties アクション (Python: get_workflow_run_properties)
- PutWorkflowRunProperties アクション (Python: put_workflow_run_properties)
- CreateBlueprint アクション (Python: create_blueprint)
- UpdateBlueprint アクション (Python: update_blueprint)
- DeleteBlueprint アクション (Python: delete_blueprint)
- ListBlueprints アクション (Python: list_blueprints)
- BatchGetBlueprints アクション (Python: batch_get_blueprints)
- StartBlueprintRun アクション (Python: start_blueprint_run)
- GetBlueprintRun アクション (Python: get_blueprint_run)
- GetBlueprintRuns アクション (Python: get_blueprint_runs)
- StartWorkflowRun アクション (Python: start_workflow_run)
- StopWorkflowRun アクション (Python: stop_workflow_run)
- ResumeWorkflowRun アクション (Python: resume_workflow_run)
- 使用状況プロファイル
  - データ型
  - ProfileConfiguration structure
  - ConfigurationObject structure
  - UsageProfileDefinition structure
  - 操作
  - CreateUsageProfile action (Python: create_usage_profile)
  - GetUsageProfile action (Python: get_usage_profile)
  - UpdateUsageProfile action (Python: update_usage_profile)
  - DeleteUsageProfile action (Python: delete_usage_profile)
  - ListUsageProfiles action (Python: list_usage_profiles)

- 機械学習 API
  - データ型
  - TransformParameters 構造
  - EvaluationMetrics 構造
  - MLTransform 構造
  - FindMatchesParameters 構造
  - FindMatchesMetrics 構造
  - ConfusionMatrix 構造
  - GlueTable 構造
  - TaskRun 構造
  - TransformFilterCriteria 構造
  - TransformSortCriteria 構造
  - TaskRunFilterCriteria 構造
  - TaskRunSortCriteria 構造
  - TaskRunProperties 構造
  - FindMatchesTaskRunProperties 構造
  - ImportLabelsTaskRunProperties 構造
  - ExportLabelsTaskRunProperties 構造
  - LabelingSetGenerationTaskRunProperties 構造
  - SchemaColumn 構造
  - TransformEncryption 構造
  - MLUserDataEncryption 構造
  - ColumnImportance 構造
  - 操作
  - CreateMLTransform アクション (Python: create_ml_transform)
  - UpdateMLTransform アクション (Python: update_ml_transform)
  - DeleteMLTransform アクション (Python: delete_ml_transform)
  - GetMLTransform アクション (Python: get_ml_transform)
  - GetMLTransforms アクション (Python: get_ml_transforms)
  - ListMLTransforms アクション(Python: list_ml_transforms)

- StartMLEvaluationTaskRun アクション (Python: start_ml_evaluation_task_run)
- StartMLLabelingSetGenerationTaskRun アクション (Python: start_ml_labeling_set_generation_task_run)
- GetMLTaskRun アクション (Python: get_ml_task_run)
- GetMLTaskRuns アクション (Python: get_ml_task_runs)
- CancelMLTaskRun アクション (Python: cancel_ml_task_run)
- StartExportLabelsTaskRun アクション (Python: start_export_labels_task_run)
- StartImportLabelsTaskRun アクション (Python: start_import_labels_task_run)

- Data Quality API
  - データ型
  - DataSource 構造
  - DataQualityRulesetListDetails 構造
  - DataQualityTargetTable 構造
  - DataQualityRulesetEvaluationRunDescription 構造
  - DataQualityRulesetEvaluationRunFilter 構造
  - DataQualityEvaluationRunAdditionalRunOptions 構造
  - DataQualityRuleRecommendationRunDescription 構造
  - DataQualityRuleRecommendationRunFilter 構造
  - DataQualityResult 構造
  - DataQualityAnalyzerResult 構造
  - DataQualityObservation 構造
  - MetricBasedObservation 構造
  - DataQualityMetricValues 構造
  - DataQualityRuleResult 構造
  - DataQualityResultDescription 構造
  - DataQualityResultFilterCriteria 構造
  - DataQualityRulesetFilterCriteria 構造
  - DataQualityAggregatedMetrics 構造
  - StatisticAnnotation structure
  - TimestampedInclusionAnnotation structure

- AnnotationError 構造
- DatapointInclusionAnnotation structure
- StatisticSummaryList リスト
- StatisticSummary structure
- RunIdentifier 構造
- StatisticModelResult structure
- 操作
- StartDataQualityRulesetEvaluationRun アクション (Python: start_data_quality_ruleset_evaluation_run)
- CancelDataQualityRulesetEvaluationRun action (Python: cancel_data_quality_ruleset_evaluation_run)
- GetDataQualityRulesetEvaluationRun action (Python: get_data_quality_ruleset_evaluation_run)
- ListDataQualityRulesetEvaluationRuns action (Python: list_data_quality_ruleset_evaluation_runs)
- StartDataQualityRuleRecommendationRun action (Python: start_data_quality_rule_recommendation_run)
- CancelDataQualityRuleRecommendationRun action (Python: cancel_data_quality_rule_recommendation_run)
- GetDataQualityRuleRecommendationRun action (Python: get_data_quality_rule_recommendation_run)
- ListDataQualityRuleRecommendationRuns action (Python: list_data_quality_rule_recommendation_runs)
- GetDataQualityResult action (Python: get_data_quality_result)
- BatchGetDataQualityResult action (Python: batch_get_data_quality_result)
- ListDataQualityResults action (Python: list_data_quality_results)
- CreateDataQualityRuleset action (Python: create_data_quality_ruleset)
- DeleteDataQualityRuleset action (Python: delete_data_quality_ruleset)
- GetDataQualityRuleset action (Python: get_data_quality_ruleset)
- ListDataQualityRulesets action (Python: list_data_quality_rulesets)
- UpdateDataQualityRuleset アクション (Python: update_data_quality_ruleset)
- ListDataQualityRulesets action (Python: list_data_quality_statistics)
- TimestampFilter 構造

- CreateDataQualityRulesetRequest structure
- GetDataQualityRulesetResponse structure
- GetDataQualityResultResponse structure
- StartDataQualityRuleRecommendationRunRequest structure
- GetDataQualityRuleRecommendationRunResponse structure
- BatchPutDataQualityStatisticAnnotation action (Python: batch_put_data_quality_statistic_annotation)
- GetDataQualityModel アクション (Python: get_data_quality_model)
- GetDataQualityModelResult action (Python: get_data_quality_model_result)
- ListDataQualityStatisticAnnotations action (Python: list_data_quality_statistic_annotations)
- PutDataQualityProfileAnnotation action (Python: put_data_quality_profile_annotation)

- 機密データの検出 API
  - データ型
  - CustomentityType 構造
  - 操作
  - CreateCustomentType アクション (Python: create_custom_entity_type)
  - DeleteCustomEntityType アクション (Python: delete_custom_entity_type)
  - GetCustomEntityType アクション (Python: get_custom_entity_type)
  - BatchGetCustomEntityTypes アクション (Python: batch_get_custom_entity_types)
  - ListCustomentTypes アクション (Python: list_custom_entity_types)
- AWS Glue の API のタグ付け
  - データ型
  - Tag 構造
  - 操作
  - TagResource アクション (Python: tag_resource)
  - UntagResource アクション (Python: untag_resource)
  - GetTags アクション (Python: get_tags)
- 一般的なデータ型
  - Tag 構造
  - DecimalNumber 構造

- ErrorDetail 構造
- PropertyPredicate 構造
- ResourceUri 構造
- ColumnStatistics 構造
- ColumnStatisticsError 構造
- ColumnError 構造
- ColumnStatisticsData 構造
- BooleanColumnStatisticsData 構造
- DateColumnStatisticsData 構造
- DecimalColumnStatisticsData 構造
- DoubleColumnStatisticsData 構造
- LongColumnStatisticsData 構造
- StringColumnStatisticsData 構造
- BinaryColumnStatisticsData 構造
- 文字列パターン

- 例外
  - AccessDeniedException 構造
  - AlreadyExistsException 構造
  - ConcurrentModificationException 構造
  - ConcurrentRunsExceededException 構造
  - CrawlerNotRunningException 構造
  - CrawlerRunningException 構造
  - CrawlerStoppingException 構造
  - EntityNotFoundException 構造
  - FederationSourceException 構造
  - FederationSourceRetryableException 構造
  - GlueEncryptionException 構造
  - IdempotentParameterMismatchException 構造
  - IllegalWorkflowStateException 構造
  - InternalServiceException 構造

- InvalidExecutionEngineException 構造
- InvalidInputException 構造
- InvalidStateException 構造
- InvalidTaskStatusTransitionException 構造
- JobDefinitionErrorException 構造
- JobRunInTerminalStateException 構造
- JobRunInvalidStateTransitionException 構造
- JobRunNotInTerminalStateException 構造
- LateRunnerException 構造
- NoScheduleException 構造
- OperationTimeoutException 構造
- ResourceNotReadyException 構造
- ResourceNumberLimitExceededException 構造
- SchedulerNotRunningException 構造
- SchedulerRunningException 構造
- SchedulerTransitioningException 構造
- UnrecognizedRunnerException 構造
- ValidationException 構造
- VersionMismatchException 構造

AWS Glue のセキュリティ API

Security API では、セキュリティのデータ型と、AWS Glue のセキュリティに関する API について説明しています。

データ型

- DataCatalogEncryptionSettings 構造
- EncryptionAtRest 構造
- ConnectionPasswordEncryption 構造
- EncryptionConfiguration 構造
- S3Encryption 構造

- CloudWatchEncryption 構造
- JobBookmarksEncryption 構造
- SecurityConfiguration 構造
- GluePolicy の構造
- DataQualityEncryption 構造

DataCatalogEncryptionSettings 構造

データカタログのセキュリティを維持するための構成情報が含まれています。

フィールド

- `EncryptionAtRest` – EncryptionAtRest オブジェクト。

  データカタログの保管時の暗号化の構成を指定します。

- `ConnectionPasswordEncryption` – ConnectionPasswordEncryption オブジェクト。

  接続パスワードが有効になっている場合、データカタログでは、お客様が用意したキーを使用して、`CreateConnection` または `UpdateConnection` の一部としてパスワードを暗号化するか、接続プロパティの `ENCRYPTED_PASSWORD` フィールドに保存します。カタログの暗号化、またはパスワードの暗号化のみ有効にすることができます。

EncryptionAtRest 構造

データカタログの保管時の暗号化の構成を指定します。

フィールド

- `CatalogEncryptionMode` – 必須: UTF-8 文字列 (有効な値: `DISABLED` | `SSE-KMS="SSEKMS"` | `SSE-KMS-WITH-SERVICE-ROLE="SSEKMSWITHSERVICEROLE"`)。

  データカタログを暗号化するための保管時の暗号化モード。

- `SseAwsKmsKeyId` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  保管時の暗号化に使用する、AWS KMS キーの ID。

- `CatalogEncryptionServiceRole` – UTF-8 文字列。Custom string pattern #51 に一致。

呼び出し元に代わり、AWS Glue が Data Catalog オブジェクトを暗号化および復号するために引き受けるロール。

ConnectionPasswordEncryption 構造

CreateConnection または UpdateConnection の一部としてパスワードを暗号化し、接続プロパティの ENCRYPTED_PASSWORD フィールドに保存するために、データカタログで使用されるデータ構造。カタログの暗号化、またはパスワードの暗号化のみ有効にすることができます。

パスワードを含む CreationConnection リクエストが到着すると、Data Catalog ではまず AWS KMS キーを使用してパスワードを暗号化します。カタログの暗号化が有効になっている場合は、続いて全体の接続オブジェクトも暗号化します。

この暗号化では、セキュリティ要件に従ってパスワードキーへのアクセスを有効にしたり制限したりするように AWS KMS キーのアクセス許可を設定する必要があります。たとえば、管理者にのみ、パスワードキーの復号のアクセス許可を付与する場合などです。

フィールド

- ReturnConnectionPasswordEncrypted – 必須: Boolean。

  ReturnConnectionPasswordEncrypted フラグが「true」に設定されている場合は、GetConnection および GetConnections の応答でパスワードは暗号化された状態のままになります。この暗号化は、カタログの暗号化とは関係なく、有効になります。

- AwsKmsKeyId – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  接続パスワードの暗号化に使用する AWS KMS キー。

  接続パスワードの保護が有効になっている場合、CreateConnection および UpdateConnection の呼び出し元には少なくとも、指定された AWS KMS キーの kms:Encrypt アクセス許可が必要です。このアクセス許可は、Data Catalog に保存する前にパスワードを暗号化するために必要です。

  復号のアクセス許可を設定することで、セキュリティ要件に従ってパスワードキーへのアクセスを有効または制限することができます。

EncryptionConfiguration 構造

暗号化構成を指定します。

フィールド

- `S3Encryption` – S3Encryption オブジェクトの配列。

  Amazon Simple Storage Service (Amazon S3) データの暗号化設定。
- `CloudWatchEncryption` – CloudWatchEncryption オブジェクト。

  Amazon CloudWatch の暗号化構成。
- `JobBookmarksEncryption` – JobBookmarksEncryption オブジェクト。

  ジョブブックマークの暗号化構成。
- `DataQualityEncryption` – DataQualityEncryption オブジェクト。

  AWS Glue Data Quality アセットの暗号化設定。

S3Encryption 構造

Amazon Simple Storage Service (Amazon S3) データを暗号化する方法を指定します。

フィールド

- `S3EncryptionMode` – UTF-8 文字列 (有効な値: `DISABLED` | `SSE-KMS="SSEKMS"` | `SSE-S3="SSES3"`)。

  Simple Storage Service (Amazon S3) データに使用する暗号化モード。
- `KmsKeyArn` – UTF-8 文字列。Custom string pattern #29 に一致。

  データの暗号化に使用する KMS キーの Amazon リソースネーム (ARN)。

CloudWatchEncryption 構造

Amazon CloudWatch データを暗号化する方法を指定します。

フィールド

- `CloudWatchEncryptionMode` – UTF-8 文字列 (有効な値: `DISABLED` | `SSE-KMS="SSEKMS"`)。

CloudWatch データに使用する暗号化モード。

- KmsKeyArn – UTF-8 文字列。Custom string pattern #29 に一致。

  データの暗号化に使用する KMS キーの Amazon リソースネーム (ARN)。

JobBookmarksEncryption 構造

ジョブブックマークデータを暗号化する方法を指定します。

フィールド

- JobBookmarksEncryptionMode – UTF-8 文字列 (有効な値: DISABLED | CSE-KMS="CSEKMS")。

  ジョブブックマークデータに使用する暗号化モード。
- KmsKeyArn – UTF-8 文字列。Custom string pattern #29 に一致。

  データの暗号化に使用する KMS キーの Amazon リソースネーム (ARN)。

SecurityConfiguration 構造

セキュリティ設定を指定します。

フィールド

- Name – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  セキュリティ構成の名前。
- CreatedTimeStamp – タイムスタンプ。

  このセキュリティ設定が作成された時刻。
- EncryptionConfiguration – EncryptionConfiguration オブジェクト。

  このセキュリティ構成に関連する暗号化構成。

GluePolicy の構造

リソースポリシーを返すための構造。

フィールド

- PolicyInJson – UTF-8 文字列、少なくとも 2 バイト長。

  リクエストされたポリシードキュメントを JSON 形式で含んでいます。
- PolicyHash – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  このポリシーに関連付けられたハッシュ値を含んでいます。
- CreateTime – タイムスタンプ。

  ポリシーの作成日時。
- UpdateTime – タイムスタンプ。

  ポリシーの前回更新日時。

DataQualityEncryption 構造

アカウント内の Data Quality アセットを暗号化する方法を規定します。

フィールド

- DataQualityEncryptionMode – UTF-8 文字列 (有効な値: DISABLED | SSE-KMS="SSEKMS")。

  Data Quality アセットの暗号化に使用する暗号化モード。これらのアセットには、データ品質ルールセット、結果、統計、異常検出モデル、観測値が含まれます。

  有効な値は、カスタマーマネージド KMS キー を使用した暗号化の場合は SSEKMS、もしくは DISABLED です。
- KmsKeyArn – UTF-8 文字列。Custom string pattern #29 に一致。

  データの暗号化に使用する KMS キーの Amazon リソースネーム (ARN)。

操作

- GetDataCatalogEncryptionSettings アクション (Python: get_data_catalog_encryption_settings)
- PutDataCatalogEncryptionSettings アクション (Python: put_data_catalog_encryption_settings)
- PutResourcePolicy アクション (Python: put_resource_policy)
- GetResourcePolicy アクション (Python: get_resource_policy)

- DeleteResourcePolicy アクション (Python: delete_resource_policy)
- CreateSecurityConfiguration アクション (Python: create_security_configuration)
- DeleteSecurityConfiguration アクション (Python: delete_security_configuration)
- GetSecurityConfiguration アクション (Python: get_security_configuration)
- GetSecurityConfigurations アクション (Python: get_security_configurations)
- GetResourcePolicies アクション (Python: get_resource_policies)

GetDataCatalogEncryptionSettings アクション (Python: get_data_catalog_encryption_settings)

指定されたカタログのセキュリティ設定を取得します。

リクエスト

- `CatalogId` – カタログ ID 文字列、1 ~ 255 バイト長、「Single-line string pattern」に一致。

  セキュリティ設定を取得するデータカタログの ID。設定しない場合は、AWS アカウント ID がデフォルトで使用されます。

レスポンス

- `DataCatalogEncryptionSettings` – DataCatalogEncryptionSettings オブジェクト。

  リクエストされたセキュリティ設定。

エラー

- `InternalServiceException`
- `InvalidInputException`
- `OperationTimeoutException`

PutDataCatalogEncryptionSettings アクション (Python: put_data_catalog_encryption_settings)

指定したカタログのセキュリティ構成を設定します。構成が完了すると、以後、指定した暗号化がすべてのカタログ書き込みに適用されます。

リクエスト

- `CatalogId` – カタログ ID 文字列、1 ~ 255 バイト長、「Single-line string pattern」に一致。

  セキュリティ設定を行うデータカタログの ID。設定しない場合は、AWS アカウント ID がデフォルトで使用されます。
- `DataCatalogEncryptionSettings` – 必須: DataCatalogEncryptionSettings オブジェクト。

  設定するセキュリティ設定。

レスポンス

- 応答パラメータはありません。

エラー

- `InternalServiceException`
- `InvalidInputException`
- `OperationTimeoutException`

PutResourcePolicy アクション (Python: put_resource_policy)

アクセスコントロールのためにデータカタログのリソースポリシーを設定します。

リクエスト

- `PolicyInJson` – 必須: UTF-8 文字列、少なくとも 2 バイト長。

  設定する、JSON 形式のポリシードキュメントを含んでいます。
- `ResourceArn` – UTF-8 文字列。1 ~ 10240 バイト長。Custom string pattern #49 に一致。

  使用しません。内部使用のみ。

- PolicyHashCondition – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  前のポリシーが PutResourcePolicy を使用して設定された時に返るハッシュ値。その目的は、ポリシーの同時変更を防ぐことです。以前に設定されたポリシーがない場合は、このパラメータは使用しないでください。

- PolicyExistsCondition – UTF-8 文字列 (有効な値: MUST_EXIST | NOT_EXIST | NONE)。

  ポリシーの値を更新するために値 MUST_EXIST が使用されます。新しいポリシーを作成するために値 NOT_EXIST が使用されます。値 NONE または null 値を使用すると、呼び出しはポリシーが存在するかどうかに依存しなくなります。

- EnableHybrid – UTF-8 文字列 (有効な値: TRUE | FALSE)。

  'TRUE' の場合、Data Catalog のリソースへのクロスアカウントアクセスを許可するために次の両方の方法を使用していることを示しています。

  - リソースポリシーを PutResourePolicy で直接更新
  - AWS Management Console で Grant permissions コマンドを使用

  マネジメントコンソールを使用してクロスアカウントアクセスを既に許可している場合、'TRUE' に設定する必要があります。そうしないと、呼び出しは失敗します。デフォルトは FALSE です。

レスポンス

- PolicyHash – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  設定されたばかりのポリシーのハッシュです。このポリシーを上書きまたは更新するこれ以降の呼び出しには、これを含める必要があります。

エラー

- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException
- InvalidInputException
- ConditionCheckFailureException

GetResourcePolicy アクション (Python: get_resource_policy)

指定されたリソースポリシーを取得します。

リクエスト

- `ResourceArn` – UTF-8 文字列。1 ~ 10240 バイト長。Custom string pattern #49 に一致。

  リソースポリシーを取得する対象の AWS Glue リソースの ARN です。指定しない場合は、Data Catalog のリソースポリシーが返されます。`GetResourcePolicies` を使用して、既存のリソースポリシーをすべて表示します。詳細については、「AWS Glue リソース ARN を特定する」を参照してください。

レスポンス

- `PolicyInJson` – UTF-8 文字列、少なくとも 2 バイト長。

  リクエストされたポリシードキュメントを JSON 形式で含んでいます。
- `PolicyHash` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  このポリシーに関連付けられたハッシュ値を含んでいます。
- `CreateTime` – タイムスタンプ。

  ポリシーの作成日時。
- `UpdateTime` – タイムスタンプ。

  ポリシーの前回更新日時。

エラー

- `EntityNotFoundException`
- `InternalServiceException`
- `OperationTimeoutException`
- `InvalidInputException`

DeleteResourcePolicy アクション (Python: delete_resource_policy)

指定されたポリシーを削除します。

リクエスト

- PolicyHashCondition – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  このポリシーが設定されたときに返されたハッシュ値です。
- ResourceArn – UTF-8 文字列。1 ~ 10240 バイト長。Custom string pattern #49 に一致。

  削除されるリソースポリシーの AWS Glue リソースの ARNです。

レスポンス

- 応答パラメータはありません。

エラー

- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException
- InvalidInputException
- ConditionCheckFailureException

CreateSecurityConfiguration アクション (Python: create_security_configuration)

新しいセキュリティ構成を作成します。セキュリティ構成は、AWS Glue が使用できるセキュリティプロパティのセットです。セキュリティ構成を使用して、保管時にデータを暗号化できます。AWS Glue のセキュリティ設定を使用する方法については、「クローラ、ジョブ、および開発エンドポイントによって書き込まれたデータの暗号化」を参照してください。

リクエスト

- Name – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  新しいセキュリティ設定の名前。
- EncryptionConfiguration – 必須: EncryptionConfiguration オブジェクト。

  新しいセキュリティ設定に関連する暗号化設定。

レスポンス

- Name – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  新しいセキュリティ設定に割り当てられた名前。
- CreatedTimestamp – タイムスタンプ。

  新しいセキュリティ設定が作成された時刻。

エラー

- AlreadyExistsException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException
- ResourceNumberLimitExceededException

DeleteSecurityConfiguration アクション (Python: delete_security_configuration)

指定したセキュリティ設定を削除します。

リクエスト

- Name – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  削除するセキュリティ設定の名前。

レスポンス

- 応答パラメータはありません。

エラー

- EntityNotFoundException
- InvalidInputException
- InternalServiceException

- OperationTimeoutException

GetSecurityConfiguration アクション (Python: get_security_configuration)

指定したセキュリティ設定を取得します。

リクエスト

- Name – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  取得するセキュリティ設定の名前。

レスポンス

- SecurityConfiguration – SecurityConfiguration オブジェクト。

  リクエストされたセキュリティ設定。

エラー

- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException

GetSecurityConfigurations アクション (Python: get_security_configurations)

すべてのセキュリティ設定のリストを取得します。

リクエスト

- MaxResults – 1 ~ 1000 の数値 (整数)。

  返される結果の最大数。
- NextToken – UTF-8 文字列。

  継続トークン (これが継続呼び出しの場合)。

応答

- SecurityConfigurations – SecurityConfiguration オブジェクトの配列。

  セキュリティ設定のリスト。
- NextToken – UTF–8 文字列。

  返されるセキュリティ設定がさらにある場合は、継続トークンを返します。

エラー

- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException

GetResourcePolicies アクション (Python: get_resource_policies)

クロスアカウントのアクセス許可の付与中に AWS Resource Access Manager によって個々のリソースに設定されたリソースポリシーを取得します。Data Catalog のリソースポリシーも取得します。

Data Catalog 設定でメタデータの暗号化を有効にしていて、AWS KMS キーへのアクセス許可がない場合、オペレーションで Data Catalog のリソースポリシーを返すことはできません。

リクエスト

- NextToken – UTF-8 文字列。

  継続トークン (これが継続リクエストの場合)。
- MaxResults – 1 ~ 1000 の数値 (整数)。

  返されるリストの最大サイズ。

応答

- GetResourcePoliciesResponseList – GluePolicy オブジェクトの配列。

個々のリソースポリシーとアカウントレベルのリソースポリシーのリスト。

- NextToken – UTF–8 文字列。

  継続トークン (戻されたリストに使用可能な最後のリソースポリシーが含まれていない場合)。

エラー

- InternalServiceException
- OperationTimeoutException
- InvalidInputException
- GlueEncryptionException

カタログオブジェクト API

カタログオブジェクト API は、AWS Glue でのカタログの操作に関連するデータ型と API について説明します。

トピック

- カタログ API
- データベース API
- Table API
- パーティション API
- 接続 API
- ユーザー定義関数 API
- Athena カタログを AWS Glue にインポートする

カタログ API

「カタログ API」 では、カタログを作成、削除、検索、更新、一覧表示するための API について説明します。

データ型

- カタログの構造

- CatalogInput 構造
- TargetRedshiftCatalog 構造
- CatalogProperties 構造
- CatalogPropertiesOutput 構造
- DataLakeAccessProperties 構造
- DataLakeAccessPropertiesOutput 構造
- FederatedCatalog 構造

カタログの構造

カタログオブジェクトは、AWS Glue データカタログ内またはフェデレーティッドソース内のデータベースの論理グループを表します。Redshift フェデレーティッドカタログ、もしくは別のアカウントまたはリージョンにある Redshift データベースへのリソースリンクを含むカタログの作成が可能になりました。

フィールド

- `CatalogId` – カタログ ID 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  カタログの ID。デフォルトのカタログへのアクセスを許可するときは、このフィールドは指定しません。

- `Name` – 必須: UTF-8 文字列。1 ~ 64 バイト長。Custom string pattern #25 に一致。

  カタログの名前。アカウント ID と同じにすることはできません。

- `ResourceArn` – UTF–8 文字列。

  カタログリソースに割り当てられた Amazon リソースネーム (ARN)。

- `Description` – 説明文字列。2048 バイト長以下。URI address multi-line string pattern に一致。

  説明文字列。2,048 バイト以下。URI アドレスの複数行の文字列パターンに一致。カタログの説明。

- `Parameters` – キーバリューペアのマップ配列。

  各キーはキー文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  各値は UTF-8 文字列で、512,000 バイト長以下です。

カタログのパラメータとプロパティを定義する、キーと値のペアのマップ配列。

- `CreateTime` – タイムスタンプ。

  カタログが作成された時刻。

- `UpdateTime` – タイムスタンプ。

  カタログが最後に更新された時刻。

- `TargetRedshiftCatalog` – TargetRedshiftCatalog オブジェクト。

  データベースリソースのリンク用のターゲットカタログを記述する `TargetRedshiftCatalog` オブジェクト。

- `FederatedCatalog` – FederatedCatalog オブジェクト。

  AWS Glue データカタログの外部のエンティティを指す `FederatedCatalog` オブジェクト。

- `CatalogProperties` – CatalogPropertiesOutput オブジェクト。

  データレイクアクセスのプロパティとその他のカスタムプロパティを指定する `CatalogProperties` オブジェクト。

- `CreateTableDefaultPermissions` – PrincipalPermissions オブジェクトの配列。

  `PrincipalPermissions` オブジェクトの配列。プリンシパル用のデフォルトのアクセス権限セットをテーブルに作成します。AWS Lake Formation で使用します。AWS Glue オペレーションの通常の過程では使用されません。

- `CreateDatabaseDefaultPermissions` – PrincipalPermissions オブジェクトの配列。

  `PrincipalPermissions` オブジェクトの配列。プリンシパル用のデフォルトのアクセス権限セットをデータベースに作成します。AWS Lake Formation で使用します。AWS Glue オペレーションの通常の過程では使用されません。

- `AllowFullTableExternalDataAccess` – UTF-8 文字列 (有効な値: `True` | `False`)。

  Lake Formation に登録されている Amazon S3 の場所にあるデータへのアクセスをサードパーティーエンジンに許可します。

CatalogInput 構造

カタログプロパティを記述する構造。

フィールド

- Description – 説明文字列。2,048 バイト長以下。URI address multi-line string pattern に一致。

  説明文字列。2,048 バイト以下。URI アドレスの複数行の文字列パターンに一致。カタログの説明。

- FederatedCatalog – FederatedCatalog オブジェクト。

  FederatedCatalog オブジェクト。AWS Glue データカタログの外部のエンティティ (Redshift データベースなど) を参照する FederatedCatalog 構造。

- Parameters – キーバリューペアのマップ配列。

  各キーはキー文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  各値は UTF-8 文字列で、512,000 バイト長以下です。

  カタログのパラメータとプロパティを定義する、キーと値のペアのマップ配列。

- TargetRedshiftCatalog – TargetRedshiftCatalog オブジェクト。

  リソースのリンク用のターゲットカタログを記述する TargetRedshiftCatalog オブジェクト。

- CatalogProperties – CatalogProperties オブジェクト。

  データレイクアクセスのプロパティとその他のカスタムプロパティを指定する CatalogProperties オブジェクト。

- CreateTableDefaultPermissions – PrincipalPermissions オブジェクトの配列。

  PrincipalPermissions オブジェクトの配列。プリンシパル用のデフォルトのアクセス権限セットをテーブルに作成します。AWS Lake Formation で使用します。通常は空のリストとして明示的に設定する必要があります。

- CreateDatabaseDefaultPermissions – PrincipalPermissions オブジェクトの配列。

  PrincipalPermissions オブジェクトの配列。プリンシパル用のデフォルトのアクセス権限セットをデータベースに作成します。AWS Lake Formation で使用します。通常は空のリストとして明示的に設定する必要があります。

- AllowFullTableExternalDataAccess – UTF-8 文字列 (有効な値: True | False)。

  Lake Formation に登録されている Amazon S3 の場所にあるデータへのアクセスをサードパーティーエンジンに許可します。

TargetRedshiftCatalog 構造

リソースのリンク用のターゲットテーブルを記述する構造。

フィールド

- `CatalogArn` – 必須: UTF–8 文字列。

  カタログリソースの Amazon リソースネーム (ARN)。

CatalogProperties 構造

データレイクアクセスのプロパティとその他のカスタムプロパティを指定する構造。

フィールド

- `DataLakeAccessProperties` – DataLakeAccessProperties オブジェクト。

  AWS Glue データカタログ内のカタログリソースに対してデータレイクアクセスを設定するプロパティを指定する `DataLakeAccessProperties` オブジェクト。
- `CustomProperties` – キーバリューペアのマップ配列。

  各キーはキー文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  各値は UTF-8 文字列で、512,000 バイト長以下です。

  カタログの追加のキーと値のプロパティ。列統計の最適化など。

CatalogPropertiesOutput 構造

カタログリソースの設定プロパティを含むプロパティ属性。

フィールド

- `DataLakeAccessProperties` – DataLakeAccessPropertiesOutput オブジェクト。

  AWS Glue データカタログ内のカタログリソースに対してデータレイクアクセスを設定する入力プロパティを持つ `DataLakeAccessProperties` オブジェクト。
- `CustomProperties` – キーバリューペアのマップ配列。

  各キーはキー文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

各値は UTF-8 文字列で、512,000 バイト長以下です。

カタログの追加のキーと値のプロパティ。列統計の最適化など。

DataLakeAccessProperties 構造

AWS Glue データカタログ内のカタログリソースに対してデータレイクアクセスを設定する入力プロパティ。

フィールド

- `DataLakeAccess` – ブール。

  Amazon Athena、Amazon EMR、AWS Glue ETL など、Redshift 以外のエンジンからデータカタログ内の Amazon Redshift データベースにアクセスする、Apache Spark アプリケーションのデータレイクアクセスをオンまたはオフにします。
- `DataTransferRole`– UTF-8 文字列。Custom string pattern #51 に一致。

  クエリ中にステージングバケットとの間でデータを転送するために AWS Glue が引き受けるロール。
- `KmsKey` – UTF–8 文字列。

  カタログと併せて作成されるステージングバケットに使用される暗号化キー。
- `CatalogType` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  ネイティブのカタログリソースに対してフェデレーションカタログタイプを指定します。現在サポートされているタイプは `aws:redshift` です。

DataLakeAccessPropertiesOutput 構造

AWS Glue データカタログ内のカタログリソースに対するデータレイクアクセス設定の出力プロパティ。

フィールド

- `DataLakeAccess` – ブール。

  データカタログ内の Amazon Redshift データベースにアクセスする Apache Spark アプリケーションのデータレイクアクセスをオンまたはオフにします。

- `DataTransferRole`– UTF-8 文字列。Custom string pattern #51 に一致。

  クエリ中にステージングバケットとの間でデータを転送するために AWS Glue が引き受けるロール。

- `KmsKey` – UTF–8 文字列。

  カタログと併せて作成されるステージングバケットに使用される暗号化キー。

- `ManagedWorkgroupName` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  カタログリソース用に作成された Redshift Serverless マネージドコンピューティングの名前。

- `ManagedWorkgroupStatus` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  Redshift Serverless マネージドコンピューティングのステータス。

- `RedshiftDatabaseName` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  マネージドコンピューティングのデフォルトの Redshift データベースリソースの名前。

- `StatusMessage` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  マネージドワークグループのステータスに関する詳細情報を提供するメッセージ。

- `CatalogType` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  ネイティブのカタログリソースに対してフェデレーションカタログタイプを指定します。現在サポートされているタイプは `aws:redshift` です。

FederatedCatalog 構造

AWS Glue データカタログの外部のエンティティを指すカタログ。

フィールド

- `Identifier` - UTF-8 文字列。1 ~ 512 バイト長。Single-line string pattern に一致。

  フェデレーションカタログの一意の識別子。

- `ConnectionName` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  外部データソース (Redshift フェデレーティッドカタログなど) への接続の名前。

- `ConnectionType` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

フェデレーティッドカタログへのアクセスに使用される接続のタイプで、外部データソースへの接続のプロトコルまたは方法を指定します。

操作

- CreateCatalog アクション (Python: create_catalog)
- UpdateCatalog アクション (Python: update_catalog)
- DeleteCatalog アクション (Python: delete_catalog)
- GetCatalog アクション (Python: get_catalog)
- GetCatalogs アクション (Python: get_catalogs)

CreateCatalog アクション (Python: create_catalog)

AWS Glue データカタログに新しいカタログを作成します。

リクエスト

- `Name` – 必須: UTF-8 文字列。1 ~ 64 バイト長。Custom string pattern #25 に一致。

  作成するカタログの名前。
- `CatalogInput` – 必須: CatalogInput オブジェクト。

  カタログのメタデータを定義する `CatalogInput` オブジェクト。
- `Tags` – キーと値のペアのマップ配列。50 ペア以下。

  各キーは UTF-8 文字列で、1 ~ 128 バイト長です。

  各値は UTF-8 文字列で、256 バイト長以下です。

  キーバリューペアのマップ配列。50 ペア以下。各キーは UTF-8 文字列で、1 ~ 128 バイト長です。各値は UTF-8 文字列で、256 バイト長以下です。カタログに割り当てるタグ。

レスポンス

- 応答パラメータはありません。

エラー

- InvalidInputException
- AlreadyExistsException
- ResourceNumberLimitExceededException
- InternalServiceException
- OperationTimeoutException
- GlueEncryptionException
- ConcurrentModificationException
- AccessDeniedException
- EntityNotFoundException
- FederatedResourceAlreadyExistsException
- FederationSourceException

UpdateCatalog アクション (Python: update_catalog)

AWS Glue データカタログ内の既存のカタログのプロパティを更新します。

リクエスト

- CatalogId – 必須: カタログ ID 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  カタログの ID。
- CatalogInput – 必須: CatalogInput オブジェクト。

  既存のカタログの新しいプロパティを指定する CatalogInput オブジェクト。

レスポンス

- 応答パラメータはありません。

エラー

- EntityNotFoundException
- InvalidInputException

- InternalServiceException
- OperationTimeoutException
- GlueEncryptionException
- ConcurrentModificationException
- AccessDeniedException
- FederationSourceException

DeleteCatalog アクション (Python: delete_catalog)

指定されたカタログを AWS Glue データカタログから削除します。

このオペレーションが完了すると、削除されたカタログ内のデータベース、テーブル (それらのテーブルに属する可能性のあるすべてのテーブルのバージョンとパーティションも含む)、ユーザー定義の関数にアクセスできなくなります。AWS Glue では、このサービスの裁量により、これらの「孤立した」リソースを適時に非同期的に削除します。

関連するすべてのリソースを直ちに削除するには、DeleteCatalog を呼び出す前に、DeleteTableVersion (または BatchDeleteTableVersion)、DeletePartition (または BatchDeletePartition)、DeleteTable (または BatchDeleteTable)、DeleteUserDefinedFunction、DeleteDatabase を使用して、カタログに属するすべてのリソースを削除します。

リクエスト

- CatalogId – 必須: カタログ ID 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  カタログの ID。

レスポンス

- 応答パラメータはありません。

エラー

- EntityNotFoundException
- InvalidInputException

- InternalServiceException
- OperationTimeoutException
- GlueEncryptionException
- ConcurrentModificationException
- AccessDeniedException
- FederationSourceException

GetCatalog アクション (Python: get_catalog)

取得するカタログの名前。すべて小文字にします。

リクエスト

- CatalogId – 必須: カタログ ID 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  カタログが属する親カタログの ID。設定されていない場合は、デフォルトで AWS のアカウント番号が使用されます。

レスポンス

- Catalog – カタログ オブジェクト。

  Catalog オブジェクト。AWS Glue データカタログ内の指定されたカタログの定義。

エラー

- InvalidInputException
- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException
- GlueEncryptionException
- AccessDeniedException
- EntityNotFoundException
- FederationSourceException

- FederationSourceRetryableException

GetCatalogs アクション (Python: get_catalogs)

AWS Glue データカタログのカタログで定義されているすべてのカタログを取得します。Redshift フェデレーティッドカタログのユースケースの場合、このオペレーションは、Redshift 名前空間カタログ内の Redshift データベースにマッピングされたカタログのリストを返します。

リクエスト

- ParentCatalogId – カタログ ID 文字列、1 ~ 255 バイト長、Single-line string pattern に一致。

  カタログが属する親カタログの ID。設定されていない場合は、デフォルトで AWS のアカウント番号が使用されます。
- NextToken – UTF-8 文字列。

  継続トークン (これが継続呼び出しの場合)。
- MaxResults – 1 ~ 1000 の数値 (整数)。

  1 回の応答で返されるカタログの最大数。
- Recursive – ブール。

  カタログ階層内のすべてのカタログを ParentCatalogId から一覧表示するかどうか。デフォルトは false です。true の場合、ParentCatalogID 階層内のすべてのカタログオブジェクトがレスポンス内に一覧表示されます。
- IncludeRoot – ブール。

  アカウントとリージョンのデフォルトカタログをレスポンス内に一覧表示するかどうか。デフォルトは false です。true と ParentCatalogId = NULL | AWS Account ID の場合、すべてのカタログとデフォルトカタログがレスポンス内に一覧表示されます。

  ParentCatalogId が null に等しくなく、この属性が false または trueとして渡されると、InvalidInputException がスローされます。

レスポンス

- CatalogList – 必須: カタログ オブジェクトの配列。

  Catalog オブジェクトの配列。指定された親カタログの Catalog オブジェクトのリスト。

- NextToken – UTF-8 文字列。

  返されたトークンのリストをページ分割するための継続トークン。リストの現在のセグメントが最後のセグメントではない場合に返されます。

エラー

- InvalidInputException
- InternalServiceException
- OperationTimeoutException
- GlueEncryptionException
- AccessDeniedException
- EntityNotFoundException
- FederationSourceException
- FederationSourceRetryableException

データベース API

Database API では、データベースのデータ型について説明し、これにはデータベースを作成、削除、検索、更新、および一覧表示するための API が含まれます。

データ型

- データベース構造
- DatabaseInput 構造
- PrincipalPermissions 構造
- DataLakePrincipal 構造
- DatabaseIdentifier 構造
- FederatedDatabase 構造

データベース構造

Database オブジェクトは、Hive メタストアまたは RDBMS に存在する可能性のあるテーブルの論理グループを表します。

フィールド

- Name – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  データベースの名前。Hive との互換性を保つため、これは保存時には小文字で折りた畳まれます。

- Description – 説明文字列。2,048 バイト長以下。URI address multi-line string pattern に一致。

  データベースの説明。

- LocationUri – Uniform resource identifier (uri)。1 ~ 1024 バイト長。URI address multi-line string pattern に一致。

  データベースの場所 (たとえば HDFS パスなど)。

- Parameters – キーバリューペアのマップ配列。

  各キーはキー文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  各値は UTF-8 文字列で、512,000 バイト長以下です。

  これらのキー/値ペアはデータベースのパラメータとプロパティを定義します。

- CreateTime – タイムスタンプ。

  メタデータデータベースがカタログに作成された時刻。

- CreateTableDefaultPermissions – PrincipalPermissions オブジェクトの配列。

  プリンシパル用のデフォルトのアクセス権限セットをテーブルに作成します。AWS Lake Formation で使用されます。AWS Glue オペレーションの通常の過程では使用されません。

- TargetDatabase – DatabaseIdentifier オブジェクト。

  リソースリンク用のターゲットデータベースを記述する DatabaseIdentifier 構造。

- CatalogId – カタログ ID 文字列、1 ~ 255 バイト長、Single-line string pattern に一致。

  データベースが存在するデータカタログの ID。

- FederatedDatabase – FederatedDatabase オブジェクト。

  AWS Glue Data Catalog の外部のエンティティを参照する FederatedDatabase 構造。

DatabaseInput 構造

データベースの作成または更新に使用される構造体。

フィールド

- Name – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  データベースの名前。Hive との互換性を保つため、これは保存時には小文字で折りた畳まれます。
- Description – 説明文字列。2,048 バイト長以下。URI address multi-line string pattern に一致。

  データベースの説明。
- LocationUri – Uniform resource identifier (uri)。1 ~ 1024 バイト長。URI address multi-line string pattern に一致。

  データベースの場所 (たとえば HDFS パスなど)。
- Parameters – キーバリューペアのマップ配列。

  各キーはキー文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  各値は UTF-8 文字列で、512,000 バイト長以下です。

  これらのキー/値ペアはデータベースのパラメータとプロパティを定義します。

  これらのキー/値ペアはデータベースのパラメータとプロパティを定義します。
- CreateTableDefaultPermissions – PrincipalPermissions オブジェクトの配列。

  プリンシパル用のデフォルトのアクセス権限セットをテーブルに作成します。AWS Lake Formation で使用されます。AWS Glue オペレーションの通常の過程では使用されません。
- TargetDatabase – DatabaseIdentifier オブジェクト。

  リソースリンク用のターゲットデータベースを記述する DatabaseIdentifier 構造。
- FederatedDatabase – FederatedDatabase オブジェクト。

  AWS Glue Data Catalog の外部のエンティティを参照する FederatedDatabase 構造。

PrincipalPermissions 構造

プリンシパルに付与される許可です。

フィールド

- `Principal` – DataLakePrincipal オブジェクト。

  許可を付与されたプリンシパル。
- `Permissions` – UTF-8 文字列の配列。

  プリンシパルに付与される許可。

DataLakePrincipal 構造

AWS Lake Formation プリンシパルです。

フィールド

- `DataLakePrincipalIdentifier` – UTF-8 文字列、1 ~ 255 バイト長。

  AWS Lake Formation プリンシパルの識別子。

DatabaseIdentifier 構造

リソースリンク用のターゲットデータベースを記述する 構造。

フィールド

- `CatalogId` – カタログ ID 文字列、1 ~ 255 バイト長、「Single-line string pattern」に一致。

  データベースが存在するデータカタログの ID。
- `DatabaseName` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  カタログデータベースの名前。
- `Region` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  ターゲットデータベースのリージョン。

FederatedDatabase 構造

AWS Glue Data Catalog の外部のエンティティを指すデータベース。

フィールド

- `Identifier` – UTF-8 文字列。1 ~ 512 バイト長。Single-line string pattern に一致。

  フェデレーションデータベースの一意の識別子。

- `ConnectionName` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  外部メタストアへの接続の名前。

- `ConnectionType` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  JDBC、ODBC、またはその他のサポートされている接続プロトコルなど、フェデレーティッドデータベースへのアクセスに使用される接続のタイプ。

操作

- CreateDatabase アクション (Python: create_database)
- UpdateDatabase アクション (Python: update_database)
- DeleteDatabase アクション (Python: delete_database)
- GetDatabase アクション (Python: get_database)
- GetDatabases アクション (Python: get_databases)

CreateDatabase アクション (Python: create_database)

データカタログに新しいデータベースを作成します。

リクエスト

- `CatalogId` – カタログ ID 文字列、1 ~ 255 バイト長、「Single-line string pattern」に一致。

  データベースを作成するデータカタログの ID。設定しない場合は、AWS アカウント ID がデフォルトで使用されます。

- `DatabaseInput` – 必須: DatabaseInput オブジェクト。

  データベースのメタデータ。

- `Tags` – キーと値のペアのマップ配列。50 ペア以下。

  各キーは UTF-8 文字列で、1 ~ 128 バイト長です。

各値は UTF-8 文字列で、256 バイト長以下です。

データベースに割り当てるタグ。

レスポンス

- 応答パラメータはありません。

エラー

- `InvalidInputException`
- `AlreadyExistsException`
- `ResourceNumberLimitExceededException`
- `InternalServiceException`
- `OperationTimeoutException`
- `GlueEncryptionException`
- `ConcurrentModificationException`
- `FederatedResourceAlreadyExistsException`
- `FederationSourceException`
- `FederationSourceRetryableException`

UpdateDatabase アクション (Python: update_database)

データカタログの既存のデータベース定義を更新します。

リクエスト

- `CatalogId` – カタログ ID 文字列、1 ~ 255 バイト長、「Single-line string pattern」に一致。

  メタデータのデータベースが存在するデータカタログの ID。設定しない場合は、AWS アカウント ID がデフォルトで使用されます。
- `Name` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  カタログで更新するデータベースの名前。Hive 互換性のために、これは小文字で表記されます。
- `DatabaseInput` – 必須: DatabaseInput オブジェクト。

カタログ内のメタデータデータベースの新しい定義を指定する DatabaseInput オブジェクト。

レスポンス

- 応答パラメータはありません。

エラー

- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException
- GlueEncryptionException
- ConcurrentModificationException
- FederationSourceException
- FederationSourceRetryableException
- AlreadyExistsException

DeleteDatabase アクション (Python: delete_database)

指定されたデータベースをデータカタログから削除します。

> **Note**
>
> このオペレーションが完了すると、削除されたデータベース内のテーブル (それらのテーブルに属する可能性のあるすべてのテーブルのバージョンとパーティションも含む) とユーザー定義関数にはアクセスできなくなります。AWS Glue では、このサービスの裁量により、これらの「孤立した」リソースを適時に非同期的に削除します。
> 関連するすべてのリソースを確実にすぐに削除するには、DeleteDatabase を呼び出す前に、DeleteTableVersion/BatchDeleteTableVersion、DeletePartition/BatchDeletePar
> を使用して、データベースに属するすべてのリソースを削除します。

リクエスト

- CatalogId – カタログ ID 文字列、1～255 バイト長、「Single-line string pattern」に一致。

  データベースが存在するデータカタログの ID。設定しない場合は、AWS アカウント ID がデフォルトで使用されます。

- Name – 必須: UTF-8 文字列。1～255 バイト長。Single-line string pattern に一致。

  削除するデータベースの名前。Hive との互換性を保つため、名前はすべて小文字にする必要があります。

レスポンス

- 応答パラメータはありません。

エラー

- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException
- ConcurrentModificationException
- FederationSourceException
- FederationSourceRetryableException

GetDatabase アクション (Python: get_database)

指定されたデータベースの定義を取得します。

リクエスト

- CatalogId – カタログ ID 文字列、1～255 バイト長、「Single-line string pattern」に一致。

  データベースが存在するデータカタログの ID。設定しない場合は、AWS アカウント ID がデフォルトで使用されます。

- Name – 必須: UTF-8 文字列。1～255 バイト長。Single-line string pattern に一致。

取得するデータベースの名前。Hive 互換性のために、これはすべて小文字にする必要があります。

レスポンス

- Database – データベース オブジェクト。

  データカタログ内の指定されたデータベースの定義。

エラー

- InvalidInputException
- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException
- GlueEncryptionException
- FederationSourceException
- FederationSourceRetryableException

GetDatabases アクション (Python: get_databases)

指定されたデータカタログで定義されているすべてのデータベースを取得します。

リクエスト

- CatalogId – カタログ ID 文字列、1 ~ 255 バイト長、「Single-line string pattern」に一致。

  Databases を取得するデータカタログの ID。設定しない場合は、AWS アカウント ID がデフォルトで使用されます。
- NextToken – UTF-8 文字列。

  継続トークン (これが継続呼び出しの場合)。
- MaxResults – 数値 (integer)。1 ~ 100。

  1 回の応答で返すデータベースの最大数。
- ResourceShareType – UTF-8 文字列 (有効な値: FOREIGN | ALL | FEDERATED)。

アカウントと共有しているデータベースを一覧表示するように指定できます。指定できる値は、FEDERATED、FOREIGN または ALL です。

- FEDERATED に設定した場合は、アカウントと共有されているフェデレーションデータベース (外部エンティティを参照) が一覧表示されます。
- FOREIGN に設定した場合は、アカウントと共有されているデータベースが一覧表示されます。
- ALL に設定した場合は、アカウントと共有されているデータベースと、ローカルアカウントのデータベースが一覧表示されます。

- AttributesToGet – UTF-8 文字列の配列。

  GetDatabases 呼び出しによって返されるデータベースフィールドを指定します。このパラメータは空のリストを受け入れません。リクエストには NAME を含める必要があります。

レスポンス

- DatabaseList – 必須: データベース オブジェクトの配列。

  指定されたカタログの Database オブジェクトのリスト。

- NextToken – UTF-8 文字列。

  返されたトークンのリストをページ分割するための継続トークン。リストの現在のセグメントが最後のセグメントではない場合に返されます。

エラー

- InvalidInputException
- InternalServiceException
- OperationTimeoutException
- GlueEncryptionException
- EntityNotFoundException
- FederationSourceException
- FederationSourceRetryableException

Table API

テーブル API は、データ型とテーブルに関連する操作について説明しています。

データ型

- Table 構造
- TableInput 構造
- FederatedTable 構造
- 列の構造
- StorageDescriptor 構造
- SchemaReference 構造
- SerDeInfo 構造
- Order 構造
- SkewedInfo 構造
- TableVersion 構造
- TableError 構造
- TableVersionError 構造
- SortCriterion 構造
- TableIdentifier 構造
- KeySchemaElement 構造
- PartitionIndex 構造
- PartitionIndexDescriptor 構造
- BackfillError 構造
- IcebergInput 構造
- OpenTableFormatInput 構造
- ViewDefinition 構造
- ViewDefinitionInput 構造
- ViewRepresentation 構造
- ViewRepresentationInput 構造

- UpdateOpenTableFormatInput 構造
- UpdateIcebergInput 構造
- CreateIcebergTableInput 構造
- UpdateIcebergTableInput 構造
- IcebergSortOrder 構造
- IcebergSortField 構造
- IcebergPartitionSpec 構造
- IcebergPartitionField 構造
- IcebergSchema 構造
- IcebergStructField 構造
- IcebergTableUpdate 構造

Table 構造

列と行で構成されている関連データのコレクションを表します。

フィールド

- Name – 必須: UTF-8 文字列。1～255 バイト長。Single-line string pattern に一致。

  テーブルの名前。Hive 互換性のために、これはすべて小文字であることが必要です。
- DatabaseName – UTF-8 文字列。1～255 バイト長。Single-line string pattern に一致。

  テーブルメタデータが存在するデータベースの名前。Hive との互換性を保つため、名前はすべて小文字にする必要があります。
- Description – 説明文字列。2,048 バイト長以下。URI address multi-line string pattern に一致。

  テーブルの説明。
- Owner – UTF-8 文字列。1～255 バイト長。Single-line string pattern に一致。

  テーブルの所有者。
- CreateTime – タイムスタンプ。

  データカタログでテーブル定義が作成された時刻。
- UpdateTime – タイムスタンプ。

テーブルが最後に更新された時刻。

- `LastAccessTime` – タイムスタンプ。

  テーブルが最後にアクセスされた時刻。これは通常 HDFS から取得されるため、信頼性が低い場合があります。

- `LastAnalyzedTime` – タイムスタンプ。

  このテーブルの列統計が最後に計算された時刻。

- `Retention` – 数値 (整数)、None 以下。

  このテーブルの保持時間。

- `StorageDescriptor` – StorageDescriptor オブジェクト。

  このテーブルの物理ストレージに関する情報を含むストレージ記述子。

- `PartitionKeys` – Column オブジェクトの配列。

  テーブルがパーティション分割される列のリスト。パーティションキーとしてプリミティブ型のみがサポートされています。

  Amazon Athena によって使用されるテーブルを作成しており、`partitionKeys` を指定していない場合は必ず、`partitionKeys` の値を空のリストに設定する必要があります。例:

  ```
  "PartitionKeys": []
  ```

- `ViewOriginalText` - UTF-8 文字列。409,600 バイト長以下。

  Apache Hive との互換性のために含まれています。AWS Glue オペレーションの通常の過程では使用されません。テーブルが `VIRTUAL_VIEW` の場合、base64 でエンコードされた特定の Athena 設定です。

- `ViewExpandedText` - UTF-8 文字列。409,600 バイト長以下。

  Apache Hive との互換性のために含まれています。AWS Glue オペレーションの通常の過程では使用されません。

- `TableType` - UTF-8 文字列。255 バイト長以下。

  このテーブルのタイプ。AWS Glue では、`EXTERNAL_TABLE` タイプでテーブルが作成されます。Athena など、その他のサービスでは、テーブルタイプを追加してテーブルが作成されることがあります。

AWS Glue 関連テーブルタイプ:

EXTERNAL_TABLE

Hive 互換属性 - Hive 以外の管理テーブルを示します。

GOVERNED

AWS Lake Formation で使用されます。AWS Glue データカタログは GOVERNED を認識します。

- Parameters – キーバリューペアのマップ配列。

  各キーはキー文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  各値は UTF-8 文字列で、512,000 バイト長以下です。

  これらのキー/値ペアは、テーブルに関連付けられたプロパティを定義します。

- CreatedBy – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  テーブルを作成した個人または団体。

- IsRegisteredWithLakeFormation – ブール。

  テーブルが AWS Lake Formation に登録されているかどうかを示します。

- TargetTable – TableIdentifier オブジェクト。

  リソースリンク用のターゲットテーブルを記述する TableIdentifier 構造。

- CatalogId – カタログ ID 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  テーブルデータベースがあるデータカタログの ID。

- VersionId – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  テーブルバージョンの ID。

- FederatedTable – FederatedTable オブジェクト。

  AWS Glue Data Catalog の外部のエンティティを参照する FederatedTable 構造。

- ViewDefinition – ViewDefinition オブジェクト。

  ビューのダイアレクト、クエリなど、ビューを定義するすべての情報を含む構造。

- IsMultiDialectView – ブール。

ビューがさまざまなクエリエンジンの SQL ダイアレクトをサポートし、それらのエンジンで読み取れるようにするかどうかを指定します。

TableInput 構造

テーブルの定義に使用される構造。

フィールド

- Name – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  テーブルの名前。Hive との互換性を保つため、これは保存時には小文字で折りた畳まれます。
- Description – 説明文字列。2,048 バイト長以下。URI address multi-line string pattern に一致。

  テーブルの説明。
- Owner – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  テーブルの所有者。Apache Hive との互換性のために含まれています。AWS Glue オペレーションの通常の過程では使用されません。
- LastAccessTime – タイムスタンプ。

  テーブルが最後にアクセスされた時刻。
- LastAnalyzedTime – タイムスタンプ。

  このテーブルの列統計が最後に計算された時刻。
- Retention – 数値 (整数)、None 以下。

  このテーブルの保持時間。
- StorageDescriptor – StorageDescriptor オブジェクト。

  このテーブルの物理ストレージに関する情報を含むストレージ記述子。
- PartitionKeys – Column オブジェクトの配列。

  テーブルがパーティション分割される列のリスト。パーティションキーとしてプリミティブ型のみがサポートされています。

  Amazon Athena によって使用されるテーブルを作成しており、partitionKeys を指定していない場合は必ず、partitionKeys の値を空のリストに設定する必要があります。例:

```
"PartitionKeys": []
```

- `ViewOriginalText` - UTF-8 文字列。409,600 バイト長以下。

  Apache Hive との互換性のために含まれています。AWS Glue オペレーションの通常の過程では使用されません。テーブルが `VIRTUAL_VIEW` の場合、base64 でエンコードされた特定の Athena 設定です。

- `ViewExpandedText` - UTF-8 文字列。409,600 バイト長以下。

  Apache Hive との互換性のために含まれています。AWS Glue オペレーションの通常の過程では使用されません。

- `TableType` - UTF-8 文字列。255 バイト長以下。

  このテーブルのタイプ。AWS Glue では、`EXTERNAL_TABLE` タイプでテーブルが作成されます。Athena など、その他のサービスでは、テーブルタイプを追加してテーブルが作成されることがあります。

  AWS Glue 関連テーブルタイプ:

  EXTERNAL_TABLE

  Hive 互換属性 - Hive 以外の管理テーブルを示します。

  GOVERNED

  AWS Lake Formation で使用されます。AWS Glue データカタログは `GOVERNED` を認識します。

- `Parameters` – キーバリューペアのマップ配列。

  各キーはキー文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  各値は UTF-8 文字列で、512,000 バイト長以下です。

  これらのキー/値ペアは、テーブルに関連付けられたプロパティを定義します。

- `TargetTable` – TableIdentifier オブジェクト。

  リソースリンク用のターゲットテーブルを記述する `TableIdentifier` 構造。

- `ViewDefinition` – ViewDefinitionInput オブジェクト。

  ビューのダイアレクト、クエリなど、ビューを定義するすべての情報を含む構造。

FederatedTable 構造

AWS Glue Data Catalog の外部のエンティティを指すテーブル。

フィールド

- `Identifier` – UTF-8 文字列。1 ~ 512 バイト長。Single-line string pattern に一致。

  フェデレーションテーブルの一意の識別子。
- `DatabaseIdentifier` – UTF-8 文字列。1 ~ 512 バイト長。Single-line string pattern に一致。

  フェデレーションデータベースの一意の識別子。
- `ConnectionName` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  外部メタストアへの接続の名前。
- `ConnectionType` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  フェデレーティッドテーブルへのアクセスに使用される接続タイプで、外部データソースに接続するためのプロトコルまたは方法を指定します。

列の構造

`Table` 内の列。

フィールド

- `Name` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  `Column` の名前。
- `Type` – UTF-8 文字列。131,072 バイト長以下。Single-line string pattern に一致。

  `Column` のデータ型。
- `Comment` – コメント文字列。255 バイト長以下。Single-line string pattern に一致。

  自由形式のテキストコメント。
- `Parameters` – キーバリューペアのマップ配列。

  各キーはキー文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  各値は UTF-8 文字列で、512,000 バイト長以下です。

これらのキーバリューペアは、列に関連付けられたプロパティを定義します。

StorageDescriptor 構造

テーブルデータの物理ストレージについて説明します。

フィールド

- `Columns` – Column オブジェクトの配列。

  テーブル内の `Columns` のリストです。
- `Location` – 場所文字列。2,056 バイト長以下。URI address multi-line string pattern に一致。

  テーブルの物理的な場所。デフォルトでは、ウェアハウスの場所、その後にウェアハウス内のデータベースの場所、その後にテーブル名が続く形式になります。
- `AdditionalLocations` – UTF-8 文字列の配列。

  Delta テーブルへのパスを指す場所のリスト。
- `InputFormat` – 書式設定文字列。128 バイト長以下。Single-line string pattern に一致。

  入力形式: `SequenceFileInputFormat` (バイナリ)、または `TextInputFormat`、もしくはカスタム形式。
- `OutputFormat` – 書式設定文字列。128 バイト長以下。Single-line string pattern に一致。

  出力形式: `SequenceFileOutputFormat` (バイナリ)、または `IgnoreKeyTextOutputFormat`、もしくはカスタム形式。
- `Compressed` – ブール。

  テーブルのデータが圧縮されている場合は `True`、圧縮されていない場合は `False`。
- `NumberOfBuckets` – 数値 (整数)。

  テーブルにディメンション列が含まれている場合、指定する必要があります。
- `SerdeInfo` – SerDeInfo オブジェクト。

  シリアライズ/デシリアライズ (SerDe) 情報。
- `BucketColumns` – UTF-8 文字列の配列。

  テーブルのリデューサーグループ化列、クラスター列、およびバケット列のリスト。

- SortColumns – 注文 オブジェクトの配列。

  テーブル内の各バケットのソート順を指定するリスト。

- Parameters – キーバリューペアのマップ配列。

  各キーはキー文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  各値は UTF-8 文字列で、512,000 バイト長以下です。

  ユーザーが指定したキーバリュー形式のプロパティ。

- SkewedInfo – SkewedInfo オブジェクト。

  列に高い頻度で表示される値 (歪んだ値) に関する情報。

- StoredAsSubDirectories – ブール。

  テーブルデータがサブディレクトリに保管されている場合は True、保管されていない場合は False。

- SchemaReference – SchemaReference オブジェクト。

  AWS Glue スキーマレジストリに保存されているスキーマを参照するオブジェクト。

  テーブルを作成するときにスキーマに空の列リストを渡し、代わりにスキーマリファレンスを使用します。

SchemaReference 構造

AWS Glue スキーマレジストリに保存されているスキーマを参照するオブジェクト。

フィールド

- SchemaId – SchemaId オブジェクト。

  スキーマアイデンティティフィールドを含む構造。こちらか SchemaVersionId のいずれかを指定する必要があります。

- SchemaVersionId – UTF-8 文字列。36 バイト長。Custom string pattern #44 に一致。

  スキーマのバージョンに割り当てられた一意の ID。こちらか SchemaId のいずれかを指定する必要があります。

- SchemaVersionNumber – 数値 (long)。1 ~ 100000。

スキーマのバージョン番号。

SerDeInfo 構造

エクストラクターおよびローダーとして機能するシリアライズ/デシリアライズプログラム (SerDe) に関する情報。

フィールド

- Name – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  SerDe 名。

- SerializationLibrary – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  通常、SerDe を実装するクラス。例は org.apache.hadoop.hive.serde2.columnar.ColumnarSerDe です。

- Parameters – キーバリューペアのマップ配列。

  各キーはキー文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  各値は UTF-8 文字列で、512,000 バイト長以下です。

  これらのキーバリューペアは SerDe の初期化パラメータを定義します。

Order 構造

ソートされた列のソート順を指定します。

フィールド

- Column – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  列の名前。

- SortOrder – 必須: 数値 (integer)。1 以下。

  列が昇順 (== 1) または降順 (==0) でソートされていることを指定します。

SkewedInfo 構造

テーブルで歪んだ値を指定します。「歪んだ値」とは、非常に高い頻度で発生する値です。

フィールド

- SkewedColumnNames – UTF-8 文字列の配列。

  歪んだ値を含む列名のリスト。
- SkewedColumnValues – UTF-8 文字列の配列。

  頻繁に出現するため、歪んだとみなされる値のリスト。
- SkewedColumnValueLocationMaps – キーバリューペアのマップ配列。

  各キーは UTF-8 文字列。

  各値は UTF-8 文字列。

  歪んだ値が含まれている列へのマッピング。

TableVersion 構造

テーブルのバージョンを指定します。

フィールド

- Table – Table オブジェクト。

  該当するテーブル。
- VersionId – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  このテーブルのバージョンを特定する ID 値。VersionId は整数の文字列表現です。各バージョンは 1 ずつ増加します。

TableError 構造

テーブル操作に対するエラーレコード。

フィールド

- TableName – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

テーブルの名前。Hive 互換性のために、これはすべて小文字であることが必要です。

- `ErrorDetail` – ErrorDetail オブジェクト。

  エラーに関する詳細。

TableVersionError 構造

テーブルバージョン操作に対するエラーレコード。

フィールド

- `TableName` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  該当するテーブルの名前。
- `VersionId` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  該当するバージョンの ID 値。`VersionID` は整数の文字列表現です。各バージョンは 1 ずつ増加します。
- `ErrorDetail` – ErrorDetail オブジェクト。

  エラーに関する詳細。

SortCriterion 構造

ソート基準となるフィールドとソート順を指定します。

フィールド

- `FieldName` – 値の文字列の長さ。1 ~ 1024 バイト長。

  ソートするフィールドの名前。
- `Sort` – UTF-8 文字列 (有効な値: `ASC="ASCENDING"` | `DESC="DESCENDING"`)。

  昇順または降順のソート。

TableIdentifier 構造

リソースリンク用のターゲットテーブルを記述する構造。

フィールド

- CatalogId – カタログ ID 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  テーブルデータベースがあるデータカタログの ID。
- DatabaseName – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  ターゲットテーブルを含むカタログデータベースの名前。
- Name – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  ターゲットテーブルの名前。
- Region – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  ターゲットテーブルのリージョン。

KeySchemaElement 構造

名前とタイプで構成されるパーティションキーペア。

フィールド

- Name – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  パーティションキーの名前。
- Type – 必須: UTF-8 文字列。131072 バイト長以下。Single-line string pattern に一致。

  パーティションキーのタイプ。

PartitionIndex 構造

パーティションインデックスの構造。

フィールド

- Keys – 必須: UTF-8 文字列の配列。1 個の以上の文字列。

  パーティションインデックスのキー。
- IndexName – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  パーティションインデックスの名前。

PartitionIndexDescriptor 構造

テーブル内のパーティションインデックスの記述子。

フィールド

- `IndexName` – 必須: UTF-8 文字列。1～255 バイト長。Single-line string pattern に一致。

  パーティションインデックスの名前。
- `Keys` – 必須: KeySchemaElement オブジェクトの配列、少なくとも 1 つの構造体。

  `KeySchemaElement` 構造体としてのパーティションインデックスの 1 つ以上のキーのリスト。
- `IndexStatus` – 必須: UTF-8 文字列 (有効な値: `CREATING` | `ACTIVE` | `DELETING` | `FAILED`)。

  インデックスパーティションのステータス。

  次のようなステータスがあります。
  - CREATING: インデックスは作成中です。インデックスが CREATING 状態の場合、インデックスまたはそのテーブルは削除できません。
  - ACTIVE: インデックスの作成に成功しました。
  - FAILED: インデックスの作成に失敗しました。
  - DELETING: インデックスは、インデックスのリストから削除されます。
- `BackfillErrors` – BackfillError オブジェクトの配列。

  既存のテーブルのパーティションインデックスを登録するときに発生する可能性があるエラーのリスト。

BackfillError 構造

既存のテーブルのパーティションインデックスを登録するときに発生する可能性があるエラーのリスト。

これらのエラーでは、インデックスの登録が失敗した理由の詳細が表示され、応答として限られた数のパーティションが示されます。これにより、障害が発生したパーティションを修正し、インデックスの登録を再試行できます。発生する可能性のある最も一般的なエラーのセットは、次のように分類されます。

- EncryptedPartitionError: パーティションが暗号化されています。

- InvalidPartitionTypeDataError: パーティション値がそのパーティション列のデータ型と一致しません。
- MissingPartitionValueError: パーティションが暗号化されています。
- UnsupportedPartitionCharacterError: パーティション値内の文字がサポートされていません。例: U+0000、U+0001、U+0002。
- InternalError: 他のエラーコードに属さないすべてのエラー。

フィールド

- `Code` – UTF-8 文字列 (有効な値: `ENCRYPTED_PARTITION_ERROR` | `INTERNAL_ERROR` | `INVALID_PARTITION_TYPE_DATA_ERROR` | `MISSING_PARTITION_VALUE_ERROR` | `UNSUPPORTED_PARTITION_CHARACTER_ERROR`)。

  既存のテーブルのパーティションインデックスを登録するときに発生したエラーのエラーコード。
- `Partitions` – PartitionValueList オブジェクトの配列。

  応答内の限られた数のパーティションのリスト。

IcebergInput 構造

カタログ内に作成する Apache Iceberg メタデータテーブルを定義する構造。

フィールド

- `MetadataOperation` – 必須: UTF-8 文字列 (有効な値: `CREATE`)。

  必須のメタデータオペレーション。これは `CREATE` にのみ設定できます。
- `Version` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  Iceberg テーブルのテーブルバージョン。デフォルトは 2 です。
- `CreateIcebergTableInput` – CreateIcebergTableInput オブジェクト。

  テーブルプロパティやメタデータ仕様など、AWS Glue Data Catalog で新しい Iceberg テーブルを作成するために必要な設定パラメータ。

OpenTableFormatInput 構造

オープン形式のテーブルを表す構造。

フィールド

- `IcebergInput` – IcebergInput オブジェクト。

  Apache Iceberg メタデータテーブルを定義する `IcebergInput` 構造を指定します。

ViewDefinition 構造

表現の詳細を含む構造。

フィールド

- `IsProtected` – ブール。

  このフラグを true に設定すると、クエリ計画中にユーザー提供のオペレーションをビューの論理プランにプッシュしないようにエンジンに指示できます。ただし、このフラグを設定しても、エンジンが従うとは限りません。エンジンのドキュメントを参照して、提供される保証を理解してください。
- `Definer` - UTF-8 文字列。20 ~ 2,048 バイト長。Single-line string pattern に一致。

  SQL のビューの定義者。
- `SubObjects` - UTF-8 文字列の配列。文字列 10 個以下。

  テーブル Amazon リソースネーム (ARN) のリスト。
- `Representations` – ViewRepresentation オブジェクトの配列。1 ~ 1000 個の構造体。

  表現のリスト。

ViewDefinitionInput 構造

AWS Glue ビューを作成または更新するための詳細を含む構造。

フィールド

- `IsProtected` – ブール。

  このフラグを true に設定すると、クエリ計画中にユーザー提供のオペレーションをビューの論理プランにプッシュしないようにエンジンに指示できます。ただし、このフラグを設定しても、エンジンが従うとは限りません。エンジンのドキュメントを参照して、提供される保証を理解してください。

- `Definer` - UTF-8 文字列。20 ~ 2,048 バイト長。Single-line string pattern に一致。

  SQL のビューの定義者。
- `Representations` – ViewRepresentationInput オブジェクトの配列、1 ~ 10 個の構造。

  ビューのダイアレクトとビューを定義するクエリを含む構造のリスト。
- `SubObjects` - UTF-8 文字列の配列。文字列 10 個以下。

  ビューを構成するベーステーブルの ARN のリスト。

ViewRepresentation 構造

ビューのダイアレクトとビューを定義するクエリを含む構造。

フィールド

- `Dialect` – UTF-8 文字列 (有効な値: `REDSHIFT` | `ATHENA` | `SPARK`)。

  クエリエンジンのダイアレクト。
- `DialectVersion` – UTF-8 文字列、1 ~ 255 バイト長。

  クエリエンジンのダイアレクトのバージョン。3.0.0 などです。
- `ViewOriginalText` - UTF-8 文字列。409,600 バイト長以下。

  `CREATE VIEW` DDL 実行中に顧客が提供した `SELECT` クエリ。この SQL は、ビューのクエリ中には使用されません (`ViewExpandedText` が代わりに使用されます)。`ViewOriginalText` は、ユーザーがビューを作成した元の DDL コマンドを表示したい `SHOW CREATE VIEW` のような場合に使用されます。
- `ViewExpandedText` - UTF-8 文字列。409,600 バイト長以下。

  ビューの拡張された SQL。この SQL は、ビューでクエリを処理するときにエンジンによって使用されます。エンジンは、ビューの作成中にオペレーションを実行して `ViewOriginalText` を `ViewExpandedText` に変換する場合があります。例:
  - 完全修飾識別子: `SELECT * from table1 -> SELECT * from db1.table1`
- `ValidationConnection` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  ビューの特定の表現の検証に使用される接続の名前。
- `IsStale` – ブール。

古いとマークされたダイアレクトは無効になり、それぞれのクエリエンジンでクエリを実行する前に更新する必要があります。

ViewRepresentationInput 構造

Lake Formation ビューを更新または作成するための表現の詳細を含む構造。

フィールド

- `Dialect` – UTF-8 文字列 (有効な値: `REDSHIFT` | `ATHENA` | `SPARK`)。

  特定の表現のエンジンタイプを指定するパラメータ。

- `DialectVersion` – UTF-8 文字列、1 ~ 255 バイト長。

  特定の表現のエンジンのバージョンを指定するパラメータ。

- `ViewOriginalText` - UTF-8 文字列。409,600 バイト長以下。

  ビューを記述する元の SQL クエリを表す文字列。

- `ValidationConnection` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  ビューの特定の表現の検証に使用される接続の名前。

- `ViewExpandedText` - UTF-8 文字列。409,600 バイト長以下。

  ビューを記述する SQL クエリおよび拡張されたリソースの ARN を表す文字列

UpdateOpenTableFormatInput 構造

AWS Glue Data Catalog でオープンテーブル形式のテーブルを更新するための入力パラメータ。Apache Iceberg などの形式固有の更新操作のラッパーとして機能します。

フィールド

- `UpdateIcebergInput` – UpdateIcebergInput オブジェクト。

  スキーマの変更、パーティションの仕様、テーブルのプロパティなど、適用するテーブルの変更を定義する Apache Iceberg 固有の更新パラメータ。

UpdateIcebergInput 構造

既存の Iceberg テーブルに適用される更新操作を含む、AWS Glue Data Catalog の Apache Iceberg テーブルの更新に固有の入力パラメータ。

フィールド

- `UpdateIcebergTableInput` – 必須: UpdateIcebergTableInput オブジェクト。

  Iceberg テーブルに適用する特定の更新操作。スキーマ、パーティション、プロパティなど、テーブルの新しい状態を定義する更新のリストが含まれます。

CreateIcebergTableInput 構造

テーブルプロパティやメタデータ仕様など、AWS Glue Data Catalog で新しい Iceberg テーブルを作成するために必要な設定パラメータ。

フィールド

- `Location` – 必須: 場所文字列。2,056 バイト長以下。URI address multi-line string pattern に一致。

  Iceberg テーブルデータが保存される S3 ロケーション。
- `Schema` – 必須: IcebergSchema オブジェクト。

  Iceberg テーブルの構造、フィールドタイプ、メタデータを指定するスキーマ定義。
- `PartitionSpec` – IcebergPartitionSpec オブジェクト。

  最適なクエリパフォーマンスのために、Iceberg テーブルデータを整理およびパーティショニングする方法を定義するパーティショニング仕様。
- `WriteOrder` – IcebergSortOrder オブジェクト。

  クエリのパフォーマンスを最適化するために各パーティション内でデータを順序付ける方法を定義する、ソート順序の仕様。
- `Properties` – キーバリューペアのマップ配列。

  各キーは UTF-8 文字列。

  各値は UTF-8 文字列。

Iceberg テーブルの追加テーブルプロパティと設定のキーと値のペア。

UpdateIcebergTableInput 構造

AWS Glue Data Catalog の既存の Iceberg テーブルに適用する更新操作が含まれ、テーブルメタデータの新しい状態を定義します。

フィールド

- `Updates` – 必須: IcebergTableUpdate オブジェクトの配列。

  スキーマの変更、パーティション仕様、テーブルプロパティを含む、Iceberg テーブルに加える変更を指定するテーブル更新操作のリスト。

IcebergSortOrder 構造

Iceberg テーブルのソート順序仕様を定義し、クエリのパフォーマンスを最適化するためにパーティション内でデータを順序付ける方法を決定します。

フィールド

- `OrderId` – 必須: 数値 (integer)。

  Iceberg テーブルのメタデータ内のこのソート順序仕様の一意の識別子。

- `Fields` – 必須: IcebergSortField オブジェクトの配列。

  Iceberg テーブルデータの順序基準を定義する、フィールドとそのソート方向のリスト。

IcebergSortField 構造

ソースフィールド、変換、ソート方向、null 値の順序など、Iceberg ソート順序仕様内の単一のフィールドを定義します。

フィールド

- `SourceId` – 必須: 数値 (integer)。

  このソートフィールドの基礎となるテーブルスキーマのソースフィールドの識別子。

- `Transform` – 必須: UTF–8 文字列。

  ID、バケット、切り捨てなど、ソート前にソースフィールドに適用される変換関数。

- `Direction` – 必須: UTF-8 文字列 (有効な値: `asc="ASC"` | `desc="DESC"`)。

  フィールドのソート方向。昇順または降順のいずれか。

- `NullOrder` – 必須: UTF-8 文字列 (有効な値: `nulls-first="NULLS_FIRST"` | `nulls-last="NULLS_LAST"`)。

  このフィールドの null 値の順序付け動作。null をソート順で最初または最後に表示するかどうかを指定します。

IcebergPartitionSpec 構造

Iceberg テーブルのパーティショニング仕様を定義し、最適なクエリパフォーマンスのためにテーブルデータを整理およびパーティショニングする方法を決定します。

フィールド

- `Fields` – 必須: IcebergPartitionField オブジェクトの配列。

  ソースフィールドとその変換など、テーブルデータのパーティショニング方法を定義するパーティションフィールドのリスト。

- `SpecId` – 数値 (整数)。

  Iceberg テーブルのメタデータ履歴内にあるこのパーティション仕様の一意の識別子。

IcebergPartitionField 構造

ソースフィールド、変換関数、パーティション名、一意の識別子など、Iceberg パーティション仕様内の単一のパーティションフィールドを定義します。

フィールド

- `SourceId` – 必須: 数値 (integer)。

  このパーティションフィールドの基礎となるテーブルスキーマのソースフィールドの識別子。

- `Transform` – 必須: UTF–8 文字列。

ID、バケット、切り捨て、年、月、日、時間など、パーティションを作成するためにソースフィールドに適用される変換関数。

- Name – 必須: UTF-8 文字列。1 ~ 1024 バイト長。Single-line string pattern に一致。

  パーティショニングされたテーブル構造に表示されるパーティションフィールドの名前。

- FieldId – 数値 (整数)。

  Iceberg テーブルのパーティション仕様内でこのパーティションフィールドに割り当てられた一意の識別子。

IcebergSchema 構造

フィールド定義、データ型、スキーマメタデータなど、Iceberg テーブルのスキーマ構造を定義します。

フィールド

- SchemaId – 数値 (整数)。

  Iceberg テーブルのスキーマの進化履歴内にあるこのスキーマバージョンの一意の識別子。

- IdentifierFieldIds – 符号付き 32 ビット整数の配列。

  テーブル内のレコードを一意に識別するフィールド識別子のリスト。行レベルの操作と重複排除に使用されます。

- Type – UTF-8 文字列 (有効な値: struct="STRUCT")。

  スキーマ構造のルートタイプ。通常は Iceberg テーブルスキーマの「構造体」です。

- Fields – 必須: IcebergStructField オブジェクトの配列。

  フィールド名、タイプ、メタデータなど、テーブルスキーマを構成するフィールド定義のリスト。

IcebergStructField 構造

識別子、名前、データ型、null 値の許容性、ドキュメントなど、Iceberg テーブルスキーマ内の単一のフィールドを定義します。

フィールド

- `Id` – 必須: 数値 (integer)。

  Iceberg テーブルスキーマ内でこのフィールドに割り当てられた一意の識別子。スキーマの進化とフィールドの追跡に使用されます。

- `Name` – 必須: UTF-8 文字列。1 ~ 1024 バイト長。Single-line string pattern に一致。

  テーブルスキーマおよびクエリ操作に表示されるフィールドの名前。

- `Type` – 必須: `IcebergDocument` という名前の空の構造。

  このフィールドのデータ型定義。含まれるデータの構造と形式を指定します。

- `Required` – 必須: Boolean。

  テーブルスキーマでこのフィールドが必要 (null 値を許容しない) かオプション (null 値を許容する) かを示します。

- `Doc` – コメント文字列。255 バイト長以下。Single-line string pattern に一致。

  このフィールドの目的と使用に関する追加のコンテキストを提供する、オプションのドキュメントまたは説明テキスト。

IcebergTableUpdate 構造

スキーマの変更、パーティショニングの変更、ソート順の調整、場所の更新、プロパティの変更など、Iceberg テーブルに適用される更新の完全なセットを定義します。

フィールド

- `Schema` – 必須: IcebergSchema オブジェクト。

  フィールド構造、データ型、またはスキーマメタデータへの変更を指定する、Iceberg テーブルの更新されたスキーマ定義。

- `PartitionSpec` – IcebergPartitionSpec オブジェクト。

  テーブルデータを再編成してパーティショニングする方法を定義する、更新されたパーティショニング仕様。

- `SortOrder` – IcebergSortOrder オブジェクト。

クエリのパフォーマンスを最適化するためにパーティション内でデータを順序付ける方法を定義する、更新されたソート順序の仕様。

- `Location` – 必須: 場所文字列。2,056 バイト長以下。URI address multi-line string pattern に一致。

  Iceberg テーブルデータが保存される更新された S3 ロケーションを定義します。

- `Properties` – キーバリューペアのマップ配列。

  各キーは UTF-8 文字列。

  各値は UTF-8 文字列。

  Iceberg テーブルのテーブルプロパティと設定の更新されたキーと値のペア。

操作

- CreateTable アクション (Python: create_table)
- UpdateTable アクション (Python: update_table)
- DeleteTable アクション (Python: delete_table)
- BatchDeleteTable アクション (Python: batch_delete_table)
- GetTable アクション (Python:get_table)
- GetTables アクション (Python: get_tables)
- GetTableVersion アクション (Python: get_table_version)
- GetTableVersions アクション (Python: get_table_versions)
- DeleteTableVersion アクション (Python: delete_table_version)
- BatchDeleteTableVersion アクション (Python: batch_delete_table_version)
- SearchTables アクション (Python: search_tables)
- GetPartitionIndexes アクション (Python: get_partition_indexes)
- CreatePartitionIndex アクション (Python: create_partition_index)
- DeletePartitionIndex アクション (Python: delete_partition_index)
- GetColumnStatisticsForTable アクション (Python: get_column_statistics_for_table)
- UpdateColumnStatisticsForTable アクション (Python: update_column_statistics_for_table)
- DeleteColumnStatisticsForTable アクション (Python: delete_column_statistics_for_table)

CreateTable アクション (Python: create_table)

データカタログで新しいテーブル定義を作成します。

リクエスト

- `CatalogId` – カタログ ID 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  `Table` を作成するデータカタログの ID。提供されない場合は、AWS アカウント ID がデフォルトで使用されます。
- `DatabaseName` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  新しいテーブルを作成するカタログデータベース。Hive 互換性のために、この名前はすべて小文字であることが必要です。
- `Name` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  AWS Glue Data Catalog に作成される、指定されたデータベース内のテーブルの一意の識別子。
- `TableInput` – 必須: TableInput オブジェクト。

  カタログで作成するメタデータテーブルを定義する `TableInput` オブジェクト。
- `PartitionIndexes` – PartitionIndex オブジェクトの配列。構造 3 個以下。

  テーブル内に作成されるパーティションインデックスのリストである `PartitionIndex` 構造。
- `TransactionId` – UTF-8 文字列。1 ~ 255 バイト長。Custom string pattern #43 に一致。

  トランザクションの ID。
- `OpenTableFormatInput` – OpenTableFormatInput オブジェクト。

  オープン形式のテーブルを作成するときに `OpenTableFormatInput` 構造を指定します。

レスポンス

- 応答パラメータはありません。

エラー

- `AlreadyExistsException`
- `InvalidInputException`

- EntityNotFoundException
- ResourceNumberLimitExceededException
- InternalServiceException
- OperationTimeoutException
- GlueEncryptionException
- ConcurrentModificationException
- ResourceNotReadyException
- FederationSourceException
- FederationSourceRetryableException

UpdateTable アクション (Python: update_table)

データカタログのメタデータテーブルを更新します。

リクエスト

- CatalogId – カタログ ID 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  テーブルが存在するデータカタログの ID。設定しない場合は、AWS アカウント ID がデフォルトで使用されます。
- DatabaseName – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  テーブルが存在するカタログデータベースの名前。Hive 互換性のために、この名前はすべて小文字であることが必要です。
- Name – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  AWS Glue Data Catalog に作成される、指定されたデータベース内のテーブルの一意の識別子。
- TableInput – TableInput オブジェクト。

  カタログのメタデータテーブルを定義するための、更新された TableInput オブジェクト。
- SkipArchive – ブール。

  UpdateTable は、デフォルトでは、更新する前に常にテーブルのアーカイブされたバージョンを作成します。ただし、skipArchive が true に設定されている場合、UpdateTable ではアーカイブされたバージョンは作成されません。
- TransactionId – UTF-8 文字列。1 ~ 255 バイト長。Custom string pattern #43 に一致。

テーブルの内容を更新するトランザクション ID。

- VersionId – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  テーブルの内容を更新するバージョン ID。

- ViewUpdateAction – UTF-8 文字列 (有効な値: ADD | REPLACE | ADD_OR_REPLACE | DROP)。

  ビューの更新時に実行されるオペレーション。

- Force – ブール。

  一致するストレージ記述子とサブオブジェクト一致要件を無視するように true に設定できるフラグ。

レスポンス

- 応答パラメータはありません。

エラー

- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException
- ConcurrentModificationException
- ResourceNumberLimitExceededException
- GlueEncryptionException
- ResourceNotReadyException
- FederationSourceException
- FederationSourceRetryableException
- AlreadyExistsException

DeleteTable アクション (Python: delete_table)

データカタログからテーブル定義を削除します。

> **Note**
>
> このオペレーションが完了すると、削除されたテーブルに属するテーブルのバージョンとパーティションにはアクセスできなくなります。AWS Glue では、このサービスの裁量により、これらの「孤立した」リソースを適時に非同期的に削除します。
> すべての関連リソースをすぐに削除するには、`DeleteTableVersion` または `BatchDeleteTableVersion` を使用し、`DeletePartition` または `BatchDeletePartition` を使用して、テーブルに属するすべてのリソースを削除してから、`DeleteTable` を呼び出します。

リクエスト

- `CatalogId` – カタログ ID 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  テーブルが存在するデータカタログの ID。設定しない場合は、AWS アカウント ID がデフォルトで使用されます。

- `DatabaseName` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  テーブルが存在するカタログデータベースの名前。Hive 互換性のために、この名前はすべて小文字であることが必要です。

- `Name` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  削除するテーブルの名前。Hive 互換性のために、この名前はすべて小文字であることが必要です。

- `TransactionId` – UTF-8 文字列。1 ~ 255 バイト長。Custom string pattern #43 に一致。

  テーブルの内容を更新するトランザクション ID。

レスポンス

- 応答パラメータはありません。

エラー

- `EntityNotFoundException`
- `InvalidInputException`

- InternalServiceException
- OperationTimeoutException
- ConcurrentModificationException
- ResourceNotReadyException
- FederationSourceException
- FederationSourceRetryableException

BatchDeleteTable アクション (Python: batch_delete_table)

一度に複数のテーブルを削除します。

> **Note**
>
> このオペレーションが完了すると、削除されたテーブルに属するテーブルのバージョンとパーティションにはアクセスできなくなります。AWS Glue では、このサービスの裁量により、これらの「孤立した」リソースを適時に非同期的に削除します。
> すべての関連リソースをすぐに削除するには、DeleteTableVersion または BatchDeleteTableVersion を使用し、DeletePartition または BatchDeletePartition を使用して、テーブルに属するすべてのリソースを削除してから、BatchDeleteTable を呼び出します。

リクエスト

- CatalogId – カタログ ID 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  テーブルが存在するデータカタログの ID。設定しない場合は、AWS アカウント ID がデフォルトで使用されます。
- DatabaseName – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  削除するテーブルが存在するカタログデータベースの名前。Hive 互換性のために、この名前はすべて小文字であることが必要です。
- TablesToDelete – 必須: UTF-8 文字列の配列。文字列 100 個以下。

  削除するテーブルのリスト。
- TransactionId – UTF-8 文字列。1 ~ 255 バイト長。Custom string pattern #43 に一致。

テーブルの内容を更新するトランザクション ID。

応答

- `Errors` – TableError オブジェクトの配列。

  指定されたテーブルの削除試行中に発生したエラーのリスト。

エラー

- `InvalidInputException`
- `EntityNotFoundException`
- `InternalServiceException`
- `OperationTimeoutException`
- `GlueEncryptionException`
- `ResourceNotReadyException`

GetTable アクション (Python:get_table)

指定されたテーブルのデータカタログ内にある `Table` 定義を取得します。

リクエスト

- `CatalogId` – カタログ ID 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  テーブルが存在するデータカタログの ID。設定しない場合は、AWS アカウント ID がデフォルトで使用されます。
- `DatabaseName` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  テーブルが存在するカタログのデータベースの名前。Hive 互換性のために、この名前はすべて小文字であることが必要です。
- `Name` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  定義を取得するテーブルの名前です。Hive 互換性のために、この名前はすべて小文字であることが必要です。
- `TransactionId` – UTF-8 文字列。1 ~ 255 バイト長。Custom string pattern #43 に一致。

テーブルの内容を更新するトランザクション ID。

- `QueryAsOfTime` – タイムスタンプ。

  テーブルの内容を読み取る時点の時間。設定されていない場合は、最新のトランザクションコミット時間が使用されます。`TransactionId` と一緒に指定することはできません。

- `IncludeStatusDetails` – ブール。

  AWS Glue データカタログビューを作成または更新するリクエストに関連するステータス詳細を含めるかどうかを指定します。

レスポンス

- `Table` – Table オブジェクト。

  指定したテーブルを定義する `Table` オブジェクト。

エラー

- `EntityNotFoundException`
- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`
- `GlueEncryptionException`
- `ResourceNotReadyException`
- `FederationSourceException`
- `FederationSourceRetryableException`

GetTables アクション (Python: get_tables)

特定の `Database` 内の、一部またはすべてのテーブル定義を取得します。

リクエスト

- `CatalogId` – カタログ ID 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

テーブルが存在するデータカタログの ID。設定しない場合は、AWS アカウント ID がデフォルトで使用されます。

- DatabaseName – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  テーブルをリスト表示するカタログ内のデータベース。Hive 互換性のために、この名前はすべて小文字であることが必要です。

- Expression - UTF-8 文字列。2,048 バイト長以下。Single-line string pattern に一致。

  正規表現パターン。存在する場合、パターンに名前が一致するテーブルのみが返されます。

- NextToken – UTF–8 文字列。

  継続トークン (これが継続呼び出しの場合)。

- MaxResults – 数値 (integer)。1 ~ 100。

  1 回の応答で返されるテーブルの最大数。

- TransactionId – UTF-8 文字列。1 ~ 255 バイト長。Custom string pattern #43 に一致。

  テーブルの内容を更新するトランザクション ID。

- QueryAsOfTime – タイムスタンプ。

  テーブルの内容を読み取る時点の時間。設定されていない場合は、最新のトランザクションコミット時間が使用されます。TransactionId と一緒に指定することはできません。

- IncludeStatusDetails – ブール。

  AWS Glue データカタログビューを作成または更新するリクエストに関連するステータス詳細を含めるかどうかを指定します。

- AttributesToGet – UTF-8 文字列の配列。

  GetTables 呼び出しによって返されるテーブルフィールドを指定します。このパラメータは空のリストを受け入れません。リクエストには NAME を含める必要があります。

  有効な値の組み合わせは次のとおりです。

  - NAME - データベースにあるすべてのテーブルの名前。
  - NAME、TABLE_TYPE - すべてのテーブルとテーブルタイプの名前。

応答

- TableList – Table オブジェクトの配列。

  リクエストされた Table オブジェクトのリスト。
- NextToken – UTF-8 文字列。

  継続トークン (現在のリストセグメントが最後ではない場合に存在)。

エラー

- EntityNotFoundException
- InvalidInputException
- OperationTimeoutException
- InternalServiceException
- GlueEncryptionException
- FederationSourceException
- FederationSourceRetryableException

GetTableVersion アクション (Python: get_table_version)

テーブルの指定されたバージョンを取得します。

リクエスト

- CatalogId – カタログ ID 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  テーブルが存在するデータカタログの ID。設定しない場合は、AWS アカウント ID がデフォルトで使用されます。
- DatabaseName – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  テーブルが存在するカタログのデータベース。Hive 互換性のために、この名前はすべて小文字であることが必要です。
- TableName – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  テーブルの名前。Hive 互換性のために、この名前はすべて小文字であることが必要です。
- VersionId – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

取得するテーブルバージョンの ID 値。VersionID は整数の文字列表現です。各バージョンは 1 ずつ増加します。

レスポンス

- TableVersion – TableVersion オブジェクト。

  リクエストされたテーブルのバージョン。

エラー

- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException
- GlueEncryptionException

GetTableVersions アクション (Python: get_table_versions)

指定したテーブルで使用可能なバージョンを指定する文字列のリストを取得します。

リクエスト

- CatalogId – カタログ ID 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  テーブルが存在するデータカタログの ID。設定しない場合は、AWS アカウント ID がデフォルトで使用されます。
- DatabaseName – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  テーブルが存在するカタログのデータベース。Hive 互換性のために、この名前はすべて小文字であることが必要です。
- TableName – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  テーブルの名前。Hive 互換性のために、この名前はすべて小文字であることが必要です。
- NextToken – UTF–8 文字列。

  継続トークン (これが最初の呼び出しでない場合)。

- `MaxResults` – 数値 (integer)。1 ~ 100。

  1 回の応答で返すテーブルバージョンの最大数。

応答

- `TableVersions` – TableVersion オブジェクトの配列。

  指定したテーブルの使用可能なバージョンを指定する文字列のリスト。
- `NextToken` – UTF–8 文字列。

  継続トークン (使用可能なバージョンのリストに最後のバージョンが含まれていない場合)。

エラー

- `EntityNotFoundException`
- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`
- `GlueEncryptionException`

DeleteTableVersion アクション (Python: delete_table_version)

テーブルの指定されたバージョンを削除します。

リクエスト

- `CatalogId` – カタログ ID 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  テーブルが存在するデータカタログの ID。設定しない場合は、AWS アカウント ID がデフォルトで使用されます。
- `DatabaseName` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  テーブルが存在するカタログのデータベース。Hive 互換性のために、この名前はすべて小文字であることが必要です。
- `TableName` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  テーブルの名前。Hive 互換性のために、この名前はすべて小文字であることが必要です。

- VersionId – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  削除するテーブルバージョンの ID。VersionID は整数の文字列表現です。各バージョンは 1 ずつ増加します。

レスポンス

- 応答パラメータはありません。

エラー

- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException

BatchDeleteTableVersion アクション (Python: batch_delete_table_version)

テーブルの指定されたバージョンのバッチを削除します。

リクエスト

- CatalogId – カタログ ID 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  テーブルが存在するデータカタログの ID。設定しない場合は、AWS アカウント ID がデフォルトで使用されます。

- DatabaseName – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  テーブルが存在するカタログのデータベース。Hive 互換性のために、この名前はすべて小文字であることが必要です。

- TableName – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  テーブルの名前。Hive 互換性のために、この名前はすべて小文字であることが必要です。

- VersionIds – 必須: UTF-8 文字列の配列。文字列 100 個以下。

  削除するバージョンの ID リスト。VersionId は整数の文字列表現です。各バージョンは 1 ずつ増加します。

応答

- `Errors` – TableVersionError オブジェクトの配列。

  指定されたテーブルバージョンを削除しようとしているときに発生したエラーのリスト。

エラー

- `EntityNotFoundException`
- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`

SearchTables アクション (Python: search_tables)

テーブルメタデータのプロパティと親データベースに基づいて一連のテーブルを検索します。テキストまたはフィルタ条件で検索できます。

Lake Formation に定義されているセキュリティポリシーに基づいて、アクセスできるテーブルのみを取得できます。テーブルが返されるようにするには、少なくともテーブルへの読み取り専用アクセスが必要です。テーブル内の一部の列にアクセスできない場合、テーブルのリストが返されるときに、これらの列は検索されません。列にはアクセスできても、これら列内のデータにアクセスできない場合は、これらの列とこれらの列に関連付けられたメタデータが検索に含まれます。

リクエスト

- `CatalogId` – カタログ ID 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  一意の識別子 (*account_id* で構成)。
- `NextToken` – UTF–8 文字列。

  継続トークン (これが継続呼び出しの場合)。
- `Filters` – PropertyPredicate オブジェクトの配列。

  キーバリューペアのリスト、および検索結果のフィルタリングに使用する比較演算子。述語に一致するすべてのエンティティを返します。

  `PropertyPredicate` 構造体の `Comparator` メンバーは時間フィールドにのみ使用され、他のフィールドタイプでは省略できます。また、`Key=Name` など文字列値を比較する場合、

ファジーマッチアルゴリズムが使用されます。Key フィールド (例えば、Name フィールドの値) は、特定の句読点文字 (-、:、# など) でトークンに分割されます。次に、各トークンは PropertyPredicate の Value メンバーと正確にマッチで比較されます。例えば、Key=Name および Value=link の場合、customer-link や xx-link-yy という名前のテーブルは返されますが、xxlinkyy の場合は返されません。

- SearchText – 値の文字列の長さ。1 ~ 1024 バイト長。

  テキスト検索に使用する文字列。

  値を引用符で囲んで指定すると、値との完全一致に基づいてフィルタリングされます。

- SortCriteria – SortCriterion オブジェクトの配列。構造 1 個以下。

  結果をフィールド名で昇順または降順でソートするための条件のリスト。

- MaxResults – 1 ~ 1000 の数値 (整数)。

  1 回の応答で返されるテーブルの最大数。

- ResourceShareType – UTF-8 文字列 (有効な値: FOREIGN | ALL | FEDERATED)。

  アカウントと共有されているテーブルを検索するように指定できます。指定できる値は、FOREIGN または ALL です。

  - FOREIGN に設定されている場合、アカウントと共有されているテーブルが検索されます。
  - ALL に設定されている場合、アカウントと共有されているテーブルとローカルアカウントのテーブルが検索されます。

- IncludeStatusDetails – ブール。

  AWS Glue データカタログビューを作成または更新するリクエストに関連するステータス詳細を含めるかどうかを指定します。

レスポンス

- NextToken – UTF-8 文字列。

  継続トークン (現在のリストセグメントが最後ではない場合に存在)。

- TableList – Table オブジェクトの配列。

  リクエストされた Table オブジェクトのリスト。SearchTables レスポンスは、ユーザーがアクセスできるテーブルのみを返します。

エラー

- InternalServiceException
- InvalidInputException
- OperationTimeoutException

GetPartitionIndexes アクション (Python: get_partition_indexes)

テーブルに関連付けられたパーティションインデックスを取得します。

リクエスト

- CatalogId – カタログ ID 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  テーブルが存在するカタログの ID。
- DatabaseName – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  パーティションインデックスを取得するデータベースの名前を指定します。
- TableName – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  パーティションインデックスを取得するテーブルの名前を指定します。
- NextToken – UTF–8 文字列。

  継続トークン (これが継続呼び出しの場合)。

応答

- PartitionIndexDescriptorList – PartitionIndexDescriptor オブジェクトの配列。

  インデックス記述子のリスト。
- NextToken – UTF-8 文字列。

  継続トークン (現在のリストセグメントが最後ではない場合に存在)。

エラー

- InternalServiceException
- OperationTimeoutException

- InvalidInputException
- EntityNotFoundException
- ConflictException

CreatePartitionIndex アクション (Python: create_partition_index)

既存のテーブルに指定されたパーティションインデックスを作成します。

リクエスト

- CatalogId – カタログ ID 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  テーブルが存在するカタログの ID。
- DatabaseName – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  パーティションインデックスを作成するデータベースの名前を指定します。
- TableName – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  パーティションインデックスを作成するテーブルの名前を指定します。
- PartitionIndex – 必須: PartitionIndex オブジェクト。

  PartitionIndex 構造を指定して、既存のテーブルにパーティションインデックスを作成します。

レスポンス

- 応答パラメータはありません。

エラー

- AlreadyExistsException
- InvalidInputException
- EntityNotFoundException
- ResourceNumberLimitExceededException
- InternalServiceException
- OperationTimeoutException

- GlueEncryptionException

DeletePartitionIndex アクション (Python: delete_partition_index)

既存のテーブルから指定されたパーティションインデックスを削除します。

リクエスト

- CatalogId – カタログ ID 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  テーブルが存在するカタログの ID。
- DatabaseName – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  パーティションインデックスを削除するデータベースの名前を指定します。
- TableName – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  パーティションインデックスを削除するテーブルの名前を指定します。
- IndexName – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  削除するパーティションインデックスの名前。

レスポンス

- 応答パラメータはありません。

エラー

- InternalServiceException
- OperationTimeoutException
- InvalidInputException
- EntityNotFoundException
- ConflictException
- GlueEncryptionException

GetColumnStatisticsForTable アクション (Python: get_column_statistics_for_table)

列のテーブル統計を取得します。

このオペレーションに必要な Identity and Access Management (IAM) アクセス許可は `GetTable` です。

リクエスト

- `CatalogId` – カタログ ID 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  該当するパーティションが存在するデータカタログの ID。提供されない場合は、AWS アカウント ID がデフォルトで使用されます。
- `DatabaseName` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  パーティションが存在するカタログデータベースの名前。
- `TableName` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  パーティションのテーブルの名前。
- `ColumnNames` – 必須: UTF-8 文字列の配列。文字列 100 個以下。

  列名のリスト。

応答

- `ColumnStatisticsList` – ColumnStatistics オブジェクトの配列。

  ColumnStatistics のリスト。
- `Errors` – ColumnError オブジェクトの配列。

  取得に失敗した ColumnStatistics のリスト。

エラー

- `EntityNotFoundException`
- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`
- `GlueEncryptionException`

UpdateColumnStatisticsForTable アクション (Python: update_column_statistics_for_table)

列のテーブル統計を作成または更新します。

このオペレーションに必要な Identity and Access Management (IAM) アクセス許可は `UpdateTable` です。

リクエスト

- `CatalogId` – カタログ ID 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  該当するパーティションが存在するデータカタログの ID。提供されない場合は、AWS アカウント ID がデフォルトで使用されます。
- `DatabaseName` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  パーティションが存在するカタログデータベースの名前。
- `TableName` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  パーティションのテーブルの名前。
- `ColumnStatisticsList` – 必須: ColumnStatistics オブジェクトの配列。構造 25 個以下。

  列の統計のリスト。

応答

- `Errors` – ColumnStatisticsError オブジェクトの配列。

  ColumnStatisticsErrors のリスト。

エラー

- `EntityNotFoundException`
- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`
- `GlueEncryptionException`

DeleteColumnStatisticsForTable アクション (Python: delete_column_statistics_for_table)

列のテーブル統計を取得します。

このオペレーションに必要な Identity and Access Management (IAM) アクセス許可は `DeleteTable` です。

リクエスト

- `CatalogId` – カタログ ID 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  該当するパーティションが存在するデータカタログの ID。提供されない場合は、AWS アカウント ID がデフォルトで使用されます。

- `DatabaseName` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  パーティションが存在するカタログデータベースの名前。

- `TableName` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  パーティションのテーブルの名前。

- `ColumnName` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  列の名前。

レスポンス

- 応答パラメータはありません。

エラー

- `EntityNotFoundException`
- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`
- `GlueEncryptionException`

パーティション API

Partition API では、パーティションを使用するためのデータ型とオペレーションについて説明します。

データ型

- パーティションの構造
- PartitionInput の構造
- PartitionSpecWithSharedStorageDescriptor 構造
- PartitionListComposingSpec 構造
- PartitionSpecProxy 構造
- PartitionValueList 構造
- セグメント構造
- PartitionError 構造
- BatchUpdatePartitionFailureEntry 構造
- BatchUpdatePartitionRequestEntry 構造
- StorageDescriptor 構造
- SchemaReference 構造
- SerDeInfo 構造
- SkewedInfo 構造

パーティションの構造

テーブルデータのスライスを表します。

フィールド

- `Values` – UTF-8 文字列の配列。

  パーティションの値。

- `DatabaseName` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  パーティションを作成するカタログデータベースの名前。

- `TableName` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  パーティションを作成するデータベーステーブルの名前。
- `CreationTime` – タイムスタンプ。

  パーティションが作成された時刻。
- `LastAccessTime` – タイムスタンプ。

  パーティションが最後にアクセスされた時刻。
- `StorageDescriptor` – StorageDescriptor オブジェクト。

  パーティションが格納されている物理的な場所に関する情報を提供します。
- `Parameters` – キーバリューペアのマップ配列。

  各キーはキー文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  各値は UTF-8 文字列で、512,000 バイト長以下です。

  これらのキー/値ペアはパーティションパラメータを定義します。
- `LastAnalyzedTime` – タイムスタンプ。

  このパーティションの列統計が最後に計算された時刻。
- `CatalogId` – カタログ ID 文字列、1 ~ 255 バイト長、Single-line string pattern に一致。

  パーティションがある Data Catalog の ID。

PartitionInput の構造

パーティションの作成と更新に使用される構造。

フィールド

- `Values` – UTF-8 文字列の配列。

  パーティションの値。SDK ではこのパラメータは必要ありませんが、有効な入力としてこのパラメータを指定する必要があります。

  新しいパーティションのキーの値は、文字列オブジェクトの配列として渡す必要があります。この配列は、Amazon S3 プレフィックスに表示されるパーティションキーと同じ順序で順序付けする必要があります。そうしないと、AWS Glue では、間違ったキーに値が追加されます。

- `LastAccessTime` – タイムスタンプ。

  パーティションが最後にアクセスされた時刻。

- `StorageDescriptor` – StorageDescriptor オブジェクト。

  パーティションが格納されている物理的な場所に関する情報を提供します。

- `Parameters` – キーバリューペアのマップ配列。

  各キーはキー文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  各値は UTF-8 文字列で、512,000 バイト長以下です。

  これらのキー/値ペアはパーティションパラメータを定義します。

- `LastAnalyzedTime` – タイムスタンプ。

  このパーティションの列統計が最後に計算された時刻。

PartitionSpecWithSharedStorageDescriptor 構造

物理的な場所を共有するパーティションのパーティション仕様。

フィールド

- `StorageDescriptor` – StorageDescriptor オブジェクト。

  共有物理ストレージ情報。

- `Partitions` – パーティション オブジェクトの配列。

  この物理的な場所を共有するパーティションのリスト。

PartitionListComposingSpec 構造

関連するパーティションを一覧表示します。

フィールド

- `Partitions` – パーティション オブジェクトの配列。

  構成仕様のパーティションのリスト。

PartitionSpecProxy 構造

指定されたパーティションへのルートパスを提供します。

フィールド

- DatabaseName – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  パーティションが存在するカタログデータベース。
- TableName – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  パーティションを含むテーブルの名前。
- RootPath – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  パーティションをアドレス指定するためのプロキシのルートパス。
- PartitionSpecWithSharedSD – PartitionSpecWithSharedStorageDescriptor オブジェクト。

  同じ物理ストレージの場所を共有するパーティションの仕様。
- PartitionListComposingSpec – PartitionListComposingSpec オブジェクト。

  パーティションのリストを指定します。

PartitionValueList 構造

パーティションを定義する値のリストが含まれています。

フィールド

- Values – 必須: UTF-8 文字列の配列。

  値のリスト。

セグメント構造

テーブルのパーティションの重複しないリージョンを定義し、複数のリクエストを並行して実行できるようにします。

フィールド

- SegmentNumber – 必須: 数値 (整数)。None 以下。

セグメントのゼロベースのインデックス番号。たとえば、セグメントの合計数が 4 の場合、SegmentNumber 値の範囲は 0～3 です。

- TotalSegments – 必須: 数値 (整数)。1 ～ 10。

  セグメントの合計数。

PartitionError 構造

パーティションのエラーに関する情報が含まれます。

フィールド

- PartitionValues – UTF-8 文字列の配列。

  パーティションを定義する値。
- ErrorDetail – ErrorDetail オブジェクト。

  パーティションエラーの詳細。

BatchUpdatePartitionFailureEntry 構造

バッチ更新パーティションエラーに関する情報が含まれます。

フィールド

- PartitionValueList - UTF-8 文字列の配列、文字列 100 個以下。

  パーティションを定義する値のリスト。
- ErrorDetail – ErrorDetail オブジェクト。

  バッチ更新パーティションエラーの詳細。

BatchUpdatePartitionRequestEntry 構造

パーティションの更新に使用される値と構造体を含む構造体。

フィールド

- PartitionValueList – 必須: UTF-8 文字列の配列。文字列 100 個以下。

パーティションを定義する値のリスト。

- `PartitionInput` – 必須: PartitionInput オブジェクト。

  パーティションの更新に使用される構造体。

StorageDescriptor 構造

テーブルデータの物理ストレージについて説明します。

フィールド

- `Columns` – Column オブジェクトの配列。

  テーブル内の `Columns` のリストです。
- `Location` – 場所文字列。2,056 バイト長以下。URI address multi-line string pattern に一致。

  テーブルの物理的な場所。デフォルトでは、ウェアハウスの場所、その後にウェアハウス内のデータベースの場所、その後にテーブル名が続く形式になります。
- `AdditionalLocations` – UTF-8 文字列の配列。

  Delta テーブルへのパスを指す場所のリスト。
- `InputFormat` – 書式設定文字列。128 バイト長以下。Single-line string pattern に一致。

  入力形式: `SequenceFileInputFormat` (バイナリ)、または `TextInputFormat`、もしくはカスタム形式。
- `OutputFormat` – 書式設定文字列。128 バイト長以下。Single-line string pattern に一致。

  出力形式: `SequenceFileOutputFormat` (バイナリ)、または `IgnoreKeyTextOutputFormat`、もしくはカスタム形式。
- `Compressed` – ブール。

  テーブルのデータが圧縮されている場合は `True`、圧縮されていない場合は `False`。
- `NumberOfBuckets` – 数値 (整数)。

  テーブルにディメンション列が含まれている場合、指定する必要があります。
- `SerdeInfo` – SerDeInfo オブジェクト。

  シリアライズ/デシリアライズ (SerDe) 情報。

- BucketColumns – UTF-8 文字列の配列。

  テーブルのリデューサーグループ化列、クラスター列、およびバケット列のリスト。

- SortColumns – 注文 オブジェクトの配列。

  テーブル内の各バケットのソート順を指定するリスト。

- Parameters – キーバリューペアのマップ配列。

  各キーはキー文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  各値は UTF-8 文字列で、512,000 バイト長以下です。

  ユーザーが指定したキーバリュー形式のプロパティ。

- SkewedInfo – SkewedInfo オブジェクト。

  列に高い頻度で表示される値 (歪んだ値) に関する情報。

- StoredAsSubDirectories – ブール。

  テーブルデータがサブディレクトリに保管されている場合は True、保管されていない場合は False。

- SchemaReference – SchemaReference オブジェクト。

  AWS Glue スキーマレジストリに保存されているスキーマを参照するオブジェクト。

  テーブルを作成するときにスキーマに空の列リストを渡し、代わりにスキーマリファレンスを使用します。

SchemaReference 構造

AWS Glue スキーマレジストリに保存されているスキーマを参照するオブジェクト。

フィールド

- SchemaId – SchemaId オブジェクト。

  スキーマアイデンティティフィールドを含む構造。こちらか SchemaVersionId のいずれかを指定する必要があります。

- SchemaVersionId – UTF-8 文字列。36 バイト長。Custom string pattern #44 に一致。

スキーマのバージョンに割り当てられた一意の ID。こちらか SchemaId のいずれかを指定する必要があります。

- SchemaVersionNumber – 数値 (long)。1 ~ 100000。

  スキーマのバージョン番号。

SerDeInfo 構造

エクストラクターおよびローダーとして機能するシリアライズ/デシリアライズプログラム (SerDe) に関する情報。

フィールド

- Name – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  SerDe 名。

- SerializationLibrary – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  通常、SerDe を実装するクラス。例は org.apache.hadoop.hive.serde2.columnar.ColumnarSerDe です。

- Parameters – キーバリューペアのマップ配列。

  各キーはキー文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  各値は UTF-8 文字列で、512,000 バイト長以下です。

  これらのキーバリューペアは SerDe の初期化パラメータを定義します。

SkewedInfo 構造

テーブルで歪んだ値を指定します。「歪んだ値」とは、非常に高い頻度で発生する値です。

フィールド

- SkewedColumnNames – UTF-8 文字列の配列。

  歪んだ値を含む列名のリスト。

- SkewedColumnValues – UTF-8 文字列の配列。

頻繁に出現するため、歪んだとみなされる値のリスト。

- SkewedColumnValueLocationMaps – キーバリューペアのマップ配列。

  各キーは UTF-8 文字列。

  各値は UTF-8 文字列。

  歪んだ値が含まれている列へのマッピング。

操作

- CreatePartition アクション (Python: create_partition)
- BatchCreatePartition アクション (Python: batch_create_partition)
- UpdatePartition アクション (Python: update_partition)
- DeletePartition アクション (Python: delete_partition)
- BatchDeletePartition アクション (Python: batch_delete_partition)
- GetPartition アクション (Python: get_partition)
- GetPartitions アクション (Python: get_partitions)
- BatchGetPartition アクション (Python: batch_get_partition)
- BatchUpdatePartition アクション (Python: batch_update_partition)
- GetColumnStatisticsForPartition アクション (Python: get_column_statistics_for_partition)
- UpdateColumnStatisticsForPartition アクション (Python: update_column_statistics_for_partition)
- DeleteColumnStatisticsForPartition アクション (Python: delete_column_statistics_for_partition)

CreatePartition アクション (Python: create_partition)

新しいパーティションを作成します。

リクエスト

- CatalogId – カタログ ID 文字列、1 ~ 255 バイト長、「Single-line string pattern」に一致。

  パーティションを作成するカタログの AWS アカウント ID。

- DatabaseName – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

パーティションが作成されるメタデータデータベースの名前。

- TableName – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  パーティションが作成されるメタデータテーブルの名前。

- PartitionInput – 必須: PartitionInput オブジェクト。

  作成されるパーティションを定義する PartitionInput 構造。

レスポンス

- 応答パラメータはありません。

エラー

- InvalidInputException
- AlreadyExistsException
- ResourceNumberLimitExceededException
- InternalServiceException
- EntityNotFoundException
- OperationTimeoutException
- GlueEncryptionException

BatchCreatePartition アクション (Python: batch_create_partition)

バッチオペレーションで 1 つ以上のパーティションを作成します。

リクエスト

- CatalogId – カタログ ID 文字列、1 ~ 255 バイト長、「Single-line string pattern」に一致。

  パーティションを作成するカタログの ID。現在、これは AWS アカウント ID である必要があります。

- DatabaseName – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  パーティションが作成されるメタデータデータベースの名前。

- TableName – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

パーティションが作成されるメタデータテーブルの名前。

- `PartitionInputList` – 必須: PartitionInput オブジェクトの配列。構造 100 個以下。

  作成されるパーティションを定義する `PartitionInput` 構造のリスト。

応答

- `Errors` – PartitionError オブジェクトの配列。

  リクエストされたパーティションを作成しようとしたときにエラーが発生しました。

エラー

- `InvalidInputException`
- `AlreadyExistsException`
- `ResourceNumberLimitExceededException`
- `InternalServiceException`
- `EntityNotFoundException`
- `OperationTimeoutException`
- `GlueEncryptionException`

UpdatePartition アクション (Python: update_partition)

パーティションを更新します。

リクエスト

- `CatalogId` – カタログ ID 文字列、1 ~ 255 バイト長、「Single-line string pattern」に一致。

  更新されるパーティションが存在するデータカタログの ID。設定しない場合は、AWS アカウント ID がデフォルトで使用されます。

- `DatabaseName` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  該当する表が存在するカタログデータベースの名前。

- `TableName` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

更新するパーティションが存在するテーブルの名前。

- PartitionValueList – 必須: UTF-8 文字列の配列。文字列 100 個以下。

  更新するパーティションを定義するパーティションキー値のリスト。

- PartitionInput – 必須: PartitionInput オブジェクト。

  パーティションを更新する新しいパーティションオブジェクト。

  Values プロパティは変更できません。パーティションのパーティションキー値を変更する場合は、パーティションを削除して再作成します。

レスポンス

- 応答パラメータはありません。

エラー

- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException
- GlueEncryptionException

DeletePartition アクション (Python: delete_partition)

指定したパーティションを削除します。

リクエスト

- CatalogId – カタログ ID 文字列、1 ~ 255 バイト長、「Single-line string pattern」に一致。

  削除されるパーティションが存在するデータカタログの ID。設定しない場合は、AWS アカウント ID がデフォルトで使用されます。

- DatabaseName – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  該当する表が存在するカタログデータベースの名前。

- TableName – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  削除するパーティションを含むテーブルの名前。

- PartitionValues – 必須: UTF-8 文字列の配列。

  パーティションを定義する値。

レスポンス

- 応答パラメータはありません。

エラー

- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException

BatchDeletePartition アクション (Python: batch_delete_partition)

バッチオペレーションで 1 つ以上のパーティションを削除します。

リクエスト

- CatalogId – カタログ ID 文字列、1 ~ 255 バイト長、「Single-line string pattern」に一致。

  削除されるパーティションが存在するデータカタログの ID。設定しない場合は、AWS アカウント ID がデフォルトで使用されます。

- DatabaseName – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  該当する表が存在するカタログデータベースの名前。

- TableName – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  削除するパーティションを含むテーブルの名前。

- PartitionsToDelete – 必須: PartitionValueList オブジェクトの配列。構造 25 個以下。

  削除されるパーティションを定義する PartitionInput 構造のリスト。

応答

- `Errors` – PartitionError オブジェクトの配列。

  リクエストされたパーティションを削除しようとしたときにエラーが発生しました。

エラー

- `InvalidInputException`
- `EntityNotFoundException`
- `InternalServiceException`
- `OperationTimeoutException`

GetPartition アクション (Python: get_partition)

指定したパーティションに関する情報を取得します。

リクエスト

- `CatalogId` – カタログ ID 文字列、1 ~ 255 バイト長、「Single-line string pattern」に一致。

  対象のパーティションが存在する Data Catalog の ID。設定しない場合は、AWS アカウント ID がデフォルトで使用されます。
- `DatabaseName` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  パーティションが存在するカタログデータベースの名前。
- `TableName` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  パーティションのテーブルの名前。
- `PartitionValues` – 必須: UTF-8 文字列の配列。

  パーティションを定義する値。

レスポンス

- `Partition` – パーティション オブジェクト。

  Partition オブジェクトの形式で、リクエストされた情報。

エラー

- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException
- GlueEncryptionException
- FederationSourceException
- FederationSourceRetryableException

GetPartitions アクション (Python: get_partitions)

テーブルのパーティションについての情報を取得します。

リクエスト

- CatalogId – カタログ ID 文字列、1 ~ 255 バイト長、「Single-line string pattern」に一致。

  該当するパーティションが存在するデータカタログの ID。設定しない場合は、AWS アカウント ID がデフォルトで使用されます。
- DatabaseName – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  パーティションが存在するカタログデータベースの名前。
- TableName – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  パーティションのテーブルの名前。
- Expression – 述語文字列、2048 バイト長以下、URI address multi-line string pattern に一致。

  返されるパーティションをフィルタリングする式。

  この式では、SQL WHERE フィルタ句と似た SQL 構文を使用します。SQL ステートメントパーサー JSQLParser が式を解析します。

  演算子: Expression API 呼び出しで使用できる演算子を以下に示します。

  =

  2 つのオペランドの値が等しいかどうかを確認します。「はい」の場合、条件は true です。

例: 「変数 a」が 10、「変数 b」が 20 とします。

(a = b) は true ではありません。

< >

2 つのオペランドの値が等しいかどうかを確認します。値が等しくない場合、条件は true です。

例: (a < > b) は true です。

>

左のオペランドの値が右のオペランドの値よりも大きいかどうかを確認します。「はい」の場合、条件は true です。

例: (a > b) は true ではありません。

<

左のオペランドの値が右のオペランドの値よりも小さいかどうかを確認します。「はい」の場合、条件は true です。

例: (a < b) は true です。

>=

左のオペランドの値が右のオペランドの値以上かどうかを確認します。「はい」の場合、条件は true です。

例: (a >= b) は true ではありません。

<=

左のオペランドの値が右のオペランドの値以下かどうかを確認します。「はい」の場合、条件は true です。

例: (a <= b) は true です。

AND、OR、IN、BETWEEN、LIKE、NOT、IS NULL

論理演算子。

サポートされているパーティションキーの型: サポートされているパーティションキーを以下に示します。

- string
- date
- timestamp
- int
- bigint
- long
- tinyint
- smallint
- decimal

無効なタイプが存在した場合は、例外が発生します。

各タイプの有効な演算子のリストを次に示します。クローラを定義する場合、partitionKey タイプはカタログパーティションとの互換性を確保するため STRING として作成されます。

サンプル API コール:

Example

このテーブル twitter_partition には 3 つのパーティションがあります。

```
year = 2015
        year = 2016
        year = 2017
```

Example

パーティション year を 2015 と等しくする

```
aws glue get-partitions --database-name dbname --table-name twitter_partition
        --expression "year*=*'2015'"
```

Example

パーティション year を 2016-2018 (これらの値を含まない) にする

```
aws glue get-partitions --database-name dbname --table-name twitter_partition
```

```
        --expression "year>'2016' AND year<'2018'"
```

Example

パーティション year を 2015-2018 (これらの値を含む) にする 次の API コールは、相互に等しくなります。

```
aws glue get-partitions --database-name dbname --table-name twitter_partition
        --expression "year>='2015' AND year<='2018'"

        aws glue get-partitions --database-name dbname --table-name
 twitter_partition
        --expression "year BETWEEN 2015 AND 2018"

        aws glue get-partitions --database-name dbname --table-name
 twitter_partition
        --expression "year IN (2015,2016,2017,2018)"
```

Example

ワイルドカードパーティションフィルタ。次のコール出力がパーティション year=2017 になります。正規表現は LIKE ではサポートされません。

```
aws glue get-partitions --database-name dbname --table-name twitter_partition
        --expression "year LIKE '%7'"
```

- NextToken – UTF–8 文字列。

  これらのパーティションを取得する最初の呼び出しでない場合は、継続トークン。

- Segment – Segment オブジェクト。

  このリクエストでスキャンするテーブルのパーティションのセグメント。

- MaxResults – 1 ~ 1000 の数値 (整数)。

  1 回の応答で返されるパーティションの最大数。

- ExcludeColumnSchema – ブール。

  true の場合、パーティション列スキーマを返さないことを指定します。パーティション値や場所など、他のパーティション属性にのみ関心がある場合に便利です。この方法では、重複データを返さないことで、大きな応答の問題を回避できます。

- TransactionId – UTF-8 文字列。1 ~ 255 バイト長。Custom string pattern #43 に一致。

  テーブルの内容を更新するトランザクション ID。
- QueryAsOfTime – タイムスタンプ。

  パーティションの内容を読み取る時点の時間。設定されていない場合は、最新のトランザクションコミット時間が使用されます。TransactionId と一緒に指定することはできません。

応答

- Partitions – パーティション オブジェクトの配列。

  リクエストされたパーティションのリスト。
- NextToken – UTF–8 文字列。

  継続トークン (戻されたパーティションのリストに最後のパーティションが含まれていない場合)。

エラー

- EntityNotFoundException
- InvalidInputException
- OperationTimeoutException
- InternalServiceException
- GlueEncryptionException
- InvalidStateException
- ResourceNotReadyException
- FederationSourceException
- FederationSourceRetryableException

BatchGetPartition アクション (Python: batch_get_partition)

バッチリクエストのパーティションを取得します。

リクエスト

- CatalogId – カタログ ID 文字列、1 ~ 255 バイト長、「Single-line string pattern」に一致。

該当するパーティションが存在するデータカタログの ID。提供されない場合は、AWS アカウント ID がデフォルトで使用されます。

- DatabaseName – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  パーティションが存在するカタログデータベースの名前。

- TableName – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  パーティションのテーブルの名前。

- PartitionsToGet – 必須: PartitionValueList オブジェクトの配列。構造 1000 個以下。

  取得するパーティションを識別するパーティション値のリスト。

応答

- Partitions – パーティション オブジェクトの配列。

  リクエストされたパーティションのリスト。

- UnprocessedKeys – PartitionValueList オブジェクトの配列。構造 1000 個以下。

  パーティションが返されなかったリクエスト内のパーティション値のリスト。

エラー

- InvalidInputException
- EntityNotFoundException
- OperationTimeoutException
- InternalServiceException
- GlueEncryptionException
- InvalidStateException
- FederationSourceException
- FederationSourceRetryableException

BatchUpdatePartition アクション (Python: batch_update_partition)

バッチ操作で 1 つ以上のパーティションを更新します。

リクエスト

- CatalogId – カタログ ID 文字列、1 ~ 255 バイト長、「Single-line string pattern」に一致。

  パーティションを更新するカタログの ID。現在、これは AWS アカウント ID である必要があります。

- DatabaseName – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  パーティションが更新されるメタデータデータベースの名前。

- TableName – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  パーティションが更新されるメタデータテーブルの名前。

- Entries – 必須: BatchUpdatePartitionRequestEntry オブジェクトの配列。1 ~ 100 個の構造。

  更新する最大 100 個の BatchUpdatePartitionRequestEntry オブジェクトのリスト。

応答

- Errors – BatchUpdatePartitionFailureEntry オブジェクトの配列。

  リクエストされたパーティションを更新しようとしたときにエラーが発生しました。BatchUpdatePartitionFailureEntry オブジェクトのリスト。

エラー

- InvalidInputException
- EntityNotFoundException
- OperationTimeoutException
- InternalServiceException
- GlueEncryptionException

GetColumnStatisticsForPartition アクション (Python: get_column_statistics_for_partition)

列のパーティション統計を取得します。

このオペレーションに必要な Identity and Access Management (IAM) アクセス許可は `GetPartition` です。

リクエスト

- `CatalogId` – カタログ ID 文字列、1 ~ 255 バイト長、「Single-line string pattern」に一致。

  該当するパーティションが存在するデータカタログの ID。提供されない場合は、AWS アカウント ID がデフォルトで使用されます。
- `DatabaseName` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  パーティションが存在するカタログデータベースの名前。
- `TableName` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  パーティションのテーブルの名前。
- `PartitionValues` – 必須: UTF-8 文字列の配列。

  パーティションを識別するパーティション値のリスト。
- `ColumnNames` – 必須: UTF-8 文字列の配列。文字列 100 個以下。

  列名のリスト。

応答

- `ColumnStatisticsList` – ColumnStatistics オブジェクトの配列。

  取得に失敗した ColumnStatistics のリスト。
- `Errors` – ColumnError オブジェクトの配列。

  列の統計データの取得中にエラーが発生しました。

エラー

- `EntityNotFoundException`
- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`
- `GlueEncryptionException`

UpdateColumnStatisticsForPartition アクション (Python: update_column_statistics_for_partition)

カラムのパーティション統計を作成または更新します。

このオペレーションに必要な Identity and Access Management (IAM) アクセス許可は `UpdatePartition` です。

リクエスト

- `CatalogId` – カタログ ID 文字列、1 ~ 255 バイト長、「Single-line string pattern」に一致。

  該当するパーティションが存在するデータカタログの ID。提供されない場合は、AWS アカウント ID がデフォルトで使用されます。
- `DatabaseName` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  パーティションが存在するカタログデータベースの名前。
- `TableName` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  パーティションのテーブルの名前。
- `PartitionValues` – 必須: UTF-8 文字列の配列。

  パーティションを識別するパーティション値のリスト。
- `ColumnStatisticsList` – 必須: ColumnStatistics オブジェクトの配列。構造 25 個以下。

  列の統計のリスト。

応答

- `Errors` – ColumnStatisticsError オブジェクトの配列。

  列の統計データの更新中にエラーが発生しました。

エラー

- `EntityNotFoundException`
- `InvalidInputException`
- `InternalServiceException`

- OperationTimeoutException
- GlueEncryptionException

DeleteColumnStatisticsForPartition アクション (Python: delete_column_statistics_for_partition)

列のパーティション列統計を削除します。

このオペレーションに必要な Identity and Access Management (IAM) アクセス許可は DeletePartition です。

リクエスト

- CatalogId – カタログ ID 文字列、1 ~ 255 バイト長、「Single-line string pattern」に一致。

  該当するパーティションが存在するデータカタログの ID。提供されない場合は、AWS アカウント ID がデフォルトで使用されます。
- DatabaseName – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  パーティションが存在するカタログデータベースの名前。
- TableName – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  パーティションのテーブルの名前。
- PartitionValues – 必須: UTF-8 文字列の配列。

  パーティションを識別するパーティション値のリスト。
- ColumnName – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  列の名前。

レスポンス

- 応答パラメータはありません。

エラー

- EntityNotFoundException
- InvalidInputException

- `InternalServiceException`
- `OperationTimeoutException`
- `GlueEncryptionException`

接続 API

Catalog API では、AWS Glue での接続の操作に関連するデータ型と API について説明します。

トピック

- 接続 API
- 接続タイプ API
- 接続メタデータとプレビュー API

接続 API

接続 API では、AWS Glue 接続のデータ型と、接続の作成、削除、更新、および一覧表示するための API について説明します。

データ型

- 接続構造
- ConnectionInput 構造
- TestConnectionInput 構造
- PhysicalConnectionRequirements 構造
- GetConnectionsFilter 構造
- AuthenticationConfiguration 構造
- AuthenticationConfigurationInput 構造
- OAuth2Properties 構造
- OAuth2PropertiesInput 構造
- OAuth2ClientApplication 構造
- AuthorizationCodeProperties 構造
- BasicAuthenticationCredentials 構造
- OAuth2Credentials 構造

接続構造

データソースへの接続を定義します。

フィールド

- Name – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  接続定義の名前。

- Description – 説明文字列。2,048 バイト長以下。URI address multi-line string pattern に一致。

  接続の説明。

- ConnectionType – UTF-8 文字列 (有効な値: JDBC | SFTP | MONGODB | KAFKA | NETWORK | MARKETPLACE | CUSTOM | SALESFORCE | VIEW_VALIDATION_REDSHIFT | VIEW_VALIDATION_ATHENA | GOOGLEADS | GOOGLESHEETS | GOOGLEANALYTICS4 | SERVICENOW | MARKETO | SAPODATA | ZENDESK | JIRACLOUD | NETSUITEERP | HUBSPOT | FACEBOOKADS | INSTAGRAMADS | ZOHOCRM | SALESFORCEPARDOT | SALESFORCEMARKETINGCLOUD | SLACK | STRIPE | INTERCOM | SNAPCHATADS)。

  接続のタイプ。現在、SFTP はサポートされていません。

- MatchCriteria - UTF-8 文字列の配列。文字列 10 個以下。

  この接続を選択する際に使用可能な条件のリスト。

- ConnectionProperties – キーと値のペアのマップ配列。100 ペア以下。

  各キーは UTF-8 文字列です (有効な値: HOST | PORT | USERNAME="USER_NAME" | PASSWORD | ENCRYPTED_PASSWORD | JDBC_DRIVER_JAR_URI | JDBC_DRIVER_CLASS_NAME | JDBC_ENGINE | JDBC_ENGINE_VERSION | CONFIG_FILES | INSTANCE_ID | JDBC_CONNECTION_URL | JDBC_ENFORCE_SSL | CUSTOM_JDBC_CERT | SKIP_CUSTOM_JDBC_CERT_VALIDATION | CUSTOM_JDBC_CERT_STRING | CONNECTION_URL | KAFKA_BOOTSTRAP_SERVERS | KAFKA_SSL_ENABLED | KAFKA_CUSTOM_CERT | KAFKA_SKIP_CUSTOM_CERT_VALIDATION | KAFKA_CLIENT_KEYSTORE | KAFKA_CLIENT_KEYSTORE_PASSWORD | KAFKA_CLIENT_KEY_PASSWORD | ENCRYPTED_KAFKA_CLIENT_KEYSTORE_PASSWORD | ENCRYPTED_KAFKA_CLIENT_KEY_PASSWORD | KAFKA_SASL_MECHANISM | KAFKA_SASL_PLAIN_USERNAME | KAFKA_SASL_PLAIN_PASSWORD | ENCRYPTED_KAFKA_SASL_PLAIN_PASSWORD | KAFKA_SASL_SCRAM_USERNAME | KAFKA_SASL_SCRAM_PASSWORD | KAFKA_SASL_SCRAM_SECRETS_ARN |

ENCRYPTED_KAFKA_SASL_SCRAM_PASSWORD | KAFKA_SASL_GSSAPI_KEYTAB | KAFKA_SASL_GSSAPI_KRB5_CONF | KAFKA_SASL_GSSAPI_SERVICE | KAFKA_SASL_GSSAPI_PRINCIPAL | SECRET_ID | CONNECTOR_URL | CONNECTOR_TYPE | CONNECTOR_CLASS_NAME | ENDPOINT | ENDPOINT_TYPE | ROLE_ARN | REGION | WORKGROUP_NAME | CLUSTER_IDENTIFIER | DATABASE)。

各値は値の文字列で、1 ~ 1,024 バイト長です。

これらのキーと値のペアは、バージョン 1 の接続スキーマを使用する場合の接続のパラメータを定義します。

- HOST - ホスト URI: データベースホストの完全修飾ドメイン名 (FQDN) または IPv4 アドレス。
- PORT - データベースホストがデータベース接続をリッスンするポート番号 (1024 ~ 65535)。
- USER_NAME - データベースにログインするための名前。USER_NAME の値文字列は「USERNAME」です。
- PASSWORD - ユーザー名に対応するパスワード (使用されている場合)。
- ENCRYPTED_PASSWORD -データカタログの暗号化設定で ConnectionPasswordEncryption を設定して接続パスワードの保護を有効にすると、このフィールドに暗号化されたパスワードが保存されます。
- JDBC_DRIVER_JAR_URI - 使用する JDBC ドライバーを含む JAR ファイルの Amazon Simple Storage Service (Amazon S3) パス。
- JDBC_DRIVER_CLASS_NAME - 使用する JDBC ドライバーのクラス名。
- JDBC_ENGINE - 使用する JDBC エンジンの名前。
- JDBC_ENGINE_VERSION - 使用する JDBC エンジンのバージョン。
- CONFIG_FILES - (将来の利用のために予約されています。)
- INSTANCE_ID - 使用するインスタンス ID。
- JDBC_CONNECTION_URL - JDBC データソースに接続するための URL。
- JDBC_ENFORCE_SSL - ホスト名のマッチによる Secure Sockets Layer (SSL) がクライアント上の JDBC 接続に適用されるかどうかを指定する大文字と小文字を区別しないブール文字列 (true、false)。デフォルトは False です。
- CUSTOM_JDBC_CERT – お客様のルート証明書を指定する Amazon S3 の場所。AWS Glue では、このルート証明書を使用して、お客様データベースに接続するときにお客様の証明書を検証します。AWS Glue で扱うのは、X.509 証明書のみです。提供する証明書は、DER エンコードし、Base64 エンコード PEM 形式で指定する必要があります。

- SKIP_CUSTOM_JDBC_CERT_VALIDATION – デフォルトでは、false です。AWS Glue は、お客様の証明書の署名アルゴリズムとサブジェクトパブリックキーアルゴリズムを検証します。署名アルゴリズムで許可されるアルゴリズムは、SHA256withRSA、SHA384withRSA、または SHA512withRSA のみです。サブジェクトパブリックキーアルゴリズムでは、キーの長さは 2048 以上である必要があります。このプロパティの値を true に設定して、AWS Glue によるお客様の証明書の検証をスキップできます。
- CUSTOM_JDBC_CERT_STRING - 中間者 (man-in-the-middle) 攻撃を防ぐためにドメインの一致または識別名の一致に使用するカスタム JDBC 証明書文字列。Oracle データベースでは、これは SSL_SERVER_CERT_DN として使用され、Microsoft SQL Server では hostNameInCertificate として使用されます。
- CONNECTION_URL - 一般的な (JDBC 以外の) データソースに接続するための URL。
- SECRET_ID – 認証情報のシークレットマネージャーに使用されるシークレット ID。
- CONNECTOR_URL – MARKETPLACE または CUSTOM 接続のコネクタ URL。
- CONNECTOR_TYPE – MARKETPLACE または CUSTOM 接続のコネクタの種類。
- CONNECTOR_CLASS_NAME – MARKETPLACE または CUSTOM 接続のコネクタクラス名。
- KAFKA_BOOTSTRAP_SERVERS - Kafkaクライアントが接続してそれ自身をブートストラップする先である Kafkaクラスター内の Apache Kafka ブローカーのアドレスを示すホストとポートのペアのコンマ区切りリスト。
- KAFKA_SSL_ENABLED – Apache Kafka 接続で SSL を有効にするか無効にするか。デフォルト値は「true」です。
- KAFKA_CUSTOM_CERT – プライベート CA 証明書ファイル (.pem 形式) の Amazon S3 URL。デフォルトは空の文字列です。
- KAFKA_SKIP_CUSTOM_CERT_VALIDATION – CA 証明書ファイルの検証をスキップするかどうか。AWS Glue では、SHA256withRSA、SHA384withRSA、SHA512withRSA の 3 つのアルゴリズムについて検証されます。デフォルト値は「false」です。
- KAFKA_CLIENT_KEYSTORE – Kafka クライアント側認証用のクライアントキーストアファイルの Amazon S3 の場所 (オプション)。
- KAFKA_CLIENT_KEYSTORE_PASSWORD – 指定されたキーストアにアクセスするためのパスワード (オプション)。
- KAFKA_CLIENT_KEY_PASSWORD – キーストアは、複数のキーで構成することができるので、これは Kafka サーバー側のキーで使用するクライアントキーにアクセスするためのパスワードです (オプション)。

- ENCRYPTED_KAFKA_CLIENT_KEYSTORE_PASSWORD – Kafka クライアントキーストアのパスワードの暗号化されたバージョン (ユーザーが AWS Glue 暗号化パスワードの設定を選択している場合)。
- ENCRYPTED_KAFKA_CLIENT_KEY_PASSWORD – Kafka クライアントキーパスワードの暗号化されたバージョン (ユーザーが AWS Glue 暗号化パスワードの設定を選択している場合)。
- KAFKA_SASL_MECHANISM – "SCRAM-SHA-512"、"GSSAPI"、"AWS_MSK_IAM"、"PLAIN"。これらはサポートされている SASL メカニズムです。
- KAFKA_SASL_PLAIN_USERNAME – 「PLAIN」メカニズムで認証するときに使用されるプレーンテキストのユーザーネーム。
- KAFKA_SASL_PLAIN_PASSWORD – 「PLAIN」メカニズムで認証するときに使用されるプレーンテキストのパスワード。
- ENCRYPTED_KAFKA_SASL_PLAIN_PASSWORD – Kafka SASL PLAIN パスワードの暗号化バージョン (ユーザーが AWS Glue 暗号化パスワードの設定を選択した場合)。
- KAFKA_SASL_SCRAM_USERNAME - 「SCRAM-SHA-512」メカニズムでの認証に使用されるプレーンテキストのユーザーネーム。
- KAFKA_SASL_SCRAM_PASSWORD - 「SCRAM-SHA-512」メカニズムでの認証に使用されるプレーンテキストのパスワード。
- ENCRYPTED_KAFKA_SASL_SCRAM_PASSWORD – Kafka SASL SCRAM パスワードの暗号化されたバージョン (ユーザーが AWS Glue 暗号化パスワードの設定を選択している場合)。
- KAFKA_SASL_SCRAM_SECRETS_ARN – AWS Secrets Manager のシークレットの Amazon Resource Name。
- KAFKA_SASL_GSSAPI_KEYTAB - Kerberos keytab ファイルの S3 の場所。キータブには、1 つ以上のプリンシパルの長期キーが保存されます。詳細については、「MIT Kerberos ドキュメント: キータブ」を参照してください。
- KAFKA_SASL_GSSAPI_KRB5_CONF - Kerberos krb5.conf ファイルの S3 の場所。krb5.conf には、KDC サーバーの場所などの Kerberos 構成情報が保存されます。詳細については、「MIT Kerberos ドキュメント: krb5.conf」を参照してください。
- KAFKA_SASL_GSSAPI_SERVICE - Kafka 構成 の sasl.kerberos.service.name で設定した通りのKerberos サービスの名前。
- KAFKA_SASL_GSSAPI_PRINCIPAL - AWS Glue によって使用される Kerberos プリンシパルの名前。詳細については、「Kafka ドキュメント: Kafka ブローカーの構成」を参照してください。
- ROLE_ARN - クエリの実行に使用されるロール。

  - REGION - AWS クエリが実行されるリージョン。
  - WORKGROUP_NAME - クエリを実行する Amazon Redshift Serverless ワークグループまたは Amazon Athena ワークグループの名前。
  - CLUSTER_IDENTIFIER - クエリを実行する Amazon Redshift クラスターのクラスター識別子。
  - DATABASE - 接続先の Amazon Redshift データベース。
- SparkProperties – キーバリューペアのマップ配列。

  各キーは UTF-8 文字列で、1 ~ 128 バイト長です。

  各値は UTF-8 文字列で、1 ~ 2,048 バイト長です。

  Spark コンピューティング環境に固有の接続プロパティ。
- AthenaProperties – キーバリューペアのマップ配列。

  各キーは UTF-8 文字列で、1 ~ 128 バイト長です。

  各値は UTF-8 文字列で、1 ~ 2,048 バイト長です。

  Athena コンピューティング環境に固有の接続プロパティ。
- PythonProperties – キーバリューペアのマップ配列。

  各キーは UTF-8 文字列で、1 ~ 128 バイト長です。

  各値は UTF-8 文字列で、1 ~ 2,048 バイト長です。

  Python コンピューティング環境に固有の接続プロパティ。
- PhysicalConnectionRequirements – PhysicalConnectionRequirements オブジェクト。

  この接続を正常に行うために必要な物理接続要件 (例: 仮想プライベートクラウド (VPC)、SecurityGroup)。
- CreationTime – タイムスタンプ。

  この接続定義が作成された時刻のタイムスタンプ。
- LastUpdatedTime – タイムスタンプ。

  この接続定義が更新された最終時刻のタイムスタンプ。
- LastUpdatedBy – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  この接続定義を最終更新したユーザー、グループ、またはロール。

- Status – UTF-8 文字列 (有効な値: READY | IN_PROGRESS | FAILED)。

  接続のステータス。READY、IN_PROGRESS、または FAILED のいずれか。

- StatusReason – UTF-8 文字列、1 ~ 16384 バイト長。

  接続ステータスの理由。

- LastConnectionValidationTime – タイムスタンプ。

  この接続が最後に検証された時刻のタイムスタンプ。

- AuthenticationConfiguration – AuthenticationConfiguration オブジェクト。

  接続の認証プロパティ。

- ConnectionSchemaVersion – 1 ~ 2 の数値 (整数)。

  この接続の接続スキーマのバージョン。バージョン 2 では、特定のコンピューティング環境のプロパティがサポートされています。

- CompatibleComputeEnvironments – UTF-8 文字列の配列。

  接続と互換性のあるコンピューティング環境のリスト。

ConnectionInput 構造

作成または更新する接続を指定するために使用される構造。

フィールド

- Name – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  コレクションの名前。

- Description – 説明文字列。2,048 バイト長以下。URI address multi-line string pattern に一致。

  接続の説明。

- ConnectionType – 必須: UTF-8 文字列 (有効な値: JDBC | SFTP | MONGODB | KAFKA | NETWORK | MARKETPLACE | CUSTOM | SALESFORCE | VIEW_VALIDATION_REDSHIFT | VIEW_VALIDATION_ATHENA | GOOGLEADS | GOOGLESHEETS | GOOGLEANALYTICS4 | SERVICENOW | MARKETO | SAPODATA | ZENDESK | JIRACLOUD | NETSUITEERP | HUBSPOT | FACEBOOKADS | INSTAGRAMADS | ZOHOCRM | SALESFORCEPARDOT | SALESFORCEMARKETINGCLOUD | SLACK | STRIPE | INTERCOM | SNAPCHATADS)。

接続のタイプ。現在、次のタイプがサポートされています。

- JDBC - Java Database Connectivity (JDBC) を介してデータベースへの接続を指定します。

  JDBC 接続では、次の ConnectionParameters を使用します。

  - 必須: (HOST、PORT、JDBC_ENGINE) のすべて、または JDBC_CONNECTION_URL。
  - 必須: (USERNAME、PASSWORD) の両方、または SECRET_ID。
  - オプション: JDBC_ENFORCE_SSL、CUSTOM_JDBC_CERT、CUSTOM_JDBC_CERT_STRING、SKIP_CUSTOM_JDB これらのパラメータは、JDBC で SSL を設定するために使用されます。

- KAFKA - Apache Kafka ストリーミングプラットフォームへの接続を指定します。

  KAFKA 接続では、次の ConnectionParameters を使用します。

  - 必須: KAFKA_BOOTSTRAP_SERVERS。
  - オプション: KAFKA_SSL_ENABLED、KAFKA_CUSTOM_CERT、KAFKA_SKIP_CUSTOM_CERT_VALIDATION。 これらのパラメータは KAFKA で SSL を設定するために使用されます。
  - オプション: KAFKA_CLIENT_KEYSTORE、KAFKA_CLIENT_KEYSTORE_PASSWORD、KAFKA_CLIENT_KEY_PASS これらのパラメータは、KAFKA の中で SSL で TLS クライアント設定を設定するために使用されます。
  - オプション: KAFKA_SASL_MECHANISM。SCRAM-SHA-512、GSSAPI、または AWS_MSK_IAM として指定できます。
  - オプション: KAFKA_SASL_SCRAM_USERNAME、KAFKA_SASL_SCRAM_PASSWORD、ENCRYPTED_KAFKA_SASL_S これらのパラメータは、KAFKA で SASL/SCRAM-SHA-512 認証を設定するために使用されます。
  - オプション: KAFKA_SASL_GSSAPI_KEYTAB、KAFKA_SASL_GSSAPI_KRB5_CONF、KAFKA_SASL_GSSAPI_SER これらのパラメータは、KAFKA で SASL/GSSAPI 認証を設定するために使用されます。

- MONGODB - MongoDB ドキュメントデータベースへの接続を指定します。

  MONGODB 接続では、次の ConnectionParameters を使用します。

  - 必須: CONNECTION_URL。
  - 必須: (USERNAME、PASSWORD) のすべて、または SECRET_ID。

- VIEW_VALIDATION_REDSHIFT - Amazon Redshift によるビューの検証に使用される接続を指定します。
- VIEW_VALIDATION_ATHENA - Amazon Athena によるビューの検証に使用される接続を指定します。
- NETWORK - Amazon Virtual Private Cloud 環境 (Amazon VPC) 内のデータソースへのネットワーク接続を指定します。

  NETWORK 接続には、ConnectionParameters は必要ありません。代わりに、PhysicalConnectionRequirements を用意します。
- MARKETPLACE – AWS Marketplace から購入したコネクタに含まれる構成設定を使用して、AWS Glue がネイティブにサポートしていないデータストアからの読み取りと書き込みを行います。

  MARKETPLACE 接続では、次の ConnectionParameters を使用します。

  - 必須: CONNECTOR_TYPE、CONNECTOR_URL、CONNECTOR_CLASS_NAME、CONNECTION_URL。
  - JDBC CONNECTOR_TYPE 接続に必須: (USERNAME、PASSWORD) のすべて、または SECRET_ID。
- CUSTOM – カスタムコネクタに含まれる設定を使用して、AWS Glue がネイティブにサポートしていないデータストアに対する読み取りと書き込みを行います。

さらに、以下の SaaS コネクタの ConnectionType がサポートされています。

- FACEBOOKADS - Facebook 広告への接続を指定。
- GOOGLEADS - Google 広告への接続を指定。
- GOOGLESHEETS - Google スプレッドシートへの接続を指定。
- GOOGLEANALYTICS4 - Google Analytics 4 への接続を指定。
- HUBSPOT - HubSpot への接続を指定。
- INSTAGRAMADS - Instagram 広告への接続を指定。
- INTERCOM - Intercom への接続を指定。
- JIRACLOUD - Jira Cloud への接続を指定。
- MARKETO - Adobe Marketo Engage への接続を指定。
- NETSUITEERP - Oracle NetSuite への接続を指定。
- SALESFORCE - OAuth 認証を使用して Salesforce への接続を指定。

- SALESFORCEMARKETINGCLOUD - Salesforce Marketing Cloud への接続を指定。
- SALESFORCEPARDOT - Salesforce Marketing Cloud Account Engagement (MCAE) への接続を指定。
- SAPODATA - SAP OData への接続を指定。
- SERVICENOW - ServiceNow への接続を指定。
- SLACK - Slack への接続を指定。
- SNAPCHATADS - Snapchat 広告への接続を指定。
- STRIPE - Stripe への接続を指定。
- ZENDESK - Zendesk への接続を指定。
- ZOHOCRM - Zoho CRM への接続を指定。

特定のコネクタに必要な接続パラメータの詳細については、「AWS Glueユーザーガイド」の「AWS Glue 接続の追加」にあるコネクタのドキュメントを参照してください。

SFTP はサポートされていません。

オプションの ConnectionProperties を使用して AWS Glue の機能を設定する方法の詳細については、「AWS Glue 接続プロパティ」を参照してください。

オプションの ConnectionProperties を使用して AWS Glue Studio の機能を設定する方法の詳細については、「コネクタと接続の使用」を参照してください。

- MatchCriteria - UTF-8 文字列の配列。文字列 10 個以下。

  この接続を選択する際に使用可能な条件のリスト。

- ConnectionProperties – 必須: キーと値のペアのマップ配列。100 ペア以下。

  各キーは UTF-8 文字列です (有効な値: HOST | PORT | USERNAME="USER_NAME" | PASSWORD | ENCRYPTED_PASSWORD | JDBC_DRIVER_JAR_URI | JDBC_DRIVER_CLASS_NAME | JDBC_ENGINE | JDBC_ENGINE_VERSION | CONFIG_FILES | INSTANCE_ID | JDBC_CONNECTION_URL | JDBC_ENFORCE_SSL | CUSTOM_JDBC_CERT | SKIP_CUSTOM_JDBC_CERT_VALIDATION | CUSTOM_JDBC_CERT_STRING | CONNECTION_URL | KAFKA_BOOTSTRAP_SERVERS | KAFKA_SSL_ENABLED | KAFKA_CUSTOM_CERT | KAFKA_SKIP_CUSTOM_CERT_VALIDATION | KAFKA_CLIENT_KEYSTORE | KAFKA_CLIENT_KEYSTORE_PASSWORD | KAFKA_CLIENT_KEY_PASSWORD | ENCRYPTED_KAFKA_CLIENT_KEYSTORE_PASSWORD | ENCRYPTED_KAFKA_CLIENT_KEY_PASSWORD | KAFKA_SASL_MECHANISM

| KAFKA_SASL_PLAIN_USERNAME | KAFKA_SASL_PLAIN_PASSWORD | ENCRYPTED_KAFKA_SASL_PLAIN_PASSWORD | KAFKA_SASL_SCRAM_USERNAME | KAFKA_SASL_SCRAM_PASSWORD | KAFKA_SASL_SCRAM_SECRETS_ARN | ENCRYPTED_KAFKA_SASL_SCRAM_PASSWORD | KAFKA_SASL_GSSAPI_KEYTAB | KAFKA_SASL_GSSAPI_KRB5_CONF | KAFKA_SASL_GSSAPI_SERVICE | KAFKA_SASL_GSSAPI_PRINCIPAL | SECRET_ID | CONNECTOR_URL | CONNECTOR_TYPE | CONNECTOR_CLASS_NAME | ENDPOINT | ENDPOINT_TYPE | ROLE_ARN | REGION | WORKGROUP_NAME | CLUSTER_IDENTIFIER | DATABASE)。

各値は値の文字列で、1 ~ 1,024 バイト長です。

以下のキーバリューペアは接続のパラメータを定義します。

- SparkProperties – キーバリューペアのマップ配列。

  各キーは UTF-8 文字列で、1 ~ 128 バイト長です。

  各値は UTF-8 文字列で、1 ~ 2,048 バイト長です。

  Spark コンピューティング環境に固有の接続プロパティ。
- AthenaProperties – キーバリューペアのマップ配列。

  各キーは UTF-8 文字列で、1 ~ 128 バイト長です。

  各値は UTF-8 文字列で、1 ~ 2,048 バイト長です。

  Athena コンピューティング環境に固有の接続プロパティ。
- PythonProperties – キーバリューペアのマップ配列。

  各キーは UTF-8 文字列で、1 ~ 128 バイト長です。

  各値は UTF-8 文字列で、1 ~ 2,048 バイト長です。

  Python コンピューティング環境に固有の接続プロパティ。
- PhysicalConnectionRequirements – PhysicalConnectionRequirements オブジェクト。

  この接続を正常に行うために必要な物理接続要件 (例: 仮想プライベートクラウド (VPC)、SecurityGroup)。
- AuthenticationConfiguration – AuthenticationConfigurationInput オブジェクト。

  接続の認証プロパティ。

- ValidateCredentials – ブール。

  接続の作成中に認証情報を検証するフラグ。デフォルトは true です。

- ValidateForComputeEnvironments – UTF-8 文字列の配列。

  指定された接続プロパティが検証されるコンピューティング環境。

TestConnectionInput 構造

サービスへの接続テストを指定するために使用される構造。

フィールド

- ConnectionType – 必須: UTF-8 文字列 (有効な値: JDBC | SFTP | MONGODB | KAFKA | NETWORK | MARKETPLACE | CUSTOM | SALESFORCE | VIEW_VALIDATION_REDSHIFT | VIEW_VALIDATION_ATHENA | GOOGLEADS | GOOGLESHEETS | GOOGLEANALYTICS4 | SERVICENOW | MARKETO | SAPODATA | ZENDESK | JIRACLOUD | NETSUITEERP | HUBSPOT | FACEBOOKADS | INSTAGRAMADS | ZOHOCRM | SALESFORCEPARDOT | SALESFORCEMARKETINGCLOUD | SLACK | STRIPE | INTERCOM | SNAPCHATADS)。

  テストする接続のタイプ。このオペレーションは、JDBC または SALESFORCE 接続タイプのみで使用できます。

- ConnectionProperties – 必須: キーと値のペアのマップ配列。100 ペア以下。

  各キーは UTF-8 文字列です (有効な値: HOST | PORT | USERNAME="USER_NAME" | PASSWORD | ENCRYPTED_PASSWORD | JDBC_DRIVER_JAR_URI | JDBC_DRIVER_CLASS_NAME | JDBC_ENGINE | JDBC_ENGINE_VERSION | CONFIG_FILES | INSTANCE_ID | JDBC_CONNECTION_URL | JDBC_ENFORCE_SSL | CUSTOM_JDBC_CERT | SKIP_CUSTOM_JDBC_CERT_VALIDATION | CUSTOM_JDBC_CERT_STRING | CONNECTION_URL | KAFKA_BOOTSTRAP_SERVERS | KAFKA_SSL_ENABLED | KAFKA_CUSTOM_CERT | KAFKA_SKIP_CUSTOM_CERT_VALIDATION | KAFKA_CLIENT_KEYSTORE | KAFKA_CLIENT_KEYSTORE_PASSWORD | KAFKA_CLIENT_KEY_PASSWORD | ENCRYPTED_KAFKA_CLIENT_KEYSTORE_PASSWORD | ENCRYPTED_KAFKA_CLIENT_KEY_PASSWORD | KAFKA_SASL_MECHANISM | KAFKA_SASL_PLAIN_USERNAME | KAFKA_SASL_PLAIN_PASSWORD | ENCRYPTED_KAFKA_SASL_PLAIN_PASSWORD | KAFKA_SASL_SCRAM_USERNAME | KAFKA_SASL_SCRAM_PASSWORD | KAFKA_SASL_SCRAM_SECRETS_ARN | ENCRYPTED_KAFKA_SASL_SCRAM_PASSWORD | KAFKA_SASL_GSSAPI_KEYTAB

| KAFKA_SASL_GSSAPI_KRB5_CONF | KAFKA_SASL_GSSAPI_SERVICE | KAFKA_SASL_GSSAPI_PRINCIPAL | SECRET_ID | CONNECTOR_URL | CONNECTOR_TYPE | CONNECTOR_CLASS_NAME | ENDPOINT | ENDPOINT_TYPE | ROLE_ARN | REGION | WORKGROUP_NAME | CLUSTER_IDENTIFIER | DATABASE)。

各値は値の文字列で、1 ~ 1,024 バイト長です。

接続のパラメータを定義するキーと値のペア。

JDBC 接続では次の接続プロパティを使用します。

- 必須: (HOST、PORT、JDBC_ENGINE) のすべて、または JDBC_CONNECTION_URL。
- 必須: (USERNAME、PASSWORD) の両方、または SECRET_ID。
- オプション: JDBC_ENFORCE_SSL、CUSTOM_JDBC_CERT、CUSTOM_JDBC_CERT_STRING、SKIP_CUSTOM_JDBC_ これらのパラメータは、JDBC で SSL を設定するために使用されます。

SALESFORCE 接続では AuthenticationConfiguration メンバーを設定する必要があります。

- AuthenticationConfiguration – AuthenticationConfigurationInput オブジェクト。

  TestConnection リクエストに認証の設定が含まれる構造。OAuth 認証を使用した Salesforce への接続に必要です。

PhysicalConnectionRequirements 構造

GetConnection レスポンスの OAuth クライアントアプリ。

フィールド

- SubnetId – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  接続で使用されるサブネット ID。
- SecurityGroupIdList - UTF-8 文字列の配列。文字列 50 個以下。

  接続で使用されるセキュリティグループ ID のリスト。
- AvailabilityZone – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  接続のアベイラビリティーゾーン。

GetConnectionsFilter 構造

GetConnections API オペレーションによって返る接続定義をフィルタリングします。

フィールド

- MatchCriteria - UTF-8 文字列の配列。文字列 10 個以下。

  接続定義を返すためにその接続定義に記録された条件と一致する必要がある条件文字列。
- ConnectionType – UTF-8 文字列 (有効な値: JDBC | SFTP | MONGODB | KAFKA | NETWORK | MARKETPLACE | CUSTOM | SALESFORCE | VIEW_VALIDATION_REDSHIFT | VIEW_VALIDATION_ATHENA | GOOGLEADS | GOOGLESHEETS | GOOGLEANALYTICS4 | SERVICENOW | MARKETO | SAPODATA | ZENDESK | JIRACLOUD | NETSUITEERP | HUBSPOT | FACEBOOKADS | INSTAGRAMADS | ZOHOCRM | SALESFORCEPARDOT | SALESFORCEMARKETINGCLOUD | SLACK | STRIPE | INTERCOM | SNAPCHATADS)。

  返す接続のタイプ。現在、SFTP はサポートされていません。
- ConnectionSchemaVersion – 1 ~ 2 の数値 (整数)。

  接続がスキーマバージョン 1 または 2 で作成されたかどうかを示します。

AuthenticationConfiguration 構造

認証の設定を含む構造。

フィールド

- AuthenticationType – UTF-8 文字列 (有効な値: BASIC | OAUTH2 | CUSTOM | IAM)。

  認証の設定を含む構造。
- SecretArn – UTF-8 文字列。Custom string pattern #36 に一致。

  認証情報を保存するシークレットマネージャー ARN。
- KmsKeyArn – UTF-8 文字列。Custom string pattern #29 に一致。

  機密認証情報の暗号化に使用される KMS キーの Amazon リソースネーム (ARN)。このキーは、認証設定に保存されている認証情報やその他の機密データを保護するために使用されます。
- OAuth2Properties – OAuth2Properties オブジェクト。

  OAuth2 認証のプロパティ。

AuthenticationConfigurationInput 構造

CreateConnection リクエストの認証の設定を含む構造。

フィールド

- AuthenticationType – UTF-8 文字列 (有効な値: BASIC | OAUTH2 | CUSTOM | IAM)。

  CreateConnection リクエストの認証の設定を含む構造。
- OAuth2Properties – OAuth2PropertiesInput オブジェクト。

  CreateConnection リクエストの OAuth2 認証のプロパティ。
- SecretArn– UTF-8 文字列。Custom string pattern #36 に一致。

  CreateConnection リクエストに認証情報を保存するシークレットマネージャー ARN。
- KmsKeyArn– UTF-8 文字列。Custom string pattern #29 に一致。

  接続の暗号化に使用される KMS キーの ARN。リクエストの入力情報としてのみ取得され、Secret Manager に保存されます。
- BasicAuthenticationCredentials – BasicAuthenticationCredentials オブジェクト。

  認証のタイプが基本認証である場合に使用される認証情報。
- CustomAuthenticationCredentials – キーバリューペアのマップ配列。

  各キーは UTF-8 文字列で、1 ~ 128 バイト長です。

  各値は UTF-8 文字列で、1 ~ 2,048 バイト長です。

  認証のタイプがカスタム認証である場合に使用される認証情報。

OAuth2Properties 構造

OAuth2 認証のプロパティを含む構造。

フィールド

- OAuth2GrantType – UTF-8 文字列 (有効な値: AUTHORIZATION_CODE | CLIENT_CREDENTIALS | JWT_BEARER)。

  OAuth2 グラントタイプ。例えば、AUTHORIZATION_CODE、JWT_BEARER、または CLIENT_CREDENTIALS です。

- OAuth2ClientApplication – OAuth2ClientApplication オブジェクト。

  クライアントアプリケーションタイプ。たとえば、AWS_MANAGED または USER_MANAGED。
- TokenUrl – UTF-8 文字列。256 バイト長以下。Custom string pattern #40 に一致。

  認可コードをアクセストークンと交換するための、プロバイダーの認証サーバーの URL。
- TokenUrlParametersMap – キーバリューペアのマップ配列。

  各キーは UTF-8 文字列で、1 ~ 128 バイト長です。

  各値は UTF-8 文字列で、1 ~ 512 バイト長です。

  トークン GET リクエストに追加されるパラメータのマップ。

OAuth2PropertiesInput 構造

CreateConnection リクエストの OAuth2 のプロパティを含む構造。

フィールド

- OAuth2GrantType – UTF-8 文字列 (有効な値: AUTHORIZATION_CODE | CLIENT_CREDENTIALS | JWT_BEARER)。

  CreateConnection リクエストの OAuth2 グラントタイプ。例えば、AUTHORIZATION_CODE、JWT_BEARER、または CLIENT_CREDENTIALS です。
- OAuth2ClientApplication – OAuth2ClientApplication オブジェクト。

  CreateConnection リクエストのクライアントアプリケーションタイプ。例えば、AWS_MANAGED、USER_MANAGED です。
- TokenUrl – UTF-8 文字列、256 バイト長以下。Custom string pattern #40 に一致。

  認可コードをアクセストークンと交換するための、プロバイダーの認証サーバーの URL。
- TokenUrlParametersMap – キーバリューペアのマップ配列。

  各キーは UTF-8 文字列で、1 ~ 128 バイト長です。

  各値は UTF-8 文字列で、1 ~ 512 バイト長です。

  トークン GET リクエストに追加されるパラメータのマップ。
- AuthorizationCodeProperties – AuthorizationCodeProperties オブジェクト。

OAuth2 AUTHORIZATION_CODE グラントタイプに必要なプロパティのセット。

- OAuth2Credentials – OAuth2Credentials オブジェクト。

  認証のタイプが OAuth2 認証である場合に使用される認証情報。

OAuth2ClientApplication 構造

接続に使用される OAuth2 クライアントアプリ。

フィールド

- UserManagedClientApplicationClientId - UTF-8 文字列。2,048 バイト長以下。Custom string pattern #37 に一致。

  ClientAppType が USER_MANAGED の場合のクライアントアプリケーションの clientID。
- AWSManagedClientApplicationReference - UTF-8 文字列。2,048 バイト長以下。Custom string pattern #37 に一致。

  AWS によって管理されている SaaS 側のクライアントアプリケーションへの参照。

AuthorizationCodeProperties 構造

OAuth2 AUTHORIZATION_CODE グラントタイプのワークフローに必要なプロパティのセット。

フィールド

- AuthorizationCode – UTF-8 文字列、1 ~ 4096 バイト長、Custom string pattern #37 に一致。

  AUTHORIZATION_CODE で付与されるワークフローの 3 番目のレッグで使用される認可コード。これは、アクセストークンと交換されると無効になるワンタイムコードであるため、この値をリクエストパラメータとして指定できます。
- RedirectUri - UTF-8 文字列、512 バイト長以下、Custom string pattern #41 に一致。

  認可コードの発行時にユーザーが認可サーバーによってリダイレクトされるリダイレクト URI。この URI は、認可コードがアクセストークンと交換されるときに使用されます。

BasicAuthenticationCredentials 構造

SecretArn の値を指定しない場合に基本的な認証情報を提供する。

フィールド

- `Username` - UTF-8 文字列、512 バイト長以下、Custom string pattern #37 に一致。

  データソースに接続するためのユーザー名。
- `Password` - UTF-8 文字列、512 バイト長以下、Custom string pattern #33 に一致。

  データソースに接続するためのパスワード。

OAuth2Credentials 構造

認証のタイプが OAuth2 認証である場合に使用される認証情報。

フィールド

- `UserManagedClientApplicationClientSecret` - UTF-8 文字列、512 バイト長以下、Custom string pattern #38 に一致。

  クライアント アプリケーションがユーザー マネージドである場合のクライアント アプリケーション シークレット。
- `AccessToken` – UTF-8 文字列。4,096 バイト長以下。Custom string pattern #38 に一致。

  認証タイプが OAuth2 である場合に使用されるアクセストークン。
- `RefreshToken` – UTF-8 文字列。4,096 バイト長以下。Custom string pattern #38 に一致。

  認証タイプが OAuth2 である場合に使用される更新トークン。
- `JwtToken` – UTF-8 文字列。8,000 バイト長以下。Custom string pattern #39 に一致。

  認証タイプが OAuth2 である場合に使用される JSON ウェブトークン (JWT)。

操作

- CreateConnection アクション (Python: create_connection)
- DeleteConnection アクション (Python: delete_connection)
- GetConnection アクション (Python: get_connection)
- GetConnections アクション (Python: get_connections)
- UpdateConnection アクション (Python: update_connection)

- TestConnection アクション (Python: test_connection)
- BatchDeleteConnection アクション (Python: batch_delete_connection)

CreateConnection アクション (Python: create_connection)

データカタログで接続定義を作成します。

フェデレーションリソースの作成に使用される接続には IAM `glue:PassConnection` 権限が必要です。

リクエスト

- `CatalogId` – カタログ ID 文字列、1 ~ 255 バイト長、「Single-line string pattern」に一致。

  接続を作成するデータカタログの ID。設定しない場合は、AWS アカウント ID がデフォルトで使用されます。
- `ConnectionInput` – 必須: ConnectionInput オブジェクト。

  作成する接続を定義する `ConnectionInput` オブジェクト。
- `Tags` – キーと値のペアのマップ配列。50 ペア以下。

  各キーは UTF-8 文字列で、1 ~ 128 バイト長です。

  各値は UTF-8 文字列で、256 バイト長以下です。

  接続に割り当てるタグ。

レスポンス

- `CreateConnectionStatus` – UTF-8 文字列 (有効な値: `READY` | `IN_PROGRESS` | `FAILED`)。

  接続作成リクエストのステータス。リクエストは、VPC 経由でトークン交換を使用して OAuth 接続を作成する場合など、特定の認証タイプでは時間がかかる場合があります。

エラー

- `AlreadyExistsException`
- `InvalidInputException`
- `OperationTimeoutException`

- ResourceNumberLimitExceededException
- GlueEncryptionException

DeleteConnection アクション (Python: delete_connection)

データカタログから接続を削除します。

リクエスト

- CatalogId – カタログ ID 文字列、1 ~ 255 バイト長、「Single-line string pattern」に一致。

  接続が存在するデータカタログの ID。設定しない場合は、AWS アカウント ID がデフォルトで使用されます。
- ConnectionName – 必須: UTF-8 文字列、1 ~ 255 バイト長、Single-line string pattern に一致。

  削除する接続の名前。

レスポンス

- 応答パラメータはありません。

エラー

- EntityNotFoundException
- OperationTimeoutException

GetConnection アクション (Python: get_connection)

データカタログから接続定義を取得します。

リクエスト

- CatalogId – カタログ ID 文字列、1 ~ 255 バイト長、「Single-line string pattern」に一致。

  接続が存在するデータカタログの ID。設定しない場合は、AWS アカウント ID がデフォルトで使用されます。
- Name – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  取得する接続定義の名前。

- HidePassword – ブール。

  パスワードを返さずに接続メタデータを取得できます。例えば、AWS Glue コンソールではこのフラグを使用して接続を取得し、パスワードは表示しません。呼び出し元に AWS KMS キーを使用してパスワードを復号するアクセス許可がない可能性があるが、それ以外の接続プロパティへのアクセス許可がある場合は、このパラメータを設定します。

- ApplyOverrideForComputeEnvironment – UTF-8 文字列 (有効な値: SPARK | ATHENA | PYTHON)。

  複数のサービスで使用できる接続の場合、指定されたコンピューティング環境の戻りプロパティを指定します。

レスポンス

- Connection – 接続 オブジェクト。

  リクエストされた接続定義。

エラー

- EntityNotFoundException
- OperationTimeoutException
- InvalidInputException
- GlueEncryptionException

GetConnections アクション (Python: get_connections)

データカタログから接続定義のリストを取得します。

リクエスト

- CatalogId – カタログ ID 文字列、1 ~ 255 バイト長、「Single-line string pattern」に一致。

  接続が存在するデータカタログの ID。設定しない場合は、AWS アカウント ID がデフォルトで使用されます。

- Filter – GetConnectionsFilter オブジェクト。

  返される接続をコントロールするフィルター。

- HidePassword – ブール。

  パスワードを返さずに接続メタデータを取得できます。例えば、AWS Glue コンソールではこのフラグを使用して接続を取得し、パスワードは表示しません。呼び出し元に AWS KMS キーを使用してパスワードを復号するアクセス許可がない可能性があるが、それ以外の接続プロパティへのアクセス許可がある場合は、このパラメータを設定します。

- NextToken – UTF-8 文字列。

  継続トークン (これが継続呼び出しの場合)。

- MaxResults – 1 ~ 1000 の数値 (整数)。

  1 回の応答で返す接続の最大数。

応答

- ConnectionList – 接続 オブジェクトの配列。

  リクエストされた接続定義のリスト。

- NextToken – UTF–8 文字列。

  返された接続のリストにフィルタリングされた接続の最後のものが含まれていない場合は、継続トークン。

エラー

- EntityNotFoundException
- OperationTimeoutException
- InvalidInputException
- GlueEncryptionException

UpdateConnection アクション (Python: update_connection)

データカタログで接続定義を更新します。

リクエスト

- CatalogId – カタログ ID 文字列、1 ~ 255 バイト長、「Single-line string pattern」に一致。

接続が存在するデータカタログの ID。設定しない場合は、AWS アカウント ID がデフォルトで使用されます。

- Name – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  更新する接続定義の名前。

- ConnectionInput – 必須: ConnectionInput オブジェクト。

  該当する接続を再定義する ConnectionInput オブジェクト。

レスポンス

- 応答パラメータはありません。

エラー

- InvalidInputException
- EntityNotFoundException
- OperationTimeoutException
- InvalidInputException
- GlueEncryptionException

TestConnection アクション (Python: test_connection)

サービスへの接続をテストして、指定したサービス認証情報を検証します。

既存の接続名を指定するか、存在しない接続入力をテストするための TestConnectionInput を指定します。両方を同時に指定するとエラーが発生します。

アクションが成功すると、サービスは HTTP 200 レスポンスを返します。

リクエスト

- ConnectionName – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  オプション。テストする接続の名前。名前のみを指定すると、オペレーションは接続を取得してそれをテストに使用します。

- CatalogId – カタログ ID 文字列、1 ~ 255 バイト長、Single-line string pattern に一致。

接続が存在するカタログの ID。

- `TestConnectionInput` – TestConnectionInput オブジェクト。

  サービスへの接続テストを指定するために使用される構造。

レスポンス

- 応答パラメータはありません。

エラー

- `InvalidInputException`
- `OperationTimeoutException`
- `ResourceNumberLimitExceededException`
- `GlueEncryptionException`
- `FederationSourceException`
- `AccessDeniedException`
- `EntityNotFoundException`
- `ConflictException`
- `InternalServiceException`

BatchDeleteConnection アクション (Python: batch_delete_connection)

データカタログから接続定義のリストを削除します。

リクエスト

- `CatalogId` – カタログ ID 文字列、1 ~ 255 バイト長、「Single-line string pattern」に一致。

  接続が存在するデータカタログの ID。設定しない場合は、AWS アカウント ID がデフォルトで使用されます。

- `ConnectionNameList` – 必須: UTF-8 文字列の配列。文字列 25 個以下。

  削除する接続の名前のリスト。

レスポンス

- `Succeeded` – UTF-8 文字列の配列。

  正常に削除された接続定義の名前のリスト。
- `Errors` – キーバリューペアのマップ配列。

  各キーは UTF-8 文字列、1～255 バイト長で、Single-line string pattern に一致します。

  各値は ErrorDetail オブジェクトです。

  エラーの詳細への正常に削除されなかった接続の名前のマップ。

エラー

- `InternalServiceException`
- `OperationTimeoutException`

接続タイプ API

接続タイプ API では、接続タイプの説明に関連する AWS Glue API について説明します。

接続管理 API

- DescribeConnectionType アクション (Python: describe_connection_type)
- ListConnectionTypes アクション (Python: list_connection_types)
- ConnectionTypeBrief 構造
- ConnectionTypeVariant の構造

DescribeConnectionType アクション (Python: describe_connection_type)

`DescribeConnectionType` API は、AWS Glue の特定の接続タイプでサポートされているオプションの詳細を提供します。

リクエスト

- `ConnectionType` – 必須: UTF-8 文字列。1～255 バイト長。Single-line string pattern に一致。

  説明する接続タイプの名前。

レスポンス

- ConnectionType – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  接続タイプの名前。

- Description – UTF-8 文字列、1024 バイト長以下。

  接続タイプの説明。

- Capabilities – 機能 オブジェクト。

  サポートされている認証タイプ、データインターフェイスタイプ (コンピューティング環境)、コネクタのデータオペレーション。

- ConnectionProperties – キーバリューペアのマップ配列。

  各キーは UTF-8 文字列で、1 ~ 128 バイト長です。

  各値は プロパティ オブジェクトです。

  コンピューティング環境間で共通の接続プロパティ。

- ConnectionOptions – キーバリューペアのマップ配列。

  各キーは UTF-8 文字列で、1 ~ 128 バイト長です。

  各値は プロパティ オブジェクトです。

  ConnectionInput.ConnectionProperties で接続を作成するときに設定できるプロパティを返します。ConnectionOptions は、データフレームに渡される接続オプションマップの Spark ETL スクリプトで設定できるパラメータを定義します。

- AuthenticationConfiguration – AuthConfiguration オブジェクト。

  接続に使用する認証のタイプ。

- ComputeEnvironmentConfigurations – キーバリューペアのマップ配列。

  各キーは UTF-8 文字列で、1 ~ 128 バイト長です。

  各値は ComputeEnvironmentConfiguration オブジェクトです。

  接続でサポートされているコンピューティング環境。

- PhysicalConnectionRequirements – キーバリューペアのマップ配列。

各キーは UTF-8 文字列で、1 ~ 128 バイト長です。

各値は プロパティ オブジェクトです。

接続の物理的な要件 (VPC、サブネット、セキュリティグループの仕様など)。

- `AthenaConnectionProperties` – キーバリューペアのマップ配列。

  各キーは UTF-8 文字列で、1 ~ 128 バイト長です。

  各値は プロパティ オブジェクトです。

  Athena コンピューティング環境に固有の接続プロパティ。

- `PythonConnectionProperties` – キーバリューペアのマップ配列。

  各キーは UTF-8 文字列で、1 ~ 128 バイト長です。

  各値は プロパティ オブジェクトです。

  Python コンピューティング環境に固有の接続プロパティ。

- `SparkConnectionProperties` – キーバリューペアのマップ配列。

  各キーは UTF-8 文字列で、1 ~ 128 バイト長です。

  各値は プロパティ オブジェクトです。

  Spark コンピューティング環境に固有の接続プロパティ。

エラー

- `ValidationException`
- `InvalidInputException`
- `InternalServiceException`

ListConnectionTypes アクション (Python: list_connection_types)

`ListConnectionTypes` API は、AWS Glue で使用可能な接続タイプを学習するための検出メカニズムを提供します。応答には、接続タイプのリストと、各接続タイプでサポートされている内容の概要が含まれます。リストに挙げられている接続タイプは、`CreateConnection` API の `ConnectionType` 値に対してサポートされているオプションのセットです。

リクエスト

- MaxResults – 1 ~ 1000 の数値 (整数)。

  返される結果の最大数。
- NextToken - UTF-8 文字列。1 ~ 2,048 バイト長。Custom string pattern #11 に一致。

  継続トークン (これが継続呼び出しの場合)。

応答

- ConnectionTypes – ConnectionTypeBrief オブジェクトの配列。

  サポートされている接続タイプに関する簡単な情報を含む ConnectionTypeBrief オブジェクトのリスト。
- NextToken - UTF-8 文字列。1 ~ 2,048 バイト長。Custom string pattern #11 に一致。

  継続トークン (現在のリストが最後ではない場合に表示されます)。

エラー

- InternalServiceException

ConnectionTypeBrief 構造

ListConnectionTypes API によって返される、サポートされている接続タイプに関する簡単な情報。

フィールド

- ConnectionType – UTF-8 文字列 (有効な値: JDBC | SFTP | MONGODB | KAFKA | NETWORK | MARKETPLACE | CUSTOM | SALESFORCE | VIEW_VALIDATION_REDSHIFT | VIEW_VALIDATION_ATHENA | GOOGLEADS | GOOGLESHEETS | GOOGLEANALYTICS4 | SERVICENOW | MARKETO | SAPODATA | ZENDESK | JIRACLOUD | NETSUITEERP | HUBSPOT | FACEBOOKADS | INSTAGRAMADS | ZOHOCRM | SALESFORCEPARDOT | SALESFORCEMARKETINGCLOUD | SLACK | STRIPE | INTERCOM | SNAPCHATADS)。

  接続タイプの名前。
- DisplayName – UTF-8 文字列、1 ~ 128 バイト長。

AWS Glue コンソールに表示される接続タイプの人間が読み取れる名前。

- `Vendor` – UTF-8 文字列、1 ~ 128 バイト長。

  この接続タイプを作成または管理するベンダーまたはプロバイダーの名前。

- `Description` – UTF-8 文字列、1024 バイト長以下。

  接続タイプの説明。

- `Categories` – UTF-8 文字列の配列。

  この接続タイプが属するカテゴリのリスト。カテゴリは、ユーザーがユースケースに基づいて適切な接続タイプをフィルタリングして見つけるために役立ちます。

- `Capabilities` – 機能 オブジェクト。

  サポートされている認証タイプ、データインターフェイスタイプ (コンピューティング環境)、コネクタのデータオペレーション。

- `LogoUrl` – UTF–8 文字列。

  接続タイプに関連付けられたロゴの URL。

- `ConnectionTypeVariants` – ConnectionTypeVariant オブジェクトの配列。

  この接続タイプで利用可能なバリアントのリスト。さまざまなバリアントは、同じ一般的な接続タイプの特定ユースケースや実装に特殊な設定を提供する場合があります。

ConnectionTypeVariant の構造

AWS Glue データカタログの接続タイプバリアントを表します。接続タイプバリアントは、同じ一般的な接続タイプのさまざまな実装に特定の設定および動作があります。

フィールド

- `ConnectionTypeVariantName` – UTF-8 文字列、1 ~ 128 バイト長。

  接続タイプバリアントの一意の識別子。この名前は、接続タイプの特定バリアントを識別するために内部で使用されます。

- `DisplayName` – UTF-8 文字列、1 ~ 128 バイト長。

  AWS Glue コンソールに表示される接続タイプバリアントにおいて、人間が読み取れる名前。

- `Description` – UTF-8 文字列、1024 バイト長以下。

目的、ユースケース、特定の設定要件など、接続タイプバリアントの詳細な説明。

- `LogoUrl` – UTF–8 文字列。

  接続タイプバリアントに関連付けられたロゴの URL。

データ型

- 検証構造
- AuthConfiguration 構造
- Capabilities 構造
- Property 構造
- AllowedValue 構造
- ComputeEnvironmentConfiguration 構造

検証構造

接続プロパティでの検証の実行方法を定義します。

フィールド

- `ValidationType` – 必須: UTF-8 文字列 (有効な値: `REGEX` | `RANGE`)。

  実行する検証のタイプ (`REGEX` など)。
- `Patterns` – UTF-8 文字列の配列。

  検証に適用されるパターンのリスト。
- `Description` – 必須: UTF-8 文字列、1 ~ 1024 バイト長。

  検証の説明。
- `MaxLength` – 数値 (整数)。

  文字列接続プロパティの最大長。
- `Maximum` – 数値 (整数)。

  検証の `RANGE` タイプを指定するときの最大値。
- `Minimum` – 数値 (整数)。

検証の RANGE タイプを指定するときの最小値。

AuthConfiguration 構造

DescribeConnectionType API によって返される接続の認証設定。

フィールド

- AuthenticationType – 必須: プロパティ オブジェクト。

  接続の認証のタイプ。
- SecretArn – プロパティ オブジェクト。

  Secrets Manager の Amazon リソースネーム (ARN)。
- OAuth2Properties – キーバリューペアのマップ配列。

  各キーは UTF-8 文字列で、1 ~ 128 バイト長です。

  各値は プロパティ オブジェクトです。

  OAuth2 プロパティのキーと値のペアのマップ。各値は Property オブジェクトです。
- BasicAuthenticationProperties – キーバリューペアのマップ配列。

  各キーは UTF-8 文字列で、1 ~ 128 バイト長です。

  各値は プロパティ オブジェクトです。

  OAuth2 プロパティのキーと値のペアのマップ。各値は Property オブジェクトです。
- CustomAuthenticationProperties – キーバリューペアのマップ配列。

  各キーは UTF-8 文字列で、1 ~ 128 バイト長です。

  各値は プロパティ オブジェクトです。

  カスタム認証プロパティのキーと値のペアのマップ。各値は Property オブジェクトです。

Capabilities 構造

DescribeConnectionType API によって返される、サポートされている認証タイプを指定します。

フィールド

- SupportedAuthenticationTypes – 必須: UTF-8 文字列の配列。

  サポートされている認証タイプのリスト。
- SupportedDataOperations – 必須: UTF-8 文字列の配列。

  サポートされているデータオペレーションのリスト。
- SupportedComputeEnvironments – 必須: UTF-8 文字列の配列。

  サポートされているコンピューティング環境のリスト。

Property 構造

コンピューティング環境の接続タイプを定義するオブジェクト。

フィールド

- Name – 必須: UTF-8 文字列。1 ~ 128 バイト長。

  プロパティの名前。
- Description – 必須: UTF-8 文字列。1024 以下のバイト長。

  プロパティの説明。
- Required – 必須: Boolean。

  プロパティが必須かどうかを示します。
- DefaultValue – UTF–8 文字列。

  プロパティのデフォルト値。
- PropertyTypes – 必須: UTF-8 文字列の配列。

  プロパティのタイプを説明します。
- AllowedValues – allowedValues オブジェクトの配列。

  プロパティに許可される値を表す AllowedValue オブジェクトのリスト。
- DataOperationScopes – UTF-8 文字列の配列。

  どのデータオペレーションをプロパティに適用できるかを示します。

AllowedValue 構造

プロパティに許可される値を表すオブジェクト。

フィールド

- `Description` – UTF-8 文字列、1024 バイト長以下。

  許可される値の説明。
- `Value` – 必須: UTF-8 文字列。1 ~ 128 バイト長。

  プロパティに許可される値。

ComputeEnvironmentConfiguration 構造

`DescribeConnectionType` API によって返されるコンピューティング環境 (Spark、Python、Athena など) の設定を含むオブジェクト。

フィールド

- `Name` – 必須: UTF-8 文字列。1 ~ 128 バイト長。

  コンピューティング環境設定の名前。
- `Description` – 必須: UTF-8 文字列。1,024 以下のバイト長。

  コンピューティング環境の説明。
- `ComputeEnvironment` – 必須: UTF-8 文字列 (有効な値: `SPARK` | `ATHENA` | `PYTHON`)。

  コンピューティング環境のタイプ。
- `SupportedAuthenticationTypes` – 必須: UTF-8 文字列の配列。

  コンピューティング環境でサポートされている認証タイプ。
- `ConnectionOptions` – 必須: キーバリューペアのマップ配列。

  各キーは UTF-8 文字列で、1 ~ 128 バイト長です。

  各値は プロパティ オブジェクトです。

  コンピューティング環境の接続オプションとして使用されるパラメータ。
- `ConnectionPropertyNameOverrides` – 必須: キーバリューペアのマップ配列。

各キーは UTF-8 文字列で、1 ~ 128 バイト長です。

各値は UTF-8 文字列で、1 ~ 128 バイト長です。

接続プロパティ名はコンピューティング環境に対して上書きされます。

- ConnectionOptionNameOverrides – 必須: キーバリューペアのマップ配列。

  各キーは UTF-8 文字列で、1 ~ 128 バイト長です。

  各値は UTF-8 文字列で、1 ~ 128 バイト長です。

  接続オプション名はコンピューティング環境に対して上書きされます。

- ConnectionPropertiesRequiredOverrides – 必須: UTF-8 文字列の配列。

  コンピューティング環境の上書きとして必要な接続プロパティ。

- PhysicalConnectionPropertiesRequired – ブール。

  コンピューティング環境に PhysicalConnectionProperties が必要かどうかを示します。

接続メタデータとプレビュー API

以下の接続 API では、接続メタデータを記述するオペレーションについて説明します。

データ型

- エンティティ構造
- フィールド構造

エンティティ構造

特定の ConnectionType でサポートされているエンティティ。

フィールド

- EntityName – UTF-8 文字列。

  エンティティの名前。

- Label – UTF–8 文字列。

エンティティに使用されるラベル。

- IsParentEntity – ブール。

  一覧表示できるサブオブジェクトがあるかどうかを判断できるブール値。

- Description – UTF–8 文字列。

  エンティティの説明。

- Category – UTF–8 文字列。

  応答に含まれるエンティティのタイプ。この値はソース接続によって異なります。例えば、Salesforce の場合は SObjects、Amazon Redshift などのソースの場合は databases、schemas、tables です。

- CustomProperties – キーバリューペアのマップ配列。

  各キーは UTF-8 文字列。

  各値は UTF-8 文字列。

  コネクタによってエンティティに返される可能性のあるキーのオプションのマップ。

フィールド構造

Field オブジェクトには、コネクタのフィールドに関連付けられたさまざまなプロパティに関する情報が含まれています。

フィールド

- FieldName – UTF-8 文字列。

  フィールドの一意の識別子。

- Label – UTF–8 文字列。

  フィールドに使用される読み取り可能なラベル。

- Description – UTF–8 文字列。

  フィールドの説明。

- FieldType – UTF-8 文字列 (有効な値: INT | SMALLINT | BIGINT | FLOAT | LONG | DATE | BOOLEAN | MAP | ARRAY | STRING | TIMESTAMP | DECIMAL | BYTE | SHORT | DOUBLE | STRUCT)。

フィールドのデータ型。

- IsPrimaryKey – ブール。

  このフィールドを特定のエンティティのプライマリキーとして使用できるかどうかを示します。

- IsNullable – ブール。

  このフィールドを null にできるかどうかを示します。

- IsRetrievable – ブール。

  このフィールドを SQL クエリの Select 句に追加できるかどうか、または取得可能かどうかを示します。

- IsFilterable – ブール。

  データをクエリするときに、このフィールドを SQL ステートメントのフィルター句 (WHERE 句) で使用できるかどうかを示します。

- IsPartitionable – ブール。

  特定のフィールドを SaaS に対して行われたクエリのパーティション化に使用できるかどうかを示します。

- IsCreateable – ブール。

  このフィールドを送信先書き込みの一部として作成できるかどうかを示します。

- IsUpdateable – ブール。

  このフィールドを送信先書き込みの一部として更新できるかどうかを示します。

- IsUpsertable – ブール。

  このフィールドを送信先書き込みの一部として upsert できるかどうかを示します。

- IsDefaultOnCreate – ブール。

  オブジェクトの作成時にこのフィールドが自動的に入力されるかどうかを示します ([作成時刻] のタイムスタンプなど)。

- SupportedValues – UTF-8 文字列の配列。

  フィールドのサポートされている値のリスト。

- SupportedFilterOperators – UTF-8 文字列の配列。

このフィールドのサポートされているフィルター演算子を示します。

- `ParentField` – UTF–8 文字列。

  ネストされたフィールドの親フィールド名。

- `NativeDataType` – UTF–8 文字列。

  「picklist」や「textarea」など、Salesforce から SaaS API によって返されるデータ型。

- `CustomProperties` – キーバリューペアのマップ配列。

  各キーは UTF-8 文字列。

  各値は UTF-8 文字列。

  返される可能性のあるキーのオプションのマップ。

操作

- ListEntities アクション (Python: list_entities)
- DescribeEntity アクション (Python: describe_entity)
- GetEntityRecords アクション (Python: get_entity_records)

ListEntities アクション (Python: list_entities)

接続タイプでサポートされている利用可能なエンティティを返します。

リクエスト

- `ConnectionName` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  任意の接続タイプをクエリするために必要な認証情報を持つ接続の名前。

- `CatalogId` – カタログ ID 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  接続を含むカタログのカタログ ID。これは null にすることができます。デフォルトでは、AWS アカウント ID がカタログ ID になります。

- `ParentEntityName` – UTF–8 文字列。

  子エンティティを一覧表示させる親エンティティの名前。このパラメータは、子エンティティを一覧表示するためにエンティティの完全修飾パスを取ります。

- NextToken - UTF-8 文字列。1 ~ 2,048 バイト長。Custom string pattern #11 に一致。

  継続トークン (これが継続呼び出しの場合)。
- DataStoreApiVersion – UTF-8 文字列。1 ~ 256 バイト長。Custom string pattern #23 に一致。

  SaaS コネクタの API バージョン。

応答

- Entities – エンティティ オブジェクトの配列。

  Entity オブジェクトのリスト。
- NextToken - UTF-8 文字列。1 ~ 2,048 バイト長。Custom string pattern #11 に一致。

  継続トークン (現在のセグメントが最後ではない場合に表示されます)。

エラー

- EntityNotFoundException
- OperationTimeoutException
- InvalidInputException
- GlueEncryptionException
- ValidationException
- FederationSourceException
- AccessDeniedException

DescribeEntity アクション (Python: describe_entity)

接続タイプで使用されるエンティティに関する詳細を、選択したエンティティの各フィールドのデータモデルの説明と併せて表示します。

応答には、エンティティを構成しているすべてのフィールドが含まれます。

リクエスト

- ConnectionName – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  接続タイプの認証情報を含む接続の名前。

- CatalogId – カタログ ID 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  接続を含むカタログのカタログ ID。これは null にすることができます。デフォルトでは、AWS アカウント ID がカタログ ID になります。

- EntityName – 必須: UTF–8 文字列。

  接続タイプから記述するエンティティの名前。

- NextToken - UTF-8 文字列。1 ~ 2,048 バイト長。Custom string pattern #11 に一致。

  継続トークン (これが継続呼び出しの場合)。

- DataStoreApiVersion – UTF-8 文字列。1 ~ 256 バイト長。Custom string pattern #23 に一致。

  データストアに使用される API のバージョン。

応答

- Fields – フィールド オブジェクトの配列。

  コネクタエンティティのフィールドを記述します。これは Field オブジェクトのリストです。Field はデータベースの列に非常によく似ています。Field オブジェクトには、コネクタのフィールドに関連付けられたさまざまなプロパティに関する情報が含まれています。

- NextToken - UTF-8 文字列。1 ~ 2,048 バイト長。Custom string pattern #11 に一致。

  継続トークン (現在のセグメントが最後ではない場合に表示されます)。

エラー

- EntityNotFoundException
- OperationTimeoutException
- InvalidInputException
- GlueEncryptionException
- ValidationException
- FederationSourceException
- AccessDeniedException

GetEntityRecords アクション (Python: get_entity_records)

この API は、特定の接続タイプ、またはネイティブの Amazon S3 ベースの AWS Glue データカタログからプレビューデータをクエリするときに使用します。

JSON BLOB の配列としてレコードを返します。各レコードは、DescribeEntity API で定義されたフィールドタイプに基づいて、Jackson JsonNode を使用してフォーマットされます。

Spark コネクタは、DescribeEntity API と同じデータ型マッピングに従ってスキーマを生成します。Spark コネクタは、行を返すときに、スキーマに一致する適切なデータ型にデータを変換します。

リクエスト

- ConnectionName – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  接続タイプの認証情報を含む接続の名前。
- CatalogId – カタログ ID 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  接続を含むカタログのカタログ ID。これは null にすることができます。デフォルトでは、AWS アカウント ID がカタログ ID になります。
- EntityName – 必須: UTF–8 文字列。

  指定された接続タイプからプレビューデータをクエリするエンティティの名前。
- NextToken - UTF-8 文字列。1 ~ 2,048 バイト長。Custom string pattern #11 に一致。

  継続トークン (これが継続呼び出しの場合)。
- DataStoreApiVersion – UTF-8 文字列。1 ~ 256 バイト長。Custom string pattern #23 に一致。

  SaaS コネクタの API バージョン。
- ConnectionOptions – キーと値のペアのマップ配列。100 ペア以下。

  各キーは UTF-8 文字列、1 ~ 256 バイト長で、Custom string pattern #18 に一致します。

  各値は UTF-8 文字列、1 ~ 256 バイト長で、Custom string pattern #17 に一致します。

  データのクエリに必要なコネクタオプション。
- FilterPredicate – UTF-8 文字列。1 ~ 100000 バイト長。

  クエリリクエストに適用可能なフィルター述語。

- `Limit` – 必須: 数値 (長さ)。1 ~ 1000。

  リクエストで取得されるレコードの数を制限します。
- `OrderBy` – UTF–8 文字列。

  応答のプレビューデータを指示するパラメータ。
- `SelectedFields` – UTF-8 文字列の配列。1 ~ 1000 の文字列。

  プレビューデータの一部として取得するフィールドのリスト。

レスポンス

- `Records` – 構造の配列。

  リクエストされた オブジェクトのリスト。
- `NextToken` - UTF-8 文字列。1 ~ 2,048 バイト長。Custom string pattern #11 に一致。

  継続トークン (現在のセグメントが最後ではない場合に表示されます)。

エラー

- `EntityNotFoundException`
- `OperationTimeoutException`
- `InvalidInputException`
- `GlueEncryptionException`
- `ValidationException`
- `FederationSourceException`
- `AccessDeniedException`

ユーザー定義関数 API

ユーザー定義関数 API では、関数の処理に使用する AWS Glue データ型およびオペレーションについて説明します。

データ型

- UserDefinedFunction 構造

- UserDefinedFunctionInput 構造

UserDefinedFunction 構造

Hive ユーザー定義関数 (UDF) 定義と同等のものを表します。

フィールド

- `FunctionName` – UTF-8 文字列、1～255 バイト長、Single-line string pattern に一致。

  関数の名前

- `DatabaseName` – UTF-8 文字列、1～255 バイト長、Single-line string pattern に一致。

  関数を含むカタログデータベースの名前。

- `ClassName` – UTF-8 文字列、1～255 バイト長、Single-line string pattern に一致。

  関数コードを含む Java クラス。

- `OwnerName` – UTF-8 文字列、1～255 バイト長、Single-line string pattern に一致。

  関数の所有者。

- `OwnerType` – UTF-8 文字列 (有効な値: `USER` | `ROLE` | `GROUP`)。

  所有者のタイプ。

- `CreateTime` – タイムスタンプ。

  関数の作成時刻。

- `ResourceUris` – ResourceUri オブジェクトの配列。構造 1000 個以下。

  関数のリソース URI。

- `CatalogId` – カタログ ID 文字列、1～255 バイト長、Single-line string pattern に一致。

  関数が存在する Data Catalog の ID。

UserDefinedFunctionInput 構造

ユーザー定義関数の作成または更新に使用される構造。

フィールド

- FunctionName – UTF-8 文字列、1 ~ 255 バイト長、Single-line string pattern に一致。

  関数の名前

- ClassName – UTF-8 文字列、1 ~ 255 バイト長、Single-line string pattern に一致。

  関数コードを含む Java クラス。

- OwnerName – UTF-8 文字列、1 ~ 255 バイト長、Single-line string pattern に一致。

  関数の所有者。

- OwnerType – UTF-8 文字列 (有効な値: USER | ROLE | GROUP)。

  所有者のタイプ。

- ResourceUris – ResourceUri オブジェクトの配列。構造 1000 個以下。

  関数のリソース URI。

操作

- CreateUserDefinedFunction アクション (Python: create_user_defined_function)
- UpdateUserDefinedFunction アクション (Python: update_user_defined_function)
- DeleteUserDefinedFunction アクション (Python: delete_user_defined_function)
- GetUserDefinedFunction アクション (Python: get_user_defined_function)
- GetUserDefinedFunctions アクション (Python: get_user_defined_functions)

CreateUserDefinedFunction アクション (Python: create_user_defined_function)

データカタログで新しい関数定義を作成します。

リクエスト

- CatalogId – カタログ ID 文字列、1 ~ 255 バイト長、Single-line string pattern に一致。

  関数を作成するデータカタログの ID。設定しない場合は、AWS アカウント ID がデフォルトで使用されます。

- DatabaseName – 必須: UTF-8 文字列、1 ~ 255 バイト長、Single-line string pattern に一致。

関数を作成するカタログデータベースの名前。

- FunctionInput – 必須: UserDefinedFunctionInput オブジェクト。

  データカタログで作成する関数を定義する FunctionInput オブジェクト。

レスポンス

- 応答パラメータはありません。

エラー

- AlreadyExistsException
- InvalidInputException
- InternalServiceException
- EntityNotFoundException
- OperationTimeoutException
- ResourceNumberLimitExceededException
- GlueEncryptionException

UpdateUserDefinedFunction アクション (Python: update_user_defined_function)

データカタログで既存の関数定義を更新します。

リクエスト

- CatalogId – カタログ ID 文字列、1 ~ 255 バイト長、Single-line string pattern に一致。

  更新する関数が存在するデータカタログの ID。設定しない場合は、AWS アカウント ID がデフォルトで使用されます。

- DatabaseName – 必須: UTF-8 文字列、1 ~ 255 バイト長、Single-line string pattern に一致。

  更新する関数が存在するカタログデータベースの名前。

- FunctionName – 必須: UTF-8 文字列、1 ~ 255 バイト長、Single-line string pattern に一致。

  関数の名前

- FunctionInput – 必須: UserDefinedFunctionInput オブジェクト。

データカタログで関数を再定義する FunctionInput オブジェクト。

レスポンス

- 応答パラメータはありません。

エラー

- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException
- GlueEncryptionException

DeleteUserDefinedFunction アクション (Python: delete_user_defined_function)

データカタログから既存の関数定義を削除します。

リクエスト

- CatalogId – カタログ ID 文字列、1 ~ 255 バイト長、Single-line string pattern に一致。

  削除する関数が存在するデータカタログの ID。提供されない場合は、AWS アカウント ID がデフォルトで使用されます。

- DatabaseName – 必須: UTF-8 文字列、1 ~ 255 バイト長、Single-line string pattern に一致。

  削除する関数が存在するカタログデータベースの名前。

- FunctionName – 必須: UTF-8 文字列、1 ~ 255 バイト長、Single-line string pattern に一致。

  削除する関数定義の名前。

レスポンス

- 応答パラメータはありません。

エラー

- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException

GetUserDefinedFunction アクション (Python: get_user_defined_function)

データカタログから指定された関数定義を取得します。

リクエスト

- CatalogId – カタログ ID 文字列、1 ~ 255 バイト長、Single-line string pattern に一致。

  取得する関数が存在するデータカタログの ID。設定しない場合は、AWS アカウント ID がデフォルトで使用されます。
- DatabaseName – 必須: UTF-8 文字列、1 ~ 255 バイト長、Single-line string pattern に一致。

  削除する関数が存在するカタログデータベースの名前。
- FunctionName – 必須: UTF-8 文字列、1 ~ 255 バイト長、Single-line string pattern に一致。

  関数の名前

レスポンス

- UserDefinedFunction – UserDefinedFunction オブジェクト。

  リクエストされた関数定義。

エラー

- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException
- GlueEncryptionException

GetUserDefinedFunctions アクション (Python: get_user_defined_functions)

データカタログから複数の関数定義を取得します。

リクエスト

- `CatalogId` – カタログ ID 文字列、1 ~ 255 バイト長、Single-line string pattern に一致。

  取得する関数が存在するデータカタログの ID。設定しない場合は、AWS アカウント ID がデフォルトで使用されます。
- `DatabaseName` – UTF-8 文字列、1 ~ 255 バイト長、Single-line string pattern に一致。

  関数が存在するカタログデータベースの名前。何も指定されていない場合は、カタログ全体のすべてのデータベースからの関数が返されます。
- `Pattern` – 必須: UTF-8 文字列、1 ~ 255 バイト長、Single-line string pattern に一致。

  返される関数定義をフィルタリングするオプションの function-name パターン文字列。
- `NextToken` – UTF-8 文字列。

  継続トークン (これが継続呼び出しの場合)。
- `MaxResults` – 数値 (integer)。1 ~ 100。

  1 回の応答で返す関数の最大数。

応答

- `UserDefinedFunctions` – UserDefinedFunction オブジェクトの配列。

  リクエストされた関数定義のリスト。
- `NextToken` – UTF-8 文字列。

  戻された関数のリストに最後のリクエストされた関数が含まれていない場合は、継続トークン。

エラー

- `EntityNotFoundException`
- `InvalidInputException`
- `OperationTimeoutException`

- InternalServiceException
- GlueEncryptionException

Athena カタログを AWS Glue にインポートする

Migration API では、Athena データカタログの AWS Glue への移行と関係のある AWS Glue データ型とオペレーションについて説明します。

データ型

- CatalogImportStatus 構造

CatalogImportStatus 構造

移行ステータス情報を含む構造。

フィールド

- ImportCompleted – ブール。

  移行が完了した場合は True、それ以外の場合は False です。

- ImportTime – タイムスタンプ。

  移行を開始した時刻。

- ImportedBy – UTF-8 文字列、1～255 バイト長、Single-line string pattern に一致。

  移行を開始したユーザーの名前。

操作

- ImportCatalogToGlue アクション (Python: import_catalog_to_glue)
- GetCatalogImportStatus アクション (Python: get_catalog_import_status)

ImportCatalogToGlue アクション (Python: import_catalog_to_glue)

既存の Amazon Athena Data Catalog を AWS Glue にインポートする

リクエスト

- `CatalogId` – カタログ ID 文字列、1 ~ 255 バイト長、Single-line string pattern に一致。

  インポートするカタログの ID。現在、これは AWS アカウント ID である必要があります。

レスポンス

- 応答パラメータはありません。

エラー

- `InternalServiceException`
- `OperationTimeoutException`

GetCatalogImportStatus アクション (Python: get_catalog_import_status)

移行操作のステータスを取得します。

リクエスト

- `CatalogId` – カタログ ID 文字列、1 ~ 255 バイト長、Single-line string pattern に一致。

  移行するカタログの ID。現在、これは AWS アカウント ID である必要があります。

レスポンス

- `ImportStatus` – CatalogImportStatus オブジェクト。

  指定したカタログ移行のステータス。

エラー

- `InternalServiceException`
- `OperationTimeoutException`

テーブルオプティマイザー API

テーブルオプティマイザー API は、圧縮を有効にして読み取りパフォーマンスを改善するための AWS Glue API の説明を記述します。

データ型

- TableOptimizer の構造
- TableOptimizerConfiguration の構造
- TableOptimizerVpcConfiguration 構造
- CompactionConfiguration 構造
- IcebergCompactionConfiguration 構造
- TableOptimizerRun の構造
- BatchGetTableOptimizerEntry の構造
- BatchTableOptimizer の構造
- BatchGetTableOptimizerError の構造
- RetentionConfiguration 構造
- IcebergRetentionConfiguration 構造
- OrphanFileDeletionConfiguration 構造
- IcebergOrphanFileDeletionConfiguration 構造
- CompactionMetrics 構造
- RetentionMetrics 構造
- OrphanFileDeletionMetrics 構造
- IcebergCompactionMetrics 構造
- IcebergRetentionMetrics 構造
- IcebergOrphanFileDeletionMetrics 構造
- RunMetrics の構造

TableOptimizer の構造

テーブルに関連付けられたオプティマイザーに関する詳細が含まれます。

フィールド

- type – UTF-8 文字列 (有効な値: compaction="COMPACTION" | retention="RETENTION" | orphan_file_deletion="ORPHAN_FILE_DELETION")。

  テーブルオプティマイザーのタイプ。有効な値は以下のとおりです。

  - compaction: テーブルオプティマイザーで圧縮を管理する場合。
  - retention: テーブルオプティマイザーでスナップショットの保持を管理する場合。
  - orphan_file_deletion: テーブルオプティマイザーで孤立ファイルの削除を管理する場合。
- configuration – TableOptimizerConfiguration オブジェクト。

  テーブルオプティマイザーを作成または更新する際に指定された TableOptimizerConfiguration オブジェクト。
- lastRun – TableOptimizerRun オブジェクト。

  テーブルオプティマイザーの前回の実行を表す TableOptimizerRun オブジェクト。

TableOptimizerConfiguration の構造

テーブルオプティマイザーの設定に関する詳細が含まれます。テーブルオプティマイザーを作成または更新する際に、この設定を渡します。

フィールド

- roleArn - UTF-8 文字列。20 ~ 2,048 バイト長。Single-line string pattern に一致。

  呼び出し元によって渡されるロール。このロールは、呼び出し元に代わってオプティマイザーに関連付けられたリソースを更新するための許可をサービスに付与します。
- enabled – ブール。

  テーブル最適化が有効かどうか。
- vpcConfiguration – TableOptimizerVpcConfiguration オブジェクト。

  テーブルオプティマイザーの VPC の設定を表す TableOptimizerVpcConfiguration オブジェクト。

  この設定は、カスタマー VPC 内のテーブルを最適化するために必要です。
- compactionConfiguration – CompactionConfiguration オブジェクト。

圧縮オプティマイザの設定。この設定は、クエリのパフォーマンスを向上させ、ストレージコストを削減するために、テーブル内のデータファイルを圧縮する方法を定義します。

- `retentionConfiguration` – RetentionConfiguration オブジェクト。

  スナップショット保持オプティマイザー用の設定。

- `orphanFileDeletionConfiguration` – OrphanFileDeletionConfiguration オブジェクト。

  孤立ファイル削除オプティマイザー用の設定。

TableOptimizerVpcConfiguration 構造

テーブルオプティマイザーの VPC 設定を記述するオブジェクト。

この設定は、カスタマー VPC 内のテーブルを最適化するために必要です。

フィールド

- `glueConnectionName` – UTF-8 文字列、少なくとも 1 バイト長。

  テーブルオプティマイザーの VPC に使用される AWS Glue 接続の名前。

CompactionConfiguration 構造

圧縮オプティマイザの設定。この設定は、クエリのパフォーマンスを向上させ、ストレージコストを削減するために、テーブル内のデータファイルを圧縮する方法を定義します。

フィールド

- `icebergConfiguration` – IcebergCompactionConfiguration オブジェクト。

  Iceberg 圧縮オプティマイザの設定。

IcebergCompactionConfiguration 構造

Iceberg 圧縮オプティマイザの設定。この設定は、Iceberg テーブルのデータファイルのレイアウトを最適化するためのパラメータを定義します。

フィールド

- strategy – UTF-8 文字列 (有効な値: binpack="BINPACK" | sort="SORT" | z-order="ZORDER")。

  圧縮に使用する戦略。次の値を指定できます:

  - binpack: 小さなファイルを大きなファイルに結合します。通常は 100MB を超えるサイズをターゲットにしながら、保留中の削除を適用します。これは、ほとんどのユースケースで推奨される圧縮戦略です。
  - sort: 圧縮中に階層的にソートされる指定された列に基づいてデータを整理し、フィルタリングされた操作のクエリパフォーマンスを向上させます。この戦略は、クエリが特定の列を頻繁にフィルタリングする場合に推奨されます。この戦略を使用するには、まず sort_order テーブルプロパティを使用して Iceberg テーブルプロパティでソート順序を定義する必要があります。
  - z-order: ソートに使用できる単一のスカラー値に複数の属性を統合することでデータ整理を最適化し、複数のディメンションにまたがって効率的なクエリを実行できるようにします。この戦略は、複数のディメンションにまたがるデータを同時にクエリする必要がある場合に推奨します。この戦略を使用するには、まず sort_order テーブルプロパティを使用して Iceberg テーブルプロパティでソート順序を定義する必要があります。

  入力が指定されていない場合は、デフォルト値の 'binpack' が使用されます。

TableOptimizerRun の構造

テーブルオプティマイザーの実行の詳細が含まれます。

フィールド

- eventType – UTF-8 文字列 (有効な値: starting="STARTING" | completed="COMPLETED" | failed="FAILED" | in_progress="IN_PROGRESS")。

  テーブルオプティマイザーの実行のステータスを表すイベントタイプ。
- startTimestamp – タイムスタンプ。

  Lake Formation 内で圧縮ジョブが開始された時点のエポックタイムスタンプを表します。
- endTimestamp – タイムスタンプ。

  圧縮ジョブが終了した時点のエポックタイムスタンプを表します。
- metrics – RunMetrics オブジェクト。

オプティマイザーの実行のメトリクスを含む RunMetrics オブジェクト。

このメンバーは廃止されました。圧縮、保持、孤立ファイルの削除については、個々のメトリクスメンバーを参照してください。

- error – UTF–8 文字列。

  オプティマイザーの実行中に発生したエラー。

- compactionMetrics – CompactionMetrics オブジェクト。

  オプティマイザーの実行のメトリクスを含む CompactionMetrics オブジェクト。

- compactionStrategy – UTF-8 文字列 (有効な値: binpack="BINPACK" | sort="SORT" | z-order="ZORDER")。

  圧縮実行に使用される戦略。圧縮プロセス中にファイルがどのように選択および組み合わされたかを判断するために適用されたアルゴリズムを示します。次の値を指定できます:

  - binpack: 小さなファイルを大きなファイルに結合します。通常は 100MB を超えるサイズをターゲットにしながら、保留中の削除を適用します。これは、ほとんどのユースケースで推奨される圧縮戦略です。
  - sort: 圧縮中に階層的にソートされる指定された列に基づいてデータを整理し、フィルタリングされた操作のクエリパフォーマンスを向上させます。この戦略は、クエリが特定の列を頻繁にフィルタリングする場合に推奨されます。この戦略を使用するには、まず sort_order テーブルプロパティを使用して Iceberg テーブルプロパティでソート順序を定義する必要があります。
  - z-order: ソートに使用できる単一のスカラー値に複数の属性を統合することでデータ整理を最適化し、複数のディメンションにまたがって効率的なクエリを実行できるようにします。この戦略は、複数のディメンションにまたがるデータを同時にクエリする必要がある場合に推奨します。この戦略を使用するには、まず sort_order テーブルプロパティを使用して Iceberg テーブルプロパティでソート順序を定義する必要があります。

- retentionMetrics – RetentionMetrics オブジェクト。

  オプティマイザーの実行のメトリクスを含む RetentionMetrics オブジェクト。

- orphanFileDeletionMetrics – OrphanFileDeletionMetrics オブジェクト。

  オプティマイザーの実行のメトリクスを含む OrphanFileDeletionMetrics オブジェクト。

BatchGetTableOptimizerEntry の構造

BatchGetTableOptimizer オペレーションで取得するテーブルオプティマイザーを表します。

フィールド

- catalogId – カタログ ID 文字列、1～255 バイト長、「Single-line string pattern」に一致。

  テーブルのカタログ ID。
- databaseName – UTF-8 文字列、少なくとも 1 バイト長。

  テーブルが存在するカタログのデータベースの名前。
- tableName – UTF-8 文字列、少なくとも 1 バイト長。

  テーブルの名前。
- type – UTF-8 文字列 (有効な値: compaction="COMPACTION" | retention="RETENTION" | orphan_file_deletion="ORPHAN_FILE_DELETION")。

  テーブルオプティマイザーのタイプ。

BatchTableOptimizer の構造

BatchGetTableOptimizer オペレーションによって返されたいずれかのテーブルオプティマイザーの詳細が含まれます。

フィールド

- catalogId – カタログ ID 文字列、1～255 バイト長、「Single-line string pattern」に一致。

  テーブルのカタログ ID。
- databaseName – UTF-8 文字列、少なくとも 1 バイト長。

  テーブルが存在するカタログのデータベースの名前。
- tableName – UTF-8 文字列、少なくとも 1 バイト長。

  テーブルの名前。
- tableOptimizer – TableOptimizer オブジェクト。

テーブルオプティマイザーの設定と前回の実行に関する詳細を含む TableOptimizer オブジェクト。

BatchGetTableOptimizerError の構造

BatchGetTableOptimizer オペレーションによって返されたエラーリスト内のいずれかのエラーに関する詳細が含まれます。

フィールド

- error – ErrorDetail オブジェクト。

  エラーに関するコードとメッセージの詳細を含む ErrorDetail オブジェクト。

- catalogId – カタログ ID 文字列、1 ~ 255 バイト長、Single-line string pattern に一致。

  テーブルのカタログ ID。

- databaseName – UTF-8 文字列、少なくとも 1 バイト長。

  テーブルが存在するカタログのデータベースの名前。

- tableName – UTF-8 文字列、少なくとも 1 バイト長。

  テーブルの名前。

- type – UTF-8 文字列 (有効な値: compaction="COMPACTION" | retention="RETENTION" | orphan_file_deletion="ORPHAN_FILE_DELETION")。

  テーブルオプティマイザーのタイプ。

RetentionConfiguration 構造

スナップショット保持オプティマイザー用の設定。

フィールド

- icebergConfiguration – IcebergRetentionConfiguration オブジェクト。

  Iceberg スナップショット保持オプティマイザー用の設定。

IcebergRetentionConfiguration 構造

Iceberg スナップショット保持オプティマイザー用の設定。

フィールド

- `snapshotRetentionPeriodInDays` – 数値 (整数)。

  Iceberg スナップショットを保持する日数。入力が指定されていない場合は、対応する Iceberg テーブルの設定フィールドが使用されます。存在しない場合は、デフォルト値の 5 が使用されます。

- `numberOfSnapshotsToRetain` – 数値 (整数)。

  保持期間内に保持する Iceberg スナップショットの数。入力が指定されていない場合は、対応する Iceberg テーブルの設定フィールドが使用されます。存在しない場合は、デフォルト値の 1 が使用されます。

- `cleanExpiredFiles` – ブール。

  false に設定すると、スナップショットはテーブルメタデータからのみ削除され、基になるデータとメタデータファイルは削除されません。

OrphanFileDeletionConfiguration 構造

孤立ファイル削除オプティマイザー用の設定。

フィールド

- `icebergConfiguration` – IcebergOrphanFileDeletionConfiguration オブジェクト。

  Iceberg 孤立ファイル削除オプティマイザー用の設定。

IcebergOrphanFileDeletionConfiguration 構造

Iceberg 孤立ファイル削除オプティマイザー用の設定。

フィールド

- `orphanFileRetentionPeriodInDays` – 数値 (整数)。

ファイルを削除する前に孤立ファイルを保持する日数。入力が指定されていない場合は、デフォルト値の 3 が使用されます。

- `location` – UTF–8 文字列。

  ファイルを検索するディレクトリを指定します (デフォルトはテーブルの場所です)。最上位のテーブルの場所ではなく、サブディレクトリを選択することができます。

CompactionMetrics 構造

オプティマイザーの実行の圧縮メトリクスを含む構造。

フィールド

- `IcebergMetrics` – IcebergCompactionMetrics オブジェクト。

  オプティマイザーの実行の Iceberg 圧縮メトリクスを含む構造。

RetentionMetrics 構造

オプティマイザーの実行の保持メトリクスを含む構造。

フィールド

- `IcebergMetrics` – IcebergRetentionMetrics オブジェクト。

  オプティマイザーの実行の Iceberg 保持メトリクスを含む構造。

OrphanFileDeletionMetrics 構造

オプティマイザーの実行の孤立ファイル削除メトリクスを含む構造。

フィールド

- `IcebergMetrics` – IcebergOrphanFileDeletionMetrics オブジェクト。

  オプティマイザーの実行の Iceberg 孤立ファイル削除メトリクスを含む構造。

IcebergCompactionMetrics 構造

オプティマイザーの実行の Iceberg の圧縮メトリクス。

フィールド

- `DpuHours` – 数値 (double)。

  ジョブによって消費された DPU 時間数。
- `NumberOfDpus` – 数値 (整数)。

  ジョブによって消費された DPU の数。最近似の整数に切り上げられます。
- `JobDurationInHour` – 数値 (double)。

  ジョブの実行時間 (単位: 時間)。

IcebergRetentionMetrics 構造

オプティマイザーの実行の Iceberg のスナップショット保持メトリクス。

フィールド

- `DpuHours` – 数値 (double)。

  ジョブによって消費された DPU 時間数。
- `NumberOfDpus` – 数値 (整数)。

  ジョブによって消費された DPU の数。最近似の整数に切り上げられます。
- `JobDurationInHour` – 数値 (double)。

  ジョブの実行時間 (単位: 時間)。

IcebergOrphanFileDeletionMetrics 構造

オプティマイザーの実行の Iceberg の孤立ファイル削除メトリクス。

フィールド

- `DpuHours` – 数値 (double)。

ジョブによって消費された DPU 時間数。

- `NumberOfDpus` – 数値 (整数)。

  ジョブによって消費された DPU の数。最近似の整数に切り上げられます。

- `JobDurationInHour` – 数値 (double)。

  ジョブの実行時間 (単位: 時間)。

RunMetrics の構造

オプティマイザーの実行のメトリクス。

この構造は廃止されました。圧縮、保持、孤立ファイルの削除については、個々のメトリクスメンバーを参照してください。

フィールド

- `NumberOfBytesCompacted` – UTF-8 文字列。

  圧縮ジョブの実行によって削除されたバイト数。

- `NumberOfFilesCompacted` – UTF–8 文字列。

  圧縮ジョブの実行によって削除されたファイル数。

- `NumberOfDpus` – UTF–8 文字列。

  ジョブによって消費された DPU の数。最近似の整数に切り上げられます。

- `JobDurationInHour` – UTF–8 文字列。

  ジョブの実行時間 (単位: 時間)。

操作

- GetTableOptimizer アクション (Python: get_table_optimizer)
- BatchGetTableOptimizer アクション (Python: batch_get_table_optimizer)
- ListTableOptimizerRuns アクション (Python: list_table_optimizer_runs)
- CreateTableOptimizer アクション (Python: create_table_optimizer)

- DeleteTableOptimizer アクション (Python: delete_table_optimizer)
- UpdateTableOptimizer アクション (Python: update_table_optimizer)

GetTableOptimizer アクション (Python: get_table_optimizer)

指定されたテーブルに関連付けられているすべてのオプティマイザーの設定を返します。

リクエスト

- `CatalogId` – 必須: カタログ ID 文字列。1～255 バイト長。Single-line string pattern に一致。

  テーブルのカタログ ID。
- `DatabaseName` – 必須: UTF-8 文字列。1～255 バイト長。Single-line string pattern に一致。

  テーブルが存在するカタログのデータベースの名前。
- `TableName` – 必須: UTF-8 文字列。1～255 バイト長。Single-line string pattern に一致。

  テーブルの名前。
- `Type` – 必須: UTF-8 文字列 (有効な値: `compaction="COMPACTION"` | `retention="RETENTION"` | `orphan_file_deletion="ORPHAN_FILE_DELETION"`)。

  テーブルオプティマイザーのタイプ。

レスポンス

- `CatalogId` – カタログ ID 文字列、1～255 バイト長、「Single-line string pattern」に一致。

  テーブルのカタログ ID。
- `DatabaseName` – UTF-8 文字列。1～255 バイト長。Single-line string pattern に一致。

  テーブルが存在するカタログのデータベースの名前。
- `TableName` – UTF-8 文字列。1～255 バイト長。Single-line string pattern に一致。

  テーブルの名前。
- `TableOptimizer` – TableOptimizer オブジェクト。

  指定されたテーブルに関連付けられたオプティマイザー。

エラー

- EntityNotFoundException
- InvalidInputException
- AccessDeniedException
- InternalServiceException
- ThrottlingException

BatchGetTableOptimizer アクション (Python: batch_get_table_optimizer)

指定されたテーブルオプティマイザーの設定を返します。

リクエスト

- Entries – 必須: BatchGetTableOptimizerEntry オブジェクトの配列。

  取得するテーブルオプティマイザーを指定する BatchGetTableOptimizerEntry オブジェクトのリスト。

応答

- TableOptimizers – BatchTableOptimizer オブジェクトの配列。

  BatchTableOptimizer オブジェクトのリスト。
- Failures – BatchGetTableOptimizerError オブジェクトの配列。

  オペレーションで発生したエラーのリスト。

エラー

- EntityNotFoundException
- InvalidInputException
- AccessDeniedException
- InternalServiceException
- ThrottlingException

ListTableOptimizerRuns アクション (Python: list_table_optimizer_runs)

特定のテーブルについての以前のオプティマイザーの実行の履歴をリストします。

リクエスト

- `CatalogId` – 必須: カタログ ID 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  テーブルのカタログ ID。
- `DatabaseName` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  テーブルが存在するカタログのデータベースの名前。
- `TableName` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  テーブルの名前。
- `Type` – 必須: UTF-8 文字列 (有効な値: `compaction="COMPACTION"` | `retention="RETENTION"` | `orphan_file_deletion="ORPHAN_FILE_DELETION"`)。

  テーブルオプティマイザーのタイプ。
- `MaxResults` – 数値 (整数)。

  各呼び出しで返すオプティマイザーの実行の最大数。
- `NextToken` – UTF-8 文字列。

  継続トークン (これが継続呼び出しの場合)。

応答

- `CatalogId` – カタログ ID 文字列、1 ~ 255 バイト長、「Single-line string pattern」に一致。

  テーブルのカタログ ID。
- `DatabaseName` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  テーブルが存在するカタログのデータベースの名前。
- `TableName` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  テーブルの名前。
- `NextToken` – UTF–8 文字列。

返されたオプティマイザーの実行のリストをページ分割するための継続トークン。リストの現在のセグメントが最後のセグメントではない場合に返されます。

- `TableOptimizerRuns` – TableOptimizerRun オブジェクトの配列。

  テーブルに関連付けられているオプティマイザーの実行のリスト。

エラー

- `EntityNotFoundException`
- `AccessDeniedException`
- `InvalidInputException`
- `ValidationException`
- `InternalServiceException`
- `ThrottlingException`

CreateTableOptimizer アクション (Python: create_table_optimizer)

特定の関数の新しいテーブル最適化を作成します。

リクエスト

- `CatalogId` – 必須: カタログ ID 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  テーブルのカタログ ID。
- `DatabaseName` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  テーブルが存在するカタログのデータベースの名前。
- `TableName` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  テーブルの名前。
- `Type` – 必須: UTF-8 文字列 (有効な値: `compaction="COMPACTION"` | `retention="RETENTION"` | `orphan_file_deletion="ORPHAN_FILE_DELETION"`)。

  テーブルオプティマイザーのタイプ。
- `TableOptimizerConfiguration` – 必須: TableOptimizerConfiguration オブジェクト。

テーブルオプティマイザーの設定を表す TableOptimizerConfiguration オブジェクト。

レスポンス

- 応答パラメータはありません。

エラー

- EntityNotFoundException
- ValidationException
- InvalidInputException
- AccessDeniedException
- AlreadyExistsException
- InternalServiceException
- ThrottlingException

DeleteTableOptimizer アクション (Python: delete_table_optimizer)

オプティマイザーと、テーブルに関連付けられているすべてのメタデータを削除します。最適化はテーブルに対して実行されなくなります。

リクエスト

- CatalogId – 必須: カタログ ID 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  テーブルのカタログ ID。
- DatabaseName – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  テーブルが存在するカタログのデータベースの名前。
- TableName – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  テーブルの名前。
- Type – 必須: UTF-8 文字列 (有効な値: compaction="COMPACTION" | retention="RETENTION" | orphan_file_deletion="ORPHAN_FILE_DELETION")。

  テーブルオプティマイザーのタイプ。

レスポンス

- 応答パラメータはありません。

エラー

- `EntityNotFoundException`
- `InvalidInputException`
- `AccessDeniedException`
- `InternalServiceException`
- `ThrottlingException`

UpdateTableOptimizer アクション (Python: update_table_optimizer)

既存のテーブルオプティマイザーの設定を更新します。

リクエスト

- `CatalogId` – 必須: カタログ ID 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  テーブルのカタログ ID。
- `DatabaseName` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  テーブルが存在するカタログのデータベースの名前。
- `TableName` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  テーブルの名前。
- `Type` – 必須: UTF-8 文字列 (有効な値: `compaction="COMPACTION"` | `retention="RETENTION"` | `orphan_file_deletion="ORPHAN_FILE_DELETION"`)。

  テーブルオプティマイザーのタイプ。
- `TableOptimizerConfiguration` – 必須: TableOptimizerConfiguration オブジェクト。

  テーブルオプティマイザーの設定を表す `TableOptimizerConfiguration` オブジェクト。

レスポンス

- 応答パラメータはありません。

エラー

- `EntityNotFoundException`
- `InvalidInputException`
- `AccessDeniedException`
- `ValidationException`
- `InternalServiceException`
- `ThrottlingException`
- `ConcurrentModificationException`

クローラーおよび分類子 API

クローラーおよび分類子 API では、AWS Glue のクローラーおよび分類子のデータ型について説明します。これには、クローラーまたは分類子を作成、削除、更新、および一覧表示するための API が含まれます。

トピック

- 分類子 API
- クローラー API
- 列統計 API
- クローラースケジューラ API

分類子 API

分類子 API では、AWS Glue 分類子のデータ型について説明します。これには、分類子を作成、削除、更新、および一覧表示するための API が含まれます。

データ型

- 分類子の構造
- GrokClassifier の構造

- XMLClassifier の構造
- JsonClassifier の構造
- CsvClassifier の構造
- CreateGrokClassifierRequest の構造
- UpdateGrokClassifierRequest の構造
- CreateXMLClassifierRequest の構造
- UpdateXMLClassifierRequest の構造
- CreateJsonClassifierRequest の構造
- UpdateJsonClassifierRequest の構造
- CreateCsvClassifierRequest の構造
- UpdateCsvClassifierRequest の構造

分類子の構造

分類子は、クロールタスクでトリガーされます。分類子は、指定されたファイルが処理可能な形式であるかどうかをチェックします。処理可能な場合、分類子は、そのデータ形式と一致する StructType オブジェクトの形式でスキーマを作成します。

AWS Glue に用意されている標準の分類子を使用することも、独自の分類子を作成して、データソースの最適な分類を行い、これらに使用する適切なスキーマを指定することもできます。分類子として、Classifier オブジェクトの各フィールドに指定されている grok 分類子、XML 分類子、JSON 分類子、または CSV 分類子のいずれかを使用できます。

フィールド

- GrokClassifier – GrokClassifier オブジェクト。

  grok を使用する分類子。
- XMLClassifier – XMLClassifier オブジェクト。

  XML コンテンツの分類子。
- JsonClassifier – JsonClassifier オブジェクト。

  JSON コンテンツの分類子。
- CsvClassifier – CsvClassifier オブジェクト。

カンマ区切り値 (CSV) の分類子。

GrokClassifier の構造

grok パターンを使用する分類子。

フィールド

- Name – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  分類子名。
- Classification – 必須: UTF–8 文字列。

  Twitter、JSON、Omniture ログなど、分類子が一致するデータ形式の識別子。
- CreationTime – タイムスタンプ。

  分類子が登録された時刻。
- LastUpdated – タイムスタンプ。

  分類子が最後に更新された時刻。
- Version – 数値 (long)。

  分類子のバージョン。
- GrokPattern – 必須: UTF-8 文字列。1 ~ 2048 バイト長。A Logstash Grok string pattern に一致。

  分類子によってデータストアに適用される grok パターン。詳細については、「カスタム分類子の書き込み」の組み込みパターンを参照してください。
- CustomPatterns - UTF-8 文字列。16,000 バイト長以下。URI address multi-line string pattern に一致。

  この分類子によって定義されたオプションのカスタム Grok パターン。詳細については、「カスタム分類子の書き込み」のカスタムパターンを参照してください。

XMLClassifier の構造

XML コンテンツの分類子。

フィールド

- Name – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  分類子名。
- Classification – 必須: UTF–8 文字列。

  分類子が一致するデータ形式の識別子。
- CreationTime – タイムスタンプ。

  分類子が登録された時刻。
- LastUpdated – タイムスタンプ。

  分類子が最後に更新された時刻。
- Version – 数値 (long)。

  分類子のバージョン。
- RowTag – UTF–8 文字列。

  解析中の XML ドキュメントの各レコードを含む要素を指定する XML タグ。これは、自己終了要素 (/> で終了) を識別することはできません。属性のみを含む空の行要素は、終了タグで終わる場合は解析できます (例: <row item_a="A" item_b="B"></row> は解析できますが、<row item_a="A" item_b="B" /> は解析できません)。

JsonClassifier の構造

JSON コンテンツの分類子。

フィールド

- Name – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  分類子名。
- CreationTime – タイムスタンプ。

  分類子が登録された時刻。
- LastUpdated – タイムスタンプ。

  分類子が最後に更新された時刻。

- `Version` – 数値 (long)。

  分類子のバージョン。

- `JsonPath` – 必須: UTF–8 文字列。

  分類子が分類するための JSON データを定義する `JsonPath` 文字列。AWS Glue は、「Writing JsonPath Custom Classifiers」に記載されているように、JsonPath のサブセットをサポートしています。

CsvClassifier の構造

カスタム `CSV` コンテンツの分類子。

フィールド

- `Name` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  分類子名。

- `CreationTime` – タイムスタンプ。

  分類子が登録された時刻。

- `LastUpdated` – タイムスタンプ。

  分類子が最後に更新された時刻。

- `Version` – 数値 (long)。

  分類子のバージョン。

- `Delimiter` – UTF-8 文字列。1 バイト長ちょうど。Custom string pattern #26 に一致。

  行内の各列エントリを区切るものを示すカスタム記号。

- `QuoteSymbol` – UTF-8 文字列。1 バイト長ちょうど。Custom string pattern #26 に一致。

  コンテンツを単一の列の値に結合するものを示すカスタム記号。列の区切り文字とは異なる必要があります。

- `ContainsHeader` – UTF-8 文字列 (有効な値: `UNKNOWN` | `PRESENT` | `ABSENT`)。

  CSV ファイルにヘッダーが含まれているかどうかを示します。

- `Header` – UTF-8 文字列の配列。

列名を表す文字列のリスト。

- `DisableValueTrimming` – ブール。

  列の値のタイプを識別する前に値をトリミングしないことを指定します。デフォルト値は `true` です。

- `AllowSingleColumn` – ブール。

  1 つの列のみを含むファイルの処理を有効にします。

- `CustomDatatypeConfigured` – ブール。

  カスタムデータ型を設定できるようにします。

- `CustomDatatypes` – UTF-8 文字列の配列。

  「BINARY」、「BOOLEAN」、「DATE」、「DECIMAL」、「DOUBLE」、「FLOAT」、「INT」、「LONG」、「SHORT」、「STRING」、「TIMESTAMP」などのカスタムデータ型のリスト。

- `Serde` – UTF-8 文字列 (有効な値: `OpenCSVSerDe` | `LazySimpleSerDe` | `None`)。

  分類子で CSV を処理するため、データカタログに適用される SerDe を設定します。有効な値は、`OpenCSVSerDe`、`LazySimpleSerDe`、`None` です。クローラーが検出を行う場合は、`None` 値を指定できます。

CreateGrokClassifierRequest の構造

作成する `CreateClassifier` の `grok` 分類子を指定します。

フィールド

- `Classification` – 必須: UTF–8 文字列。

  Twitter、JSON、Omniture ログ、Amazon CloudWatch Logs など、分類子が一致するデータ形式の識別子。

- `Name` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  新しい分類子の名前。

- `GrokPattern` – 必須: UTF-8 文字列。1 ~ 2048 バイト長。A Logstash Grok string pattern に一致。

この分類子によって使用される grok パターン。

- `CustomPatterns` - UTF-8 文字列。16,000 バイト長以下。URI address multi-line string pattern に一致。

  この分類子によって使用されたオプションのカスタム Grok パターン。

UpdateGrokClassifierRequest の構造

`UpdateClassifier` に渡すときに更新する grok 分類子を指定します。

フィールド

- `Name` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  `GrokClassifier` の名前。
- `Classification` – UTF–8 文字列。

  Twitter、JSON、Omniture ログ、Amazon CloudWatch Logs など、分類子が一致するデータ形式の識別子。
- `GrokPattern` - UTF-8 文字列。1 ~ 2,048 バイト長。A Logstash Grok string pattern に一致。

  この分類子によって使用される grok パターン。
- `CustomPatterns` - UTF-8 文字列。16,000 バイト長以下。URI address multi-line string pattern に一致。

  この分類子によって使用されたオプションのカスタム Grok パターン。

CreateXMLClassifierRequest の構造

作成する `CreateClassifier` の XML 分類子を指定します。

フィールド

- `Classification` – 必須: UTF–8 文字列。

  分類子が一致するデータ形式の識別子。
- `Name` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  分類子名。

- RowTag – UTF–8 文字列。

  解析中の XML ドキュメントの各レコードを含む要素を指定する XML タグ。これは、自己終了要素 (/> で終了) を識別することはできません。属性のみを含む空の行要素は、終了タグで終わる場合は解析できます (例: <row item_a="A" item_b="B"></row> は解析できますが、<row item_a="A" item_b="B" /> は解析できません)。

UpdateXMLClassifierRequest の構造

更新する XML 分類子を指定します。

フィールド

- Name – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  分類子名。

- Classification – UTF–8 文字列。

  分類子が一致するデータ形式の識別子。

- RowTag – UTF–8 文字列。

  解析中の XML ドキュメントの各レコードを含む要素を指定する XML タグ。これは、自己終了要素 (/> で終了) を識別することはできません。属性のみを含む空の行要素は、終了タグで終わる場合は解析できます (例: <row item_a="A" item_b="B"></row> は解析できますが、<row item_a="A" item_b="B" /> は解析できません)。

CreateJsonClassifierRequest の構造

作成する CreateClassifier の JSON 分類子を指定します。

フィールド

- Name – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  分類子名。

- JsonPath – 必須: UTF–8 文字列。

分類子が分類するための JSON データを定義する JsonPath 文字列。AWS Glue は、「Writing JsonPath Custom Classifiers」に記載されているように、JsonPath のサブセットをサポートしています。

UpdateJsonClassifierRequest の構造

更新する JSON 分類子を指定します。

フィールド

- Name – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  分類子名。

- JsonPath – UTF–8 文字列。

  分類子が分類するための JSON データを定義する JsonPath 文字列。AWS Glue は、「Writing JsonPath Custom Classifiers」に記載されているように、JsonPath のサブセットをサポートしています。

CreateCsvClassifierRequest の構造

作成する CreateClassifier の カスタム CSV 分類子を指定します。

フィールド

- Name – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  分類子名。

- Delimiter – UTF-8 文字列。1 バイト長ちょうど。Custom string pattern #26 に一致。

  行内の各列エントリを区切るものを示すカスタム記号。

- QuoteSymbol – UTF-8 文字列。1 バイト長ちょうど。Custom string pattern #26 に一致。

  コンテンツを単一の列の値に結合するものを示すカスタム記号。列の区切り文字とは異なる必要があります。

- ContainsHeader – UTF-8 文字列 (有効な値: UNKNOWN | PRESENT | ABSENT)。

  CSV ファイルにヘッダーが含まれているかどうかを示します。

- Header – UTF-8 文字列の配列。

  列名を表す文字列のリスト。
- DisableValueTrimming – ブール。

  列の値のタイプを識別する前に値をトリミングしないことを指定します。デフォルト値は True です。
- AllowSingleColumn – ブール。

  1 つの列のみを含むファイルの処理を有効にします。
- CustomDatatypeConfigured – ブール。

  カスタムデータ型の設定を有効にします。
- CustomDatatypes – UTF-8 文字列の配列。

  サポートされているカスタムデータ型のリストを作成します。
- Serde – UTF-8 文字列 (有効な値: OpenCSVSerDe | LazySimpleSerDe | None)。

  分類子で CSV を処理するため、データカタログに適用される SerDe を設定します。有効な値は、OpenCSVSerDe、LazySimpleSerDe、None です。クローラーが検出を行う場合は、None 値を指定できます。

UpdateCsvClassifierRequest の構造

更新する カスタム CSV 分類子を指定します。

フィールド

- Name – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  分類子名。
- Delimiter – UTF-8 文字列。1 バイト長ちょうど。Custom string pattern #26 に一致。

  行内の各列エントリを区切るものを示すカスタム記号。
- QuoteSymbol – UTF-8 文字列。1 バイト長ちょうど。Custom string pattern #26 に一致。

  コンテンツを単一の列の値に結合するものを示すカスタム記号。列の区切り文字とは異なる必要があります。
- ContainsHeader – UTF-8 文字列 (有効な値: UNKNOWN | PRESENT | ABSENT)。

CSV ファイルにヘッダーが含まれているかどうかを示します。

- `Header` – UTF-8 文字列の配列。

  列名を表す文字列のリスト。

- `DisableValueTrimming` – ブール。

  列の値のタイプを識別する前に値をトリミングしないことを指定します。デフォルト値は True です。

- `AllowSingleColumn` – ブール。

  1 つの列のみを含むファイルの処理を有効にします。

- `CustomDatatypeConfigured` – ブール。

  カスタムデータ型の設定を指定します。

- `CustomDatatypes` – UTF-8 文字列の配列。

  サポートされているカスタムデータ型のリストを指定します。

- `Serde` – UTF-8 文字列 (有効な値: `OpenCSVSerDe` | `LazySimpleSerDe` | `None`)。

  分類子で CSV を処理するため、データカタログに適用される SerDe を設定します。有効な値は、`OpenCSVSerDe`、`LazySimpleSerDe`、`None` です。クローラーが検出を行う場合は、`None` 値を指定できます。

操作

- CreateClassifier アクション (Python: create_classifier)
- DeleteClassifier アクション (Python: delete_classifier)
- GetClassifier アクション (Python: get_classifier)
- GetClassifiers アクション (Python: get_classifiers)
- UpdateClassifier アクション (Python: update_classifier)

CreateClassifier アクション (Python: create_classifier)

ユーザーのアカウントに分類子を作成します。これは、どのリクエストのフィールドが存在するかに応じて、`GrokClassifier`、`XMLClassifier`、`JsonClassifier`、または `CsvClassifier` である場合があります。

リクエスト

- GrokClassifier – CreateGrokClassifierRequest オブジェクト。

  作成する分類子を指定する GrokClassifier オブジェクト。
- XMLClassifier – CreateXMLClassifierRequest オブジェクト。

  作成する分類子を指定する XMLClassifier オブジェクト。
- JsonClassifier – CreateJsonClassifierRequest オブジェクト。

  作成する分類子を指定する JsonClassifier オブジェクト。
- CsvClassifier – CreateCsvClassifierRequest オブジェクト。

  作成する分類子を指定する CsvClassifier オブジェクト。

レスポンス

- 応答パラメータはありません。

エラー

- AlreadyExistsException
- InvalidInputException
- OperationTimeoutException

DeleteClassifier アクション (Python: delete_classifier)

データカタログから分類子を削除します。

リクエスト

- Name – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  削除する分類子の名前。

レスポンス

- 応答パラメータはありません。

エラー

- EntityNotFoundException
- OperationTimeoutException

GetClassifier アクション (Python: get_classifier)

分類子を名前で取得します。

リクエスト

- Name – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  取得する分類子の名前。

レスポンス

- Classifier – 分類子 オブジェクト。

  リクエストされた分類子。

エラー

- EntityNotFoundException
- OperationTimeoutException

GetClassifiers アクション (Python: get_classifiers)

データカタログの分類子オブジェクトのすべてを一覧表示します。

リクエスト

- MaxResults – 1 ~ 1000 の数値 (整数)。

  返されるリストのサイズ (オプション)。
- NextToken – UTF–8 文字列。

  オプションの継続トークン。

応答

- Classifiers – 分類子 オブジェクトの配列。

  分類子オブジェクトのリクエストされたリスト。

- NextToken – UTF–8 文字列。

  継続トークン。

エラー

- OperationTimeoutException

UpdateClassifier アクション (Python: update_classifier)

既存の分類子を変更します (フィールドが存在するかどうかに応じて GrokClassifier、XMLClassifier、JsonClassifier、または CsvClassifier)。

リクエスト

- GrokClassifier – UpdateGrokClassifierRequest オブジェクト。

  フィールドが更新された GrokClassifier オブジェクト。

- XMLClassifier – UpdateXMLClassifierRequest オブジェクト。

  フィールドが更新された XMLClassifier オブジェクト。

- JsonClassifier – UpdateJsonClassifierRequest オブジェクト。

  フィールドが更新された JsonClassifier オブジェクト。

- CsvClassifier – UpdateCsvClassifierRequest オブジェクト。

  フィールドが更新された CsvClassifier オブジェクト。

レスポンス

- 応答パラメータはありません。

エラー

- `InvalidInputException`
- `VersionMismatchException`
- `EntityNotFoundException`
- `OperationTimeoutException`

クローラー API

クローラー API では、AWS Glue クローラーのデータ型と、クローラーを作成、削除、更新、および一覧表示するための API について説明します。

データ型

- Crawler 構造
- Schedule 構造
- CrawlerTargets 構造
- S3Target 構造
- S3DeltaCatalogTarget 構造
- S3DeltaDirectTarget 構造
- JdbcTarget 構造
- MongoDBTarget 構造
- DynamoDBTarget 構造
- DeltaTarget 構造
- IcebergTarget 構造
- HudiTarget 構造
- CatalogTarget 構造
- CrawlerMetrics 構造
- CrawlerHistory 構造
- CrawlsFilter 構造
- SchemaChangePolicy 構造

- LastCrawlInfo 構造
- RecrawlPolicy 構造
- LineageConfiguration 構造
- LakeFormationConfiguration 構造

Crawler 構造

データソースを検査し、分類子を使用してスキーマを判別しようとするクローラープログラムを指定します。成功すると、クローラーはデータソースに関するメタデータを AWS Glue Data Catalog に記録します。

フィールド

- `Name` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  クローラー名。
- `Role` – UTF–8 文字列。

  Amazon Simple Storage Service (Amazon S3) データなどの顧客リソースへのアクセスに使用される IAM ロールの Amazon リソースネーム (ARN)。
- `Targets` – CrawlerTargets オブジェクト。

  クロールするターゲットのコレクション。
- `DatabaseName` – UTF–8 文字列。

  クローラーの出力が保存されている場所のデータベース名。
- `Description` – 説明文字列。2,048 バイト長以下。URI address multi-line string pattern に一致。

  クローラーの説明。
- `Classifiers` – UTF-8 文字列の配列。

  クローラーに関連付けられたカスタム分類子を指定する UTF-8 文字列のリスト。
- `RecrawlPolicy` – RecrawlPolicy オブジェクト。

  データセット全体を再度クロールするか、前回のクローラー実行以降に追加されたフォルダのみをクロールするかを指定するポリシー。
- `SchemaChangePolicy` – SchemaChangePolicy オブジェクト。

クローラーの更新と削除の動作を指定するポリシー。

- LineageConfiguration – LineageConfiguration オブジェクト。

  クローラーに対してデータ系統を有効にするかどうかを指定する設定。

- State – UTF-8 文字列 (有効な値: READY | RUNNING | STOPPING)。

  クローラーが実行中かどうか、あるいは実行が保留中かどうかを示します。

- TablePrefix - UTF-8 文字列。128 バイト長以下。

  作成されたテーブルの名前に追加されるプレフィックス。

- Schedule – スケジュール オブジェクト。

  スケジュールされたクローラーの場合、クローラーが実行されるスケジュール。

- CrawlElapsedTime – 数値 (long)。

  クローラーが実行されている場合は、最後のクロールが開始されてから経過した合計時間が含まれます。

- CreationTime – タイムスタンプ。

  クローラーが作成された時刻。

- LastUpdated – タイムスタンプ。

  クローラーが最後に更新された時刻。

- LastCrawl – LastCrawlInfo オブジェクト。

  最後のクロールのステータス、およびエラーが発生した場合はエラー情報。

- Version – 数値 (long)。

  クローラーのバージョン。

- Configuration – UTF–8 文字列。

  クローラーの構成情報。このバージョン付きの JSON 文字列では、クローラーの動作特性を指定できます。詳細については、「クローラー設定オプションの設定」を参照してください。

- CrawlerSecurityConfiguration - UTF-8 文字列。128 バイト長以下。

  このクローラーで使用される SecurityConfiguration 構造の名前。

- LakeFormationConfiguration – LakeFormationConfiguration オブジェクト。

クローラーが IAM ロールの認証情報の代わりに AWS Lake Formation の認証情報を使用すべきかどうかを指定します。

Schedule 構造

cron ステートメントを使用してイベントをスケジュールするスケジューリングオブジェクト。

フィールド

- ScheduleExpression – UTF-8 文字列。

  スケジュールを指定するために使用される cron 式 (ジョブとクローラーの時間ベースのスケジュールを参照してください。) たとえば、毎日 12:15 UTC に何かを実行するには、cron(15 12 * * ? *) を指定します。
- State – UTF-8 文字列 (有効な値: SCHEDULED | NOT_SCHEDULED | TRANSITIONING)。

  スケジュールの状態。

CrawlerTargets 構造

クロールするデータストアを指定します。

フィールド

- S3Targets – S3Target オブジェクトの配列。

  Amazon Simple Storage Service (Amazon S3) のターゲットを指定します。
- JdbcTargets – JdbcTarget オブジェクトの配列。

  JDBC ターゲットを指定します。
- MongoDBTargets – MongoDBTarget オブジェクトの配列。

  Amazon DocumentDB または MongoDB のターゲットを指定します。
- DynamoDBTargets – DynamoDBTarget オブジェクトの配列。

  Amazon DynamoDB のターゲットを指定します。
- CatalogTargets – CatalogTarget オブジェクトの配列。

  AWS Glue Data Catalog ターゲットを指定します。

- `DeltaTargets` – DeltaTarget オブジェクトの配列。

  Delta データストアのターゲットを指定します。
- `IcebergTargets` – IcebergTarget オブジェクトの配列。

  Apache Iceberg データストアのターゲットを指定します。
- `HudiTargets` – HudiTarget オブジェクトの配列。

  Apache Hudi データストアのターゲットを指定します。

S3Target 構造

Amazon Simple Storage Service (Amazon S3) のデータストアを指定します。

フィールド

- `Path` – UTF-8 文字列。

  Simple Storage Service (Amazon S3) ターゲットへのパス。
- `Exclusions` – UTF-8 文字列の配列。

  クロールから除外するために使用される glob パターンのリスト。詳細については、「クローラーを使用したカタログテーブル」を参照してください。
- `ConnectionName` – UTF-8 文字列。1 ~ 2048 バイト長。

  ジョブまたはクローラーが Amazon Virtual Private Cloud 環境 (Amazon VPC) 内の Amazon S3 のデータにアクセスすることを可能にする接続の名前。
- `SampleSize` – 数値 (整数)。

  データセット内のサンプルファイルをクロールするときにクロールされる各リーフフォルダ内のファイル数を設定します。設定されていない場合、すべてのファイルがクロールされます。有効な値は、1 から 249 までの整数です。
- `EventQueueArn` – UTF–8 文字列。

  有効な Amazon SQS の ARN。例えば、`arn:aws:sqs:region:account:sqs`。
- `DlqEventQueueArn` – UTF–8 文字列。

  有効な Amazon デッドレター SQS ARN。例えば、`arn:aws:sqs:region:account:deadLetterQueue`。

S3DeltaCatalogTarget 構造

AWS Glue データカタログ内の Delta Lake データソースに書き込むターゲットを指定します。

フィールド

- Name – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

  データターゲットの名前。
- Inputs – 必須: UTF-8 文字列の配列。1 個の文字列。

  データターゲットへの入力であるノード。
- PartitionKeys – UTF-8 文字列の配列。

  一連のキーを使用してネイティブパーティショニングを指定します。
- Table – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  書き込むデータベーステーブルの名前。
- Database – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  書き込むデータベースの名前。
- AdditionalOptions – キーバリューペアのマップ配列。

  各キーは、Custom string pattern #59 に適合する UTF-8 文字列です。

  各値は、Custom string pattern #59 に適合する UTF-8 文字列です。

  コネクタの追加接続オプションを指定します。
- SchemaChangePolicy – CatalogSchemaChangePolicy オブジェクト。

  クローラの更新の動作を指定するポリシー。

S3DeltaDirectTarget 構造

Amazon S3 内の Delta Lake データソースに書き込むターゲットを指定します。

フィールド

- Name – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

データターゲットの名前。

- Inputs – 必須: UTF-8 文字列の配列。1 個の文字列。

  データターゲットへの入力であるノード。

- PartitionKeys – UTF-8 文字列の配列。

  一連のキーを使用してネイティブパーティショニングを指定します。

- Path – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  書き込み先の Delta Lake データソースの Amazon S3 パス。

- Compression – 必須: UTF-8 文字列 (有効な値: uncompressed="UNCOMPRESSED" | snappy="SNAPPY")。

  データの圧縮方法を指定します。データに標準のファイル拡張子が付いている場合、このオプションは一般的に不要です。指定できる値は "gzip" および "bzip" です。

- NumberTargetPartitions – UTF–8 文字列。

  Amazon S3 全体に Delta Lake データセットファイルを配布するターゲットパーティション数を指定します。

- Format – 必須: UTF-8 文字列 (有効な値: json="JSON" | csv="CSV" | avro="AVRO" | orc="ORC" | parquet="PARQUET" | hudi="HUDI" | delta="DELTA" | iceberg="ICEBERG" | hyper="HYPER" | xml="XML")。

  ターゲットのデータ出力形式を指定します。

- AdditionalOptions – キーバリューペアのマップ配列。

  各キーは、Custom string pattern #59 に適合する UTF-8 文字列です。

  各値は、Custom string pattern #59 に適合する UTF-8 文字列です。

  コネクタの追加接続オプションを指定します。

- SchemaChangePolicy – DirectSchemaChangePolicy オブジェクト。

  クローラの更新の動作を指定するポリシー。

JdbcTarget 構造

クロールする JDBC データストアを指定します。

フィールド

- `ConnectionName` – UTF-8 文字列。1 ~ 2,048 バイト長。

  JDBC ターゲットに接続するために使用する接続名。

- `Path` – UTF–8 文字列。

  JDBC ターゲットのパス。

- `Exclusions` – UTF-8 文字列の配列。

  クロールから除外するために使用される glob パターンのリスト。詳細については、「クローラーを使用したカタログテーブル」を参照してください。

- `EnableAdditionalMetadata` – UTF-8 文字列の配列。

  `RAWTYPES` または `COMMENTS` の値を指定して、テーブルのレスポンスでその他のメタデータを有効にできます。`RAWTYPES` はネイティブレベルのデータ型、`COMMENTS` はデータベース内の列またはテーブルに関連するコメントを提供します。

  その他のメタデータが必要ない場合は、フィールドを空白のままにしてください。

MongoDBTarget 構造

クロールする Amazon DocumentDB または MongoDB データストアを指定します。

フィールド

- `ConnectionName` – UTF-8 文字列。1 ~ 2,048 バイト長。

  Amazon DocumentDB または MongoDB ターゲットに接続するために使用する接続名。

- `Path` – UTF–8 文字列。

  Amazon DocumentDB または MongoDB ターゲット (データベース/コレクション) のパス。

- `ScanAll` – ブール。

すべてのレコードをスキャンするか、テーブルから行をサンプリングするかを示します。テーブルが高スループットテーブルではない場合、すべてのレコードのスキャンには時間がかかることがあります。

`true` 値はすべてのレコードをスキャンすることを意味し、`false` 値はレコードをサンプリングすることを意味します。値を指定しないと、`true` 値にデフォルト設定されます。

DynamoDBTarget 構造

クロールする Amazon DynamoDB テーブルを指定します。

フィールド

- `Path` – UTF-8 文字列。

  クロールする DynamoDB テーブルの名前。

- `scanAll` – ブール。

  すべてのレコードをスキャンするか、テーブルから行をサンプリングするかを示します。テーブルが高スループットテーブルではない場合、すべてのレコードのスキャンには時間がかかることがあります。

  `true` 値はすべてのレコードをスキャンすることを意味し、`false` 値はレコードをサンプリングすることを意味します。値を指定しないと、`true` 値にデフォルト設定されます。

- `scanRate` – 数値 (double)。

  AWS Glue クローラーで使用する設定済み読み取りキャパシティーユニットの割合を指定します。読み取りキャパシティーユニットは、DynamoDB で定義されている用語で、テーブルに対して実行できる読み取り回数/秒のレート制限として機能する数値です。

  有効な値は NULL または 0.1 ~ 1.5 の値です。NULL 値は、ユーザーが値を指定しない場合に使用され、設定済み読み取りキャパシティーユニットでは 0.5 (プロビジョニングされたテーブルの場合)、は最大の設定済み読み取りキャパシティーユニットの場合は 0.25 (オンデマンドモードを使用するテーブルの場合) にデフォルト設定されます。

DeltaTarget 構造

1 つ以上の Delta テーブルをクロールする Delta データストアを指定します。

フィールド

- `DeltaTables` – UTF-8 文字列の配列。

  Delta テーブルへの Simple Storage Service (Amazon S3) パスのリスト。

- `ConnectionName` – UTF-8 文字列。1 ~ 2,048 バイト長。

  Delta テーブルターゲットに接続するために使用する接続の名前。

- `WriteManifest` – ブール。

  マニフェストファイルを Delta テーブルパスに書き込むかどうかを指定します。

- `CreateNativeDeltaTable` – ブール。

  クローラーがネイティブテーブルを作成するかどうかを指定します。これにより、Delta トランザクションログの直接クエリをサポートするクエリエンジンとの統合が可能になります。

IcebergTarget 構造

Amazon S3 内の Iceberg テーブルが格納されている Apache Iceberg データソースを指定します。

フィールド

- `Paths` – UTF-8 文字列の配列。

  `s3://bucket/prefix` のように、Iceberg メタデータフォルダを含む 1 つ以上の Amazon S3 パス。

- `ConnectionName` – UTF-8 文字列。1 ~ 2,048 バイト長。

  Iceberg ターゲットに接続するために使用する接続の名前。

- `Exclusions` – UTF-8 文字列の配列。

  クロールから除外するために使用される glob パターンのリスト。詳細については、「クローラーを使用したカタログテーブル」を参照してください。

- `MaximumTraversalDepth` – 数値 (整数)。

  クローラーが Amazon S3 パス内の Iceberg メタデータフォルダを発見するために横断できる Amazon S3 パスの最大深度。クローラーの実行時間を制限するために使用されます。

HudiTarget 構造

Apache Hudi データソースを指定します。

フィールド

- `Paths` – UTF-8 文字列の配列。

  Hudi の Amazon S3 場所文字列の配列。それぞれに Hudi テーブルのメタデータファイルが存在するルートフォルダを示します。Hudi フォルダは、ルートフォルダの子フォルダ内に存在する場合があります。

  クローラーは、Hudi フォルダのパス以下にあるすべてのフォルダをスキャンします。

- `ConnectionName` – UTF-8 文字列。1 ~ 2,048 バイト長。

  Hudi ターゲットに接続するために使用する接続の名前。Hudi ファイルが VPC 認可を必要とするバケットに格納されている場合、ここで接続プロパティを設定できます。

- `Exclusions` – UTF-8 文字列の配列。

  クロールから除外するために使用される glob パターンのリスト。詳細については、「クローラーを使用したカタログテーブル」を参照してください。

- `MaximumTraversalDepth` – 数値 (整数)。

  クローラーが Amazon S3 パス内の Hudi メタデータフォルダを発見するために横断できる Amazon S3 パスの最大深度。クローラーの実行時間を制限するために使用されます。

CatalogTarget 構造

AWS Glue Data Catalog ターゲットを指定します。

フィールド

- `DatabaseName` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  同期するデータベースの名前。

- `Tables` – 必須: UTF-8 文字列の配列。1 個の以上の文字列。

  同期するテーブルのリスト。

- `ConnectionName` – UTF-8 文字列、1 ~ 2,048 バイト長。

NETWORK 接続タイプとペアになっている Catalog 接続タイプを使用するときにクロールのターゲットとなる Amazon S3-backed データカタログテーブルの接続の名前。

- EventQueueArn – UTF–8 文字列。

  有効な Amazon SQS の ARN。例えば、arn:aws:sqs:region:account:sqs。

- DlqEventQueueArn – UTF–8 文字列。

  有効な Amazon デッドレター SQS ARN。例えば、arn:aws:sqs:region:account:deadLetterQueue。

CrawlerMetrics 構造

指定されたクローラーのメトリクス。

フィールド

- CrawlerName – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  クローラー名。

- TimeLeftSeconds – 数値 (double)。None 以下。

  実行中のクロールを完了までの予測時間。

- StillEstimating – ブール。

  クローラーがこの実行を完了するのにどれくらいの時間がかかるかをまだ見積もっている場合は true です。

- LastRuntimeSeconds – 数値 (double)。None 以下。

  クローラーの最新の実行にかかる時間 (秒単位)。

- MedianRuntimeSeconds – 数値 (double)。None 以下。

  このクローラーの実行時間の中央値 (秒単位)。

- TablesCreated – 数値 (整数)、None 以下。

  このクローラーで作成されたテーブルの数。

- TablesUpdated – 数値 (整数)、None 以下。

  このクローラーで更新されたテーブルの数。

- `TablesDeleted` – 数値 (整数)、None 以下。

  このクローラーで削除されたテーブルの数。

CrawlerHistory 構造

クローラーの実行に関する情報が含まれます。

フィールド

- `CrawlId` – UTF-8 文字列。

  各クロールの UUID 識別子。
- `State` – UTF-8 文字列 (有効な値: `RUNNING` | `COMPLETED` | `FAILED` | `STOPPED`)。

  クロールの状態。
- `StartTime` – タイムスタンプ。

  クロールが開始された日時。
- `EndTime` – タイムスタンプ。

  クロールが終了された日時。
- `Summary` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  特定のクロールに関する JSON 形式の実行サマリー。追加、更新、または削除されたカタログテーブルとパーティションが含まれます。
- `ErrorMessage` – 説明文字列。2,048 バイト長以下。URI address multi-line string pattern に一致。

  エラーが発生した場合は、クロールに関連付けられたエラーメッセージ。
- `LogGroup` – UTF-8 文字列。1 ~ 512 バイト長。Log group string pattern に一致。

  クロールに関連付けられたロググループ。
- `LogStream` – UTF-8 文字列。1 ~ 512 バイト長。Log-stream string pattern に一致。

  クロールに関連付けられたログストリーム。
- `MessagePrefix` – UTF-8 文字列、1 ~ 255 バイト長、Single-line string pattern に一致。

  このクロールに関する CloudWatch メッセージのプレフィックス。

- DPUHour – 数値 (double)。None 以下。

  クロールに使用されるデータ処理単位 (DPU) の数 (時間単位)。

CrawlsFilter 構造

指定されたクローラーのクローラー実行をフィルタリングするために使用できるフィールド、コンパレータ、および値のリスト。

フィールド

- FieldName – UTF-8 文字列 (有効な値: CRAWL_ID | STATE | START_TIME | END_TIME | DPU_HOUR)。

  指定されたクローラーに対するクローラーの実行をフィルタリングするために使用されるキー。各フィールド名に有効な値は次のとおりです。
  - CRAWL_ID: クロールの UUID 識別子を表す文字列。
  - STATE: クロールの状態を表す文字列。
  - START_TIME および END_TIME: epoch タイムスタンプ (ミリ秒単位)。
  - DPU_HOUR: クロールに使用されるデータ処理単位 (DPU) の数 (時間単位)。
- FilterOperator - UTF-8 文字列 (有効値: GT | GE | LT | LE | EQ | NE).。

  値を操作する定義済みのコンパレータ。利用できる演算子は次のとおりです。
  - GT: より大きい。
  - GE: 以上。
  - LT: より小さい。
  - LE: 以下。
  - EQ: 等しい。
  - NE: 等しくない。
- FieldValue – UTF–8 文字列。

  クロールフィールドでの比較のために提供される値。

SchemaChangePolicy 構造

クローラーの更新と削除の動作を指定するポリシー。

フィールド

- UpdateBehavior – UTF-8 文字列 (有効な値: LOG | UPDATE_IN_DATABASE)。

  クローラーが変更されたスキーマを検出したときの更新動作。

- DeleteBehavior – UTF-8 文字列 (有効な値: LOG | DELETE_FROM_DATABASE | DEPRECATE_IN_DATABASE)。

  クローラーが削除されたオブジェクトを検出したときの削除動作。

LastCrawlInfo 構造

最新のクロールについてのステータスとエラー情報。

フィールド

- Status – UTF-8 文字列 (有効な値: SUCCEEDED | CANCELLED | FAILED)。

  最後のクロールのステータス。

- ErrorMessage – 説明文字列、2,048 バイト長以下、URI address multi-line string pattern に一致。

  エラーが発生した場合、最後のクロールに関するエラー情報。

- LogGroup – UTF-8 文字列。1 ~ 512 バイト長。Log group string pattern に一致。

  最後のクロールのロググループ。

- LogStream – UTF-8 文字列。1 ~ 512 バイト長。Log-stream string pattern に一致。

  最後のクロールのログストリーム。

- MessagePrefix – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  このクロールについてのメッセージのプレフィックス。

- StartTime – タイムスタンプ。

  クロールが開始された時刻。

RecrawlPolicy 構造

最初のクロールの完了後に Amazon S3 データソースをクロールするときに、データセット全体を再度クロールするか、前回のクローラーの実行以降に追加されたフォルダのみをクロールするかを指定します。詳細については、デベロッパーガイドの「AWS Glue の増分クロール」を参照してください。

フィールド

- `RecrawlBehavior` – UTF-8 文字列 (有効な値: `CRAWL_EVERYTHING` | `CRAWL_NEW_FOLDERS_ONLY` | `CRAWL_EVENT_MODE`)。

  データセット全体を再度クロールするか、前回のクローラーの実行以降に追加されたフォルダのみをクロールするかを指定します。

  `CRAWL_EVERYTHING` という値は、データセット全体を再度クロールすることを指定します。

  `CRAWL_NEW_FOLDERS_ONLY` という値は、前回のクローラー実行後に追加されたフォルダのみをクロールすることを指定します。

  `CRAWL_EVENT_MODE` の値は Simple Storage Service (Amazon S3) イベントによって識別される変更のみをクロールするように指定します。

LineageConfiguration 構造

クローラーのデータ系統設定を指定します。

フィールド

- `CrawlerLineageSettings` – UTF-8 文字列 (有効な値: `ENABLE` | `DISABLE`)。

  クローラーに対してデータ系統を有効にするかどうかを指定します。次の値を指定できます:

  - ENABLE: クローラーのデータ系統を有効にします。
  - DISABLE：クローラーのデータ系統を無効にします。

LakeFormationConfiguration 構造

クローラーの AWS Lake Formation 構成設定を指定します。

フィールド

- `UseLakeFormationCredentials` – ブール。

  IAM ロールの認証情報の代わりに AWS Lake Formation の認証情報を使用するかどうかを指定します。
- `AccountId` - UTF-8 文字列。12 バイト長以下。

  クロスアカウントクロールに必要です。ターゲットデータと同じアカウントのクロールでは、null のままにすることができます。

操作

- CreateCrawler アクション (Python: create_crawler)
- DeleteCrawler アクション (Python: delete_crawler)
- GetCrawler アクション (Python: get_crawler)
- GetCrawlers アクション (Python: get_crawlers)
- GetCrawlerMetrics アクション (Python: get_crawler_metrics)
- UpdateCrawler アクション (Python: update_crawler)
- StartCrawler アクション (Python: start_crawler)
- StopCrawler アクション (Python: stop_crawler)
- BatchGetCrawlers Action (Python: batch_get_crawlers)
- ListCrawlers アクション (Python: list_crawlers)
- ListCrawls アクション (Python: list_crawls)

CreateCrawler アクション (Python: create_crawler)

指定されたターゲット、ロール、設定、およびオプションのスケジュールを使用して、新しいクローラーを作成します。`s3Targets`、`jdbcTargets`、または`DynamoDBTargets` フィールドで、少なくとも 1 つ以上のクロールを指定する必要があります。

リクエスト

- `Name` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  新しいクローラーの名前。

- `Role` – 必須: UTF–8 文字列。

  新しいクローラーが顧客リソースにアクセスするために使用する IAM ロール、または IAM ロールの Amazon リソースネーム (ARN)。

- `DatabaseName` – UTF–8 文字列。

  `arn:aws:daylight:us-east-1::database/sometable/*` など、結果が書き込まれる AWS Glue データベース。

- `Description` – 説明文字列。2,048 バイト長以下。URI address multi-line string pattern に一致。

  新しいクローラーの説明。

- `Targets` – 必須: CrawlerTargets オブジェクト。

  クロールするターゲットのコレクションのリスト。

- `Schedule` – UTF–8 文字列。

  スケジュールを指定するために使用される `cron` 式 (ジョブとクローラーの時間ベースのスケジュールを参照してください。) たとえば、毎日 12:15 UTC に何かを実行するには、`cron(15 12 * * ? *)` を指定します。

- `Classifiers` – UTF-8 文字列の配列。

  ユーザーが登録したカスタム分類子のリスト。デフォルトでは、すべての組み込みの分類子がクロールに含まれますが、これらのカスタム分類子によって常に分類別のデフォルトの分類子が上書きされます。

- `TablePrefix` - UTF-8 文字列。128 バイト長以下。

  作成されたカタログテーブルに使用されるテーブルプレフィックス。

- `SchemaChangePolicy` – SchemaChangePolicy オブジェクト。

  クローラーの更新と削除動作のためのポリシー。

- `RecrawlPolicy` – RecrawlPolicy オブジェクト。

  データセット全体を再度クロールするか、前回のクローラー実行以降に追加されたフォルダのみをクロールするかを指定するポリシー。

- `LineageConfiguration` – LineageConfiguration オブジェクト。

  クローラーのデータ系統設定を指定します。

- `LakeFormationConfiguration` – LakeFormationConfiguration オブジェクト。

クローラーの AWS Lake Formation 構成設定を指定します。

- `Configuration` – UTF–8 文字列。

  クローラーの構成情報。このバージョン付きの JSON 文字列では、クローラーの動作特性を指定できます。詳細については、「クローラー設定オプションの設定」を参照してください。

- `CrawlerSecurityConfiguration` - UTF-8 文字列。128 バイト長以下。

  このクローラーで使用される `SecurityConfiguration` 構造の名前。

- `Tags` – キーと値のペアのマップ配列。50 ペア以下。

  各キーは UTF-8 文字列で、1 ~ 128 バイト長です。

  各値は UTF-8 文字列で、256 バイト長以下です。

  このクローラーリクエストで使用するタグ。クローラーへのアクセスを制限するためにタグを使用することができます。AWS Glue のタグの詳細については、デベロッパーガイドの「AWS Tags in AWS Glue」を参照してください。

レスポンス

- 応答パラメータはありません。

エラー

- `InvalidInputException`
- `AlreadyExistsException`
- `OperationTimeoutException`
- `ResourceNumberLimitExceededException`

DeleteCrawler アクション (Python: delete_crawler)

クローラーの状態が `RUNNING` でないかぎり、指定したクローラーを AWS Glue Data Catalog から削除します。

リクエスト

- `Name` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

削除するクローラーの名前。

レスポンス

- 応答パラメータはありません。

エラー

- `EntityNotFoundException`
- `CrawlerRunningException`
- `SchedulerTransitioningException`
- `OperationTimeoutException`

GetCrawler アクション (Python: get_crawler)

指定されたクローラーのメタデータを取得します。

リクエスト

- `Name` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  メタデータを取得するクローラーの名前。

レスポンス

- `Crawler` – Crawler オブジェクト。

  指定されたクローラーのメタデータ。

エラー

- `EntityNotFoundException`
- `OperationTimeoutException`

GetCrawlers アクション (Python: get_crawlers)

顧客アカウントで定義されたすべてのクローラーのメタデータを取得します。

リクエスト

- `MaxResults` – 1 ~ 1000 の数値 (整数)。

  各呼び出しで返されるクローラーの数。
- `NextToken` – UTF–8 文字列。

  継続トークン (これが継続リクエストの場合)。

応答

- `Crawlers` – Crawler オブジェクトの配列。

  クローラーメタデータのリスト。
- `NextToken` – UTF–8 文字列。

  継続トークン (返されるリストがこの顧客アカウントで定義されたリストの最後に達していない場合)。

エラー

- `OperationTimeoutException`

GetCrawlerMetrics アクション (Python: get_crawler_metrics)

指定されたクローラーに関するメトリクスを取得します。

リクエスト

- `CrawlerNameList` - UTF-8 文字列の配列、文字列 100 個以下。

  メトリクスを取得するクローラーの名前のリスト。
- `MaxResults` – 1 ~ 1000 の数値 (整数)。

  返されるリストの最大サイズ。

- NextToken – UTF-8 文字列。

  継続トークン (これが継続呼び出しの場合)。

応答

- CrawlerMetricsList – CrawlerMetrics オブジェクトの配列。

  指定されたクローラーのメトリクスのリスト。
- NextToken – UTF–8 文字列。

  継続トークン (戻されたリストに最後に使用可能なメトリクスが含まれていない場合)。

エラー

- OperationTimeoutException

UpdateCrawler アクション (Python: update_crawler)

クローラーを更新します。クローラーが実行されている場合、クローラーを更新する前に StopCrawler を使用してクローラーを停止する必要があります。

リクエスト

- Name – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  新しいクローラーの名前。
- Role – UTF–8 文字列。

  新しいクローラーが顧客リソースにアクセスするために使用する IAM ロール、または IAM ロールの Amazon リソースネーム (ARN)。
- DatabaseName – UTF–8 文字列。

  arn:aws:daylight:us-east-1::database/sometable/* など、結果が保存される AWS Glue データベース。
- Description - UTF-8 文字列。2,048 バイト長以下。URI address multi-line string pattern に一致。

  新しいクローラーの説明。

- `Targets` – CrawlerTargets オブジェクト。

  クロールするターゲットのリスト。

- `Schedule` – UTF–8 文字列。

  スケジュールを指定するために使用される cron 式 (ジョブとクローラーの時間ベースのスケジュールを参照してください。) たとえば、毎日 12:15 UTC に何かを実行するには、`cron(15 12 * * ? *)` を指定します。

- `Classifiers` – UTF-8 文字列の配列。

  ユーザーが登録したカスタム分類子のリスト。デフォルトでは、すべての組み込みの分類子がクロールに含まれますが、これらのカスタム分類子によって常に分類別のデフォルトの分類子が上書きされます。

- `TablePrefix` - UTF-8 文字列。128 バイト長以下。

  作成されたカタログテーブルに使用されるテーブルプレフィックス。

- `SchemaChangePolicy` – SchemaChangePolicy オブジェクト。

  クローラーの更新と削除動作のためのポリシー。

- `RecrawlPolicy` – RecrawlPolicy オブジェクト。

  データセット全体を再度クロールするか、前回のクローラー実行以降に追加されたフォルダのみをクロールするかを指定するポリシー。

- `LineageConfiguration` – LineageConfiguration オブジェクト。

  クローラーのデータ系統設定を指定します。

- `LakeFormationConfiguration` – LakeFormationConfiguration オブジェクト。

  クローラーの AWS Lake Formation 構成設定を指定します。

- `Configuration` – UTF–8 文字列。

  クローラーの構成情報。このバージョン付きの JSON 文字列では、クローラーの動作特性を指定できます。詳細については、「クローラー設定オプションの設定」を参照してください。

- `CrawlerSecurityConfiguration` - UTF-8 文字列。128 バイト長以下。

  このクローラーで使用される `SecurityConfiguration` 構造の名前。

レスポンス

- 応答パラメータはありません。

エラー

- `InvalidInputException`
- `VersionMismatchException`
- `EntityNotFoundException`
- `CrawlerRunningException`
- `OperationTimeoutException`

StartCrawler アクション (Python: start_crawler)

スケジュールされているものに関係なく、指定されたクローラーを使用してクロールを開始します。クローラーがすでに実行中である場合は、CrawlerRunningException が返されます。

リクエスト

- Name – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  開始するクローラーの名前。

レスポンス

- 応答パラメータはありません。

エラー

- `EntityNotFoundException`
- `CrawlerRunningException`
- `OperationTimeoutException`

StopCrawler アクション (Python: stop_crawler)

指定されたクローラーが実行されている場合は、クロールを停止します。

リクエスト

- Name – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  停止するクローラーの名前。

レスポンス

- 応答パラメータはありません。

エラー

- EntityNotFoundException
- CrawlerNotRunningException
- CrawlerStoppingException
- OperationTimeoutException

BatchGetCrawlers Action (Python: batch_get_crawlers)

指定されたクローラー名のリストのリソースメタデータのリストを返します。ListCrawlers オペレーションを呼び出した後で、このオペレーションを呼び出すことで、アクセス許可が付与されているデータにアクセスできます。このオペレーションは、タグを使用するアクセス許可条件を含め、すべての IAM のアクセス許可をサポートします。

リクエスト

- CrawlerNames – 必須: UTF-8 文字列の配列。文字列 100 個以下。

  クローラー名のリスト。これは ListCrawlers 操作から返された名前であることもあります。

応答

- Crawlers – Crawler オブジェクトの配列。

  クローラー定義のリスト。
- CrawlersNotFound - UTF-8 文字列の配列、文字列 100 個以下。

  クローラーの名前のリストが見つかりません。

エラー

- InvalidInputException
- OperationTimeoutException

ListCrawlers アクション (Python: list_crawlers)

この AWS アカウントのすべてのクローラーリソース、または指定されたタグを持つリソースの名前を取得します。このオペレーションにより、アカウントで利用可能なリソースとその名前を確認できます。

このオペレーションはオプションの Tags フィールドを受け取ります。このフィールドを応答のフィルターとして使用すると、タグ付きリソースをグループとして取得できます。タグフィルタリングの使用を選択した場合は、タグが付いたリソースのみが取得されます。

リクエスト

- MaxResults – 1 ~ 1000 の数値 (整数)。

  返されるリストの最大サイズ。
- NextToken – UTF–8 文字列。

  継続トークン (これが継続リクエストの場合)。
- Tags – キーと値のペアのマップ配列。50 ペア以下。

  各キーは UTF-8 文字列で、1 ~ 128 バイト長です。

  各値は UTF-8 文字列で、256 バイト長以下です。

  これらのタグ付きリソースのみを返すように指定します。

レスポンス

- CrawlerNames - UTF-8 文字列の配列、文字列 100 個以下。

  アカウント内のすべてのクローラーの名前、または指定されたタグを持つクローラーの名前。
- NextToken – UTF–8 文字列。

  継続トークン (戻されたリストに最後に使用可能なメトリクスが含まれていない場合)。

エラー

- `OperationTimeoutException`

ListCrawls アクション (Python: list_crawls)

指定されたクローラーのすべてのクロールを返します。クローラー履歴機能の起動日以降に発生したクロールのみを返し、最大 12 ヶ月分のクロールのみを保持します。古いクロールは返されません。

この API は以下の操作に使用できます。

- 指定されたクローラーのすべてのクロールを取得します。
- 指定されたクローラーのすべてのクロールを、制限されたカウント内に取得します。
- 特定された時間範囲内に、指定されたクローラーのすべてのクロールを取得します。
- 特定の状態、クロール ID、または DPU 時間値を持つ指定されたクローラーのすべてのクロールを取得します。

リクエスト

- `CrawlerName` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  取得する実行のクローラー名。
- `MaxResults` – 1 ~ 1000 の数値 (整数)。

  返される結果の最大数。デフォルトは 20 で、最大は 100 です。
- `Filters` – CrawlsFilter オブジェクトの配列。

  `CrawlsFilter` オブジェクトのリストで指定した条件で、クロールをフィルタリングします。
- `NextToken` – UTF-8 文字列。

  継続トークン (これが継続呼び出しの場合)。

応答

- `Crawls` – CrawlerHistory オブジェクトの配列。

  条件を満たしたクロール実行を表す `CrawlerHistory` オブジェクトのリスト。
- `NextToken` – UTF-8 文字列。

返されたトークンのリストをページ分割するための継続トークン。リストの現在のセグメントが最後のセグメントではない場合に返されます。

エラー

- `EntityNotFoundException`
- `OperationTimeoutException`
- `InvalidInputException`

列統計 API

列統計 API は、テーブル内の列の統計を返す AWS Glue API の説明を記述します。

データ型

- ColumnStatisticsTaskRun の構造
- ColumnStatisticsTaskSettings 構造
- ExecutionAttempt 構造

ColumnStatisticsTaskRun の構造

列統計実行の詳細を示すオブジェクト。

フィールド

- `CustomerId` - UTF-8 文字列。12 バイト長以下。

  AWS アカウント ID。
- `ColumnStatisticsTaskRunId` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  特定の列統計タスク実行の識別子。
- `DatabaseName` – UTF–8 文字列。

  テーブルが存在するデータベース。
- `TableName` – UTF–8 文字列。

列統計が生成されるテーブルの名前。

- ColumnNameList – UTF-8 文字列の配列。

  列名のリスト。何も指定されない場合は、テーブルのすべての列名がデフォルトで使用されます。

- CatalogID – カタログ ID 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  テーブルが存在するデータカタログの ID。提供されない場合は、AWS アカウント ID がデフォルトで使用されます。

- Role – UTF–8 文字列。

  統計を生成するためにサービスが引き受ける IAM ロール。

- SampleSize – 数値 (double)。100 以下。

  統計の生成に使用される行の割合。何も指定されない場合は、テーブル全体が統計の生成に使用されます。

- SecurityConfiguration - UTF-8 文字列。128 バイト長以下。

  列統計タスク実行の CloudWatch ログを暗号化するために使用されるセキュリティ設定の名前。

- NumberOfWorkers – 数値 (整数)。1 以上。

  列統計の生成に使用されるワーカーの数。ジョブは、最大 25 個のインスタンスを自動スケールするように事前設定されています。

- WorkerType – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  統計の生成に使用されるワーカーのタイプ。デフォルトは g.1x です。

- ComputationType – UTF-8 文字列 (有効な値: FULL | INCREMENTAL)。

  列統計の計算のタイプ。

- Status – UTF-8 文字列 (有効な値: STARTING | RUNNING | SUCCEEDED | FAILED | STOPPED)。

  タスク実行のステータス。

- CreationTime – タイムスタンプ。

  このタスクが作成された時刻。

- LastUpdated – タイムスタンプ。

  このタスクが変更された前回の時点。

- `StartTime` – タイムスタンプ。

  タスクの開始時刻。
- `EndTime` – タイムスタンプ。

  タスクの終了時刻。
- `ErrorMessage` – 説明文字列。2,048 バイト長以下。URI address multi-line string pattern に一致。

  ジョブのエラーメッセージ。
- `DPUSeconds` – 数値 (double)。None 以下。

  すべての自動スケーリングされたワーカーについて計算された DPU 使用量 (秒)。

ColumnStatisticsTaskSettings 構造

列統計のタスクの設定。

フィールド

- `DatabaseName` – UTF-8 文字列。

  テーブルが存在するデータベースの名前。
- `TableName` – UTF–8 文字列。

  列統計が生成されるテーブルの名前。
- `Schedule` – スケジュール オブジェクト。

  CRON 構文で指定された列統計を実行するスケジュール。
- `ColumnNameList` – UTF-8 文字列の配列。

  統計を実行する列名のリスト。
- `CatalogID` – カタログ ID 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  データベースが存在するデータカタログの ID。
- `Role` – UTF–8 文字列。

  列統計の実行に使用されるロール。
- `SampleSize` – 数値 (double)。100 以下。

サンプルのデータの割合。

- SecurityConfiguration - UTF-8 文字列。128 バイト長以下。

  CloudWatch ログを暗号化するために使用されるセキュリティ設定の名前。

- ScheduleType – UTF-8 文字列 (有効な値: CRON | AUTO)。

  列統計タスクのスケジュールのタイプ。想定される値は、CRON または AUTO です。

- SettingSource – UTF-8 文字列 (有効な値: CATALOG | TABLE)。

  列統計タスクの設定元。想定される値は、CATALOG または TABLE です。

- LastExecutionAttempt – ExecutionAttempt オブジェクト。

  列統計タスク実行の最後の ExecutionAttempt。

ExecutionAttempt 構造

列統計タスク実行の実行試行。

フィールド

- Status – UTF-8 文字列 (有効な値: FAILED | STARTED)。

  列統計タスクの最後の実行のステータス。

- ColumnStatisticsTaskRunId – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  列統計タスクの最後の実行のタスク実行 ID。

- ExecutionTimestamp – タイムスタンプ。

  列統計タスクの最後の実行が発生したときのタイムスタンプ。

- ErrorMessage – 説明文字列。2,048 バイト長以下。URI address multi-line string pattern に一致。

  列統計タスクの最後の実行に関連付けられたエラーメッセージ。

操作

- StartColumnStatisticsTaskRun アクション (Python: start_column_statistics_task_run)

- GetColumnStatisticsTaskRun アクション (Python: get_column_statistics_task_run)
- GetColumnStatisticsTaskRuns アクション (Python: get_column_statistics_task_runs)
- ListColumnStatisticsTaskRuns アクション (Python: list_column_statistics_task_runs)
- StopColumnStatisticsTaskRun アクション (Python: stop_column_statistics_task_run)
- CreateColumnStatisticsTaskSettings アクション (Python: create_column_statistics_task_settings)
- UpdateColumnStatisticsTaskSettings アクション (Python: update_column_statistics_task_settings)
- GetColumnStatisticsTaskSettings アクション (Python: get_column_statistics_task_settings)
- DeleteColumnStatisticsTaskSettings アクション (Python: delete_column_statistics_task_settings)
- StartColumnStatisticsTaskRunSchedule アクション (Python: start_column_statistics_task_run_schedule)
- StopColumnStatisticsTaskRunSchedule アクション (Python: stop_column_statistics_task_run_schedule)

StartColumnStatisticsTaskRun アクション (Python: start_column_statistics_task_run)

指定されたテーブルと列のために列統計タスク実行を開始します。

リクエスト

- `DatabaseName` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  テーブルが存在するデータベースの名前。
- `TableName` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern 一致。

  統計を生成するテーブルの名前。
- `ColumnNameList` – UTF-8 文字列の配列。

  統計を生成する列名のリスト。何も指定されない場合は、テーブルのすべての列名がデフォルトで使用されます。
- `Role` – 必須: UTF-8 文字列。1 ~ 255 バイト長、Single-line string pattern に一致。

  統計を生成するためにサービスが引き受ける IAM ロール。
- `SampleSize` – 数値 (double)。100 以下。

  統計の生成に使用される行の割合。何も指定されない場合は、テーブル全体が統計の生成に使用されます。

- `CatalogID` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  テーブルが存在するデータカタログの ID。提供されない場合は、AWS アカウント ID がデフォルトで使用されます。

- `SecurityConfiguration` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  列統計タスク実行の CloudWatch ログを暗号化するために使用されるセキュリティ設定の名前。

レスポンス

- `ColumnStatisticsTaskRunId` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  列統計タスク実行の識別子。

エラー

- `AccessDeniedException`
- `EntityNotFoundException`
- `ColumnStatisticsTaskRunningException`
- `OperationTimeoutException`
- `ResourceNumberLimitExceededException`
- `InvalidInputException`

GetColumnStatisticsTaskRun アクション (Python: get_column_statistics_task_run)

タスク実行 ID を指定して、タスク実行に関連付けられたメタデータ/情報を取得します。

リクエスト

- `ColumnStatisticsTaskRunId` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  特定の列統計タスク実行の識別子。

レスポンス

- ColumnStatisticsTaskRun – ColumnStatisticsTaskRun オブジェクト。

  列統計実行の詳細を表す ColumnStatisticsTaskRun オブジェクト。

エラー

- EntityNotFoundException
- OperationTimeoutException
- InvalidInputException

GetColumnStatisticsTaskRuns アクション (Python: get_column_statistics_task_runs)

指定されたテーブルに関連付けられたすべての実行に関する情報を取得します。

リクエスト

- DatabaseName – 必須: UTF–8 文字列。

  テーブルが存在するデータベースの名前。
- TableName – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  テーブルの名前。
- MaxResults – 1 ~ 1000 の数値 (整数)。

  応答の最大サイズ。
- NextToken – UTF-8 文字列。

  継続トークン (これが継続呼び出しの場合)。

応答

- ColumnStatisticsTaskRuns – ColumnStatisticsTaskRun オブジェクトの配列。

  列統計タスク実行のリスト。
- NextToken – UTF–8 文字列。

  継続トークン (すべてのタスク実行がまだ返されていない場合)。

エラー

- OperationTimeoutException

ListColumnStatisticsTaskRuns アクション (Python: list_column_statistics_task_runs)

特定のアカウントについてのすべてのタスク実行をリストします。

リクエスト

- MaxResults – 1 ~ 1000 の数値 (整数)。

  応答の最大サイズ。
- NextToken – UTF-8 文字列。

  継続トークン (これが継続呼び出しの場合)。

応答

- ColumnStatisticsTaskRunIds - UTF-8 文字列の配列、文字列 100 個以下。

  列統計タスク実行 ID のリスト。
- NextToken – UTF–8 文字列。

  継続トークン (すべてのタスク実行 ID がまだ返されていない場合)。

エラー

- OperationTimeoutException

StopColumnStatisticsTaskRun アクション (Python: stop_column_statistics_task_run)

指定されたテーブルについてのタスク実行を停止します。

リクエスト

- DatabaseName – 必須: UTF–8 文字列。

  テーブルが存在するデータベースの名前。

- `TableName` – 必須: UTF-8 文字列。1～255 バイト長。Single-line string pattern に一致。

  テーブルの名前。

レスポンス

- 応答パラメータはありません。

エラー

- `EntityNotFoundException`
- `ColumnStatisticsTaskNotRunningException`
- `ColumnStatisticsTaskStoppingException`
- `OperationTimeoutException`

CreateColumnStatisticsTaskSettings アクション (Python: create_column_statistics_task_settings)

列統計タスクの設定を作成します。

リクエスト

- `DatabaseName` – 必須: UTF-8 文字列。1～255 バイト長。Single-line string pattern に一致。

  テーブルが存在するデータベースの名前。
- `TableName` – 必須: UTF-8 文字列。1～255 バイト長。Single-line string pattern に一致。

  列統計が生成されるテーブルの名前。
- `Role` – 必須: UTF-8 文字列。1～255 バイト長。Single-line string pattern に一致。

  列統計の実行に使用されるロール。
- `Schedule` – UTF–8 文字列。

  CRON 構文で指定された列統計を実行するスケジュール。
- `ColumnNameList` – UTF-8 文字列の配列。

  統計を実行する列名のリスト。

- SampleSize – 数値 (double)。100 以下。

  サンプルのデータの割合。

- CatalogID – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  データベースが存在するデータカタログの ID。

- SecurityConfiguration – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  CloudWatch ログを暗号化するために使用されるセキュリティ設定の名前。

- Tags – キーと値のペアのマップ配列。50 ペア以下。

  各キーは UTF-8 文字列で、1 ~ 128 バイト長です。

  各値は UTF-8 文字列で、256 バイト長以下です。

  タグのマップ。

レスポンス

- 応答パラメータはありません。

エラー

- AlreadyExistsException
- AccessDeniedException
- EntityNotFoundException
- InvalidInputException
- OperationTimeoutException
- ResourceNumberLimitExceededException
- ColumnStatisticsTaskRunningException

UpdateColumnStatisticsTaskSettings アクション (Python: update_column_statistics_task_settings)

列統計タスクの設定を更新します。

リクエスト

- DatabaseName – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  テーブルが存在するデータベースの名前。

- TableName – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  列統計が生成されるテーブルの名前。

- Role – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  列統計の実行に使用されるロール。

- Schedule – UTF–8 文字列。

  CRON 構文で指定された列統計を実行するスケジュール。

- ColumnNameList – UTF-8 文字列の配列。

  統計を実行する列名のリスト。

- SampleSize – 数値 (double)。100 以下。

  サンプルのデータの割合。

- CatalogID – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  データベースが存在するデータカタログの ID。

- SecurityConfiguration – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  CloudWatch ログを暗号化するために使用されるセキュリティ設定の名前。

レスポンス

- 応答パラメータはありません。

エラー

- AccessDeniedException
- EntityNotFoundException
- InvalidInputException

- VersionMismatchException
- OperationTimeoutException

GetColumnStatisticsTaskSettings アクション (Python: get_column_statistics_task_settings)

列統計タスクの設定を取得します。

リクエスト

- DatabaseName – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  テーブルが存在するデータベースの名前。
- TableName – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  列統計を取得するテーブルの名前。

レスポンス

- ColumnStatisticsTaskSettings – ColumnStatisticsTaskSettings オブジェクト。

  列統計タスクの設定を表す ColumnStatisticsTaskSettings オブジェクト。

エラー

- EntityNotFoundException
- InvalidInputException
- OperationTimeoutException

DeleteColumnStatisticsTaskSettings アクション (Python: delete_column_statistics_task_settings)

列統計タスクの設定を削除します。

リクエスト

- DatabaseName – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

テーブルが存在するデータベースの名前。

- TableName – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  列統計を削除するテーブルの名前。

レスポンス

- 応答パラメータはありません。

エラー

- EntityNotFoundException
- InvalidInputException
- OperationTimeoutException

StartColumnStatisticsTaskRunSchedule アクション (Python: start_column_statistics_task_run_schedule)

列統計タスク実行スケジュールを開始します。

リクエスト

- DatabaseName – 必須: UTF-8 文字列、1 ~ 255 バイト長、Single-line string pattern に一致。

  テーブルが存在するデータベースの名前。

- TableName – 必須: UTF-8 文字列、1 ~ 255 バイト長、Single-line string pattern に一致。

  列統計タスク実行スケジュールを開始するテーブルの名前。

レスポンス

- 応答パラメータはありません。

エラー

- AccessDeniedException

- EntityNotFoundException
- InvalidInputException
- OperationTimeoutException

StopColumnStatisticsTaskRunSchedule アクション (Python: stop_column_statistics_task_run_schedule)

列統計タスク実行スケジュールを停止します。

リクエスト

- DatabaseName – 必須: UTF-8 文字列、1 ~ 255 バイト長、Single-line string pattern に一致。

  テーブルが存在するデータベースの名前。
- TableName – 必須: UTF-8 文字列、1 ~ 255 バイト長、Single-line string pattern に一致。

  列統計タスク実行スケジュールを停止するテーブルの名前。

レスポンス

- 応答パラメータはありません。

エラー

- EntityNotFoundException
- InvalidInputException
- OperationTimeoutException

例外

- ColumnStatisticsTaskRunningException の構造
- ColumnStatisticsTaskNotRunningException の構造
- ColumnStatisticsTaskStoppingException の構造
- ColumnStatisticsTaskAutoConcurrencyLimitException 構造
- InvalidCatalogSettingException 構造

ColumnStatisticsTaskRunningException の構造

列統計生成ジョブの実行中に別のジョブの開始を試行するとスローされる例外。

フィールド

- Message – UTF-8 文字列。

  問題を説明するメッセージ。

ColumnStatisticsTaskNotRunningException の構造

実行中のタスクがないときにタスク実行の停止を試行するとスローされる例外。

フィールド

- Message – UTF-8 文字列。

  問題を説明するメッセージ。

ColumnStatisticsTaskStoppingException の構造

タスク実行の停止を試行するとスローされる例外。

フィールド

- Message – UTF-8 文字列。

  問題を説明するメッセージ。

ColumnStatisticsTaskAutoConcurrencyLimitException 構造

自動統計ジョブの同時実行数の上限に既に達している場合にスローされる例外。

フィールド

- Message – UTF-8 文字列。

  問題を説明するメッセージ。

InvalidCatalogSettingException 構造

カタログ設定に問題がある場合にスローされる例外。

フィールド

- `Message` – UTF-8 文字列。

  問題を説明するメッセージ。

クローラースケジューラ API

クローラースケジューラ API では、AWS Glue クローラーのデータ型と、クローラーを作成、削除、更新、および一覧表示するための API について説明します。

データ型

- Schedule 構造

Schedule 構造

`cron` ステートメントを使用してイベントをスケジュールするスケジューリングオブジェクト。

フィールド

- `ScheduleExpression` – UTF-8 文字列。

  スケジュールを指定するために使用される `cron` 式 (ジョブとクローラーの時間ベースのスケジュールを参照してください。) たとえば、毎日 12:15 UTC に何かを実行するには、`cron(15 12 * * ? *)` を指定します。

- `State` – UTF-8 文字列 (有効な値: `SCHEDULED` | `NOT_SCHEDULED` | `TRANSITIONING`)。

  スケジュールの状態。

操作

- UpdateCrawlerSchedule アクション (Python: update_crawler_schedule)
- StartCrawlerSchedule アクション (Python: start_crawler_schedule)

- StopCrawlerSchedule アクション (Python: stop_crawler_schedule)

UpdateCrawlerSchedule アクション (Python: update_crawler_schedule)

cron 式を使用してクローラーのスケジュールを更新します。

リクエスト

- CrawlerName – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  スケジュールを更新するクローラーの名前。
- Schedule – UTF–8 文字列。

  スケジュールを指定するために使用される更新された cron 式 (ジョブとクローラーの時間ベースのスケジュールを参照)。たとえば、毎日 12:15 UTC に何かを実行するには、cron(15 12 * * ? *) を指定します。

レスポンス

- 応答パラメータはありません。

エラー

- EntityNotFoundException
- InvalidInputException
- VersionMismatchException
- SchedulerTransitioningException
- OperationTimeoutException

StartCrawlerSchedule アクション (Python: start_crawler_schedule)

クローラーがすでに実行中、またはスケジュールの状態がすでに SCHEDULED でなければ、指定されたクローラーのスケジュールの状態を SCHEDULED に変更します。

リクエスト

- CrawlerName – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

スケジュールするクローラーの名前。

レスポンス

- 応答パラメータはありません。

エラー

- `EntityNotFoundException`
- `SchedulerRunningException`
- `SchedulerTransitioningException`
- `NoScheduleException`
- `OperationTimeoutException`

StopCrawlerSchedule アクション (Python: stop_crawler_schedule)

指定されたクローラーのスケジュールの状態を `NOT_SCHEDULED` に設定しますが、クローラーがすでに実行中の場合は停止されません。

リクエスト

- `CrawlerName` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  スケジュールの状態を設定するクローラーの名前。

レスポンス

- 応答パラメータはありません。

エラー

- `EntityNotFoundException`
- `SchedulerNotRunningException`
- `SchedulerTransitioningException`
- `OperationTimeoutException`

ETL スクリプト API の自動生成

ETL スクリプト生成 API では、AWS Glue で ETL スクリプトを生成するためのデータ型と API について説明します。

データ型

- CodeGenNode 構造
- CodeGenNodeArg 構造
- CodeGenEdge 構造
- 場所の構造
- CatalogEntry 構造
- MappingEntry 構造

CodeGenNode 構造

Directed Acyclic Graph (DAG) でノードを表す

フィールド

- `Id` – 必須: UTF-8 文字列、1 ~ 255 バイト長、Identifier string pattern に一致。

  ノードのグラフ内で一意のノード識別子。
- `NodeType` – 必須: UTF-8 文字列。

  このノードのタイプ。
- `Args` – 必須: CodeGenNodeArg オブジェクトの配列。構造 50 個以下。

  ノードのプロパティ、名前と値のペアの形式。
- `LineNumber` – 数値 (整数)。

  ノードの行数。

CodeGenNodeArg 構造

ノードの引数またはプロパティ。

フィールド

- Name – 必須: UTF-8 文字列。

  引数またはプロパティの名前。
- Value – 必須: UTF-8 文字列。

  引数またはプロパティの値。
- Param – ブール。

  値がパラメータとして使用される場合は True。

CodeGenEdge 構造

Directed Acyclic Graph (DAG) で方向のエッジを表します。

フィールド

- Source – 必須: UTF-8 文字列、1～255 バイト長、Identifier string pattern に一致。

  エッジが始まるノードの ID。
- Target – 必須: UTF-8 文字列、1～255 バイト長、Identifier string pattern に一致。

  エッジが終了するノードの ID。
- TargetParameter – UTF-8 文字列。

  エッジのターゲット。

場所の構造

リソースの場所。

フィールド

- Jdbc – CodeGenNodeArg オブジェクトの配列。構造 50 個以下。

  JDBC の場所。
- S3 – CodeGenNodeArg オブジェクトの配列。構造 50 個以下。

  Amazon Simple Storage Service (Amazon S3) の場所。

- DynamoDB – CodeGenNodeArg オブジェクトの配列。構造 50 個以下。

  Amazon DynamoDB テーブルの場所。

CatalogEntry 構造

AWS Glue Data Catalog でテーブル定義を指定します。

フィールド

- DatabaseName – 必須: UTF-8 文字列、1 ~ 255 バイト長、Single-line string pattern に一致。

  テーブルメタデータが存在するデータベース。
- TableName – 必須: UTF-8 文字列、1 ~ 255 バイト長、Single-line string pattern に一致。

  該当するテーブルの名前。

MappingEntry 構造

マッピングを定義します。

フィールド

- SourceTable – UTF-8 文字列。

  ソーステーブルの名前。
- SourcePath – UTF-8 文字列。

  ソースパス。
- SourceType – UTF-8 文字列。

  ソースタイプ。
- TargetTable – UTF-8 文字列。

  ターゲットテーブル。
- TargetPath – UTF-8 文字列。

  ターゲットパス。
- TargetType – UTF-8 文字列。

ターゲットのタイプ。

操作

- CreateScript アクション (Python: create_script)
- GetDataflowGraph アクション (Python: get_dataflow_graph)
- GetMapping アクション (Python: get_mapping)
- GetPlan アクション (Python: get_plan)

CreateScript アクション (Python: create_script)

Directed Acyclic Graph (DAG) をコードに変換します。

リクエスト

- `DagNodes` – CodeGenNode オブジェクトの配列。

  DAG 内のノードのリスト。
- `DagEdges` – CodeGenEdge オブジェクトの配列。

  DAG 内のエッジのリスト。
- `Language` – UTF-8 文字列 (有効な値: `PYTHON` | `SCALA`)。

  DAG から生成されたコードのプログラミング言語。

レスポンス

- `PythonScript` – UTF-8 文字列。

  DAG から生成された Python スクリプト。
- `ScalaCode` – UTF-8 文字列。

  DAG から生成された Scala コード。

エラー

- InvalidInputException
- InternalServiceException
- OperationTimeoutException

GetDataflowGraph アクション (Python: get_dataflow_graph)

Python スクリプトを Directed Acyclic Graph (DAG) に変換します。

リクエスト

- PythonScript – UTF-8 文字列。

  変換する Python スクリプト。

応答

- DagNodes – CodeGenNode オブジェクトの配列。

  結果の DAG 内のノードのリスト。
- DagEdges – CodeGenEdge オブジェクトの配列。

  結果の DAG 内のエッジのリスト。

エラー

- InvalidInputException
- InternalServiceException
- OperationTimeoutException

GetMapping アクション (Python: get_mapping)

マッピングを作成します。

リクエスト

- Source – 必須: CatalogEntry オブジェクト。

ソーステーブルを指定します。

- `Sinks` – CatalogEntry オブジェクトの配列。

  ターゲットテーブルのリスト。

- `Location` – ロケーション オブジェクト。

  マッピングのパラメータ。

レスポンス

- `Mapping` – 必須: MappingEntry オブジェクトの配列。

  指定されたターゲットへのマッピングのリスト。

エラー

- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`
- `EntityNotFoundException`

GetPlan アクション (Python: get_plan)

指定されたマッピングを実行するコードを取得します。

リクエスト

- `Mapping` – 必須: MappingEntry オブジェクトの配列。

  ソーステーブルからターゲットテーブルへのマッピングのリスト。

- `Source` – 必須: CatalogEntry オブジェクト。

  ソーステーブル。

- `Sinks` – CatalogEntry オブジェクトの配列。

  ターゲットテーブル。

- `Location` – ロケーション オブジェクト。

  マッピングのパラメータ。

- `Language` – UTF-8 文字列 (有効な値: PYTHON | SCALA)。

  マッピングを実行するコードのプログラミング言語。

- `AdditionalPlanOptionsMap` – キーバリューペアのマップ配列。

  各キーは UTF-8 文字列。

  各値は UTF-8 文字列。

  追加のオプションのキー値パラメータを保持するマップ。

  現在、次のキーと値のペアがサポートされています。

  - `inferSchema` – AWS Glue ジョブによって生成されたデフォルトスクリプトの `inferSchema` の設定を true にするか false にするかを指定します。例えば、`inferSchema` を true に設定するには、次のキーと値のペアを渡します。

    ```
    --additional-plan-options-map '{"inferSchema":"true"}'
    ```

レスポンス

- `PythonScript` – UTF-8 文字列。

  マッピングを実行する Python スクリプト。

- `ScalaCode` – UTF-8 文字列。

  マッピングを実行する Scala コード。

エラー

- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`

ビジュアルジョブ API

ビジュアルジョブ API を使用すると、AWS Glue ジョブのビジュアル設定に対応した JSON オブジェクトから AWS Glue APIを使用して、データ統合ジョブを作成できます。

`CodeGenConfigurationNodes` のリストは、作成したジョブ用 AWS Glue Studio内のDAG を登録するジョブ API の作成または更新用に提供され、関連するコードを生成します。

データ型

- CodeGenConfiguration 構造
- JDBCConnectorOptions 構造
- StreamingDataProviewOptions 構造
- AthenaConnectorSource 構造
- JDBCConnectorSource 構造
- SparkConnectorSource 構造
- CatalogSource 構造
- MySQLCatalogSource 構造
- PostgreSQLCatalogSource 構造
- OracleSQLCatalogSource 構造
- MicrosoftSQLServerCatalogSource 構造
- CatalogKinesisSource 構造
- DirectKinesisSource 構造
- KinesisStreamingSourceOptions 構造
- CatalogKafkaSource 構造
- DirectKafkaSource 構造
- KafkaStreamingSourceOptions 構造
- RedShiftSource 構造
- AmazonRedshiftSource 構造
- AmazonRedshiftNodeData 構造
- AmazonRedshiftAdvancedOption 構造

- Option 構造
- S3CatalogSource 構造
- S3SourceAdditionalOptions 構造
- S3CsvSource 構造
- DirectJDBCSource 構造
- S3DirectSourceAdditionalOptions 構造
- S3JsonSource 構造
- S3ParquetSource 構造
- S3DeltaSource 構造
- S3CatalogDeltaSource 構造
- CatalogDeltaSource 構造
- S3HudiSource 構造
- S3CatalogHudiSource 構造
- S3ExcelSource の構造
- CatalogHudiSource 構造
- DynamoDBCatalogSource 構造
- RelationalCatalogSource 構造
- JDBCConnectorTarget 構造
- SparkConnectorTerg 構造
- BasicCatalogTerg 構造
- MySQLCatalogTarget 構造
- PostgreSQLCatalogTarget 構造
- OracleSQLCatalogTarget 構造
- MicrosoftSQLServerCatalogTarget 構造
- RedShiftTarget 構造
- AmazonRedshiftTarget 構造
- UpsertRedshiftTargetOptions 構造
- S3CatalogTarget 構造

- S3GlueParquetTarget 構造
- CatalogSchemaChangePolicy 構造
- S3directTarget 構造
- S3HudiCatalogTarget 構造
- S3HudiDirectTarget 構造
- S3DeltaCatalogTarget 構造
- S3DeltaDirectTarget 構造
- S3HyperDirectTarget 構造
- S3IcebergDirectTarget 構造
- DirectSchemaChangePolicy 構造
- ApplyMapping 構造
- Mapping 構造
- SelectFields 構造
- DropFields 構造
- RenameField 構造
- スピゴット構造
- Join 構造
- JoinColumn 構造
- SplitFields 構造
- SelectFromCollection 構造
- FillMissingValues 構造
- Filter 構造
- FilterExpression 構造
- FilterValue 構造
- CustomCode 構造
- SparkSQL 構造
- SqlAlias 構造
- DropNullFields 構造

- NullCheckboxList 構造
- NullValueField 構造
- Datatype 構造
- Merge 構造
- Union 構造
- PIIDetection 構造
- Aggregate 構造
- DropDuplicates 構造
- GovernedCatalogTarget 構造
- GovernedCatalogSource 構造
- AggregateOperation 構造
- GlueSchema 構造
- GlueStudioSchemaColumn 構造
- GlueStudioColumn 構造
- DynamicTransform 構造
- TransformConfigParameter 構造
- EvaluateDataQuality 構造
- DQResultsPublishingOptions 構造
- DQStopJobOnFailureOptions 構造
- EvaluateDataQualityMultiFrame 構造
- Recipe 構造
- RecipeReference 構造
- SnowflakeNodeData 構造
- SnowflakeSource 構造
- SnowflakeTarget 構造
- ConnectorDataSource の構造
- ConnectorDataTarget の構造
- RecipeStep 構造
- RecipeAction 構造

- ConditionExpression 構造

CodeGenConfiguration 構造

CodeGenConfigurationNode は、すべての有効なノードタイプを列挙します。そのメンバー変数は 1 つしか入力できません。

フィールド

- AthenaConnectorSource – AthenaConnectorSource オブジェクト。

  Amazon Athena データソースへのコネクタを指定します。
- JDBCConnectorSource – jdbcConnectorSource オブジェクト。

  JDBC データソースへのコネクタを指定します。
- SparkConnectorSource – SparkConnectorSource オブジェクト。

  Apache Spark データソースへのコネクタを指定します。
- CatalogSource – CatalogSource オブジェクト。

  AWS Glueデータカタログにデータストアを指定します。
- RedshiftSource – RedShiftSource オブジェクト。

  Amazon Redshift データストアを指定します。
- S3CatalogSource – S3CatalogSource オブジェクト。

  AWS Glueデータカタログに Amazon S3 データストアを指定します。
- S3CsvSource – S3CsvSource オブジェクト。

  Amazon S3 に格納されているコマンド区切り値 (CSV) データストアを指定します。
- S3ExcelSource – S3ExcelSource オブジェクト。

  Amazon S3 の Excel ファイルを読み取るための設定パラメータを定義します。
- S3JsonSource – S3jsonSource オブジェクト。

  Amazon S3 の JSON データストアを指定します。
- S3ParquetSource – S3parquetSource オブジェクト。

Amazon S3 に保存されている Apache Parquet データストアを指定します。

- `RelationalCatalogSource` – RelationalCatalogSource オブジェクト。

  AWS Glue データカタログ内のリレーショナルカタログデータストアを指定します。

- `DynamoDBCatalogSource` – DynamoDBCatalogSource オブジェクト。

  AWS Glue データカタログ内の DynamoDBC カタログデータストアを指定します。

- `JDBCConnectorTarget` – jdbcConnectorTarget オブジェクト。

  Apache Parquet 列指向ストレージで Amazon S3 に書き込むデータターゲットを指定します。

- `SparkConnectorTarget` – SparkConnectorTarget オブジェクト。

  Apache Spark コネクタを使用するターゲットを指定します。

- `CatalogTarget` – BasicCatalogTarget オブジェクト。

  AWS Glue データカタログテーブルを使用するターゲットを指定します。

- `RedshiftTarget` – RedshiftTarget オブジェクト。

  Amazon Redshift を使用するターゲットを指定します。

- `S3CatalogTarget` – S3CatalogTarget オブジェクト。

  AWS Glue データカタログを使用して Amazon S3 に書き込むデータターゲットを指定します。

- `S3GlueParquetTarget` – S3GlueParquetTarget オブジェクト。

  Apache Parquet 列指向ストレージで Amazon S3 に書き込むデータターゲットを指定します。

- `S3HyperDirectTarget` – S3HyperDirectTarget オブジェクト。

  HyperDirect の最適化機能を使用して Amazon S3 にデータを書き込むための設定パラメータを定義します。

- `S3DirectTarget` – S3directTarget オブジェクト。

  Amazon S3 に書き込むデータターゲットを指定します。

- `S3IcebergDirectTarget` – S3IcebergDirectTarget オブジェクト。

  Apache Iceberg テーブルとして Amazon S3 にデータを書き込むための設定パラメータを定義します。

- `ApplyMapping` – ApplyMapping オブジェクト。

データソースのマップデータプロパティキーを、データターゲットのデータプロパティキーに変換指定します。キーの名前を変更したり、データタイプを変更したり、データセットから削除するキーを選択できます。

- SelectFields – SelectFields オブジェクト。

  保持するデータプロパティキーの選択変換を指定します。

- DropFields – DropFields オブジェクト。

  削除するデータプロパティキーを選択する変換を指定します。

- RenameField – RenameField オブジェクト。

  1 つのデータプロパティキーの名前を変更する変換を指定します。

- Spigot – スピゴット オブジェクト。

  Amazon S3 バケットにデータのサンプルを書き込むための変換を指定します。

- Join – Join オブジェクト。

  指定したデータプロパティキーの比較フレーズを使用して、2 つのデータセットを 1 つに結合する変換を指定します。結合タイプは、内部結合、外部結合、左結合、右結合、左半結合、左反結合を使用できます。

- SplitFields – SplitFields オブジェクト。

  データプロパティキーを 2 つの DynamicFrames に分割する変換を指定します。出力は DynamicFrames のコレクションです。一方は選択したデータプロパティキー、他方は残っている方のデータプロパティキーを持ちます。

- SelectFromCollection – SelectFromCollection オブジェクト。

  DynamicFrame のコレクションから 1 つの DynamicFrames を選択するトランスフォームを指定します。出力は選択された DynamicFrame です。

- FillMissingValues – FillMissingValues オブジェクト。

  変換を使用して、データセット内に欠落値があるレコードを検索し、補完により決定する値を持つ新しいフィールドを追加します。入力データセットは、欠落値を決定する機械学習モデルのトレーニングに使用されます。

- Filter – フィルター オブジェクト。

  フィルター条件に基づいて、データセットを 2 つに分割する変換を指定します。

- `CustomCode` – CustomCode オブジェクト。

  データ変換を実行するためにカスタムコードを使用する変換を指定します。結果には、DynamicFrames のコレクションが出力されます。

- `SparkSQL` – SparkSQL オブジェクト。

  データを変換するためにSpark SQL 構文を使用して、SQL クエリを入力する変換を指定します。出力は、単一の `DynamicFrame` です。

- `DirectKinesisSource` – DirectKinesisSource オブジェクト。

  直接 Amazon Kinesis データソースを指定します。

- `DirectKafkaSource` – DirectKafkaSource オブジェクト。

  Apache Kafka データストアを指定します。

- `CatalogKinesisSource` – CatalogKinesisSource オブジェクト。

  AWS Glueデータカタログで Kinesis データソースを指定します。

- `CatalogKafkaSource` – CatalogKafkaSource オブジェクト。

  データカタログで Apache Kafka データストアを指定します。

- `DropNullFields` – DropNullFields オブジェクト。

  列のすべての値が「null」である場合、データセットから列を削除する変換を指定します。デフォルトでは、AWS Glue Studio は null オブジェクトを認識しますが、空の文字列、『null』が 1 の整数、または 0 などのプレースホルダなどの文字列では、自動的に『null』として認識されません。

- `Merge` – 結合 オブジェクト。

  レコードを識別するために、プライマリキーに基づいて `DynamicFrame` とステージング `DynamicFrame` を結合する変換を指定します。重複レコード (同じプライマリキーを持つレコード) は重複除外されません。

- `Union` – Union オブジェクト。

  2 つ以上のデータセットの行を 1 つの結果に結合する変換を指定します。

- `PIIDetection` – PIIDetection オブジェクト。

  PII データを識別、削除、またはマスクする変換を指定します。

- `Aggregate` – 集計 オブジェクト。

選択したフィールドによって行をグループ化し、指定された関数を使用して集計値を計算する変換を指定します。

- `DropDuplicates` – DropDuplicates オブジェクト。

  繰り返しデータの行をデータセットから削除する変換を指定します。

- `GovernedCatalogTarget` – GovernedCatalogTarget オブジェクト。

  管理されたカタログに書き込むデータターゲットを指定します。

- `GovernedCatalogSource` – GovernedCatalogSource オブジェクト。

  管理されたデータカタログ内のデータソースを指定します。

- `MicrosoftSQLServerCatalogSource` – MicrosoftSQLServerCatalogSource オブジェクト。

  AWS Glueデータカタログ内の Microsoft SQL Server データソースを指定します。

- `MySQLCatalogSource` – MySQLCatalogSource オブジェクト。

  AWS Glueデータカタログ内の MySQL データソースを指定します。

- `OracleSQLCatalogSource` – OracleSQLCatalogSource オブジェクト。

  AWS Glueデータカタログ内の Oracle データソースを指定します。

- `PostgreSQLCatalogSource` – PostgreSQLCatalogSource オブジェクト。

  AWS Glueデータカタログ内の PostgresSQL データソースを指定します。

- `MicrosoftSQLServerCatalogTarget` – MicrosoftSQLServerCatalogTarget オブジェクト。

  Microsoft SQL を使用するターゲットを指定します。

- `MySQLCatalogTarget` – MySQLCatalogTarget オブジェクト。

  MySQL を使用するターゲットを指定します。

- `OracleSQLCatalogTarget` – OracleSQLCatalogTarget オブジェクト。

  Oracle SQL を使用するターゲットを指定します。

- `PostgreSQLCatalogTarget` – PostgreSQLCatalogTarget オブジェクト。

  Postgres SQL を使用するターゲットを指定します。

- `DynamicTransform` – DynamicTransform オブジェクト。

ユーザーが作成したカスタムビジュアル変換を指定します。

- `EvaluateDataQuality` – EvaluateDataQuality オブジェクト。

  データ品質評価基準を指定します。

- `S3CatalogHudiSource` – S3CatalogHudiSource オブジェクト。

  AWS Glue データカタログ内に登録されている Hudi データソースを指定します。データソースは Amazon S3 に保存する必要があります。

- `CatalogHudiSource` – CatalogHudiSource オブジェクト。

  AWS Glue データカタログ内に登録されている Hudi データソースを指定します。

- `S3HudiSource` – S3HudiSource オブジェクト。

  Amazon S3 内に保存されている Hudi データソースを指定します。

- `S3HudiCatalogTarget` – S3HudiCatalogTarget オブジェクト。

  AWS Glue データカタログ内の Hudi データソースに書き込むターゲットを指定します。

- `S3HudiDirectTarget` – S3HudiDirectTarget オブジェクト。

  Amazon S3 内の Hudi データソースに書き込むターゲットを指定します。

- `S3CatalogDeltaSource` – S3CatalogDeltaSource オブジェクト。

  AWS Glue データカタログ内に登録されている Delta Lake データソースを指定します。データソースは Amazon S3 に保存する必要があります。

- `CatalogDeltaSource` – CatalogDeltaSource オブジェクト。

  AWS Glue データカタログ内に登録されている Delta Lake データソースを指定します。

- `S3DeltaSource` – S3DeltaSource オブジェクト。

  Amazon S3 内に保存されている Delta Lake データソースを指定します。

- `S3DeltaCatalogTarget` – S3DeltaCatalogTarget オブジェクト。

  AWS Glue データカタログ内の Delta Lake データソースに書き込むターゲットを指定します。

- `S3DeltaDirectTarget` – S3DeltaDirectTarget オブジェクト。

  Amazon S3 内の Delta Lake データソースに書き込むターゲットを指定します。

- `AmazonRedshiftSource` – AmazonRedshiftSource オブジェクト。

Amazon Redshift 内のデータソースに書き込むターゲットを指定します。

- `AmazonRedshiftTarget` – AmazonRedshiftTarget オブジェクト。

  Amazon Redshift 内のデータターゲットに書き込むターゲットを指定します。

- `EvaluateDataQualityMultiFrame` – EvaluateDataQualityMultiFrame オブジェクト。

  データ品質評価基準を指定します。複数の入力データを許可し、DynamicFrames のコレクションを返します。

- `Recipe` – レシピ オブジェクト。

  AWS Glue DataBrew レシピノードを指定します。

- `SnowflakeSource` – SnowflakeSource オブジェクト。

  Snowflake データソースを指定します。

- `SnowflakeTarget` – SnowflakeTarget オブジェクト。

  Snowflake データソースに書き込むターゲットを指定します。

- `ConnectorDataSource` – ConnectorDataSource オブジェクト。

  標準の接続オプションを使用して生成されたソースを指定します。

- `ConnectorDataTarget` – ConnectorDataTarget オブジェクト。

  標準の接続オプションを使用して生成されたターゲットを指定します。

JDBCConnectorOptions 構造

コネクタの追加接続オプション。

フィールド

- `FilterPredicate` – UTF-8 文字列。Custom string pattern #59 に一致。

  ソースからのデータをフィルタリングする追加の条件句。例:

  ```
  BillingCity='Mountain View'
  ```

  テーブル名ではなくクエリを使用する場合は、指定された `filterPredicate` でクエリが動作することを確認します。

- PartitionColumn – UTF-8 文字列。Custom string pattern #59 に一致。

  パーティショニングに使用される整数列の名前。このオプションは、lowerBound、upperBound、および numPartitions に含まれている場合にのみ機能します。このオプションの機能は、Spark SQL JDBC リーダーのものと同様です。

- LowerBound – 数値 (long)。None 以下。

  パーティションストライドを決定するために使用される partitionColumn の最小値を示す整数。

- UpperBound – 数値 (long)。None 以下。

  パーティションストライドを決定するために使用される partitionColumn の最大値を示す整数。

- NumPartitions – 数値 (long)。None 以下。

  パーティション数。この値は、lowerBound (範囲に含まれる) と upperBound (範囲に含まれない) とともに使用され、partitionColumn の分割で使用するために生成された WHERE 句の式のためのパーティションストライドを形成します。

- JobBookmarkKeys – UTF-8 文字列の配列。

  ソートするジョブブックマークキーの名前。

- JobBookmarkKeysSortOrder – UTF-8 文字列。Custom string pattern #59 に一致。

  昇順または降順のソート順を指定します。

- DataTypeMapping – キーバリューペアのマップ配列。

  各キーは UTF-8 文字列 (有効な値: ARRAY | BIGINT | BINARY | BIT | BLOB | BOOLEAN | CHAR | CLOB | DATALINK | DATE | DECIMAL | DISTINCT | DOUBLE | FLOAT | INTEGER | JAVA_OBJECT | LONGNVARCHAR | LONGVARBINARY | LONGVARCHAR | NCHAR | NCLOB | NULL | NUMERIC | NVARCHAR | OTHER | REAL | REF | REF_CURSOR | ROWID | SMALLINT | SQLXML | STRUCT | TIME | TIME_WITH_TIMEZONE | TIMESTAMP | TIMESTAMP_WITH_TIMEZONE | TINYINT | VARBINARY | VARCHAR) です。

  各値は UTF-8 文字列 (有効な値: DATE | STRING | TIMESTAMP | INT | FLOAT | LONG | BIGDECIMAL | BYTE | SHORT | DOUBLE) です。

  JDBC データタイプ から AWS Glue データタイプに対するマッピングを構築する、カスタムのデータタイプマッピング。例えば、オプション "dataTypeMapping":{"FLOAT":"STRING"}

はドライバーの ResultSet.getString() メソッドを呼び出すことで、JDBC タイプの FLOAT のデータフィールドを Java String タイプにマッピングし、それを使用して AWS Glue レコードを構築します。ResultSet オブジェクトは各ドライバによって実装されるため、その動作は使用するドライバにより決定されます。ドライバによる変換の実行方法については、JDBC ドライバのドキュメントを参照してください。

StreamingDataProviewOptions 構造

データのサンプルを表示するためのデータプレビューに関連するオプションを指定します。

フィールド

- PollingTime – 10 以上の数値 (long)。

  ミリ秒単位のポーリング時間。
- RecordPollingLimit – 1 以上の数値 (long)。

  ポーリングされるレコード数の制限。

AthenaConnectorSource 構造

Amazon Athena データソースへのコネクタを指定します。

フィールド

- Name – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

  データソースの名前。
- ConnectionName – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  コネクタに関連付けられている接続の名前。
- ConnectorName – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  AWS Glue Stuido のデータストアへのアクセスを支援するコネクタの名前。
- ConnectionType – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  marketplace.athena や custom.athena など、Amazon Athena データストアへの接続を指定する接続のタイプ。
- ConnectionTable – UTF-8 文字列。Custom string pattern #59 に一致。

データソース内のテーブルの名前。

- SchemaName – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  読み取り元となる Cloudwatch ロググループの名前。例えば、/aws-glue/jobs/output。
- OutputSchemas – GlueSchema オブジェクトの配列。

  カスタム Athena ソース用のデータスキーマを指定します。

JDBCConnectorSource 構造

JDBC データソースへのコネクタを指定します。

フィールド

- Name – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

  データソースの名前。
- ConnectionName – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  コネクタに関連付けられている接続の名前。
- ConnectorName – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  AWS Glue Stuido のデータストアへのアクセスを支援するコネクタの名前。
- ConnectionType – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  marketplace.jdbc や custom.jdbc など、JDBC データストアへの接続を指定する接続のタイプ。
- AdditionalOptions – JDBCConnectorOptions オブジェクト。

  コネクタの追加接続オプション。
- ConnectionTable – UTF-8 文字列。Custom string pattern #59 に一致。

  データソース内のテーブルの名前。
- Query – UTF-8 文字列。Custom string pattern #60 に一致。

  データを取得するテーブルまたは SQL クエリ。ConnectionTable または query を指定できます。両方を指定することはできません。
- OutputSchemas – GlueSchema オブジェクトの配列。

カスタム JDBC ソース用のデータスキーマを指定します。

SparkConnectorSource 構造

Apache Spark データソースへのコネクタを指定します。

フィールド

- Name – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

  データソースの名前。
- ConnectionName – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  コネクタに関連付けられている接続の名前。
- ConnectorName – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  AWS Glue Stuido のデータストアへのアクセスを支援するコネクタの名前。
- ConnectionType – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  marketplace.spark や custom.spark などのApache Spark データストアへの接続を指定する接続のタイプ。
- AdditionalOptions – キーバリューペアのマップ配列。

  各キーは、Custom string pattern #59 に適合する UTF-8 文字列です。

  各値は、Custom string pattern #59 に適合する UTF-8 文字列です。

  コネクタの追加接続オプション。
- OutputSchemas – GlueSchema オブジェクトの配列。

  カスタム Spark ソース用のデータスキーマを指定します。

CatalogSource 構造

AWS Glueデータカタログにデータストアを指定します。

フィールド

- Name – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

データストアの名前。

- `Database` – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  読み取りデータベースの名前。

- `Table` – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  読み取りデータベースのテーブルの名前。

MySQLCatalogSource 構造

AWS Glueデータカタログ内の MySQL データソースを指定します。

フィールド

- `Name` – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

  データソースの名前。

- `Database` – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  読み取りデータベースの名前。

- `Table` – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  読み取りデータベースのテーブルの名前。

PostgreSQLCatalogSource 構造

AWS Glueデータカタログ内の PostgresSQL データソースを指定します。

フィールド

- `Name` – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

  データソースの名前。

- `Database` – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  読み取りデータベースの名前。

- `Table` – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

読み取りデータベースのテーブルの名前。

OracleSQLCatalogSource 構造

AWS Glueデータカタログ内の Oracle データソースを指定します。

フィールド

- Name – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

  データソースの名前。
- Database – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  読み取りデータベースの名前。
- Table – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  読み取りデータベースのテーブルの名前。

MicrosoftSQLServerCatalogSource 構造

AWS Glueデータカタログ内の Microsoft SQL Server データソースを指定します。

フィールド

- Name – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

  データソースの名前。
- Database – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  読み取りデータベースの名前。
- Table – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  読み取りデータベースのテーブルの名前。

CatalogKinesisSource 構造

AWS Glueデータカタログで Kinesis データソースを指定します。

フィールド

- Name – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

  データソースの名前。
- WindowSize – 数値 (整数)、None 以下。

  各マイクロバッチの処理にかかる時間。
- DetectSchema – ブール。

  受信データからスキーマを自動的に決定するかの有無。
- Table – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  読み取りデータベースのテーブルの名前。
- Database – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  読み取りデータベースの名前。
- StreamingOptions – KinesisStreamingSourceOptions オブジェクト。

  Kinesis ストリーミングデータソースの追加オプション。
- DataPreviewOptions – StreamingDataPreviewOptions オブジェクト。

  データプレビューの追加オプション。

DirectKinesisSource 構造

直接 Amazon Kinesis のデータソースを指定します。

フィールド

- Name – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

  データソースの名前。
- WindowSize – 数値 (整数)、None 以下。

  各マイクロバッチの処理にかかる時間。
- DetectSchema – ブール。

  受信データからスキーマを自動的に決定するかどうか。

- `StreamingOptions` – KinesisStreamingSourceOptions オブジェクト。

  Kinesis ストリーミングデータソースの追加オプション。

- `DataPreviewOptions` – StreamingDataPreviewOptions オブジェクト。

  データプレビューの追加オプション。

KinesisStreamingSourceOptions 構造

Amazon Kinesis ストリーミングデータソースの追加オプション。

フィールド

- `EndpointUrl` – UTF-8 文字列。Custom string pattern #59 に一致。

  Kinesis エンドポイントの URL。

- `StreamName` – UTF-8 文字列。Custom string pattern #59 に一致。

  Kinesis データストリームの名前。

- `Classification` – UTF-8 文字列。Custom string pattern #59 に一致。

  オプションの分類。

- `Delimiter` – UTF-8 文字列。Custom string pattern #59 に一致。

  区切り記号文字を指定します。

- `StartingPosition` – UTF-8 文字列 (有効な値: `latest="LATEST"` | `trim_horizon="TRIM_HORIZON"` | `earliest="EARLIEST"` | `timestamp="TIMESTAMP"`)。

  Kinesis データストリーム内の、データの読み取り開始位置。指定できる値は `"latest"`、`"trim_horizon"`、`"earliest"`、またはパターン `yyyy-mm-ddTHH:MM:SSZ` の UTC 形式のタイムスタンプ文字列 (`Z` は UTC タイムゾーンのオフセットを +/- で表します。例: 『2023-04-04T08:00:00-04:00』) です。デフォルト値は `"latest"` です。

  注意: 「StartingPosition」の値に UTC 形式のタイムスタンプ文字列を使用できるのは、AWS Glue バージョン 4.0 以降のみです。

- `MaxFetchTimeInMs` – 数値 (long)。None 以下。

ジョブエグゼキューターが Kinesis データストリームから現在のバッチのレコードを読み取るために費やした最大時間は、ミリ秒 (ms) 単位で指定されます。この時間内に複数の GetRecords API コールを行うことができます。デフォルト値は 1000 です。

- MaxFetchRecordsPerShard – 数値 (long)。None 以下。

  1 マイクロバッチ当たりに Kinesis データストリームでシャードごとにフェッチするレコードの最大数。メモ: ストリーミングジョブが既に Kinesis (同じ get-records 呼び出しで) から余分なレコードを読み取っている場合、クライアントはこの制限を超えることができます。MaxFetchRecordsPerShard が厳密である必要がある場合、MaxRecordPerRead の倍数にする必要があります。デフォルト値は 100000 です。

- MaxRecordPerRead – 数値 (long)。None 以下。

  各 getRecords オペレーションごとに、Kinesis データストリームからフェッチするレコードの最大数。デフォルト値は 10000 です。

- AddIdleTimeBetweenReads – ブール。

  2 つの連続する getRecords オペレーション間の遅延時間を追加します。デフォルト値は "False" です。このオプションは、AWS Glue バージョン 2.0 以降でのみ設定可能です。

- IdleTimeBetweenReadsInMs – 数値 (long)。None 以下。

  2 つの連続する getRecords オペレーション間での、最短の遅延時間 (ミリ秒単位で指定)。デフォルト値は 1000 です。このオプションは、AWS Glue バージョン 2.0 以降でのみ設定可能です。

- DescribeShardInterval – 数値 (long)。None 以下。

  スクリプトが呼び出す 2 つの ListShards API コール間での、再シャーディングを考慮すべき最小時間。デフォルト値は 1s です。

- NumRetries – 数値 (整数)、None 以下。

  Kinesis Data Streams API リクエストを再試行する最大の回数。デフォルト値は 3 です。

- RetryIntervalMs – 数値 (long)。None 以下。

  Kinesis Data Streams API 呼び出しを再試行するまでのクールオフ期間 (ミリ秒単位で指定)。デフォルト値は 1000 です。

- MaxRetryIntervalMs – 数値 (long)。None 以下。

  再試行で 2 つの Kinesis Data Streams API を呼び出す間の最大クールオフ期間 (ミリ秒単位で指定)。デフォルト値は 10000 です。

- `AvoidEmptyBatches` – ブール。

  バッチ処理を開始する前に、Kinesis データストリームで未読のデータをチェックすることで、空のマイクロバッチジョブを作成しないようにします。デフォルト値は `"False"` です。

- `StreamArn` – UTF-8 文字列。Custom string pattern #59 に一致。

  Kinesis データストリームの Amazon リソースネーム (ARN)。

- `RoleArn` – UTF-8 文字列。Custom string pattern #59 に一致。

  AWS Security Token Service (AWS STS) の使用を引き受けるロールの、Amazon リソースネーム (ARN)。このロールには、Kinesis データストリームのレコードの説明操作または読み取り操作の権限が必要です。このパラメーターは、別のアカウントのデータストリームにアクセスするときに使用する必要があります。`"awsSTSSessionName"` と組み合わせて使用します。

- `RoleSessionName` – UTF-8 文字列。Custom string pattern #59 に一致。

  AWS STS を使用するロールを引き受ける、セッションの識別子。このパラメータは、別のアカウントのデータストリームにアクセスするときに使用する必要があります。`"awsSTSRoleARN"` と組み合わせて使用します。

- `AddRecordTimestamp` – UTF-8 文字列。Custom string pattern #59 に一致。

  このオプションが「true」に設定されている場合、データ出力には、対応するレコードがストリームによって受信された時刻を表示する「__src_timestamp」という名前が付けられた追加の列が含まれます。デフォルト値は、「false」です。このオプションは AWS Glue バージョン 4.0 以降でサポートされています。

- `EmitConsumerLagMetrics` – UTF-8 文字列。Custom string pattern #59 に一致。

  このオプションを「true」に設定すると、バッチごとに、ストリームが受信した最も古いレコードと、それが AWS Glue で CloudWatch に到着した時間との間隔のメトリクスが出力されます。メトリクスの名前は「glue.driver.streaming.maxConsumerLagInMs」です。デフォルト値は、「false」です。このオプションは AWS Glue バージョン 4.0 以降でサポートされています。

- `StartingTimestamp` – UTF–8 文字列。

  データの読み取りを開始する Kinesis データストリーム内のレコードのタイムスタンプ。指定できる値は、パターン `yyyy-mm-ddTHH:MM:SSZ` のUTC 形式のタイムスタンプ文字列 (Z は UTC タイムゾーンのオフセットを +/- で表します。例: 『2023-04-04T08:00:00+08:00』) です。

CatalogKafkaSource 構造

データカタログで Apache Kafka データストアを指定します。

フィールド

- `Name` – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

  データストアの名前。

- `WindowSize` – 数値 (整数)、None 以下。

  各マイクロバッチの処理にかかる時間。

- `DetectSchema` – ブール。

  受信データからスキーマを自動的に決定するかの有無。

- `Table` – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  読み取りデータベースのテーブルの名前。

- `Database` – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  読み取りデータベースの名前。

- `StreamingOptions` – KafkaStreamingSourceOptions オブジェクト。

  ストリーミングオプションを指定します。

- `DataPreviewOptions` – StreamingDataPreviewOptions オブジェクト。

  データのサンプルを表示するためのデータプレビューに関連するオプションを指定します。

DirectKafkaSource 構造

Apache Kafka データストアを指定します。

フィールド

- `Name` – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

  データストアの名前。

- `StreamingOptions` – KafkaStreamingSourceOptions オブジェクト。

ストリーミングオプションを指定します。

- `WindowSize` – 数値 (整数)、None 以下。

  各マイクロバッチの処理にかかる時間。

- `DetectSchema` – ブール。

  受信データからスキーマを自動的に決定するかどうか。

- `DataPreviewOptions` – StreamingDataPreviewOptions オブジェクト。

  データのサンプルを表示するためのデータプレビューに関連するオプションを指定します。

KafkaStreamingSourceOptions 構造

ストリーミングの追加オプション。

フィールド

- `BootstrapServers` – UTF-8 文字列。Custom string pattern #59 に一致。

  ブートストラップサーバーの URL のリスト (例: `b-1.vpc-test-2.o4q88o.c6.kafka.us-east-1.amazonaws.com:9094`)。このオプションは API 呼び出しで指定するか、データカタログ内のテーブルメタデータで定義する必要があります。

- `SecurityProtocol` – UTF-8 文字列。Custom string pattern #59 に一致。

  ブローカーと通信するために使用されるプロトコル。使用できる値は、`"SSL"` または `"PLAINTEXT"` です。

- `ConnectionName` – UTF-8 文字列。Custom string pattern #59 に一致。

  コレクションの名前。

- `TopicName` – UTF-8 文字列。Custom string pattern #59 に一致。

  Apache Kafka で指定されたトピック名。少なくくとも `"topicName"`、`"assign"`、`"subscribePattern"` の内いずれかを指定する必要があります。

- `Assign` – UTF-8 文字列。Custom string pattern #59 に一致。

  消費する特有の `TopicPartitions`。少なくくとも `"topicName"`、`"assign"`、`"subscribePattern"` の内いずれかを指定する必要があります。

- `SubscribePattern` – UTF-8 文字列。Custom string pattern #59 に一致。

  サブスクライブする先のトピックリストを識別する Java の正規表現文字列。少なくくとも `"topicName"`、`"assign"`、`"subscribePattern"` の内いずれかを指定する必要があります。
- `Classification` – UTF-8 文字列。Custom string pattern #59 に一致。

  オプションの分類。
- `Delimiter` – UTF-8 文字列。Custom string pattern #59 に一致。

  区切り記号文字を指定します。
- `StartingOffsets` – UTF-8 文字列。Custom string pattern #59 に一致。

  Kafka トピック内で、データの読み取りを開始する位置。使用できる値は、`"earliest"` または `"latest"` です。デフォルト値は `"latest"` です。
- `EndingOffsets` – UTF-8 文字列。Custom string pattern #59 に一致。

  バッチクエリの終了位置。設定が可能な値は、`"latest"` または、各 `TopicPartition` の終了オフセットを指定する JSON 文字列のいずれかです。
- `PollTimeoutMs` – 数値 (long)。None 以下。

  Spark ジョブエグゼキュータで、Kafka からデータをポーリングする際のタイムアウト値 (ミリ秒単位)。デフォルト値は `512` です。
- `NumRetries` – 数値 (整数)、None 以下。

  Kafka オフセットのフェッチが失敗したと判断される前の再試行回数。デフォルト値は `3` です。
- `RetryIntervalMs` – 数値 (long)。None 以下。

  Kafka オフセットのフェッチを開始するまでの待機時間 (ミリ秒)。デフォルト値は `10` です。
- `MaxOffsetsPerTrigger` – 数値 (long)。None 以下。

  処理されるオフセットの最大数を、トリガー間隔ごとのレート上限で指定する値。指定されたオフセットの合計数は、異なるボリュームの `topicPartitions` 間で均等に分割されます。デフォルト値はnullです。この場合、コンシューマーは既知の最新のオフセットまで、すべてのオフセットを読み取ります。
- `MinPartitions` – 数値 (整数)、None 以下。

  Kafka から読み取ることを想定する、最小のパーティション数。デフォルト値はnullです。これは、Spark パーティションの数が Kafka パーティションの数に等しいことを意味します。

- `IncludeHeaders` – ブール。

  Kafka ヘッダーを含めるかどうかを決定します。このオプションが『true』に設定されている場合、データ出力には、『glue_streaming_kafka_headers』という名前で `Array[Struct(key: String, value: String)]` タイプの列が追加されます。デフォルト値は『false』です。このオプションは AWS Glue バージョン 3.0 以降でのみ使用可能です。

- `AddRecordTimestamp` – UTF-8 文字列。Custom string pattern #59 に一致。

  このオプションが「true」に設定されている場合、データ出力には、対応するレコードがトピックによって受信された時刻を表示する「__src_timestamp」という名前が付けられた追加の列が含まれます。デフォルト値は、「false」です。このオプションは AWS Glue バージョン 4.0 以降でサポートされています。

- `EmitConsumerLagMetrics` – UTF-8 文字列。Custom string pattern #59 に一致。

  このオプションを 'true' に設定すると、バッチごとに、トピックが受信した最も古いレコードと、それが AWS Glue で CloudWatch に到着した時間との間隔のメトリクススが出力されます。メトリクスの名前は「glue.driver.streaming.maxConsumerLagInMs」です。デフォルト値は、「false」です。このオプションは AWS Glue バージョン 4.0 以降でサポートされています。

- `StartingTimestamp` – UTF–8 文字列。

  データの読み取りを開始する Kafka トピック内のレコードのタイムスタンプ。指定できる値は、パターン `yyyy-mm-ddTHH:MM:SSZ` のUTC 形式のタイムスタンプ文字列 (Z は UTC タイムゾーンのオフセットを +/- で表します。例: 『2023-04-04T08:00:00+08:00』) です。

  `StartingTimestamp` または `StartingOffsets` のいずれかのみを設定する必要があります。

RedShiftSource 構造

Amazon Redshift データストアを指定します。

フィールド

- `Name` – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

  Amazon Redshift データストアの名前。

- `Database` – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  読み込むデータベース。

- `Table` – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  読み取り元のデータベーステーブル。
- `RedshiftTmpDir` – UTF-8 文字列。Custom string pattern #59 に一致。

  データベースからコピーするときに一時データをステージングできる Amazon S3 パス。
- `TmpDirIAMRole` – UTF-8 文字列。Custom string pattern #59 に一致。

  アクセス許可を持つ IAM ロール。

AmazonRedshiftSource 構造

Amazon Redshift ソースを指定します。

フィールド

- `Name` – UTF-8 文字列。Custom string pattern #61 に一致。

  Amazon Redshift ソースの名前。
- `Data` – AmazonRedshiftNodeData オブジェクト。

  Amazon Reshift ソースノードのデータを指定します。

AmazonRedshiftNodeData 構造

Amazon Redshift ノードを指定します。

フィールド

- `AccessType` – UTF-8 文字列。Custom string pattern #58 に一致。

  Redshift 接続のアクセスタイプ。直接接続またはカタログ接続が可能です。
- `SourceType` – UTF-8 文字列。Custom string pattern #58 に一致。

  特定のテーブルがソースかカスタムクエリかを指定するソースタイプ。
- `Connection` – オプション オブジェクト。

  Redshift クラスターへの AWS Glue の接続。
- `Schema` – オプション オブジェクト。

直接接続で作業するときの Redshift スキーマの名前。

- `Table` – オプション オブジェクト。

  直接接続で作業するときの Redshift テーブルの名前。

- `CatalogDatabase` – オプション オブジェクト。

  データカタログで作業するときの AWS Glue データカタログデータベースの名前。

- `CatalogTable` – オプション オブジェクト。

  データカタログで作業するときの AWS Glue データカタログテーブルの名前。

- `CatalogRedshiftSchema` – UTF–8 文字列。

  データカタログで作業するときの Redshift スキーマの名前。

- `CatalogRedshiftTable` – UTF–8 文字列。

  読み取り元のデータベーステーブル。

- `TempDir` – UTF-8 文字列。Custom string pattern #59 に一致。

  データベースからコピーするときに一時データをステージングできる Amazon S3 パス。

- `IamRole` – オプション オブジェクト。

  オプション。S3 に接続するときに使用するロールの名前。空欄のままにすると、IAM ロールはデフォルトでジョブのロールになります。

- `AdvancedOptions` – AmazonRedshiftAdvancedOption オブジェクトの配列。

  Redshift クラスターに接続するときのオプションの値。

- `SampleQuery` – UTF–8 文字列。

  SourceType が「query」の場合に Redshift ソースからデータを取得するために使用される SQL。

- `PreAction` – UTF–8 文字列。

  upsert を用いる MERGE または APPEND を実行する前に使用される SQL。

- `PostAction` – UTF–8 文字列。

  upsert を用いる MERGE または APPEND を実行する前に使用される SQL。

- `Action` – UTF–8 文字列。

Redshift クラスターへの書き込み方法を指定します。

- `TablePrefix` – UTF-8 文字列。Custom string pattern #58 に一致。

  テーブルへのプレフィックスを指定します。

- `Upsert` – ブール。

  APPEND を実行するときに Redshift シンクで使用するアクション。

- `MergeAction` – UTF-8 文字列。Custom string pattern #58 に一致。

  Redshift シンク内の MERGE の処理方法を決定するときに使用するアクション。

- `MergeWhenMatched` – UTF-8 文字列。Custom string pattern #58 に一致。

  既存のレコードが新しいレコードと一致する場合、Redshift シンク内の MERGE の処理方法を決定するときに使用するアクション。

- `MergeWhenNotMatched` – UTF-8 文字列。Custom string pattern #58 に一致。

  既存のレコードが新しいレコードと一致しない場合、Redshift シンク内の MERGE の処理方法を決定するときに使用するアクション。

- `MergeClause` – UTF–8 文字列。

  一致するレコードを処理するためにカスタムマージで使用される SQL。

- `CrawlerConnection` – UTF–8 文字列。

  使用するカタログテーブルに関連する接続の名前を指定します。

- `TableSchema` – オプション オブジェクトの配列。

  特定のノードにおけるスキーマ出力の配列。

- `StagingTable` – UTF–8 文字列。

  upsert を用いる MERGE または APPEND を実行するときに使用する一時的なステージングテーブルの名前。

- `SelectedColumns` – オプション オブジェクトの配列。

  upsert を用いる MERGE または APPEND を実行するときに、一致するレコードを決定するために使用する列の名前のリスト。

AmazonRedshiftAdvancedOption 構造

Redshift クラスターに接続するときのオプションの値を指定します。

フィールド

- `Key` – UTF-8 文字列。

  追加接続オプションのキー。
- `Value` – UTF–8 文字列。

  追加接続オプションの値。

Option 構造

オプションの値を指定します。

フィールド

- `Value` – UTF-8 文字列。Custom string pattern #59 に一致。

  オプションの値を指定します。
- `Label` – UTF-8 文字列。Custom string pattern #59 に一致。

  オプションのラベルを指定します。
- `Description` – UTF-8 文字列。Custom string pattern #59 に一致。

  オプションの説明を指定します。

S3CatalogSource 構造

AWS Glueデータカタログに Amazon S3 データストアを指定します。

フィールド

- `Name` – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

  データストアの名前。
- `Database` – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

読み込むデータベース。

- Table – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  読み取り元のデータベーステーブル。

- PartitionPredicate – UTF-8 文字列。Custom string pattern #59 に一致。

  この述語を満たすパーティションは削除されます。これらのパーティションの保存期間内のファイルは削除されません。"" を設定 – デフォルトでは空です。

- AdditionalOptions – S3SourceAdditionalOptions オブジェクト。

  追加の接続オプションを指定します。

S3SourceAdditionalOptions 構造

Amazon S3 データストアの追加の接続オプションを指定します。

フィールド

- BoundedSize – 数値 (long)。

  処理されるバイトのデータセットのターゲットサイズの上限を設定します。

- BoundedFiles – 数値 (long)。

  処理されるファイルのターゲット数の上限を設定します。

S3CsvSource 構造

Amazon S3 に格納されているコマンド区切り値 (CSV) データストアを指定します。

フィールド

- Name – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

  データストアの名前。

- Paths – 必須: UTF-8 文字列の配列。

  読み取りのソースとなる Amazon S3 パスのリスト。

- CompressionType – UTF-8 文字列 (有効な値: gzip="GZIP" | bzip2="BZIP2")。

データの圧縮方法を指定します。データに標準のファイル拡張子が付いている場合、このオプションは一般的に不要です。指定できる値は "gzip" および "bzip" です。

- Exclusions – UTF-8 文字列の配列。

  除外する Unix スタイルの glob パターンの JSON リストを含む文字列。例えば、"[\"**.pdf\"]" はすべての PDF ファイルを除外します。

- GroupSize – UTF-8 文字列。Custom string pattern #59 に一致。

  ターゲットグループのサイズ (バイト単位)。デフォルトは、入力データのサイズとクラスターのサイズに基づいて計算されます。入力ファイルが 50,000 個未満の場合、このオプションを有効にするには、"groupFiles" を "inPartition" に設定する必要があります。

- GroupFiles – UTF-8 文字列。Custom string pattern #59 に一致。

  入力ファイルが 50,000 個を超える場合、デフォルトでファイルのグループ化が有効化されます。入力ファイルが 50,000 個未満の場合にグループ化を有効化するには、このパラメータに『inPartition』を設定します。入力ファイルが 50,000 個を超える場合に、グループ化を無効にするには、このパラメータを "none" に設定します。

- Recurse – ブール。

  true に設定した場合は、指定したパスの下にあるすべてのサブディレクトリ内のファイルを再帰的に読み取ります。

- MaxBand – 数値 (整数)、None 以下。

  このオプションでは、s3 リストの一貫性が認められるまでの期間をミリ秒単位で指定します。Amazon S3 の結果整合性を担保するために、直前の maxBand ミリ秒以内の変更タイムスタンプが付いたファイルが、特に JobBookmarks の使用時に追跡されます。ほとんどのユーザーはこのオプションを設定する必要はありません。デフォルトは 900,000 ミリ秒 (15 分) です。

- MaxFilesInBand – 数値 (整数)、None 以下。

  このオプションは、直前の maxBand 秒間に保存するファイルの最大数を指定します。この数を超えた場合、余分なファイルはスキップされ、次のジョブ実行時にのみ処理されます。

- AdditionalOptions – S3DirectSourceAdditionalOptions オブジェクト。

  追加の接続オプションを指定します。

- Separator – 必須: UTF-8 文字列 (有効な値: comma="COMMA" | ctrla="CTRLA" | pipe="PIPE" | semicolon="SEMICOLON" | tab="TAB")。

区切り記号文字を指定します。デフォルトではカンマ: 『,』 ですが、他の任意の文字を指定できます。

- `Escaper` – UTF-8 文字列。Custom string pattern #59 に一致。

  エスケープに使用する文字を指定します。このオプションは、CSV ファイルを読み取る場合にのみ使用します。デフォルト値は none です。有効にすると、直後の文字はそのまま使用されます。ただし、よく知られている小さいエスケープセット (\n、\r、\t、\0) を除きます。

- `QuoteChar` – 必須: UTF-8 文字列 (有効な値: `quote="QUOTE"` | `quillemet="QUILLEMET"` | `single_quote="SINGLE_QUOTE"` | `disabled="DISABLED"`)。

  引用に使用する文字を指定します。デフォルト値は二重引用符 `'"'` です。これに `-1` を設定すると、全体的に引用が無効になります。

- `Multiline` – ブール。

  単一のレコードが複数行にまたがることができるかどうかを指定するブール値。これが発生するのは、フィールドに引用符で囲まれた改行文字がある場合などです。複数行にまたがるレコードがある場合は、このオプションを True に設定する必要があります。デフォルト値は `False` であり、解析時によりアグレッシブなファイル分割を可能にします。

- `WithHeader` – ブール。

  最初の行をヘッダーとして扱うかどうかを指定するブール値。デフォルト値は `False` です。

- `WriteHeader` – ブール。

  ヘッダーを出力に書き込むかどうかを指定するブール値。デフォルト値は `True` です。

- `SkipFirst` – ブール。

  最初のデータ行をスキップするかどうかを指定するブール値。デフォルト値は `False` です。

- `OptimizePerformance` – ブール。

  高度な SIMD CSV リーダーで、Apache Arrow ベースの列指向メモリ形式を使用するかどうかを指定するブール値。AWS Glue バージョン 3.0 でのみ使用可能

- `OutputSchemas` – GlueSchema オブジェクトの配列。

  S3 CSV ソース用のデータスキーマを指定します。

DirectJDBCSource 構造

直接 JDBC ソース接続を指定します。

フィールド

- Name – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

  JDBC ソース接続の名前。
- Database – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  JDBC ソース接続のデータベース。
- Table – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  JDBC ソース接続のテーブル。
- ConnectionName – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  JDBC ソースの接続名。
- ConnectionType – 必須: UTF-8 文字列 (有効な値: sqlserver | mysql | oracle | postgresql | redshift)。

  JDBC ソースの接続タイプ。
- RedshiftTmpDir – UTF-8 文字列。Custom string pattern #59 に一致。

  JDBC Redshift ソースの一時ディレクトリ。

S3DirectSourceAdditionalOptions 構造

Amazon S3 データストアの追加の接続オプションを指定します。

フィールド

- BoundedSize – 数値 (long)。

  処理されるバイトのデータセットのターゲットサイズの上限を設定します。
- BoundedFiles – 数値 (long)。

  処理されるファイルのターゲット数の上限を設定します。
- EnableSamplePath – ブール。

オプションを設定しサンプルパスを有効にします。

- SamplePath – UTF-8 文字列。Custom string pattern #59 に一致。

  有効にした場合は、サンプルパスを指定します。

S3JsonSource 構造

Amazon S3 の JSON データストアを指定します。

フィールド

- Name – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

  データストアの名前。

- Paths – 必須: UTF-8 文字列の配列。

  読み取りのソースとなる Amazon S3 パスのリスト。

- CompressionType – UTF-8 文字列 (有効な値: gzip="GZIP" | bzip2="BZIP2")。

  データの圧縮方法を指定します。データに標準のファイル拡張子が付いている場合、このオプションは一般的に不要です。指定できる値は "gzip" および "bzip" です。

- Exclusions – UTF-8 文字列の配列。

  除外する Unix スタイルの glob パターンの JSON リストを含む文字列。例えば、"[\"**.pdf\"]" はすべての PDF ファイルを除外します。

- GroupSize – UTF-8 文字列。Custom string pattern #59 に一致。

  ターゲットグループのサイズ (バイト単位)。デフォルトは、入力データのサイズとクラスターのサイズに基づいて計算されます。入力ファイルが 50,000 個未満の場合、このオプションを有効にするには、"groupFiles" を "inPartition" に設定する必要があります。

- GroupFiles – UTF-8 文字列。Custom string pattern #59 に一致。

  入力ファイルが 50,000 個を超える場合、デフォルトでファイルのグループ化が有効化されます。入力ファイルが 50,000 個未満の場合にグループ化を有効化するには、このパラメータに『inPartition』を設定します。入力ファイルが 50,000 個を超える場合に、グループ化を無効にするには、このパラメータを "none" に設定します。

- Recurse – ブール。

true に設定した場合は、指定したパスの下にあるすべてのサブディレクトリ内のファイルを再帰的に読み取ります。

- `MaxBand` – 数値 (整数)、None 以下。

  このオプションでは、s3 リストの一貫性が認められるまでの期間をミリ秒単位で指定します。Amazon S3 の結果整合性を担保するために、直前の maxBand ミリ秒以内の変更タイムスタンプが付いたファイルが、特に JobBookmarks の使用時に追跡されます。ほとんどのユーザーはこのオプションを設定する必要はありません。デフォルトは 900,000 ミリ秒 (15 分) です。

- `MaxFilesInBand` – 数値 (整数)、None 以下。

  このオプションは、直前の maxBand 秒間に保存するファイルの最大数を指定します。この数を超えた場合、余分なファイルはスキップされ、次のジョブ実行時にのみ処理されます。

- `AdditionalOptions` – S3DirectSourceAdditionalOptions オブジェクト。

  追加の接続オプションを指定します。

- `JsonPath` – UTF-8 文字列。Custom string pattern #59 に一致。

  JSON データを定義する JsonPath 文字列。

- `Multiline` – ブール。

  単一のレコードが複数行にまたがることができるかどうかを指定するブール値。これが発生するのは、フィールドに引用符で囲まれた改行文字がある場合などです。複数行にまたがるレコードがある場合は、このオプションを True に設定する必要があります。デフォルト値は `False` であり、解析時によりアグレッシブなファイル分割を可能にします。

- `OutputSchemas` – GlueSchema オブジェクトの配列。

  S3 JSON ソース用のデータスキーマを指定します。

S3ParquetSource 構造

Amazon S3 に保存されている Apache Parquet データストアを指定します。

フィールド

- `Name` – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

  データストアの名前。

- `Paths` – 必須: UTF-8 文字列の配列。

  読み取りのソースとなる Amazon S3 パスのリスト。

- `CompressionType` – UTF-8 文字列 (有効な値: `snappy="SNAPPY"` | `lzo="LZO"` | `gzip="GZIP"` | `brotli="BROTLI"` | `lz4="LZ4"` | `uncompressed="UNCOMPRESSED"` | `none="NONE"`)。

  データの圧縮方法を指定します。データに標準のファイル拡張子が付いている場合、このオプションは一般的に不要です。指定できる値は `"gzip"` および `"bzip"` です。

- `Exclusions` – UTF-8 文字列の配列。

  除外する Unix スタイルの glob パターンの JSON リストを含む文字列。例えば、"[\"**.pdf\"]" はすべての PDF ファイルを除外します。

- `GroupSize` – UTF-8 文字列。Custom string pattern #59 に一致。

  ターゲットグループのサイズ (バイト単位)。デフォルトは、入力データのサイズとクラスターのサイズに基づいて計算されます。入力ファイルが 50,000 個未満の場合、このオプションを有効にするには、`"groupFiles"` を `"inPartition"` に設定する必要があります。

- `GroupFiles` – UTF-8 文字列。Custom string pattern #59 に一致。

  入力ファイルが 50,000 個を超える場合、デフォルトでファイルのグループ化が有効化されます。入力ファイルが 50,000 個未満の場合にグループ化を有効化するには、このパラメータに『inPartition』を設定します。入力ファイルが 50,000 個を超える場合に、グループ化を無効にするには、このパラメータを `"none"` に設定します。

- `Recurse` – ブール。

  true に設定した場合は、指定したパスの下にあるすべてのサブディレクトリ内のファイルを再帰的に読み取ります。

- `MaxBand` – 数値 (整数)、None 以下。

  このオプションでは、s3 リストの一貫性が認められるまでの期間をミリ秒単位で指定します。Amazon S3 の結果整合性を担保するために、直前の maxBand ミリ秒以内の変更タイムスタンプが付いたファイルが、特に JobBookmarks の使用時に追跡されます。ほとんどのユーザーはこのオプションを設定する必要はありません。デフォルトは 900,000 ミリ秒 (15 分) です。

- `MaxFilesInBand` – 数値 (整数)、None 以下。

  このオプションは、直前の maxBand 秒間に保存するファイルの最大数を指定します。この数を超えた場合、余分なファイルはスキップされ、次のジョブ実行時にのみ処理されます。

- `AdditionalOptions` – S3DirectSourceAdditionalOptions オブジェクト。

  追加の接続オプションを指定します。

- `OutputSchemas` – GlueSchema オブジェクトの配列。

  S3 Parquet ソース用のデータスキーマを指定します。

S3DeltaSource 構造

Amazon S3 内に保存されている Delta Lake データソースを指定します。

フィールド

- `Name` – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

  Delta Lake ソースの名前。

- `Paths` – 必須: UTF-8 文字列の配列。

  読み取りのソースとなる Amazon S3 パスのリスト。

- `AdditionalDeltaOptions` – キーバリューペアのマップ配列。

  各キーは、Custom string pattern #59 に適合する UTF-8 文字列です。

  各値は、Custom string pattern #59 に適合する UTF-8 文字列です。

  追加の接続オプションを指定します。

- `AdditionalOptions` – S3DirectSourceAdditionalOptions オブジェクト。

  コネクタの追加オプションを指定します。

- `OutputSchemas` – GlueSchema オブジェクトの配列。

  Delta Lake ソース用のデータスキーマを指定します。

S3CatalogDeltaSource 構造

AWS Glue データカタログ内に登録されている Delta Lake データソースを指定します。データソースは Amazon S3 に保存する必要があります。

フィールド

- Name – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

  Delta Lake データソースの名前。
- Database – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  読み取りデータベースの名前。
- Table – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  読み取りデータベースのテーブルの名前。
- AdditionalDeltaOptions – キーバリューペアのマップ配列。

  各キーは、Custom string pattern #59 に適合する UTF-8 文字列です。

  各値は、Custom string pattern #59 に適合する UTF-8 文字列です。

  追加の接続オプションを指定します。
- OutputSchemas – GlueSchema オブジェクトの配列。

  Delta Lake ソース用のデータスキーマを指定します。

CatalogDeltaSource 構造

AWS Glue データカタログ内に登録されている Delta Lake データソースを指定します。

フィールド

- Name – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

  Delta Lake データソースの名前。
- Database – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  読み取りデータベースの名前。
- Table – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  読み取りデータベースのテーブルの名前。
- AdditionalDeltaOptions – キーバリューペアのマップ配列。

  各キーは、Custom string pattern #59 に適合する UTF-8 文字列です。

各値は、Custom string pattern #59 に適合する UTF-8 文字列です。

追加の接続オプションを指定します。

- `OutputSchemas` – GlueSchema オブジェクトの配列。

  Delta Lake ソース用のデータスキーマを指定します。

S3HudiSource 構造

Amazon S3 内に保存されている Hudi データソースを指定します。

フィールド

- `Name` – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

  Hudi ソースの名前。
- `Paths` – 必須: UTF-8 文字列の配列。

  読み取りのソースとなる Amazon S3 パスのリスト。
- `AdditionalHudiOptions` – キーバリューペアのマップ配列。

  各キーは、Custom string pattern #59 に適合する UTF-8 文字列です。

  各値は、Custom string pattern #59 に適合する UTF-8 文字列です。

  追加の接続オプションを指定します。
- `AdditionalOptions` – S3DirectSourceAdditionalOptions オブジェクト。

  コネクタの追加オプションを指定します。
- `OutputSchemas` – GlueSchema オブジェクトの配列。

  Hudi ソース用のデータスキーマを指定します。

S3CatalogHudiSource 構造

AWS Glue データカタログ内に登録されている Hudi データソースを指定します。Hudi データソースは Amazon S3 に保存する必要があります。

フィールド

- Name – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

  Hudi データソースの名前。
- Database – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  読み取りデータベースの名前。
- Table – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  読み取りデータベースのテーブルの名前。
- AdditionalHudiOptions – キーバリューペアのマップ配列。

  各キーは、Custom string pattern #59 に適合する UTF-8 文字列です。

  各値は、Custom string pattern #59 に適合する UTF-8 文字列です。

  追加の接続オプションを指定します。
- OutputSchemas – GlueSchema オブジェクトの配列。

  Hudi ソース用のデータスキーマを指定します。

S3ExcelSource の構造

S3 Excel データソースを指定します。

フィールド

- Name – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

  S3 Excel データソースの名前。
- Paths – 必須: UTF-8 文字列の配列。

  Excel ファイルがある S3 パス。
- CompressionType – UTF-8 文字列 (有効な値: snappy="SNAPPY" | lzo="LZO" | gzip="GZIP" | brotli="BROTLI" | lz4="LZ4" | uncompressed="UNCOMPRESSED" | none="NONE")。

  Excel ファイルに使用される圧縮形式。
- Exclusions – UTF-8 文字列の配列。

特定のファイルまたはパスが処理されないように除外するパターン。

- `GroupSize` – UTF-8 文字列。Custom string pattern #59 に一致。

  バッチ処理するファイルグループのサイズを定義します。

- `GroupFiles` – UTF-8 文字列。Custom string pattern #59 に一致。

  処理するためにファイルをグループ化する方法を指定します。

- `Recurse` – ブール。

  サブディレクトリを再帰的に処理するかどうか示します。

- `MaxBand` – 数値 (整数)、None 以下。

  使用する処理バンドの最大数。

- `MaxFilesInBand` – 数値 (整数)、None 以下。

  バンドごとに処理するファイルの最大数。

- `AdditionalOptions` – S3DirectSourceAdditionalOptions オブジェクト。

  S3 の直接ソース処理の追加設定オプション。

- `NumberRows` – 数値 (long)。

  各 Excel ファイルで処理する行数。

- `SkipFooter` – 数値 (整数)、None 以下。

  各 Excel ファイルの最後にスキップする行数。

- `OutputSchemas` – GlueSchema オブジェクトの配列。

  処理されたデータに適用する AWS Glue スキーマ。

CatalogHudiSource 構造

AWS Glue データカタログ内に登録されている Hudi データソースを指定します。

フィールド

- `Name` – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

  Hudi データソースの名前。

- `Database` – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  読み取りデータベースの名前。
- `Table` – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  読み取りデータベースのテーブルの名前。
- `AdditionalHudiOptions` – キーバリューペアのマップ配列。

  各キーは、Custom string pattern #59 に適合する UTF-8 文字列です。

  各値は、Custom string pattern #59 に適合する UTF-8 文字列です。

  追加の接続オプションを指定します。
- `OutputSchemas` – GlueSchema オブジェクトの配列。

  Hudi ソース用のデータスキーマを指定します。

DynamoDBCatalogSource 構造

AWS Glue データカタログ内の DynamoDB データソースを指定します。

フィールド

- `Name` – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

  データソースの名前。
- `Database` – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  読み取りデータベースの名前。
- `Table` – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  読み取りデータベースのテーブルの名前。

RelationalCatalogSource 構造

AWS Glue データカタログ内の、リレーショナルデータベースデータソースを指定します。

フィールド

- `Name` – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

データソースの名前。

- `Database` – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  読み取りデータベースの名前。

- `Table` – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  読み取りデータベースのテーブルの名前。

JDBCConnectorTarget 構造

Apache Parquet 列指向ストレージで Amazon S3 に書き込むデータターゲットを指定します。

フィールド

- `Name` – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

  データターゲットの名前。

- `Inputs` – 必須: UTF-8 文字列の配列。1 個の文字列。

  データターゲットへの入力であるノード。

- `ConnectionName` – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  コネクタに関連付けられている接続の名前。

- `ConnectionTable` – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  データターゲットのテーブルの名前。

- `ConnectorName` – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  使用されるコネクタの名前。

- `ConnectionType` – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  JDBC データターゲットへの接続を指定する marketplace.jdbc や custom.jdbc などの接続のタイプ。

- `AdditionalOptions` – キーバリューペアのマップ配列。

  各キーは、Custom string pattern #59 に適合する UTF-8 文字列です。

  各値は、Custom string pattern #59 に適合する UTF-8 文字列です。

コネクタの追加接続オプション。

- `OutputSchemas` – GlueSchema オブジェクトの配列。

  JDBC ターゲット用のデータスキーマを指定します。

SparkConnectorTerg 構造

Apache Spark コネクタを使用するターゲットを指定します。

フィールド

- `Name` – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

  データターゲットの名前。

- `Inputs` – 必須: UTF-8 文字列の配列。1 個の文字列。

  データターゲットへの入力であるノード。

- `ConnectionName` – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  Apache Spark コネクタの接続の名前。

- `ConnectorName` – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  Apache Spark コネクタの名前。

- `ConnectionType` – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  marketplace.spark や custom.spark などのApache Spark データストアへの接続を指定する接続のタイプ。

- `AdditionalOptions` – キーバリューペアのマップ配列。

  各キーは、Custom string pattern #59 に適合する UTF-8 文字列です。

  各値は、Custom string pattern #59 に適合する UTF-8 文字列です。

  コネクタの追加接続オプション。

- `OutputSchemas` – GlueSchema オブジェクトの配列。

  カスタム Spark ターゲット用のデータスキーマを指定します。

BasicCatalogTerg 構造

AWS Glue データカタログテーブルを使用するターゲットを指定します。

フィールド

- Name – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

  データターゲットの名前。

- Inputs – 必須: UTF-8 文字列の配列。1 個の文字列。

  データターゲットへの入力であるノード。

- PartitionKeys – UTF-8 文字列の配列。

  特定のキーまたはキーのセットに基づいて、複数のパーティションまたはシャードにデータを分散するために使用されるパーティションキー。

- Database – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  ターゲットとして使用するテーブルを含むデータベース。このデータベースは、データカタログに既に存在している必要があります。

- Table – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  出力データのスキーマを定義するテーブル。このテーブルは、 のデータカタログに既に存在している必要があります。

MySQLCatalogTarget 構造

MySQL を使用するターゲットを指定します。

フィールド

- Name – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

  データターゲットの名前。

- Inputs – 必須: UTF-8 文字列の配列。1 個の文字列。

  データターゲットへの入力であるノード。

- Database – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

書き込むデータベースの名前。

- Table – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  書き込むデータベーステーブルの名前。

PostgreSQLCatalogTarget 構造

Postgres SQL を使用するターゲットを指定します。

フィールド

- Name – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

  データターゲットの名前。

- Inputs – 必須: UTF-8 文字列の配列。1 個の文字列。

  データターゲットへの入力であるノード。

- Database – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  書き込むデータベースの名前。

- Table – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  書き込むデータベーステーブルの名前。

OracleSQLCatalogTarget 構造

Oracle SQL を使用するターゲットを指定します。

フィールド

- Name – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

  データターゲットの名前。

- Inputs – 必須: UTF-8 文字列の配列。1 個の文字列。

  データターゲットへの入力であるノード。

- Database – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

書き込むデータベースの名前。

- `Table` – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  書き込むデータベーステーブルの名前。

MicrosoftSQLServerCatalogTarget 構造

Microsoft SQL を使用するターゲットを指定します。

フィールド

- `Name` – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

  データターゲットの名前。

- `Inputs` – 必須: UTF-8 文字列の配列。1 個の文字列。

  データターゲットへの入力であるノード。

- `Database` – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  書き込むデータベースの名前。

- `Table` – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  書き込むデータベーステーブルの名前。

RedShiftTarget 構造

Amazon Redshift を使用するターゲットを指定します。

フィールド

- `Name` – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

  データターゲットの名前。

- `Inputs` – 必須: UTF-8 文字列の配列。1 個の文字列。

  データターゲットへの入力であるノード。

- `Database` – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

書き込むデータベースの名前。

- `Table` – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  書き込むデータベーステーブルの名前。

- `RedshiftTmpDir` – UTF-8 文字列。Custom string pattern #59 に一致。

  データベースからコピーするときに一時データをステージングできる Amazon S3 パス。

- `TmpDirIAMRole` – UTF-8 文字列。Custom string pattern #59 に一致。

  アクセス許可を持つ IAM ロール。

- `UpsertRedshiftOptions` – UpsertRedshiftTargetOptions オブジェクト。

  Redshift ターゲットに書き込む際の upsert 処理を設定するためのオプションセット。

AmazonRedshiftTarget 構造

Amazon Redshift ターゲットを指定します。

フィールド

- `Name` – UTF-8 文字列。Custom string pattern #61 に一致。

  Amazon Redshift ターゲットの名前。

- `Data` – AmazonRedshiftNodeData オブジェクト。

  Amazon Redshift ターゲットノードのデータを指定します。

- `Inputs` – UTF-8 文字列の配列、1 個の文字列。

  データターゲットへの入力であるノード。

UpsertRedshiftTargetOptions 構造

Redshift ターゲットに書き込む際の upsert 処理を設定するオプション。

フィールド

- `TableLocation` – UTF-8 文字列。Custom string pattern #59 に一致。

Redshift テーブルの物理的な場所。

- `ConnectionName` – UTF-8 文字列。Custom string pattern #59 に一致。

  Redshift に書き込むために使用する接続名。

- `UpsertKeys` – UTF-8 文字列の配列。

  更新または挿入のどちらを実行するかを決定するためのキー。

S3CatalogTarget 構造

AWS Glue データカタログを使用して Amazon S3 に書き込むデータターゲットを指定します。

フィールド

- `Name` – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

  データターゲットの名前。

- `Inputs` – 必須: UTF-8 文字列の配列。1 個の文字列。

  データターゲットへの入力であるノード。

- `PartitionKeys` – UTF-8 文字列の配列。

  一連のキーを使用してネイティブパーティショニングを指定します。

- `Table` – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  書き込むデータベーステーブルの名前。

- `Database` – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  書き込むデータベースの名前。

- `SchemaChangePolicy` – CatalogSchemaChangePolicy オブジェクト。

  クローラの更新の動作を指定するポリシー。

S3GlueParquetTarget 構造

Apache Parquet 列指向ストレージで Amazon S3 に書き込むデータターゲットを指定します。

フィールド

- Name – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

  データターゲットの名前。
- Inputs – 必須: UTF-8 文字列の配列。1 個の文字列。

  データターゲットへの入力であるノード。
- PartitionKeys – UTF-8 文字列の配列。

  一連のキーを使用してネイティブパーティショニングを指定します。
- Path – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  書き込む単一の Amazon S3 パス。
- Compression – UTF-8 文字列 (有効な値: snappy="SNAPPY" | lzo="LZO" | gzip="GZIP" | brotli="BROTLI" | lz4="LZ4" | uncompressed="UNCOMPRESSED" | none="NONE")。

  データの圧縮方法を指定します。データに標準のファイル拡張子が付いている場合、このオプションは一般的に不要です。指定できる値は "gzip" および "bzip" です。
- NumberTargetPartitions – UTF–8 文字列。

  AWS Glue を使用して Amazon S3 に書き込むときに Parquet ファイルのターゲットパーティション数を指定します。
- SchemaChangePolicy – DirectSchemaChangePolicy オブジェクト。

  クローラの更新の動作を指定するポリシー。

CatalogSchemaChangePolicy 構造

クローラの更新の動作を指定するポリシー。

フィールド

- EnableUpdateCatalog – ブール。

  クローラが変更されたスキーマを検出したとき、指定の更新動作を使用するかどうか。
- UpdateBehavior – UTF-8 文字列 (有効な値: UPDATE_IN_DATABASE | LOG)。

  クローラーが変更されたスキーマを検出したときの更新動作。

S3directTarget 構造

Amazon S3 に書き込むデータターゲットを指定します。

フィールド

- `Name` – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

  データターゲットの名前。

- `Inputs` – 必須: UTF-8 文字列の配列。1 個の文字列。

  データターゲットへの入力であるノード。

- `PartitionKeys` – UTF-8 文字列の配列。

  一連のキーを使用してネイティブパーティショニングを指定します。

- `Path` – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  書き込む単一の Amazon S3 パス。

- `Compression` – UTF-8 文字列。Custom string pattern #59 に一致。

  データの圧縮方法を指定します。データに標準のファイル拡張子が付いている場合、このオプションは一般的に不要です。指定できる値は `"gzip"` および `"bzip"` です。

- `NumberTargetPartitions` – UTF–8 文字列。

  Amazon S3 に直接データを書き込むときのターゲットパーティション数を指定します。

- `Format` – 必須: UTF-8 文字列 (有効な値: `json="JSON"` | `csv="CSV"` | `avro="AVRO"` | `orc="ORC"` | `parquet="PARQUET"` | `hudi="HUDI"` | `delta="DELTA"` | `iceberg="ICEBERG"` | `hyper="HYPER"` | `xml="XML"`)。

  ターゲットのデータ出力形式を指定します。

- `SchemaChangePolicy` – DirectSchemaChangePolicy オブジェクト。

  クローラの更新の動作を指定するポリシー。

S3HudiCatalogTarget 構造

AWS Glue データカタログ内の Hudi データソースに書き込むターゲットを指定します。

フィールド

- Name – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

  データターゲットの名前。
- Inputs – 必須: UTF-8 文字列の配列。1 個の文字列。

  データターゲットへの入力であるノード。
- PartitionKeys – UTF-8 文字列の配列。

  一連のキーを使用してネイティブパーティショニングを指定します。
- Table – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  書き込むデータベーステーブルの名前。
- Database – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  書き込むデータベースの名前。
- AdditionalOptions – 必須: キーバリューペアのマップ配列。

  各キーは、Custom string pattern #59 に適合する UTF-8 文字列です。

  各値は、Custom string pattern #59 に適合する UTF-8 文字列です。

  コネクタの追加接続オプションを指定します。
- SchemaChangePolicy – CatalogSchemaChangePolicy オブジェクト。

  クローラの更新の動作を指定するポリシー。

S3HudiDirectTarget 構造

Amazon S3 内の Hudi データソースに書き込むターゲットを指定します。

フィールド

- Name – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

  データターゲットの名前。
- Inputs – 必須: UTF-8 文字列の配列。1 個の文字列。

  データターゲットへの入力であるノード。

- Path – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  書き込み先の Hudi データソースの Amazon S3 パス。

- Compression – 必須: UTF-8 文字列 (有効な値: gzip="GZIP" | lzo="LZO" | uncompressed="UNCOMPRESSED" | snappy="SNAPPY")。

  データの圧縮方法を指定します。データに標準のファイル拡張子が付いている場合、このオプションは一般的に不要です。指定できる値は "gzip" および "bzip" です。

- NumberTargetPartitions – UTF–8 文字列。

  Amazon S3 全体で Hudi データセットファイルを配布するためのターゲットパーティション数を指定します。

- PartitionKeys – UTF-8 文字列の配列。

  一連のキーを使用してネイティブパーティショニングを指定します。

- Format – 必須: UTF-8 文字列 (有効な値: json="JSON" | csv="CSV" | avro="AVRO" | orc="ORC" | parquet="PARQUET" | hudi="HUDI" | delta="DELTA" | iceberg="ICEBERG" | hyper="HYPER" | xml="XML")。

  ターゲットのデータ出力形式を指定します。

- AdditionalOptions – 必須: キーバリューペアのマップ配列。

  各キーは、Custom string pattern #59 に適合する UTF-8 文字列です。

  各値は、Custom string pattern #59 に適合する UTF-8 文字列です。

  コネクタの追加接続オプションを指定します。

- SchemaChangePolicy – DirectSchemaChangePolicy オブジェクト。

  クローラの更新の動作を指定するポリシー。

S3DeltaCatalogTarget 構造

AWS Glue データカタログ内の Delta Lake データソースに書き込むターゲットを指定します。

フィールド

- Name – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

データターゲットの名前。

- `Inputs` – 必須: UTF-8 文字列の配列。1 個の文字列。

  データターゲットへの入力であるノード。

- `PartitionKeys` – UTF-8 文字列の配列。

  一連のキーを使用してネイティブパーティショニングを指定します。

- `Table` – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  書き込むデータベーステーブルの名前。

- `Database` – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  書き込むデータベースの名前。

- `AdditionalOptions` – キーバリューペアのマップ配列。

  各キーは、Custom string pattern #59 に適合する UTF-8 文字列です。

  各値は、Custom string pattern #59 に適合する UTF-8 文字列です。

  コネクタの追加接続オプションを指定します。

- `SchemaChangePolicy` – CatalogSchemaChangePolicy オブジェクト。

  クローラの更新の動作を指定するポリシー。

S3DeltaDirectTarget 構造

Amazon S3 内の Delta Lake データソースに書き込むターゲットを指定します。

フィールド

- `Name` – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

  データターゲットの名前。

- `Inputs` – 必須: UTF-8 文字列の配列。1 個の文字列。

  データターゲットへの入力であるノード。

- `PartitionKeys` – UTF-8 文字列の配列。

一連のキーを使用してネイティブパーティショニングを指定します。

- Path – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  書き込み先の Delta Lake データソースの Amazon S3 パス。

- Compression – 必須: UTF-8 文字列 (有効な値: uncompressed="UNCOMPRESSED" | snappy="SNAPPY")。

  データの圧縮方法を指定します。データに標準のファイル拡張子が付いている場合、このオプションは一般的に不要です。指定できる値は "gzip" および "bzip" です。

- NumberTargetPartitions – UTF–8 文字列。

  Amazon S3 全体に Delta Lake データセットファイルを配布するターゲットパーティション数を指定します。

- Format – 必須: UTF-8 文字列 (有効な値: json="JSON" | csv="CSV" | avro="AVRO" | orc="ORC" | parquet="PARQUET" | hudi="HUDI" | delta="DELTA" | iceberg="ICEBERG" | hyper="HYPER" | xml="XML")。

  ターゲットのデータ出力形式を指定します。

- AdditionalOptions – キーバリューペアのマップ配列。

  各キーは、Custom string pattern #59 に適合する UTF-8 文字列です。

  各値は、Custom string pattern #59 に適合する UTF-8 文字列です。

  コネクタの追加接続オプションを指定します。

- SchemaChangePolicy – DirectSchemaChangePolicy オブジェクト。

  クローラの更新の動作を指定するポリシー。

S3HyperDirectTarget 構造

Amazon S3 に書き込む HyperDirect データターゲットを指定します。

フィールド

- Name – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

  HyperDirect ターゲットノードの一意の識別子。

- `Inputs` – 必須: UTF-8 文字列の配列。1 個の文字列。

  HyperDirect ターゲットの入力ソースを指定します。
- `PartitionKeys` – UTF-8 文字列の配列。

  出力データのパーティショニング戦略を定義します。
- `Path` – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  出力データが書き込まれる S3 露kーション。
- `Compression` – UTF-8 文字列 (有効な値: `uncompressed="UNCOMPRESSED"`)。

  出力データに適用する圧縮タイプ。
- `SchemaChangePolicy` – DirectSchemaChangePolicy オブジェクト。

  書き込みオペレーション中にスキーマの変更が処理される方法を定義します。

S3IcebergDirectTarget 構造

Amazon S3 の Iceberg データソースに書き込むターゲットを指定します。

フィールド

- `Name` – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

  データパイプラインの Iceberg ターゲットノードに一意の識別子を指定します。
- `Inputs` – 必須: UTF-8 文字列の配列。1 個の文字列。

  この Iceberg ターゲットにデータを提供する 1 つの入力ソースを定義します。
- `PartitionKeys` – UTF-8 文字列の配列。

  S3 の Iceberg テーブルデータをパーティショニングするために使用される列を指定します。
- `Path` – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  Iceberg テーブルデータが保存される S3 ロケーションを定義します。
- `Format` – 必須: UTF-8 文字列 (有効な値: `json="JSON"` | `csv="CSV"` | `avro="AVRO"` | `orc="ORC"` | `parquet="PARQUET"` | `hudi="HUDI"` | `delta="DELTA"` | `iceberg="ICEBERG"` | `hyper="HYPER"` | `xml="XML"`)。

  Iceberg テーブルデータの保存に使用されるファイル形式を指定します (Parquet や ORC など)。

- AdditionalOptions – キーバリューペアのマップ配列。

  各キーは、Custom string pattern #59 に適合する UTF-8 文字列です。

  各値は、Custom string pattern #59 に適合する UTF-8 文字列です。

  Iceberg テーブルの動作をカスタマイズするための追加の設定オプションを提供します。

- SchemaChangePolicy – DirectSchemaChangePolicy オブジェクト。

  Iceberg テーブルにデータを書き込むときにスキーマ変更を処理する方法を定義します。

- Compression – 必須: UTF-8 文字列 (有効な値: gzip="GZIP" | lzo="LZO" | uncompressed="UNCOMPRESSED" | snappy="SNAPPY")。

  S3 の Iceberg テーブルファイルに使用される圧縮コーデックを指定します。

- NumberTargetPartitions – UTF–8 文字列。

  S3 全体 に Iceberg テーブルファイルを配布するターゲットパーティション数を設定します。

DirectSchemaChangePolicy 構造

クローラの更新の動作を指定するポリシー。

フィールド

- EnableUpdateCatalog – ブール。

  クローラが変更されたスキーマを検出したとき、指定の更新動作を使用するかどうか。

- UpdateBehavior – UTF-8 文字列 (有効な値: UPDATE_IN_DATABASE | LOG)。

  クローラーが変更されたスキーマを検出したときの更新動作。

- Table – UTF-8 文字列。Custom string pattern #59 に一致。

  スキーマ変更ポリシーが適用されるデータベース内のテーブルを指定します。

- Database– UTF-8 文字列、「Custom string pattern #59」に一致。

  スキーマ変更ポリシーを適用するデータベースを指定します。

ApplyMapping 構造

データソースのマップデータプロパティキーを、データターゲットのデータプロパティキーに変換指定します。キーの名前を変更したり、データタイプを変更したり、データセットから削除するキーを選択できます。

フィールド

- `Name` – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

  変換ノードの名前。
- `Inputs` – 必須: UTF-8 文字列の配列。1 個の文字列。

  ノード名で識別されるデータ入力。
- `Mapping` – 必須: Mapping オブジェクトの配列。

  データソースのデータプロパティキーを、データターゲットのデータプロパティキーにマッピングします。

Mapping 構造

データプロパティキーのマッピングを指定します。

フィールド

- `ToKey` – UTF-8 文字列。Custom string pattern #59 に一致。

  マッピングを適用した後に、列名を何にするのかを示します。`FromPath` と同じでも構いません。
- `FromPath` – UTF-8 文字列の配列。

  変更するテーブルまたは列。
- `FromType` – UTF-8 文字列。Custom string pattern #59 に一致。

  修正されるデータのタイプ。
- `ToType` – UTF-8 文字列。Custom string pattern #59 に一致。

  データの修正先のデータタイプ。
- `Dropped` – ブール。

true の場合、列は削除されます。

- Children – Mapping オブジェクトの配列。

  ネストされたデータ構造にのみ適用されます。親構造を変更し、その子構造を変更する場合は、このデータ構造に記入できます。それはまた Mapping であり、FromPath はこの構造から親の FromPath プラス FromPath です。

  子部分に、次のような構造があるとします。

  ```
  { "FromPath": "OuterStructure", "ToKey": "OuterStructure", "ToType":
  "Struct", "Dropped": false, "Chidlren": [{ "FromPath": "inner", "ToKey":
  "inner", "ToType": "Double", "Dropped": false, }] }
  ```

  次のような Mapping を指定します。

  ```
  { "FromPath": "OuterStructure", "ToKey": "OuterStructure", "ToType":
  "Struct", "Dropped": false, "Chidlren": [{ "FromPath": "inner", "ToKey":
  "inner", "ToType": "Double", "Dropped": false, }] }
  ```

SelectFields 構造

保持するデータプロパティキーを選択するトランスフォームを指定します。

フィールド

- Name – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

  変換ノードの名前。

- Inputs – 必須: UTF-8 文字列の配列。1 個の文字列。

  ノード名で識別されるデータ入力。

- Paths – 必須: UTF-8 文字列の配列。

  データ構造内の変数への JSON パス。

DropFields 構造

削除するデータプロパティキーを選択する変換を指定します。

フィールド

- Name – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

  変換ノードの名前。
- Inputs – 必須: UTF-8 文字列の配列。1 個の文字列。

  ノード名で識別されるデータ入力。
- Paths – 必須: UTF-8 文字列の配列。

  データ構造内の変数への JSON パス。

RenameField 構造

1 つのデータプロパティキーの名前を変更する変換を指定します。

フィールド

- Name – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

  変換ノードの名前。
- Inputs – 必須: UTF-8 文字列の配列。1 個の文字列。

  ノード名で識別されるデータ入力。
- SourcePath – 必須: UTF-8 文字列の配列。

  ソースデータのデータ構造内の変数への JSON パス。
- TargetPath – 必須: UTF-8 文字列の配列。

  ターゲットデータのデータ構造内の変数への JSON パス。

スピゴット構造

Amazon S3 バケットにデータのサンプルを書き込むための変換を指定します。

フィールド

- Name – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

  変換ノードの名前。

- `Inputs` – 必須: UTF-8 文字列の配列。1 個の文字列。

  ノード名で識別されるデータ入力。

- `Path` – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  Amazon S3 パスでは、データセットから Amazon S3 バケットの JSON ファイルにレコードのサブセットが書き出されます。

- `Topk` – 数値 (integer)。100 以下。

  データセットの先頭から書き込むレコードの数を指定します。

- `Prob` – 数値 (double)。1 以下。

  特定のレコードを選ぶ確率 (最大値が 1 の 10 進値)。値 1 は、データセットから読み込まれた各行をサンプル出力に含めることを示します。

Join 構造

指定したデータプロパティキーの比較フレーズを使用して、2 つのデータセットを 1 つに結合する変換を指定します。結合タイプは、内部結合、外部結合、左結合、右結合、左半結合、左反結合を使用できます。

フィールド

- `Name` – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

  変換ノードの名前。

- `Inputs` – 必須: UTF-8 文字列の配列。2 個の文字列。

  ノード名で識別されるデータ入力。

- `JoinType` – 必須: UTF-8 文字列 (有効な値: `equijoin="EQUIJOIN"` | `left="LEFT"` | `right="RIGHT"` | `outer="OUTER"` | `leftsemi="LEFT_SEMI"` | `leftanti="LEFT_ANTI"`)。

  データセットで実行する結合の種類を指定します。

- `Columns` – 必須: JoinColumn オブジェクトの配列。2 個の構造。

  結合する 2 つの列のリスト。

JoinColumn 構造

結合する列を指定します。

フィールド

- From – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  結合する列。
- Keys – 必須: UTF-8 文字列の配列。

  結合される列のキー。

SplitFields 構造

データプロパティキーを 2 つの DynamicFrames に分割する変換を指定します。出力は DynamicFrames のコレクションです。一方は選択したデータプロパティキー、他方は残っている方のデータプロパティキーを持ちます。

フィールド

- Name – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

  変換ノードの名前。
- Inputs – 必須: UTF-8 文字列の配列。1 個の文字列。

  ノード名で識別されるデータ入力。
- Paths – 必須: UTF-8 文字列の配列。

  データ構造内の変数への JSON パス。

SelectFromCollection 構造

DynamicFrame のコレクションから 1 つの DynamicFrames を選択するトランスフォームを指定します 出力は選択された DynamicFrame です。

フィールド

- Name – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

変換ノードの名前。

- `Inputs` – 必須: UTF-8 文字列の配列。1 個の文字列。

  ノード名で識別されるデータ入力。

- `Index` – 必須: 数値 (整数)。None 以下。

  選択される DynamicFrame のインデックス。

FillMissingValues 構造

変換を使用して、データセット内に欠落値があるレコードを検索し、補完により決定する値を持つ新しいフィールドを追加します。入力データセットは、欠落値を決定する機械学習モデルのトレーニングに使用されます。

フィールド

- `Name` – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

  変換ノードの名前。

- `Inputs` – 必須: UTF-8 文字列の配列。1 個の文字列。

  ノード名で識別されるデータ入力。

- `ImputedPath` – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  帰属するデータセットのデータ構造内の変数への JSON パス。

- `FilledPath` – UTF-8 文字列。Custom string pattern #59 に一致。

  データセットのデータ構造内の変数への JSON パスを入力します。

Filter 構造

フィルター条件に基づいて、データセットを 2 つに分割する変換を指定します。

フィールド

- `Name` – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

  変換ノードの名前。

- `Inputs` – 必須: UTF-8 文字列の配列。1 個の文字列。

  ノード名で識別されるデータ入力。

- `LogicalOperator` – 必須: UTF-8 文字列 (有効な値: `AND` | `OR`)。

  キー値を指定値と比較して行をフィルタリングするために使用される演算子。

- `Filters` – 必須: FilterExpression オブジェクトの配列。

  フィルタ式を指定します。

FilterExpression 構造

フィルタ表現を指定します。

フィールド

- `Operation` – 必須: UTF-8 文字列 (有効な値: `EQ` | `LT` | `GT` | `LTE` | `GTE` | `REGEX` | `ISNULL`)。

  表現で実行するオペレーションの種類。

- `Negated` – ブール。

  その表現を無効にするかどうか。

- `Values` – 必須: FilterValue オブジェクトの配列。

  フィルタ値のリスト。

FilterValue 構造

`FilterExpression` の値リストにある単一のエントリを表します。

フィールド

- `Type` – 必須: UTF-8 文字列 (有効な値: `COLUMNEXTRACTED` | `CONSTANT`)。

  フィルタ値のタイプ。

- `Value` – 必須: UTF-8 文字列の配列。

  関連させる値。

CustomCode 構造

データ変換を実行するためにカスタムコードを使用する変換を指定します。結果には、DynamicFrames のコレクションが出力されます。

フィールド

- Name – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

  変換ノードの名前。

- Inputs – 必須: UTF-8 文字列の配列。1 個の以上の文字列。

  ノード名で識別されるデータ入力。

- Code – 必須: UTF-8 文字列。Custom string pattern #52 に一致。

  データ変換を実行するために使用されるカスタムコード。

- ClassName – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  カスタムコードノードクラスに定義された名前。

- OutputSchemas – GlueSchema オブジェクトの配列。

  カスタムコード変換用のデータスキーマを指定します。

SparkSQL 構造

データを変換する Spark SQL 構文を使用して、SQL クエリを入力する変換を指定します。出力は、単一の DynamicFrame です。

フィールド

- Name – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

  変換ノードの名前。

- Inputs – 必須: UTF-8 文字列の配列。1 個の以上の文字列。

  ノード名で識別されるデータ入力。SQL クエリで使用する各入力ノードにテーブル名を関連付けることができます。選択する名前は、Spark SQL の規則を満たす必要があります。

- SqlQuery – 必須: UTF-8 文字列。Custom string pattern #60 に一致。

Spark SQL 構文を使用し、単一のデータセットを返す SQL クエリ。

- `SqlAliases` – 必須: SqlAlias オブジェクトの配列。

  エイリアスのリスト。エイリアスを使用すると、特定の入力に対して SQL で使用する名前を指定できます。たとえば、「myDataSource」という名前のデータソースがあるとします。`From` を MyDataSourceとして、`Alias` を sqlNameとして指定する場合、SQLで次のことができます。

  ```
  select * from SqlName
  ```

  これにより、MyDataSource からデータを取得します。

- `OutputSchemas` – GlueSchema オブジェクトの配列。

  SparkSQL 変換用のデータスキーマを指定します。

SqlAlias 構造

`SqlAliases` の値リストにある単一のエントリを表します。

フィールド

- `From` – 必須: UTF-8 文字列。Custom string pattern #58 に一致。

  テーブルまたはテーブル内の列。

- `Alias` – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  テーブルまたはテーブル内の列に与えられた一時的な名前。

DropNullFields 構造

列のすべての値が 'null' である場合に、データセットから列を削除する変換を指定します。デフォルトでは、AWS Glue Studio は null オブジェクトを認識しますが、空の文字列、『null』が 1 の整数、または 0 などのプレースホルダなどの文字列では、自動的に『null』として認識されません。

フィールド

- `Name` – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

  変換ノードの名前。

- `Inputs` – 必須: UTF-8 文字列の配列。1 個の文字列。

  ノード名で識別されるデータ入力。

- `NullCheckBoxList` – NullCheckboxList オブジェクト。

  特定の値が削除のために Null 値として認識されるかどうかを表す構造。

- `NullTextList` – NullValueField オブジェクトの配列。構造 50 個以下。

  データセットに特有のNullプレースホルダとして使用される 0 や他の値などはカスタムの NULL 値を表す NullValueField 構造のリストを指定する構造。

  Null プレースホルダの値とデータタイプの両方がデータと一致する場合にのみ、`DropNullFields` 変換でカスタム NULL 値を削除します。

NullCheckboxList 構造

特定の値が削除の Null 値として認識されるかどうかを表します。

フィールド

- `IsEmpty` – ブール。

  空の文字列を Null 値と見なすことを指定します。

- `IsNullString` – ブール。

  「null」の単語を綴る値をnull値と見なすことを指定します。

- `IsNegOne` – ブール。

  -1 の整数値が Null 値と見なすことを指定します。

NullValueField 構造

データセットに固有の null プレースホルダとして使用される 0 や他の値などのカスタムの null 値を表します。

フィールド

- `Value` – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

null プレースホルダの値。

- `Datatype` – 必須: Datatype オブジェクト。

  値のデータタイプ。

Datatype 構造

値のデータタイプを表す構造。

フィールド

- `Id` – 必須: UTF-8 文字列。Custom string pattern #58 に一致。

  値のデータタイプ。

- `Label` – 必須: UTF-8 文字列。Custom string pattern #58 に一致。

  データタイプに割り当てられたラベル。

Merge 構造

レコードを識別するために、DynamicFrame プライマリキーに基づく DynamicFrame ステージングに結合変換を指定します。重複レコード (同じプライマリキーを持つレコード) は重複除外されません。

フィールド

- `Name` – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

  変換ノードの名前。

- `Inputs` – 必須: UTF-8 文字列の配列。2 個の文字列。

  ノード名で識別されるデータ入力。

- `Source` – 必須: UTF-8 文字列。Custom string pattern #58 に一致。

  DynamicFrame ステージングと結合する DynamicFrame ソース。

- `PrimaryKeys` – 必須: UTF-8 文字列の配列。

ソースおよびステージング動的フレームからのレコードを照合するプライマリキーフィールドのリスト。

Union 構造

2 つ以上のデータセットの行を 1 つの結果に結合する変換を指定します。

フィールド

- Name – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

  変換ノードの名前。
- Inputs – 必須: UTF-8 文字列の配列。2 個の文字列。

  変換用のノード ID 入力。
- UnionType – 必須: UTF-8 文字列 (有効な値: ALL | DISTINCT)。

  Union 変換のタイプを示します。

  データソースの全行を結果として生じる DynamicFrame に結合するため ALL を指定します。結果として生じるユニオンでは、重複する行は削除されません。

  結果として生じる DynamicFrame 内の重複した行を削除するため DISTINCT を指定します。

PIIDetection 構造

PII データを識別、削除、またはマスクする変換を指定します。

フィールド

- Name – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

  変換ノードの名前。
- Inputs – 必須: UTF-8 文字列の配列。1 個の文字列。

  変換用のノード ID 入力。
- PiiType – 必須: UTF-8 文字列 (有効な値: RowAudit | RowMasking | ColumnAudit | ColumnMasking)。

PIIDetection 変換のタイプを示します。

- `EntityTypesToDetect` – 必須: UTF-8 文字列の配列。

  PIIDetection 変換が PII データとして識別するエンティティのタイプを示します。

  PII タイプのエンティティには以下が含まれます。PERSON_NAME、DATE、USA_SNN、EMAIL、USA_ITIN、USA_PASSPORT_NUMBER、PHONE_
- `OutputColumnName` – UTF-8 文字列。Custom string pattern #59 に一致。

  その行で検出されたエンティティタイプを含む、すべての出力列名を示します。
- `SampleFraction` – 数値 (double)。1 以下。

  PII エンティティのスキャン時にサンプリングするデータの割合を示します。
- `ThresholdFraction` – 数値 (double)。1 以下。

  PII データとして識別されるために、列内で適合する必要があるデータの割合を示します。
- `MaskValue` – UTF-8 文字列、256 バイト長以下。Custom string pattern #56 に一致。

  検出されたエンティティを置き換える値を示します。

Aggregate 構造

選択したフィールドによって行をグループ化し、指定された関数を使用して集計値を計算する変換を指定します。

フィールド

- `Name` – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

  変換ノードの名前。
- `Inputs` – 必須: UTF-8 文字列の配列。1 個の文字列。

  集約変換の入力として使用するフィールドと行を指定します。
- `Groups` – 必須: UTF-8 文字列の配列。

  グループ化に使用するフィールドを指定します。
- `Aggs` – 必須: AggregateOperation オブジェクトの配列、1 ~ 30 個の構造。

指定したフィールドで実行する集計関数を指定します。

DropDuplicates 構造

繰り返しデータの行をデータセットから削除する変換を指定します。

フィールド

- `Name` – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

  変換ノードの名前。
- `Inputs` – 必須: UTF-8 文字列の配列。1 個の文字列。

  ノード名で識別されるデータ入力。
- `Columns` – UTF-8 文字列の配列。

  繰り返しがある場合に、それをマージまたは削除する列の名前。

GovernedCatalogTarget 構造

AWS Glue データカタログを使用して Amazon S3 に書き込むデータターゲットを指定します。

フィールド

- `Name` – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

  データターゲットの名前。
- `Inputs` – 必須: UTF-8 文字列の配列。1 個の文字列。

  データターゲットへの入力であるノード。
- `PartitionKeys` – UTF-8 文字列の配列。

  一連のキーを使用してネイティブパーティショニングを指定します。
- `Table` – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  書き込むデータベーステーブルの名前。
- `Database` – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

書き込むデータベースの名前。

- SchemaChangePolicy – CatalogSchemaChangePolicy オブジェクト。

  管理されたカタログを更新する際の動作を指定するポリシー。

GovernedCatalogSource 構造

管理された AWS Glue データカタログ内のデータストアを指定します。

フィールド

- Name – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

  データストアの名前。

- Database – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  読み込むデータベース。

- Table – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  読み取り元のデータベーステーブル。

- PartitionPredicate – UTF-8 文字列。Custom string pattern #59 に一致。

  この述語を満たすパーティションは削除されます。これらのパーティションの保存期間内のファイルは削除されません。"" を設定 – デフォルトでは空です。

- AdditionalOptions – S3SourceAdditionalOptions オブジェクト。

  追加の接続オプションを指定します。

AggregateOperation 構造

集約変換で集約を実行するために必要なパラメータのセットを指定します。

フィールド

- Column – 必須: UTF-8 文字列の配列。

  集計関数を適用するデータセットの列を指定します。

- `AggFunc` – 必須: UTF-8 文字列 (有効な値: `avg` | `countDistinct` | `count` | `first` | `last` | `kurtosis` | `max` | `min` | `skewness` | `stddev_samp` | `stddev_pop` | `sum` | `sumDistinct` | `var_samp` | `var_pop`)。

  適用する集計関数を指定します。

  使用可能な集計関数には、avg
  countDistinct、count、first、last、kurtosis、max、min、skewness、stddev_samp、stddev_pop、sum、s
  などがあります。

GlueSchema 構造

AWS Glue でスキーマを決定できない場合に、ユーザー定義のスキーマを指定します。

フィールド

- `Columns` – GlueStudioSchemaColumn オブジェクトの配列。

  AWS Glue スキーマを作成する列の定義を指定します。

GlueStudioSchemaColumn 構造

AWS Glue スキーマ定義内の列を 1 つ指定します。

フィールド

- `Name` – 必須: バイト長が 1024 以下で Single-line string pattern に適合する、UTF-8 文字列。

  AWS Glue Studio スキーマ内にある列の名前。
- `Type` – UTF-8 文字列。131,072 バイト長以下。Single-line string pattern に一致。

  AWS Glue Studio スキーマ内にあるこの列のハイブタイプ。

GlueStudioColumn 構造

AWS Glue Studio 内の列を 1 つ指定します。

フィールド

- `Key` – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

AWS Glue Studio 内の列のキー。

- `FullPath` – 必須: UTF-8 文字列の配列。

  AWS Glue Studio 内の列の完全な URL。

- `Type` – 必須: UTF-8 文字列 (有効な値: `array="ARRAY"` | `bigint="BIGINT"` | `bigint array="BIGINT_ARRAY"` | `binary="BINARY"` | `binary array="BINARY_ARRAY"` | `boolean="BOOLEAN"` | `boolean array="BOOLEAN_ARRAY"` | `byte="BYTE"` | `byte array="BYTE_ARRAY"` | `char="CHAR"` | `char array="CHAR_ARRAY"` | `choice="CHOICE"` | `choice array="CHOICE_ARRAY"` | `date="DATE"` | `date array="DATE_ARRAY"` | `decimal="DECIMAL"` | `decimal array="DECIMAL_ARRAY"` | `double="DOUBLE"` | `double array="DOUBLE_ARRAY"` | `enum="ENUM"` | `enum array="ENUM_ARRAY"` | `float="FLOAT"` | `float array="FLOAT_ARRAY"` | `int="INT"` | `int array="INT_ARRAY"` | `interval="INTERVAL"` | `interval array="INTERVAL_ARRAY"` | `long="LONG"` | `long array="LONG_ARRAY"` | `object="OBJECT"` | `short="SHORT"` | `short array="SHORT_ARRAY"` | `smallint="SMALLINT"` | `smallint array="SMALLINT_ARRAY"` | `string="STRING"` | `string array="STRING_ARRAY"` | `timestamp="TIMESTAMP"` | `timestamp array="TIMESTAMP_ARRAY"` | `tinyint="TINYINT"` | `tinyint array="TINYINT_ARRAY"` | `varchar="VARCHAR"` | `varchar array="VARCHAR_ARRAY"` | `null="NULL"` | `unknown="UNKNOWN"` | `unknown array="UNKNOWN_ARRAY"`)。

  AWS Glue Studio 内の列のタイプ。

- `Children` – 構造の配列。

  AWS Glue Studio 内の親列の子。

DynamicTransform 構造

動的変換を実行するために必要なパラメータのセットを指定します。

フィールド

- `Name` – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  動的変換の名前を指定します。

- `TransformName` – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  AWS Glue Studio ビジュアルエディタに表示される動的変換の名前を指定します。

- `Inputs` – 必須: UTF-8 文字列の配列。1 個の文字列。

  必要な動的変換の入力を指定します。
- `Parameters` – TransformConfigParameter オブジェクトの配列。

  動的変換のパラメータを指定します。
- `FunctionName` – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  動的変換の関数の名前を指定します。
- `Path` – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  動的変換ソースファイルおよび設定ファイルのパスを指定します。
- `Version` – UTF-8 文字列。Custom string pattern #59 に一致。

  このフィールドは使用されず、将来のリリースで非推奨となります。
- `OutputSchemas` – GlueSchema オブジェクトの配列。

  動的変換用のデータスキーマを指定します。

TransformConfigParameter 構造

動的変換の設定ファイル内のパラメータを指定します。

フィールド

- `Name` – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  動的変換の設定ファイル内のパラメータの名前を指定します。
- `Type` – 必須: UTF-8 文字列 (有効な値: `str="STR"` | `int="INT"` | `float="FLOAT"` | `complex="COMPLEX"` | `bool="BOOL"` | `list="LIST"` | `null="NULL"`)。

  動的変換の設定ファイル内のパラメータタイプを指定します。
- `ValidationRule` – UTF-8 文字列。Custom string pattern #59 に一致。

  動的変換の設定ファイル内の検証ルールを指定します。
- `ValidationMessage` – UTF-8 文字列。Custom string pattern #59 に一致。

  動的変換の設定ファイル内の検証メッセージを指定します。
- `Value` – UTF-8 文字列の配列。

動的変換の設定ファイル内のパラメータの値を指定します。

- `ListType` – UTF-8 文字列 (有効な値: `str="STR"` | `int="INT"` | `float="FLOAT"` | `complex="COMPLEX"` | `bool="BOOL"` | `list="LIST"` | `null="NULL"`)。

  動的変換の設定ファイル内のパラメータのリスト型を指定します。

- `IsOptional` – ブール。

  動的変換の設定ファイル内のパラメータがオプションかどうかを指定します。

EvaluateDataQuality 構造

データ品質評価基準を指定します。

フィールド

- `Name` – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

  データ品質評価の名前。

- `Inputs` – 必須: UTF-8 文字列の配列。1 個の文字列。

  データ品質評価の入力。

- `Ruleset` – 必須: UTF-8 文字列、1 ~ 65536 バイト長、Custom string pattern #57 に一致。

  データ品質評価のルールセット。

- `Output` – UTF-8 文字列 (有効な値: `PrimaryInput` | `EvaluationResults`)。

  データ品質評価の出力。

- `PublishingOptions` – DQResultsPublishingOptions オブジェクト。

  結果の発行方法を設定するオプション。

- `StopJobOnFailureOptions` – DQStopJobOnFailureOptions オブジェクト。

  データ品質評価に失敗した場合にジョブを停止する方法を設定するオプション。

DQResultsPublishingOptions 構造

データ品質評価の結果の発行方法を設定するオプション。

フィールド

- EvaluationContext – UTF-8 文字列。Custom string pattern #58 に一致。

  評価のコンテキスト。
- ResultsS3Prefix – UTF-8 文字列。Custom string pattern #59 に一致。

  結果に追加された Amazon S3 プレフィックス。
- CloudWatchMetricsEnabled – ブール。

  データ品質結果のメトリクスを有効にします。
- ResultsPublishingEnabled – ブール。

  データ品質結果の発行を有効にします。

DQStopJobOnFailureOptions 構造

データ品質評価に失敗した場合にジョブを停止する方法を設定するオプション。

フィールド

- StopJobOnFailureTiming – UTF-8 文字列 (有効な値: Immediate | AfterDataLoad)。

  データ品質評価が失敗した場合にジョブを停止するタイミング。オプションは Immediate または AfterDataLoad です。

EvaluateDataQualityMultiFrame 構造

データ品質評価基準を指定します。

フィールド

- Name – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

  データ品質評価の名前。
- Inputs – 必須: UTF-8 文字列の配列。1 個の以上の文字列。

  データ品質評価の入力。このリストにおける最初の入力はプライマリデータソースです。
- AdditionalDataSources – キーバリューペアのマップ配列。

各キーは、Custom string pattern #61 に適合する UTF-8 文字列です。

各値は、Custom string pattern #59 に適合する UTF-8 文字列です。

プライマリを除くすべてのデータソースのエイリアス。

- `Ruleset` – 必須: UTF-8 文字列、1 ~ 65536 バイト長、Custom string pattern #57 に一致。

  データ品質評価のルールセット。

- `PublishingOptions` – DQResultsPublishingOptions オブジェクト。

  結果の発行方法を設定するオプション。

- `AdditionalOptions` – キーバリューペアのマップ配列。

  各キーは UTF-8 文字列 (有効な値: `performanceTuning.caching="CacheOption"` | `observations.scope="ObservationsOption"`)。

  各値は UTF-8 文字列。

  変換のランタイム動作を設定するオプション。

- `StopJobOnFailureOptions` – DQStopJobOnFailureOptions オブジェクト。

  データ品質評価に失敗した場合にジョブを停止する方法を設定するオプション。

Recipe 構造

AWS Glue ジョブで AWS Glue DataBrew レシピを使用する AWS Glue Studio ノード。

フィールド

- `Name` – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

  AWS Glue Studio ノードの名前。

- `Inputs` – 必須: UTF-8 文字列の配列。1 個の文字列。

  レシピノードへの入力となるノード。これは ID によって識別されます。

- `RecipeReference` – RecipeReference オブジェクト。

  ノードが使用する DataBrew レシピへの参照。

- `RecipeSteps` – RecipeStep オブジェクトの配列。

レシピノードで使用される変換ステップ。

RecipeReference 構造

AWS Glue DataBrew レシピへの参照。

フィールド

- `RecipeArn` – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  DataBrew レシピの ARN。
- `RecipeVersion` – 必須: UTF-8 文字列、1～16 バイト長。

  DataBrew レシピの RecipeVersion。

SnowflakeNodeData 構造

AWS Glue Studio で Snowflake ノードの設定を指定します。

フィールド

- `SourceType` – UTF-8 文字列。Custom string pattern #58 に一致。

  取得したデータの指定方法を指定します。有効な値: `"table"`、 `"query"`。
- `Connection` – オプション オブジェクト。

  Snowflake エンドポイントへの AWS Glue データカタログ接続を指定します。
- `Schema` – UTF–8 文字列。

  ノードが使用する Snowflake データベーススキーマを指定します。
- `Table` – UTF–8 文字列。

  ノードが使用する Snowflake テーブルを指定します。
- `Database` – UTF–8 文字列。

  ノードが使用する Snowflake データベースを指定します。
- `TempDir` – UTF-8 文字列。Custom string pattern #59 に一致。

現在使用されていません。

- `IamRole` – オプション オブジェクト。

  現在使用されていません。

- `AdditionalOptions` – キーバリューペアのマップ配列。

  各キーは、Custom string pattern #59 に適合する UTF-8 文字列です。

  各値は、Custom string pattern #59 に適合する UTF-8 文字列です。

  Snowflake コネクタに渡される追加オプションを指定します。ノードの他の場所でオプションが指定されている場合、こちらが優先されます。

- `SampleQuery` – UTF–8 文字列。

  `query` SourceType でデータを取得するために使用する SQL 文字列。

- `PreAction` – UTF–8 文字列。

  Snowflake コネクタが標準アクションを実行する前に実行される SQL 文字列。

- `PostAction` – UTF–8 文字列。

  Snowflake コネクタが標準アクションを実行した後に実行される SQL 文字列。

- `Action` – UTF–8 文字列。

  既存のデータを持つテーブルに書き込むときに実行するアクションを指定します。有効な値は、`append`、`merge`、`truncate`、`drop` です。

- `Upsert` – ブール。

  アクションが `append` の場合に使用します。行が既に存在する場合の解決動作を指定します。true の場合、既存の行が更新されます。false の場合、それらの行が挿入されます。

- `MergeAction` – UTF-8 文字列。Custom string pattern #58 に一致。

  マージアクションを指定します。有効な値: `simple`、`custom`。simple の場合、マージ動作は `MergeWhenMatched` と `MergeWhenNotMatched` によって定義されます。custom の場合、`MergeClause` によって定義されます。

- `MergeWhenMatched` – UTF-8 文字列。Custom string pattern #58 に一致。

  マージ時に既存のデータと一致するレコードを解決する方法を指定します。有効な値: `update`、`delete`。

- MergeWhenNotMatched – UTF-8 文字列。Custom string pattern #58 に一致。

  マージ時に既存のデータと一致しないレコードを処理する方法を指定します。有効な値: insert、none。

- MergeClause – UTF–8 文字列。

  カスタムマージ動作を指定する SQL ステートメント。

- StagingTable – UTF–8 文字列。

  merge または upsert を行う append アクションを実行するときに使用されるステージングテーブルの名前。データはこのテーブルに書き込まれ、生成されたポストアクションによって table に移動されます。

- SelectedColumns – オプション オブジェクトの配列。

  マージや upsert の一致を検出するときに、レコードを識別するために組み合わせる列を指定します。value、label、 description キーを使用する構造のリストです。各構造は列を記述します。

- AutoPushdown – ブール。

  自動クエリプッシュダウンを有効にするかどうかを指定します。プッシュダウンが有効になっている場合、Spark でクエリを実行すると、クエリの一部が Snowflake サーバーに「プッシュダウン」できる場合にクエリがプッシュダウンされます。これにより、一部のクエリのパフォーマンスが向上します。

- TableSchema – オプション オブジェクトの配列。

  ノードのターゲットスキーマを手動で定義します。value、label、description キーを使用する構造のリストです。各構造は列を定義します。

SnowflakeSource 構造

Snowflake データソースを指定します。

フィールド

- Name – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

  Snowflake データソースの名前。

- Data – 必須: SnowflakeNodeData オブジェクト。

Snowflake データソースの設定。

- `OutputSchemas` – GlueSchema オブジェクトの配列。

  出力データのユーザー定義スキーマを指定します。

SnowflakeTarget 構造

Snowflake ターゲットを指定します。

フィールド

- `Name` – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

  Snowflake ターゲットの名前。

- `Data` – 必須: SnowflakeNodeData オブジェクト。

  Snowflake ターゲットノードのデータを指定します。

- `Inputs` – UTF-8 文字列の配列、1 個の文字列。

  データターゲットへの入力であるノード。

ConnectorDataSource の構造

標準の接続オプションを使用して生成されたソースを指定します。

フィールド

- `Name` – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

  このソースノードの名前。

- `ConnectionType` – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  基盤となる AWS Glue ライブラリに提供される `connectionType`。ノードタイプアは、次の接続タイプをサポートします。

  - `opensearch`
  - `azuresql`
  - `azurecosmos`

  - bigquery
  - saphana
  - teradata
  - vertica
- Data – 必須: キーバリューペアのマップ配列。

  各キーは UTF-8 文字列。

  各値は UTF-8 文字列。

  ノードの接続オプションを指定するマップ。AWS Glue ドキュメントの「接続パラメータ」セクションで、対応する接続タイプ用の標準接続オプションを確認できます。
- OutputSchemas – GlueSchema オブジェクトの配列。

  このソース用のデータスキーマを指定します。

ConnectorDataTarget の構造

標準の接続オプションを使用して生成されたターゲットを指定します。

フィールド

- Name – 必須: UTF-8 文字列。Custom string pattern #61 に一致。

  このターゲットノードの名前。
- ConnectionType – 必須: UTF-8 文字列。Custom string pattern #59 に一致。

  基盤となる AWS Glue ライブラリに提供される connectionType。ノードタイプアは、次の接続タイプをサポートします。
  - opensearch
  - azuresql
  - azurecosmos
  - bigquery
  - saphana
  - teradata
  - vertica

- `Data` – 必須: キーバリューペアのマップ配列。

  各キーは UTF-8 文字列。

  各値は UTF-8 文字列。

  ノードの接続オプションを指定するマップ。AWS Glue ドキュメントの「接続パラメータ」セクションで、対応する接続タイプ用の標準接続オプションを確認できます。
- `Inputs` – UTF-8 文字列の配列、1 個の文字列。

  データターゲットへの入力であるノード。

RecipeStep 構造

AWS Glue Studio データ準備レシピノードで使用されるレシピステップ。

フィールド

- `Action` – 必須: RecipeAction オブジェクト。

  レシピステップの変換アクション。
- `ConditionExpressions` – ConditionExpression オブジェクトの配列。

  レシピステップの条件式。

RecipeAction 構造

AWS Glue Studio データ準備レシピノードで定義されるアクション。

フィールド

- `Operation` – 必須: UTF-8 文字列、1 ~ 128 バイト長、「Custom string pattern #54」に一致。

  レシピアクションのオペレーション。
- `Parameters` – キーバリューペアのマップ配列。

  各キーは UTF-8 文字列、1 ~ 128 バイト長で、Custom string pattern #55 に一致します。

  各値は UTF-8 文字列で、1 ~ 32768 バイト長です。

レシピアクションのパラメータ。

ConditionExpression 構造

AWS Glue Studio データ準備レシピノードで定義される条件式。

フィールド

- `Condition` – 必須: UTF-8 文字列、1 ~ 128 バイト長、「Custom string pattern #54」に一致。

  条件式の条件。
- `Value` – UTF-8 文字列、1024 バイト長以下。

  条件式の値。
- `TargetColumn` – 必須: UTF-8 文字列、1 ~ 1024 バイト長。

  条件式のターゲット列。

ジョブ API

Jobs API では、ジョブのデータ型について説明しており、これには、AWS Glue でジョブ、ジョブ実行、およびトリガーを操作するための API が含まれます。

トピック

- ジョブ
- ジョブ実行
- トリガー

ジョブ

Jobs API では、AWS Glue でのジョブの作成、更新、削除、表示に関連するデータ型と API について説明します。

データ型

- Job 構造
- ExecutionProperty 構造

- NotificationProperty 構造
- JobCommand 構造
- ConnectionsList 構造
- JobUpdate 構造
- SourceControlDetails の構造

Job 構造

ジョブ定義を指定します。

フィールド

- Name – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  このジョブ定義に割り当てる名前。
- JobMode – UTF-8 文字列 (有効な値: SCRIPT="" | VISUAL="" | NOTEBOOK="")。

  ジョブの作成方法を説明するモード。次の値を指定できます:
  - SCRIPT - ジョブは AWS Glue Studio スクリプトエディタを使用して作成されました。
  - VISUAL - ジョブは AWS Glue Studio ビジュアルエディタを使用して作成されました。
  - NOTEBOOK - ジョブはインタラクティブセッションノートブックを使用して作成されました。

  JobMode フィールドが欠落しているか null の場合、SCRIPT がデフォルト値として割り当てられます。
- JobRunQueuingEnabled – ブール。

  このジョブのジョブ実行に対し、ジョブ実行キューイングが有効であるかどうかを指定します。

  true の値は、ジョブ実行キューイングがジョブ実行に対して有効であることを意味します。false または未入力の場合、ジョブ実行はキューイングの対象になりません。

  このフィールドがジョブ実行で設定された値と一致しない場合、ジョブ実行フィールドの値が使用されます。
- Description – 説明文字列。2048 バイト長以下。URI address multi-line string pattern に一致。

  ジョブの説明。
- LogUri – UTF–8 文字列。

このフィールドは、将来の利用のために予約されています。

- `Role` – UTF–8 文字列。

  このジョブに関連付けられている IAM ロールの名前または Amazon リソースネーム (ARN)。

- `CreatedOn` – タイムスタンプ。

  このジョブ定義を作成した日時。

- `LastModifiedOn` – タイムスタンプ。

  このジョブ定義を変更した最後の時点。

- `ExecutionProperty` – ExecutionProperty オブジェクト。

  このジョブに許可される同時実行の最大数を指定する `ExecutionProperty`。

- `Command` – JobCommand オブジェクト。

  このジョブを実行する `JobCommand`。

- `DefaultArguments` – キーバリューペアのマップ配列。

  各キーは UTF-8 文字列。

  各値は UTF-8 文字列。

  名前と値のペアとして指定された、このジョブを実行するごとのデフォルトの引数。

  独自のジョブ実行スクリプトが消費する引数だけでなく、AWS Glue が消費する引数もここで指定できます。

  ジョブ引数はログに記録される場合があります。プレーンテキストのシークレットを引数として渡さないでください。ジョブ内に保持する場合は、AWS Glue 接続、AWS Secrets Manager または他のシークレット管理メカニズムから取得します。

  独自のジョブ引数を指定および使用する方法については、デベロッパーガイドのトピック「Python で AWS Glue API を呼び出す」を参照してください。

  Spark ジョブの設定時にこのフィールドに指定できる引数については、デベロッパーガイドのトピック「AWS Glueで使用するスペシャルパラメータ」を参照してください。

  Ray ジョブの設定時にこのフィールドに指定できる引数については、デベロッパーガイドの「Ray ジョブでジョブパラメータを使用する」を参照してください。

- NonOverridableArguments – キーバリューペアのマップ配列。

  各キーは UTF-8 文字列。

  各値は UTF-8 文字列。

  名前と値のペアとして指定された、ジョブ実行時にジョブ引数を指定しても上書きされないこのジョブの引数。

- Connections – ConnectionsList オブジェクト。

  このジョブに使用される接続。

- MaxRetries – 数値 (整数)。

  ジョブ実行の失敗後に、このジョブを再試行する最大回数。

- AllocatedCapacity – 数値 (整数)。

  このフィールドは廃止されました。代わりに MaxCapacity を使用します。

  このジョブの実行に割り当てられた AWS Glue データ処理ユニット (DPU) の数。最低 2 つの DPUを割り当てることができます。デフォルトは 10 です。DPU は処理能力を相対的に測定するもので、4 個の vCPU のコンピューティング性能と 16 GB のメモリで構成されています。詳細については、「AWS Glue 料金表ページ」を参照してください。

- Timeout – 数値 (整数)。1 以上。

  ジョブのタイムアウト (分)。ジョブ実行が終了済みになって TIMEOUT ステータスに入るまでに、ジョブ実行でリソースを消費できる最大時間です。

  ジョブのタイムアウト値は 7 日または 10080 分未満である必要があります。それ以外の場合は、ジョブは例外をスローします。

  値を空白のままにすると、タイムアウトはデフォルトで 2,880 分に設定されます。

  タイムアウト値が 7 日を超える既存の AWS Glue ジョブは、デフォルトで 7 日に設定されます。たとえば、バッチジョブに 20 日間のタイムアウトを指定している場合、7 日目に停止します。

  ストリーミングジョブでメンテナンスウィンドウを設定している場合、7 日後にメンテナンスウィンドウ中に再起動されます。

- MaxCapacity – 数値 (double)。

Glue バージョン 1.0 以前のジョブで、標準ワーカータイプを使用して、このジョブの実行時に割り当てることができる AWS Glue データ処理ユニット (DPU) の数。DPU は処理能力を相対的に測定するもので、4 個の vCPU のコンピューティング性能と 16 GB のメモリで構成されています。詳細については、「AWS Glue 料金表ページ」を参照してください。

一方、Glue バージョン 2.0 以降のジョブでは、`Maximum capacity` を指定できません。代わりに、`Worker type` と `Number of workers` を指定する必要があります。

`WorkerType` および `NumberOfWorkers` を使用している場合は `MaxCapacity` を設定しないでください。

`MaxCapacity` に割り当てることができる値は、Python シェルジョブ、Apache Spark ETL ジョブ、Apache Spark ストリーミング ETL ジョブのいずれを実行しているかによって異なります。

- Python シェルジョブを指定すると (`JobCommand.Name="pythonshell"`)、0.0625 または 1 DPU のいずれかを割り当てることができます。デフォルトは 0.0625 DPU です。
- Apache Spark ETL ジョブ (`JobCommand.Name="glueetl"`) または Apache Spark ストリーミング ETL ジョブ (`JobCommand.Name="gluestreaming"`) を指定した場合は、2～100 の DPU を割り当てることができます。デフォルトでは 10 DPU になっています。このジョブタイプには、小数の DPU 割り当てを指定できません。

- `WorkerType` – UTF-8 文字列 (有効な値: `Standard="" | G.1X="" | G.2X="" | G.025X="" | G.4X="" | G.8X="" | Z.2X=""`)。

  ジョブの実行時に割り当てられる事前定義済みのワーカーの種類。Spark ジョブに使用できる値は G.1X、G.2X、G.4X、G.8X、または G.025X です。Ray ジョブに使用できる値は Z.2X です。

  - `G.1X` ワーカータイプでは、各ワーカーは 1 DPU (4 vCPU、16 GB のメモリ、94 GB のディスク) にマッピングされており、ワーカーごとに 1 個のエグゼキュターを提供します。データ変換、結合、クエリなどのワークロードには、ほとんどのジョブを実行するためのスケーラブルで費用対効果の高い方法として、このワーカータイプをお勧めします。
  - `G.2X` ワーカータイプでは、各ワーカーは 2 DPU (8 vCPU、32 GB のメモリ、138 GB のディスク) にマッピングされており、ワーカーごとに 1 個のエグゼキュターを提供します。データ変換、結合、クエリなどのワークロードには、ほとんどのジョブを実行するためのスケーラブルで費用対効果の高い方法として、このワーカータイプをお勧めします。
  - `G.4X` ワーカータイプでは、各ワーカーは 4 DPU (16 vCPU、64 GB のメモリ、256 GB のディスク) にマッピングされており、ワーカーごとに 1 個のエグゼキュターを提供します。ワークロードに含まれる変換、集約、結合、クエリへの要求が非常に厳しいジョブには、このワーカータイプをお勧めします。このワーカータイプは、米国東部 (オハイオ)、米国東部 (バージニア

北部)、米国西部 (北カリフォルニア)、米国西部 (オレゴン)、アジアパシフィック (ムンバイ)、アジアパシフィック (ソウル)、アジアパシフィック (シンガポール)、アジアパシフィック (シドニー)、アジアパシフィック (東京)、カナダ (中部)、欧州 (フランクフルト)、欧州 (アイルランド)、欧州 (ロンドン)、欧州 (スペイン)、欧州 (ストックホルム)、南米 (サンパウロ) の AWS リージョンで、AWS Glue バージョン 3.0 以降の Spark ETL に対してのみ使用できます。

- `G.8X` ワーカータイプでは、各ワーカーは 8 DPU (32 vCPU、128 GB のメモリ、512 GB のディスク) にマッピングされており、ワーカーごとに 1 個のエグゼキュターを提供します。ワークロードに含まれる変換、集約、結合、クエリへの要求が非常に厳しいジョブには、このワーカータイプをお勧めします。このワーカータイプは、AWS Glue バージョン 3.0 以降の Spark ETL ジョブでのみ使用可能で、`G.4X` ワーカータイプでサポートされているのと同じ AWS リージョンで使用できます。
- `G.025X` ワーカータイプでは、各ワーカーは 0.25 DPU (2 vCPU、4 GB のメモリ、84 GB のディスク) にマッピングされており、ワーカーごとに 1 個のエグゼキュターを提供します。少量のストリーミングジョブには、このワーカータイプをお勧めします。このワーカータイプは、AWS Glue バージョン 3.0 以降のストリーミングジョブでのみ使用できます。
- `Z.2X` ワーカータイプでは、各ワーカーは 2 M-DPU (8 vCPU、64 GB のメモリ、128 GB のディスク) にマッピングされており、オートスケーラーに基づき最大 8 個の Ray ワーカーを提供します。

- `NumberOfWorkers` – 数値 (整数)。

  ジョブの実行時に割り当てられた、定義済みの `workerType` ワーカー数。

- `SecurityConfiguration` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  このジョブで使用される `SecurityConfiguration` 構造の名前。

- `NotificationProperty` – NotificationProperty オブジェクト。

  ジョブ通知の設定プロパティを指定します。

- `Running` – ブール。

  このフィールドは、将来の利用のために予約されています。

- `GlueVersion` – UTF-8 文字列。1 ~ 255 バイト長。Custom string pattern #47 に一致。

  Spark ジョブでは、AWS Glue がジョブで使用できる Apache Spark と Python のバージョンは `GlueVersion` によって決まります。Python バージョンは、Spark タイプのジョブでサポートされるバージョンを示します。

Ray ジョブの場合、`GlueVersion` を `4.0` 以降に設定する必要があります。ただし、Ray ジョブで使用できる Ray、Python、および追加ライブラリのバージョンは、Job コマンドの `Runtime` パラメータによって決まります。

利用可能な AWS Glue のバージョン、および対応する Spark および Python のバージョンの詳細については、デベロッパーガイドの「Glue バージョン」を参照してください。

Glue バージョンを指定せずに作成されたジョブは、デフォルトで Glue 0.9 に設定されます。

- `CodeGenConfigurationNodes` – キーバリューペアのマップ配列。

  各キーは、Custom string pattern #58 に適合する UTF-8 文字列です。

  各値は CodeGenConfigurationNode オブジェクトです。

  Glue Studio ビジュアルコンポーネントと、Glue Studio によるコード生成の両方がベースとする、有向非巡回グラフの表現。

- `ExecutionClass` – UTF-8 文字列。16 バイト長以下 (有効値: `FLEX=""` | `STANDARD=""`)。

  ジョブが標準実行クラスまたは柔軟な実行クラスのどちらで実行されるのかを示します。標準実行クラスは、素早くジョブを起動する必要があり、専用のリソースが必要な時間的な制約のあるワークロードに最適です。

  柔軟な実行クラスは、開始時刻と完了時刻が変化する時間的制約が厳しいジョブに適しています。

  AWS Glue バージョン 3.0 以上のジョブの場合のみ、コマンドタイプ `glueetl` で `ExecutionClass` を `FLEX` に設定できます。柔軟な実行クラスは Spark ジョブで使用できます。

- `SourceControlDetails` – SourceControlDetails オブジェクト。

  ジョブのソース管理設定の詳細。これにより、リモートリポジトリとの間でジョブアーティファクトを同期できます。

- `MaintenanceWindow` – UTF-8 文字列。Custom string pattern #34 に一致。

  このフィールドでは、ストリーミングジョブのメンテナンスウィンドウの曜日と時間を指定します。AWS Glue は定期的にメンテナンスアクティビティを実行します。AWS Glue は、これらのメンテナンスウィンドウ中にストリーミングジョブを再起動する必要があります。

  AWS Glue は、指定されたメンテナンスウィンドウの 3 時間以内にジョブを再起動します。たとえば、月曜日の午前 10 時 (GMT) にメンテナンスウィンドウを設定すると、ジョブは午前 10 時 (GMT) から午後 1 時 (GMT) までの間に再起動されます。

- `ProfileName` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  ジョブに関連関連付けられた AWS Glue 使用プロフィールの名前。

ExecutionProperty 構造

ジョブの実行プロパティ。

フィールド

- `MaxConcurrentRuns` – 数値 (整数)。

  このジョブで許可される同時実行の最大数。デフォルトは 1 です。このしきい値に達すると、エラーが返されます。指定できる最大値は、サービスの制限によってコントロールされます。

NotificationProperty 構造

通知の構成プロパティを指定します。

フィールド

- `NotifyDelayAfter` – 数値 (整数)。1 以上。

  ジョブの実行が開始された後、ジョブ実行遅延通知を送信するまでの待機時間 (分単位)。

JobCommand 構造

ジョブの実行時に実行するコードを指定します。

フィールド

- `Name` – UTF-8 文字列。

  ジョブコマンドの名前。Apache Spark ETL ジョブの場合は、`glueetl` を指定する必要があります。Python シェルジョブの場合は、`pythonshell` を指定する必要があります。Apache Spark ストリーミング ETL ジョブの場合は、`gluestreaming` を指定する必要があります。Ray ジョブの場合は、`glueray` を指定する必要があります。

- `ScriptLocation` - UTF-8 文字列。400000 バイト長以下。

ジョブを実行するスクリプトへの Amazon Simple Storage Service (Amazon S3) パスを指定します。

- `PythonVersion` – UTF-8 文字列。Custom string pattern #48 に一致。

  Python シェルジョブを実行するために使用中の Python のバージョン。指定できる値は、2 または 3 です。

- `Runtime` - UTF-8 文字列。64 バイト長以下。Custom string pattern #33 に一致。

  Ray ジョブではランタイムを使用して、環境で使用可能な Ray、Python、および他のライブラリのバージョンを指定します。このフィールドは、他のジョブタイプでは使用されません。サポートされているランタイム環境の値については、「AWS Glue デベロッパーガイド」の「サポートされている Ray のランタイム環境」を参照してください。

ConnectionsList 構造

ジョブが使用する接続を指定します。

フィールド

- `Connections` - UTF-8 文字列の配列。文字列 1,000 個以下。

  ジョブが使用する接続のリスト。

JobUpdate 構造

既存のジョブ定義を更新するための情報を指定します。以前のジョブ定義はこの情報によって完全に上書きされます。

フィールド

- `JobMode` – UTF-8 文字列 (有効な値: `SCRIPT=""` | `VISUAL=""` | `NOTEBOOK=""`)。

  ジョブの作成方法を説明するモード。次の値を指定できます:

  - `SCRIPT` - ジョブは AWS Glue Studio スクリプトエディタを使用して作成されました。
  - `VISUAL` - ジョブは AWS Glue Studio ビジュアルエディタを使用して作成されました。
  - `NOTEBOOK` - ジョブはインタラクティブセッションノートブックを使用して作成されました。

JobMode フィールドが欠落しているか null の場合、SCRIPT がデフォルト値として割り当てられます。

- JobRunQueuingEnabled – ブール。

  このジョブのジョブ実行に対し、ジョブ実行キューイングが有効であるかどうかを指定します。

  true の値は、ジョブ実行キューイングがジョブ実行に対して有効であることを意味します。false または未入力の場合、ジョブ実行はキューイングの対象になりません。

  このフィールドがジョブ実行で設定された値と一致しない場合、ジョブ実行フィールドの値が使用されます。

- Description – 説明文字列。2,048 バイト長以下。URI address multi-line string pattern に一致。

  定義するジョブの説明。

- LogUri – UTF–8 文字列。

  このフィールドは、将来の利用のために予約されています。

- Role – UTF–8 文字列。

  このジョブに関連付けられている IAM ロールの名前または Amazon リソースネーム (ARN) (必須)。

- ExecutionProperty – ExecutionProperty オブジェクト。

  このジョブに許可される同時実行の最大数を指定する ExecutionProperty。

- Command – JobCommand オブジェクト。

  このジョブを実行する JobCommand (必須)。

- DefaultArguments – キーバリューペアのマップ配列。

  各キーは UTF-8 文字列。

  各値は UTF-8 文字列。

  名前と値のペアとして指定された、このジョブを実行するごとのデフォルトの引数。

  独自のジョブ実行スクリプトが消費する引数だけでなく、AWS Glue が消費する引数もここで指定できます。

ジョブ引数はログに記録される場合があります。プレーンテキストのシークレットを引数として渡さないでください。ジョブ内に保持する場合は、AWS Glue 接続、AWS Secrets Manager または他のシークレット管理メカニズムから取得します。

独自のジョブ引数を指定および使用する方法については、デベロッパーガイドのトピック「Python で AWS Glue API を呼び出す」を参照してください。

Spark ジョブの設定時にこのフィールドに指定できる引数については、デベロッパーガイドのトピック「AWS Glueで使用するスペシャルパラメータ」を参照してください。

Ray ジョブの設定時にこのフィールドに指定できる引数については、デベロッパーガイドの「Ray ジョブでジョブパラメータを使用する」を参照してください。

- `NonOverridableArguments` – キーバリューペアのマップ配列。

  各キーは UTF-8 文字列。

  各値は UTF-8 文字列。

  名前と値のペアとして指定された、ジョブ実行時にジョブ引数を指定しても上書きされないこのジョブの引数。

- `Connections` – ConnectionsList オブジェクト。

  このジョブに使用される接続。

- `MaxRetries` – 数値 (整数)。

  失敗した場合にこのジョブを再試行する最大回数。

- `AllocatedCapacity` – 数値 (整数)。

  このフィールドは廃止されました。代わりに `MaxCapacity` を使用します。

  このジョブに割り当てる AWS Glue データ処理ユニット (DPU) の数。最低 2 つの DPUを割り当てることができます。デフォルトは 10 です。DPU は処理能力を相対的に測定するもので、4 個の vCPU のコンピューティング性能と 16 GB のメモリで構成されています。詳細については、「AWS Glue 料金表ページ」を参照してください。

- `Timeout` – 数値 (整数)。1 以上。

  ジョブのタイムアウト (分)。ジョブ実行が終了済みになって `TIMEOUT` ステータスに入るまでに、ジョブ実行でリソースを消費できる最大時間です。

ジョブのタイムアウト値は 7 日または 10080 分未満である必要があります。それ以外の場合は、ジョブは例外をスローします。

値を空白のままにすると、タイムアウトはデフォルトで 2,880 分に設定されます。

タイムアウト値が 7 日を超える既存の AWS Glue ジョブは、デフォルトで 7 日に設定されます。たとえば、バッチジョブに 20 日間のタイムアウトを指定している場合、7 日目に停止します。

ストリーミングジョブでメンテナンスウィンドウを設定している場合、7 日後にメンテナンスウィンドウ中に再起動されます。

- `MaxCapacity` – 数値 (double)。

  Glue バージョン 1.0 以前のジョブで、標準ワーカータイプを使用して、このジョブの実行時に割り当てることができる AWS Glue データ処理ユニット (DPU) の数。DPU は処理能力を相対的に測定するもので、4 個の vCPU のコンピューティング性能と 16 GB のメモリで構成されています。詳細については、「AWS Glue 料金表ページ」を参照してください。

  一方、Glue バージョン 2.0 以降のジョブでは、`Maximum capacity` を指定できません。代わりに、`Worker type` と `Number of workers` を指定する必要があります。

  `WorkerType` および `NumberOfWorkers` を使用している場合は `MaxCapacity` を設定しないでください。

  `MaxCapacity` に割り当てることができる値は、Python シェルジョブ、Apache Spark ETL ジョブ、Apache Spark ストリーミング ETL ジョブのいずれを実行しているかによって異なります。

  - Python シェルジョブを指定すると (`JobCommand.Name="pythonshell"`)、0.0625 または 1 DPU のいずれかを割り当てることができます。デフォルトは 0.0625 DPU です。
  - Apache Spark ETL ジョブ (`JobCommand.Name="glueetl"`) または Apache Spark ストリーミング ETL ジョブ (`JobCommand.Name="gluestreaming"`) を指定した場合は、2～100 の DPU を割り当てることができます。デフォルトでは 10 DPU になっています。このジョブタイプには、小数の DPU 割り当てを指定できません。

- `WorkerType` – UTF-8 文字列 (有効な値: `Standard="" | G.1X="" | G.2X="" | G.025X="" | G.4X="" | G.8X="" | Z.2X=""`)。

  ジョブの実行時に割り当てられる事前定義済みのワーカーの種類。Spark ジョブに使用できる値は G.1X、G.2X、G.4X、G.8X、または G.025X です。Ray ジョブに使用できる値は Z.2X です。詳細については、「Spark ジョブのジョブプロパティの定義」を参照してください

- NumberOfWorkers – 数値 (整数)。

  ジョブの実行時に割り当てられた、定義済みの workerType ワーカー数。

- SecurityConfiguration – UTF-8 文字列。1～255 バイト長。Single-line string pattern に一致。

  このジョブで使用される SecurityConfiguration 構造の名前。

- NotificationProperty – NotificationProperty オブジェクト。

  ジョブ通知の設定プロパティを指定します。

- GlueVersion – UTF-8 文字列。1～255 バイト長。Custom string pattern #47 に一致。

  Spark ジョブでは、AWS Glue がジョブで使用できる Apache Spark と Python のバージョンは GlueVersion によって決まります。Python バージョンは、Spark タイプのジョブでサポートされるバージョンを示します。

  Ray ジョブの場合、GlueVersion を 4.0 以降に設定する必要があります。ただし、Ray ジョブで使用できる Ray、Python、および追加ライブラリのバージョンは、Job コマンドの Runtime パラメータによって決まります。

  利用可能な AWS Glue のバージョン、および対応する Spark および Python のバージョンの詳細については、デベロッパーガイドの「Glue バージョン」を参照してください。

  Glue バージョンを指定せずに作成されたジョブは、デフォルトで Glue 0.9 に設定されます。

- CodeGenConfigurationNodes – キーバリューペアのマップ配列。

  各キーは、Custom string pattern #58 に適合する UTF-8 文字列です。

  各値は CodeGenConfigurationNode オブジェクトです。

  Glue Studio ビジュアルコンポーネントと、Glue Studio によるコード生成の両方がベースとする、有向非巡回グラフの表現。

- ExecutionClass – UTF-8 文字列。16 バイト長以下 (有効値: FLEX="" | STANDARD="")。

  ジョブが標準実行クラスまたは柔軟な実行クラスのどちらで実行されるのかを示します。標準の実行クラスは、素早くジョブを起動する必要があり、専用のリソースが必要な時間的な制約のあるワークロードに最適です。

  柔軟な実行クラスは、開始時刻と完了時刻が変化する時間的制約が厳しいジョブに適しています。

AWS Glue バージョン 3.0 以上のジョブの場合のみ、コマンドタイプ glueetl で ExecutionClass を FLEX に設定できます。柔軟な実行クラスは Spark ジョブで使用できます。

- SourceControlDetails – SourceControlDetails オブジェクト。

  ジョブのソース管理設定の詳細。これにより、リモートリポジトリとの間でジョブアーティファクトを同期できます。

- MaintenanceWindow – UTF-8 文字列。Custom string pattern #34 に一致。

  このフィールドでは、ストリーミングジョブのメンテナンスウィンドウの曜日と時間を指定します。AWS Glue は定期的にメンテナンスアクティビティを実行します。AWS Glue は、これらのメンテナンスウィンドウ中にストリーミングジョブを再起動する必要があります。

  AWS Glue は、指定されたメンテナンスウィンドウの 3 時間以内にジョブを再起動します。たとえば、月曜日の午前 10 時 (GMT) にメンテナンスウィンドウを設定すると、ジョブは午前 10 時 (GMT) から午後 1 時 (GMT) までの間に再起動されます。

- ProfileName – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  ジョブに関連関連付けられた AWS Glue 使用プロフィールの名前。

SourceControlDetails の構造

ジョブのソース管理設定の詳細。これにより、リモートリポジトリとの間でジョブアーティファクトを同期できます。

フィールド

- Provider – UTF-8 文字列 (有効な値: GITHUB | AWS_CODE_COMMIT)。

  リモートリポジトリのプロバイダ。

- Repository – UTF-8 文字列。1 ~ 512 バイト長。

  ジョブのアーティファクトを含むリモートリポジトリの名前。

- Owner – UTF-8 文字列。1 ~ 512 バイト長。

  ジョブのアーティファクトを含むリモートリポジトリの所有者。

- Branch – UTF-8 文字列。1 ~ 512 バイト長。

  リモートリポジトリ内のオプションのブランチ。

- `Folder` – UTF-8 文字列。1 ~ 512 バイト長。

  リモートリポジトリ内のオプションのフォルダ。
- `LastCommitId` – UTF-8 文字列。1 ~ 512 バイト長。

  リモートリポジトリ内のコミットの最後のコミット ID。
- `LastSyncTimestamp` – UTF-8 文字列。1 ~ 512 バイト長。

  ジョブ同期が最後に実行された日時。
- `AuthStrategy` – UTF-8 文字列 (有効な値: `PERSONAL_ACCESS_TOKEN` | `AWS_SECRETS_MANAGER`)。

  認証のタイプ。AWS Secrets Manager に保存されている認証トークンまたは個人用アクセストークンのいずれかです。
- `AuthToken` – UTF-8 文字列。1 ~ 512 バイト長。

  認可トークンの値。

操作

- CreateJob アクション (Python: create_job)
- UpdateJob アクション (Python: update_job)
- GetJob アクション (Python: get_job)
- GetJobs アクション (Python: get_jobs)
- DeleteJob アクション (Python: delete_job)
- ListJobs アクション (Python: list_jobs)
- BatchGetJobs アクション (Python: batch_get_jobs)

CreateJob アクション (Python: create_job)

新しいジョブ定義を作成します。

リクエスト

- `Name` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  このジョブ定義に割り当てる名前。 アカウント内で一意にする必要があります。

- JobMode – UTF-8 文字列 (有効な値: SCRIPT="" | VISUAL="" | NOTEBOOK="")。

  ジョブの作成方法を説明するモード。次の値を指定できます:

  - SCRIPT - ジョブは AWS Glue Studio スクリプトエディタを使用して作成されました。
  - VISUAL - ジョブは AWS Glue Studio ビジュアルエディタを使用して作成されました。
  - NOTEBOOK - ジョブはインタラクティブセッションノートブックを使用して作成されました。

  JobMode フィールドが欠落しているか null の場合、SCRIPT がデフォルト値として割り当てられます。

- JobRunQueuingEnabled – ブール。

  このジョブのジョブ実行に対し、ジョブ実行キューイングが有効であるかどうかを指定します。

  true の値は、ジョブ実行キューイングがジョブ実行に対して有効であることを意味します。false または未入力の場合、ジョブ実行はキューイングの対象になりません。

  このフィールドがジョブ実行で設定された値と一致しない場合、ジョブ実行フィールドの値が使用されます。

- Description – 説明文字列。2,048 バイト長以下。URI address multi-line string pattern に一致。

  定義するジョブの説明。

- LogUri – UTF–8 文字列。

  このフィールドは、将来の利用のために予約されています。

- Role – 必須: UTF–8 文字列。

  このジョブに関連付けられている IAM ロールの名前または Amazon リソースネーム (ARN)。

- ExecutionProperty – ExecutionProperty オブジェクト。

  このジョブに許可される同時実行の最大数を指定する ExecutionProperty。

- Command – 必須: JobCommand オブジェクト。

  このジョブを実行する JobCommand。

- DefaultArguments – キーバリューペアのマップ配列。

  各キーは UTF-8 文字列。

  各値は UTF-8 文字列。

名前と値のペアとして指定された、このジョブを実行するごとのデフォルトの引数。

独自のジョブ実行スクリプトが消費する引数だけでなく、AWS Glue が消費する引数もここで指定できます。

ジョブ引数はログに記録される場合があります。プレーンテキストのシークレットを引数として渡さないでください。ジョブ内に保持する場合は、AWS Glue 接続、AWS Secrets Manager または他のシークレット管理メカニズムから取得します。

独自のジョブ引数を指定および使用する方法については、デベロッパーガイドのトピック「Python で AWS Glue API を呼び出す」を参照してください。

Spark ジョブの設定時にこのフィールドに指定できる引数については、デベロッパーガイドのトピック「AWS Glueで使用するスペシャルパラメータ」を参照してください。

Ray ジョブの設定時にこのフィールドに指定できる引数については、デベロッパーガイドの「Ray ジョブでジョブパラメータを使用する」を参照してください。

- `NonOverridableArguments` – キーバリューペアのマップ配列。

  各キーは UTF-8 文字列。

  各値は UTF-8 文字列。

  名前と値のペアとして指定された、ジョブ実行時にジョブ引数を指定しても上書きされないこのジョブの引数。

- `Connections` – ConnectionsList オブジェクト。

  このジョブに使用される接続。

- `MaxRetries` – 数値 (整数)。

  失敗した場合にこのジョブを再試行する最大回数。

- `AllocatedCapacity` – 数値 (整数)。

  このパラメータは廃止されました。代わりに `MaxCapacity` を使用します。

  このジョブに割り当てる AWS Glue データ処理ユニット (DPU) の数。最低 2 つの DPUを割り当てることができます。デフォルトは 10 です。DPU は処理能力を相対的に測定するもので、4 個の vCPU のコンピューティング性能と 16 GB のメモリで構成されています。詳細については、「AWS Glue 料金表ページ」を参照してください。

- `Timeout` – 数値 (整数)。1 以上。

  ジョブのタイムアウト (分)。ジョブ実行が終了済みになって `TIMEOUT` ステータスに入るまでに、ジョブ実行でリソースを消費できる最大時間です。

  ジョブのタイムアウト値は 7 日または 10080 分未満である必要があります。それ以外の場合は、ジョブは例外をスローします。

  値を空白のままにすると、タイムアウトはデフォルトで 2,880 分に設定されます。

  タイムアウト値が 7 日を超える既存の AWS Glue ジョブは、デフォルトで 7 日に設定されます。たとえば、バッチジョブに 20 日間のタイムアウトを指定している場合、7 日目に停止します。

  ストリーミングジョブでメンテナンスウィンドウを設定している場合、7 日後にメンテナンスウィンドウ中に再起動されます。

- `MaxCapacity` – 数値 (double)。

  Glue バージョン 1.0 以前のジョブで、標準ワーカータイプを使用して、このジョブの実行時に割り当てることができる AWS Glue データ処理ユニット (DPU) の数。DPU は処理能力を相対的に測定するもので、4 個の vCPU のコンピューティング性能と 16 GB のメモリで構成されています。詳細については、「AWS Glue 料金表ページ」を参照してください。

  一方、Glue バージョン 2.0 以降のジョブでは、`Maximum capacity` を指定できません。代わりに、`Worker type` と `Number of workers` を指定する必要があります。

  `WorkerType` および `NumberOfWorkers` を使用している場合は `MaxCapacity` を設定しないでください。

  `MaxCapacity` に割り当てることができる値は、Python シェルジョブ、Apache Spark ETL ジョブ、Apache Spark ストリーミング ETL ジョブのいずれを実行しているかによって異なります。

  - Python シェルジョブを指定すると (`JobCommand.Name="pythonshell"`)、0.0625 または 1 DPU のいずれかを割り当てることができます。デフォルトは 0.0625 DPU です。
  - Apache Spark ETL ジョブ (`JobCommand.Name="glueetl"`) または Apache Spark ストリーミング ETL ジョブ (`JobCommand.Name="gluestreaming"`) を指定した場合は、2 ~ 100 の DPU を割り当てることができます。デフォルトでは 10 DPU になっています。このジョブタイプには、小数の DPU 割り当てを指定できません。

- `SecurityConfiguration` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

このジョブで使用される SecurityConfiguration 構造の名前。

- Tags – キーと値のペアのマップ配列。50 ペア以下。

  各キーは UTF-8 文字列で、1 ~ 128 バイト長です。

  各値は UTF-8 文字列で、256 バイト長以下です。

  このジョブと一緒に使用するタグです。ジョブへのアクセスを制限するためにタグを使用することができます。AWS Glue のタグの詳細については、デベロッパーガイドの「AWS Tags in AWS Glue」を参照してください。

- NotificationProperty – NotificationProperty オブジェクト。

  ジョブ通知の設定プロパティを指定します。

- GlueVersion – UTF-8 文字列。1 ~ 255 バイト長。Custom string pattern #47 に一致。

  Spark ジョブでは、AWS Glue がジョブで使用できる Apache Spark と Python のバージョンは GlueVersion によって決まります。Python バージョンは、Spark タイプのジョブでサポートされるバージョンを示します。

  Ray ジョブの場合、GlueVersion を 4.0 以降に設定する必要があります。ただし、Ray ジョブで使用できる Ray、Python、および追加ライブラリのバージョンは、Job コマンドの Runtime パラメータによって決まります。

  利用可能な AWS Glue のバージョン、および対応する Spark および Python のバージョンの詳細については、デベロッパーガイドの「Glue バージョン」を参照してください。

  Glue バージョンを指定せずに作成されたジョブは、デフォルトで Glue 0.9 に設定されます。

- NumberOfWorkers – 数値 (整数)。

  ジョブの実行時に割り当てられた、定義済みの workerType ワーカー数。

- WorkerType – UTF-8 文字列 (有効な値: Standard="" | G.1X="" | G.2X="" | G.025X="" | G.4X="" | G.8X="" | Z.2X="")。

  ジョブの実行時に割り当てられる事前定義済みのワーカーの種類。Spark ジョブに使用できる値は G.1X、G.2X、G.4X、G.8X、または G.025X です。Ray ジョブに使用できる値は Z.2X です。

  - G.1X ワーカータイプでは、各ワーカーは 1 DPU (4 vCPU、16 GB のメモリ、94 GB のディスク) にマッピングされており、ワーカーごとに 1 個のエグゼキュターを提供します。データ変

換、結合、クエリなどのワークロードには、ほとんどのジョブを実行するためのスケーラブルで費用対効果の高い方法として、このワーカータイプをお勧めします。

- G.2X ワーカータイプでは、各ワーカーは 2 DPU (8 vCPU、32 GB のメモリ、138 GB のディスク) にマッピングされており、ワーカーごとに 1 個の エグゼキュターを提供します。データ変換、結合、クエリなどのワークロードには、ほとんどのジョブを実行するためのスケーラブルで費用対効果の高い方法として、このワーカータイプをお勧めします。
- G.4X ワーカータイプでは、各ワーカーは 4 DPU (16 vCPU、64 GB のメモリ、256 GB のディスク) にマッピングされており、ワーカーごとに 1 個の エグゼキュターを提供します。ワークロードに含まれる変換、集約、結合、クエリへの要求が非常に厳しいジョブには、このワーカータイプをお勧めします。このワーカータイプは、米国東部 (オハイオ)、米国東部 (バージニア北部)、米国西部 (北カリフォルニア)、米国西部 (オレゴン)、アジアパシフィック (ムンバイ)、アジアパシフィック (ソウル)、アジアパシフィック (シンガポール)、アジアパシフィック (シドニー)、アジアパシフィック (東京)、カナダ (中部)、欧州 (フランクフルト)、欧州 (アイルランド)、欧州 (ロンドン)、欧州 (スペイン)、欧州 (ストックホルム)、南米 (サンパウロ) の AWS リージョンで、AWS Glue バージョン 3.0 以降の Spark ETL に対してのみ使用できます。
- G.8X ワーカータイプでは、各ワーカーは 8 DPU (32 vCPU、128 GB のメモリ、512 GB のディスク) にマッピングされており、ワーカーごとに 1 個の エグゼキュターを提供します。ワークロードに含まれる変換、集約、結合、クエリへの要求が非常に厳しいジョブには、このワーカータイプをお勧めします。このワーカータイプは、AWS Glue バージョン 3.0 以降の Spark ETL ジョブでのみ使用可能で、G.4X ワーカータイプでサポートされているのと同じ AWS リージョンで使用できます。
- G.025X ワーカータイプでは、各ワーカーは 0.25 DPU (2 vCPU、4 GB のメモリ、84 GB のディスク) にマッピングされており、ワーカーごとに 1 個の エグゼキュターを提供します。少量のストリーミングジョブには、このワーカータイプをお勧めします。このワーカータイプは、AWS Glue バージョン 3.0 以降のストリーミングジョブでのみ使用できます。
- Z.2X ワーカータイプでは、各ワーカーは 2 M-DPU (8 vCPU、64 GB のメモリ、128 GB のディスク) にマッピングされており、オートスケーラーに基づき最大 8 個の Ray ワーカーを提供します。

- CodeGenConfigurationNodes – キーバリューペアのマップ配列。

  各キーは、Custom string pattern #58 に適合する UTF-8 文字列です。

  各値は CodeGenConfigurationNode オブジェクトです。

  Glue Studio ビジュアルコンポーネントと、Glue Studio によるコード生成の両方がベースとする、有向非巡回グラフの表現。

- ExecutionClass – UTF-8 文字列。16 バイト長以下 (有効値: FLEX="" | STANDARD="")。

  ジョブが標準実行クラスまたは柔軟な実行クラスのどちらで実行されるのかを示します。標準の実行クラスは、素早くジョブを起動する必要があり、専用のリソースが必要な時間的な制約のあるワークロードに最適です。

  柔軟な実行クラスは、開始時刻と完了時刻が変化する時間的制約が厳しいジョブに適しています。

  AWS Glue バージョン 3.0 以上のジョブの場合のみ、コマンドタイプ glueetl で ExecutionClass を FLEX に設定できます。柔軟な実行クラスは Spark ジョブで使用できます。
- SourceControlDetails – SourceControlDetails オブジェクト。

  ジョブのソース管理設定の詳細。これにより、リモートリポジトリとの間でジョブアーティファクトを同期できます。
- MaintenanceWindow – UTF-8 文字列。Custom string pattern #34 に一致。

  このフィールドでは、ストリーミングジョブのメンテナンスウィンドウの曜日と時間を指定します。AWS Glue は定期的にメンテナンスアクティビティを実行します。AWS Glue は、これらのメンテナンスウィンドウ中にストリーミングジョブを再起動する必要があります。

  AWS Glue は、指定されたメンテナンスウィンドウの 3 時間以内にジョブを再起動します。たとえば、月曜日の午前 10 時 (GMT) にメンテナンスウィンドウを設定すると、ジョブは午前 10 時 (GMT) から午後 1 時 (GMT) までの間に再起動されます。
- ProfileName – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  ジョブに関連関連付けられた AWS Glue 使用プロフィールの名前。

レスポンス

- Name – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  このジョブ定義に指定された一意の名前。

エラー

- InvalidInputException
- IdempotentParameterMismatchException
- AlreadyExistsException

- InternalServiceException
- OperationTimeoutException
- ResourceNumberLimitExceededException
- ConcurrentModificationException

UpdateJob アクション (Python: update_job)

既存のジョブ定義を更新します。以前のジョブ定義はこの情報によって完全に上書きされます。

リクエスト

- JobName – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  更新するジョブ定義の名前。

- JobUpdate – 必須: JobUpdate オブジェクト。

  ジョブ定義の更新に使用する値を指定します。指定されていない設定は削除されるか、デフォルト値にリセットされます。

- ProfileName – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  ジョブに関連関連付けられた AWS Glue 使用プロフィールの名前。

レスポンス

- JobName – UTF-8 文字列、1 ~ 255 バイト長。Single-line string pattern に一致。

  更新されたジョブ定義の名前を返します。

エラー

- InvalidInputException
- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException
- ConcurrentModificationException

GetJob アクション (Python: get_job)

既存のジョブ定義を取得します。

リクエスト

- JobName – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  取得するジョブ定義の名前。

レスポンス

- Job – Job オブジェクト。

  リクエストされたジョブ定義。

エラー

- InvalidInputException
- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException

GetJobs アクション (Python: get_jobs)

すべての現在のジョブ定義を取得します。

リクエスト

- NextToken – UTF-8 文字列。

  継続トークン (これが継続呼び出しの場合)。
- MaxResults – 1 ~ 1000 の数値 (整数)。

  応答の最大サイズ。

応答

- Jobs – Job オブジェクトの配列。

ジョブ定義のリスト。

- NextToken – UTF–8 文字列。

  継続トークン (一部のジョブ定義がまだ返されていない場合)。

エラー

- InvalidInputException
- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException

DeleteJob アクション (Python: delete_job)

指定したジョブ定義を削除します。ジョブ定義が見つからない場合、例外はスローされません。

リクエスト

- JobName – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  削除するジョブ定義の名前。

レスポンス

- JobName – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  削除されたジョブ定義の名前。

エラー

- InvalidInputException
- InternalServiceException
- OperationTimeoutException

ListJobs アクション (Python: list_jobs)

この AWS アカウントのすべてのジョブリソース、または指定されたタグを持つリソースの名前を取得します。このオペレーションにより、アカウントで利用可能なリソースとその名前を確認できます。

このオペレーションはオプションの Tags フィールドを受け取ります。このフィールドを応答のフィルターとして使用すると、タグ付きリソースをグループとして取得できます。タグフィルタリングの使用を選択した場合は、タグが付いたリソースのみが取得されます。

リクエスト

- NextToken – UTF-8 文字列。

  継続トークン (これが継続リクエストの場合)。
- MaxResults – 1 ~ 1000 の数値 (整数)。

  返されるリストの最大サイズ。
- Tags – キーと値のペアのマップ配列。50 ペア以下。

  各キーは UTF-8 文字列で、1 ~ 128 バイト長です。

  各値は UTF-8 文字列で、256 バイト長以下です。

  これらのタグ付きリソースのみを返すように指定します。

レスポンス

- JobNames – UTF-8 文字列の配列。

  アカウント内のすべてのジョブの名前、または指定されたタグを持つジョブの名前。
- NextToken – UTF–8 文字列。

  継続トークン (戻されたリストに最後に使用可能なメトリクスが含まれていない場合)。

エラー

- InvalidInputException
- EntityNotFoundException

- InternalServiceException
- OperationTimeoutException

BatchGetJobs アクション (Python: batch_get_jobs)

指定されたジョブ名のリストのリソースメタデータのリストを返します。ListJobs オペレーションを呼び出した後で、このオペレーションを呼び出すことで、アクセス許可が付与されているデータにアクセスできます。このオペレーションは、タグを使用するアクセス許可条件を含め、すべての IAM のアクセス許可をサポートします。

リクエスト

- JobNames – 必須: UTF-8 文字列の配列。

  ジョブ名のリスト。これは ListJobs 操作から返された名前であることもあります。

応答

- Jobs – Job オブジェクトの配列。

  ジョブ定義のリスト。
- JobsNotFound – UTF-8 文字列の配列。

  ジョブの名前のリストが見つかりません。

エラー

- InternalServiceException
- OperationTimeoutException
- InvalidInputException

ジョブ実行

Jobs Runs API では、AWS Glue における、ジョブの実行の開始、停止、表示と、ジョブのブックマークのリセットに関連するデータ型と API について説明します。ジョブ実行履歴には、ワークフローとジョブ実行のために 90 日間アクセスできます。

データ型

- JobRun の構造
- Predecessor 構造
- JobBookmarkEntry 構造
- BatchStopJobRunSuccessfulSubmission 構造
- BatchStopJobRunError 構造
- NotificationProperty 構造

JobRun の構造

ジョブ実行についての情報が含まれています。

フィールド

- `Id` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  このジョブ実行の ID。
- `Attempt` – 数値 (整数)。

  このジョブを実行しようと試みた回数。
- `PreviousRunId` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  このジョブの以前の実行の ID。例えば、`StartJobRun` アクションで指定された `JobRunId` を表します。
- `TriggerName` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  このジョブ実行を開始したトリガーの名前。
- `JobName` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  この実行で使用されているジョブ定義の名前。
- `JobMode` – UTF-8 文字列 (有効な値: `SCRIPT=""` | `VISUAL=""` | `NOTEBOOK=""`)。

  ジョブの作成方法を説明するモード。次の値を指定できます:
  - `SCRIPT` - ジョブは AWS Glue Studio スクリプトエディタを使用して作成されました。
  - `VISUAL` - ジョブは AWS Glue Studio ビジュアルエディタを使用して作成されました。

  - NOTEBOOK - ジョブはインタラクティブセッションノートブックを使用して作成されました。

  JobMode フィールドが欠落しているか null の場合、SCRIPT がデフォルト値として割り当てられます。

- JobRunQueuingEnabled – ブール。

  ジョブ実行に対してジョブ実行キューイングが有効であるかどうかを指定します。

  true の値は、ジョブ実行キューイングがジョブ実行に対して有効であることを意味します。false または未入力の場合、ジョブ実行はキューイングの対象になりません。

- StartedOn – タイムスタンプ。

  このジョブ実行が開始された日付と時刻。

- LastModifiedOn – タイムスタンプ。

  このジョブ実行が最後に変更された時刻。

- CompletedOn – タイムスタンプ。

  このジョブ実行が完了した日付と時刻。

- JobRunState – UTF-8 文字列 (有効な値:STARTING | RUNNING | STOPPING | STOPPED | SUCCEEDED | FAILED | TIMEOUT | ERROR | WAITING | EXPIRED)。

  現在のジョブ実行の状態。異常終了したジョブのステータスの詳細については、「AWS Glue Job Run のステータス」を参照してください。

- Arguments – キーバリューペアのマップ配列。

  各キーは UTF-8 文字列。

  各値は UTF-8 文字列。

  この実行に関連付けられているジョブの引数。このジョブ実行では、ジョブ定義自体に設定されているデフォルト引数を置き換えます。

  独自のジョブ実行スクリプトが消費する引数だけでなく、AWS Glue が消費する引数もここで指定できます。

  ジョブ引数はログに記録される場合があります。プレーンテキストのシークレットを引数として渡さないでください。ジョブ内に保持する場合は、AWS Glue 接続、AWS Secrets Manager または他のシークレット管理メカニズムから取得します。

独自のジョブ引数を指定および使用する方法については、デベロッパーガイドのトピック「Python で AWS Glue API を呼び出す」を参照してください。

Spark ジョブの設定時にこのフィールドに指定できる引数については、デベロッパーガイドのトピック「AWS Glueで使用するスペシャルパラメータ」を参照してください。

Ray ジョブの設定時にこのフィールドに指定できる引数については、デベロッパーガイドの「Ray ジョブでジョブパラメータを使用する」を参照してください。

- `ErrorMessage` – UTF–8 文字列。

  このジョブ実行に関連付けられているエラーメッセージ。

- `PredecessorRuns` – Predecessor オブジェクトの配列。

  このジョブ実行に先行するもののリスト。

- `AllocatedCapacity` – 数値 (整数)。

  このフィールドは廃止されました。代わりに `MaxCapacity` を使用します。

  この JobRun に割り当てられた AWS Glue データ処理ユニット (DPU) の数。2 ~ 100 DPU の範囲で割り当てることができます。デフォルト値は 10 です。DPU は処理能力を相対的に測定するもので、4 個の vCPU のコンピューティング性能と 16 GB のメモリで構成されています。詳細については、「AWS Glue 料金表ページ」を参照してください。

- `ExecutionTime` – 数値 (整数)。

  ジョブ実行でリソースを消費した時間 (秒)。

- `Timeout` – 数値 (整数)。1 以上。

  JobRun のタイムアウト (分)。ジョブ実行が終了済みになって `TIMEOUT` ステータスに入るまでに、ジョブ実行でリソースを消費できる最大時間です。この値は、親ジョブで設定したタイムアウト値を上書きします。

  ジョブのタイムアウト値は 7 日または 10080 分未満である必要があります。それ以外の場合は、ジョブは例外をスローします。

  値を空白のままにすると、タイムアウトはデフォルトで 2,880 分に設定されます。

  タイムアウト値が 7 日を超える既存の AWS Glue ジョブは、デフォルトで 7 日に設定されます。たとえば、バッチジョブに 20 日間のタイムアウトを指定している場合、7 日目に停止します。

ストリーミングジョブでメンテナンスウィンドウを設定している場合、7 日後にメンテナンスウィンドウ中に再起動されます。

- `MaxCapacity` – 数値 (double)。

  Glue バージョン 1.0 以前のジョブで、標準ワーカータイプを使用して、このジョブの実行時に割り当てることができる AWS Glue データ処理ユニット (DPU) の数。DPU は処理能力を相対的に測定するもので、4 個の vCPU のコンピューティング性能と 16 GB のメモリで構成されています。詳細については、「AWS Glue 料金表ページ」を参照してください。

  一方、Glue バージョン 2.0 以降のジョブでは、`Maximum capacity` を指定できません。代わりに、`Worker type` と `Number of workers` を指定する必要があります。

  `WorkerType` および `NumberOfWorkers` を使用している場合は `MaxCapacity` を設定しないでください。

  `MaxCapacity` に割り当てることができる値は、Python シェルジョブ、Apache Spark ETL ジョブ、Apache Spark ストリーミング ETL ジョブのいずれを実行しているかによって異なります。

  - Python シェルジョブを指定すると (`JobCommand.Name="pythonshell"`)、0.0625 または 1 DPU のいずれかを割り当てることができます。デフォルトは 0.0625 DPU です。
  - Apache Spark ETL ジョブ (`JobCommand.Name="glueetl"`) または Apache Spark ストリーミング ETL ジョブ (`JobCommand.Name="gluestreaming"`) を指定した場合は、2～100 の DPU を割り当てることができます。デフォルトでは 10 DPU になっています。このジョブタイプには、小数の DPU 割り当てを指定できません。

- `WorkerType` – UTF-8 文字列 (有効な値: `Standard="" | G.1X="" | G.2X="" | G.025X="" | G.4X="" | G.8X="" | Z.2X=""`)。

  ジョブの実行時に割り当てられる事前定義済みのワーカーの種類。Spark ジョブに使用できる値は G.1X、G.2X、G.4X、G.8X、または G.025X です。Ray ジョブに使用できる値は Z.2X です。

  - `G.1X` ワーカータイプでは、各ワーカーは 1 DPU (4 vCPU、16 GB のメモリ、94 GB のディスク) にマッピングされており、ワーカーごとに 1 個のエグゼキューターを提供します。データ変換、結合、クエリなどのワークロードには、ほとんどのジョブを実行するためのスケーラブルで費用対効果の高い方法として、このワーカータイプをお勧めします。
  - `G.2X` ワーカータイプでは、各ワーカーは 2 DPU (8 vCPU、32 GB のメモリ、138 GB のディスク) にマッピングされており、ワーカーごとに 1 個のエグゼキューターを提供します。データ変換、結合、クエリなどのワークロードには、ほとんどのジョブを実行するためのスケーラブルで費用対効果の高い方法として、このワーカータイプをお勧めします。

- G.4X ワーカータイプでは、各ワーカーは 4 DPU (16 vCPU、64 GB のメモリ、256 GB のディスク) にマッピングされており、ワーカーごとに 1 個のエグゼキュターを提供します。ワークロードに含まれる変換、集約、結合、クエリへの要求が非常に厳しいジョブには、このワーカータイプをお勧めします。このワーカータイプは、AWS Glue バージョン 3.0 以降の Spark ETL ジョブでのみ使用可能で、以下の AWS リージョンで使用できます。米国東部 (オハイオ)、米国東部 (バージニア北部)、米国西部 (オレゴン)、アジアパシフィック (シンガポール)、アジアパシフィック (シドニー)、アジアパシフィック (東京)、カナダ (中部)、欧州 (フランクフルト)、欧州 (アイルランド)、欧州 (ストックホルム)。
- G.8X ワーカータイプでは、各ワーカーは 8 DPU (32 vCPU、128 GB のメモリ、512 GB のディスク) にマッピングされており、ワーカーごとに 1 個のエグゼキュターを提供します。ワークロードに含まれる変換、集約、結合、クエリへの要求が非常に厳しいジョブには、このワーカータイプをお勧めします。このワーカータイプは、AWS Glue バージョン 3.0 以降の Spark ETL ジョブでのみ使用可能で、G.4X ワーカータイプでサポートされているのと同じ AWS リージョンで使用できます。
- G.025X ワーカータイプでは、各ワーカーは 0.25 DPU (2 vCPU、4 GB のメモリ、84 GB のディスク) にマッピングされており、ワーカーごとに 1 個のエグゼキュターを提供します。少量のストリーミングジョブには、このワーカータイプをお勧めします。このワーカータイプは、AWS Glue バージョン 3.0 以降のストリーミングジョブでのみ使用できます。
- Z.2X ワーカータイプでは、各ワーカーは 2 M-DPU (8 vCPU、64 GB のメモリ、128 GB のディスク) にマッピングされており、オートスケーラーに基づき最大 8 個の Ray ワーカーを提供します。

- NumberOfWorkers – 数値 (整数)。

  ジョブの実行時に割り当てられた、定義済みの workerType ワーカー数。

- SecurityConfiguration – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  このジョブ実行で使用される SecurityConfiguration 構造の名前。

- LogGroupName – UTF–8 文字列。

  セキュアなログ記録のための、ロググループの名前。AWS KMS を使用して、Amazon CloudWatch のサーバー側で暗号化することができます。この名前には /aws-glue/jobs/ を指定できます。その場合、デフォルトの暗号化は NONE です。ロール名および SecurityConfiguration 名 (つまり /aws-glue/jobs-yourRoleName-yourSecurityConfigurationName/) を追加すると、そのセキュリティ設定がロググループの暗号化に使用されます。

- NotificationProperty – NotificationProperty オブジェクト。

  ジョブ実行通知の構成プロパティを指定します。

- GlueVersion – UTF-8 文字列。1 ~ 255 バイト長。Custom string pattern #47 に一致。

  Spark ジョブでは、AWS Glue がジョブで使用できる Apache Spark と Python のバージョンは GlueVersion によって決まります。Python バージョンは、Spark タイプのジョブでサポートされるバージョンを示します。

  Ray ジョブの場合、GlueVersion を 4.0 以降に設定する必要があります。ただし、Ray ジョブで使用できる Ray、Python、および追加ライブラリのバージョンは、Job コマンドの Runtime パラメータによって決まります。

  利用可能な AWS Glue のバージョン、および対応する Spark および Python のバージョンの詳細については、デベロッパーガイドの「Glue バージョン」を参照してください。

  Glue バージョンを指定せずに作成されたジョブは、デフォルトで Glue 0.9 に設定されます。

- DPUSeconds – 数値 (double)。

  このフィールドは、実行クラス FLEX で実行するジョブまたは自動スケーリングが有効になっているときに実行するジョブのいずれかについて設定できます。これは、ジョブ実行のライフサイクル中に各エグゼキューターが実行された合計時間 (秒単位) を DPU 係数 (G.1X ワーカーの場合は 1、G.2X ワーカーの場合は 2、G.025X ワーカーの場合は 0.25) で乗じたものを表します。Auto Scaling ジョブの場合に、この値は executionEngineRuntime * MaxCapacity とは異なる場合がありますが、これは特定の時間に実行される工グゼキュータの数が MaxCapacity より少ないことがあるためです。この場合、DPUSeconds の値が executionEngineRuntime * MaxCapacity より小さくなる可能性があります。

- ExecutionClass – UTF-8 文字列。16 バイト長以下 (有効値: FLEX="" | STANDARD="")。

  ジョブが標準実行クラスまたは柔軟な実行クラスのどちらで実行されるのかを示します。標準の実行クラスは、素早くジョブを起動する必要があり、専用のリソースが必要な時間的な制約のあるワークロードに最適です。

  柔軟な実行クラスは、開始時刻と完了時刻が変化する時間的制約が厳しいジョブに適しています。

  AWS Glue バージョン 3.0 以上のジョブの場合のみ、コマンドタイプ glueetl で ExecutionClass を FLEX に設定できます。柔軟な実行クラスは Spark ジョブで使用できます。

- MaintenanceWindow – UTF-8 文字列。Custom string pattern #34 に一致。

このフィールドでは、ストリーミングジョブのメンテナンスウィンドウの曜日と時間を指定します。AWS Glue は定期的にメンテナンスアクティビティを実行します。AWS Glue は、これらのメンテナンスウィンドウ中にストリーミングジョブを再起動する必要があります。

AWS Glue は、指定されたメンテナンスウィンドウの 3 時間以内にジョブを再起動します。たとえば、月曜日の午前 10 時 (GMT) にメンテナンスウィンドウを設定すると、ジョブは午前 10 時 (GMT) から午後 1 時 (GMT) までの間に再起動されます。

- `ProfileName` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  ジョブ実行に関連付けられた AWS Glue 使用プロフィールの名前。
- `StateDetail` - UTF-8 文字列。400,000 バイト長以下。

  このフィールドは、ジョブ実行の状態に関連する詳細を格納します。フィールドは null 許容です。

  例えば、ジョブ実行キューイングの結果としてジョブ実行が WAITING 状態のとき、フィールドにはジョブ実行がその状態になっている理由が示されます。

Predecessor 構造

このジョブ実行をトリガーした条件トリガーの述語に使用されたジョブ実行。

フィールド

- `JobName` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  先行するジョブ実行で使用したジョブ定義の名前。
- `RunId` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  先行するジョブ実行のジョブ実行 ID。

JobBookmarkEntry 構造

ジョブの処理を再開できるポイントを定義します。

フィールド

- `JobName` – UTF-8 文字列。

該当するジョブの名前。

- Version – 数値 (整数)。

  ジョブのバージョン。

- Run – 数値 (整数)。

  実行 ID 番号。

- Attempt – 数値 (整数)。

  試行 ID 番号。

- PreviousRunId – UTF–8 文字列。

  前回のジョブ実行に関連付けられた一意の実行識別子。

- RunId – UTF–8 文字列。

  実行 ID 番号。

- JobBookmark – UTF–8 文字列。

  ブックマーク自体。

BatchStopJobRunSuccessfulSubmission 構造

指定された JobRun を停止するリクエストの成功を記録します。

フィールド

- JobName – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  停止したジョブ実行で使用したジョブ定義の名前。

- JobRunId – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  停止したジョブ実行の JobRunId。

BatchStopJobRunError 構造

指定したジョブ実行を停止しようとして発生したエラーを記録します。

フィールド

- JobName – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  該当するジョブ実行で使用したジョブ定義の名前。
- JobRunId – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  該当するジョブ実行の JobRunId。
- ErrorDetail – ErrorDetail オブジェクト。

  発生したエラーに関する詳細を指定します。

NotificationProperty 構造

通知の構成プロパティを指定します。

フィールド

- NotifyDelayAfter – 数値 (整数)。1 以上。

  ジョブの実行が開始された後、ジョブ実行遅延通知を送信するまでの待機時間 (分単位)。

操作

- StartJobRun アクション (Python: start_job_run)
- BatchStopJobRun アクション (Python: batch_stop_job_run)
- GetJobRun アクション (Python: get_job_run)
- GetJobRuns アクション (Python: get_job_runs)
- GetJobBookmark アクション (Python: get_job_bookmark)
- GetJobBookmarks アクション (Python: get_job_bookmarks)
- ResetJobBookmark アクション (Python: reset_job_bookmark)

StartJobRun アクション (Python: start_job_run)

ジョブ定義を使用してジョブ実行を開始します。

リクエスト

- JobName – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  使用するジョブ定義の名前。

- JobRunQueuingEnabled – ブール。

  ジョブ実行に対してジョブ実行キューイングが有効であるかどうかを指定します。

  true の値は、ジョブ実行キューイングがジョブ実行に対して有効であることを意味します。false または未入力の場合、ジョブ実行はキューイングの対象になりません。

- JobRunId – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  再試行する以前の JobRun の ID。

- Arguments – キーバリューペアのマップ配列。

  各キーは UTF-8 文字列。

  各値は UTF-8 文字列。

  この実行に関連付けられているジョブの引数。このジョブ実行では、ジョブ定義自体に設定されているデフォルト引数を置き換えます。

  独自のジョブ実行スクリプトが消費する引数だけでなく、AWS Glue が消費する引数もここで指定できます。

  ジョブ引数はログに記録される場合があります。プレーンテキストのシークレットを引数として渡さないでください。ジョブ内に保持する場合は、AWS Glue 接続、AWS Secrets Manager または他のシークレット管理メカニズムから取得します。

  独自のジョブ引数を指定および使用する方法については、デベロッパーガイドのトピック「Python で AWS Glue API を呼び出す」を参照してください。

  Spark ジョブの設定時にこのフィールドに指定できる引数については、デベロッパーガイドのトピック「AWS Glueで使用するスペシャルパラメータ」を参照してください。

  Ray ジョブの設定時にこのフィールドに指定できる引数については、デベロッパーガイドの「Ray ジョブでジョブパラメータを使用する」を参照してください。

- AllocatedCapacity – 数値 (整数)。

このフィールドは廃止されました。代わりに MaxCapacity を使用します。

この JobRun に割り当てる AWS Glue データ処理ユニット (DPU) の数。最低 2 つの DPUを割り当てることができます。デフォルトは 10 です。DPU は処理能力を相対的に測定するもので、4 個の vCPU のコンピューティング性能と 16 GB のメモリで構成されています。詳細については、「AWS Glue 料金表ページ」を参照してください。

- Timeout – 数値 (整数)。1 以上。

  JobRun のタイムアウト (分)。ジョブ実行が終了済みになって TIMEOUT ステータスに入るまでに、ジョブ実行でリソースを消費できる最大時間です。この値は、親ジョブで設定したタイムアウト値を上書きします。

  ジョブのタイムアウト値は 7 日または 10080 分未満である必要があります。それ以外の場合は、ジョブは例外をスローします。

  値を空白のままにすると、タイムアウトはデフォルトで 2,880 分に設定されます。

  タイムアウト値が 7 日を超える既存の AWS Glue ジョブは、デフォルトで 7 日に設定されます。たとえば、バッチジョブに 20 日間のタイムアウトを指定している場合、7 日目に停止します。

  ストリーミングジョブでメンテナンスウィンドウを設定している場合、7 日後にメンテナンスウィンドウ中に再起動されます。

- MaxCapacity – 数値 (double)。

  Glue バージョン 1.0 以前のジョブで、標準ワーカータイプを使用して、このジョブの実行時に割り当てることができる AWS Glue データ処理ユニット (DPU) の数。DPU は処理能力を相対的に測定するもので、4 個の vCPU のコンピューティング性能と 16 GB のメモリで構成されています。詳細については、「AWS Glue 料金表ページ」を参照してください。

  一方、Glue バージョン 2.0 以降のジョブでは、Maximum capacity を指定できません。代わりに、Worker type と Number of workers を指定する必要があります。

  WorkerType および NumberOfWorkers を使用している場合は MaxCapacity を設定しないでください。

  MaxCapacity に割り当てることができる値は、Python シェルジョブ、Apache Spark ETL ジョブ、Apache Spark ストリーミング ETL ジョブのいずれを実行しているかによって異なります。

- Python シェルジョブを指定すると (`JobCommand.Name="pythonshell"`)、0.0625 または 1 DPU のいずれかを割り当てることができます。デフォルトは 0.0625 DPU です。
- Apache Spark ETL ジョブ (`JobCommand.Name="glueetl"`) または Apache Spark ストリーミング ETL ジョブ (`JobCommand.Name="gluestreaming"`) を指定した場合は、2～100 の DPU を割り当てることができます。デフォルトでは 10 DPU になっています。このジョブタイプには、小数の DPU 割り当てを指定できません。

- `SecurityConfiguration` – UTF-8 文字列。1～255 バイト長。Single-line string pattern に一致。

  このジョブ実行で使用される `SecurityConfiguration` 構造の名前。

- `NotificationProperty` – NotificationProperty オブジェクト。

  ジョブ実行通知の構成プロパティを指定します。

- `WorkerType` – UTF-8 文字列 (有効な値: `Standard="" | G.1X="" | G.2X="" | G.025X="" | G.4X="" | G.8X="" | Z.2X=""`)。

  ジョブの実行時に割り当てられる事前定義済みのワーカーの種類。Spark ジョブに使用できる値は G.1X、G.2X、G.4X、G.8X、または G.025X です。Ray ジョブに使用できる値は Z.2X です。

  - `G.1X` ワーカータイプでは、各ワーカーは 1 DPU (4 vCPU、16 GB のメモリ、94 GB のディスク) にマッピングされており、ワーカーごとに 1 個のエグゼキュターを提供します。データ変換、結合、クエリなどのワークロードには、ほとんどのジョブを実行するためのスケーラブルで費用対効果の高い方法として、このワーカータイプをお勧めします。
  - `G.2X` ワーカータイプでは、各ワーカーは 2 DPU (8 vCPU、32 GB のメモリ、138 GB のディスク) にマッピングされており、ワーカーごとに 1 個のエグゼキュターを提供します。データ変換、結合、クエリなどのワークロードには、ほとんどのジョブを実行するためのスケーラブルで費用対効果の高い方法として、このワーカータイプをお勧めします。
  - `G.4X` ワーカータイプでは、各ワーカーは 4 DPU (16 vCPU、64 GB のメモリ、256 GB のディスク) にマッピングされており、ワーカーごとに 1 個のエグゼキュターを提供します。ワークロードに含まれる変換、集約、結合、クエリへの要求が非常に厳しいジョブには、このワーカータイプをお勧めします。このワーカータイプは、AWS Glue バージョン 3.0 以降の Spark ETL ジョブでのみ使用可能で、以下の AWS リージョンで使用できます。米国東部 (オハイオ)、米国東部 (バージニア北部)、米国西部 (オレゴン)、アジアパシフィック (シンガポール)、アジアパシフィック (シドニー)、アジアパシフィック (東京)、カナダ (中部)、欧州 (フランクフルト)、欧州 (アイルランド)、欧州 (ストックホルム)。
  - `G.8X` ワーカータイプでは、各ワーカーは 8 DPU (32 vCPU、128 GB のメモリ、512 GB のディスク) にマッピングされており、ワーカーごとに 1 個のエグゼキュターを提供します。ワーク

ロードに含まれる変換、集約、結合、クエリへの要求が非常に厳しいジョブには、このワーカータイプをお勧めします。このワーカータイプは、AWS Glue バージョン 3.0 以降の Spark ETL ジョブでのみ使用可能で、G.4X ワーカータイプでサポートされているのと同じ AWS リージョンで使用できます。

- G.025X ワーカータイプでは、各ワーカーは 0.25 DPU (2 vCPU、4 GB のメモリ、84 GB のディスク) にマッピングされており、ワーカーごとに 1 個のエグゼキュターを提供します。少量のストリーミングジョブには、このワーカータイプをお勧めします。このワーカータイプは、AWS Glue バージョン 3.0 以降のストリーミングジョブでのみ使用できます。
- Z.2X ワーカータイプでは、各ワーカーは 2 M-DPU (8 vCPU、64 GB のメモリ、128 GB のディスク) にマッピングされており、オートスケーラーに基づき最大 8 個の Ray ワーカーを提供します。

- NumberOfWorkers – 数値 (整数)。

  ジョブの実行時に割り当てられた、定義済みの workerType ワーカー数。

- ExecutionClass – UTF-8 文字列。16 バイト長以下 (有効値: FLEX="" | STANDARD="")。

  ジョブが標準実行クラスまたは柔軟な実行クラスのどちらで実行されるのかを示します。標準の実行クラスは、素早くジョブを起動する必要があり、専用のリソースが必要な時間的な制約のあるワークロードに最適です。

  柔軟な実行クラスは、開始時刻と完了時刻が変化する時間的制約が厳しいジョブに適しています。

  AWS Glue バージョン 3.0 以上のジョブの場合のみ、コマンドタイプ glueetl で ExecutionClass を FLEX に設定できます。柔軟な実行クラスは Spark ジョブで使用できます。

- ProfileName – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  ジョブ実行に関連付けられた AWS Glue 使用プロフィールの名前。

レスポンス

- JobRunId – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  このジョブ実行に割り当てられた ID。

エラー

- InvalidInputException

- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException
- ResourceNumberLimitExceededException
- ConcurrentRunsExceededException

BatchStopJobRun アクション (Python: batch_stop_job_run)

指定したジョブ定義の 1 つ以上のジョブ実行を停止します。

リクエスト

- JobName – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  ジョブ実行を停止するジョブ定義の名前。
- JobRunIds – 必須: UTF-8 文字列の配列。1 ~ 25 個の文字列。

  このジョブ定義で停止する JobRunIds のリスト。

応答

- SuccessfulSubmissions – BatchStopJobRunSuccessfulSubmission オブジェクトの配列。

  正常に送信された停止対象の JobRun のリスト。
- Errors – BatchStopJobRunError オブジェクトの配列。

  JobRuns を停止しようとして発生したエラーのリスト。各エラーで発生した JobRunId とエラーの詳細が含まれます。

エラー

- InvalidInputException
- InternalServiceException
- OperationTimeoutException

GetJobRun アクション (Python: get_job_run)

指定されたジョブ実行のメタデータを取得します。ジョブ実行履歴には、ワークフローとジョブ実行のために 365 日間アクセスできます。

リクエスト

- `JobName` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  実行中のジョブ定義の名前。
- `RunId` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  ジョブ実行の ID。
- `PredecessorsIncluded` – ブール。

  以前の実行のリストが返される場合は、true です。

レスポンス

- `JobRun` – JobRun オブジェクト。

  リクエストされたジョブ実行のメタデータ。

エラー

- `InvalidInputException`
- `EntityNotFoundException`
- `InternalServiceException`
- `OperationTimeoutException`

GetJobRuns アクション (Python: get_job_runs)

特定のジョブ定義に該当するすべての実行のメタデータを取得します。

`GetJobRuns` は、最新のジョブを最初に返して、ジョブの実行を時系列で返します。

リクエスト

- `JobName` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

すべてのジョブ実行を取得するジョブ定義の名前。

- NextToken – UTF-8 文字列。

  継続トークン (これが継続呼び出しの場合)。

- MaxResults – 1 未満または 200 を超えない数値 (整数)。

  応答の最大サイズ。

応答

- JobRuns – JobRun オブジェクトの配列。

  ジョブ実行のメタデータオブジェクトのリスト。

- NextToken – UTF–8 文字列。

  継続トークン (一部のリクエストされたジョブがまだ返されていない場合)。

エラー

- InvalidInputException
- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException

GetJobBookmark アクション (Python: get_job_bookmark)

ジョブブックマークエントリに関する情報を返します。

ジョブのブックマークの有効化と使用の詳細については、以下を参照してください。

- ジョブのブックマークを使用した処理済みデータの追跡
- AWS Glue によって使用されるジョブパラメータ
- Job 構造

リクエスト

- JobName – 必須: UTF–8 文字列。

  該当するジョブの名前。
- Version – 数値 (整数)。

  ジョブのバージョン。
- RunId – UTF–8 文字列。

  このジョブの実行に関連付けられた一意の実行識別子。

レスポンス

- JobBookmarkEntry – JobBookmarkEntry オブジェクト。

  ジョブの処理を再開できるポイントを定義する構造。

エラー

- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException
- ValidationException

GetJobBookmarks アクション (Python: get_job_bookmarks)

ジョブブックマークエントリに関する情報を返します。リストは、バージョン番号の小さい順に並べられます。

ジョブのブックマークの有効化と使用の詳細については、以下を参照してください。

- ジョブのブックマークを使用した処理済みデータの追跡
- AWS Glue によって使用されるジョブパラメータ
- Job 構造

リクエスト

- JobName – 必須: UTF–8 文字列。

  該当するジョブの名前。

- MaxResults – 数値 (整数)。

  応答の最大サイズ。

- NextToken – 数値 (整数)。

  継続トークン (これが継続呼び出しの場合)。

応答

- JobBookmarkEntries – JobBookmarkEntry オブジェクトの配列。

  ジョブの処理を再開できるポイントを定義するジョブブックマークエントリのリスト。

- NextToken – 数値 (整数)。

  すべてのエントリが返される場合は 1、要求されたすべてのジョブ実行が返されなかった場合は 1 より大きい値を持つ、継続トークン。

エラー

- InvalidInputException
- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException

ResetJobBookmark アクション (Python: reset_job_bookmark)

ブックマークエントリをリセットします。

ジョブのブックマークの有効化と使用の詳細については、以下を参照してください。

- ジョブのブックマークを使用した処理済みデータの追跡
- AWS Glue によって使用されるジョブパラメータ

- Job 構造

リクエスト

- JobName – 必須: UTF–8 文字列。

  該当するジョブの名前。
- RunId – UTF–8 文字列。

  このジョブの実行に関連付けられた一意の実行識別子。

レスポンス

- JobBookmarkEntry – JobBookmarkEntry オブジェクト。

  ブックマークエントリのリセット。

エラー

- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException

トリガー

Triggers API では、AWS Glue におけるジョブトリガーの作成、更新、削除と、開始および停止に関連するデータ型と API について説明します。

データ型

- トリガー構造
- TriggerUpdate 構造
- 述語構造
- 条件の構造
- アクション構造

- EventBatchingCondition 構造

トリガー構造

特定のトリガーに関する情報です。

フィールド

- Name – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  トリガーの名前。

- WorkflowName – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  トリガーに関連付けられているワークフローの名前。

- Id – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  将来の利用のために予約されています。

- Type – UTF-8 文字列 (有効な値: SCHEDULED | CONDITIONAL | ON_DEMAND | EVENT)。

  これがトリガーのタイプです。

- State – UTF-8 文字列 (有効な値: CREATING | CREATED | ACTIVATING | ACTIVATED | DEACTIVATING | DEACTIVATED | DELETING | UPDATING)。

  現在のトリガーの状態。

- Description – 説明文字列。2,048 バイト長以下。URI address multi-line string pattern に一致。

  このトリガーの説明。

- Schedule – UTF–8 文字列。

  スケジュールを指定するために使用される cron 式 (ジョブとクローラーの時間ベースのスケジュールを参照してください。) たとえば、毎日 12:15 UTC に何かを実行するには、cron(15 12 * * ? *) を指定します。

- Actions – アクション オブジェクトの配列。

  このトリガーによって開始されるアクション。

- Predicate – 述語 オブジェクト。

  このトリガーの述語は、いつトリガーを起動するかを定義します。

- `EventBatchingCondition` – EventBatchingCondition オブジェクト。

  EventBridge イベントトリガーが起動する前に、満たす必要があるBatch 条件 (指定された受信イベント数またはバッチ時間ウィンドウの期限切れ)。

TriggerUpdate 構造

トリガーの更新に使用する情報を提供するために使用される構造です。このオブジェクトは、前のトリガー定義を完全に上書きして更新します。

フィールド

- `Name` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  将来の利用のために予約されています。

- `Description` – 説明文字列。2,048 バイト長以下。URI address multi-line string pattern に一致。

  このトリガーの説明。

- `Schedule` – UTF–8 文字列。

  スケジュールを指定するために使用される `cron` 式 (ジョブとクローラーの時間ベースのスケジュールを参照してください。) たとえば、毎日 12:15 UTC に何かを実行するには、`cron(15 12 * * ? *)` を指定します。

- `Actions` – アクション オブジェクトの配列。

  このトリガーによって開始されるアクション。

- `Predicate` – 述語 オブジェクト。

  このトリガーの述語は、いつトリガーを起動するかを定義します。

- `EventBatchingCondition` – EventBatchingCondition オブジェクト。

  EventBridge イベントトリガーが起動する前に、満たす必要があるBatch 条件 (指定された受信イベント数またはバッチ時間ウィンドウの期限切れ)。

述語構造

トリガーがいつ起動するかを決定するトリガーの述語を定義します。

フィールド

- Logical – UTF-8 文字列 (有効な値: AND | ANY)。

  1 つの条件のみが表示されている場合のオプションフィールドです。複数の条件が表示されている場合、このフィールドは必須です。
- Conditions – 条件 オブジェクトの配列。構造 500 個以下。

  トリガーがいつ起動するかを決定する条件のリスト。

条件の構造

トリガーが起動する条件を定義します。

フィールド

- LogicalOperator – UTF-8 文字列 (有効な値: EQUALS)。

  論理演算子。
- JobName – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  JobRuns のこの条件が適用され、このトリガーが待機するジョブの名前。
- State – UTF-8 文字列 (有効な値:STARTING | RUNNING | STOPPING | STOPPED | SUCCEEDED | FAILED | TIMEOUT | ERROR | WAITING | EXPIRED)。

  条件の状態。現在、トリガーがリッスンできるジョブ状態は SUCCEEDED、STOPPED、FAILED、および TIMEOUT のみです。トリガーがリッスンできるクローラー状態は SUCCEEDED、FAILED、および CANCELLED のみです。
- CrawlerName – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  この条件が適用されるクローラーの名前。
- CrawlState - UTF-8 文字列 (有効値: RUNNING | CANCELLING | CANCELLED | SUCCEEDED | FAILED | ERROR).。

  この条件が適用されるクローラーの状態。

アクション構造

トリガーによって開始されるアクションを定義します。

フィールド

- JobName – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  実行されるジョブの名前。

- Arguments – キーバリューペアのマップ配列。

  各キーは UTF-8 文字列。

  各値は UTF-8 文字列。

  このトリガーが起動するときに使用されるジョブ引数。このジョブ実行では、ジョブ定義自体に設定されているデフォルト引数を置き換えます。

  独自のジョブ実行スクリプトが消費する引数だけでなく、AWS Glue が消費する引数もここで指定できます。

  独自のジョブ引数を指定および使用する方法については、デベロッパーガイドのトピック「Calling AWS Glue APIs in Python」を参照してください。

  AWS Glue がジョブを設定するために使用するキーと値のペアについては、デベロッパーガイドのトピック「Special Parameters Used by AWS Glue」を参照してください。

- Timeout – 数値 (整数)。1 以上。

  JobRun のタイムアウト (分)。ジョブ実行が終了済みになって TIMEOUT ステータスに入るまでに、ジョブ実行でリソースを消費できる最大時間です。これにより、親ジョブで設定したタイムアウト値が上書きされます。

  ジョブのタイムアウト値は 7 日または 10080 分未満である必要があります。それ以外の場合は、ジョブは例外をスローします。

  値を空白のままにすると、タイムアウトはデフォルトで 2,880 分に設定されます。

  タイムアウト値が 7 日を超える既存の AWS Glue ジョブは、デフォルトで 7 日に設定されます。たとえば、バッチジョブに 20 日間のタイムアウトを指定している場合、7 日目に停止します。

  ストリーミングジョブでメンテナンスウィンドウを設定している場合、7 日後にメンテナンスウィンドウ中に再起動されます。

- SecurityConfiguration – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

このジョブで使用される SecurityConfiguration 構造の名前。

- NotificationProperty – NotificationProperty オブジェクト。

  ジョブ実行通知の構成プロパティを指定します。

- CrawlerName – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  このアクションで使用されるクローラーの名前。

EventBatchingCondition 構造

EventBridge イベントトリガーが起動する前に、満たす必要があるBatch 条件 (指定された受信イベント数またはバッチ時間ウィンドウの期限切れ)。

フィールド

- BatchSize – 必須: 数値 (integer)。1 ~ 100。

  EventBridge イベントトリガーが発生する前に、Amazon EventBridge から受信する必要のあるイベントの数。

- BatchWindow – 数値 (integer)。1 ~ 900。

  EventBridge イベントトリガーが起動するまでの時間 (秒単位)。最初のイベントを受信したときからの時間です。

操作

- CreateTrigger アクション (Python: create_trigger)
- StartTrigger アクション (Python: start_trigger)
- GetTrigger アクション (Python: get_trigger)
- GetTriggers アクション (Python: get_triggers)
- UpdateTrigger アクション (Python: update_trigger)
- StopTrigger アクション (Python: stop_trigger)
- DeleteTrigger アクション (Python: delete_trigger)
- ListTriggers アクション (Python: list_triggers)
- BatchGetTriggers アクション (Python: batch_get_triggers)

CreateTrigger アクション (Python: create_trigger)

新しいトリガーを作成します。

ジョブ引数はログに記録される場合があります。プレーンテキストのシークレットを引数として渡さないでください。ジョブ内に保持する場合は、AWS Glue 接続、AWS Secrets Manager、または他のシークレット管理メカニズムから取得します。

リクエスト

- `Name` – 必須: UTF-8 文字列。1～255 バイト長。Single-line string pattern に一致。

  トリガーの名前。

- `WorkflowName` – UTF-8 文字列。1～255 バイト長。Single-line string pattern に一致。

  トリガーに関連付けられているワークフローの名前。

- `Type` – 必須: UTF-8 文字列 (有効な値: `SCHEDULED` | `CONDITIONAL` | `ON_DEMAND` | `EVENT`)。

  新しいトリガーのタイプ。

- `Schedule` – UTF–8 文字列。

  スケジュールを指定するために使用される `cron` 式 (ジョブとクローラーの時間ベースのスケジュールを参照してください。) たとえば、毎日 12:15 UTC に何かを実行するには、`cron(15 12 * * ? *)` を指定します。

  このフィールドは、トリガータイプが SCHEDULED の場合に必要です。

- `Predicate` – 述語 オブジェクト。

  新しいトリガーがいつ起動するかを指定する述語です。

  このフィールドは、トリガータイプが `CONDITIONAL` の場合に必要です。

- `Actions` – 必須: アクション オブジェクトの配列。

  このトリガーが起動したときに開始されるアクション。

- `Description` – 説明文字列。2,048 バイト長以下、URI address multi-line string pattern に一致。

  新しいトリガーの説明。

- `StartOnCreation` – ブール。

作成時に SCHEDULED および CONDITIONAL トリガーを起動するには、true に設定します。True は ON_DEMAND トリガーではサポートされません。

- Tags – キーと値のペアのマップ配列。50 ペア以下。

  各キーは UTF-8 文字列で、1 ~ 128 バイト長です。

  各値は UTF-8 文字列で、256 バイト長以下です。

  このトリガーで使用するタグ。トリガーへのアクセスを制限するためにタグを使用することができます。AWS Glue のタグの詳細については、デベロッパーガイドの「AWS Tags in AWS Glue」を参照してください。

- EventBatchingCondition – EventBatchingCondition オブジェクト。

  EventBridge イベントトリガーが起動する前に、満たす必要があるBatch 条件 (指定された受信イベント数またはバッチ時間ウィンドウの期限切れ)。

レスポンス

- Name – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  トリガーの名前。

エラー

- AlreadyExistsException
- EntityNotFoundException
- InvalidInputException
- IdempotentParameterMismatchException
- InternalServiceException
- OperationTimeoutException
- ResourceNumberLimitExceededException
- ConcurrentModificationException

StartTrigger アクション (Python: start_trigger)

既存のトリガーを開始します。さまざまなタイプのトリガーの開始方法については、「ジョブのトリガー」を参照してください。

リクエスト

- Name – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  開始するトリガーの名前。

レスポンス

- Name – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  開始されたトリガーの名前。

エラー

- `InvalidInputException`
- `InternalServiceException`
- `EntityNotFoundException`
- `OperationTimeoutException`
- `ResourceNumberLimitExceededException`
- `ConcurrentRunsExceededException`

GetTrigger アクション (Python: get_trigger)

トリガーの定義を取得します。

リクエスト

- Name – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  取得するトリガーの名前。

レスポンス

- `Trigger` – Trigger トリガー) オブジェクト。

  リクエストされたトリガー定義。

エラー

- `EntityNotFoundException`
- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`

GetTriggers アクション (Python: get_triggers)

ジョブに関連付けられているすべてのトリガーを取得します。

リクエスト

- `NextToken` – UTF-8 文字列。

  継続トークン (これが継続呼び出しの場合)。
- `DependentJobName` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  トリガーを取得するジョブの名前。このジョブを開始できるトリガーが返されます。このようなトリガーがない場合、すべてのトリガーが返されます。
- `MaxResults` – 1 未満または 200 を超えない数値 (整数)。

  応答の最大サイズ。

応答

- `Triggers` – Trigger トリガー) オブジェクトの配列。

  指定されたジョブのトリガーのリスト。
- `NextToken` – UTF–8 文字列。

  リクエストされたトリガーのすべてがまだ返されていない場合は、継続トークン。

エラー

- `EntityNotFoundException`
- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`

UpdateTrigger アクション (Python: update_trigger)

トリガー定義を更新します。

ジョブ引数は口グに記録される場合があります。プレーンテキストのシークレットを引数として渡さないでください。ジョブ内に保持する場合は、AWS Glue 接続、AWS Secrets Manager、または他のシークレット管理メカニズムから取得します。

リクエスト

- `Name` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  更新するトリガーの名前。
- `TriggerUpdate` – 必須: TriggerUpdate オブジェクト。

  トリガーの更新に使用する新しい値。

レスポンス

- `Trigger` – Trigger トリガー) オブジェクト。

  結果として生じるトリガー定義。

エラー

- `InvalidInputException`
- `InternalServiceException`
- `EntityNotFoundException`
- `OperationTimeoutException`
- `ConcurrentModificationException`

StopTrigger アクション (Python: stop_trigger)

指定されたトリガーを停止します。

リクエスト

- Name – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  停止するトリガーの名前。

レスポンス

- Name – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  停止したトリガーの名前。

エラー

- `InvalidInputException`
- `InternalServiceException`
- `EntityNotFoundException`
- `OperationTimeoutException`
- `ConcurrentModificationException`

DeleteTrigger アクション (Python: delete_trigger)

指定されたトリガーを削除します。トリガーが見つからない場合、例外はスローされません。

リクエスト

- Name – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  削除するトリガーの名前。

レスポンス

- Name – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  削除されたトリガーの名前。

エラー

- InvalidInputException
- InternalServiceException
- OperationTimeoutException
- ConcurrentModificationException

ListTriggers アクション (Python: list_triggers)

この AWS アカウントのすべてのトリガーリソース、または指定されたタグを持つリソースの名前を取得します。このオペレーションにより、アカウントで利用可能なリソースとその名前を確認できます。

このオペレーションはオプションの Tags フィールドを受け取ります。このフィールドを応答のフィルターとして使用すると、タグ付きリソースをグループとして取得できます。タグフィルタリングの使用を選択した場合は、タグが付いたリソースのみが取得されます。

リクエスト

- NextToken – UTF-8 文字列。

  継続トークン (これが継続リクエストの場合)。

- DependentJobName – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  トリガーを取得するジョブの名前。このジョブを開始できるトリガーが返されます。このようなトリガーがない場合は、すべてのトリガーが返ります。

- MaxResults – 1 未満または 200 を超えない数値 (整数)。

  返されるリストの最大サイズ。

- Tags – キーと値のペアのマップ配列。50 ペア以下。

  各キーは UTF-8 文字列で、1 ~ 128 バイト長です。

  各値は UTF-8 文字列で、256 バイト長以下です。

  これらのタグ付きリソースのみを返すように指定します。

レスポンス

- TriggerNames – UTF-8 文字列の配列。

  アカウント内のすべてのトリガーの名前、または指定されたタグを持つトリガーの名前。

- NextToken – UTF–8 文字列。

  継続トークン (戻されたリストに最後に使用可能なメトリクスが含まれていない場合)。

エラー

- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException

BatchGetTriggers アクション (Python: batch_get_triggers)

指定されたトリガー名のリストのリソースメタデータのリストを返します。ListTriggers オペレーションを呼び出した後で、このオペレーションを呼び出すことで、アクセス許可が付与されているデータにアクセスできます。このオペレーションは、タグを使用するアクセス許可条件を含め、すべての IAM のアクセス許可をサポートします。

リクエスト

- TriggerNames – 必須: UTF-8 文字列の配列。

  トリガー名のリスト。これは ListTriggers オペレーションから返される名前です。

応答

- Triggers – Trigger トリガー) オブジェクトの配列。

  トリガー定義のリスト。

- TriggersNotFound – UTF-8 文字列の配列。

  トリガーの名前のリストが見つかりません。

エラー

- `InternalServiceException`
- `OperationTimeoutException`
- `InvalidInputException`

AWS Glue の統合 API

データ型

- 統合構造
- IntegrationConfig 構造
- IntegrationPartition 構造
- IntegrationError 構造
- IntegrationFilter 構造
- InboundIntegration 構造
- SourceProcessingProperties 構造
- TargetProcessingProperties 構造
- SourceTableConfig 構造
- TargetTableConfig 構造

統合構造

ゼロ ETL 統合について説明します。

フィールド

- `SourceArn` – 必須: UTF-8 文字列。1 ~ 128 バイト長。

  統合のソースの ARN。
- `TargetArn` – 必須: UTF-8 文字列。1 ~ 128 バイト長。

  統合のターゲットの ARN。
- `Description` – UTF-8 文字列。1,000 バイト長以下。Custom string pattern #12 に一致。

統合の説明

- IntegrationName – 必須: UTF-8 文字列。1 ~ 128 バイト長。

  統合の一意の名前。

- IntegrationArn – 必須: UTF-8 文字列。1 ~ 128 バイト長。

  統合の Amazon リソースネーム (ARN)。

- KmsKeyId – UTF-8 文字列。1 ~ 2,048 バイト長。

  チャネルの暗号化に使用される KMS キーの ARN。

- AdditionalEncryptionContext – キーバリューペアのマップ配列。

  各キーは UTF-8 文字列。

  各値は UTF-8 文字列。

  暗号化のための追加のコンテキスト情報を含む、シークレット以外のキーと値のペアのオプションセット。これは KMSKeyId が指定されている場合にのみ提供されます。

- Tags – Tag オブジェクトの配列。

  キーと値のペアのリストで構成されるリソースに割り当てられるメタデータ。

- Status – 必須: UTF-8 文字列 (有効な値: CREATING | ACTIVE | MODIFYING | FAILED | DELETING | SYNCING | NEEDS_ATTENTION)。

  次のようなステータスがあります。

  - CREATING: 統合を作成中です。
  - ACTIVE: 統合の作成が完了しました。
  - MODIFYING: 統合を変更中です。
  - FAILED: 統合を作成できませんでした。
  - DELETING: 統合が削除されました。
  - SYNCING: 統合を同期中です。
  - NEEDS_ATTENTION: 統合では同期などに注意を払う必要があります。

- CreateTime – 必須: タイムスタンプ。

  統合が作成された時刻 (UTC)。

- IntegrationConfig – IntegrationConfig オブジェクト。

統合に関連付けられたプロパティ。

- `Errors` – IntegrationError オブジェクトの配列。

  統合に関連するエラーのリスト。

- `DataFilter` – UTF-8 文字列。1 ~ 2,048 バイト長。

  Maxwell フィルター構文を使用して統合のソーステーブルを選択します。

IntegrationConfig 構造

統合に関連付けられたプロパティ。

フィールド

- `RefreshInterval` – UTF-8 文字列、1 ~ 128 バイト長。

  CDC (Change Data Capture) プルまたは増分ロードが発生する頻度を指定します。このパラメータは、更新レートを特定のデータ更新パターン、システム負荷に関する考慮事項、パフォーマンス最適化の目標に柔軟に合わせることができます。時間増分は 15 分から 8640 分 (6 日間) に設定できます。現在、`RefreshInterval` の作成のみをサポートしています。

- `SourceProperties` – キーバリューペアのマップ配列。

  各キーは UTF-8 文字列。

  各値は UTF-8 文字列。

  統合ソースの追加プロパティを指定するキーと値のペアのコレクション。これらのプロパティにより、データ統合操作中に ODB ソースの動作をカスタマイズするために使用できる設定オプションが提供されます。

IntegrationPartition 構造

ターゲットでのデータのパーティション化の方法を説明する構造。

フィールド

- `FieldName` – UTF-8 文字列、1 ~ 128 バイト長。

ターゲットでのデータのパーティション化に使用されるフィールド名。各行に一意の値を持つ列 (「LastModifiedTimestamp」や「SystemModTimeStamp」など) をパーティション列として使用することは避けてください。このような列は、多数の小さなパーティションを作成するため、パーティション分割には適していません。パフォーマンスの問題が発生する可能性があります。

- `FunctionSpec` – UTF-8 文字列、1 ~ 128 バイト長。

  ターゲットのデータをパーティション分割するために使用する関数を指定します。このパラメータで許可される唯一の値は "identity" (文字列) です。"identity" 関数は、ターゲット上のデータパーティション分割がソースと同じスキームに従うことを保証します。したがって、ソースデータのパーティション構造がターゲット送信先に保持されます。

- `ConversionSpec` – UTF-8 文字列、1 ~ 128 バイト長。

  ソースデータのタイムスタンプ形式を指定します。次の値を指定できます:

  - `epoch_sec` – UNIX エポックタイムスタンプ (秒単位)
  - `epoch_milli` – UNIX エポックタイムスタンプ (ミリ秒単位)
  - `iso` – ISO 8601 形式のタイムスタンプ

  > **Note**
  >
  > タイムスタンプベースのパーティション関数 (年、月、日、時間) を使用するときにのみ `ConversionSpec` を指定します。AWS Glueゼロ ETL はこのパラメータを使用し、パーティショニング前にソースデータをタイムスタンプ形式に正しく変換します。
  > `identity` パーティション関数を持つ高カーディナリティ列を使用しないでください。高カーディナリティ列には次のものが含まれます。
  >
  > - プライマリキー
  > - タイムスタンプフィールド (`LastModifiedTimestamp` や `CreatedDate` など)
  > - システム生成されたタイムスタンプ
  >
  > ID パーティショニングを持つ高カーディナリティ列を使用すると、小さなパーティションが多く作成され、取り込みパフォーマンスが大幅に低下する可能性があります。

IntegrationError 構造

ゼロ ETL 統合に関連するエラー。

フィールド

- ErrorCode – UTF-8 文字列、1 ~ 128 バイト長。

  このエラーに関連付けられたコード。
- ErrorMessage – UTF-8 文字列。1 ~ 2048 バイト長。

  エラーを説明するメッセージ。

IntegrationFilter 構造

DescribeIntegrations リクエストを呼び出すときに使用できるフィルター。

フィールド

- Name – UTF-8 文字列、1 ~ 128 バイト長。

  フィルターの名前。
- Values – UTF-8 文字列の配列。

  フィルタ値のリスト。

InboundIntegration 構造

リソースにデータを書き込む統合の構造。

フィールド

- SourceArn – 必須: UTF-8 文字列。1 ~ 128 バイト長。

  統合のソースリソースの ARN。
- TargetArn – 必須: UTF-8 文字列。1 ~ 128 バイト長。

  統合のターゲットリソースの ARN。
- IntegrationArn – 必須: UTF-8 文字列。1 ~ 128 バイト長。

  ゼロ ETL 統合の ARN。
- Status – 必須: UTF-8 文字列 (有効な値: CREATING | ACTIVE | MODIFYING | FAILED | DELETING | SYNCING | NEEDS_ATTENTION)。

次のようなステータスがあります。

- CREATING: 統合を作成中です。
- ACTIVE: 統合の作成が完了しました。
- MODIFYING: 統合を変更中です。
- FAILED: 統合を作成できませんでした。
- DELETING: 統合が削除されました。
- SYNCING: 統合を同期中です。
- NEEDS_ATTENTION: 統合では同期などに注意を払う必要があります。

- `CreateTime` – 必須: タイムスタンプ。

  統合が作成された時刻 (UTC)。

- `IntegrationConfig` – IntegrationConfig オブジェクト。

  統合に関連付けられたプロパティ。

- `Errors` – IntegrationError オブジェクトの配列。

  統合に関連するエラーのリスト。

SourceProcessingProperties 構造

統合ソースに関連付けられたリソースプロパティ。

フィールド

- `RoleArn` – UTF-8 文字列、1 ~ 128 バイト長。

  AWS Glue 接続にアクセスするための IAM ロール。

TargetProcessingProperties 構造

統合ターゲットに関連付けられたリソースプロパティ。

フィールド

- `RoleArn` – UTF-8 文字列、1 ~ 128 バイト長。

  AWS Glue データベースにアクセスするための IAM ロール。

- KmsArn – UTF-8 文字列。1 ~ 2,048 バイト長。

  暗号化に使用される KMS キーの ARN。

- ConnectionName – UTF-8 文字列、1 ~ 128 バイト長。

  カスタマー VPC で実行されている AWS Glue ジョブを設定する AWS Glue ネットワーク接続。

- EventBusArn – UTF-8 文字列。1 ~ 2048 バイト長。

  統合ステータスの通知を受け取る Eventbridge イベントバスの ARN。

SourceTableConfig 構造

ソースレッグがソースからのデータを処理する際に使用するプロパティ。

フィールド

- Fields – UTF-8 文字列の配列。

  列レベルのフィルタリングに使用されるフィールドのリスト。現在サポートされていません。

- FilterPredicate – UTF-8 文字列、1 ~ 128 バイト長。

  行レベルのフィルタリングに使用される条件句。現在サポートされていません。

- PrimaryKey – UTF-8 文字列の配列。

  このテーブルのプライマリキーセットを指定します。現在、リクエストに応じて SAP Entity0f エンティティ専用にサポートされています。AWS サポートに連絡して、この機能を利用できるようにします。

- RecordUpdateField – UTF-8 文字列、1 ~ 128 バイト長。

  増分プルのタイムスタンプベースのフィールド。現在サポートされていません。

TargetTableConfig 構造

ターゲットのデータをパーティション化するためにターゲットレッグで使用されるプロパティ。

フィールド

- UnnestSpec – UTF-8 文字列 (有効な値: TOPLEVEL | FULL | NOUNNEST)。

ネストされたオブジェクトを最上位の要素に平板化する方法を指定します。有効な値は "TOPLEVEL"、"FULL"、"NOUNNEST" のいずれかです。

- `PartitionSpec` – IntegrationPartition オブジェクトの配列。

  ターゲットのファイルレイアウトを決定します。

- `TargetTableName` – UTF-8 文字列、1 ~ 128 バイト長。

  ターゲットテーブルのオプションの名前です。

操作

- CreateIntegration アクション (Python: create_integration)
- ModifyIntegration アクション (Python: modify_integration)
- DescribeIntegrations アクション (Python: describe_integrations)
- DeleteIntegration アクション (Python: delete_integration)
- DescribeInboundIntegrations アクション (Python: describe_inbound_integrations)
- CreateIntegrationTableProperties アクション (Python: create_integration_table_properties)
- UpdateIntegrationTableProperties アクション (Python: update_integration_table_properties)
- GetIntegrationTableProperties アクション (Python: get_integration_table_properties)
- DeleteIntegrationTableProperties アクション (Python: delete_integration_table_properties)
- CreateIntegrationResourceProperty アクション (Python: create_integration_resource_property)
- UpdateIntegrationResourceProperty アクション (Python: update_integration_resource_property)
- GetIntegrationResourceProperty アクション (Python: get_integration_resource_property)
- UntagResource アクション (Python: untag_resource)
- ListTagsForResource アクション (Python: list_tags_for_resource)

CreateIntegration アクション (Python: create_integration)

Amazon リソースネーム (ARN) を持つ 2 つのリソースである `SourceArn` と `TargetArn` の間で、発信者のアカウントにゼロ ETL 統合を作成します。

リクエスト

- `IntegrationName` – 必須: UTF-8 文字列。1 ~ 128 バイト長。

AWS Glue での統合の一意の名前。

- SourceArn – 必須: UTF-8 文字列。1 ~ 128 バイト長。

  統合のソースリソースの ARN。

- TargetArn – 必須: UTF-8 文字列。1 ~ 128 バイト長。

  統合のターゲットリソースの ARN。

- Description – UTF-8 文字列。1,000 バイト長以下。Custom string pattern #12 に一致。

  統合の説明。

- DataFilter – UTF-8 文字列。1 ~ 2,048 バイト長。

  Maxwell フィルター構文を使用して統合のソーステーブルを選択します。

- KmsKeyId – UTF-8 文字列。1 ~ 2,048 バイト長。

  チャネルの暗号化に使用される KMS キーの ARN。

- AdditionalEncryptionContext – キーバリューペアのマップ配列。

  各キーは UTF-8 文字列。

  各値は UTF-8 文字列。

  暗号化のための追加のコンテキスト情報を含む、シークレット以外のキーと値のペアのオプションセット。これは KMSKeyId が指定されている場合にのみ提供されます。

- Tags – Tag オブジェクトの配列。

  キーと値のペアのリストで構成されるリソースに割り当てられるメタデータ。

- IntegrationConfig – IntegrationConfig オブジェクト。

  構成設定。

レスポンス

- SourceArn – 必須: UTF-8 文字列。1 ~ 128 バイト長。

  統合のソースリソースの ARN。

- TargetArn – 必須: UTF-8 文字列。1 ~ 128 バイト長。

統合のターゲットリソースの ARN。

- `IntegrationName` – 必須: UTF-8 文字列。1 ~ 128 バイト長。

  AWS Glue での統合の一意の名前。

- `Description` - UTF-8 文字列。1,000 バイト長以下。Custom string pattern #12 に一致。

  統合の説明。

- `IntegrationArn` – 必須: UTF-8 文字列。1 ~ 128 バイト長。

  作成した統合の Amazon リソースネーム (ARN)。

- `KmsKeyId` – UTF-8 文字列。1 ~ 2,048 バイト長。

  チャネルの暗号化に使用される KMS キーの ARN。

- `AdditionalEncryptionContext` – キーバリューペアのマップ配列。

  各キーは UTF-8 文字列。

  各値は UTF-8 文字列。

  暗号化のための追加のコンテキスト情報を含む、シークレット以外のキーと値のペアのオプションセット。

- `Tags` – Tag オブジェクトの配列。

  キーと値のペアのリストで構成されるリソースに割り当てられるメタデータ。

- `Status` – 必須: UTF-8 文字列 (有効な値: `CREATING` | `ACTIVE` | `MODIFYING` | `FAILED` | `DELETING` | `SYNCING` | `NEEDS_ATTENTION`)。

  作成中の統合のステータス。

  次のようなステータスがあります。

  - CREATING: 統合を作成中です。
  - ACTIVE: 統合の作成が完了しました。
  - MODIFYING: 統合を変更中です。
  - FAILED: 統合を作成できませんでした。
  - DELETING: 統合が削除されました。
  - SYNCING: 統合を同期中です。

  - NEEDS_ATTENTION: 統合では同期などに注意を払う必要があります。
- `CreateTime` – 必須: タイムスタンプ。

  統合が作成された時刻 (UTC)。
- `Errors` – IntegrationError オブジェクトの配列。

  統合の作成に関連するエラーのリスト。
- `DataFilter` – UTF-8 文字列、1 ~ 2048 バイト長。

  Maxwell フィルター構文を使用して統合のソーステーブルを選択します。
- `IntegrationConfig` – IntegrationConfig オブジェクト。

  構成設定。

エラー

- `ValidationException`
- `AccessDeniedException`
- `ResourceNotFoundException`
- `InternalServerException`
- `IntegrationConflictOperationFault`
- `IntegrationQuotaExceededFault`
- `KMSKeyNotAccessibleFault`
- `EntityNotFoundException`
- `InternalServiceException`
- `ConflictException`
- `ResourceNumberLimitExceededException`
- `InvalidInputException`

ModifyIntegration アクション (Python: modify_integration)

発信者のアカウントでゼロ ETL 統合を変更します。

リクエスト

- `IntegrationIdentifier` – 必須: UTF-8 文字列。1 ~ 128 バイト長。

  統合の Amazon リソースネーム (ARN)。
- `Description` – UTF-8 文字列。1,000 バイト長以下。Custom string pattern #12 に一致。

  統合の説明。
- `DataFilter` – UTF-8 文字列。1 ~ 2,048 バイト長。

  Maxwell フィルター構文を使用して統合のソーステーブルを選択します。
- `IntegrationName` – UTF-8 文字列、1 ~ 128 バイト長。

  AWS Glue での統合の一意の名前。

レスポンス

- `SourceArn` – 必須: UTF-8 文字列。1 ~ 128 バイト長。

  統合のソースの ARN。
- `TargetArn` – 必須: UTF-8 文字列。1 ~ 128 バイト長。

  統合のターゲットの ARN。
- `IntegrationName` – 必須: UTF-8 文字列。1 ~ 128 バイト長。

  AWS Glue での統合の一意の名前。
- `Description` - UTF-8 文字列。1,000 バイト長以下。Custom string pattern #12 に一致。

  統合の説明。
- `IntegrationArn` – 必須: UTF-8 文字列。1 ~ 128 バイト長。

  統合の Amazon リソースネーム (ARN)。
- `KmsKeyId` – UTF-8 文字列。1 ~ 2,048 バイト長。

  チャネルの暗号化に使用される KMS キーの ARN。
- `AdditionalEncryptionContext` – キーバリューペアのマップ配列。

  各キーは UTF-8 文字列。

各値は UTF-8 文字列。

暗号化のための追加のコンテキスト情報を含む、シークレット以外のキーと値のペアのオプションセット。

- `Tags` – Tag オブジェクトの配列。

  キーと値のペアのリストで構成されるリソースに割り当てられるメタデータ。

- `Status` – 必須: UTF-8 文字列 (有効な値: `CREATING` | `ACTIVE` | `MODIFYING` | `FAILED` | `DELETING` | `SYNCING` | `NEEDS_ATTENTION`)。

  変更中の統合のステータス。

  次のようなステータスがあります。

  - CREATING: 統合を作成中です。
  - ACTIVE: 統合の作成が完了しました。
  - MODIFYING: 統合を変更中です。
  - FAILED: 統合を作成できませんでした。
  - DELETING: 統合が削除されました。
  - SYNCING: 統合を同期中です。
  - NEEDS_ATTENTION: 統合では同期などに注意を払う必要があります。

- `CreateTime` – 必須: タイムスタンプ。

  統合が作成された時刻 (UTC)。

- `Errors` – IntegrationError オブジェクトの配列。

  統合の変更に関連するエラーのリスト。

- `DataFilter` – UTF-8 文字列。1 ~ 2,048 バイト長。

  Maxwell フィルター構文を使用して統合のソーステーブルを選択します。

エラー

- `ValidationException`
- `AccessDeniedException`
- `InternalServerException`

- IntegrationNotFoundFault
- IntegrationConflictOperationFault
- InvalidIntegrationStateFault
- EntityNotFoundException
- InternalServiceException
- ConflictException
- InvalidStateException
- InvalidInputException

DescribeIntegrations アクション (Python: describe_integrations)

この API は統合のリストを取得するために使用されます。

リクエスト

- IntegrationIdentifier – UTF-8 文字列、1 ~ 128 バイト長。

  統合の Amazon リソースネーム (ARN)。
- Marker – UTF-8 文字列、1 ~ 128 バイト長。

  後続のリクエストにおけるレスポンスレコードの次回のセット開始点を示す値。
- MaxRecords – 数値 (整数)。

  出力で返される項目の総数。
- Filters – IntegrationFilter オブジェクトの配列。

  結果をフィルタリングするためのキーと値のリスト。サポートされているキーは "Status"、"IntegrationName"、"SourceArn" です。IntegrationName は 1 つの値のみに制限されています。

応答

- Integrations – Integration オブジェクトの配列。

  ゼロ ETL 統合のリスト。
- Marker – UTF-8 文字列、1 ~ 128 バイト長。

後続のリクエストにおけるレスポンスレコードの次回のセット開始点を示す値。

エラー

- `ValidationException`
- `AccessDeniedException`
- `InternalServerException`
- `IntegrationNotFoundFault`
- `EntityNotFoundException`
- `InternalServiceException`
- `InvalidInputException`

DeleteIntegration アクション (Python: delete_integration)

指定されたゼロ ETL 統合を削除します。

リクエスト

- `IntegrationIdentifier` – 必須: UTF-8 文字列。1 ~ 128 バイト長。

  統合の Amazon リソースネーム (ARN)。

レスポンス

- `SourceArn` – 必須: UTF-8 文字列。1 ~ 128 バイト長。

  統合のソースの ARN。
- `TargetArn` – 必須: UTF-8 文字列。1 ~ 128 バイト長。

  統合のターゲットの ARN。
- `IntegrationName` – 必須: UTF-8 文字列。1 ~ 128 バイト長。

  AWS Glue での統合の一意の名前。
- `Description` - UTF-8 文字列。1,000 バイト長以下。Custom string pattern #12 に一致。

  統合の説明。

- `IntegrationArn` – 必須: UTF-8 文字列。1 ~ 128 バイト長。

  統合の Amazon リソースネーム (ARN)。

- `KmsKeyId` – UTF-8 文字列。1 ~ 2,048 バイト長。

  チャネルの暗号化に使用される KMS キーの ARN。

- `AdditionalEncryptionContext` – キーバリューペアのマップ配列。

  各キーは UTF-8 文字列。

  各値は UTF-8 文字列。

  暗号化のための追加のコンテキスト情報を含む、シークレット以外のキーと値のペアのオプションセット。

- `Tags` – Tag オブジェクトの配列。

  キーと値のペアのリストで構成されるリソースに割り当てられるメタデータ。

- `Status` – 必須: UTF-8 文字列 (有効な値: `CREATING` | `ACTIVE` | `MODIFYING` | `FAILED` | `DELETING` | `SYNCING` | `NEEDS_ATTENTION`)。

  削除される統合のステータス。

  次のようなステータスがあります。

  - CREATING: 統合を作成中です。
  - ACTIVE: 統合の作成が完了しました。
  - MODIFYING: 統合を変更中です。
  - FAILED: 統合を作成できませんでした。
  - DELETING: 統合が削除されました。
  - SYNCING: 統合を同期中です。
  - NEEDS_ATTENTION: 統合では同期などに注意を払う必要があります。

- `CreateTime` – 必須: タイムスタンプ。

  統合が作成された時刻 (UTC)。

- `Errors` – IntegrationError オブジェクトの配列。

  統合に関連するエラーのリスト。

- `DataFilter` – UTF-8 文字列。1 ~ 2,048 バイト長。

Maxwell フィルター構文を使用して統合のソーステーブルを選択します。

エラー

- ValidationException
- AccessDeniedException
- InternalServerException
- IntegrationNotFoundFault
- IntegrationConflictOperationFault
- InvalidIntegrationStateFault
- EntityNotFoundException
- InternalServiceException
- ConflictException
- InvalidStateException
- InvalidInputException

DescribeInboundIntegrations アクション (Python: describe_inbound_integrations)

指定された統合のインバウンド統合のリストを返します。

リクエスト

- IntegrationArn – UTF-8 文字列、1 ~ 128 バイト長。

  統合の Amazon リソースネーム (ARN)。
- Marker – UTF-8 文字列、1 ~ 128 バイト長。

  ページ分割を始める場所を指定するトークン。こちらは以前に切り捨てられた応答のマーカーです。
- MaxRecords – 数値 (整数)。

  出力で返される項目の総数。
- TargetArn – UTF-8 文字列、1 ~ 128 バイト長。

統合内のターゲットリソースの Amazon リソースネーム (ARN)。

応答

- InboundIntegrations – InboundIntegration オブジェクトの配列。

  インバウンド統合のリスト。
- Marker – UTF-8 文字列、1 ~ 128 バイト長。

  後続のリクエストにおけるレスポンスレコードの次回のセット開始点を示す値。

エラー

- ValidationException
- AccessDeniedException
- InternalServerException
- IntegrationNotFoundFault
- TargetResourceNotFound
- OperationNotSupportedException
- EntityNotFoundException
- InternalServiceException
- InvalidInputException

CreateIntegrationTableProperties アクション (Python: create_integration_table_properties)

この API は、レプリケートする必要があるテーブルに対してオプションのオーバーライドプロパティを提供するために使用されます。これらのプロパティには、ソーステーブルとターゲットテーブルのフィルタリングとパーティション化のためのプロパティを含めることができます。ソースとターゲットの両方のプロパティを設定するには、AWS Glue 接続 ARN を SourceTableConfig で ResourceArn として、AWS Glue データベース ARN を TargetTableConfig で ResourceArn としてそれぞれ指定し、同じ API を呼び出す必要があります。

リクエスト

- ResourceArn – 必須: UTF-8 文字列。1 ~ 128 バイト長。

  統合テーブルプロパティを作成するターゲットテーブルの Amazon リソースネーム (ARN)。現在、この API はターゲットテーブルの統合テーブルプロパティ作成のみをサポートしており、提供される ARN は AWS Glue データカタログのターゲットテーブルの ARN である必要があります。ソース接続の統合テーブルプロパティの作成によるサポート (接続 ARN の使用) はまだ実装されておらず、今後のリリースに追加されます。

- TableName – 必須: UTF-8 文字列。1 ~ 128 バイト長。

  レプリケートするテーブルの名前。

- SourceTableConfig – SourceTableConfig オブジェクト。

  ソーステーブル設定の構造。サポートされているソースプロパティのリストについては、SourceTableConfig 構造を参照してください。

- TargetTableConfig – TargetTableConfig オブジェクト。

  ターゲットテーブル設定の構造。

レスポンス

- 応答パラメータはありません。

エラー

- ValidationException
- AccessDeniedException
- ResourceNotFoundException
- InternalServerException
- EntityNotFoundException
- InternalServiceException
- InvalidInputException

UpdateIntegrationTableProperties アクション (Python: update_integration_table_properties)

この API は、レプリケートする必要があるテーブルに対してオプションのオーバーライドプロパティを提供するために使用されます。これらのプロパティには、ソーステーブルとターゲットテーブルのフィルタリングとパーティション化のためのプロパティを含めることができます。ソースとターゲットの両方のプロパティを設定するには、AWS Glue 接続 ARN を `SourceTableConfig` で `ResourceArn` として、AWS Glue データベース ARN を `TargetTableConfig` で `ResourceArn` としてそれぞれ指定し、同じ API を呼び出す必要があります。

オーバーライドは、同じ `ResourceArn` とソーステーブルを使用するすべての統合に反映されます。

リクエスト

- `ResourceArn` – 必須: UTF-8 文字列。1 ~ 128 バイト長。

  ソースの接続 ARN、またはターゲットのデータベース ARN。
- `TableName` – 必須: UTF-8 文字列。1 ~ 128 バイト長。

  レプリケートするテーブルの名前。
- `SourceTableConfig` – SourceTableConfig オブジェクト。

  ソーステーブル設定の構造。
- `TargetTableConfig` – TargetTableConfig オブジェクト。

  ターゲットテーブル設定の構造。

レスポンス

- 応答パラメータはありません。

エラー

- `ValidationException`
- `AccessDeniedException`
- `ResourceNotFoundException`
- `InternalServerException`

- EntityNotFoundException
- InternalServiceException
- InvalidInputException

GetIntegrationTableProperties アクション (Python: get_integration_table_properties)

この API は、レプリケートする必要があるテーブルに対してオプションのオーバーライドプロパティを取得するために使用されます。これらのプロパティには、ソーステーブルとターゲットテーブルのフィルタリングとパーティションのためのプロパティを含めることができます。

リクエスト

- ResourceArn – 必須: UTF-8 文字列。1 ~ 128 バイト長。

  統合テーブルプロパティを取得するターゲットテーブルの Amazon リソースネーム (ARN)。現在、この API はターゲットテーブルのプロパティ取得のみをサポートしており、提供される ARN は AWS Glue データカタログのターゲットテーブルの ARN である必要があります。ソース接続の統合テーブルプロパティの取得によるサポート (接続 ARN の使用) はまだ実装されておらず、今後のリリースに追加されます。
- TableName – 必須: UTF-8 文字列。1 ~ 128 バイト長。

  レプリケートするテーブルの名前。

レスポンス

- ResourceArn – UTF-8 文字列、1 ~ 128 バイト長。

  統合テーブルプロパティを取得するターゲットテーブルの Amazon リソースネーム (ARN)。現在、この API はターゲットテーブルのプロパティ取得のみをサポートしており、提供される ARN は AWS Glue データカタログのターゲットテーブルの ARN である必要があります。ソース接続の統合テーブルプロパティの取得によるサポート (接続 ARN の使用) はまだ実装されておらず、今後のリリースに追加されます。
- TableName – UTF-8 文字列、1 ~ 128 バイト長。

  レプリケートするテーブルの名前。
- SourceTableConfig – SourceTableConfig オブジェクト。

ソーステーブル設定の構造。

- `TargetTableConfig` – TargetTableConfig オブジェクト。

  ターゲットテーブル設定の構造。

エラー

- `ValidationException`
- `AccessDeniedException`
- `ResourceNotFoundException`
- `InternalServerException`
- `EntityNotFoundException`
- `InternalServiceException`
- `InvalidInputException`

DeleteIntegrationTableProperties アクション (Python: delete_integration_table_properties)

レプリケートする必要があるテーブル用に作成されたテーブルプロパティを削除します。

リクエスト

- `ResourceArn` – 必須: UTF-8 文字列。1 ~ 128 バイト長。

  ソースの接続 ARN、またはターゲットのデータベース ARN。
- `TableName` – 必須: UTF-8 文字列。1 ~ 128 バイト長。

  レプリケートするテーブルの名前。

レスポンス

- 応答パラメータはありません。

エラー

- ValidationException
- AccessDeniedException
- ResourceNotFoundException
- InternalServerException
- EntityNotFoundException
- InternalServiceException
- InvalidInputException

CreateIntegrationResourceProperty アクション (Python: create_integration_resource_property)

この API は、AWS Glue 接続の ResourceProperty (ソースの場合) または AWS Glue データベース ARN (ターゲットの場合) をセットアップするために使用できます。これらのプロパティには、接続またはデータベースにアクセスするためのロールを含めることができます。ソースとターゲットの両方のプロパティを設定するには、AWS Glue 接続 ARN を SourceProcessingProperties で ResourceArn として、AWS Glue データベース ARN を TargetProcessingProperties で ResourceArn としてそれぞれ指定し、同じ API を呼び出す必要があります。

リクエスト

- ResourceArn – 必須: UTF-8 文字列。1 ~ 128 バイト長。

  ソースの接続 ARN、またはターゲットのデータベース ARN。
- SourceProcessingProperties – SourceProcessingProperties オブジェクト。

  統合ソースに関連付けられたリソースプロパティ。
- TargetProcessingProperties – TargetProcessingProperties オブジェクト。

  統合ターゲットに関連付けられたリソースプロパティ。

レスポンス

- ResourceArn – 必須: UTF-8 文字列。1 ~ 128 バイト長。

  ソースの接続 ARN、またはターゲットのデータベース ARN。

- SourceProcessingProperties – SourceProcessingProperties オブジェクト。

  統合ソースに関連付けられたリソースプロパティ。
- TargetProcessingProperties – TargetProcessingProperties オブジェクト。

  統合ターゲットに関連付けられたリソースプロパティ。

エラー

- ValidationException
- AccessDeniedException
- ConflictException
- InternalServerException
- ResourceNotFoundException
- EntityNotFoundException
- InternalServiceException
- InvalidInputException

UpdateIntegrationResourceProperty アクション (Python: update_integration_resource_property)

この API は、AWS Glue 接続の ResourceProperty (ソースの場合) または AWS Glue データベース ARN (ターゲットの場合) を更新するために使用できます。これらのプロパティには、接続またはデータベースにアクセスするためのロールを含めることができます。同じリソースを複数の統合で使用できるため、リソースプロパティを更新すると、そのリソースを使用しているすべての統合に影響します。

リクエスト

- ResourceArn – 必須: UTF-8 文字列。1 ~ 128 バイト長。

  ソースの接続 ARN、またはターゲットのデータベース ARN。
- SourceProcessingProperties – SourceProcessingProperties オブジェクト。

  統合ソースに関連付けられたリソースプロパティ。
- TargetProcessingProperties – TargetProcessingProperties オブジェクト。

統合ターゲットに関連付けられたリソースプロパティ。

レスポンス

- `ResourceArn` – UTF-8 文字列、1 ~ 128 バイト長。

  ソースの接続 ARN、またはターゲットのデータベース ARN。
- `SourceProcessingProperties` – SourceProcessingProperties オブジェクト。

  統合ソースに関連付けられたリソースプロパティ。
- `TargetProcessingProperties` – TargetProcessingProperties オブジェクト。

  統合ターゲットに関連付けられたリソースプロパティ。

エラー

- `ValidationException`
- `AccessDeniedException`
- `InternalServerException`
- `ResourceNotFoundException`
- `EntityNotFoundException`
- `InternalServiceException`
- `InvalidInputException`

GetIntegrationResourceProperty アクション (Python: get_integration_resource_property)

この API は、AWS Glue 接続の `ResourceProperty` (ソースの場合) または AWS Glue データベース ARN (ターゲットの場合) の取得に使用されます。

リクエスト

- `ResourceArn` – 必須: UTF-8 文字列。1 ~ 128 バイト長。

  ソースの接続 ARN、またはターゲットのデータベース ARN。

レスポンス

- ResourceArn – UTF-8 文字列、1 ~ 128 バイト長。

  ソースの接続 ARN、またはターゲットのデータベース ARN。
- SourceProcessingProperties – SourceProcessingProperties オブジェクト。

  統合ソースに関連付けられたリソースプロパティ。
- TargetProcessingProperties – TargetProcessingProperties オブジェクト。

  統合ターゲットに関連付けられたリソースプロパティ。

エラー

- ValidationException
- AccessDeniedException
- InternalServerException
- ResourceNotFoundException
- EntityNotFoundException
- InternalServiceException
- InvalidInputException

UntagResource アクション (Python: untag_resource)

統合リソースから指定したタグを削除します。

リクエスト

- ResourceArn – 必須: UTF-8 文字列。1 ~ 10,240 バイト長。Custom string pattern #49 に一致。

  統合リソースの Amazon リソースネーム (ARN)。
- TagsToRemove – 必須: UTF-8 文字列の配列。文字列 50 個以下。

  リソースから削除するメタデータタグのリスト。

レスポンス

- 応答パラメータはありません。

エラー

- `ResourceNotFoundException`

ListTagsForResource アクション (Python: list_tags_for_resource)

指定したリソースに割り当てたメタデータタグの一覧。

リクエスト

- `ResourceARN` – 必須: UTF-8 文字列。1 ~ 10,240 バイト長。Custom string pattern #49 に一致。

  リソースのリソース ARN。

レスポンス

- `Tags` – Tag オブジェクトの配列。構造体 10 個以下。

  タグのリスト。

エラー

- `ResourceNotFoundException`

例外

- ResourceNotFoundException structure
- InternalServerException 構造
- IntegrationAlreadyExistsFault 構造
- IntegrationConflictOperationFault 構造
- IntegrationQuotaExceededFault 構造
- KMSKeyNotAccessibleFault (構造)
- IntegrationNotFoundFault 構造

- TargetResourceNotFound 構造
- InvalidIntegrationStateFault 構造

ResourceNotFoundException structure

リソースが見つかりませんでした。

フィールド

- Message – UTF-8 文字列。

  問題を説明するメッセージ。

InternalServerException 構造

内部サーバーエラーが発生しました。

フィールド

- Message – UTF-8 文字列。

  問題を説明するメッセージ。

IntegrationAlreadyExistsFault 構造

指定された統合は既に存在します。

フィールド

- Message – UTF-8 文字列。

  問題を説明するメッセージ。

IntegrationConflictOperationFault 構造

リクエストされたオペレーションは別のオペレーションと競合しています。

フィールド

- Message – UTF-8 文字列。

問題を説明するメッセージ。

IntegrationQuotaExceededFault 構造

統合によって処理されたデータがクォータを超えました。

フィールド

- Message – UTF-8 文字列。

  問題を説明するメッセージ。

KMSKeyNotAccessibleFault (構造)

指定された KMS キーにアクセスできません。

フィールド

- Message – UTF-8 文字列。

  問題を説明するメッセージ。

IntegrationNotFoundFault 構造

指定された統合が見つかりませんでした。

フィールド

- Message – UTF-8 文字列。

  問題を説明するメッセージ。

TargetResourceNotFound 構造

ターゲットリソースが見つかりませんでした。

フィールド

- Message – UTF-8 文字列。

問題を説明するメッセージ。

InvalidIntegrationStateFault 構造

統合が無効な状態です。

フィールド

- `Message` – UTF-8 文字列。

  問題を説明するメッセージ。

インタラクティブセッション API

インタラクティブセッション API は、AWS Glue インタラクティブセッションを使用して、データ統合に向けた抽出、変換、ロード (ETL) スクリプトを構築およびテストする際に関連する AWS Glue API について説明します。

データ型

- セッション構造
- SessionCommand の構造
- Statement 構造
- StatementOutput 構造
- StatementOutputData 構造
- ConnectionsList 構造

セッション構造

リモート Spark ランタイム環境が実行されている期間。

フィールド

- `Id` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  セッションの ID
- `CreatedOn` – タイムスタンプ。

セッションが作成された日時。

- `Status` - UTF-8 文字列 (有効値: `PROVISIONING` | `READY` | `FAILED` | `TIMEOUT` | `STOPPING` | `STOPPED`).。

  セッションのステータスです。

- `ErrorMessage` – 説明文字列。2048 バイト長以下。URI address multi-line string pattern に一致。

  セッション中に表示されるエラーメッセージです。

- `Description` – 説明文字列。2048 バイト長以下。URI address multi-line string pattern に一致。

  セッションの説明です。

- `Role` - UTF-8 文字列。20 ~ 2,048 バイト長。Custom string pattern #30 に一致。

  セッションに関連付けられている IAM ロールの名前または Amazon リソースネーム (ARN)。

- `Command` – SessionCommand オブジェクト。

  コマンドオブジェクト。「sessionCommand」を参照してください。

- `DefaultArguments` – キーバリューペアのマップ配列。75 ペア以下。

  各キーは UTF-8 文字列、1 ~ 128 バイト長で、Custom string pattern #31 に一致します。

  各値は UTF-8 文字列、4,096 バイト長で、URI address multi-line string pattern に一致します。

  キーバリューペアのマップ配列。最大ペア数は 75 です。

- `Connections` – ConnectionsList オブジェクト。

  セッションに使用される接続の数。

- `Progress` – 数値 (double)。

  セッションのコード実行の進行状況。

- `MaxCapacity` – 数値 (double)。

  ジョブの実行に割り当てることができる AWS Glue データ処理ユニット (DPU) の数。DPU は処理能力を相対的に測定するもので、4 個の vCPU のコンピューティング性能と 16 GB のメモリで構成されています。

- `SecurityConfiguration` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

セッションで使用される SecurityConfiguration 構造の名前。

- `GlueVersion` – UTF-8 文字列。1 ~ 255 バイト長。Custom string pattern #47 に一致。

  AWS Glueバージョンにより、AWS Glueがサポートする Apache Spark と Python のバージョンが決定されます。GlueVersion は 2.0 以上である必要があります。

- `DataAccessId` – UTF-8 文字列、1 ~ 36 バイト。

  セッションのデータアクセス ID。

- `PartitionId` – UTF-8 文字列、1 ~ 36 バイト。

  セッションのパーティション ID。

- `NumberOfWorkers` – 数値 (整数)。

  セッションに使用する、定義済み `WorkerType` のワーカー数。

- `WorkerType` – UTF-8 文字列 (有効な値: `Standard=""` | `G.1X=""` | `G.2X=""` | `G.025X=""` | `G.4X=""` | `G.8X=""` | `Z.2X=""`)。

  セッションの実行時に割り当てられる事前定義済みのワーカーの種類。Spark セッション用に `G.1X`、`G.2X`、`G.4X`、または `G.8X` の値を使用できます。Ray セッション用に 値 `Z.2X` を使用できます。

- `CompletedOn` – タイムスタンプ。

  このセッションが完了した日付と時刻。

- `ExecutionTime` – 数値 (double)。

  セッションの合計実行時間。

- `DPUSeconds` – 数値 (double)。

  セッションによって消費された DPU (式: ExecutionTime * MaxCapacity)。

- `IdleTimeout` – 数値 (整数)。

  セッションがタイムアウトするまでのアイドル時の分数。

- `ProfileName` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  セッションに関連付けられた AWS Glue 使用プロフィールの名前。

SessionCommand の構造

ジョブを実行する SessionCommand。

フィールド

- Name – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  SessionCommand の名前を指定します。「glueetl」または「gluestreaming」を使用できます。
- PythonVersion– UTF-8 文字列。Custom string pattern #48 に一致。

  Python バージョンを指定します。Python バージョンは、Spark タイプのジョブでサポートされるバージョンを示します。

Statement 構造

セッションで特定のアクションを発生させるためのステートメントまたはリクエスト。

フィールド

- Id – 数値 (整数)。

  ステートメントの ID。
- Code – UTF–8 文字列。

  ステートメントの実行コード。
- State - UTF-8 文字列 (有効値: WAITING | RUNNING | AVAILABLE | CANCELLING | CANCELLED | ERROR).。

  リクエストがアクションされている間の状態。
- Output – StatementOutput オブジェクト。

  JSON 形式での出力。
- Progress – 数値 (double)。

  コード実行の進行状況。
- StartedOn – 数値 (long)。

  ジョブ定義が開始された UNIX の日時。

- `CompletedOn` – 数値 (long)。

  ジョブ定義が完了した UNIX の日時。

StatementOutput 構造

JSON 形式のコード実行の出力。

フィールド

- `Data` – StatementOutputData オブジェクト。

  コード実行の出力。
- `ExecutionCount` – 数値 (整数)。

  出力の実行回数。
- `Status` - UTF-8 文字列 (有効値: `WAITING` | `RUNNING` | `AVAILABLE` | `CANCELLING` | `CANCELLED` | `ERROR`).。

  コード実行の出力状態。
- `ErrorName` – UTF–8 文字列。

  出力内エラーの名前。
- `ErrorValue` – UTF–8 文字列。

  出力のエラー値。
- `Traceback` – UTF-8 文字列の配列。

  出力のトレースバック。

StatementOutputData 構造

JSON 形式のコード実行の出力。

フィールド

- `TextPlain` – UTF-8 文字列。

  テキスト形式のコード実行の出力。

ConnectionsList 構造

ジョブが使用する接続を指定します。

フィールド

- `Connections` - UTF-8 文字列の配列。文字列 1,000 個以下。

  ジョブが使用する接続のリスト。

操作

- CreateSession アクション (Python: create_session)
- StopSession アクション (Python: stop_session)
- DeleteSession アクション (Python: delete_session)
- GetSession アクション (Python: get_session)
- ListSessions アクション (Python: list_sessions)
- RunStatement アクション (Python: run_statement)
- CancelStatement アクション (Python: cancel_statement)
- GetStatement アクション (Python: get_statement)
- ListStatements アクション (Python: list_statements)

CreateSession アクション (Python: create_session)

新しいセッションを作成します。

リクエスト

新しいセッションの作成をリクエストします。

- `Id` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  セッションリクエストの ID。
- `Description` – 説明文字列。2,048 バイト長以下。URI address multi-line string pattern に一致。

  セッションの説明です。
- `Role` – 必須: UTF-8 文字列。20 ~ 2,048 バイト長。Custom string pattern #30 に一致。

IAM ロール の ARN

- Command – 必須: SessionCommand オブジェクト。

  ジョブを実行する SessionCommand。

- Timeout – 数値 (整数)。1 以上。

  セッションがタイムアウトするまでの時間 (分)。Spark ETL ジョブのデフォルト値は 48 時間 (2,880 分) です。他のジョブタイプについては、ドキュメントを参照してください。

- IdleTimeout – 数値 (整数)。1 以上。

  セッションがタイムアウトするまでのアイドル時の分数。Spark ETL ジョブのデフォルトは、タイムアウト値です。他のジョブタイプについては、ドキュメントを参照してください。

- DefaultArguments – キーバリューペアのマップ配列。75 ペア以下。

  各キーは UTF-8 文字列、1 ~ 128 バイト長で、Custom string pattern #31 に一致します。

  各値は UTF-8 文字列、4,096 バイト長で、URI address multi-line string pattern に一致します。

  キーバリューペアのマップ配列。最大ペア数は 75 です。

- Connections – ConnectionsList オブジェクト。

  セッションに使用する接続の数。

- MaxCapacity – 数値 (double)。

  ジョブの実行に割り当てることができる AWS Glue データ処理ユニット (DPU) の数。DPU は処理能力を相対的に測定するもので、4 個の vCPU のコンピューティング性能と 16 GB のメモリで構成されています。

- NumberOfWorkers – 数値 (整数)。

  セッションに使用する、定義済み WorkerType のワーカー数。

- WorkerType – UTF-8 文字列 (有効な値: Standard="" | G.1X="" | G.2X="" | G.025X="" | G.4X="" | G.8X="" | Z.2X="")。

  ジョブの実行時に割り当てられる事前定義済みのワーカーの種類。Spark ジョブに使用できる値は G.1X、G.2X、G.4X、または G.8X です。Ray ノートブックに使用できる値は Z.2X です。

  - G.1X ワーカータイプでは、各ワーカーは 1 DPU (4 vCPU、16 GB のメモリ、94 GB のディスク) にマッピングされており、ワーカーごとに 1 個のエグゼキューターを提供します。データ変

換、結合、クエリなどのワークロードには、ほとんどのジョブを実行するためのスケーラブルで費用対効果の高い方法として、このワーカータイプをお勧めします。

- G.2X ワーカータイプでは、各ワーカーは 2 DPU (8 vCPU、32 GB のメモリ、138 GB のディスク) にマッピングされており、ワーカーごとに 1 個のエグゼキュターを提供します。データ変換、結合、クエリなどのワークロードには、ほとんどのジョブを実行するためのスケーラブルで費用対効果の高い方法として、このワーカータイプをお勧めします。
- G.4X ワーカータイプでは、各ワーカーは 4 DPU (16 vCPU、64 GB のメモリ、256 GB のディスク) にマッピングされており、ワーカーごとに 1 個のエグゼキュターを提供します。ワークロードに含まれる変換、集約、結合、クエリへの要求が非常に厳しいジョブには、このワーカータイプをお勧めします。このワーカータイプは、AWS Glue バージョン 3.0 以降の Spark ETL ジョブでのみ使用可能で、以下の AWS リージョンで使用できます。米国東部 (オハイオ)、米国東部 (バージニア北部)、米国西部 (オレゴン)、アジアパシフィック (シンガポール)、アジアパシフィック (シドニー)、アジアパシフィック (東京)、カナダ (中部)、欧州 (フランクフルト)、欧州 (アイルランド)、欧州 (ストックホルム)。
- G.8X ワーカータイプでは、各ワーカーは 8 DPU (32 vCPU、128 GB のメモリ、512 GB のディスク) にマッピングされており、ワーカーごとに 1 個のエグゼキュターを提供します。ワークロードに含まれる変換、集約、結合、クエリへの要求が非常に厳しいジョブには、このワーカータイプをお勧めします。このワーカータイプは、AWS Glue バージョン 3.0 以降の Spark ETL ジョブでのみ使用可能で、G.4X ワーカータイプでサポートされているのと同じ AWS リージョンで使用できます。
- Z.2X ワーカータイプでは、各ワーカーは 2 M-DPU (8 vCPU、64 GB のメモリ、128 GB のディスク) にマッピングされており、オートスケーラーに基づき最大 8 個の Ray ワーカーを提供します。

- `SecurityConfiguration` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  セッションで使用される SecurityConfiguration 構造の名前

- `GlueVersion` – UTF-8 文字列。1 ~ 255 バイト長。Custom string pattern #47 に一致。

  AWS Glueバージョンにより、AWS Glueがサポートする Apache Spark と Python のバージョンが決定されます。GlueVersion は 2.0 以上である必要があります。

- `DataAccessId` – UTF-8 文字列、1 ~ 36 バイト。

  セッションのデータアクセス ID。

- `PartitionId` – UTF-8 文字列、1 ~ 36 バイト。

セッションのパーティション ID。

- `Tags` – キーと値のペアのマップ配列。50 ペア以下。

  各キーは UTF-8 文字列で、1 ~ 128 バイト長です。

  各値は UTF-8 文字列で、256 バイト長以下です。

  セッションに属するキーバリューペアのマップ (タグ)。
- `RequestOrigin` – UTF-8 文字列。1 ~ 128 バイト長。Custom string pattern #31 に一致。

  リクエストの送信元。
- `ProfileName` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  セッションに関連付けられた AWS Glue 使用プロフィールの名前。

レスポンス

- `Session` – セッション オブジェクト。

  レスポンス内のセッションオブジェクトを返します。

エラー

- `AccessDeniedException`
- `IdempotentParameterMismatchException`
- `InternalServiceException`
- `OperationTimeoutException`
- `InvalidInputException`
- `ValidationException`
- `AlreadyExistsException`
- `ResourceNumberLimitExceededException`

StopSession アクション (Python: stop_session)

セッションを停止します。

リクエスト

- `Id` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  停止するセッションの ID。
- `RequestOrigin` – UTF-8 文字列。1 ~ 128 バイト長。Custom string pattern #31 に一致。

  リクエストの送信元。

レスポンス

- `Id` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  停止したセッションの ID を返します。

エラー

- `AccessDeniedException`
- `InternalServiceException`
- `OperationTimeoutException`
- `InvalidInputException`
- `IllegalSessionStateException`
- `ConcurrentModificationException`

DeleteSession アクション (Python: delete_session)

セッションを削除します。

リクエスト

- `Id` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  削除するセッションの ID。
- `RequestOrigin` – UTF-8 文字列。1 ~ 128 バイト長。Custom string pattern #31 に一致。

  セッション削除リクエスト送信元の名前。

レスポンス

- Id – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  削除したセッションの ID を返します。

エラー

- AccessDeniedException
- InternalServiceException
- OperationTimeoutException
- InvalidInputException
- IllegalSessionStateException
- ConcurrentModificationException

GetSession アクション (Python: get_session)

セッションを取得します。

リクエスト

- Id – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  セッションの ID
- RequestOrigin – UTF-8 文字列。1 ~ 128 バイト長。Custom string pattern #31 に一致。

  リクエストの送信元。

レスポンス

- Session – セッション オブジェクト。

  セッションオブジェクトはレスポンスで返されます。

エラー

- AccessDeniedException

- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException
- InvalidInputException

ListSessions アクション (Python: list_sessions)

セッションのリストを取得します。

リクエスト

- NextToken - UTF-8 文字列。400,000 バイト長以下。

  次の結果セットのトークン、または追加の結果がない場合は null。
- MaxResults – 1 ~ 1000 の数値 (整数)。

  結果の最大数。
- Tags – キーと値のペアのマップ配列。50 ペア以下。

  各キーは UTF-8 文字列で、1 ~ 128 バイト長です。

  各値は UTF-8 文字列で、256 バイト長以下です。

  セッションに属するタグ。
- RequestOrigin – UTF-8 文字列。1 ~ 128 バイト長。Custom string pattern #31 に一致。

  リクエストの送信元。

レスポンス

- Ids – UTF-8 文字列の配列。

  セッションの ID を返します。
- Sessions – セッション オブジェクトの配列。

  セッションオブジェクトを返します。
- NextToken - UTF-8 文字列。400,000 バイト長以下。

次の結果セットのトークン、または追加の結果がない場合は null。

エラー

- AccessDeniedException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException

RunStatement アクション (Python: run_statement)

ステートメントを実行します。

リクエスト

- SessionId – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  実行するステートメントのセッション ID。
- Code – 必須: UTF-8 文字列。68,000 バイト長未満。

  実行するステートメントのコード。
- RequestOrigin – UTF-8 文字列。1 ~ 128 バイト長。Custom string pattern #31 に一致。

  リクエストの送信元。

レスポンス

- Id – 数値 (整数)。

  実行されたステートメントの ID を返します。

エラー

- EntityNotFoundException
- AccessDeniedException
- InternalServiceException

- OperationTimeoutException
- InvalidInputException
- ValidationException
- ResourceNumberLimitExceededException
- IllegalSessionStateException

CancelStatement アクション (Python: cancel_statement)

ステートメントをキャンセルします。

リクエスト

- SessionId – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  キャンセルするステートメントのセッション ID。
- Id – 必須: 数値 (integer)。

  キャンセルするステートメントの ID。
- RequestOrigin – UTF-8 文字列。1 ~ 128 バイト長。Custom string pattern #31 に一致。

  ステートメントのキャンセルをリクエストした送信元。

レスポンス

- 応答パラメータはありません。

エラー

- AccessDeniedException
- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException
- InvalidInputException
- IllegalSessionStateException

GetStatement アクション (Python: get_statement)

ステートメントを取得します。

リクエスト

- `SessionId` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  ステートメントのセッション ID。
- `Id` – 必須: 数値 (integer)。

  ステートメントの ID。
- `RequestOrigin` – UTF-8 文字列。1 ~ 128 バイト長。Custom string pattern #31 に一致。

  リクエストの送信元。

レスポンス

- `Statement` – Statement オブジェクト。

  ステートメントを返します。

エラー

- `AccessDeniedException`
- `EntityNotFoundException`
- `InternalServiceException`
- `OperationTimeoutException`
- `InvalidInputException`
- `IllegalSessionStateException`

ListStatements アクション (Python: list_statements)

セッションのステートメントをリスト表示します。

リクエスト

- `SessionId` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

ステートメントのセッション ID。

- `RequestOrigin` – UTF-8 文字列。1 ~ 128 バイト長。Custom string pattern #31 に一致。

  ステートメントのリスト表示をリクエストした送信元。

- `NextToken` - UTF-8 文字列。400,000 バイト長以下。

  継続トークン (これが継続呼び出しの場合)。

応答

- `Statements` – Statement オブジェクトの配列。

  ステートメントのリストを返します。

- `NextToken` - UTF-8 文字列。400,000 バイト長以下。

  一部のステートメントがまだ返されていない場合は、継続トークン。

エラー

- `AccessDeniedException`
- `EntityNotFoundException`
- `InternalServiceException`
- `OperationTimeoutException`
- `InvalidInputException`
- `IllegalSessionStateException`

開発エンドポイント API

開発エンドポイント API では、カスタム DevEndpoint を使用したテストに関連した AWS Glue API について説明します。

データ型

- DevEndpoint 構造
- DevEndpointCustomLibraries 構造

DevEndpoint 構造

開発者が (ETL) スクリプトをリモートでデバッグ、抽出、変換、ロードする開発エンドポイント。

フィールド

- EndpointName – UTF-8 文字列。

  DevEndpoint の名前。
- RoleArn – UTF-8 文字列。AWS IAM ARN string pattern に一致。

  この DevEndpoint で使用される IAM ロールの Amazon リソースネーム (ARN)。
- SecurityGroupIds – UTF-8 文字列の配列。

  この DevEndpoint で使用されるセキュリティグループ識別子のリスト。
- SubnetId – UTF–8 文字列。

  この DevEndpoint のサブネット ID。
- YarnEndpointAddress – UTF–8 文字列。

  この DevEndpoint で使用される YARN エンドポイントアドレス。
- PrivateAddress – UTF–8 文字列。

  DevEndpoint が VPC 内に作成されている場合は、VPC 内の DevEndpoint にアクセスするためのプライベート IP アドレス。PrivateAddress フィールドは、VPC 内に DevEndpoint を作成する場合にのみ表示されます。
- ZeppelinRemoteSparkInterpreterPort – 数値 (整数)。

  リモート Apache Spark インタープリタの Apache Zeppelin ポート。
- PublicAddress – UTF–8 文字列。

  この DevEndpoint で使用されるパブリック IP アドレス。PublicAddress フィールドは、非 Virtual Private Cloud (VPC) の DevEndpoint を作成する場合にのみ存在します。
- Status – UTF–8 文字列。

  DevEndpoint の現在の状態。
- WorkerType – UTF-8 文字列 (有効な値: Standard="" | G.1X="" | G.2X="" | G.025X="" | G.4X="" | G.8X="" | Z.2X="")。

開発エンドポイントに割り当てられる事前定義されたワーカーのタイプ。使用できる値は、Standard、G.1X、または G.2X です。

- `Standard` ワーカータイプでは、各ワーカーは 4 vCPU、16 GB のメモリ、50 GB のディスク、ワーカーあたり 2 個のエグゼキュターを提供します。
- `G.1X` ワーカータイプでは、各ワーカーは 1 DPU (4 vCPU、16 GB のメモリ、64 GB のディスク) にマッピングされており、ワーカーごとに 1 個のエグゼキュターを提供します。メモリを大量に消費するジョブには、このワーカータイプをお勧めします。
- `G.2X` ワーカータイプでは、各ワーカーは 2 DPU (8 vCPU、32 GB のメモリ、128 GB のディスク) にマッピングされており、ワーカーごとに 1 個のエグゼキュターを提供します。メモリを大量に消費するジョブには、このワーカータイプをお勧めします。

既知の問題: `G.2X WorkerType` 設定で開発エンドポイントを作成すると、開発エンドポイントの Spark ドライバーは 4 vCPU、16 GB のメモリ、および 64 GB のディスクで実行されます。

- `GlueVersion` – UTF-8 文字列。1 ~ 255 バイト長。Custom string pattern #47 に一致。

  Glue バージョンは、AWS Glue がサポートする Apache Spark と Python のバージョンを決定します。Python バージョンは、開発エンドポイントで ETL スクリプトを実行するためにサポートされているバージョンを示します。

  利用可能な AWS Glue のバージョン、および対応する Spark および Python のバージョンの詳細については、デベロッパーガイドの「Glue version」を参照してください。

  Glue バージョンを指定せずに作成された開発エンドポイントは、デフォルトで Glue 0.9 に設定されます。

  `CreateDevEndpoint` または `UpdateDevEndpoint` API で `Arguments` パラメータを使用して、開発エンドポイントの Python サポートのバージョンを指定できます。引数を指定しない場合、バージョンはデフォルトで Python 2 になります。

- `NumberOfWorkers` – 数値 (整数)。

  開発エンドポイントに割り当てられた、 定義された `workerType` のワーカーの数。

  定義可能なワーカーの最大数は、299 (`G.1X`) または 149 (`G.2X`) です。

- `NumberOfNodes` – 数値 (整数)。

  この `DevEndpoint` に割り当てられた AWS Glue データ処理ユニット (DPU) の数。

- `AvailabilityZone` – UTF–8 文字列。

この DevEndpoint が配置されている AWS アベイラビリティーゾーン。

- VpcId – UTF–8 文字列。

  この DevEndpoint によって使用される Virtual Private Cloud (VPC) の ID。

- ExtraPythonLibsS3Path – UTF–8 文字列。

  DevEndpoint にロードする Amazon S3 バケットの 1 つ以上の Python ライブラリへのパス。複数の値はコンマで区切られた完全なパスでなければなりません。

  > **Note**
  >
  > DevEndpoint を持つ純粋な Python ライブラリのみを使用することができます。pandas Python データ解析ライブラリなど、C の拡張に依存するライブラリは現在サポートされていません。

- ExtraJarsS3Path – UTF–8 文字列。

  DevEndpoint にロードする S3 バケットの 1 つ以上の Java .jar へのパス。

  > **Note**
  >
  > DevEndpoint を持つ純粋な Java/Scala ライブラリのみを使用することができます。

- FailureReason – UTF–8 文字列。

  この DevEndpoint で現在障害が発生している原因。

- LastUpdateStatus – UTF–8 文字列。

  最終更新のステータス。

- CreatedTimestamp – タイムスタンプ。

  この DevEndpoint が作成された時点。

- LastModifiedTimestamp – タイムスタンプ。

  この DevEndpoint が最後に変更された時点。

- PublicKey – UTF–8 文字列。

この DevEndpoint が認証に使用するパブリックキー。この属性は、使用が推奨される属性がパブリックキーであるため、下位互換性のために提供されています。

- PublicKeys - UTF-8 文字列の配列。文字列 5 個以下。

  認証のために DevEndpoints により使用されるパブリックキーのリスト。パブリックキーを使用するとクライアントごとに異なるプライベートキーを設定できるため、この属性は単一のパブリックキーで使用することが推奨されます。

  > **Note**
  >
  > 以前にパブリックキーを使用してエンドポイントを作成した場合は、パブリックキーのリストを設定できるようにそのキーを削除する必要があります。パブリックキーのコンテンツを deletePublicKeys 属性で、新しいキーのリストを addPublicKeys 属性で UpdateDevEndpoint API オペレーションを呼び出します。

- SecurityConfiguration – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  この DevEndpoint で使用される SecurityConfiguration 構造の名前。

- Arguments – キーと値のペアのマップ配列。100 ペア以下。

  各キーは UTF-8 文字列。

  各値は UTF-8 文字列。

  DevEndpoint を設定するために使用される引数のマップ。

  有効な引数は次のとおりです。

  - "--enable-glue-datacatalog": ""

  CreateDevEndpoint または UpdateDevEndpoint API で Arguments パラメータを使用して、開発エンドポイントの Python サポートのバージョンを指定できます。引数を指定しない場合、バージョンはデフォルトで Python 2 になります。

DevEndpointCustomLibraries 構造

開発エンドポイントにロードされるカスタムライブラリ。

フィールド

- `ExtraPythonLibsS3Path` – UTF-8 文字列。

  `DevEndpoint` にロードする Amazon Simple Storage Service (Amazon S3) バケットの 1 つ以上の Python ライブラリへのパス。複数の値はコンマで区切られた完全なパスでなければなりません。

  > **Note**
  >
  > `DevEndpoint` を持つ純粋な Python ライブラリのみを使用することができます。pandas Python データ解析ライブラリなど、C の拡張に依存するライブラリは現在サポートされていません。

- `ExtraJarsS3Path` – UTF–8 文字列。

  `DevEndpoint` にロードする S3 バケットの 1 つ以上の Java `.jar` へのパス。

  > **Note**
  >
  > `DevEndpoint` を持つ純粋な Java/Scala ライブラリのみを使用することができます。

操作

- CreateDevEndpoint アクション (Python: create_dev_endpoint)
- UpdateDevEndpoint アクション (Python: update_dev_endpoint)
- DeleteDevEndpoint アクション (Python: delete_dev_endpoint)
- GetDevEndpoint アクション (Python: get_dev_endpoint)
- GetDevEndpoints アクション (Python: get_dev_endpoints)
- BatchGetDevEndpoints アクション (Python: batch_get_dev_endpoints)
- ListDevEndpoints アクション (Python: list_dev_endpoints)

CreateDevEndpoint アクション (Python: create_dev_endpoint)

新しい開発エンドポイントを作成します。

リクエスト

- EndpointName – 必須: UTF–8 文字列。

  新しい DevEndpoint に割り当てられる名前。

- RoleArn – 必須: UTF-8 文字列。AWS IAM ARN string pattern に一致。

  DevEndpoint の IAM ロール。

- SecurityGroupIds – UTF-8 文字列の配列。

  新しい DevEndpoint によって使用されるセキュリティグループのセキュリティグループ ID。

- SubnetId – UTF–8 文字列。

  使用する新しい DevEndpoint のサブネット ID。

- PublicKey – UTF–8 文字列。

  この DevEndpoint が認証に使用するパブリックキー。この属性は、使用が推奨される属性がパブリックキーであるため、下位互換性のために提供されています。

- PublicKeys - UTF-8 文字列の配列。文字列 5 個以下。

  認証のために開発エンドポイントで使用されるパブリックキーのリスト。パブリックキーを使用するとクライアントごとに異なるプライベートキーを設定できるため、この属性は単一のパブリックキーで使用することが推奨されます。

  > **Note**
  >
  > 以前にパブリックキーを使用してエンドポイントを作成した場合は、パブリックキーのリストを設定できるようにそのキーを削除する必要があります。パブリックキーのコンテンツを deletePublicKeys 属性で、新しいキーのリストを addPublicKeys 属性で指定して、UpdateDevEndpoint API オペレーションを呼び出します。

- NumberOfNodes – 数値 (整数)。

  この DevEndpoint に割り当てる AWS Glue データ処理ユニット (DPU) の数。

- WorkerType – UTF-8 文字列 (有効な値: Standard="" | G.1X="" | G.2X="" | G.025X="" | G.4X="" | G.8X="" | Z.2X="")。

  開発エンドポイントに割り当てられる事前定義されたワーカーのタイプ。使用できる値は、Standard、G.1X、または G.2X です。

  - `Standard` ワーカータイプでは、各ワーカーは 4 vCPU、16 GB のメモリ、50 GB のディスク、ワーカーあたり 2 個のエグゼキュターを提供します。
  - `G.1X` ワーカータイプでは、各ワーカーは 1 DPU (4 vCPU、16 GB のメモリ、64 GB のディスク) にマッピングされており、ワーカーごとに 1 個のエグゼキュターを提供します。メモリを大量に消費するジョブには、このワーカータイプをお勧めします。
  - `G.2X` ワーカータイプでは、各ワーカーは 2 DPU (8 vCPU、32 GB のメモリ、128 GB のディスク) にマッピングされており、ワーカーごとに 1 個のエグゼキュターを提供します。メモリを大量に消費するジョブには、このワーカータイプをお勧めします。

  既知の問題: `G.2X` `WorkerType` 設定で開発エンドポイントを作成すると、開発エンドポイントの Spark ドライバーは 4 vCPU、16 GB のメモリ、および 64 GB のディスクで実行されます。

- `GlueVersion` – UTF-8 文字列。1 ~ 255 バイト長。Custom string pattern #47 に一致。

  Glue バージョンは、AWS Glue がサポートする Apache Spark と Python のバージョンを決定します。Python バージョンは、開発エンドポイントで ETL スクリプトを実行するためにサポートされているバージョンを示します。

  利用可能な AWS Glue のバージョン、および対応する Spark および Python のバージョンの詳細については、デベロッパーガイドの「Glue version」を参照してください。

  Glue バージョンを指定せずに作成された開発エンドポイントは、デフォルトで Glue 0.9 に設定されます。

  `CreateDevEndpoint` または `UpdateDevEndpoint` API で `Arguments` パラメータを使用して、開発エンドポイントの Python サポートのバージョンを指定できます。引数を指定しない場合、バージョンはデフォルトで Python 2 になります。

- `NumberOfWorkers` – 数値 (整数)。

  開発エンドポイントに割り当てられた、 定義された `workerType` のワーカーの数。

  定義可能なワーカーの最大数は、299 (`G.1X`) または 149 (`G.2X`) です。

- `ExtraPythonLibsS3Path` – UTF–8 文字列。

  `DevEndpoint` にロードする Amazon S3 バケットの 1 つ以上の Python ライブラリへのパス。複数の値はコンマで区切られた完全なパスでなければなりません。

> **Note**
>
> DevEndpoint を持つ純粋な Python ライブラリのみを使用することができます。pandas Python データ解析ライブラリなど、C の拡張機能に依存するライブラリはまだサポートされていません。

- ExtraJarsS3Path – UTF–8 文字列。

  DevEndpoint にロードする S3 バケットの 1 つ以上の Java .jar へのパス。

- SecurityConfiguration – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  この DevEndpoint で使用される SecurityConfiguration 構造の名前。

- Tags – キーと値のペアのマップ配列。50 ペア以下。

  各キーは UTF-8 文字列で、1 ~ 128 バイト長です。

  各値は UTF-8 文字列で、256 バイト長以下です。

  この DevEndpoint で使用するタグ。DevEndpoint へのアクセスを制限するためにタグを使用することができます。AWS Glue のタグの詳細については、デベロッパーガイドの「AWS Tags in AWS Glue」を参照してください。

- Arguments – キーと値のペアのマップ配列。100 ペア以下。

  各キーは UTF-8 文字列。

  各値は UTF-8 文字列。

  DevEndpoint を設定するために使用される引数のマップ。

レスポンス

- EndpointName – UTF-8 文字列。

  新しい DevEndpoint に割り当てられた名前。

- Status – UTF–8 文字列。

  新しい DevEndpoint の現在の状態。

- SecurityGroupIds – UTF-8 文字列の配列。

  新しい DevEndpoint に割り当てられたセキュリティグループ。

- SubnetId – UTF–8 文字列。

  新しい DevEndpoint に割り当てられたサブネット ID。

- RoleArn – UTF-8 文字列。AWS IAM ARN string pattern に一致。

  新しい DevEndpoint に割り当てられたロールの Amazon リソースネーム (ARN)。

- YarnEndpointAddress – UTF–8 文字列。

  この DevEndpoint で使用される YARN エンドポイントのアドレス。

- ZeppelinRemoteSparkInterpreterPort – 数値 (整数)。

  リモート Apache Spark インタープリタの Apache Zeppelin ポート。

- NumberOfNodes – 数値 (整数)。

  この DevEndpoint に割り当てられた AWS Glue データ処理ユニット (DPU) の数。

- WorkerType – UTF-8 文字列 (有効な値: Standard="" | G.1X="" | G.2X="" | G.025X="" | G.4X="" | G.8X="" | Z.2X="")。

  開発エンドポイントに割り当てられる事前定義されたワーカーのタイプ。使用できる値は、Standard、G.1X、または G.2X です。

- GlueVersion – UTF-8 文字列。1 ~ 255 バイト長。Custom string pattern #47 に一致。

  Glue バージョンは、AWS Glue がサポートする Apache Spark と Python のバージョンを決定します。Python バージョンは、開発エンドポイントで ETL スクリプトを実行するためにサポートされているバージョンを示します。

  利用可能な AWS Glue のバージョン、および対応する Spark および Python のバージョンの詳細については、デベロッパーガイドの「Glue version」を参照してください。

- NumberOfWorkers – 数値 (整数)。

  開発エンドポイントに割り当てられた、 定義された workerType のワーカーの数。

- AvailabilityZone – UTF–8 文字列。

  この DevEndpoint が配置されている AWS アベイラビリティーゾーン。

- VpcId – UTF–8 文字列。

この DevEndpoint によって使用される Virtual Private Cloud (VPC) の ID。

- ExtraPythonLibsS3Path – UTF–8 文字列。

  DevEndpoint にロードする S3 バケットの 1 つ以上の Python ライブラリへのパス。

- ExtraJarsS3Path – UTF–8 文字列。

  DevEndpoint にロードする S3 バケットの 1 つ以上の Java .jar ファイルへのパス。

- FailureReason – UTF–8 文字列。

  この DevEndpoint で現在障害が発生している原因。

- SecurityConfiguration – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  この DevEndpoint で使用されている SecurityConfiguration 構造の名前。

- CreatedTimestamp – タイムスタンプ。

  この DevEndpoint が最後に変更された時点。

- Arguments – キーと値のペアのマップ配列。100 ペア以下。

  各キーは UTF-8 文字列。

  各値は UTF-8 文字列。

  この DevEndpoint を設定するために使用される引数のマップ。

  有効な引数は次のとおりです。

  - "--enable-glue-datacatalog": ""

  CreateDevEndpoint または UpdateDevEndpoint API で Arguments パラメータを使用して、開発エンドポイントの Python サポートのバージョンを指定できます。引数を指定しない場合、バージョンはデフォルトで Python 2 になります。

エラー

- AccessDeniedException
- AlreadyExistsException
- IdempotentParameterMismatchException

- InternalServiceException
- OperationTimeoutException
- InvalidInputException
- ValidationException
- ResourceNumberLimitExceededException

UpdateDevEndpoint アクション (Python: update_dev_endpoint)

指定された開発エンドポイントを更新します。

リクエスト

- EndpointName – 必須: UTF–8 文字列。

  更新される DevEndpoint の名前。
- PublicKey – UTF–8 文字列。

  使用する DevEndpoint のパブリックキー。
- AddPublicKeys - UTF-8 文字列の配列。文字列 5 個以下。

  使用する DevEndpoint のパブリックキーのリスト。
- DeletePublicKeys - UTF-8 文字列の配列。文字列 5 個以下。

  DevEndpoint から削除するパブリックキーのリスト。
- CustomLibraries – DevEndpointCustomLibraries オブジェクト。

  DevEndpoint でロードされるカスタム Python または Java ライブラリ。
- UpdateEtlLibraries – ブール。

  開発エンドポイントでロードされるカスタムライブラリのリストを更新する必要がある場合は True、それ以外の場合は False。
- DeleteArguments – UTF-8 文字列の配列。

  DevEndpoint の設定に使用される引数のマップから削除される引数キーのリスト。
- AddArguments – キーと値のペアのマップ配列。100 ペア以下。

  各キーは UTF-8 文字列。

各値は UTF-8 文字列。

DevEndpoint の設定に使用される引数のマップを追加する引数のマップ。

有効な引数は次のとおりです。

- "--enable-glue-datacatalog": ""

CreateDevEndpoint または UpdateDevEndpoint API で Arguments パラメータを使用して、開発エンドポイントの Python サポートのバージョンを指定できます。引数を指定しない場合、バージョンはデフォルトで Python 2 になります。

レスポンス

- 応答パラメータはありません。

エラー

- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException
- InvalidInputException
- ValidationException

DeleteDevEndpoint アクション (Python: delete_dev_endpoint)

指定された開発エンドポイントを削除します。

リクエスト

- EndpointName – 必須: UTF–8 文字列。

  DevEndpoint の名前。

レスポンス

- 応答パラメータはありません。

エラー

- `EntityNotFoundException`
- `InternalServiceException`
- `OperationTimeoutException`
- `InvalidInputException`

GetDevEndpoint アクション (Python: get_dev_endpoint)

指定した開発エンドポイントに関する情報を取得します。

> **Note**
>
> Virtual Private Cloud (VPC) で開発エンドポイントを作成する場合、AWS Glue はプライベート IP アドレスのみを返します。パブリック IP アドレスフィールドは入力されません。VPC 以外の開発エンドポイントを作成する場合、AWS Glue はパブリック IP アドレスのみを返します。

リクエスト

- `EndpointName` – 必須: UTF–8 文字列。

  情報を取得する `DevEndpoint` の名前。

レスポンス

- `DevEndpoint` – DevEndpoint オブジェクト。

  `DevEndpoint` の定義。

エラー

- `EntityNotFoundException`
- `InternalServiceException`
- `OperationTimeoutException`
- `InvalidInputException`

GetDevEndpoints アクション (Python: get_dev_endpoints)

この AWS アカウントのすべての開発エンドポイントを取得します。

> **Note**
>
> Virtual Private Cloud (VPC) で開発エンドポイントを作成する場合、AWS Glue はプライベート IP アドレスのみを返します。パブリック IP アドレスフィールドには入力されません。VPC 以外の開発エンドポイントを作成する場合、AWS Glue はパブリック IP アドレスのみを返します。

リクエスト

- `MaxResults` – 1 ~ 1000 の数値 (整数)。

  返される情報の最大サイズ。
- `NextToken` – UTF-8 文字列。

  継続トークン (これが継続呼び出しの場合)。

応答

- `DevEndpoints` – DevEndpoint オブジェクトの配列。

  `DevEndpoint` の定義のリスト。
- `NextToken` – UTF–8 文字列。

  一部の `DevEndpoint` 定義がまだ返されていない場合は、継続トークン。

エラー

- `EntityNotFoundException`
- `InternalServiceException`
- `OperationTimeoutException`
- `InvalidInputException`

BatchGetDevEndpoints アクション (Python: batch_get_dev_endpoints)

指定された開発エンドポイントのリストのリソースメタデータのリストを返します。ListDevEndpoints オペレーションを呼び出した後で、このオペレーションを呼び出すことで、アクセス許可が付与されているデータにアクセスできます。このオペレーションは、タグを使用するアクセス許可条件を含め、すべての IAM のアクセス許可をサポートします。

リクエスト

- customerAccountId – UTF-8 文字列。

  AWS アカウント ID。
- DevEndpointNames – 必須: UTF-8 文字列の配列。1 ~ 25 個の文字列。

  DevEndpoint 名のリスト。これは ListDevEndpoint オペレーションから返される名前です。

応答

- DevEndpoints – DevEndpoint オブジェクトの配列。

  DevEndpoint の定義のリスト。
- DevEndpointsNotFound – UTF-8 文字列の配列。1 ~ 25 個の文字列。

  DevEndpoints のリストが見つかりません。

エラー

- AccessDeniedException
- InternalServiceException
- OperationTimeoutException
- InvalidInputException

ListDevEndpoints アクション (Python: list_dev_endpoints)

この AWS アカウントのすべての DevEndpoint リソースまたは指定されたタグを持つリソースの名前を取得します。このオペレーションにより、アカウントで利用可能なリソースとその名前を確認できます。

このオペレーションはオプションの Tags フィールドを受け取ります。このフィールドを応答のフィルターとして使用すると、タグ付きリソースをグループとして取得できます。タグフィルタリングの使用を選択した場合は、タグが付いたリソースのみが取得されます。

リクエスト

- NextToken – UTF-8 文字列。

  継続トークン (これが継続リクエストの場合)。

- MaxResults – 1 ~ 1000 の数値 (整数)。

  返されるリストの最大サイズ。

- Tags – キーと値のペアのマップ配列。50 ペア以下。

  各キーは UTF-8 文字列で、1 ~ 128 バイト長です。

  各値は UTF-8 文字列で、256 バイト長以下です。

  これらのタグ付きリソースのみを返すように指定します。

レスポンス

- DevEndpointNames – UTF-8 文字列の配列。

  アカウント内のすべての DevEndpoint の名前、または指定されたタグを持つ DevEndpoint の名前。

- NextToken – UTF–8 文字列。

  継続トークン (戻されたリストに最後に使用可能なメトリクスが含まれていない場合)。

エラー

- InvalidInputException
- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException

スキーマレジストリ

スキーマレジストリ API では、AWS Glue でのスキーマの操作に関連するデータ型と API について説明します。

データ型

- RegistryId 構造
- RegistryListItem 構造
- MetadataInfo 構造
- OtherMetadataValueListItem 構造
- SchemaListItem 構造
- SchemaVersionListItem 構造
- MetadataKeyValuePair 構造
- SchemaVersionErrorItem 構造
- ErrorDetails 構造
- SchemaVersionNumber 構造
- SchemaId 構造

RegistryId 構造

レジストリ名と Amazon リソースネーム (ARN) を含むことができるラッパー構造体。

フィールド

- `RegistryName` – UTF-8 文字列。1 ~ 255 バイト長。Custom string pattern #45 に一致。

  レジストリの名前。ルックアップにのみ使用されます。`RegistryArn` または `RegistryName` のいずれかを指定する必要があります。
- `RegistryArn` – UTF-8 文字列。1 ~ 10240 バイト長。Custom string pattern #49 に一致。

  更新するレジストリの ARN。`RegistryArn` または `RegistryName` のいずれかを指定する必要があります。

RegistryListItem 構造

レジストリの詳細を含む構造。

フィールド

- `RegistryName` – UTF-8 文字列。1 ~ 255 バイト長。Custom string pattern #45 に一致。

  レジストリの名前。
- `RegistryArn` – UTF-8 文字列。1 ~ 10240 バイト長。Custom string pattern #49 に一致。

  レジストリの Amazon リソースネーム (ARN)
- `Description` – 説明文字列。2,048 バイト長以下。URI address multi-line string pattern に一致。

  レジストリの説明。
- `Status` – UTF-8 文字列 (有効な値: `AVAILABLE` | `DELETING`)。

  レジストリのステータス。
- `CreatedTime` – UTF–8 文字列。

  レジストリが作成されたデータ。
- `UpdatedTime` – UTF–8 文字列。

  レジストリが更新された日付。

MetadataInfo 構造

スキーマバージョンのメタデータ情報を含む構造。

フィールド

- `MetadataValue` – UTF-8 文字列。1 ~ 256 バイト長。Custom string pattern #14 に一致。

  メタデータキーの対応する値。
- `CreatedTime` – UTF–8 文字列。

  エントリの作成時刻。
- `OtherMetadataValueList` – OtherMetadataValueListItem オブジェクトの配列。

  同じメタデータキーに属する他のメタデータ。

OtherMetadataValueListItem 構造

同じメタデータキーに属するスキーマバージョンの他のメタデータが含まれる構造。

フィールド

- `MetadataValue` – UTF-8 文字列。1 ~ 256 バイト長。Custom string pattern #14 に一致。

  同じメタデータキーに属する他のメタデータ用の、それらに対応したメタデータキーの値。
- `CreatedTime` – UTF–8 文字列。

  エントリの作成時刻。

SchemaListItem 構造

スキーマに対する最小限の詳細が含まれるオブジェクト。

フィールド

- `RegistryName` – UTF-8 文字列。1 ~ 255 バイト長。Custom string pattern #45 に一致。

  スキーマが存在するレジストリの名前。
- `SchemaName` – UTF-8 文字列。1 ~ 255 バイト長。Custom string pattern #45 に一致。

  スキーマの名前。
- `SchemaArn` – UTF-8 文字列。1 ~ 10240 バイト長。Custom string pattern #49 に一致。

  スキーマの Amazon リソースネーム (ARN)。
- `Description` – 説明文字列。2,048 バイト長以下。URI address multi-line string pattern に一致。

  スキーマの説明。
- `SchemaStatus` – UTF-8 文字列 (有効な値: `AVAILABLE` | `PENDING` | `DELETING`)。

  スキーマのステータス。
- `CreatedTime` – UTF–8 文字列。

  スキーマが作成された日時。
- `UpdatedTime` – UTF–8 文字列。

  スキーマが更新された日時。

SchemaVersionListItem 構造

スキーマのバージョンに関する詳細を含むオブジェクト。

フィールド

- SchemaArn – UTF-8 文字列。1 ~ 10240 バイト長。Custom string pattern #49 に一致。

  スキーマの Amazon リソースネーム (ARN)。
- SchemaVersionId – UTF-8 文字列。36 バイト長ちょうど。Custom string pattern #44 に一致。

  スキーマバージョンの一意の識別子。
- VersionNumber – 数値 (long)。1 ~ 100000。

  スキーマのバージョン番号。
- Status – UTF-8 文字列 (有効な値: AVAILABLE | PENDING | FAILURE | DELETING)。

  スキーマバージョンのステータス。
- CreatedTime – UTF–8 文字列。

  スキーマバージョンが作成された日時。

MetadataKeyValuePair 構造

メタデータのキーバリューペアが含まれる構造。

フィールド

- MetadataKey – UTF-8 文字列。1 ~ 128 バイト長。Custom string pattern #14 に一致。

  メタデータキー。
- MetadataValue – UTF-8 文字列。1 ~ 256 バイト長。Custom string pattern #14 に一致。

  メタデータキーの対応する値。

SchemaVersionErrorItem 構造

スキーマバージョンに対するオペレーションに対するエラーの詳細が含まれるオブジェクト。

フィールド

- `VersionNumber` – 数値 (long)。1 ~ 100000。

  スキーマのバージョン番号。
- `ErrorDetails` – ErrorDetails オブジェクト。

  スキーマバージョンのエラーの詳細。

ErrorDetails 構造

エラーの詳細を含むオブジェクト。

フィールド

- `ErrorCode` – UTF-8 文字列。

  エラーのエラーコード。
- `ErrorMessage` – UTF–8 文字列。

  エラーのエラーメッセージ。

SchemaVersionNumber 構造

スキーマのバージョン情報を含む構造。

フィールド

- `LatestVersion` – ブール。

  スキーマで利用可能な最新バージョン。
- `VersionNumber` – 数値 (long)。1 ~ 100000。

  スキーマのバージョン番号。

SchemaId 構造

AWS Glue スキーマレジストリ内のスキーマの一意の ID。

フィールド

- SchemaArn – UTF-8 文字列。1 ~ 10240 バイト長。Custom string pattern #49 に一致。

  スキーマの Amazon リソースネーム (ARN)。SchemaArn または SchemaName のいずれかを指定する必要があります。

- SchemaName – UTF-8 文字列。1 ~ 255 バイト長。Custom string pattern #45 に一致。

  スキーマの名前。SchemaArn または SchemaName のいずれかを指定する必要があります。

- RegistryName – UTF-8 文字列。1 ~ 255 バイト長。Custom string pattern #45 に一致。

  スキーマを含むスキーマレジストリの名前。

操作

- CreateRegistry アクション (Python: create_registry)
- CreateSchema アクション (Python: create_schema)
- GetSchema アクション (Python: get_schema)
- ListSchemaVersions アクション (Python: list_schema_versions)
- GetSchemaVersion アクション (Python: get_schema_version)
- GetSchemaVersionsDiff アクション (Python: get_schema_versions_diff)
- ListRegistries アクション (Python: list_registries)
- ListSchemas アクション (Python: list_schemas)
- RegisterSchemaVersion アクション (Python: register_schema_version)
- UpdateSchema アクション (Python: update_schema)
- CheckSchemaVersionValidity アクション (Python: check_schema_version_validity)
- UpdateRegistry アクション (Python: update_registry)
- GetSchemaByDefinition アクション (Python: get_schema_by_definition)
- GetRegistry アクション (Python: get_registry)
- PutSchemaVersionMetadata アクション (Python: put_schema_version_metadata)
- QuerySchemaVersionMetadata アクション (Python: query_schema_version_metadata)
- RemoveSchemaVersionMetadata アクション (Python: remove_schema_version_metadata)

- DeleteRegistry アクション (Python: delete_registry)
- DeleteSchema アクション (Python: delete_schema)
- DeleteSchemaVersions アクション (Python: delete_schema_versions)

CreateRegistry アクション (Python: create_registry)

スキーマのコレクションを保持するために使用できる新しいレジストリを作成します。

リクエスト

- `RegistryName` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Custom string pattern #45 に一致。

  作成するスキーマの名前の最大長は 255 で、英字、数字、ハイフン、アンダースコア、ドル記号、ハッシュ記号のみを使用できます。空白はありません。
- `Description` – 説明文字列。2,048 バイト長以下。URI address multi-line string pattern に一致。

  レジストリの説明。説明がない場合、これに対するデフォルト値はありません。
- `Tags` – キーと値のペアのマップ配列。50 ペア以下。

  各キーは UTF-8 文字列で、1 ~ 128 バイト長です。

  各値は UTF-8 文字列で、256 バイト長以下です。

  キーと値のペアを含み、コンソール、コマンドライン、または API で検索できる AWS タグ。

レスポンス

- `RegistryArn` – UTF-8 文字列。1 ~ 10240 バイト長。Custom string pattern #49 に一致。

  新たに作成されたレジストリの Amazon リソースネーム (ARN)。
- `RegistryName` – UTF-8 文字列。1 ~ 255 バイト長。Custom string pattern #45 に一致。

  レジストリの名前。
- `Description` – 説明文字列。2,048 バイト長以下。URI address multi-line string pattern に一致。

  レジストリの説明。
- `Tags` – キーと値のペアのマップ配列。50 ペア以下。

  各キーは UTF-8 文字列で、1 ~ 128 バイト長です。

各値は UTF-8 文字列で、256 バイト長以下です。

レジストリのタグ。

エラー

- `InvalidInputException`
- `AccessDeniedException`
- `AlreadyExistsException`
- `ResourceNumberLimitExceededException`
- `ConcurrentModificationException`
- `InternalServiceException`

CreateSchema アクション (Python: create_schema)

新しいスキーマセットを作成し、スキーマ定義を登録します。スキーマセットがすでに存在する場合は、実際にバージョンを登録せずにエラーを返します。

スキーマセットが作成されると、バージョンチェックポイントが最初のバージョンに設定されます。互換モードが「DISABLED」であると、最初のスキーマバージョンの後に追加のスキーマバージョンが追加されないように制限されます。他のすべての互換モードでは、`RegisterSchemaVersion` API が使用された場合、2 番目以降のバージョンでのみ互換性設定の検証が適用されます。

この API が `RegistryId` なしで呼び出されたとき、レジストリデータベーステーブルに「default-registry」のエントリがまだ存在しなければ、エントリが作成されます。

リクエスト

- `RegistryId` – RegistryId オブジェクト。

  これは、レジストリ ID フィールドを含むラッパー図形です。これを指定しない場合、デフォルトのレジストリが使用されます。同じものの ARN 形式は次のようになります: `arn:aws:glue:us-east-2:<customer id>:registry/default-registry:random-5-letter-id`。
- `SchemaName` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Custom string pattern #45 に一致。

  作成するスキーマの名前の最大長は 255 で、文字、数字、ハイフン、アンダースコア、ドル記号、またはハッシュ記号のみを使用できます。空白はありません。

- DataFormat – 必須: UTF-8 文字列 (有効な値: AVRO | JSON | PROTOBUF)。

  スキーマ定義のデータ形式。現在サポートされている形式は、AVRO、JSON、および PROTOBUF です。

- Compatibility – UTF-8 文字列 (有効な値: NONE | DISABLED | BACKWARD | BACKWARD_ALL | FORWARD | FORWARD_ALL | FULL | FULL_ALL)。

  スキーマの互換モード。指定できる値は以下のとおりです。

  - NONE: 互換モードは適用されません。この選択は、開発シナリオで使用するか、スキーマに適用する互換モードがわからない場合に使用できます。新しいバージョンが追加されたとき、互換性チェックを受けずに受け入れられます。
  - DISABLED: 互換性についてこの選択をすると、特定のスキーマのバージョン管理が不要になります。この選択を使用して、スキーマの今後のバージョン管理を不要にできます。
  - BACKWARD: データ受信者が現在のスキーマバージョンと直前のスキーマバージョンの両方を読み取ることができるため、この互換性の選択をお勧めします。つまり、例えば、以前のバージョンを使用して読み取れなくならないように、新しいスキーマバージョンでデータフィールドを削除したり、これらのフィールドのタイプを変更したりできなくなります。
  - BACKWARD_ALL: この互換性の選択により、データ受信者は、現在および以前のすべてのスキーマバージョンの両方を読み取ることができます。この選択は、フィールドを削除したり、オプションのフィールドを追加したり、以前のすべてのスキーマバージョンとの互換性を確認する必要がある場合に使用できます。
  - FORWARD: この互換性の選択により、データ受信者は、現在のスキーマバージョンとすぐ次のスキーマバージョンの両方を読み取ることができますが、それより後のバージョンは必ずしも読み取れません。この選択は、フィールドを追加したり、オプションフィールドを削除したりする必要があり、最後のスキーマバージョンとの互換性だけをチェックする必要がある場合に使用できます。
  - FORWARD_ALL: この互換性の選択により、データ受信者は、新しく登録されるすべてのスキーマのプロデューサーによって書き込まれたデータを読み取ることができます。この選択は、フィールドを追加したり、オプションフィールドを削除したり、以前のすべてのスキーマバージョンとの互換性を確認したりする必要がある場合に使用できます。
  - FULL: この互換性の選択により、データ受信者は、スキーマの直前または直後のバージョンを使用してプロデューサーによって書き込まれたデータを読み取ることができますが、それより前または後のバージョンは必ずしも読み取れません。この選択は、オプションフィールドを追加または削除する必要があり、最後のスキーマバージョンとの互換性だけをチェックする必要がある場合に使用できます。

  - FULL_ALL: この互換性の選択により、データ受信者は、すべての以前のスキーマバージョンを使用してプロデューサーによって書き込まれたデータを読み取ることができます。この選択は、オプションのフィールドを追加または削除したり、以前のすべてのスキーマバージョンとの互換性を確認したりする必要がある場合に使用できます。
- `Description` – 説明文字列。2,048 バイト長以下。URI address multi-line string pattern に一致。

  スキーマの説明 (省略可能)。説明がない場合、これに自動的に適用されるデフォルト値はありません。
- `Tags` – キーと値のペアのマップ配列。50 ペア以下。

  各キーは UTF-8 文字列で、1 ~ 128 バイト長です。

  各値は UTF-8 文字列で、256 バイト長以下です。

  キーと値のペアを含み、コンソール、コマンドライン、または API で検索できる AWS タグ。指定した場合、AWS tags-on-create パターンに従います。
- `SchemaDefinition` – UTF-8 文字列。1 ~ 170000 バイト長。Custom string pattern #13 に一致。

  `SchemaName` の `DataFormat` 設定を使用したスキーマ定義。

レスポンス

- `RegistryName` – UTF-8 文字列。1 ~ 255 バイト長。Custom string pattern #45 に一致。

  レジストリの名前。
- `RegistryArn` – UTF-8 文字列。1 ~ 10240 バイト長。Custom string pattern #49 に一致。

  レジストリの Amazon リソースネーム (ARN)
- `SchemaName` – UTF-8 文字列。1 ~ 255 バイト長。Custom string pattern #45 に一致。

  スキーマの名前。
- `SchemaArn` – UTF-8 文字列。1 ~ 10240 バイト長。Custom string pattern #49 に一致。

  スキーマの Amazon リソースネーム (ARN)。
- `Description` – 説明文字列。2,048 バイト長以下。URI address multi-line string pattern に一致。

  スキーマの説明 (作成時に指定されている場合)。
- `DataFormat` – UTF-8 文字列 (有効な値: `AVRO` | `JSON` | `PROTOBUF`)。

スキーマ定義のデータ形式。現在サポートされている形式は、AVRO、JSON、および PROTOBUF です。

- Compatibility – UTF-8 文字列 (有効な値: NONE | DISABLED | BACKWARD | BACKWARD_ALL | FORWARD | FORWARD_ALL | FULL | FULL_ALL)。

  スキーマの互換モード。

- SchemaCheckpoint – 数値 (long)。1 ~ 100000。

  チェックポイントのバージョン番号 (互換モードが最後に変更されたときの変更回数)。

- LatestSchemaVersion – 数値 (long)。1 ~ 100000。

  返されるスキーマ定義に関連付けられたスキーマの最新バージョン。

- NextSchemaVersion – 数値 (long)。1 ~ 100000。

  返されたスキーマ定義に関連付けられたスキーマの次のバージョン。

- SchemaStatus – UTF-8 文字列 (有効な値: AVAILABLE | PENDING | DELETING)。

  スキーマのステータス。

- Tags – キーと値のペアのマップ配列。50 ペア以下。

  各キーは UTF-8 文字列で、1 ~ 128 バイト長です。

  各値は UTF-8 文字列で、256 バイト長以下です。

  スキーマのタグ。

- SchemaVersionId – UTF-8 文字列。36 バイト長ちょうど。Custom string pattern #44 に一致。

  最初のスキーマバージョンの一意の識別子。

- SchemaVersionStatus – UTF-8 文字列 (有効な値: AVAILABLE | PENDING | FAILURE | DELETING)。

  最初に作成されたスキーマバージョンのステータス。

エラー

- InvalidInputException
- AccessDeniedException

- EntityNotFoundException
- AlreadyExistsException
- ResourceNumberLimitExceededException
- ConcurrentModificationException
- InternalServiceException

GetSchema アクション (Python: get_schema)

指定されたスキーマについて詳細に説明します。

リクエスト

- SchemaId – 必須: SchemaId オブジェクト。

  スキーマ ID フィールドを含むラッパー構造体です。構造体には以下が含まれています。

  - SchemaId$SchemaArn: スキーマの Amazon リソースネーム (ARN)。SchemaArn または SchemaName、および RegistryName を指定する必要があります。
  - SchemaId$SchemaName: スキーマの名前。SchemaArn または SchemaName、および RegistryName を指定する必要があります。

レスポンス

- RegistryName – UTF-8 文字列。1 ~ 255 バイト長。Custom string pattern #45 に一致。

  レジストリの名前。
- RegistryArn – UTF-8 文字列。1 ~ 10240 バイト長。Custom string pattern #49 に一致。

  レジストリの Amazon リソースネーム (ARN)
- SchemaName – UTF-8 文字列。1 ~ 255 バイト長。Custom string pattern #45 に一致。

  スキーマの名前。
- SchemaArn – UTF-8 文字列。1 ~ 10240 バイト長。Custom string pattern #49 に一致。

  スキーマの Amazon リソースネーム (ARN)。
- Description – 説明文字列。2,048 バイト長以下。URI address multi-line string pattern に一致。

  スキーマの説明 (作成時に指定されている場合)。

- DataFormat – UTF-8 文字列 (有効な値: AVRO | JSON | PROTOBUF)。

  スキーマ定義のデータ形式。現在サポートされている形式は、AVRO、JSON、および PROTOBUF です。

- Compatibility – UTF-8 文字列 (有効な値: NONE | DISABLED | BACKWARD | BACKWARD_ALL | FORWARD | FORWARD_ALL | FULL | FULL_ALL)。

  スキーマの互換モード。

- SchemaCheckpoint – 数値 (long)。1 ~ 100000。

  チェックポイントのバージョン番号 (互換モードが最後に変更されたときの変更回数)。

- LatestSchemaVersion – 数値 (long)。1 ~ 100000。

  返されるスキーマ定義に関連付けられたスキーマの最新バージョン。

- NextSchemaVersion – 数値 (long)。1 ~ 100000。

  返されたスキーマ定義に関連付けられたスキーマの次のバージョン。

- SchemaStatus – UTF-8 文字列 (有効な値: AVAILABLE | PENDING | DELETING)。

  スキーマのステータス。

- CreatedTime – UTF–8 文字列。

  スキーマが作成された日時。

- UpdatedTime – UTF–8 文字列。

  スキーマが更新された日時。

エラー

- InvalidInputException
- AccessDeniedException
- EntityNotFoundException
- InternalServiceException

ListSchemaVersions アクション (Python: list_schema_versions)

作成したスキーマバージョンのリストを、最小限の情報とともに返します。Deleted ステータスのスキーマのバージョンは、結果に含まれません。使用可能なスキーマバージョンがない場合、空の結果が返されます。

リクエスト

- SchemaId – 必須: SchemaId オブジェクト。

  スキーマ ID フィールドを含むラッパー構造体です。構造体には以下が含まれています。

  - SchemaId$SchemaArn: スキーマの Amazon リソースネーム (ARN)。SchemaArn または SchemaName、および RegistryName を指定する必要があります。
  - SchemaId$SchemaName: スキーマの名前。SchemaArn または SchemaName、および RegistryName を指定する必要があります。
- MaxResults – 数値 (integer)。1 ~ 100。

  1 ページあたりに必要な結果の最大数。値が指定されていない場合、これはデフォルトで 1 ページあたり 25 に設定されます。
- NextToken – UTF-8 文字列。

  継続トークン (これが継続呼び出しの場合)。

応答

- Schemas – SchemaVersionListItem オブジェクトの配列。

  各スキーマのバージョンの詳細を含む SchemaVersionList オブジェクトの配列。
- NextToken – UTF-8 文字列。

  返されたトークンのリストをページ分割するための継続トークン。リストの現在のセグメントが最後のセグメントではない場合に返されます。

エラー

- InvalidInputException
- AccessDeniedException
- EntityNotFoundException

- InternalServiceException

GetSchemaVersion アクション (Python: get_schema_version)

スキーマのバージョンの作成または登録時に割り当てられた一意の ID によって指定されたスキーマを取得します。Deleted ステータスのスキーマのバージョンは、結果に含まれません。

リクエスト

- SchemaId – SchemaId オブジェクト。

  スキーマ ID フィールドを含むラッパー構造体です。構造体には以下が含まれています。

  - SchemaId$SchemaArn: スキーマの Amazon リソースネーム (ARN)。SchemaArn または SchemaName、および RegistryName を指定する必要があります。
  - SchemaId$SchemaName: スキーマの名前。SchemaArn または SchemaName、および RegistryName を指定する必要があります。
- SchemaVersionId – UTF-8 文字列。36 バイト長ちょうど。Custom string pattern #44 に一致。

  スキーマバージョンの SchemaVersionId。このフィールドは、スキーマ ID による取得に必須です。このスキーマ ID か SchemaId ラッパーのいずれかを指定する必要があります。
- SchemaVersionNumber – SchemaVersionNumber オブジェクト。

  スキーマのバージョン番号。

レスポンス

- SchemaVersionId – UTF-8 文字列。36 バイト長ちょうど。Custom string pattern #44 に一致。

  スキーマバージョンの SchemaVersionId。
- SchemaDefinition – UTF-8 文字列。1 ~ 170000 バイト長。Custom string pattern #13 に一致。

  スキーマ ID のスキーマ定義。
- DataFormat – UTF-8 文字列 (有効な値: AVRO | JSON | PROTOBUF)。

  スキーマ定義のデータ形式。現在サポートされている形式は、AVRO、JSON、および PROTOBUF です。
- SchemaArn – UTF-8 文字列。1 ~ 10240 バイト長。Custom string pattern #49 に一致。

スキーマの Amazon リソースネーム (ARN)。

- `VersionNumber` – 数値 (long)。1 ~ 100000。

  スキーマのバージョン番号。

- `Status` – UTF-8 文字列 (有効な値: `AVAILABLE` | `PENDING` | `FAILURE` | `DELETING`)。

  スキーマバージョンのステータス。

- `CreatedTime` – UTF–8 文字列。

  スキーマバージョンが作成された日時。

エラー

- `InvalidInputException`
- `AccessDeniedException`
- `EntityNotFoundException`
- `InternalServiceException`

GetSchemaVersionsDiff アクション (Python: get_schema_versions_diff)

スキーマレジストリに格納された 2 つのスキーマバージョン間で、指定された差分タイプによるスキーマバージョンの差分を取得します。

この API を使用すると、同じスキーマの下にある 2 つのスキーマ定義間で 2 つのスキーマバージョンを比較できます。

リクエスト

- `SchemaId` – 必須: SchemaId オブジェクト。

  スキーマ ID フィールドを含むラッパー構造体です。構造体には以下が含まれています。

  - SchemaId$SchemaArn: スキーマの Amazon リソースネーム (ARN)。`SchemaArn` または `SchemaName` のいずれかを指定する必要があります。
  - SchemaId$SchemaName: スキーマの名前。`SchemaArn` または `SchemaName` のいずれかを指定する必要があります。

- `FirstSchemaVersionNumber` – 必須: SchemaVersionNumber オブジェクト。

比較する 2 つのスキーマバージョンのうち、1 番目のスキーマバージョンです。

- SecondSchemaVersionNumber – 必須: SchemaVersionNumber オブジェクト。

  比較する 2 つのスキーマバージョンのうち、2 番目のスキーマバージョンです。

- SchemaDiffType – 必須: UTF-8 文字列 (有効な値: SYNTAX_DIFF)。

  現在サポートされている差分タイプである SYNTAX_DIFF が参照されます。

レスポンス

- Diff – UTF-8 文字列。1 ~ 340000 バイト長。Custom string pattern #13 に一致。

  JsonPatch 形式の文字列としてのスキーマの違い。

エラー

- InvalidInputException
- EntityNotFoundException
- AccessDeniedException
- InternalServiceException

ListRegistries アクション (Python: list_registries)

作成したレジストリの一覧を、最小限のレジストリ情報とともに返します。Deleting ステータスのレジストリは結果に含まれません。利用可能なレジストリがない場合、空の結果が返されます。

リクエスト

- MaxResults – 数値 (integer)。1 ~ 100。

  1 ページあたりに必要な結果の最大数。値が指定されていない場合、これはデフォルトで 1 ページあたり 25 に設定されます。

- NextToken – UTF-8 文字列。

  継続トークン (これが継続呼び出しの場合)。

応答

- `Registries` – RegistryListItem オブジェクトの配列。

  各レジストリの最小の詳細を含む `RegistryDetailedListItem` オブジェクトの配列。

- `NextToken` – UTF-8 文字列。

  返されたトークンのリストをページ分割するための継続トークン。リストの現在のセグメントが最後のセグメントではない場合に返されます。

エラー

- `InvalidInputException`
- `AccessDeniedException`
- `InternalServiceException`

ListSchemas アクション (Python: list_schemas)

スキーマのリストを最小限の詳細とともに返します。Deleting ステータスのスキーマは結果に含まれません。使用可能なスキーマがない場合、空の結果が返されます。

`RegistryId` が指定されていない場合、レジストリ間のすべてのスキーマは API レスポンスの一部になります。

リクエスト

- `RegistryId` – RegistryId オブジェクト。

  レジストリ名と Amazon リソースネーム (ARN) を含むことができるラッパー構造体。

- `MaxResults` – 数値 (integer)。1 ~ 100。

  1 ページあたりに必要な結果の最大数。値が指定されていない場合、これはデフォルトで 1 ページあたり 25 に設定されます。

- `NextToken` – UTF-8 文字列。

  継続トークン (これが継続呼び出しの場合)。

応答

- Schemas – SchemaListItem オブジェクトの配列。

  各スキーマの詳細を含む SchemaListItem オブジェクトの配列。

- NextToken – UTF-8 文字列。

  返されたトークンのリストをページ分割するための継続トークン。リストの現在のセグメントが最後のセグメントではない場合に返されます。

エラー

- InvalidInputException
- AccessDeniedException
- EntityNotFoundException
- InternalServiceException

RegisterSchemaVersion アクション (Python: register_schema_version)

既存のスキーマに新しいバージョンを追加します。スキーマの新しいバージョンがスキーマセットの互換性要件を満たしていない場合、エラーを返します。この API では、スキーマセットがまだスキーマレジストリに存在しない場合、新しいスキーマセットは作成されず、404 エラーが返されます。

これがスキーマレジストリに登録される最初のスキーマ定義である場合、この API はスキーマバージョンを保存し、すぐに戻ります。それ以外の場合、この呼び出しは、互換モードのために他のオペレーションよりも長時間実行される可能性があります。SchemaVersionId を使用して GetSchemaVersion API を呼び出して、互換モードをチェックできます。

同じスキーマ定義がすでにバージョンとして Schema Registry に格納されている場合は、既存のスキーマのスキーマ ID が呼び出し元に返されます。

リクエスト

- SchemaId – 必須: SchemaId オブジェクト。

  スキーマ ID フィールドを含むラッパー構造体です。構造体には以下が含まれています。

  - SchemaId$SchemaArn: スキーマの Amazon リソースネーム (ARN)。SchemaArn または SchemaName、および RegistryName を指定する必要があります。
  - SchemaId$SchemaName: スキーマの名前。SchemaArn または SchemaName、および RegistryName を指定する必要があります。
- SchemaDefinition – 必須: UTF-8 文字列。1 ~ 170000 バイト長。Custom string pattern #13 に一致。

  SchemaName の DataFormat 設定を使用したスキーマ定義。

レスポンス

- SchemaVersionId – UTF-8 文字列。36 バイト長ちょうど。Custom string pattern #44 に一致。

  このスキーマのバージョンを表す一意の ID。
- VersionNumber – 数値 (long)。1 ~ 100000。

  このスキーマのバージョン (これが最初のバージョンの場合、同期フローのみ)。
- Status – UTF-8 文字列 (有効な値: AVAILABLE | PENDING | FAILURE | DELETING)。

  スキーマバージョンのステータス。

エラー

- InvalidInputException
- AccessDeniedException
- EntityNotFoundException
- ResourceNumberLimitExceededException
- ConcurrentModificationException
- InternalServiceException

UpdateSchema アクション (Python: update_schema)

スキーマセットの説明、互換性設定、またはバージョンチェックポイントを更新します。

互換性設定を更新する場合、その呼び出しでは、新しい互換性設定を持つスキーマバージョンのセット全体の互換性は検証されません。Compatibility の値が指定されている場合、VersionNumber

(チェックポイント) も指定する必要があります。APIは、整合性のためにチェックポイントのバージョン番号を検証します。

VersionNumber (チェックポイント) の値が指定されている場合、Compatibility はオプションであり、スキーマのチェックポイントを設定/リセットするために使用できます。

この更新は、スキーマが AVAILABLE 状態である場合にのみ発生します。

リクエスト

- SchemaId – 必須: SchemaId オブジェクト。

  スキーマ ID フィールドを含むラッパー構造体です。構造体には以下が含まれています。

  - SchemaId$SchemaArn: スキーマの Amazon リソースネーム (ARN)。SchemaArn または SchemaName のいずれかを指定する必要があります。
  - SchemaId$SchemaName: スキーマの名前。SchemaArn または SchemaName のいずれかを指定する必要があります。
- SchemaVersionNumber – SchemaVersionNumber オブジェクト。

  チェックポイントに必要なバージョン番号。VersionNumber または Compatibility のいずれかを指定する必要があります。
- Compatibility – UTF-8 文字列 (有効な値: NONE | DISABLED | BACKWARD | BACKWARD_ALL | FORWARD | FORWARD_ALL | FULL | FULL_ALL)。

  スキーマの新しい互換性設定。
- Description – 説明文字列。2,048 バイト長以下。URI address multi-line string pattern に一致。

  スキーマの新しい説明。

レスポンス

- SchemaArn – UTF-8 文字列。1 ~ 10240 バイト長。Custom string pattern #49 に一致。

  スキーマの Amazon リソースネーム (ARN)。
- SchemaName – UTF-8 文字列。1 ~ 255 バイト長。Custom string pattern #45 に一致。

  スキーマの名前。
- RegistryName – UTF-8 文字列。1 ~ 255 バイト長。Custom string pattern #45 に一致。

スキーマを含むレジストリの名前。

エラー

- InvalidInputException
- AccessDeniedException
- EntityNotFoundException
- ConcurrentModificationException
- InternalServiceException

CheckSchemaVersionValidity アクション (Python: check_schema_version_validity)

指定されたスキーマを検証します。この呼び出しには副作用はなく、指定されたスキーマを使用して DataFormat を形式として使用して単に検証します。スキーマセット名を取らないため、互換性チェックは実行されません。

リクエスト

- DataFormat – 必須: UTF-8 文字列 (有効な値: AVRO | JSON | PROTOBUF)。

  スキーマ定義のデータ形式。現在サポートされている形式は、AVRO、JSON、および PROTOBUF です。

- SchemaDefinition – 必須: UTF-8 文字列。1 ~ 170000 バイト長。Custom string pattern #13 に一致。

  検証する必要があるスキーマの定義。

レスポンス

- Valid – ブール。

  スキーマが有効な場合は true を返し、そうでない場合は false を返します。

- Error – UTF-8 文字列。1 ~ 5000 バイト長。

  検証失敗のエラーメッセージ。

エラー

- InvalidInputException
- AccessDeniedException
- InternalServiceException

UpdateRegistry アクション (Python: update_registry)

スキーマのコレクションを保持するために使用される既存のレジストリを更新します。更新されたプロパティはレジストリに関連づけられており、レジストリ内のスキーマは変更されません。

リクエスト

- RegistryId – 必須: RegistryId オブジェクト。

  これは、レジストリ名と Amazon リソースネーム (ARN) を含む可能性があるラッパー構造です。
- Description – 必須: 説明文字列。2048 バイト長以下。URI address multi-line string pattern に一致。

  レジストリの説明。説明が指定されていない場合、このフィールドは更新されません。

レスポンス

- RegistryName – UTF-8 文字列。1 ~ 255 バイト長。Custom string pattern #45 に一致。

  更新されたレジストリの名前。
- RegistryArn – UTF-8 文字列。1 ~ 10240 バイト長。Custom string pattern #49 に一致。

  更新されたレジストリの Amazon リソースネーム (ARN)

エラー

- InvalidInputException
- AccessDeniedException
- EntityNotFoundException
- ConcurrentModificationException
- InternalServiceException

GetSchemaByDefinition アクション (Python: get_schema_by_definition)

SchemaDefinition によってスキーマを取得します。スキーマ定義は、スキーマレジストリに送信され、正規化され、ハッシュされます。ハッシュが SchemaName または ARN (指定されていない場合はデフォルトのレジストリ) のスコープ内で一致する場合、そのスキーマのメタデータが返されます。一致しない場合は、404 すなわち NotFound エラーが返されます。Deleted ステータスのスキーマバージョンは結果に含まれません。

リクエスト

- SchemaId – 必須: SchemaId オブジェクト。

  スキーマ ID フィールドを含むラッパー構造体です。構造体には以下が含まれています。

  - SchemaId$SchemaArn: スキーマの Amazon リソースネーム (ARN)。SchemaArn または SchemaName のいずれかを指定する必要があります。
  - SchemaId$SchemaName: スキーマの名前。SchemaArn または SchemaName のいずれかを指定する必要があります。
- SchemaDefinition – 必須: UTF-8 文字列。1 ~ 170000 バイト長。Custom string pattern #13 に一致。

  スキーマの詳細が必要なスキーマの定義。

レスポンス

- SchemaVersionId – UTF-8 文字列。36 バイト長ちょうど。Custom string pattern #44 に一致。

  スキーマバージョンのスキーマ ID。
- SchemaArn – UTF-8 文字列。1 ~ 10240 バイト長。Custom string pattern #49 に一致。

  スキーマの Amazon リソースネーム (ARN)。
- DataFormat – UTF-8 文字列 (有効な値: AVRO | JSON | PROTOBUF)。

  スキーマ定義のデータ形式。現在サポートされている形式は、AVRO、JSON、および PROTOBUF です。
- Status – UTF-8 文字列 (有効な値: AVAILABLE | PENDING | FAILURE | DELETING)。

  スキーマバージョンのステータス。
- CreatedTime – UTF–8 文字列。

スキーマが作成された日時。

エラー

- InvalidInputException
- AccessDeniedException
- EntityNotFoundException
- InternalServiceException

GetRegistry アクション (Python: get_registry)

指定されたレジストリの詳細について説明します。

リクエスト

- RegistryId – 必須: RegistryId オブジェクト。

  これは、レジストリ名と Amazon リソースネーム (ARN) を含む可能性があるラッパー構造です。

レスポンス

- RegistryName – UTF-8 文字列。1 ~ 255 バイト長。Custom string pattern #45 に一致。

  レジストリの名前。
- RegistryArn – UTF-8 文字列。1 ~ 10240 バイト長。Custom string pattern #49 に一致。

  レジストリの Amazon リソースネーム (ARN)
- Description – 説明文字列。2,048 バイト長以下。URI address multi-line string pattern に一致。

  レジストリの説明。
- Status – UTF-8 文字列 (有効な値: AVAILABLE | DELETING)。

  レジストリのステータス。
- CreatedTime – UTF–8 文字列。

  レジストリが作成された日時。
- UpdatedTime – UTF–8 文字列。

レジストリが更新された日時。

エラー

- InvalidInputException
- AccessDeniedException
- EntityNotFoundException
- InternalServiceException

PutSchemaVersionMetadata アクション (Python: put_schema_version_metadata)

指定されたスキーマバージョン ID のメタデータのキーバリューペアを配置します。各スキーマバージョンに対して最大 10 個のキーバリューペアを持つことができます。これらは、1 つ以上の呼び出しで追加できます。

リクエスト

- SchemaId – SchemaId オブジェクト。

  スキーマの一意の ID。
- SchemaVersionNumber – SchemaVersionNumber オブジェクト。

  スキーマのバージョン番号。
- SchemaVersionId – UTF-8 文字列。36 バイト長ちょうど。Custom string pattern #44 に一致。

  スキーマバージョンの一意のバージョン ID。
- MetadataKeyValue – 必須: MetadataKeyValuePair オブジェクト。

  メタデータキーの対応する値。

レスポンス

- SchemaArn – UTF-8 文字列。1 ~ 10240 バイト長。Custom string pattern #49 に一致。

  スキーマの Amazon リソースネーム (ARN)。

- SchemaName – UTF-8 文字列。1 ~ 255 バイト長。Custom string pattern #45 に一致。

  スキーマの名前。

- RegistryName – UTF-8 文字列。1 ~ 255 バイト長。Custom string pattern #45 に一致。

  レジストリの名前。

- LatestVersion – ブール。

  スキーマの最新バージョン。

- VersionNumber – 数値 (long)。1 ~ 100000。

  スキーマのバージョン番号。

- SchemaVersionId – UTF-8 文字列。36 バイト長ちょうど。Custom string pattern #44 に一致。

  スキーマバージョンの一意のバージョン ID。

- MetadataKey – UTF-8 文字列。1 ~ 128 バイト長。Custom string pattern #14 に一致。

  メタデータキー。

- MetadataValue – UTF-8 文字列。1 ~ 256 バイト長。Custom string pattern #14 に一致。

  メタデータキーの値。

エラー

- InvalidInputException
- AccessDeniedException
- AlreadyExistsException
- EntityNotFoundException
- ResourceNumberLimitExceededException

QuerySchemaVersionMetadata アクション (Python: query_schema_version_metadata)

スキーマバージョンのメタデータ情報をクエリします。

リクエスト

- SchemaId – SchemaId オブジェクト。

  スキーマ名と Amazon リソースネーム (ARN) を含む可能性のあるラッパー構造体。
- SchemaVersionNumber – SchemaVersionNumber オブジェクト。

  スキーマのバージョン番号。
- SchemaVersionId – UTF-8 文字列。36 バイト長ちょうど。Custom string pattern #44 に一致。

  スキーマバージョンの一意のバージョン ID。
- MetadataList – MetadataKeyValuePair オブジェクトの配列。

  キーバリューペアからメタデータを検索します。キーバリューペアが指定されていない場合は、すべてのメタデータ情報がフェッチされます。
- MaxResults – 数値 (整数)。1 ~ 50。

  1 ページあたりに必要な結果の最大数。値が指定されていない場合、これはデフォルトで 1 ページあたり 25 に設定されます。
- NextToken – UTF-8 文字列。

  継続トークン (これが継続呼び出しの場合)。

応答

- MetadataInfoMap – キーバリューペアのマップ配列。

  各キーは UTF-8 文字列、1 ~ 128 バイト長で、Custom string pattern #14 に一致します。

  各値は MetadataInfo オブジェクトです。

  メタデータキーおよび関連する値のマップ。
- SchemaVersionId – UTF-8 文字列。36 バイト長ちょうど。Custom string pattern #44 に一致。

  スキーマバージョンの一意のバージョン ID。
- NextToken – UTF-8 文字列。

  返されたトークンのリストをページ分割するための継続トークン。リストの現在のセグメントが最後のセグメントではない場合に返されます。

エラー

- InvalidInputException
- AccessDeniedException
- EntityNotFoundException

RemoveSchemaVersionMetadata アクション (Python: remove_schema_version_metadata)

指定されたスキーマバージョン ID のスキーマバージョンメタデータからキーバリューペアを削除します。

リクエスト

- SchemaId – SchemaId オブジェクト。

  スキーマ名と Amazon リソースネーム (ARN) を含む可能性のあるラッパー構造体。
- SchemaVersionNumber – SchemaVersionNumber オブジェクト。

  スキーマのバージョン番号。
- SchemaVersionId – UTF-8 文字列。36 バイト長ちょうど。Custom string pattern #44 に一致。

  スキーマバージョンの一意のバージョン ID。
- MetadataKeyValue – 必須: MetadataKeyValuePair オブジェクト。

  メタデータキーの値。

レスポンス

- SchemaArn – UTF-8 文字列。1 ~ 10240 バイト長。Custom string pattern #49 に一致。

  スキーマの Amazon リソースネーム (ARN)。
- SchemaName – UTF-8 文字列。1 ~ 255 バイト長。Custom string pattern #45 に一致。

  スキーマの名前。
- RegistryName – UTF-8 文字列。1 ~ 255 バイト長。Custom string pattern #45 に一致。

  レジストリの名前。

- LatestVersion – ブール。

  スキーマの最新バージョン。

- VersionNumber – 数値 (long)。1 ~ 100000。

  スキーマのバージョン番号。

- SchemaVersionId – UTF-8 文字列。36 バイト長ちょうど。Custom string pattern #44 に一致。

  スキーマバージョンのバージョン ID。

- MetadataKey – UTF-8 文字列。1 ~ 128 バイト長。Custom string pattern #14 に一致。

  メタデータキー。

- MetadataValue – UTF-8 文字列。1 ~ 256 バイト長。Custom string pattern #14 に一致。

  メタデータキーの値。

エラー

- InvalidInputException
- AccessDeniedException
- EntityNotFoundException

DeleteRegistry アクション (Python: delete_registry)

スキーマとそのすべてのバージョンを含め、レジストリ全体を削除します。削除オペレーションのステータスを取得するには、非同期呼び出し後に GetRegistry API を呼び出します。レジストリを削除すると、レジストリに対する UpdateRegistry、CreateSchema、UpdateSchema、および RegisterSchemaVersion API など、すべてのオンラインオペレーションが非アクティブ化されます。

リクエスト

- RegistryId – 必須: RegistryId オブジェクト。

  これは、レジストリ名と Amazon リソースネーム (ARN) を含む可能性があるラッパー構造です。

レスポンス

- RegistryName – UTF-8 文字列。1 ~ 255 バイト長。Custom string pattern #45 に一致。

  削除されるレジストリの名前。
- RegistryArn – UTF-8 文字列。1 ~ 10240 バイト長。Custom string pattern #49 に一致。

  削除されるレジストリの Amazon リソースネーム (ARN)。
- Status – UTF-8 文字列 (有効な値: AVAILABLE | DELETING)。

  レジストリのステータス。オペレーションが正常終了すると、Deleting ステータスを返します。

エラー

- InvalidInputException
- EntityNotFoundException
- AccessDeniedException
- ConcurrentModificationException

DeleteSchema アクション (Python: delete_schema)

スキーマセットとそのすべてのバージョンを含め、スキーマセット全体を削除します。削除オペレーションのステータスを取得するには、非同期呼び出し後に GetSchema API を呼び出します。レジストリを削除すると、スキーマに対するGetSchemaByDefinition および RegisterSchemaVersion API など、すべてのオンライン操作が非アクティブ化されます。

リクエスト

- SchemaId – 必須: SchemaId オブジェクト。

  これは、スキーマ名と Amazon リソースネーム (ARN) を含む可能性のあるラッパー構造です。

レスポンス

- SchemaArn – UTF-8 文字列。1 ~ 10240 バイト長。Custom string pattern #49 に一致。

  削除されるスキーマの Amazon リソースネーム (ARN)。

- SchemaName – UTF-8 文字列。1 ~ 255 バイト長。Custom string pattern #45 に一致。

  削除されるスキーマの名前。
- Status – UTF-8 文字列 (有効な値: AVAILABLE | PENDING | DELETING)。

  スキーマのステータス。

エラー

- InvalidInputException
- EntityNotFoundException
- AccessDeniedException
- ConcurrentModificationException

DeleteSchemaVersions アクション (Python: delete_schema_versions)

指定したスキーマからバージョンを削除します。バージョン番号または範囲を指定できます。互換モードが BACKWARDS_FULL など、必要なバージョンの削除を禁止している場合、エラーが返されます。この呼び出しの後に GetSchemaVersions API を呼び出すと、削除されたバージョンのステータスが一覧表示されます。

バージョン番号の範囲にチェックポイントが付加されたバージョンが含まれている場合、APIは 409 の競合を返し、削除を続行しません。このAPIを使用する前に、まず、DeleteSchemaCheckpoint APIを使用してチェックポイントを削除する必要があります。

DeleteSchemaVersions API を使用して、スキーマセット内の最初のスキーマバージョンを削除することはできません。最初のスキーマバージョンは、DeleteSchema API でのみ削除できます。この操作により、スキーマバージョンの下にアタッチされた SchemaVersionMetadata も削除されます。データベースでは必ず物理削除 (ハード削除) となります。

互換モードが BACKWARDS_FULL など、必要なバージョンの削除を禁止している場合、エラーが返されます。

リクエスト

- SchemaId – 必須: SchemaId オブジェクト。

  これは、スキーマ名と Amazon リソースネーム (ARN) を含む可能性のあるラッパー構造です。

- `Versions` – 必須: UTF-8 文字列。1 ~ 100000 バイト長。Custom string pattern #15 に一致。

  バージョン範囲は、次のような形式で指定することができます。

  - 単一のバージョン番号、5
  - 範囲、5-8: deletes versions 5, 6, 7, 8

応答

- `SchemaVersionErrors` – SchemaVersionErrorItem オブジェクトの配列。

  `SchemaVersionErrorItem` オブジェクトのリスト。それぞれにはエラーとスキーマのバージョンが含まれています。

エラー

- `InvalidInputException`
- `EntityNotFoundException`
- `AccessDeniedException`
- `ConcurrentModificationException`

ワークフロー

ワークフロー API では、AWS Glue でのワークフローの作成、更新、表示に関連するデータ型と API について説明します。ジョブ実行履歴には、ワークフローとジョブ実行のために 90 日間アクセスできます。

データ型

- JobNodeDetails の構造
- CrawlerNodeDetails 構造
- TriggerNodeDetails 構造
- Crawl 構造
- Node 構造
- Edge 構造
- Workflow 構造

- WorkflowGraph 構造
- WorkflowRun 構造
- WorkflowRunStatistics 構造
- StartingEventBatchCondition 構造
- Blueprint 構造
- BlueprintDetails 構造
- LastActiveDefinition 構造
- BlueprintRun 構造

JobNodeDetails の構造

ワークフロー内のジョブノードの詳細。

フィールド

- `JobRuns` – JobRun オブジェクトの配列。

  ジョブノードが表すジョブ実行に関する情報。

CrawlerNodeDetails 構造

ワークフロー内に存在するクローラノードの詳細。

フィールド

- `Crawls` – Crawl オブジェクトの配列。

  クロールノードが表すクロールのリスト。

TriggerNodeDetails 構造

ワークフロー内に存在するトリガーノードの詳細。

フィールド

- `Trigger` – Trigger トリガー) オブジェクト。

  トリガーノードが表すトリガーの情報。

Crawl 構造

ワークフロー内のクロールの詳細。

フィールド

- State - UTF-8 文字列 (有効値: RUNNING | CANCELLING | CANCELLED | SUCCEEDED | FAILED | ERROR).。

  クローラの状態。
- StartedOn – タイムスタンプ。

  クロールが開始された日時。
- CompletedOn – タイムスタンプ。

  クロールが完了した日時。
- ErrorMessage – 説明文字列。2,048 バイト長以下。URI address multi-line string pattern に一致。

  クロールに関連付けられたエラーメッセージ。
- LogGroup – UTF-8 文字列。1 ~ 512 バイト長。Log group string pattern に一致。

  クロールに関連付けられたロググループ。
- LogStream – UTF-8 文字列。1 ~ 512 バイト長。Log-stream string pattern に一致。

  クロールに関連付けられたログストリーム。

Node 構造

ノードは、ワークフローグラフ上で、AWS Glue コンポーネント (トリガー、クローラ、またはジョブ) を表しています。

フィールド

- Type – UTF-8 文字列 (有効な値: CRAWLER | JOB | TRIGGER)。

  ノードが表す AWS Glue コンポーネントのタイプ。
- Name – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

ノードが表す AWS Glue コンポーネントの名前。

- `UniqueId` – UTF-8 文字列。1～255 バイト長。Single-line string pattern に一致。

  ワークフロー内のノードに割り当てられた一意の ID。

- `TriggerDetails` – TriggerNodeDetails オブジェクト。

  ノードがトリガーを表すときのトリガーの詳細。

- `JobDetails` – JobNodeDetails オブジェクト。

  ノードがジョブを表すときのジョブの詳細。

- `CrawlerDetails` – CrawlerNodeDetails オブジェクト。

  ノードがクローラを表すときのクローラの詳細。

Edge 構造

エッジは、エッジが属するワークフローの一部である 2 つの AWS Glue コンポーネント間の有向接続を表します。

フィールド

- `SourceId` – UTF-8 文字列。1～255 バイト長。Single-line string pattern に一致。

  エッジの始点となるワークフロー内のノードの一意の ID。

- `DestinationId` – UTF-8 文字列。1～255 バイト長。Single-line string pattern に一致。

  エッジの終点となるワークフロー内のノードの一意の ID。

Workflow 構造

ワークフローは、複雑な ETL タスクを完了するために実行される複数の関連する AWS Glue ジョブとクローラのコレクションです。ワークフローは、それに含まれるすべてのジョブとクローラの実行とモニタリングを管理します。

フィールド

- `Name` – UTF-8 文字列。1～255 バイト長。Single-line string pattern に一致。

ワークフローの名前。

- `Description` – UTF–8 文字列。

  ワークフローの説明。

- `DefaultRunProperties` – キーバリューペアのマップ配列。

  各キーは UTF-8 文字列、1 ~ 255 バイト長で、Single-line string pattern に一致します。

  各値は UTF-8 文字列。

  ワークフローの各実行の一部として使用されるプロパティのコレクション。実行プロパティは、ワークフローの各ジョブで使用できるようになっています。ジョブは、フロー内の次のジョブのためにプロパティを変更できます。

- `CreatedOn` – タイムスタンプ。

  ワークフローが作成された日時。

- `LastModifiedOn` – タイムスタンプ。

  ワークフローが最終に変更された日時。

- `LastRun` – WorkflowRun オブジェクト。

  ワークフローの最終の実行に関する情報。

- `Graph` – WorkflowGraph オブジェクト。

  ワークフローに属するすべての AWS Glue コンポーネントをノードとして表し、これらのコンポーネント間の有向接続をエッジとして表すグラフ。

- `CreationStatus` – UTF-8 文字列 (有効な値: `CREATING` | `CREATED` | `CREATION_FAILED`)。

  ワークフローの作成ステータス。

- `MaxConcurrentRuns` – 数値 (整数)。

  このパラメータを使用すると、データが不必要に複数回更新されないようにしたり、コストを管理したり、場合によってはコンポーネントジョブの同時実行の最大数を超えないようにしたりできます。このパラメータを空白のままにした場合、ワークフロー実行の数に制限はありません。

- `BlueprintDetails` – BlueprintDetails オブジェクト。

  この構造は、この特定のワークフローの作成元となるブループリントの詳細を示します。

WorkflowGraph 構造

ワークフローグラフは、ワークフロー内に存在するすべての AWS Glue コンポーネントと、これらのコンポーネント間のすべての有向接続を含むワークフロー全体を表します。

フィールド

- Nodes – ノード オブジェクトの配列。

  ワークフロー内でノードとして表される AWS Glue コンポーネントのリスト。
- Edges – Edge オブジェクトの配列。

  ワークフローに属するノード間のすべての有向接続のリスト。

WorkflowRun 構造

ワークフロー実行は、すべてのランタイム情報を提供するワークフローの実行です。

フィールド

- Name – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  実行されたワークフローの名前。
- WorkflowRunId – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  このワークフロー実行の ID。
- PreviousRunId – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  前回のワークフロー実行の ID。
- WorkflowRunProperties – キーバリューペアのマップ配列。

  各キーは UTF-8 文字列、1 ~ 255 バイト長で、Single-line string pattern に一致します。

  各値は UTF-8 文字列。

  実行中に設定されたワークフロー実行のプロパティ。
- StartedOn – タイムスタンプ。

  ワークフロー実行の開始日時。
- CompletedOn – タイムスタンプ。

ワークフロー実行の完了日時。

- `Status` – UTF-8 文字列 (有効な値: `RUNNING` | `COMPLETED` | `STOPPING` | `STOPPED` | `ERROR`)。

  ワークフロー実行のステータス。

- `ErrorMessage` – UTF–8 文字列。

  このエラーメッセージは、ワークフローの実行の開始時に発生した可能性のあるエラーを示します。現在のところ、唯一のエラーメッセージは「Concurrent runs exceeded for workflow: `foo`」(ワークフローの同時実行数を超えました:) のみです。

- `Statistics` – WorkflowRunStatistics オブジェクト。

  実行の統計情報。

- `Graph` – WorkflowGraph オブジェクト。

  ワークフローに属するすべての AWS Glue コンポーネントをノードとして表し、これらのコンポーネント間の有向接続をエッジとして表すグラフ。

- `StartingEventBatchCondition` – StartingEventBatchCondition オブジェクト。

  ワークフロー実行を開始したバッチ条件。

WorkflowRunStatistics 構造

ワークフロー実行の統計は、ワークフロー実行に関する統計情報を提供します。

フィールド

- `TotalActions` – 数値 (整数)。

  ワークフロー実行のアクションの合計数。

- `TimeoutActions` – 数値 (整数)。

  タイムアウトしたアクションの合計数。

- `FailedActions` – 数値 (整数)。

  失敗したアクションの合計数。

- `StoppedActions` – 数値 (整数)。

  停止したアクションの合計数。

- SucceededActions – 数値 (整数)。

  成功したアクションの合計数。

- RunningActions – 数値 (整数)。

  実行中状態のアクションの合計数。

- ErroredActions – 数値 (整数)。

  ワークフロー実行で ERROR 状態のジョブ実行数を示します。

- WaitingActions – 数値 (整数)。

  ワークフロー実行で WAITING 状態のジョブ実行数を示します。

StartingEventBatchCondition 構造

ワークフロー実行を開始したバッチ条件。到着したイベント数がバッチサイズに到達したか (BatchSize メンバーがゼロ以外の場合)、バッチウィンドウの時間が経過したか (BatchWindow メンバーがゼロ以外の場合) のいずれかです。

フィールド

- BatchSize – 数値 (整数)。

  バッチ内のイベント数。

- BatchWindow – 数値 (整数)。

  バッチウィンドウの持続時間 (秒単位)。

Blueprint 構造

ブループリントの詳細。

フィールド

- Name – UTF-8 文字列。1 ~ 128 バイト長。Custom string pattern #31 に一致。

  ブループリントの名前。

- Description – UTF-8 文字列。1 ~ 512 バイト長。

ブループリントの説明。

- `CreatedOn` – タイムスタンプ。

  ブループリントが登録された日時。

- `LastModifiedOn` – タイムスタンプ。

  ブループリントの最終変更日時。

- `ParameterSpec` – UTF-8 文字列。1 ~ 131072 バイト長。

  ブループリントのパラメータ仕様のリストを示す JSON 文字列。

- `BlueprintLocation` – UTF–8 文字列。

  ブループリントを公開する Simple Storage Service (Amazon S3) のパスを指定します。

- `BlueprintServiceLocation` – UTF–8 文字列。

  `CreateBlueprint/UpdateBlueprint` を呼び出してブループリントを AWS Glue に登録するときに、ブループリントがコピーされる Simple Storage Service (Amazon S3) のパスを指定します。

- `Status` – UTF-8 文字列 (有効な値: `CREATING` | `ACTIVE` | `UPDATING` | `FAILED`)。

  ブループリント登録のステータス。

  - Creating – ブループリントの登録が進行中です。
  - Active – ブループリントは正常に登録されました。
  - Updating – ブループリント登録の更新が進行中です。
  - Failed – ブループリントの登録に失敗しました。

- `ErrorMessage` – UTF–8 文字列。

  エラーメッセージ。

- `LastActiveDefinition` – LastActiveDefinition オブジェクト。

  ブループリントに複数のバージョンがあり、最新バージョンに何らかのエラーがある場合、この属性は、サービスで使用できる正常な最後のブループリント定義を示します。

BlueprintDetails 構造

ブループリントの詳細。

フィールド

- `BlueprintName` – UTF-8 文字列。1 ~ 128 バイト長。Custom string pattern #31 に一致。

  ブループリントの名前。
- `RunId` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  このブループリントの実行 ID。

LastActiveDefinition 構造

ブループリントに複数のバージョンがあり、最新バージョンに何らかのエラーがある場合、この属性は、サービスで使用できる正常な最後のブループリント定義を示します。

フィールド

- `Description` – UTF-8 文字列。1 ~ 512 バイト長。

  ブループリントの説明。
- `LastModifiedOn` – タイムスタンプ。

  ブループリントの最終変更日時。
- `ParameterSpec` – UTF-8 文字列。1 ~ 131072 バイト長。

  ブループリントのパラメータを指定する JSON 文字列。
- `BlueprintLocation` – UTF–8 文字列。

  AWS Glue デベロッパーによってブループリントが公開される Simple Storage Service (Amazon S3) 内のパスを指定します。
- `BlueprintServiceLocation` – UTF–8 文字列。

  ブループリントの作成または更新時にコピーされる Simple Storage Service (Amazon S3) 内のパスを指定します。

BlueprintRun 構造

ブループリントの実行の詳細。

フィールド

- BlueprintName – UTF-8 文字列。1 ~ 128 バイト長。Custom string pattern #31 に一致。

  ブループリントの名前。

- RunId – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  このブループリント実行の実行 ID。

- WorkflowName – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  ブループリントが正常に実行された結果として作成されるワークフローの名前。ブループリントの実行にエラーがある場合、ワークフローは作成されません。

- State – UTF-8 文字列 (有効な値: RUNNING | SUCCEEDED | FAILED | ROLLING_BACK)。

  ブループリントの実行の状態。可能な値は以下のとおりです。

  - Running – ブループリントの実行が進行中です。
  - Succeeded – ブループリントの実行が正常に完了しました。
  - Failed – ブループリントの実行が失敗し、ロールバックが完了しました。
  - Rolling Back – ブループリントの実行が失敗し、ロールバックが進行中です。

- StartedOn – タイムスタンプ。

  ブループリントの実行が開始された日時。

- CompletedOn – タイムスタンプ。

  ブループリントの実行が完了した日時。

- ErrorMessage – UTF–8 文字列。

  ブループリントの実行中に発生したエラーを示します。

- RollbackErrorMessage – UTF–8 文字列。

  ワークフローのエンティティの作成中にエラーが発生した場合は、作成されたエンティティをその時点までロールバックして削除を試みます。この属性は、作成されたエンティティを削除しようとしたときにエラーが発生したことを示します。

- Parameters – UTF-8 文字列。1 ~ 131072 バイト長。

  文字列のブループリントパラメータです。Blueprint$ParameterSpec で定義されているパラメータ仕様から必要な各キーの値を指定する必要があります。

- `RoleArn` – UTF-8 文字列。1 ~ 1,024 バイト長。Custom string pattern #30 に一致。

  ロール ARN。このロールは、AWS Glue サービスが継承し、ワークフローとワークフローの他のエンティティを作成するために使用されます。

操作

- CreateWorkflow アクション (Python: create_workflow)
- UpdateWorkflow アクション (Python: update_workflow)
- DeleteWorkflow アクション (Python: delete_workflow)
- GetWorkflow アクション (Python: get_workflow)
- ListWorkflows アクション (Python: list_workflows)
- BatchGetWorkflows アクション (Python: batch_get_workflows)
- GetWorkflowRun アクション (Python: get_workflow_run)
- GetWorkflowRuns アクション (Python: get_workflow_runs)
- GetWorkflowRunProperties アクション (Python: get_workflow_run_properties)
- PutWorkflowRunProperties アクション (Python: put_workflow_run_properties)
- CreateBlueprint アクション (Python: create_blueprint)
- UpdateBlueprint アクション (Python: update_blueprint)
- DeleteBlueprint アクション (Python: delete_blueprint)
- ListBlueprints アクション (Python: list_blueprints)
- BatchGetBlueprints アクション (Python: batch_get_blueprints)
- StartBlueprintRun アクション (Python: start_blueprint_run)
- GetBlueprintRun アクション (Python: get_blueprint_run)
- GetBlueprintRuns アクション (Python: get_blueprint_runs)
- StartWorkflowRun アクション (Python: start_workflow_run)
- StopWorkflowRun アクション (Python: stop_workflow_run)
- ResumeWorkflowRun アクション (Python: resume_workflow_run)

CreateWorkflow アクション (Python: create_workflow)

新しいワークフローを作成します。

リクエスト

- Name – 必須: UTF-8 文字列。1～255 バイト長。Single-line string pattern に一致。

  ワークフローに割り当てる名前。アカウント内で一意である必要があります。

- Description - UTF-8 文字列。120,000 バイト長以下。

  ワークフローの説明。

- DefaultRunProperties – キーバリューペアのマップ配列。

  各キーは UTF-8 文字列、1～255 バイト長で、Single-line string pattern に一致します。

  各値は UTF-8 文字列。

  ワークフローの各実行の一部として使用されるプロパティのコレクション。

  実行プロパティがログに記録される場合があります。プレーンテキストのシークレットをプロパティとして渡さないでください。ワークフロー実行内でシークレットを使用する場合は、AWS Glue 接続、AWS Secrets Manager、または他のシークレット管理メカニズムからシークレットを取得します。

- Tags – キーと値のペアのマップ配列。50 ペア以下。

  各キーは UTF-8 文字列で、1～128 バイト長です。

  各値は UTF-8 文字列で、256 バイト長以下です。

  このワークフローで使用するタグ。

- MaxConcurrentRuns – 数値 (整数)。

  このパラメータを使用すると、データが不必要に複数回更新されないようにしたり、コストを管理したり、場合によってはコンポーネントジョブの同時実行の最大数を超えないようにしたりできます。このパラメータを空白のままにした場合、ワークフロー実行の数に制限はありません。

レスポンス

- Name – UTF-8 文字列。1～255 バイト長。Single-line string pattern に一致。

  リクエストの一部として指定されたワークフローの名前。

エラー

- `AlreadyExistsException`
- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`
- `ResourceNumberLimitExceededException`
- `ConcurrentModificationException`

UpdateWorkflow アクション (Python: update_workflow)

既存のワークフローを更新します。

リクエスト

- `Name` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  更新するワークフローの名前。
- `Description` - UTF-8 文字列。120,000 バイト長以下。

  ワークフローの説明。
- `DefaultRunProperties` – キーバリューペアのマップ配列。

  各キーは UTF-8 文字列、1 ~ 255 バイト長で、Single-line string pattern に一致します。

  各値は UTF-8 文字列。

  ワークフローの各実行の一部として使用されるプロパティのコレクション。

  実行プロパティがログに記録される場合があります。プレーンテキストのシークレットをプロパティとして渡さないでください。ワークフロー実行内でシークレットを使用する場合は、AWS Glue 接続、AWS Secrets Manager、または他のシークレット管理メカニズムからシークレットを取得します。
- `MaxConcurrentRuns` – 数値 (整数)。

  このパラメータを使用すると、データが不必要に複数回更新されないようにしたり、コストを管理したり、場合によってはコンポーネントジョブの同時実行の最大数を超えないようにしたりできます。このパラメータを空白のままにした場合、ワークフロー実行の数に制限はありません。

レスポンス

- Name – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  入力で指定されたワークフローの名前。

エラー

- InvalidInputException
- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException
- ConcurrentModificationException

DeleteWorkflow アクション (Python: delete_workflow)

ワークフローを削除します。

リクエスト

- Name – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  削除するワークフローの名前。

レスポンス

- Name – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  入力で指定されたワークフローの名前。

エラー

- InvalidInputException
- InternalServiceException
- OperationTimeoutException
- ConcurrentModificationException

GetWorkflow アクション (Python: get_workflow)

ワークフローのリソースメタデータを取得します。

リクエスト

- Name – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  取得するワークフローの名前。
- IncludeGraph – ブール。

  ワークフローのリソースメタデータを返すときに、グラフを含めるかどうかを指定します。

レスポンス

- Workflow – ワークフロー オブジェクト。

  ワークフローのリソースメタデータ。

エラー

- InvalidInputException
- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException

ListWorkflows アクション (Python: list_workflows)

アカウントで作成されたワークフローの名前を一覧表示します。

リクエスト

- NextToken – UTF-8 文字列。

  継続トークン (これが継続リクエストの場合)。
- MaxResults – 1 未満または 25 を超えない数値 (整数)。

  返されるリストの最大サイズ。

レスポンス

- Workflows – UTF-8 文字列の配列。1 ~ 25 個の文字列。

  アカウント内のワークフローの名前のリスト。
- NextToken – UTF–8 文字列。

  継続トークン (一部のワークフロー名が返されていない場合)。

エラー

- InvalidInputException
- InternalServiceException
- OperationTimeoutException

BatchGetWorkflows アクション (Python: batch_get_workflows)

特定のワークフロー名のリストに対応するリソースメタデータのリストを返します。ListWorkflows オペレーションを呼び出した後で、このオペレーションを呼び出すことで、アクセス許可が付与されているデータにアクセスできます。このオペレーションは、タグを使用するアクセス許可条件を含め、すべての IAM のアクセス許可をサポートします。

リクエスト

- Names – 必須: UTF-8 文字列の配列。1 ~ 25 個の文字列。

  ワークフロー名のリスト。これらの名前は ListWorkflows オペレーションから返される場合があります。
- IncludeGraph – ブール。

  ワークフローのリソースメタデータを返すときに、グラフを含めるかどうかを指定します。

レスポンス

- Workflows – ワークフロー オブジェクトの配列。1 ~ 25 個の構造。

  ワークフローのリソースメタデータのリスト。
- MissingWorkflows – UTF-8 文字列の配列。1 ~ 25 個の文字列。

検出されなかったワークフロー名のリスト。

エラー

- `InternalServiceException`
- `OperationTimeoutException`
- `InvalidInputException`

GetWorkflowRun アクション (Python: get_workflow_run)

特定のワークフロー実行のメタデータを取得します。ジョブ実行履歴には、ワークフローとジョブ実行のために 90 日間アクセスできます。

リクエスト

- `Name` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  実行中のワークフローの名前。
- `RunId` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  ワークフロー実行の ID。
- `IncludeGraph` – ブール。

  レスポンスにワークフローグラフを含めるかどうかを指定します。

レスポンス

- `Run` – WorkflowRun オブジェクト。

  リクエストされたワークフロー実行のメタデータ。

エラー

- `InvalidInputException`
- `EntityNotFoundException`
- `InternalServiceException`

- OperationTimeoutException

GetWorkflowRuns アクション (Python: get_workflow_runs)

特定のワークフローのすべての実行のメタデータを取得します。

リクエスト

- Name – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  返すべき実行のメタデータが属するワークフローの名前。
- IncludeGraph – ブール。

  レスポンスにワークフローグラフを含めるかどうかを指定します。
- NextToken – UTF–8 文字列。

  応答の最大サイズ。
- MaxResults – 1 ~ 1000 の数値 (整数)。

  レスポンスに含めるワークフロー実行の最大数。

レスポンス

- Runs – WorkflowRun オブジェクトの配列。1 ~ 1000 個の構造体。

  ワークフロー実行のメタデータオブジェクトのリスト。
- NextToken – UTF–8 文字列。

  継続トークン (リクエストしたワークフロー実行の一部が返されていない場合)。

エラー

- InvalidInputException
- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException

GetWorkflowRunProperties アクション (Python: get_workflow_run_properties)

実行中に設定されたワークフロー実行のプロパティを取得します。

リクエスト

- Name – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  実行されたワークフローの名前。
- RunId – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  返す必要がある実行プロパティが属するワークフロー実行の ID。

レスポンス

- RunProperties – キーバリューペアのマップ配列。

  各キーは UTF-8 文字列、1 ~ 255 バイト長で、Single-line string pattern に一致します。

  各値は UTF-8 文字列。

  指定した実行の進行中に設定されたワークフロー実行のプロパティ。

エラー

- InvalidInputException
- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException

PutWorkflowRunProperties アクション (Python: put_workflow_run_properties)

特定のワークフロー実行に対して、指定されたワークフロー実行プロパティを挿入します。指定された実行内に該当するプロパティが既に存在する場合は、既存の値が上書きされます。それ以外の場合は、既存のプロパティにプロパティが追加されます。

リクエスト

- Name – 必須: UTF-8 文字列。1～255 バイト長。Single-line string pattern に一致。

  実行されたワークフローの名前。

- RunId – 必須: UTF-8 文字列。1～255 バイト長。Single-line string pattern に一致。

  実行プロパティを更新する対象のワークフロー実行の ID。

- RunProperties – 必須: キーバリューペアのマップ配列。

  各キーは UTF-8 文字列、1～255 バイト長で、Single-line string pattern に一致します。

  各値は UTF-8 文字列。

  指定された実行に対して挿入するプロパティ。

  実行プロパティがログに記録される場合があります。プレーンテキストのシークレットをプロパティとして渡さないでください。ワークフロー実行内でシークレットを使用する場合は、AWS Glue 接続、AWS Secrets Manager、または他のシークレット管理メカニズムからシークレットを取得します。

レスポンス

- 応答パラメータはありません。

エラー

- AlreadyExistsException
- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException
- ResourceNumberLimitExceededException
- ConcurrentModificationException

CreateBlueprint アクション (Python: create_blueprint)

ブループリントを AWS Glue に登録します。

リクエスト

- Name – 必須: UTF-8 文字列。1 ~ 128 バイト長。Custom string pattern #31 に一致。

  ブループリントの名前。

- Description – UTF-8 文字列。1 ~ 512 バイト長。

  ブループリントの説明。

- BlueprintLocation – 必須: UTF-8 文字列。1 ~ 8192 バイト長。Custom string pattern #32 に一致。

  ブループリントを公開する Amazon S3 のパスを指定します。

- Tags – キーと値のペアのマップ配列。50 ペア以下。

  各キーは UTF-8 文字列で、1 ~ 128 バイト長です。

  各値は UTF-8 文字列で、256 バイト長以下です。

  このブループリントに適用されるタグ。

レスポンス

- Name – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  登録されたブループリントの名前を返します。

エラー

- AlreadyExistsException
- InvalidInputException
- OperationTimeoutException
- InternalServiceException
- ResourceNumberLimitExceededException

UpdateBlueprint アクション (Python: update_blueprint)

登録済みのブループリントを更新します。

リクエスト

- Name – 必須: UTF-8 文字列。1 ~ 128 バイト長。Custom string pattern #31 に一致。

  ブループリントの名前。

- Description – UTF-8 文字列。1 ~ 512 バイト長。

  ブループリントの説明。

- BlueprintLocation – 必須: UTF-8 文字列。1 ~ 8192 バイト長。Custom string pattern #32 に一致。

  ブループリントを公開する Amazon S3 のパスを指定します。

レスポンス

- Name – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  更新されたブループリントの名前を返します。

エラー

- EntityNotFoundException
- ConcurrentModificationException
- InvalidInputException
- OperationTimeoutException
- InternalServiceException
- IllegalBlueprintStateException

DeleteBlueprint アクション (Python: delete_blueprint)

既存のブループリントを削除します。

リクエスト

- Name – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  削除するブループリントの名前。

レスポンス

- Name – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  削除されたブループリントの名前を返します。

エラー

- InvalidInputException
- OperationTimeoutException
- InternalServiceException

ListBlueprints アクション (Python: list_blueprints)

アカウント内のすべてのブループリント名をリスト表示します。

リクエスト

- NextToken – UTF-8 文字列。

  継続トークン (これが継続リクエストの場合)。
- MaxResults – 1 未満または 25 を超えない数値 (整数)。

  返されるリストの最大サイズ。
- Tags – キーと値のペアのマップ配列。50 ペア以下。

  各キーは UTF-8 文字列で、1 ~ 128 バイト長です。

  各値は UTF-8 文字列で、256 バイト長以下です。

  リストを AWS リソースタグでフィルタリングします。

レスポンス

- Blueprints – UTF-8 文字列の配列。

  アカウント内のブループリントの名前のリスト。
- NextToken – UTF–8 文字列。

  継続トークン (返されていないワークフロー名がある場合)。

エラー

- InvalidInputException
- InternalServiceException
- OperationTimeoutException

BatchGetBlueprints アクション (Python: batch_get_blueprints)

ブループリントのリストに関する情報を取得します。

リクエスト

- Names – 必須: UTF-8 文字列の配列。1 ~ 25 個の文字列。

  ブループリント名のリスト。
- IncludeBlueprint – ブール。

  レスポンスにブループリントを含めるかどうかを指定します。
- IncludeParameterSpec – ブール。

  レスポンスにブループリントのパラメータを JSON 文字列として含めるかどうかを指定します。

応答

- Blueprints – Blueprint オブジェクトの配列。

  ブループリントのリストを Blueprints オブジェクトとして返します。
- MissingBlueprints – UTF-8 文字列の配列。

検出されなかった BlueprintNames のリストを返します。

エラー

- InternalServiceException
- OperationTimeoutException
- InvalidInputException

StartBlueprintRun アクション (Python: start_blueprint_run)

指定したブループリントの新しい実行を開始します。

リクエスト

- BlueprintName – 必須: UTF-8 文字列。1 ~ 128 バイト長。Custom string pattern #31 に一致。

  ブループリントの名前。
- Parameters – UTF-8 文字列。1 ~ 131072 バイト長。

  パラメータを BlueprintParameters オブジェクトとして指定します。
- RoleArn – 必須: UTF-8 文字列。1 ~ 1024 バイト長。Custom string pattern #30 に一致。

  ワークフローの作成に使用される IAM ロールを指定します。

レスポンス

- RunId – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  このブループリント実行の実行 ID。

エラー

- InvalidInputException
- OperationTimeoutException
- InternalServiceException
- ResourceNumberLimitExceededException

- EntityNotFoundException
- IllegalBlueprintStateException

GetBlueprintRun アクション (Python: get_blueprint_run)

ブループリント実行の詳細を取得します。

リクエスト

- BlueprintName – 必須: UTF-8 文字列。1 ~ 128 バイト長。Custom string pattern #31 に一致。

  ブループリントの名前。
- RunId – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  取得するブループリント実行の実行 ID。

レスポンス

- BlueprintRun – BlueprintRun オブジェクト。

  BlueprintRun オブジェクトを返します。

エラー

- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException

GetBlueprintRuns アクション (Python: get_blueprint_runs)

指定したブループリントでのブループリント実行の詳細を取得します。

リクエスト

- BlueprintName – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  ブループリントの名前。
- NextToken – UTF–8 文字列。

継続トークン (これが継続リクエストの場合)。

- MaxResults – 1 ~ 1000 の数値 (整数)。

  返されるリストの最大サイズ。

応答

- BlueprintRuns – BlueprintRun オブジェクトの配列。

  BlueprintRun オブジェクトのリストを返します。

- NextToken – UTF–8 文字列。

  継続トークン (一部のブループリント実行が返されていない場合)。

エラー

- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException
- InvalidInputException

StartWorkflowRun アクション (Python: start_workflow_run)

指定したワークフローの新しい実行を開始します。

リクエスト

- Name – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  開始するワークフローの名前。

- RunProperties – キーバリューペアのマップ配列。

  各キーは UTF-8 文字列、1 ~ 255 バイト長で、Single-line string pattern に一致します。

  各値は UTF-8 文字列。

  新しいワークフロー実行のためのワークフロー実行プロパティ。

実行プロパティがログに記録される場合があります。プレーンテキストのシークレットをプロパティとして渡さないでください。ワークフロー実行内でシークレットを使用する場合は、AWS Glue 接続、AWS Secrets Manager、または他のシークレット管理メカニズムからシークレットを取得します。

レスポンス

- RunId – UTF-8 文字列。1 ~ 255 バイト長。「Single-line string pattern」に一致。

  新しい実行の ID。

エラー

- InvalidInputException
- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException
- ResourceNumberLimitExceededException
- ConcurrentRunsExceededException

StopWorkflowRun アクション (Python: stop_workflow_run)

指定したワークフロー実行の実行を停止します。

リクエスト

- Name – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  停止するワークフローの名前。
- RunId – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  停止するワークフロー実行の ID。

レスポンス

- 応答パラメータはありません。

エラー

- InvalidInputException
- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException
- IllegalWorkflowStateException

ResumeWorkflowRun アクション (Python: resume_workflow_run)

以前に部分的に完了したワークフロー実行から一部のノードを再起動し、ワークフロー実行を再開します。選択したノードと、選択したノードの下流にあるすべてのノードが実行されます。

リクエスト

- Name – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  再開するワークフローの名前。

- RunId – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  再開するワークフロー実行の ID。

- NodeIds – 必須: UTF-8 文字列の配列。

  再起動するノードのノード ID のリスト。再起動するノードは、元の実行で実行を試行したことがなければなりません。

レスポンス

- RunId – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  再開されたワークフロー実行に割り当てられた新しい ID。ワークフロー実行を再開するたびに、新しい実行 ID が割り当てられます。

- NodeIds – UTF-8 文字列の配列。

  実際に再起動されたノードのノード ID のリスト。

エラー

- `InvalidInputException`
- `EntityNotFoundException`
- `InternalServiceException`
- `OperationTimeoutException`
- `ConcurrentRunsExceededException`
- `IllegalWorkflowStateException`

使用状況プロファイル

使用状況プロファイル API では、AWS Glue での使用状況プロファイルの作成、更新、表示に関連するデータ型と API について説明します。

データ型

- ProfileConfiguration structure
- ConfigurationObject structure
- UsageProfileDefinition structure

ProfileConfiguration structure

管理者が AWS Glue 使用状況プロファイルで設定するジョブとセッションの値を指定します。

[フィールド]

- `SessionConfiguration` – キーバリューペアのマップ配列。

  各キーは UTF-8 文字列、1～255 バイト長で、Single-line string pattern に一致します。

  各値は ConfigurationObject オブジェクトです。

  AWS Glue セッションの設定パラメータのキーと値のマップ。
- `JobConfiguration` – キーバリューペアのマップ配列。

  各キーは UTF-8 文字列、1～255 バイト長で、Single-line string pattern に一致します。

各値は ConfigurationObject オブジェクトです。

AWS Glue ジョブの設定パラメータのキーバリューマップ。

ConfigurationObject structure

AWS Glue 使用状況プロファイルで設定されたジョブまたはセッションパラメータごとに管理者が設定する値を指定します。

[フィールド]

- `DefaultValue` – UTF-8 文字列、1 ~ 128 バイト長、「Custom string pattern #35」に一致。

  パラメータのデフォルト値。
- `AllowedValues` – UTF-8 文字列の配列。

  パラメータの許容値リスト。
- `MinValue` – UTF-8 文字列、1 ~ 128 バイト長、「Custom string pattern #35」に一致。

  パラメータの最小許容値。
- `MaxValue` – UTF-8 文字列、1 ~ 128 バイト長、「Custom string pattern #35」に一致。

  パラメータの最大許容値。

UsageProfileDefinition structure

AWS Glue 使用状況プロファイルについて説明します。

[フィールド]

- `Name` – UTF-8 文字列、1 ~ 255 バイト長、Single-line string pattern に一致。

  使用状況プロファイルの名前。
- `Description` – 説明文字列、2048 バイト長以下、URI address multi-line string pattern に一致。

  使用状況プロファイルの説明。
- `CreatedOn` – タイムスタンプ。

  使用状況プロファイルが作成された日時。

- `LastModifiedOn` – タイムスタンプ。

  使用状況プロファイルの最終変更日時。

操作

- CreateUsageProfile action (Python: create_usage_profile)
- GetUsageProfile action (Python: get_usage_profile)
- UpdateUsageProfile action (Python: update_usage_profile)
- DeleteUsageProfile action (Python: delete_usage_profile)
- ListUsageProfiles action (Python: list_usage_profiles)

CreateUsageProfile action (Python: create_usage_profile)

AWS Glue 使用状況プロファイルを作成します。

リクエスト

- `Name` – 必須: UTF-8 文字列、1～255 バイト長、「Single-line string pattern」に一致。

  使用状況プロファイルの名前。
- `Description` – 説明文字列、2048 バイト長以下、URI address multi-line string pattern に一致。

  使用状況プロファイルの説明。
- `Configuration` – 必須: ProfileConfiguration オブジェクト。

  プロファイルのジョブ値とセッション値を指定する `ProfileConfiguration` オブジェクト。
- `Tags` – キーと値のペアのマップ配列。50 ペア以下。

  各キーは UTF-8 文字列で、1～128 バイト長です。

  各値は UTF-8 文字列で、256 バイト長以下です。

  使用状況プロファイルに適用されるタグのリスト。

レスポンス

- `Name` – UTF-8 文字列、1～255 バイト長、「Single-line string pattern」に一致。

作成された使用状況プロファイルの名前。

エラー

- `InvalidInputException`
- `InternalServiceException`
- `AlreadyExistsException`
- `OperationTimeoutException`
- `ResourceNumberLimitExceededException`
- `OperationNotSupportedException`

GetUsageProfile action (Python: get_usage_profile)

指定された AWS Glue 使用状況プロファイルに関する情報を取得します。

リクエスト

- `Name` – 必須: UTF-8 文字列、1～255 バイト長、「Single-line string pattern」に一致。

  取得する使用状況プロファイルの名前。

レスポンス

- `Name` – UTF-8 文字列、1～255 バイト長、「Single-line string pattern」に一致。

  使用状況プロファイルの名前。
- `Description` – 説明文字列、2048 バイト長以下、URI address multi-line string pattern に一致。

  使用状況プロファイルの説明。
- `Configuration` – ProfileConfiguration オブジェクト。

  プロファイルのジョブ値とセッション値を指定する `ProfileConfiguration` オブジェクト。
- `CreatedOn` – タイムスタンプ。

  使用状況プロファイルが作成された日時。
- `LastModifiedOn` – タイムスタンプ。

使用状況プロファイルの最終変更日時。

エラー

- `InvalidInputException`
- `InternalServiceException`
- `EntityNotFoundException`
- `OperationTimeoutException`
- `OperationNotSupportedException`

UpdateUsageProfile action (Python: update_usage_profile)

AWS Glue 使用状況プロファイルを更新します。

リクエスト

- Name – 必須: UTF-8 文字列、1 ~ 255 バイト長、「Single-line string pattern」に一致。

  使用状況プロファイルの名前。
- `Description` – 説明文字列、2048 バイト長以下、URI address multi-line string pattern に一致。

  使用状況プロファイルの説明。
- `Configuration` – 必須: ProfileConfiguration オブジェクト。

  プロファイルのジョブ値とセッション値を指定する `ProfileConfiguration` オブジェクト。

レスポンス

- Name – UTF-8 文字列、1 ~ 255 バイト長、「Single-line string pattern」に一致。

  更新された使用状況プロファイルの名前。

エラー

- `InvalidInputException`
- `InternalServiceException`

- EntityNotFoundException
- OperationTimeoutException
- OperationNotSupportedException
- ConcurrentModificationException

DeleteUsageProfile action (Python: delete_usage_profile)

AWS Glue の指定された使用状況プロファイルを削除します。

リクエスト

- Name – 必須: UTF-8 文字列、1 ~ 255 バイト長、「Single-line string pattern」に一致。

  削除された使用状況プロファイルの名前。

レスポンス

- 応答パラメータはありません。

エラー

- InvalidInputException
- InternalServiceException
- OperationTimeoutException
- OperationNotSupportedException

ListUsageProfiles action (Python: list_usage_profiles)

すべての AWS Glue 使用状況プロファイルを一覧表示します。

リクエスト

- NextToken - UTF-8 文字列。400,000 バイト長以下。

  継続トークン (これが継続呼び出しの場合)。
- MaxResults – 1 未満または 200 を超えない数値 (整数)。

1 回の応答で返される使用状況プロファイルの最大数。

応答

- `Profiles` – UsageProfileDefinition オブジェクトの配列。

  使用状況プロファイル (`UsageProfileDefinition`) オブジェクトのリスト。
- `NextToken` - UTF-8 文字列。400,000 バイト長以下。

  継続トークン (現在のリストセグメントが最後ではない場合に存在)。

エラー

- `InternalServiceException`
- `OperationTimeoutException`
- `InvalidInputException`
- `OperationNotSupportedException`

機械学習 API

機械学習 API では、機械学習のデータ型について説明します。これには、変換を作成、削除、更新したり、機械学習のタスク実行を開始したりするための API が含まれます。

データ型

- TransformParameters 構造
- EvaluationMetrics 構造
- MLTransform 構造
- FindMatchesParameters 構造
- FindMatchesMetrics 構造
- ConfusionMatrix 構造
- GlueTable 構造
- TaskRun 構造
- TransformFilterCriteria 構造

- TransformSortCriteria 構造
- TaskRunFilterCriteria 構造
- TaskRunSortCriteria 構造
- TaskRunProperties 構造
- FindMatchesTaskRunProperties 構造
- ImportLabelsTaskRunProperties 構造
- ExportLabelsTaskRunProperties 構造
- LabelingSetGenerationTaskRunProperties 構造
- SchemaColumn 構造
- TransformEncryption 構造
- MLUserDataEncryption 構造
- ColumnImportance 構造

TransformParameters 構造

機械学習変換に関連付けられたアルゴリズム固有のパラメータ。

フィールド

- `TransformType` – 必須: UTF-8 文字列 (有効な値: `FIND_MATCHES`)。

  機械学習変換のタイプ。

  機械学習変換のタイプの詳細については、「機械学習変換の作成」を参照してください。
- `FindMatchesParameters` – FindMatchesParameters オブジェクト。

  一致検索アルゴリズムのパラメータ。

EvaluationMetrics 構造

評価メトリクスは、機械学習変換の品質を推定します。

フィールド

- `TransformType` – 必須: UTF-8 文字列 (有効な値: `FIND_MATCHES`)。

  機械学習変換のタイプ。

- FindMatchesMetrics – FindMatchesMetrics オブジェクト。

  「一致の検索」アルゴリズムの評価メトリクス。

MLTransform 構造

機械学習変換の構造。

フィールド

- TransformId – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  機械学習変換用に生成される一意の変換 ID。ID は一意であることが保証されており、変更されません。
- Name – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  機械学習変換のユーザー定義名。名前は一意であることが保証されておらず、いつでも変更される可能性があります。
- Description – 説明文字列。2,048 バイト長以下。URI address multi-line string pattern に一致。

  機械学習変換に対する長い形式のユーザー定義による説明。説明は一意であることが保証されておらず、いつでも変更される可能性があります。
- Status – UTF-8 文字列 (有効な値: NOT_READY | READY | DELETING)。

  機械学習変換の現在のステータス。
- CreatedOn – タイムスタンプ。

  タイムスタンプ。この機械学習変換の作成日時。
- LastModifiedOn – タイムスタンプ。

  タイムスタンプ。この機械学習変換の最終変更日時。
- InputRecordTables – GlueTable オブジェクトの配列。構造体 10 個以下。

  変換で使用される AWS Glue テーブル定義のリスト。
- Parameters – TransformParameters オブジェクト。

  TransformParameters オブジェクト。パラメータを使用して機械学習変換の動作を調整 (カスタマイズ) できます。そのために、変換に学習させる要素と、さまざまなトレードオフ (適合率と再現率、精度とコストなど) の選好を指定します。

- `EvaluationMetrics` – EvaluationMetrics オブジェクト。

  `EvaluationMetrics` オブジェクト。評価メトリクスは、機械学習変換の品質を推定します。

- `LabelCount` – 数値 (整数)。

  この変換のために AWS Glue で生成されるラベリングファイルのカウント識別子。より適切な変換を作成する場合、ラベリングファイルのダウンロード、ラベル付け、アップロードを繰り返すことができます。

- `Schema` – SchemaColumn オブジェクトの配列。構造体 100 個以下。

  この変換の実行対象となる列とデータ型を表すキーバリューペアのマップ。100 列が上限です。

- `Role` – UTF–8 文字列。

  必要なアクセス許可を持つ IAM ロールの Amazon リソースネーム (ARN) の名前。必要なアクセス許可には、AWS Glue リソースに対する AWS Glue サービスロールのアクセス許可と、変換に必要な Amazon S3 アクセス許可の両方が含まれます。

  - このロールには、AWS Glue のリソースへのアクセスを許可する AWS Glue サービスロールのアクセス許可が必要です。「Attach a Policy to IAM Users That Access AWS Glue」を参照してください。
  - このロールは、この変換のタスク実行で使用される Amazon Simple Storage Service (Amazon S3) のソース、ターゲット、一時ディレクトリ、スクリプト、およびライブラリへのアクセス許可を必要とします。

- `GlueVersion` – UTF-8 文字列。1 ~ 255 バイト長。Custom string pattern #47 に一致。

  この値により、この機械学習変換と互換性がある AWS Glue のバージョンが決定します。ほとんどのお客様に、Glue 1.0 が推奨されます。値が設定されていない場合、Glue の互換性はデフォルトで Glue 0.9 に設定されます。詳細については、デベロッパーガイドの「AWS Glue Versions」を参照してください。

- `MaxCapacity` – 数値 (double)。

  この変換のタスク実行に割り当てられる AWS Glue データ処理ユニット (DPU) の数。DPU は、2 ~ 100 の範囲で割り当てることができます。デフォルトは 10 です。DPU は処理能力を相対的に測定するもので、4 個の vCPU のコンピューティング性能と 16 GB のメモリで構成されています。詳細については、「AWS Glue 料金表ページ」を参照してください。

  `MaxCapacity` は、 `NumberOfWorkers` と `WorkerType` との相互排他的なオプションです 。

- NumberOfWorkers または WorkerType のいずれかが設定されている場合、 MaxCapacity は設定できません。
- MaxCapacity が設定されている場合、NumberOfWorkers または WorkerType は設定できません。
- WorkerType が設定されている場合、NumberOfWorkers は必須です (逆も同様です) 。
- MaxCapacity と NumberOfWorkers は両方とも少なくとも 1 である必要があります。

WorkerType フィールドを Standard 以外の値に設定すると、MaxCapacity フィールドが自動的に設定され、読み取り専用になります。

- WorkerType – UTF-8 文字列 (有効な値: Standard="" | G.1X="" | G.2X="" | G.025X="" | G.4X="" | G.8X="" | Z.2X="")。

  この変換のタスク実行時に割り当てられる事前定義済みワーカーのタイプ。使用できる値は、Standard、G.1X、または G.2X です。

  - Standard ワーカータイプでは、各ワーカーは 4 vCPU、16 GB のメモリ、50 GB のディスク、ワーカーあたり 2 個の エグゼキュターを提供します。
  - G.1X ワーカータイプでは、各ワーカーは 4 vCPU、16 GB のメモリ、64 GB のディスク、ワーカーあたり 1 個のエグゼキュターを提供します。
  - G.2X ワーカータイプでは、各ワーカーは 8 vCPU、32 GB のメモリ、128 GB のディスク、ワーカーあたり 1 個のエグゼキュターを提供します。

  MaxCapacity は、 NumberOfWorkers と WorkerType との相互排他的なオプションです 。

  - NumberOfWorkers または WorkerType のいずれかが設定されている場合、 MaxCapacity は設定できません。
  - MaxCapacity が設定されている場合、NumberOfWorkers または WorkerType は設定できません。
  - WorkerType が設定されている場合、NumberOfWorkers は必須です (逆も同様です) 。
  - MaxCapacity と NumberOfWorkers は両方とも少なくとも 1 である必要があります。
- NumberOfWorkers – 数値 (整数)。

  変換のタスク実行時に割り当てられる定義済み workerType のワーカー数。

  WorkerType が設定されている場合、NumberOfWorkers は必須です (逆も同様です) 。
- Timeout – 数値 (整数)。1 以上。

  機械学習変換のタイムアウト (分単位)。

- `MaxRetries` – 数値 (整数)。

  機械学習変換の MLTaskRun が失敗した後の最大再試行回数。

- `TransformEncryption` – TransformEncryption オブジェクト。

  ユーザーデータへのアクセスに適用される変換の、保管時の暗号化設定。機械学習の変換では、KMS を使用して Amazon S3 で暗号化されたユーザーデータにアクセスできます。

FindMatchesParameters 構造

「一致検索」変換を設定するためのパラメータ。

フィールド

- `PrimaryKeyColumnName` – UTF-8 文字列。1 ~ 1024 バイト長。Single-line string pattern に一致。

  ソーステーブルの行を一意に識別する列の名前。一致するレコードを識別するために使用します。

- `PrecisionRecallTradeoff` – 数値 (double)。1.0 以下。

  変換を調整して適合率と再現率のバランスを取るときに選択する値。値 0.5 は指定なし、値 1.0 は純粋な適合率寄りとなり、値 0.0 は再現率寄りであることを意味します。これはトレードオフの関係にあるため、1.0 に近い値を選択すると、再現率が非常に低くなります。0.0 に近い値を選択すると、適合率が非常に低くなります。

  適合率メトリクスは、モデルが一致を推定して、その推定が正確である度合いを示します。

  再現率メトリクスは、実際の一致件数に対して、モデルが一致を推定した件数を示します。

- `AccuracyCostTradeoff` – 数値 (double)。1.0 以下。

  変換を調整して精度とコストのバランスを取るときに選択する値。値 0.5 は、システムが精度とコスト懸念のバランスを取っていることを意味します。値 1.0 は完全な精度を意味し、通常、コストが高くなります。著しく高くなる場合もあります。値 0.0 は、完全なコスト優先を意味し、FindMatches 変換の精度が低くなります。精度が許容できないレベルになる場合もあります。

  精度は、変換が真陽性と真陰性を正確に検索する度合いを測定します。精度を高めるには、マシンリソースとコストを増やす必要があります。ただし、再現率も高くなります。

コストは、変換を実行するために消費されるコンピューティングリソースの数およびそれに伴う金額を測定します。

- `EnforceProvidedLabels` – ブール。

  ユーザーから指定されたラベルに出力を一致させるためにオン/オフを切り替える値。値が `True` の場合、`find matches` 変換は指定されたラベルに出力を一致させます。この結果は通常の合成結果より優先されます。値が `False` の場合、`find matches` 変換は、すべての指定されたラベルを優先するとは限らず、結果はトレーニングされたモデルに依存します。

  この値を true に設定すると、合成の実行時間が長くなる場合があります。

FindMatchesMetrics 構造

「一致の検索」アルゴリズムの評価メトリクス。機械学習変換の品質を測定するには、変換によっていくつかの一致を推定し、その結果を同じデータセットの既知の一致と比較します。品質メトリクスは、データのサブセットに基づくものであるため、厳密ではありません。

フィールド

- `AreaUnderPRCurve` – 数値 (double)。1.0 以下。

  適合率/再現率曲線の下の部分の面積 (AUPRC) は、変換の品質全体を測定するための単一の数値です。適合率と再現率のために行われた選択とは関係ありません。値が高いほど、適合率と再現率のトレードオフが適切であることを示します。

  詳細については、Wikipedia の「適合率と再現率」を参照してください。

- `Precision` – 数値 (double)。1.0 以下。

  適合率メトリクスは、変換が一致を推定したときに、その推定が正しい度合いを示します。具体的には、真陽性の総数に対して、変換が実際に検索した真陽性の割合を測定します。

  詳細については、Wikipedia の「適合率と再現率」を参照してください。

- `Recall` – 数値 (double)。1.0 以下。

  再現率メトリクスは、実際の一致件数に対して、変換が推定した一致件数の割合です。具体的には、ソースデータのレコード総数に対して、変換が検索した真陽性の件数の割合です。

  詳細については、Wikipedia の「適合率と再現率」を参照してください。

- F1 – 数値 (double)。1.0 以下。

  最大の F1 メトリクスは、変換の精度 (1 ~ 0) を示します。値 1 は最適な精度です。

  詳細については、Wikipedia の「F1 スコア」を参照してください。
- `ConfusionMatrix` – ConfusionMatrix オブジェクト。

  混同行列は、変換が何を正確に推定しているか、どのような種類のエラーを犯しているかを示します。

  詳細については、Wikipedia の「混同行列」を参照してください。
- `ColumnImportances` – ColumnImportance オブジェクトの配列。構造体 100 個以下。

  重要度の降順でソートされている列重要度メトリクスを含む `ColumnImportance` 構造のリスト。

ConfusionMatrix 構造

混同行列は、変換が何を正確に推定しているか、どのような種類のエラーを犯しているかを示します。

詳細については、Wikipedia の「混同行列」を参照してください。

フィールド

- `NumTruePositives` – 数値 (long)。

  変換の混同行列において、変換が正確に検索したデータ内の一致の数。
- `NumFalsePositives` – 数値 (long)。

  変換の混同行列において、変換が間違って一致として分類したデータ内の不一致の数。
- `NumTrueNegatives` – 数値 (long)。

  変換の混同行列において、変換が正しく拒否したデータ内の不一致の数。
- `NumFalseNegatives` – 数値 (long)。

  変換の混同行列において、変換が検索しなかったデータ内の一致の数。

GlueTable 構造

入力データまたは出力データとして使用される AWS Glue Data Catalog 内のデータベースとテーブル。

フィールド

- `DatabaseName` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  AWS Glue Data Catalog のデータベース名。
- `TableName` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  AWS Glue Data Catalog のテーブル名。
- `CatalogId` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  AWS Glue Data Catalog 用の一意の識別子。
- `ConnectionName` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  AWS Glue Data Catalog への接続の名前。
- `AdditionalOptions` – キーと値のペアのマップ配列。1 ~ 10 個のペア。

  各キーは UTF-8 文字列、1 ~ 255 バイト長で、Single-line string pattern に一致します。

  各値は説明文字列であり、2048 バイト長以下で、URI address multi-line string pattern に一致します。

  テーブルの追加オプション。現在、次の 2 つのキーがサポートされています。
  - `pushDownPredicate`: データセットのすべてのファイルを一覧表示して読み込むことなく、パーティションをフィルタリングするためのキー。
  - `catalogPartitionPredicate`: AWS Glue Data Catalog のパーティションインデックスを使用して、サーバー側のパーティションプルーニングを使用するためのキー。

TaskRun 構造

機械学習変換に関連付けられているサンプリングパラメータ。

フィールド

- `TransformId` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

変換に対する一意の識別子。

- TaskRunId – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  このタスク実行に対する一意の識別子。

- Status – UTF-8 文字列 (有効な値: RUNNING | FINISHED | FAILED | PENDING_EXECUTION | TIMED_OUT | CANCELING | CANCELED | RECEIVED_BY_TASKRUNNER)。

  リクエストされたタスク実行の現在のステータス。

- LogGroupName – UTF–8 文字列。

  このタスク実行に関連付けられている、安全なログ記録用のロググループの名前。

- Properties – TaskRunProperties オブジェクト。

  このタスク実行に関連付けられている設定プロパティを指定します。

- ErrorString – UTF–8 文字列。

  このタスク実行に関連付けられているエラー文字列のリスト。

- StartedOn – タイムスタンプ。

  このタスク実行の開始日時。

- LastModifiedOn – タイムスタンプ。

  リクエストされたタスク実行の最終更新日時。

- CompletedOn – タイムスタンプ。

  リクエストされたタスク実行の最終完了日時。

- ExecutionTime – 数値 (整数)。

  タスク実行でリソースを消費した時間 (秒)。

TransformFilterCriteria 構造

機械学習変換のフィルタリング基準。

フィールド

- Name – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

機械学習変換のフィルタリングに使用する一意の変換名。

- `TransformType` – UTF-8 文字列 (有効な値: `FIND_MATCHES`)。

  機械学習変換のフィルタリングに使用する機械学習変換のタイプ。

- `Status` – UTF-8 文字列 (有効な値: `NOT_READY` | `READY` | `DELETING`)。

  変換の直近の既知のステータスを使用して機械学習変換のリストをフィルタリングします (変換が使用可能かどうかを示します)。有効な値は「NOT_READY」、「READY」、または「DELETING」です。

- `GlueVersion` – UTF-8 文字列。1 ~ 255 バイト長。Custom string pattern #47 に一致。

  この値により、この機械学習変換と互換性がある AWS Glue のバージョンが決定します。ほとんどのお客様に、Glue 1.0 が推奨されます。値が設定されていない場合、Glue の互換性はデフォルトで Glue 0.9 に設定されます。詳細については、デベロッパーガイドの「AWS Glue Versions」を参照してください。

- `CreatedBefore` – タイムスタンプ。

  この日時より前に作成された変換をフィルタリングします。

- `CreatedAfter` – タイムスタンプ。

  この日時より後に作成された変換をフィルタリングします。

- `LastModifiedBefore` – タイムスタンプ。

  この日付より前に最終変更された変換をフィルタリングします。

- `LastModifiedAfter` – タイムスタンプ。

  この日付より後に最終更新された変換をフィルタリングします。

- `Schema` – SchemaColumn オブジェクトの配列。構造体 100 個以下。

  特定のスキーマを持つデータセットをフィルタリングします。`Map<Column, Type>` オブジェクトは、この変換で使用できるスキーマを表すキーと値のペアの配列です。`Column` は列名、`Type` はデータ型 (整数や文字列など) です。100 列が上限です。

TransformSortCriteria 構造

機械学習変換に関連付けられている並べ替え基準。

フィールド

- Column – 必須: UTF-8 文字列 (有効な値: NAME | TRANSFORM_TYPE | STATUS | CREATED | LAST_MODIFIED)。

  機械学習変換に関連付けられている並べ替え基準で使用する列。

- SortDirection – 必須: UTF-8 文字列 (有効な値: DESCENDING | ASCENDING)。

  機械学習変換に関連付けられている並べ替え基準で使用する並べ替え方向。

TaskRunFilterCriteria 構造

機械学習変換のタスク実行のフィルタリング基準。

フィールド

- TaskRunType – UTF-8 文字列 (有効な値: EVALUATION | LABELING_SET_GENERATION | IMPORT_LABELS | EXPORT_LABELS | FIND_MATCHES)。

  タスク実行のタイプ。

- Status – UTF-8 文字列 (有効な値: RUNNING | FINISHED | FAILED | PENDING_EXECUTION | TIMED_OUT | CANCELING | CANCELED | RECEIVED_BY_TASKRUNNER)。

  タスク実行の現在のステータス。

- StartedBefore – タイムスタンプ。

  この日付より前に開始されたタスク実行をフィルタリングします。

- StartedAfter – タイムスタンプ。

  この日付より後に開始されたタスク実行をフィルタリングします。

TaskRunSortCriteria 構造

機械学習変換のタスク実行リストを並べ替えるために使用する並べ替え基準。

フィールド

- Column – 必須: UTF-8 文字列 (有効な値: TASK_RUN_TYPE | STATUS | STARTED)。

機械学習変換のタスク実行リストを並べ替えるために使用する列。

- SortDirection – 必須: UTF-8 文字列 (有効な値: DESCENDING | ASCENDING)。

  機械学習変換のタスク実行リストを並べ替えるために使用する並べ替え方向。

TaskRunProperties 構造

タスク実行の設定プロパティ。

フィールド

- TaskType – UTF-8 文字列 (有効な値: EVALUATION | LABELING_SET_GENERATION | IMPORT_LABELS | EXPORT_LABELS | FIND_MATCHES)。

  タスク実行のタイプ。

- ImportLabelsTaskRunProperties – ImportLabelsTaskRunProperties オブジェクト。

  「ラベルのインポート」タスク実行の設定プロパティ。

- ExportLabelsTaskRunProperties – ExportLabelsTaskRunProperties オブジェクト。

  「ラベルのエクスポート」タスク実行の設定プロパティ。

- LabelingSetGenerationTaskRunProperties – LabelingSetGenerationTaskRunProperties オブジェクト。

  「ラベリングセットの生成」タスク実行の設定プロパティ。

- FindMatchesTaskRunProperties – FindMatchesTaskRunProperties オブジェクト。

  「一致検索」タスク実行の設定プロパティ。

FindMatchesTaskRunProperties 構造

「一致検索」タスク実行の設定プロパティを指定します。

フィールド

- JobId – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  「一致検索」タスク実行のジョブ ID。

- JobName – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  「一致検索」タスク実行のジョブに割り当てられた名前。
- JobRunId – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  「一致検索」タスク実行のジョブ実行 ID。

ImportLabelsTaskRunProperties 構造

「ラベルのインポート」タスク実行の設定プロパティを指定します。

フィールド

- InputS3Path – UTF-8 文字列。

  ラベルをインポートする元の Amazon Simple Storage Service (Amazon S3) のパス。
- Replace – ブール。

  既存のラベルを上書きするかどうかを示します。

ExportLabelsTaskRunProperties 構造

「ラベルのエクスポート」タスク実行の設定プロパティを指定します。

フィールド

- OutputS3Path – UTF-8 文字列。

  ラベルをエクスポートする先の Amazon Simple Storage Service (Amazon S3) のパス。

LabelingSetGenerationTaskRunProperties 構造

「ラベリングセットの生成」タスク実行の設定プロパティを指定します。

フィールド

- OutputS3Path – UTF-8 文字列。

  ラベリングセットを生成する Amazon Simple Storage Service (Amazon S3) のパス。

SchemaColumn 構造

この変換の実行対象となる列とデータ型を表すキーバリューペア。MLTransform の Schema パラメータには、これらの構造を最大 100 個まで含めることができます。

フィールド

- Name – UTF-8 文字列。1 ~ 1024 バイト長。Single-line string pattern に一致。

  列の名前。
- DataType – UTF-8 文字列。131,072 バイト長以下。Single-line string pattern に一致。

  列のデータ型。

TransformEncryption 構造

ユーザーデータへのアクセスに適用される変換の、保管時の暗号化設定。機械学習の変換では、KMS を使用して Amazon S3 で暗号化されたユーザーデータにアクセスできます。

さらに、インポートされたラベルとトレーニングされた変換は、顧客が提供した KMS キーを使用して暗号化できるようになりました。

フィールド

- MlUserDataEncryption – MLUserDataEncryption オブジェクト。

  暗号化モードとお客様が用意した KMS キー ID MLUserDataEncryption を含むオブジェクト。
- TaskRunSecurityConfigurationName – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  セキュリティ構成の名前。

MLUserDataEncryption 構造

ユーザーデータへのアクセスに適用される変換の、保管時の暗号化設定。

フィールド

- MlUserDataEncryptionMode – 必須: UTF-8 文字列 (有効な値: DISABLED | SSE-KMS="SSEKMS")。

ユーザーデータに適用される暗号化モード。次の値を指定できます:

- DISABLED: 暗号化は無効です
- SSEKMS: Simple Storage Service (Amazon S3) に保存されたユーザーデータに、AWS Key Management Service (SSE-KMS) によるサーバー側暗号化を使用します。

- `KmsKeyId` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  ユーザーが用意した KMS キーの ID。

ColumnImportance 構造

列の列名と列重要度スコアを含む構造。

列の重要度は、レコード内のどの列が他の列よりも重要かを識別することで、モデルへの列貢献度を理解するのに役立ちます。

フィールド

- `ColumnName` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  列の名前。
- `Importance` – 数値 (double)。1.0 以下。

  列の列重要度スコア (10 進数)。

操作

- CreateMLTransform アクション (Python: create_ml_transform)
- UpdateMLTransform アクション (Python: update_ml_transform)
- DeleteMLTransform アクション (Python: delete_ml_transform)
- GetMLTransform アクション (Python: get_ml_transform)
- GetMLTransforms アクション (Python: get_ml_transforms)
- ListMLTransforms アクション(Python: list_ml_transforms)
- StartMLEvaluationTaskRun アクション (Python: start_ml_evaluation_task_run)
- StartMLLabelingSetGenerationTaskRun アクション (Python: start_ml_labeling_set_generation_task_run)

- GetMLTaskRun アクション (Python: get_ml_task_run)
- GetMLTaskRuns アクション (Python: get_ml_task_runs)
- CancelMLTaskRun アクション (Python: cancel_ml_task_run)
- StartExportLabelsTaskRun アクション (Python: start_export_labels_task_run)
- StartImportLabelsTaskRun アクション (Python: start_import_labels_task_run)

CreateMLTransform アクション (Python: create_ml_transform)

AWS Glue 機械学習変換を作成します。このオペレーションでは、変換を作成します。また、この変換をトレーニングするために必要なすべてのパラメータも作成します。

機械学習変換 (FindMatches 変換など) を使用してデータの重複を除去するプロセスの最初のステップとして、このオペレーションを呼び出します。アルゴリズムのために使用するパラメータに加えて、オプションの Description を指定できます。

データから学習して高品質の機械学習変換を作成する一環として、ユーザーに代わって AWS Glue が実行するタスクに対して特定のパラメータを指定することもできます。これらのパラメータには Role と、必要に応じて AllocatedCapacity、Timeout、MaxRetries が含まれます。詳細については、「ジョブ」を参照してください。

リクエスト

- Name – 必須: UTF-8 文字列。1～255 バイト長。Single-line string pattern に一致。

  変換の作成時に変換に付ける一意の名前。
- Description – 説明文字列。2,048 バイト長以下。URI address multi-line string pattern に一致。

  定義する機械学習変換の説明。デフォルトは空の文字列です。
- InputRecordTables – 必須: GlueTable オブジェクトの配列。構造 10 個以下。

  変換で使用される AWS Glue テーブル定義のリスト。
- Parameters – 必須: TransformParameters オブジェクト。

  使用する変換タイプに固有のアルゴリズムパラメータ。条件付きで変換タイプに依存します。
- Role – 必須: UTF–8 文字列。

必要なアクセス許可を持つ IAM ロールの Amazon リソースネーム (ARN) の名前。必要なアクセス許可には、AWS Glue リソースに対する AWS Glue サービスロールのアクセス許可と、変換に必要な Amazon S3 アクセス許可の両方が含まれます。

- このロールには、AWS Glue のリソースへのアクセスを許可する AWS Glue サービスロールのアクセス許可が必要です。「Attach a Policy to IAM Users That Access AWS Glue」を参照してください。
- このロールは、この変換のタスク実行で使用される Amazon Simple Storage Service (Amazon S3) のソース、ターゲット、一時ディレクトリ、スクリプト、およびライブラリへのアクセス許可を必要とします。

- `GlueVersion` – UTF-8 文字列。1 ~ 255 バイト長。Custom string pattern #47 に一致。

  この値により、この機械学習変換と互換性がある AWS Glue のバージョンが決定します。ほとんどのお客様に、Glue 1.0 が推奨されます。値が設定されていない場合、Glue の互換性はデフォルトで Glue 0.9 に設定されます。詳細については、デベロッパーガイドの「AWS Glue Versions」を参照してください。

- `MaxCapacity` – 数値 (double)。

  この変換のタスク実行に割り当てられる AWS Glue データ処理ユニット (DPU) の数。DPU は、2 ~ 100 の範囲で割り当てることができます。デフォルトは 10 です。DPU は処理能力を相対的に測定するもので、4 個の vCPU のコンピューティング性能と 16 GB のメモリで構成されています。詳細については、「AWS Glue 料金表ページ」を参照してください。

  `MaxCapacity` は、 `NumberOfWorkers` と `WorkerType` との相互排他的なオプションです 。

  - `NumberOfWorkers` または `WorkerType` のいずれかが設定されている場合、 `MaxCapacity` は設定できません。
  - `MaxCapacity` が設定されている場合、`NumberOfWorkers` または `WorkerType` は設定できません。
  - `WorkerType` が設定されている場合、`NumberOfWorkers` は必須です (逆も同様です) 。
  - `MaxCapacity` と `NumberOfWorkers` は両方とも少なくとも 1 である必要があります。

  `WorkerType` フィールドを `Standard` 以外の値に設定すると、`MaxCapacity` フィールドが自動的に設定され、読み取り専用になります。

  `WorkerType` フィールドを `Standard` 以外の値に設定すると、`MaxCapacity` フィールドが自動的に設定され、読み取り専用になります。

- WorkerType – UTF-8 文字列 (有効な値: Standard="" | G.1X="" | G.2X="" | G.025X="" | G.4X="" | G.8X="" | Z.2X="")。

  このタスクの実行時に割り当てられる事前定義済みワーカーのタイプ。使用できる値は、Standard、G.1X、または G.2X です。

  - Standard ワーカータイプでは、各ワーカーは 4 vCPU、16 GB のメモリ、50 GB のディスク、ワーカーあたり 2 個のエグゼキュターを提供します。
  - G.1X ワーカータイプでは、各ワーカーは 4 vCPU、16 GB のメモリ、64 GB のディスク、ワーカーあたり 1 個のエグゼキュターを提供します。
  - G.2X ワーカータイプでは、各ワーカーは 8 vCPU、32 GB のメモリ、128 GB のディスク、ワーカーあたり 1 個のエグゼキュターを提供します。

  MaxCapacity は、 NumberOfWorkers と WorkerType との相互排他的なオプションです 。

  - NumberOfWorkers または WorkerType のいずれかが設定されている場合、 MaxCapacity は設定できません。
  - MaxCapacity が設定されている場合、NumberOfWorkers または WorkerType は設定できません。
  - WorkerType が設定されている場合、NumberOfWorkers は必須です (逆も同様です) 。
  - MaxCapacity と NumberOfWorkers は両方とも少なくとも 1 である必要があります。
- NumberOfWorkers – 数値 (整数)。

  このタスクの実行時に割り当てられる定義済み workerType のワーカー数。

  WorkerType が設定されている場合、NumberOfWorkers は必須です (逆も同様です) 。
- Timeout – 数値 (整数)。1 以上。

  この変換のタスク実行のタイムアウト (分単位)。これは、この変換のタスク実行が終了して TIMEOUT ステータスに入るまでに、タスク実行がリソースを消費できる最大時間です。デフォルト値は 2,880 分 (48 時間) です。
- MaxRetries – 数値 (整数)。

  タスク実行の失敗後に、この変換のタスクを再試行する最大回数。
- Tags – キーと値のペアのマップ配列。50 ペア以下。

  各キーは UTF-8 文字列で、1 ~ 128 バイト長です。

  各値は UTF-8 文字列で、256 バイト長以下です。

この機械学習変換で使用するタグ。タグを使用して、機械学習変換へのアクセスを制限できます。AWS Glue のタグの詳細については、デベロッパーガイドの「AWS Tags in AWS Glue」を参照してください。

- `TransformEncryption` – TransformEncryption オブジェクト。

  ユーザーデータへのアクセスに適用される変換の、保管時の暗号化設定。機械学習の変換では、KMS を使用して Amazon S3 で暗号化されたユーザーデータにアクセスできます。

レスポンス

- `TransformId` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  変換用に生成された一意の識別子。

エラー

- `AlreadyExistsException`
- `InvalidInputException`
- `OperationTimeoutException`
- `InternalServiceException`
- `AccessDeniedException`
- `ResourceNumberLimitExceededException`
- `IdempotentParameterMismatchException`

UpdateMLTransform アクション (Python: update_ml_transform)

既存の機械学習変換を更新します。アルゴリズムパラメータを調整して、より良い結果を達成するには、このオペレーションを呼び出します。

このオペレーションを呼び出した後で、`StartMLEvaluationTaskRun` オペレーションを呼び出して新しいパラメータが目標を達成した度合い (機械学習変換の品質やコスト効率の向上など) を評価できます。

リクエスト

- `TransformId` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

変換の作成時に生成された一意の識別子。

- Name – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  変換の作成時に付けた一意の変換名。

- Description – 説明文字列。2,048 バイト長以下。URI address multi-line string pattern に一致。

  変換の説明。デフォルトは空の文字列です。

- Parameters – TransformParameters オブジェクト。

  使用する変換タイプ (アルゴリズム) に固有の設定パラメータ。条件付きで変換タイプに依存します。

- Role – UTF–8 文字列。

  必要なアクセス許可を持つ IAM ロールの Amazon リソースネーム (ARN) の名前。

- GlueVersion – UTF-8 文字列。1 ~ 255 バイト長。Custom string pattern #47 に一致。

  この値により、この機械学習変換と互換性がある AWS Glue のバージョンが決定します。ほとんどのお客様に、Glue 1.0 が推奨されます。値が設定されていない場合、Glue の互換性はデフォルトで Glue 0.9 に設定されます。詳細については、デベロッパーガイドの「AWS Glue Versions」を参照してください。

- MaxCapacity – 数値 (double)。

  この変換のタスク実行に割り当てられる AWS Glue データ処理ユニット (DPU) の数。DPU は、2 ~ 100 の範囲で割り当てることができます。デフォルトは 10 です。DPU は処理能力を相対的に測定するもので、4 個の vCPU のコンピューティング性能と 16 GB のメモリで構成されています。詳細については、「AWS Glue 料金表ページ」を参照してください。

  WorkerType フィールドを Standard 以外の値に設定すると、MaxCapacity フィールドが自動的に設定され、読み取り専用になります。

- WorkerType – UTF-8 文字列 (有効な値: Standard="" | G.1X="" | G.2X="" | G.025X="" | G.4X="" | G.8X="" | Z.2X="")。

  このタスクの実行時に割り当てられる事前定義済みワーカーのタイプ。使用できる値は、Standard、G.1X、または G.2X です。

  - Standard ワーカータイプでは、各ワーカーは 4 vCPU、16 GB のメモリ、50 GB のディスク、ワーカーあたり 2 個のエグゼキュターを提供します。

  - G.1X ワーカータイプでは、各ワーカーは 4 vCPU、16 GB のメモリ、64 GB のディスク、ワーカーあたり 1 個のエグゼキュターを提供します。
  - G.2X ワーカータイプでは、各ワーカーは 8 vCPU、32 GB のメモリ、128 GB のディスク、ワーカーあたり 1 個のエグゼキュターを提供します。
- NumberOfWorkers – 数値 (整数)。

  このタスクの実行時に割り当てられる定義済み workerType のワーカー数。
- Timeout - 数値 (整数)。1 以上。

  この変換のタスク実行のタイムアウト (分単位)。これは、この変換のタスク実行が終了して TIMEOUT ステータスに入るまでに、タスク実行がリソースを消費できる最大時間です。デフォルト値は 2,880 分 (48 時間) です。
- MaxRetries – 数値 (整数)。

  タスク実行の失敗後に、この変換のタスクを再試行する最大回数。

レスポンス

- TransformId – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  更新された変換の一意の識別子。

エラー

- EntityNotFoundException
- InvalidInputException
- OperationTimeoutException
- InternalServiceException
- AccessDeniedException

DeleteMLTransform アクション (Python: delete_ml_transform)

AWS Glue 機械学習変換を削除します。機械学習変換は、人が提供した例を反映して、実行すべき変換の詳細を機械学習を通じて学習する特殊なタイプの変換です。これらの変換は AWS Glue が保存します。変換が不要になった場合は、DeleteMLTransforms を呼び出して削除できます。ただし、AWS Glue ジョブで削除後の変換を参照しようとすると、正常に実行されません。

リクエスト

- `TransformId` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  削除する変換の一意の識別子。

レスポンス

- `TransformId` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  削除した変換の一意の識別子。

エラー

- `EntityNotFoundException`
- `InvalidInputException`
- `OperationTimeoutException`
- `InternalServiceException`

GetMLTransform アクション (Python: get_ml_transform)

AWS Glue 機械学習変換のアーティファクトおよびそのすべての対応するメタデータを取得します。機械学習変換は、人が提供した例を反映して、実行すべき変換の詳細を機械学習を通じて学習する特殊なタイプの変換です。これらの変換は AWS Glue が保存します。`GetMLTransform` を呼び出して変換のメタデータを取得できます。

リクエスト

- `TransformId` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  変換の一意の識別子。変換の作成時に生成されます。

レスポンス

- `TransformId` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  変換の一意の識別子。変換の作成時に生成されます。
- `Name` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

変換の作成時に付けた一意の変換名。

- `Description` – 説明文字列。2,048 バイト長以下。URI address multi-line string pattern に一致。

  変換の説明。

- `Status` – UTF-8 文字列 (有効な値: `NOT_READY` | `READY` | `DELETING`)。

  変換の直近の既知のステータス (変換が使用可能かどうかを示します)。有効な値は「NOT_READY」、「READY」、または「DELETING」です。

- `CreatedOn` – タイムスタンプ。

  変換の作成日時。

- `LastModifiedOn` – タイムスタンプ。

  変換の最終変更日時。

- `InputRecordTables` – GlueTable オブジェクトの配列。構造体 10 個以下。

  変換で使用される AWS Glue テーブル定義のリスト。

- `Parameters` – TransformParameters オブジェクト。

  使用するアルゴリズムに固有の設定パラメータ。

- `EvaluationMetrics` – EvaluationMetrics オブジェクト。

  最新の評価メトリクス。

- `LabelCount` – 数値 (整数)。

  この変換に使用できるラベルの数。

- `Schema` – SchemaColumn オブジェクトの配列。構造体 100 個以下。

  この変換で使用できるスキーマを表す `Map<Column, Type>` オブジェクト。100 列が上限です。

- `Role` – UTF–8 文字列。

  必要なアクセス許可を持つ IAM ロールの Amazon リソースネーム (ARN) の名前。

- `GlueVersion` – UTF-8 文字列。1 ~ 255 バイト長。Custom string pattern #47 に一致。

  この値により、この機械学習変換と互換性がある AWS Glue のバージョンが決定します。ほとんどのお客様に、Glue 1.0 が推奨されます。値が設定されていない場合、Glue の互換性はデフォル

トで Glue 0.9 に設定されます。詳細については、デベロッパーガイドの「AWS Glue Versions」を参照してください。

- `MaxCapacity` – 数値 (double)。

  この変換のタスク実行に割り当てられる AWS Glue データ処理ユニット (DPU) の数。DPU は、2 ~ 100 の範囲で割り当てることができます。デフォルトは 10 です。DPU は処理能力を相対的に測定するもので、4 個の vCPU のコンピューティング性能と 16 GB のメモリで構成されています。詳細については、「AWS Glue 料金表ページ」を参照してください。

  `WorkerType` フィールドを `Standard` 以外の値に設定すると、`MaxCapacity` フィールドが自動的に設定され、読み取り専用になります。

- `WorkerType` – UTF-8 文字列 (有効な値: `Standard=""` | `G.1X=""` | `G.2X=""` | `G.025X=""` | `G.4X=""` | `G.8X=""` | `Z.2X=""`)。

  このタスクの実行時に割り当てられる事前定義済みワーカーのタイプ。使用できる値は、Standard、G.1X、または G.2X です。

  - `Standard` ワーカータイプでは、各ワーカーは 4 vCPU、16 GB のメモリ、50 GB のディスク、ワーカーあたり 2 個のエグゼキュターを提供します。
  - `G.1X` ワーカータイプでは、各ワーカーは 4 vCPU、16 GB のメモリ、64 GB のディスク、ワーカーあたり 1 個のエグゼキュターを提供します。
  - `G.2X` ワーカータイプでは、各ワーカーは 8 vCPU、32 GB のメモリ、128 GB のディスク、ワーカーあたり 1 個のエグゼキュターを提供します。

- `NumberOfWorkers` – 数値 (整数)。

  このタスクの実行時に割り当てられる定義済み `workerType` のワーカー数。

- `Timeout` - 数値 (整数)。1 以上。

  この変換のタスク実行のタイムアウト (分単位)。これは、この変換のタスク実行が終了して `TIMEOUT` ステータスに入るまでに、タスク実行がリソースを消費できる最大時間です。デフォルト値は 2,880 分 (48 時間) です。

- `MaxRetries` – 数値 (整数)。

  タスク実行の失敗後に、この変換のタスクを再試行する最大回数。

- `TransformEncryption` – TransformEncryption オブジェクト。

  ユーザーデータへのアクセスに適用される変換の、保管時の暗号化設定。機械学習の変換では、KMS を使用して Amazon S3 で暗号化されたユーザーデータにアクセスできます。

エラー

- EntityNotFoundException
- InvalidInputException
- OperationTimeoutException
- InternalServiceException

GetMLTransforms アクション (Python: get_ml_transforms)

既存の AWS Glue 機械学習変換のソートおよびフィルタリング可能なリストを取得します。機械学習変換は、人が提供した例を反映して、実行すべき変換の詳細を機械学習を通じて学習する特殊なタイプの変換です。これらの変換は AWS Glue が保存します。変換のメタデータを取得するには GetMLTransforms を呼び出します。

リクエスト

- NextToken – UTF-8 文字列。

  結果をオフセットするページ分割されたトークン。

- MaxResults – 1 ~ 1000 の数値 (整数)。

  返される結果の最大数。

- Filter – TransformFilterCriteria オブジェクト。

  フィルター変換基準。

- Sort – TransformSortCriteria オブジェクト。

  並べ替え基準。

レスポンス

- Transforms – 必須: MLTransform オブジェクトの配列。

  機械学習変換のリスト。

- NextToken – UTF–8 文字列。

  ページネーショントークン (さらに結果がある場合)。

エラー

- EntityNotFoundException
- InvalidInputException
- OperationTimeoutException
- InternalServiceException

ListMLTransforms アクション(Python: list_ml_transforms)

この AWS アカウントの既存の AWS Glue 機械学習変換、または指定したタグの付いたリソースの、ソートおよびフィルタリング可能なリストを取得します。このオペレーションはオプションの Tags フィールドを受け取ります。このフィールドをレスポンスのフィルタとして使用すると、タグ付きリソースをグループとして取得できます。タグフィルタリングの使用を選択した場合は、タグが付いたリソースのみが取得されます。

リクエスト

- NextToken – UTF-8 文字列。

  継続トークン (これが継続リクエストの場合)。

- MaxResults – 1 ~ 1000 の数値 (整数)。

  返されるリストの最大サイズ。

- Filter – TransformFilterCriteria オブジェクト。

  機械学習変換のフィルタ処理に使用される TransformFilterCriteria。

- Sort – TransformSortCriteria オブジェクト。

  機械学習変換のソートに使用される TransformSortCriteria。

- Tags – キーと値のペアのマップ配列。50 ペア以下。

  各キーは UTF-8 文字列で、1 ~ 128 バイト長です。

  各値は UTF-8 文字列で、256 バイト長以下です。

  これらのタグ付きリソースのみを返すように指定します。

レスポンス

- TransformIds – 必須: UTF-8 文字列の配列。

  アカウントのすべての機械学習変換、または指定したタグの付いた機械学習変換の識別子。
- NextToken – UTF–8 文字列。

  継続トークン (戻されたリストに最後に使用可能なメトリクスが含まれていない場合)。

エラー

- EntityNotFoundException
- InvalidInputException
- OperationTimeoutException
- InternalServiceException

StartMLEvaluationTaskRun アクション (Python: start_ml_evaluation_task_run)

変換の品質を推定するためのタスクを開始します。

真の例としてラベルセットを指定すると、AWS Glue 機械学習は、これらの例のいくつかから学習します。残りのラベルは、品質を推定するためのテストとして使用されます。

実行用の一意の識別子を返します。GetMLTaskRun を呼び出して、EvaluationTaskRun の状態に関する詳細を取得できます。

リクエスト

- TransformId – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  機械学習変換の一意の識別子。

レスポンス

- TaskRunId – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  この実行に関連付けられた一意の識別子。

エラー

- EntityNotFoundException
- InvalidInputException
- OperationTimeoutException
- InternalServiceException
- ConcurrentRunsExceededException
- MLTransformNotReadyException

StartMLLabelingSetGenerationTaskRun アクション (Python: start_ml_labeling_set_generation_task_run)

機械学習変換のアクティブな学習ワークフローを開始し、ラベルセットを生成してラベルを追加することで、変換の品質を向上させます。

StartMLLabelingSetGenerationTaskRun が完了すると、AWS Glue によって「ラベリングセット」または人が回答する一連の質問が生成されます。

FindMatches 変換の場合、これらの質問は「これらの行を、一致するレコードのみで構成されるグループに分ける適切な方法は何ですか?」という形式になります。

ラベリングプロセスが完了すると、StartImportLabelsTaskRun を呼び出してラベルをアップロードできます。StartImportLabelsTaskRun が完了すると、今後のすべての機械学習変換の実行で新規および改善されたラベルが使用され、高品質の変換が実行されます。

注: 生成されたラベル付けセットを OutputS3Path に書き込むために使用されるロールは、CreateMLTransform APIで指定された機械学習変換に関連付けられたロールです。

リクエスト

- TransformId – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  機械学習変換の一意の識別子。

- OutputS3Path – 必須: UTF–8 文字列。

  ラベリングセットを生成する Amazon Simple Storage Service (Amazon S3) のパス。

レスポンス

- TaskRunId – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  このタスクに関連付けられている一意の実行識別子。

エラー

- EntityNotFoundException
- InvalidInputException
- OperationTimeoutException
- InternalServiceException
- ConcurrentRunsExceededException

GetMLTaskRun アクション (Python: get_ml_task_run)

機械学習変換に対する特定のタスク実行の詳細を取得します。機械学習のタスク実行は、さまざまな機械学習ワークフローの一環として AWS Glue がユーザーに代わって実行する非同期タスクです。任意のタスク実行の統計情報を確認するには、TaskRunID およびその親変換の TransformID を使用して GetMLTaskRun を呼び出します。

リクエスト

- TransformId – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  機械学習変換の一意の識別子。
- TaskRunId – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  タスク実行の一意の識別子。

レスポンス

- TransformId – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  タスク実行の一意の識別子。
- TaskRunId – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  この実行に関連付けられた一意の実行識別子。

- Status – UTF-8 文字列 (有効な値: RUNNING | FINISHED | FAILED | PENDING_EXECUTION | TIMED_OUT | CANCELING | CANCELED | RECEIVED_BY_TASKRUNNER)。

  このタスク実行のステータス。
- LogGroupName – UTF–8 文字列。

  タスク実行に関連付けられているロググループの名前。
- Properties – TaskRunProperties オブジェクト。

  タスク実行に関連付けられているプロパティのリスト。
- ErrorString – UTF–8 文字列。

  タスク実行に関連付けられているエラー文字列。
- StartedOn – タイムスタンプ。

  このタスク実行の開始日時。
- LastModifiedOn – タイムスタンプ。

  このタスク実行の最終変更日時。
- CompletedOn – タイムスタンプ。

  このタスク実行の完了日時。
- ExecutionTime – 数値 (整数)。

  タスク実行でリソースを消費した時間 (秒)。

エラー

- EntityNotFoundException
- InvalidInputException
- OperationTimeoutException
- InternalServiceException

GetMLTaskRuns アクション (Python: get_ml_task_runs)

機械学習変換の実行のリストを取得します。機械学習のタスク実行は、さまざまな機械学習ワークフローの一環として AWS Glue がユーザーに代わって実行する非同期タスクです。機械学習のタスク

実行の並べ替え可能およびフィルタリング可能なリストは、GetMLTaskRuns を呼び出して取得できます。この呼び出しで、親変換の TransformID とこのセクションに記載しているその他のオプションのパラメータを使用します。

このオペレーションは、実行履歴のリストを返します。また、ページ分割する必要があります。

リクエスト

- TransformId – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  機械学習変換の一意の識別子。
- NextToken – UTF–8 文字列。

  結果をページ分割するためのトークン。デフォルトは空です。
- MaxResults – 1 ~ 1000 の数値 (整数)。

  返される結果の最大数。
- Filter – TaskRunFilterCriteria オブジェクト。

  タスク実行のフィルタリング基準 (TaskRunFilterCriteria 構造)。
- Sort – TaskRunSortCriteria オブジェクト。

  タスク実行の並べ替え基準件 (TaskRunSortCriteria 構造)。

応答

- TaskRuns – TaskRun オブジェクトの配列。

  変換に関連付けられているタスク実行のリスト。
- NextToken – UTF–8 文字列。

  ページネーショントークン (さらに結果がある場合)。

エラー

- EntityNotFoundException
- InvalidInputException
- OperationTimeoutException

- InternalServiceException

CancelMLTaskRun アクション (Python: cancel_ml_task_run)

タスク実行をキャンセル (停止) します。機械学習のタスク実行は、さまざまな機械学習ワークフローの一環として AWS Glue がユーザーに代わって実行する非同期タスクです。機械学習のタスク実行は、CancelMLTaskRun を呼び出していつでもキャンセルできます。この呼び出しで、タスク実行の親変換の TransformID とタスク実行の TaskRunId を使用します。

リクエスト

- TransformId – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  機械学習変換の一意の識別子。
- TaskRunId – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  タスク実行に対する一意の識別子。

レスポンス

- TransformId – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  機械学習変換の一意の識別子。
- TaskRunId – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  タスク実行の一意の識別子。
- Status – UTF-8 文字列 (有効な値: RUNNING | FINISHED | FAILED | PENDING_EXECUTION | TIMED_OUT | CANCELING | CANCELED | RECEIVED_BY_TASKRUNNER)。

  この実行のステータス。

エラー

- EntityNotFoundException
- InvalidInputException
- OperationTimeoutException
- InternalServiceException

StartExportLabelsTaskRun アクション (Python: start_export_labels_task_run)

特定の変換に関するすべてのラベル付きデータをエクスポートするための非同期タスクを開始します。このタスクは、通常のアクティブな学習ワークフローに属さない唯一のラベル関連の API コールです。すべての既存のラベルを同時に操作するときは、通常、`StartExportLabelsTaskRun` を使用します (以前に真として送信したラベルを削除または変更する場合など)。この API オペレーションでは、エクスポートするラベルの `TransformId` と、ラベルをエクスポートする先の Amazon Simple Storage Service (Amazon S3) のパスを使用できます。このオペレーションは、`TaskRunId` を返します。タスク実行のステータスは、`GetMLTaskRun` API を呼び出して確認できます。

リクエスト

- `TransformId` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  機械学習変換の一意の識別子。

- `OutputS3Path` – 必須: UTF–8 文字列。

  ラベルをエクスポートする先となる Simple Storage Service (Amazon S3) へのパス。

レスポンス

- `TaskRunId` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  タスク実行の一意の識別子。

エラー

- `EntityNotFoundException`
- `InvalidInputException`
- `OperationTimeoutException`
- `InternalServiceException`

StartImportLabelsTaskRun アクション (Python: start_import_labels_task_run)

機械学習変換をトレーニングしてその品質を向上させるために使用する追加のラベル (真の例) を提供できます。この API オペレーションは、通常、StartMLLabelingSetGenerationTaskRun の呼び出しで始まり、最終的に機械学習変換の品質向上につながるアクティブな学習ワークフローの一環として使用します。

StartMLLabelingSetGenerationTaskRun が完了すると、AWS Glue 機械学習によって人が回答すべき一連の質問が生成されます。(これらの質問に回答することは、機械学習ワークフローにおいて「ラベリング」とも呼ばれます)。FindMatches 変換の場合、これらの質問は「これらの行を、一致するレコードのみで構成されるグループに分ける適切な方法は何ですか?」という形式になります。ラベリングプロセスが完了すると、ユーザーは StartImportLabelsTaskRun を呼び出して回答/ラベルをアップロードできます。StartImportLabelsTaskRun が完了すると、今後のすべての機械学習変換の実行で新規および改善されたラベルが使用され、より高品質の変換が実行されます。

デフォルトでは、 StartMLLabelingSetGenerationTaskRun は引き続き学習を継続し、ユーザーがアップロードしたすべてのラベルを結合します。ただし、Replace を true に設定した場合を除きます。Replace を true に設定すると、StartImportLabelsTaskRun は以前にアップロードされたすべてのラベルを削除して無視し、ユーザーがアップロードする正確なセットからのみ学習します。ラベルの置き換えが役立つのは、以前に間違ったラベルをアップロードしたことに気付き、これらが変換の品質に悪影響を及ぼしていると思われる場合です。

タスク実行のステータスは、GetMLTaskRun オペレーションを呼び出して確認できます。

リクエスト

- TransformId – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  機械学習変換の一意の識別子。

- InputS3Path – 必須: UTF–8 文字列。

  ラベルをインポートする元となる Amazon Simple Storage Service (Amazon S3) へのパス。

- ReplaceAllLabels – ブール。

  既存のラベルを上書きするかどうかを示します。

レスポンス

- `TaskRunId` – UTF-8 文字列。1～255 バイト長。Single-line string pattern に一致。

  タスク実行の一意の識別子。

エラー

- `EntityNotFoundException`
- `InvalidInputException`
- `OperationTimeoutException`
- `ResourceNumberLimitExceededException`
- `InternalServiceException`

Data Quality API

Data Quality API は、データ品質のデータ型を提供し、データ品質ルールセット、実行、および評価を作成、削除、または更新するための API を含んでいます。

データ型

- DataSource 構造
- DataQualityRulesetListDetails 構造
- DataQualityTargetTable 構造
- DataQualityRulesetEvaluationRunDescription 構造
- DataQualityRulesetEvaluationRunFilter 構造
- DataQualityEvaluationRunAdditionalRunOptions 構造
- DataQualityRuleRecommendationRunDescription 構造
- DataQualityRuleRecommendationRunFilter 構造
- DataQualityResult 構造
- DataQualityAnalyzerResult 構造
- DataQualityObservation 構造
- MetricBasedObservation 構造

- DataQualityMetricValues 構造
- DataQualityRuleResult 構造
- DataQualityResultDescription 構造
- DataQualityResultFilterCriteria 構造
- DataQualityRulesetFilterCriteria 構造
- DataQualityAggregatedMetrics 構造
- StatisticAnnotation structure
- TimestampedInclusionAnnotation structure
- AnnotationError 構造
- DatapointInclusionAnnotation structure
- StatisticSummaryList リスト
- StatisticSummary structure
- RunIdentifier 構造
- StatisticModelResult structure

DataSource 構造

データ品質結果が必要なデータソース (AWS Glue テーブル)。

フィールド

- `GlueTable` – 必須: GlueTable オブジェクト。

  AWS Glue テーブル。

DataQualityRulesetListDetails 構造

`GetDataQualityRuleset` によって返されるデータ品質ルールセットを記述します。

フィールド

- `Name` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  データ品質ルールセットの名前。
- `Description` – 説明文字列。2,048 バイト長以下。URI address multi-line string pattern に一致。

データ品質ルールセットの説明。

- `CreatedOn` – タイムスタンプ。

  データ品質ルールセットの作成日時。

- `LastModifiedOn` – タイムスタンプ。

  データ品質ルールセットの最終変更日時。

- `TargetTable` – DataQualityTargetTable オブジェクト。

  AWS Glue テーブルを表すオブジェクト。

- `RecommendationRunId` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  推奨実行からルールセットが作成されると、この 2 つをリンクさせるためにこの実行 ID が生成されます。

- `RuleCount` – 数値 (整数)。

  ルールセット内のルールの数。

DataQualityTargetTable 構造

AWS Glue テーブルを表すオブジェクト。

フィールド

- `TableName` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  AWS Glue テーブルの名前。

- `DatabaseName` – 必須: UTF-8 文字列、1 ~ 255 バイト長、Single-line string pattern に一致。

  AWS Glue テーブルが存在するデータベースの名前。

- `CatalogId` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  AWS Glue テーブルが存在するカタログの ID。

DataQualityRulesetEvaluationRunDescription 構造

データ品質ルールセットの評価実行の結果を記述します。

フィールド

- RunId – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  この実行に関連付けられた一意の実行識別子。
- Status – UTF-8 文字列 (有効な値: RUNNING | FINISHED | FAILED | PENDING_EXECUTION | TIMED_OUT | CANCELING | CANCELED | RECEIVED_BY_TASKRUNNER)。

  この実行のステータス。
- StartedOn – タイムスタンプ。

  実行が開始された日時。
- DataSource – DataSource オブジェクト。

  実行に関連付けられたデータソース (AWS Glue テーブル)。

DataQualityRulesetEvaluationRunFilter 構造

フィルタリング基準。

フィールド

- DataSource – 必須: DataSource オブジェクト。

  実行に関連付けられたデータソース (AWS Glue テーブル) に基づいてフィルタリングします。
- StartedBefore – タイムスタンプ。

  この時刻より前に開始された実行で結果をフィルタリングします。
- StartedAfter – タイムスタンプ。

  この時刻以降に開始された実行で結果をフィルタリングします。

DataQualityEvaluationRunAdditionalRunOptions 構造

評価実行で指定できるその他の実行オプション。

フィールド

- CloudWatchMetricsEnabled – ブール。

CloudWatch メトリクスを有効にするかどうか。

- `ResultsS3Prefix` – UTF–8 文字列。

  結果を保存する Amazon S3 のプレフィックス。

- `CompositeRuleEvaluationMethod` – UTF-8 文字列 (有効な値: `COLUMN` | `ROW`)。

  ルールセットの複合ルールの評価方法を ROW/COLUMN に設定する

DataQualityRuleRecommendationRunDescription 構造

データ品質ルールの推奨実行の結果を記述します。

フィールド

- `RunId` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  この実行に関連付けられた一意の実行識別子。

- `Status` – UTF-8 文字列 (有効な値: `RUNNING` | `FINISHED` | `FAILED` | `PENDING_EXECUTION` | `TIMED_OUT` | `CANCELING` | `CANCELED` | `RECEIVED_BY_TASKRUNNER`)。

  この実行のステータス。

- `StartedOn` – タイムスタンプ。

  この実行の開始日時。

- `DataSource` – DataSource オブジェクト。

  推奨実行に関連付けられたデータソース (AWS Glue テーブル)。

DataQualityRuleRecommendationRunFilter 構造

データ品質の推奨実行を一覧表示するためのフィルター。

フィールド

- `DataSource` – 必須: DataSource オブジェクト。

  指定したデータソース (AWS Glue テーブル) に基づいてフィルタリングします。

- `StartedBefore` – タイムスタンプ。

指定した時刻より前に開始された結果の場合、時刻に基づいてフィルタリングします。

- `StartedAfter` – タイムスタンプ。

  指定した時刻以降に開始された結果の場合、時刻に基づいてフィルタリングします。

DataQualityResult 構造

データ品質結果を記述します。

フィールド

- `ResultId` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  データ品質結果に対する一意の結果 ID。

- `ProfileId` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  データ品質結果のプロファイル ID。

- `Score` – 数値 (double)。1.0 以下。

  集計されたデータ品質のスコア。ルールの合計数に対する合格ルールの比率を表します。

- `DataSource` – DataSource オブジェクト。

  データ品質結果に関連付けられたテーブル (存在する場合)。

- `RulesetName` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  データ品質結果に関連付けられたルールセットの名前。

- `EvaluationContext` – UTF–8 文字列。

  AWS Glue Studio のジョブのコンテキストでは、通常、キャンバス内の各ノードには何らかの名前が割り当てられ、データ品質ノードには名前が付けられます。複数のノードの場合、`evaluationContext` でノードを区別できます。

- `StartedOn` – タイムスタンプ。

  このデータ品質の実行の開始日時。

- `CompletedOn` – タイムスタンプ。

  このデータ品質の実行の完了日時。

- JobName – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  データ品質結果に関連付けられたジョブ名 (存在する場合)。

- JobRunId – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  データ品質結果に関連付けられたジョブ実行 ID (存在する場合)。

- RulesetEvaluationRunId – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  このデータ品質結果のルールセット評価に対する一意の実行 ID。

- RuleResults – DataQualityRuleResult オブジェクトの配列。構造 2000 個以下。

  各ルールの結果を表す DataQualityRuleResult オブジェクトのリスト。

- AnalyzerResults – DataQualityAnalyzerResult オブジェクトの配列。構造 2000 個以下。

  各アナライザーの結果を表す DataQualityAnalyzerResult オブジェクトのリスト。

- Observations – DataQualityObservation オブジェクトの配列。構造 50 個以下。

  ルールとアナライザーを評価した後に生成された観測値を表す DataQualityObservation オブジェクトのリスト。

- AggregatedMetrics – DataQualityAggregatedMetrics オブジェクト。

  行レベルの結果に基づく合格/不合格統計を含む、処理された行とルールの合計数を示す DataQualityAggregatedMetrics オブジェクトの概要。

DataQualityAnalyzerResult 構造

データ品質アナライザーの評価の結果を説明します。

フィールド

- Name – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  データ品質アナライザーの名前。

- Description - UTF-8 文字列。2,048 バイト長以下。URI address multi-line string pattern に一致。

  データ品質アナライザーの説明。

- `EvaluationMessage` - UTF-8 文字列。2,048 バイト長以下。URI address multi-line string pattern に一致。

  評価メッセージ。
- `EvaluatedMetrics` – キーバリューペアのマップ配列。

  各キーは UTF-8 文字列、1～255 バイト長で、Single-line string pattern に一致します。

  各値は数値 (double) です。

  アナライザーの評価に関連付けられているメトリクスのマップ。

DataQualityObservation 構造

ルールおよびアナライザーを評価した後に生成される観測値について説明します。

フィールド

- `Description` - UTF-8 文字列。2,048 バイト長以下。URI address multi-line string pattern に一致。

  データ品質観察の説明。
- `MetricBasedObservation` – MetricBasedObservation オブジェクト。

  評価されたデータ品質メトリクスに基づく観測値を表す `MetricBasedObservation` タイプのオブジェクト。

MetricBasedObservation 構造

評価されたデータ品質メトリクスに基づいて生成されたメトリクスベースの観測値について説明します。

フィールド

- `MetricName` – UTF-8 文字列。1～255 バイト長。Single-line string pattern に一致。

  観測値を生成するために使用されるデータ品質メトリクスの名前。
- `StatisticId` – UTF-8 文字列。1～255 バイト長。Single-line string pattern に一致。

  統計 ID。

- `MetricValues` – DataQualityMetricValues オブジェクト。

  データ品質メトリクス値の分析を表す `DataQualityMetricValues` タイプのオブジェクト。

- `NewRules` – UTF-8 文字列の配列。

  データ品質メトリクス値に基づく観測値の一部として生成された新しいデータ品質ルールのリスト。

DataQualityMetricValues 構造

履歴データの分析に基づくデータ品質メトリクス値を説明します。

フィールド

- `ActualValue` – 数値 (double)。

  データ品質メトリクスの実際の値。

- `ExpectedValue` – 数値 (double)。

  履歴データの分析に基づくデータ品質メトリクスの期待値。

- `LowerLimit` – 数値 (double)。

  履歴データの分析に基づくデータ品質メトリクスの下限値。

- `UpperLimit` – 数値 (double)。

  履歴データの分析に基づくデータ品質メトリクスの上限値。

DataQualityRuleResult 構造

データ品質ルールの評価の結果を記述します。

フィールド

- `Name` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  データ品質ルールの名前。

- `Description` - UTF-8 文字列。2,048 バイト長以下。URI address multi-line string pattern に一致。

データ品質ルールの説明。

- `EvaluationMessage` - UTF-8 文字列。2,048 バイト長以下。URI address multi-line string pattern に一致。

  評価メッセージ。

- `Result` – UTF-8 文字列 (有効な値: `PASS` | `FAIL` | `ERROR`)。

  ルールの合格または不合格のステータス。

- `EvaluatedMetrics` – キーバリューペアのマップ配列。

  各キーは UTF-8 文字列、1 ~ 255 バイト長で、Single-line string pattern に一致します。

  各値は数値 (double) です。

  ルールの評価に関連するメトリクスのマップ。

- `EvaluatedRule` - UTF-8 文字列。2,048 バイト長以下。URI address multi-line string pattern に一致。

  評価済みのルール。

- `RuleMetrics` – キーバリューペアのマップ配列。

  各キーは UTF-8 文字列、1 ~ 255 バイト長で、Single-line string pattern に一致します。

  各値は数値 (double) です。

  行レベルの結果に基づくルールの評価に関連したメトリクスを含むマップ。

DataQualityResultDescription 構造

データ品質結果を記述します。

フィールド

- `ResultId` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  このデータ品質結果に対する一意の結果 ID。

- `DataSource` – DataSource オブジェクト。

  データ品質結果に関連付けられたテーブル名。

- JobName – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  データ品質結果に関連付けられたジョブ名。

- JobRunId – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  データ品質結果に関連付けられたジョブ実行 ID。

- StartedOn – タイムスタンプ。

  このデータ品質結果の実行が開始された時刻。

DataQualityResultFilterCriteria 構造

データ品質結果を返すために使用される基準。

フィールド

- DataSource – DataSource オブジェクト。

  指定したデータソースで結果をフィルタリングします。例えば、AWS Glue テーブルのすべての結果を取得します。

- JobName – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  指定したジョブ名で結果をフィルタリングします。

- JobRunId – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  指定したジョブ実行 ID で結果をフィルタリングします。

- StartedAfter – タイムスタンプ。

  この時刻以降に開始された実行で結果をフィルタリングします。

- StartedBefore – タイムスタンプ。

  この時刻より前に開始された実行で結果をフィルタリングします。

DataQualityRulesetFilterCriteria 構造

データ品質ルールセットのフィルタリングに使用される基準。

フィールド

- Name – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  ルールセットのフィルタリング基準の名前。
- Description – 説明文字列。2,048 バイト長以下。URI address multi-line string pattern に一致。

  ルールセットのフィルタリング基準の説明。
- CreatedBefore – タイムスタンプ。

  この日付より前に作成されたルールセットをフィルタリングします。
- CreatedAfter – タイムスタンプ。

  この日付以降に作成されたルールセットをフィルタリングします。
- LastModifiedBefore – タイムスタンプ。

  この日付より前に最終変更されたルールセットをフィルタリングします。
- LastModifiedAfter – タイムスタンプ。

  この日付より後に最終変更されたルールセットをフィルタリングします。
- TargetTable – DataQualityTargetTable オブジェクト。

  ターゲットテーブルの名前およびデータベース名。

DataQualityAggregatedMetrics 構造

行レベルの結果に基づく合格/不合格統計を含む、処理された行とルールの合計数を示すメトリクスの概要。

フィールド

- TotalRowsProcessed – 数値 (double)。

  データ品質評価中に処理された行の合計数。
- TotalRowsPassed – 数値 (double)。

  該当するすべてのデータ品質ルールに合格した行の合計数。
- TotalRowsFailed – 数値 (double)。

1 つ以上のデータ品質ルールに失敗した行の合計数。

- `TotalRulesProcessed` – 数値 (double)。

  評価されたデータ品質ルールの総数。

- `TotalRulesPassed` – 数値 (double)。

  評価基準に合格したデータ品質ルールの総数。

- `TotalRulesFailed` – 数値 (double)。

  評価基準を満たさなかったデータ品質ルールの総数。

StatisticAnnotation structure

統計注釈。

フィールド

- `ProfileId` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  プロファイル ID。

- `StatisticId` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  統計 ID。

- `StatisticRecordedOn` – タイムスタンプ。

  注釈付き統計が記録されたタイムスタンプ。

- `InclusionAnnotation` – TimestampedInclusionAnnotation オブジェクト。

  統計に適用される包含注釈。

TimestampedInclusionAnnotation structure

タイムスタンプ付き包含アノテーション。

フィールド

- `Value` – UTF-8 文字列 (有効な値: `INCLUDE` | `EXCLUDE`)。

包含注釈値。

- `LastModifiedOn` – タイムスタンプ。

  包含注釈が最後に変更されたタイムスタンプ。

AnnotationError 構造

失敗した注釈。

フィールド

- `ProfileId` – UTF-8 文字列。1～255 バイト長。Single-line string pattern に一致。

  失敗した注釈のプロファイル ID。

- `StatisticId` – UTF-8 文字列。1～255 バイト長。Single-line string pattern に一致。

  失敗した注釈の統計 ID。

- `FailureReason` – 説明文字列。2,048 バイト長以下。URI address multi-line string pattern に一致。

  注釈が失敗した理由。

DatapointInclusionAnnotation structure

包含アノテーション。

フィールド

- `ProfileId` – UTF-8 文字列。1～255 バイト長。Single-line string pattern に一致。

  統計が属するデータ品質プロファイルの ID。

- `StatisticId` – UTF-8 文字列。1～255 バイト長。Single-line string pattern に一致。

  統計 ID。

- `InclusionAnnotation` – UTF-8 文字列 (有効な値: `INCLUDE` | `EXCLUDE`)。

  統計に適用する包含注釈値。

StatisticSummaryList リスト

StatisticSummary のリスト。

StatisticSummary オブジェクトの配列。

StatisticSummary のリスト。

StatisticSummary structure

統計に関する概要情報。

フィールド

- StatisticId – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  統計 ID。
- ProfileId – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  プロファイル ID。
- RunIdentifier – RunIdentifier オブジェクト。

  実行識別子
- StatisticName – UTF-8 文字列。1 ~ 255 バイト長。Custom string pattern #16 に一致。

  統計の名前。
- DoubleValue – 数値 (double)。

  統計の値。
- EvaluationLevel – UTF-8 文字列 (有効な値: Dataset="DATASET" | Column="COLUMN" | Multicolumn="MULTICOLUMN")。

  統計の評価レベル。使用できる値: Dataset、Column、Multicolumn。
- ColumnsReferenced – UTF-8 文字列の配列。

  統計によって参照される列のリスト。
- ReferencedDatasets – UTF-8 文字列の配列。

  統計によって参照されるデータセットのリスト。

- StatisticProperties – キーバリューペアのマップ配列。

  各キーは UTF-8 文字列、1～255 バイト長で、Single-line string pattern に一致します。

  各値は説明文字列であり、2048 バイト長以下で、URI address multi-line string pattern に一致します。

  NameString と DescriptionString を含む StatisticPropertiesMap
- RecordedOn – タイムスタンプ。

  統計が記録されたタイムスタンプ。
- InclusionAnnotation – TimestampedInclusionAnnotation オブジェクト。

  統計の包含注釈。

RunIdentifier 構造

実行識別子。

フィールド

- RunId – UTF-8 文字列。1～255 バイト長。Single-line string pattern に一致。

  実行 ID。
- JobRunId – UTF-8 文字列。1～255 バイト長。Single-line string pattern に一致。

  ジョブ実行 ID。

StatisticModelResult structure

統計モデルの結果。

フィールド

- LowerBound – 数値 (double)。

  下限。
- UpperBound – 数値 (double)。

  上限。

- `PredictedValue` – 数値 (double)。

  予測された値。

- `ActualValue` – 数値 (double)。

  実際の値。

- `Date` – タイムスタンプ。

  日付。

- `InclusionAnnotation` – UTF-8 文字列 (有効な値: `INCLUDE` | `EXCLUDE`)。

  包含アノテーション。

操作

- StartDataQualityRulesetEvaluationRun アクション (Python: start_data_quality_ruleset_evaluation_run)
- CancelDataQualityRulesetEvaluationRun action (Python: cancel_data_quality_ruleset_evaluation_run)
- GetDataQualityRulesetEvaluationRun action (Python: get_data_quality_ruleset_evaluation_run)
- ListDataQualityRulesetEvaluationRuns action (Python: list_data_quality_ruleset_evaluation_runs)
- StartDataQualityRuleRecommendationRun action (Python: start_data_quality_rule_recommendation_run)
- CancelDataQualityRuleRecommendationRun action (Python: cancel_data_quality_rule_recommendation_run)
- GetDataQualityRuleRecommendationRun action (Python: get_data_quality_rule_recommendation_run)
- ListDataQualityRuleRecommendationRuns action (Python: list_data_quality_rule_recommendation_runs)
- GetDataQualityResult action (Python: get_data_quality_result)
- BatchGetDataQualityResult action (Python: batch_get_data_quality_result)
- ListDataQualityResults action (Python: list_data_quality_results)
- CreateDataQualityRuleset action (Python: create_data_quality_ruleset)
- DeleteDataQualityRuleset action (Python: delete_data_quality_ruleset)

- GetDataQualityRuleset action (Python: get_data_quality_ruleset)
- ListDataQualityRulesets action (Python: list_data_quality_rulesets)
- UpdateDataQualityRuleset アクション (Python: update_data_quality_ruleset)
- ListDataQualityRulesets action (Python: list_data_quality_statistics)
- TimestampFilter 構造
- CreateDataQualityRulesetRequest structure
- GetDataQualityRulesetResponse structure
- GetDataQualityResultResponse structure
- StartDataQualityRuleRecommendationRunRequest structure
- GetDataQualityRuleRecommendationRunResponse structure
- BatchPutDataQualityStatisticAnnotation action (Python: batch_put_data_quality_statistic_annotation)
- GetDataQualityModel アクション (Python: get_data_quality_model)
- GetDataQualityModelResult action (Python: get_data_quality_model_result)
- ListDataQualityStatisticAnnotations action (Python: list_data_quality_statistic_annotations)
- PutDataQualityProfileAnnotation action (Python: put_data_quality_profile_annotation)

StartDataQualityRulesetEvaluationRun アクション (Python: start_data_quality_ruleset_evaluation_run)

ルールセット定義 (推奨または独自のもの) を作成したら、この操作を呼び出して、ルールセットをデータソース (AWS Glue テーブル) に対して評価します。`GetDataQualityResult` API で取得できる結果は評価で計算されます。

リクエスト

- `DataSource` – 必須: DataSource オブジェクト。

  この実行に関連付けられたデータソース (AWS Glue テーブル)。

- `Role` – 必須: UTF-8 文字列。

  実行の結果を暗号化するために提供される IAM ロール。

- `NumberOfWorkers` – 数値 (整数)。

実行に使用される G.1X ワーカーの数。デフォルトは 5 です。

- Timeout – 数値 (整数)。1 以上。

  実行のタイムアウト (分)。これは、実行が終了済みになって TIMEOUT ステータスに入るまでに、その実行でリソースを消費できる最大時間です。デフォルト値は 2,880 分 (48 時間) です。

- ClientToken – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  冪等性を保つために使用されるもので、同じリソースの複数のインスタンスを作成または起動しないように、ランダムな ID (UUID など) に設定することをお勧めします。

- AdditionalRunOptions – DataQualityEvaluationRunAdditionalRunOptions オブジェクト。

  評価実行で指定できるその他の実行オプション。

- RulesetNames – 必須: UTF-8 文字列の配列、1 ~ 10 個の文字列。

  ルールセット名のリスト。

- AdditionalDataSources – キーバリューペアのマップ配列。

  各キーは UTF-8 文字列、1 ~ 255 バイト長で、Single-line string pattern に一致します。

  各値は DataSource オブジェクトです。

  評価実行で指定できる他のデータソースへの参照文字列のマップ。

レスポンス

- RunId – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  この実行に関連付けられた一意の実行識別子。

エラー

- InvalidInputException
- EntityNotFoundException
- OperationTimeoutException
- InternalServiceException
- ConflictException

CancelDataQualityRulesetEvaluationRun action (Python: cancel_data_quality_ruleset_evaluation_run)

ルールセットがデータソースに対して評価されている実行をキャンセルします。

リクエスト

- RunId – 必須: UTF-8 文字列。1～255 バイト長。Single-line string pattern に一致。

  この実行に関連付けられた一意の実行識別子。

レスポンス

- 応答パラメータはありません。

エラー

- EntityNotFoundException
- InvalidInputException
- OperationTimeoutException
- InternalServiceException

GetDataQualityRulesetEvaluationRun action (Python: get_data_quality_ruleset_evaluation_run)

ルールセットがデータソースに対して評価される特定の実行を取得します。

リクエスト

- RunId – 必須: UTF-8 文字列。1～255 バイト長。Single-line string pattern に一致。

  この実行に関連付けられた一意の実行識別子。

レスポンス

- RunId – UTF-8 文字列。1～255 バイト長。Single-line string pattern に一致。

  この実行に関連付けられた一意の実行識別子。

- DataSource – DataSource オブジェクト。

  この評価実行に関連付けられたデータソース (AWS Glue テーブル)。

- Role – UTF–8 文字列。

  実行の結果を暗号化するために提供される IAM ロール。

- NumberOfWorkers – 数値 (整数)。

  実行に使用される G.1X ワーカーの数。デフォルトは 5 です。

- Timeout – 数値 (整数)。1 以上。

  実行のタイムアウト (分)。これは、実行が終了済みになって TIMEOUT ステータスに入るまでに、その実行でリソースを消費できる最大時間です。デフォルト値は 2,880 分 (48 時間) です。

- AdditionalRunOptions – DataQualityEvaluationRunAdditionalRunOptions オブジェクト。

  評価実行で指定できるその他の実行オプション。

- Status – UTF-8 文字列 (有効な値: RUNNING | FINISHED | FAILED | PENDING_EXECUTION | TIMED_OUT | CANCELING | CANCELED | RECEIVED_BY_TASKRUNNER)。

  この実行のステータス。

- ErrorString – UTF–8 文字列。

  実行に関連付けられているエラー文字列。

- StartedOn – タイムスタンプ。

  この実行の開始日時。

- LastModifiedOn – タイムスタンプ。

  タイムスタンプ。このデータ品質ルールの推奨実行が変更された最後の時間ポイント。

- CompletedOn – タイムスタンプ。

  この実行の完了日時。

- ExecutionTime – 数値 (整数)。

  実行でリソースを消費した時間 (秒)。

- RulesetNames – UTF-8 文字列の配列、1 ~ 10 個の文字列。

実行用のルールセット名のリスト。現在、このパラメータは 1 つのルールセット名しかとりません。

- `ResultIds` – UTF-8 文字列の配列、1 ~ 10 個の文字列。

  実行用のデータ品質結果の結果 ID のリスト。

- `AdditionalDataSources` – キーバリューペアのマップ配列。

  各キーは UTF-8 文字列、1 ~ 255 バイト長で、Single-line string pattern に一致します。

  各値は DataSource オブジェクトです。

  評価実行で指定できる他のデータソースへの参照文字列のマップ。

エラー

- `EntityNotFoundException`
- `InvalidInputException`
- `OperationTimeoutException`
- `InternalServiceException`

ListDataQualityRulesetEvaluationRuns action (Python: list_data_quality_ruleset_evaluation_runs)

フィルタリング基準を満たすすべての実行が一覧表示され、ルールセットがデータソースに対して評価されます。

リクエスト

- `Filter` – DataQualityRulesetEvaluationRunFilter オブジェクト。

  フィルタリング基準。

- `NextToken` – UTF–8 文字列。

  結果をオフセットするページ分割されたトークン。

- `MaxResults` – 1 ~ 1000 の数値 (整数)。

  返される結果の最大数。

応答

- Runs – DataQualityRulesetEvaluationRunDescription オブジェクトの配列。

  データ品質ルールセットの実行を表す DataQualityRulesetEvaluationRunDescription オブジェクトのリスト。

- NextToken – UTF–8 文字列。

  ページネーショントークン (さらに結果がある場合)。

エラー

- InvalidInputException
- OperationTimeoutException
- InternalServiceException

StartDataQualityRuleRecommendationRun action (Python: start_data_quality_rule_recommendation_run)

どのルールを記述するのかわからない場合は、ルールの生成に使用される推奨実行を開始します。AWS GlueData Quality はデータを分析し、推奨されるルールセットの候補を提案します。その後、ルールセットをトリアージし、生成されたルールセットを好みに合わせて変更できます。

推奨事項の実行は、実行から 90 日後に自動的に削除されます。

リクエスト

Data Quality ルールのレコメンデーションリクエストのリクエスト。

- DataSource – 必須: DataSource オブジェクト。

  この実行に関連付けられたデータソース (AWS Glue テーブル)。

- Role – 必須: UTF-8 文字列。

  実行の結果を暗号化するために提供される IAM ロール。

- NumberOfWorkers – 数値 (整数)。

  実行に使用される G.1X ワーカーの数。デフォルトは 5 です。

- `Timeout` – 数値 (整数)。1 以上。

  実行のタイムアウト (分)。これは、実行が終了済みになって `TIMEOUT` ステータスに入るまでに、その実行でリソースを消費できる最大時間です。デフォルト値は 2,880 分 (48 時間) です。

- `CreatedRulesetName` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  ルールセットの名前。

- `DataQualitySecurityConfiguration` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  データ品質暗号化オプションを使用して作成されたセキュリティ設定の名前。

- `ClientToken` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  冪等性を保つために使用されるもので、同じリソースの複数のインスタンスを作成または起動しないように、ランダムな ID (UUID など) に設定することをお勧めします。

レスポンス

- `RunId` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  この実行に関連付けられた一意の実行識別子。

エラー

- `InvalidInputException`
- `OperationTimeoutException`
- `InternalServiceException`
- `ConflictException`

CancelDataQualityRuleRecommendationRun action (Python: cancel_data_quality_rule_recommendation_run)

ルールの生成に使用されていた指定の推奨実行をキャンセルします。

リクエスト

- `RunId` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

この実行に関連付けられた一意の実行識別子。

レスポンス

- 応答パラメータはありません。

エラー

- `EntityNotFoundException`
- `InvalidInputException`
- `OperationTimeoutException`
- `InternalServiceException`

GetDataQualityRuleRecommendationRun action (Python: get_data_quality_rule_recommendation_run)

ルールの生成に使用された指定の推奨実行を取得します。

リクエスト

- `RunId` – 必須: UTF-8 文字列。1～255 バイト長。Single-line string pattern に一致。

  この実行に関連付けられた一意の実行識別子。

レスポンス

Data Quality ルールのレコメンデーション実行のレスポンス。

- `RunId` – UTF-8 文字列。1～255 バイト長。Single-line string pattern に一致。

  この実行に関連付けられた一意の実行識別子。
- `DataSource` – DataSource オブジェクト。

  この実行に関連付けられたデータソース (AWS Glue テーブル)。
- `Role` – UTF–8 文字列。

  実行の結果を暗号化するために提供される IAM ロール。

- NumberOfWorkers – 数値 (整数)。

  実行に使用される G.1X ワーカーの数。デフォルトは 5 です。

- Timeout – 数値 (整数)。1 以上。

  実行のタイムアウト (分)。これは、実行が終了済みになって TIMEOUT ステータスに入るまでに、その実行でリソースを消費できる最大時間です。デフォルト値は 2,880 分 (48 時間) です。

- Status – UTF-8 文字列 (有効な値: RUNNING | FINISHED | FAILED | PENDING_EXECUTION | TIMED_OUT | CANCELING | CANCELED | RECEIVED_BY_TASKRUNNER)。

  この実行のステータス。

- ErrorString – UTF–8 文字列。

  実行に関連付けられているエラー文字列。

- StartedOn – タイムスタンプ。

  この実行の開始日時。

- LastModifiedOn – タイムスタンプ。

  タイムスタンプ。このデータ品質ルールの推奨実行が変更された最後の時間ポイント。

- CompletedOn – タイムスタンプ。

  この実行の完了日時。

- ExecutionTime – 数値 (整数)。

  実行でリソースを消費した時間 (秒)。

- RecommendedRuleset – UTF-8 文字列、1 ~ 65536 バイト長。

  開始ルールの推奨実行が完了すると、推奨ルールセット (ルールのセット) が作成されます。このメンバーには、これらのルールがデータ品質定義言語 (DQDL) 形式で格納されています。

- CreatedRulesetName – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  実行によって作成されたルールセットの名前。

- DataQualitySecurityConfiguration – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  データ品質暗号化オプションを使用して作成されたセキュリティ設定の名前。

エラー

- EntityNotFoundException
- InvalidInputException
- OperationTimeoutException
- InternalServiceException

ListDataQualityRuleRecommendationRuns action (Python: list_data_quality_rule_recommendation_runs)

フィルタリング基準を満たす推奨実行を一覧表示します。

リクエスト

- Filter – DataQualityRuleRecommendationRunFilter オブジェクト。

  フィルタリング基準。
- NextToken – UTF–8 文字列。

  結果をオフセットするページ分割されたトークン。
- MaxResults – 1 ~ 1000 の数値 (整数)。

  返される結果の最大数。

応答

- Runs – DataQualityRuleRecommendationRunDescription オブジェクトの配列。

  DataQualityRuleRecommendationRunDescription オブジェクトのリスト。
- NextToken – UTF–8 文字列。

  ページネーショントークン (さらに結果がある場合)。

エラー

- InvalidInputException
- OperationTimeoutException

- InternalServiceException

GetDataQualityResult action (Python: get_data_quality_result)

データ品質ルール評価の結果を取得します。

リクエスト

- ResultId – 必須: UTF-8 文字列。1～255 バイト長。Single-line string pattern に一致。

  データ品質結果に対する一意の結果 ID。

レスポンス

データ品質結果のレスポンス。

- ResultId – UTF-8 文字列。1～255 バイト長。Single-line string pattern に一致。

  データ品質結果に対する一意の結果 ID。
- ProfileId – UTF-8 文字列。1～255 バイト長。Single-line string pattern に一致。

  データ品質結果のプロファイル ID。
- Score – 数値 (double)。1.0 以下。

  集計されたデータ品質のスコア。ルールの合計数に対する合格ルールの比率を表します。
- DataSource – DataSource オブジェクト。

  データ品質結果に関連付けられたテーブル (存在する場合)。
- RulesetName – UTF-8 文字列。1～255 バイト長。Single-line string pattern に一致。

  データ品質結果に関連付けられたルールセットの名前。
- EvaluationContext – UTF–8 文字列。

  AWS Glue Studio のジョブのコンテキストでは、通常、キャンバス内の各ノードには何らかの名前が割り当てられ、データ品質ノードには名前が付けられます。複数のノードの場合、evaluationContext でノードを区別できます。
- StartedOn – タイムスタンプ。

  このデータ品質結果の実行が開始された日時。

- CompletedOn – タイムスタンプ。

  このデータ品質結果の実行が完了した日時。

- JobName – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  データ品質結果に関連付けられたジョブ名 (存在する場合)。

- JobRunId – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  データ品質結果に関連付けられたジョブ実行 ID (存在する場合)。

- RulesetEvaluationRunId – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  ルールセット評価に関連付けられた一意の実行 ID。

- RuleResults – DataQualityRuleResult オブジェクトの配列。構造 2000 個以下。

  各ルールの結果を表す DataQualityRuleResult オブジェクトのリスト。

- AnalyzerResults – DataQualityAnalyzerResult オブジェクトの配列。構造 2000 個以下。

  各アナライザーの結果を表す DataQualityAnalyzerResult オブジェクトのリスト。

- Observations – DataQualityObservation オブジェクトの配列。構造 50 個以下。

  ルールとアナライザーを評価した後に生成された観測値を表す DataQualityObservation オブジェクトのリスト。

- AggregatedMetrics – DataQualityAggregatedMetrics オブジェクト。

  行レベルの結果に基づく合格/不合格統計を含む、処理された行とルールの合計数を示す DataQualityAggregatedMetrics オブジェクトの概要。

エラー

- InvalidInputException
- OperationTimeoutException
- InternalServiceException
- EntityNotFoundException

BatchGetDataQualityResult action (Python: batch_get_data_quality_result)

指定した結果 ID のデータ品質結果のリストを取得します。

リクエスト

- `ResultIds` – 必須: UTF-8 文字列の配列、1 ~ 100 個の文字列。

  データ品質結果に対する一意の結果 ID のリスト。

レスポンス

- `Results` – 必須: DataQualityResult オブジェクトの配列。

  データ品質結果を表す `DataQualityResult` オブジェクトのリスト。
- `ResultsNotFound` – UTF-8 文字列の配列、1 ~ 100 個の文字列。

  結果が見つからなかった結果 ID のリスト。

エラー

- `InvalidInputException`
- `OperationTimeoutException`
- `InternalServiceException`

ListDataQualityResults action (Python: list_data_quality_results)

アカウントのすべてのデータ品質実行結果を返します。

リクエスト

- `Filter` – DataQualityResultFilterCriteria オブジェクト。

  フィルタリング基準。
- `NextToken` – UTF–8 文字列。

  結果をオフセットするページ分割されたトークン。
- `MaxResults` – 1 ~ 1000 の数値 (整数)。

返される結果の最大数。

レスポンス

- `Results` – 必須: DataQualityResultDescription オブジェクトの配列。

  `DataQualityResultDescription` オブジェクトのリスト。
- `NextToken` – UTF–8 文字列。

  ページネーショントークン (さらに結果がある場合)。

エラー

- `InvalidInputException`
- `OperationTimeoutException`
- `InternalServiceException`

CreateDataQualityRuleset action (Python: create_data_quality_ruleset)

指定した AWS Glue テーブルに DQDL ルールが適用されたデータ品質ルールセットを作成します。

データ品質定義言語 (DQDL) を使用して、ルールセットを作成します。詳細については、「AWS Glue デベロッパーガイド」を参照してください。

リクエスト

データ品質評価ルールセットを作成するリクエスト。

- `Name` – 必須: UTF-8 文字列。1～255 バイト長。Single-line string pattern に一致。

  データ品質ルールセットに対する一意の名前。
- `Description` – 説明文字列。2,048 バイト長以下。URI address multi-line string pattern に一致。

  データ品質ルールセットの説明。
- `Ruleset` – 必須: UTF-8 文字列、1～65536 バイト長。

  データ品質定義言語 (DQDL) ルールセット。詳細については、「AWS Glue デベロッパーガイド」を参照してください。

- Tags – キーと値のペアのマップ配列。50 ペア以下。

  各キーは UTF-8 文字列で、1 ~ 128 バイト長です。

  各値は UTF-8 文字列で、256 バイト長以下です。

  データ品質ルールセットに適用されるタグのリスト。
- TargetTable – DataQualityTargetTable オブジェクト。

  データ品質ルールセットに関連付けられたターゲットテーブル。
- RecommendationRunId – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  推奨実行に対する一意の実行 ID。
- DataQualitySecurityConfiguration – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  データ品質暗号化オプションを使用して作成されたセキュリティ設定の名前。
- ClientToken – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  冪等性を保つために使用されるもので、同じリソースの複数のインスタンスを作成または起動しないように、ランダムな ID (UUID など) に設定することをお勧めします。

レスポンス

- Name – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  データ品質ルールセットに対する一意の名前。

エラー

- InvalidInputException
- AlreadyExistsException
- OperationTimeoutException
- InternalServiceException
- ResourceNumberLimitExceededException

DeleteDataQualityRuleset action (Python: delete_data_quality_ruleset)

データ品質ルールセットを削除します。

リクエスト

- Name – 必須: UTF-8 文字列。1～255 バイト長。Single-line string pattern に一致。

  データ品質ルールセットの名前。

レスポンス

- 応答パラメータはありません。

エラー

- EntityNotFoundException
- InvalidInputException
- OperationTimeoutException
- InternalServiceException

GetDataQualityRuleset action (Python: get_data_quality_ruleset)

既存のルールセットを識別子または名前で返します。

リクエスト

- Name – 必須: UTF-8 文字列。1～255 バイト長。Single-line string pattern に一致。

  ルールセットの名前。

レスポンス

データ品質ルールセットレスポンスを返します。

- Name – UTF-8 文字列。1～255 バイト長。Single-line string pattern に一致。

  ルールセットの名前。
- Description – 説明文字列。2,048 バイト長以下。URI address multi-line string pattern に一致。

ルールセットの説明。

- `Ruleset` – UTF-8 文字列、1 ~ 65536 バイト長。

  データ品質定義言語 (DQDL) ルールセット。詳細については、「AWS Glue デベロッパーガイド」を参照してください。

- `TargetTable` – DataQualityTargetTable オブジェクト。

  ターゲットテーブルの名前およびデータベース名。

- `CreatedOn` – タイムスタンプ。

  タイムスタンプ。このデータ品質ルールセットが作成された日時。

- `LastModifiedOn` – タイムスタンプ。

  タイムスタンプ。このデータ品質ルールセットが変更された最後の時間ポイント。

- `RecommendationRunId` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  推奨実行からルールセットが作成されると、この 2 つをリンクさせるためにこの実行 ID が生成されます。

- `DataQualitySecurityConfiguration` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  データ品質暗号化オプションを使用して作成されたセキュリティ設定の名前。

エラー

- `EntityNotFoundException`
- `InvalidInputException`
- `OperationTimeoutException`
- `InternalServiceException`

ListDataQualityRulesets action (Python: list_data_quality_rulesets)

指定した AWS Glue テーブルのリスト用のルールセットのページ分割されたリストを返します。

リクエスト

- `NextToken` – UTF-8 文字列。

結果をオフセットするページ分割されたトークン。

- `MaxResults` – 1 ~ 1000 の数値 (整数)。

  返される結果の最大数。

- `Filter` – DataQualityRulesetFilterCriteria オブジェクト。

  フィルタリング基準。

- `Tags` – キーと値のペアのマップ配列。50 ペア以下。

  各キーは UTF-8 文字列で、1 ~ 128 バイト長です。

  各値は UTF-8 文字列で、256 バイト長以下です。

  キーと値のペアタグのリスト。

応答

- `Rulesets` – DataQualityRulesetListDetails オブジェクトの配列。

  指定した AWS Glue テーブルのリスト用のルールセットのページ分割されたリスト。

- `NextToken` – UTF–8 文字列。

  ページネーショントークン (さらに結果がある場合)。

エラー

- `EntityNotFoundException`
- `InvalidInputException`
- `OperationTimeoutException`
- `InternalServiceException`

UpdateDataQualityRuleset アクション (Python: update_data_quality_ruleset)

指定したデータ品質ルールセットを更新します。

リクエスト

- Name – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  データ品質ルールセットの名前。

- Description – 説明文字列。2,048 バイト長以下。URI address multi-line string pattern に一致。

  ルールセットの説明。

- Ruleset – UTF-8 文字列、1 ~ 65536 バイト長。

  データ品質定義言語 (DQDL) ルールセット。詳細については、「AWS Glue デベロッパーガイド」を参照してください。

レスポンス

- Name – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  データ品質ルールセットの名前。

- Description – 説明文字列。2,048 バイト長以下。URI address multi-line string pattern に一致。

  ルールセットの説明。

- Ruleset – UTF-8 文字列、1 ~ 65536 バイト長。

  データ品質定義言語 (DQDL) ルールセット。詳細については、「AWS Glue デベロッパーガイド」を参照してください。

エラー

- EntityNotFoundException
- AlreadyExistsException
- IdempotentParameterMismatchException
- InvalidInputException
- OperationTimeoutException
- InternalServiceException
- ResourceNumberLimitExceededException

ListDataQualityRulesets action (Python: list_data_quality_statistics)

データ品質統計のリストを取得します。

リクエスト

- `StatisticId` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  統計 ID。
- `ProfileId` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  プロファイル ID。
- `TimestampFilter` – TimestampFilter オブジェクト。

  タイムスタンプフィルター。
- `MaxResults` – 1 ~ 1000 の数値 (整数)。

  このリクエストで返す結果の最大数。
- `NextToken` – UTF–8 文字列。

  結果の次ページをリクエストするページネーショントークン。

応答

- `Statistics` – StatisticSummary オブジェクトの配列。

  `StatisticSummaryList`。
- `NextToken` – UTF–8 文字列。

  結果の次ページをリクエストするページネーショントークン。

エラー

- `EntityNotFoundException`
- `InvalidInputException`
- `InternalServiceException`

TimestampFilter 構造

タイムスタンプフィルター。

フィールド

- `RecordedBefore` – タイムスタンプ。

  統計を結果に含める前のタイムスタンプ。
- `RecordedAfter` – タイムスタンプ。

  統計を結果に含める必要があるタイムスタンプ。

CreateDataQualityRulesetRequest structure

データ品質評価ルールセットを作成するリクエスト。

フィールド

- `Name` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  データ品質ルールセットに対する一意の名前。
- `Description` – 説明文字列。2,048 バイト長以下。URI address multi-line string pattern に一致。

  データ品質ルールセットの説明。
- `Ruleset` – 必須: UTF-8 文字列、1 ~ 65536 バイト長。

  データ品質定義言語 (DQDL) ルールセット。詳細については、「AWS Glue デベロッパーガイド」を参照してください。
- `Tags` – キーと値のペアのマップ配列。50 ペア以下。

  各キーは UTF-8 文字列で、1 ~ 128 バイト長です。

  各値は UTF-8 文字列で、256 バイト長以下です。

  データ品質ルールセットに適用されるタグのリスト。
- `TargetTable` – DataQualityTargetTable オブジェクト。

  データ品質ルールセットに関連付けられたターゲットテーブル。

- `RecommendationRunId` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  推奨実行に対する一意の実行 ID。

- `DataQualitySecurityConfiguration` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  データ品質暗号化オプションを使用して作成されたセキュリティ設定の名前。

- `ClientToken` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  べき等性を保つために使用されるもので、同じリソースの複数のインスタンスを作成または起動しないように、ランダムな ID (UUID など) に設定することをお勧めします。

GetDataQualityRulesetResponse structure

データ品質ルールセットレスポンスを返します。

フィールド

- `Name` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  ルールセットの名前。

- `Description` – 説明文字列。2,048 バイト長以下。URI address multi-line string pattern に一致。

  ルールセットの説明。

- `Ruleset` – UTF-8 文字列、1 ~ 65536 バイト長。

  データ品質定義言語 (DQDL) ルールセット。詳細については、「AWS Glue デベロッパーガイド」を参照してください。

- `TargetTable` – DataQualityTargetTable オブジェクト。

  ターゲットテーブルの名前およびデータベース名。

- `CreatedOn` – タイムスタンプ。

  タイムスタンプ。このデータ品質ルールセットが作成された日時。

- `LastModifiedOn` – タイムスタンプ。

  タイムスタンプ。このデータ品質ルールセットが変更された最後の時間ポイント。

- `RecommendationRunId` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

推奨実行からルールセットが作成されると、この 2 つをリンクさせるためにこの実行 ID が生成されます。

- `DataQualitySecurityConfiguration` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  データ品質暗号化オプションを使用して作成されたセキュリティ設定の名前。

GetDataQualityResultResponse structure

データ品質結果のレスポンス。

フィールド

- `ResultId` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  データ品質結果に対する一意の結果 ID。
- `ProfileId` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  データ品質結果のプロファイル ID。
- `Score` – 数値 (double)。1.0 以下。

  集計されたデータ品質のスコア。ルールの合計数に対する合格ルールの比率を表します。
- `DataSource` – DataSource オブジェクト。

  データ品質結果に関連付けられたテーブル (存在する場合)。
- `RulesetName` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  データ品質結果に関連付けられたルールセットの名前。
- `EvaluationContext` – UTF–8 文字列。

  AWS Glue Studio のジョブのコンテキストでは、通常、キャンバス内の各ノードには何らかの名前が割り当てられ、データ品質ノードには名前が付けられます。複数のノードの場合、`evaluationContext` でノードを区別できます。
- `StartedOn` – タイムスタンプ。

  このデータ品質結果の実行が開始された日時。
- `CompletedOn` – タイムスタンプ。

このデータ品質結果の実行が完了した日時。

- JobName – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  データ品質結果に関連付けられたジョブ名 (存在する場合)。

- JobRunId – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  データ品質結果に関連付けられたジョブ実行 ID (存在する場合)。

- RulesetEvaluationRunId – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  ルールセット評価に関連付けられた一意の実行 ID。

- RuleResults – DataQualityRuleResult オブジェクトの配列。構造 2000 個以下。

  各ルールの結果を表す DataQualityRuleResult オブジェクトのリスト。

- AnalyzerResults – DataQualityAnalyzerResult オブジェクトの配列。構造 2000 個以下。

  各アナライザーの結果を表す DataQualityAnalyzerResult オブジェクトのリスト。

- Observations – DataQualityObservation オブジェクトの配列。構造 50 個以下。

  ルールとアナライザーを評価した後に生成された観測値を表す DataQualityObservation オブジェクトのリスト。

- AggregatedMetrics – DataQualityAggregatedMetrics オブジェクト。

  行レベルの結果に基づく合格/不合格統計を含む、処理された行とルールの合計数を示す DataQualityAggregatedMetrics オブジェクトの概要。

StartDataQualityRuleRecommendationRunRequest structure

Data Quality ルールのレコメンデーションリクエストのリクエスト。

フィールド

- DataSource – 必須: DataSource オブジェクト。

  この実行に関連付けられたデータソース (AWS Glue テーブル)。

- Role – 必須: UTF-8 文字列。

  実行の結果を暗号化するために提供される IAM ロール。

- `NumberOfWorkers` – 数値 (整数)。

  実行に使用される `G.1X` ワーカーの数。デフォルトは 5 です。

- `Timeout` – 数値 (整数)。1 以上。

  実行のタイムアウト (分)。これは、実行が終了済みになって `TIMEOUT` ステータスに入るまでに、その実行でリソースを消費できる最大時間です。デフォルト値は 2,880 分 (48 時間) です。

- `CreatedRulesetName` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  ルールセットの名前。

- `DataQualitySecurityConfiguration` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  データ品質暗号化オプションを使用して作成されたセキュリティ設定の名前。

- `ClientToken` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  べき等性を保つために使用されるもので、同じリソースの複数のインスタンスを作成または起動しないように、ランダムな ID (UUID など) に設定することをお勧めします。

GetDataQualityRuleRecommendationRunResponse structure

Data Quality ルールのレコメンデーション実行のレスポンス。

フィールド

- `RunId` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  この実行に関連付けられた一意の実行識別子。

- `DataSource` – DataSource オブジェクト。

  この実行に関連付けられたデータソース (AWS Glue テーブル)。

- `Role` – UTF–8 文字列。

  実行の結果を暗号化するために提供される IAM ロール。

- `NumberOfWorkers` – 数値 (整数)。

  実行に使用される `G.1X` ワーカーの数。デフォルトは 5 です。

- `Timeout` – 数値 (整数)。1 以上。

実行のタイムアウト (分)。これは、実行が終了済みになって TIMEOUT ステータスに入るまでに、その実行でリソースを消費できる最大時間です。デフォルト値は 2,880 分 (48 時間) です。

- Status – UTF-8 文字列 (有効な値: RUNNING | FINISHED | FAILED | PENDING_EXECUTION | TIMED_OUT | CANCELING | CANCELED | RECEIVED_BY_TASKRUNNER)。

  この実行のステータス。

- ErrorString – UTF–8 文字列。

  実行に関連付けられているエラー文字列。

- StartedOn – タイムスタンプ。

  この実行の開始日時。

- LastModifiedOn – タイムスタンプ。

  タイムスタンプ。このデータ品質ルールの推奨実行が変更された最後の時間ポイント。

- CompletedOn – タイムスタンプ。

  この実行の完了日時。

- ExecutionTime – 数値 (整数)。

  実行でリソースを消費した時間 (秒)。

- RecommendedRuleset – UTF-8 文字列、1 ~ 65536 バイト長。

  開始ルールの推奨実行が完了すると、推奨ルールセット (ルールのセット) が作成されます。このメンバーには、これらのルールがデータ品質定義言語 (DQDL) 形式で格納されています。

- CreatedRulesetName – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  実行によって作成されたルールセットの名前。

- DataQualitySecurityConfiguration – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  データ品質暗号化オプションを使用して作成されたセキュリティ設定の名前。

BatchPutDataQualityStatisticAnnotation action (Python: batch_put_data_quality_statistic_annotation)

特定のデータ品質統計のデータポイントを経時的に注釈付けします。

リクエスト

- `InclusionAnnotations` – 必須: DatapointInclusionAnnotation オブジェクトの配列。

  `DatapointInclusionAnnotation` のリスト。
- `ClientToken` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  クライアントトークン。

応答

- `FailedInclusionAnnotations` – AnnotationError オブジェクトの配列。

  `AnnotationError` のリスト。

エラー

- `EntityNotFoundException`
- `InvalidInputException`
- `InternalServiceException`
- `ResourceNumberLimitExceededException`

GetDataQualityModel アクション (Python: get_data_quality_model)

モデルのトレーニングステータスを詳細情報 (CompletedOn, StartedOn, FailureReason) とともに取得します。

リクエスト

- `StatisticId` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  統計 ID。
- `ProfileId` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

プロファイル ID。

レスポンス

- `Status` – UTF-8 文字列 (有効な値: `RUNNING` | `SUCCEEDED` | `FAILED`)。

  データ品質モデルのトレーニングステータス。
- `StartedOn` – タイムスタンプ。

  データ品質モデルトレーニングが開始されたタイムスタンプ。
- `CompletedOn` – タイムスタンプ。

  データ品質モデルトレーニングが完了したときのタイムスタンプ。
- `FailureReason` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  トレーニング失敗の理由。

エラー

- `EntityNotFoundException`
- `InvalidInputException`
- `OperationTimeoutException`
- `InternalServiceException`

GetDataQualityModelResult action (Python: get_data_quality_model_result)

特定のプロファイル ID の統計の予測を取得します。

リクエスト

- `StatisticId` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  統計 ID。
- `ProfileId` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  プロファイル ID。

レスポンス

- CompletedOn – タイムスタンプ。

  データ品質モデルトレーニングが完了したときのタイムスタンプ。
- Model – StatisticModelResult オブジェクトの配列。

  StatisticModelResult のリスト。

エラー

- EntityNotFoundException
- InvalidInputException
- OperationTimeoutException
- InternalServiceException

ListDataQualityStatisticAnnotations action (Python: list_data_quality_statistic_annotations)

データ品質統計の注釈を取得します。

リクエスト

- StatisticId – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  統計 ID。
- ProfileId – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  プロファイル ID。
- TimestampFilter – TimestampFilter オブジェクト。

  タイムスタンプフィルター。
- MaxResults – 1 ~ 1000 の数値 (整数)。

  このリクエストで返す結果の最大数。
- NextToken – UTF–8 文字列。

  次の結果一式を取得するためのページネーショントークン。

応答

- Annotations – StatisticAnnotation オブジェクトの配列。

  統計に適用された StatisticAnnotation のリスト

- NextToken – UTF–8 文字列。

  次の結果一式を取得するためのページネーショントークン。

エラー

- InvalidInputException
- InternalServiceException

PutDataQualityProfileAnnotation action (Python: put_data_quality_profile_annotation)

プロファイルのすべてのデータポイントに注釈を付けます。

リクエスト

- ProfileId – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  注釈を付けるデータ品質モニタリングプロファイルの ID。

- InclusionAnnotation – 必須: UTF-8 文字列 (有効な値: INCLUDE | EXCLUDE)。

  プロファイルに適用する包含注釈値。

レスポンス

- 応答パラメータはありません。

エラー

- EntityNotFoundException
- InvalidInputException
- InternalServiceException

機密データの検出 API

機密データ検出 API では、構造化データの列と行全体にある機密データを検出するために使用される API について説明します。

データ型

- CustomentityType 構造

CustomentityType 構造

構造化データの列と行全体にある機密データを検出するためのカスタムパターンを表すオブジェクトです。

フィールド

- Name – 必須: UTF-8 文字列、1～255 バイト長、Single-line string pattern に一致。

  後で、取得したり削除したりすることができるカスタムパターンの名前です。この名前は AWS アカウントごとに一意である必要があります。
- RegexString – 必須: UTF-8 文字列、1～255 バイト長、Single-line string pattern に一致。

  カスタムパターン内にある機密データの検出に使用される正規表現の文字列。
- ContextWords – UTF-8 文字列の配列、1～20 個の文字列。

  コンテキスト単語のリスト。正規表現の範囲内でこれらのコンテキスト単語が見つからない場合、データは機密データとして検出されません。

  コンテキスト単語が渡されない場合は、正規表現のみがチェックされます。

操作

- CreateCustomentType アクション (Python: create_custom_entity_type)
- DeleteCustomEntityType アクション (Python: delete_custom_entity_type)
- GetCustomEntityType アクション (Python: get_custom_entity_type)
- BatchGetCustomEntityTypes アクション (Python: batch_get_custom_entity_types)
- ListCustomentTypes アクション (Python: list_custom_entity_types)

CreateCustomentType アクション (Python: create_custom_entity_type)

構造化データの列と行全体にある機密データを検出するために使用されるカスタムパターンを作成します。

作成するカスタムパターンごとに、正規表現とオプションのコンテキスト単語リストを作成し、指定します。コンテキスト単語が渡されない場合は、正規表現のみがチェックされます。

リクエスト

- Name – 必須: UTF-8 文字列、1～255 バイト長、「Single-line string pattern」に一致。

  後で、取得したり削除したりすることができるカスタムパターンの名前です。この名前は AWS アカウントごとに一意である必要があります。
- RegexString – 必須: UTF-8 文字列、1～255 バイト長、Single-line string pattern に一致。

  カスタムパターン内にある機密データの検出に使用される正規表現の文字列。
- ContextWords – UTF-8 文字列の配列、1～20 個の文字列。

  コンテキスト単語のリスト。正規表現の範囲内でこれらのコンテキスト単語が見つからない場合、データは機密データとして検出されません。

  コンテキスト単語が渡されない場合は、正規表現のみがチェックされます。
- Tags – キーと値のペアのマップ配列。50 ペア以下。

  各キーは UTF-8 文字列で、1～128 バイト長です。

  各値は UTF-8 文字列で、256 バイト長以下です。

  カスタムエンティティタイプに適用されるタグのリスト。

レスポンス

- Name – UTF-8 文字列、1～255 バイト長、「Single-line string pattern」に一致。

  作成したカスタムパターンの名前。

エラー

- AccessDeniedException

- AlreadyExistsException
- IdempotentParameterMismatchException
- InternalServiceException
- InvalidInputException
- OperationTimeoutException
- ResourceNumberLimitExceededException

DeleteCustomEntityType アクション (Python: delete_custom_entity_type)

カスタムパターン名を指定して、カスタムパターンを削除します。

リクエスト

- Name – 必須: UTF-8 文字列、1 ~ 255 バイト長、「Single-line string pattern」に一致。

  削除するカスタムパターンの名前です。

レスポンス

- Name – UTF-8 文字列、1 ~ 255 バイト長、「Single-line string pattern」に一致。

  削除したカスタムパターンの名前です。

エラー

- EntityNotFoundException
- AccessDeniedException
- InternalServiceException
- InvalidInputException
- OperationTimeoutException

GetCustomEntityType アクション (Python: get_custom_entity_type)

カスタムパターン名を指定して、カスタムパターンの詳細を取得します。

リクエスト

- Name – 必須: UTF-8 文字列、1～255 バイト長、「Single-line string pattern」に一致。

  取得するカスタムパターンの名前。

レスポンス

- Name – UTF-8 文字列、1～255 バイト長、「Single-line string pattern」に一致。

  取得したカスタムパターンの名前。
- RegexString – UTF-8 文字列、1～255 バイト長、Single-line string pattern に一致。

  カスタムパターン内にある機密データの検出に使用される正規表現の文字列。
- ContextWords – UTF-8 文字列の配列、1～20 個の文字列。

  カスタムパターンを作成した時に指定している場合は、コンテキスト単語のリストです。正規表現の範囲内でこれらのコンテキスト単語が見つからない場合、データは機密データとして検出されません。

エラー

- EntityNotFoundException
- AccessDeniedException
- InternalServiceException
- InvalidInputException
- OperationTimeoutException

BatchGetCustomEntityTypes アクション (Python: batch_get_custom_entity_types)

名前リストで指定されたカスタムパターンの詳細を取得します。

リクエスト

- Names – 必須: UTF-8 文字列の配列、1～50 個の文字列。

取得するカスタムパターンの名前リストです。

応答

- CustomEntityTypes – CustomEntityType オブジェクトの配列。

  作成したカスタムパターンを表す CustomEntityType オブジェクトのリストです。
- CustomEntityTypesNotFound – UTF-8 文字列の配列、1 ~ 50 個の文字列。

  検出できなかったカスタムパターンの名前リストです。

エラー

- InvalidInputException
- InternalServiceException
- OperationTimeoutException

ListCustomentTypes アクション (Python: list_custom_entity_types)

作成したすべてのカスタムパターンをリスト表示します。

リクエスト

- NextToken – UTF-8 文字列。

  結果をオフセットするページ分割されたトークン。
- MaxResults – 1 ~ 1000 の数値 (整数)。

  返される結果の最大数。
- Tags – キーと値のペアのマップ配列。50 ペア以下。

  各キーは UTF-8 文字列で、1 ~ 128 バイト長です。

  各値は UTF-8 文字列で、256 バイト長以下です。

  キーと値のペアタグのリスト。

応答

- CustomEntityTypes – CustomEntityType オブジェクトの配列。

  カスタムパターンを表す CustomEntityType オブジェクトのリストです。
- NextToken – UTF-8 文字列。

  ページ分割トークン (さらに結果がある場合)。

エラー

- InvalidInputException
- OperationTimeoutException
- InternalServiceException

AWS Glue の API のタグ付け

データ型

- Tag 構造

Tag 構造

Tag オブジェクトは、AWS リソースに割り当てることができるラベルを表します。タグはそれぞれ、1 つのキーとオプションの 1 つの値で設定されており、どちらもお客様側が定義します。

リソース AWS Glue へのアクセス制御や、AWS タグ AWS Glue および指定 AWS Glue リソース ARN の詳細については、開発者ガイドを参照してください。

フィールド

- key – UTF-8 文字列、1 ~ 128 バイト長。

  タグキー。オブジェクトにタグを作成するときにキーが必要です。キーでは大文字と小文字が区別され、プレフィックス aws を含めることはできません。
- value - UTF-8 文字列、256 バイト長以下。

タグ値。オブジェクトにタグを作成するときの値はオプションです。値では大文字と小文字が区別され、プレフィックス aws を含めることはできません。

操作

- TagResource アクション (Python: tag_resource)
- UntagResource アクション (Python: untag_resource)
- GetTags アクション (Python: get_tags)

TagResource アクション (Python: tag_resource)

リソースにタグを追加します。タグとは、AWS のリソースに付けることができるラベルです。AWS Glue では、特定のリソースにのみタグを付けることができます。どのリソースにタグを付けることができるかについては、「AWS Tags in AWS Glue」を参照してください。

タグ関連の API を呼び出すためのタグ付けのアクセス許可に加えて、接続でタグ付け API を呼び出すための `glue:GetConnection` アクセス許可と、データベースでタグ付け API を呼び出すための `glue:GetDatabase` アクセス許可も必要です。

リクエスト

- `ResourceArn` – 必須: UTF-8 文字列。1 ~ 10,240 バイト長。Custom string pattern #49 に一致。

  タグを追加する AWS Glue リソースの ARN。AWS Glue リソースの ARN の詳細については、「AWS Glue ARN string pattern」を参照してください。
- `TagsToAdd` – 必須: キーと値のペアのマップ配列。50 ペア以下。

  各キーは UTF-8 文字列で、1 ~ 128 バイト長です。

  各値は UTF-8 文字列で、256 バイト長以下です。

  このリソースに追加するタグ。

レスポンス

- 応答パラメータはありません。

エラー

- `ResourceNotFoundException`

UntagResource アクション (Python: untag_resource)

統合リソースから指定したタグを削除します。

リクエスト

- `ResourceArn` – 必須: UTF-8 文字列。1 ~ 10,240 バイト長。Custom string pattern #49 に一致。

  統合リソースの Amazon リソースネーム (ARN)。
- `TagsToRemove` – 必須: UTF-8 文字列の配列。文字列 50 個以下。

  リソースから削除するメタデータタグのリスト。

レスポンス

- 応答パラメータはありません。

エラー

- `ResourceNotFoundException`

GetTags アクション (Python: get_tags)

リソースに関連付けられているタグのリストを取得します。

リクエスト

- `ResourceArn` – 必須: UTF-8 文字列。1 ~ 10,240 バイト長。Custom string pattern #49 に一致。

  タグを取得する対象のリソースの Amazon リソースネーム (ARN)。

レスポンス

- `Tags` – キーと値のペアのマップ配列。50 ペア以下。

各キーは UTF-8 文字列で、1 ~ 128 バイト長です。

各値は UTF-8 文字列で、256 バイト長以下です。

リクエストされたタグ。

エラー

- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`
- `EntityNotFoundException`

一般的なデータ型

一般的なデータ型では、AWS Glue でのその他の一般的なデータ型について説明します。

Tag 構造

Tag オブジェクトは、AWS リソースに割り当てることができるラベルを表します。タグはそれぞれ、1 つのキーとオプションの 1 つの値で設定されており、どちらもお客様側が定義します。

リソース AWS Glue へのアクセス制御や、AWS タグ AWS Glue および指定 AWS Glue リソース ARN の詳細については、開発者ガイドを参照してください。

フィールド

- key – UTF-8 文字列、1 ~ 128 バイト長。

  タグキー。オブジェクトにタグを作成するときにキーが必要です。キーでは大文字と小文字が区別され、プレフィックス aws を含めることはできません。

- value - UTF-8 文字列、256 バイト長以下。

  タグ値。オブジェクトにタグを作成するときの値はオプションです。値では大文字と小文字が区別され、プレフィックス aws を含めることはできません。

DecimalNumber 構造

10 進数形式の数値が含まれます。

フィールド

- `UnscaledValue` – 必須: Blob。

  スケールされていない数値。
- `Scale` – 必須: 数値 (integer)。

  スケールされていない値のどの位置に小数点を置くかを決定するスケール。

ErrorDetail 構造

エラーに関する詳細が含まれています。

フィールド

- `ErrorCode` – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  このエラーに関連付けられたコード。
- `ErrorMessage` – 説明文字列。2,048 バイト長以下。URI address multi-line string pattern に一致。

  エラーを説明するメッセージ。

PropertyPredicate 構造

プロパティの述語を定義します。

フィールド

- `Key` – 値の文字列の長さ。1 ~ 1,024 バイト長。

  プロパティのキー。
- `Value` – 値の文字列の長さ。1 ~ 1,024 バイト長。

  プロパティの値。

- `Comparator` – UTF-8 文字列 (有効な値: `EQUALS` | `GREATER_THAN` | `LESS_THAN` | `GREATER_THAN_EQUALS` | `LESS_THAN_EQUALS`)。

  このプロパティを他のプロパティと比較するために使用されたコンパレータ。

ResourceUri 構造

関数リソースの URI。

フィールド

- `ResourceType` – UTF-8 文字列 (有効な値: `JAR` | `FILE` | `ARCHIVE`)。

  リソースのタイプ。

- `Uri` – Uniform resource identifier (uri)。1 ~ 1024 バイト長。URI address multi-line string pattern に一致。

  リソースにアクセスするための URI。

ColumnStatistics 構造

テーブルまたはパーティションに対して生成された列レベルの統計を表します。

フィールド

- `ColumnName` – 必須: UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  統計が属する列の名前。

- `ColumnType` – 必須: 型名。20,000 バイト長以下。Single-line string pattern に一致。

  列のデータ型。

- `AnalyzedTime` – 必須: タイムスタンプ。

  列統計が生成された時点を示すタイムスタンプ。

- `StatisticsData` – 必須: ColumnStatisticsData オブジェクト。

  統計データ値を含む `ColumnStatisticData` オブジェクト。

ColumnStatisticsError 構造

失敗した ColumnStatistics オブジェクトと失敗の理由をカプセル化します。

フィールド

- ColumnStatistics – ColumnStatistics オブジェクト。

  列の ColumnStatistics。
- Error – ErrorDetail オブジェクト。

  オペレーションが失敗した理由を示すエラーメッセージ。

ColumnError 構造

エラーがある列の名前とエラーの理由について要約します。

フィールド

- ColumnName – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  エラーがある列の名前。
- Error – ErrorDetail オブジェクト。

  オペレーションが失敗した理由を示すエラーメッセージ。

ColumnStatisticsData 構造

個々のタイプの列統計データが含まれます。Type 属性で 1 つのデータオブジェクトのみを設定し、指示します。

フィールド

- Type – 必須: UTF-8 文字列 (有効な値: BOOLEAN | DATE | DECIMAL | DOUBLE | LONG | STRING | BINARY)。

  列の統計データの型。
- BooleanColumnStatisticsData – BooleanColumnStatisticsData オブジェクト。

  Boolean の列統計データ。

- `DateColumnStatisticsData` – DateColumnStatisticsData オブジェクト。

  Date の列統計データ。

- `DecimalColumnStatisticsData` – DecimalColumnStatisticsData オブジェクト。

  Decimal の列統計データ。内部の UnscaledValue は Base64 でエンコードされたバイナリオブジェクトであり、10 進数のスケーリングされていない値である 2 の補数表現がビッグエンディアンで格納されています。

- `DoubleColumnStatisticsData` – DoubleColumnStatisticsData オブジェクト。

  Double の列統計データ。

- `LongColumnStatisticsData` – LongColumnStatisticsData オブジェクト。

  Long の列統計データ。

- `StringColumnStatisticsData` – StringColumnStatisticsData オブジェクト。

  文字列の列統計データ。

- `BinaryColumnStatisticsData` – BinaryColumnStatisticsData オブジェクト。

  Binary の列統計データ。

BooleanColumnStatisticsData 構造

Boolean データ列でサポートされる列統計を定義します。

フィールド

- `NumberOfTrues` – 必須: 数値 (long)。None 以下。

  列内の true の値の数。

- `NumberOfFalses` – 必須: 数値 (long)。None 以下。

  列内の false の値の数。

- `NumberOfNulls` – 必須: 数値 (long)。None 以下。

  列内の null 値の数。

DateColumnStatisticsData 構造

タイムスタンプデータ列でサポートされている列統計を定義します。

フィールド

- `MinimumValue` – タイムスタンプ。

  列内の最小値。

- `MaximumValue` – タイムスタンプ。

  列内の最大値。

- `NumberOfNulls` – 必須: 数値 (long)。None 以下。

  列内の null 値の数。

- `NumberOfDistinctValues` – 必須: 数値 (long)。None 以下。

  列内の異なる値の数です。

DecimalColumnStatisticsData 構造

固定小数点数データ列でサポートされている列統計を定義します。

フィールド

- `MinimumValue` – DecimalNumber オブジェクト。

  列内の最小値。

- `MaximumValue` – DecimalNumber オブジェクト。

  列内の最大値。

- `NumberOfNulls` – 必須: 数値 (long)。None 以下。

  列内の null 値の数。

- `NumberOfDistinctValues` – 必須: 数値 (long)。None 以下。

  列内の異なる値の数です。

DoubleColumnStatisticsData 構造

浮動小数点数データ列でサポートされている列統計を定義します。

フィールド

- `MinimumValue` – 数値 (double)。

  列内の最小値。

- `MaximumValue` – 数値 (double)。

  列内の最大値。

- `NumberOfNulls` – 必須: 数値 (long)。None 以下。

  列内の null 値の数。

- `NumberOfDistinctValues` – 必須: 数値 (long)。None 以下。

  列内の異なる値の数です。

LongColumnStatisticsData 構造

整数データ列でサポートされる列統計を定義します。

フィールド

- `MinimumValue` – 数値 (long)。

  列内の最小値。

- `MaximumValue` – 数値 (long)。

  列内の最大値。

- `NumberOfNulls` – 必須: 数値 (long)。None 以下。

  列内の null 値の数。

- `NumberOfDistinctValues` – 必須: 数値 (long)。None 以下。

  列内の異なる値の数です。

StringColumnStatisticsData 構造

文字シーケンスのデータ値に対してサポートされる列統計を定義します。

フィールド

- `MaximumLength` – 必須: 数値 (long)。None 以下。

  列内の最も長い文字列のサイズです。
- `AverageLength` – 必須: 数値 (double)。None 以下。

  列内の文字列の平均長。
- `NumberOfNulls` – 必須: 数値 (long)。None 以下。

  列内の null 値の数。
- `NumberOfDistinctValues` – 必須: 数値 (long)。None 以下。

  列内の異なる値の数です。

BinaryColumnStatisticsData 構造

ビットシーケンスデータ値に対してサポートされる列統計を定義します。

フィールド

- `MaximumLength` – 必須: 数値 (long)。None 以下。

  列内の最も長いビットシーケンスのサイズ。
- `AverageLength` – 必須: 数値 (double)。None 以下。

  列内の平均ビットシーケンス長。
- `NumberOfNulls` – 必須: 数値 (long)。None 以下。

  列内の null 値の数。

文字列パターン

API は、さまざまな文字列パラメータとメンバーの有効コンテンツを定義するために、以下の正規表現を使用します。

- 単一行文字列パターン – 「[\u0020-\uD7FF\uE000-\uFFFD\uD800\uDC00-\uDBFF\uDFFF\t]*」
- URI アドレスの複数行文字列パターン – 「[\u0020-\uD7FF\uE000-\uFFFD\uD800\uDC00-\uDBFF\uDFFF\r\n\t]*」
- Logstash Grok 文字列パターン - 「[\u0020-\uD7FF\uE000-\uFFFD\uD800\uDC00-\uDBFF\uDFFF\r\t]*」
- 識別子文字列パターン - 「[A-Za-z_][A-Za-z0-9_]*」
- AWS IAM ARN 文字列パターン - "arn:aws:iam::\d{12}:role/.*"
- バージョン文字列パターン - 「^[a-zA-Z0-9-_]+$」
- ロググループ文字列パターン - 「[\.\-_/#A-Za-z0-9]+」
- ログストリーム文字列パターン - 「[^:*]*」
- カスタム文字列パターン #10 – "[a-zA-Z0-9-_]+"
- カスタム文字列パターン #11 – "[-a-zA-Z0-9+=/:_]*"
- カスタム文字列パターン #12 – "[\S\s]*"
- カスタム文字列パターン #13 – ".*\S.*"
- カスタム文字列パターン #14 – "[a-zA-Z0-9-=._/@]+"
- カスタム文字列パターン #15 – "[1-9][0-9]*|[1-9][0-9]*-[1-9][0-9]*"
- カスタム文字列パターン #16 – "[A-Z][A-Za-z\.]+"
- カスタム文字列パターン #17 – "[\S]*"
- カスタム文字列パターン #18 – "[\w]*"
- カスタム文字列パターン #19 – "arn:aws[a-z\-]*:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\-_/]+"
- カスタム文字列パターン #20 – "subnet-[a-z0-9]+"
- カスタム文字列パターン #21 – "\d{12}"
- カスタム文字列パターン #22 – "([a-z]+)-([a-z]+-)?([a-z]+)-[0-9]+[a-z]+"
- カスタム文字列パターン #23 – "[a-zA-Z0-9.-]*"
- カスタム文字列パターン #24 – "arn:aws[a-z0-9\-]*:lambda:[a-z0-9\-]+:\d{12}:function:([\w\-]{1,64})"
- カスタム文字列パターン #25 – "^(?!(.*[.\/\\]|aws:)).*$"
- カスタム文字列パターン #26 – "[^\r\n]"

- カスタム文字列パターン #27 – "^\w+\.\w+\.\w+$"
- カスタム文字列パターン #28 – "^\w+\.\w+$"
- カスタム文字列パターン #29 – "^$|arn:aws[a-z0-9-]*:kms:.*"
- カスタム文字列パターン #30 – "arn:aws[^:]*:iam::[0-9]*:role/.+"
- カスタム文字列パターン #31 – "[\.\-_A-Za-z0-9]+"
- カスタム文字列パターン #32 – "^s3://([^/]+)/([^/]+/)*([^/]+)$"
- カスタム文字列パターン #33 – ".*"
- カスタム文字列パターン #34 – "^(Sun|Mon|Tue|Wed|Thu|Fri|Sat):([01]?[0-9]|2[0-3])$"
- カスタム文字列パターン #35 – "[a-zA-Z0-9_.-]+"
- カスタム文字列パターン #36 – "^arn:aws(-(cn|us-gov|iso(-[bef])?))?:secretsmanager:.*$"
- カスタム文字列パターン #37 – "\S+"
- カスタム文字列パターン #38 – "^[\x20-\x7E]*$"
- カスタム文字列パターン #39 – "^([a-zA-Z0-9_=]+)\.([a-zA-Z0-9_=]+)\.([a-zA-Z0-9_\-\+\/=]*)"
- カスタム文字列パターン #40 – "^(https?)://[-a-zA-Z0-9+&@#/%?=~_|!:,.;]*[-a-zA-Z0-9+&@#/%=~_|]"
- カスタム文字列パターン #41 – "^(https?):\/\/[^\s/$.?#].[^\s]*$"
- カスタム文字列パターン #42 – "^subnet-[a-z0-9]+$"
- カスタム文字列パターン #43 – "[\p{L}\p{N}\p{P}]*"
- カスタム文字列パターン #44 – "[a-f0-9]{8}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{12}"
- カスタム文字列パターン #45 – "[a-zA-Z0-9-_$#.]+"
- カスタム文字列パターン #46 – "^\d{12}$"
- カスタム文字列パターン #47 – "^(\w+\.)+\w+$"
- カスタム文字列パターン #48 – "^([2-3]|3[.]9)$"
- カスタム文字列パターン #49 – "arn:aws(-(cn|us-gov|iso(-[bef])?))?:glue:.*"
- カスタム文字列パターン #50 – "(^arn:aws(-(cn|us-gov|iso(-[bef])?))?:iam::\w{12}:root)"

- カスタム文字列パターン #51 – "^arn:aws(-(cn|us-gov|iso(-[bef])?))?:iam::[0-9]{12}:role/.+"
- カスタム文字列パターン #52 – "[\s\S]*"
- カスタム文字列パターン #53 – "([\u0020-\uD7FF\uE000-\uFFFD\uD800\uDC00-\uDBFF\uDFFF]|[^\S\r\n"'= ;])*"
- カスタム文字列パターン #54 – "^[A-Z_]+$"
- カスタム文字列パターン #55 – "^[A-Za-z0-9]+$"
- カスタム文字列パターン #56 – "[*A-Za-z0-9_-]*"
- カスタム文字列パターン #57 – "([\u0020-\u007E\r\s\n])*"
- カスタム文字列パターン #58 – "[A-Za-z0-9_-]*"
- カスタム文字列パターン #59 – "([\u0009\u000B\u000C\u0020-\uD7FF\uE000-\uFFFD\uD800\uDC00-\uDBFF\uDFFF])*"
- カスタム文字列パターン #60 – "([\u0020-\uD7FF\uE000-\uFFFD\uD800\uDC00-\uDBFF\uDFFF\s])*"
- カスタム文字列パターン #61 – "([^\r\n])*"

例外

このセクションでは、問題の原因を見つけて解決するために使用できる AWS Glue の例外について説明します。機械学習に関連する例外の HTTP エラーコードと文字列の詳細については、「the section called “AWS Glue 機械学習の例外”」を参照してください。

AccessDeniedException 構造

リソースへのアクセスが拒否されました。

フィールド

- Message – UTF-8 文字列。

  問題を説明するメッセージ。

AlreadyExistsException 構造

作成または追加するリソースはすでに存在します。

フィールド

- Message – UTF-8 文字列。

  問題を説明するメッセージ。

ConcurrentModificationException 構造

2 つのプロセスが同時にリソースを変更しようとしています。

フィールド

- Message – UTF-8 文字列。

  問題を説明するメッセージ。

ConcurrentRunsExceededException 構造

同時に実行されているジョブが多すぎます。

フィールド

- Message – UTF-8 文字列。

  問題を説明するメッセージ。

CrawlerNotRunningException 構造

指定されたクローラーが実行されていません。

フィールド

- Message – UTF-8 文字列。

  問題を説明するメッセージ。

CrawlerRunningException 構造

クローラーが既に実行されているため、操作を実行できません。

フィールド

- Message – UTF-8 文字列。

  問題を説明するメッセージ。

CrawlerStoppingException 構造

指定されたクローラーが停止しています。

フィールド

- Message – UTF-8 文字列。

  問題を説明するメッセージ。

EntityNotFoundException 構造

指定されたエンティティは存在しません

フィールド

- Message – UTF-8 文字列。

  問題を説明するメッセージ。
- FromFederationSource – ブール。

  例外がフェデレーションされたソースに関連するものかどうかを示します。

FederationSourceException 構造

ソースのフェデレーションに失敗しました。

フィールド

- FederationSourceErrorCode – UTF-8 文字列 (有効な値:AccessDeniedException | EntityNotFoundException | InvalidCredentialsException | InvalidInputException | InvalidResponseException | OperationTimeoutException | OperationNotSupportedException | InternalServiceException | PartialFailureException | ThrottlingException)。

問題のエラーコード。

- Message – UTF–8 文字列。

  問題を説明するメッセージ。

FederationSourceRetryableException 構造

ソースのフェデレーションに失敗しましたが、操作を再試行できます。

フィールド

- Message – UTF-8 文字列。

  問題を説明するメッセージ。

GlueEncryptionException 構造

暗号化オペレーションが失敗しました。

フィールド

- Message – UTF-8 文字列。

  問題を説明するメッセージ。

IdempotentParameterMismatchException 構造

同じ一意の識別子が 2 つの異なるレコードに関連付けられていました。

フィールド

- Message – UTF-8 文字列。

  問題を説明するメッセージ。

IllegalWorkflowStateException 構造

リクエストされたオペレーションを実行するには、ワークフローの状態が無効です。

フィールド

- Message – UTF-8 文字列。

  問題を説明するメッセージ。

InternalServiceException 構造

内部サービスエラーが発生しました。

フィールド

- Message – UTF-8 文字列。

  問題を説明するメッセージ。

InvalidExecutionEngineException 構造

不明または無効な実行エンジンが指定されました。

フィールド

- message – UTF-8 文字列。

  問題を説明するメッセージ。

InvalidInputException 構造

指定された入力は無効です。

フィールド

- Message – UTF-8 文字列。

  問題を説明するメッセージ。
- FromFederationSource – ブール。

  例外がフェデレーションされたソースに関連するものかどうかを示します。

InvalidStateException 構造

データが無効な状態であることを示すエラー。

フィールド

- Message – UTF-8 文字列。

  問題を説明するメッセージ。

InvalidTaskStatusTransitionException 構造

あるタスクから次のタスクへの適切な移行に失敗しました。

フィールド

- message – UTF-8 文字列。

  問題を説明するメッセージ。

JobDefinitionErrorException 構造

ジョブ定義が無効です。

フィールド

- message – UTF-8 文字列。

  問題を説明するメッセージ。

JobRunInTerminalStateException 構造

ジョブ実行の終了状態が障害を通知しています。

フィールド

- message – UTF-8 文字列。

  問題を説明するメッセージ。

JobRunInvalidStateTransitionException 構造

ジョブ実行で、ソース状態からターゲット状態への無効な移行が発生しました。

フィールド

- jobRunId – UTF-8 文字列。1 ~ 255 バイト長。Single-line string pattern に一致。

  該当するジョブ実行の ID。
- message – UTF–8 文字列。

  問題を説明するメッセージ。
- sourceState – UTF-8 文字列 (有効な値:STARTING | RUNNING | STOPPING | STOPPED | SUCCEEDED | FAILED | TIMEOUT | ERROR | WAITING | EXPIRED)。

  ソースの状態。
- targetState – UTF-8 文字列 (有効な値:STARTING | RUNNING | STOPPING | STOPPED | SUCCEEDED | FAILED | TIMEOUT | ERROR | WAITING | EXPIRED)。

  ターゲットの状態。

JobRunNotInTerminalStateException 構造

ジョブ実行が終了状態ではありません。

フィールド

- message – UTF-8 文字列。

  問題を説明するメッセージ。

LateRunnerException 構造

ジョブランナーが遅延しています。

フィールド

- Message – UTF-8 文字列。

  問題を説明するメッセージ。

NoScheduleException 構造

該当するスケジュールがありません。

フィールド

- Message – UTF-8 文字列。

  問題を説明するメッセージ。

OperationTimeoutException 構造

オペレーションがタイムアウトしました。

フィールド

- Message – UTF-8 文字列。

  問題を説明するメッセージ。

ResourceNotReadyException 構造

リソースのトランザクション準備ができていませんでした。

フィールド

- Message – UTF-8 文字列。

  問題を説明するメッセージ。

ResourceNumberLimitExceededException 構造

リソースの数値制限を超えました。

フィールド

- Message – UTF-8 文字列。

  問題を説明するメッセージ。

SchedulerNotRunningException 構造

指定されたスケジューラが実行されていません。

フィールド

- Message – UTF-8 文字列。

  問題を説明するメッセージ。

SchedulerRunningException 構造

指定されたスケジューラは既に実行されています。

フィールド

- Message – UTF-8 文字列。

  問題を説明するメッセージ。

SchedulerTransitioningException 構造

指定されたスケジューラが移行中です。

フィールド

- Message – UTF-8 文字列。

  問題を説明するメッセージ。

UnrecognizedRunnerException 構造

ジョブランナーが認識されませんでした。

フィールド

- Message – UTF-8 文字列。

  問題を説明するメッセージ。

ValidationException 構造

値を検証できませんでした。

フィールド

- Message – UTF-8 文字列。

  問題を説明するメッセージ。

VersionMismatchException 構造

バージョンの競合がありました。

フィールド

- message – UTF-8 文字列。

  問題を説明するメッセージ。

AWS SDK を使用した AWS Glue API コード例

以下は、AWS Glue Software Development Kit (SDK) で AWS を使用する方法を説明するコード例です。

基本は、重要なオペレーションをサービス内で実行する方法を示すコード例です。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

AWS SDK デベロッパーガイドとコード例の詳細なリストについては、「このサービスを AWS SDK で使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

開始方法

Hello AWS Glue

次のコード例は、AWS Glue の使用を開始する方法を示しています。

.NET

SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
namespace GlueActions;

public class HelloGlue
{
    private static ILogger logger = null!;

    static async Task Main(string[] args)
    {
        // Set up dependency injection for AWS Glue.
        using var host = Host.CreateDefaultBuilder(args)
```

```
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
 LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft",
 LogLevel.Trace))
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonGlue>()
                .AddTransient<GlueWrapper>()
            )
            .Build();

        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<HelloGlue>();
        var glueClient = host.Services.GetRequiredService<IAmazonGlue>();

        var request = new ListJobsRequest();

        var jobNames = new List<string>();

        do
        {
            var response = await glueClient.ListJobsAsync(request);
            jobNames.AddRange(response.JobNames);
            request.NextToken = response.NextToken;
        }
        while (request.NextToken is not null);

        Console.Clear();
        Console.WriteLine("Hello, Glue. Let's list your existing Glue Jobs:");
        if (jobNames.Count == 0)
        {
            Console.WriteLine("You don't have any AWS Glue jobs.");
        }
        else
        {
            jobNames.ForEach(Console.WriteLine);
        }
    }
}
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「ListJobs」を参照してください。

C++

SDK for C++

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

CMakeLists.txt CMake ファイルのコード。

```
# Set the minimum required version of CMake for this project.
cmake_minimum_required(VERSION 3.13)

# Set the AWS service components used by this project.
set(SERVICE_COMPONENTS glue)

# Set this project's name.
project("hello_glue")

# Set the C++ standard to use to build this target.
# At least C++ 11 is required for the AWS SDK for C++.
set(CMAKE_CXX_STANDARD 11)

# Use the MSVC variable to determine if this is a Windows build.
set(WINDOWS_BUILD ${MSVC})

if (WINDOWS_BUILD) # Set the location where CMake can find the installed
 libraries for the AWS SDK.
    string(REPLACE ";" "/aws-cpp-sdk-all;" SYSTEM_MODULE_PATH
 "${CMAKE_SYSTEM_PREFIX_PATH}/aws-cpp-sdk-all")
    list(APPEND CMAKE_PREFIX_PATH ${SYSTEM_MODULE_PATH})
endif ()

# Find the AWS SDK for C++ package.
find_package(AWSSDK REQUIRED COMPONENTS ${SERVICE_COMPONENTS})

if (WINDOWS_BUILD AND AWSSDK_INSTALL_AS_SHARED_LIBS)
```

```
    # Copy relevant AWS SDK for C++ libraries into the current binary directory
 for running and debugging.

    # set(BIN_SUB_DIR "/Debug") # if you are building from the command line you
 may need to uncomment this
                                # and set the proper subdirectory to the
 executables' location.

    AWSSDK_CPY_DYN_LIBS(SERVICE_COMPONENTS ""
 ${CMAKE_CURRENT_BINARY_DIR}${BIN_SUB_DIR})
endif ()

add_executable(${PROJECT_NAME}
        hello_glue.cpp)

target_link_libraries(${PROJECT_NAME}
        ${AWSSDK_LINK_LIBRARIES})
```

hello_glue.cpp ソースファイルのコード。

```
#include <aws/core/Aws.h>
#include <aws/glue/GlueClient.h>
#include <aws/glue/model/ListJobsRequest.h>
#include <iostream>

/*
 *  A "Hello Glue" starter application which initializes an AWS Glue client and
 lists the
 *  AWS Glue job definitions.
 *
 *  main function
 *
 *  Usage: 'hello_glue'
 *
 */

int main(int argc, char **argv) {
    Aws::SDKOptions options;
    // Optionally change the log level for debugging.
//   options.loggingOptions.logLevel = Utils::Logging::LogLevel::Debug;
    Aws::InitAPI(options); // Should only be called once.
    int result = 0;
```

```
    {
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

        Aws::Glue::GlueClient glueClient(clientConfig);

        std::vector<Aws::String> jobs;

        Aws::String nextToken;  // Used for pagination.
        do {
            Aws::Glue::Model::ListJobsRequest listJobsRequest;
            if (!nextToken.empty()) {
                listJobsRequest.SetNextToken(nextToken);
            }

            Aws::Glue::Model::ListJobsOutcome listRunsOutcome =
 glueClient.ListJobs(
                    listJobsRequest);

            if (listRunsOutcome.IsSuccess()) {
                const std::vector<Aws::String> &jobNames =
 listRunsOutcome.GetResult().GetJobNames();
                jobs.insert(jobs.end(), jobNames.begin(), jobNames.end());

                nextToken = listRunsOutcome.GetResult().GetNextToken();
            } else {
                std::cerr << "Error listing jobs. "
                          << listRunsOutcome.GetError().GetMessage()
                          << std::endl;
                result = 1;
                break;
            }
        } while (!nextToken.empty());

        std::cout << "Your account has " << jobs.size() << " jobs."
                  << std::endl;
        for (size_t i = 0; i < jobs.size(); ++i) {
            std::cout << "   " << i + 1 << ". " << jobs[i] << std::endl;
        }
    }
    Aws::ShutdownAPI(options); // Should only be called once.
    return result;
}
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「ListJobs」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
package com.example.glue;

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.glue.GlueClient;
import software.amazon.awssdk.services.glue.model.ListJobsRequest;
import software.amazon.awssdk.services.glue.model.ListJobsResponse;
import java.util.List;

public class HelloGlue {
    public static void main(String[] args) {
        GlueClient glueClient = GlueClient.builder()
                .region(Region.US_EAST_1)
                .build();

        listJobs(glueClient);
    }

    public static void listJobs(GlueClient glueClient) {
        ListJobsRequest request = ListJobsRequest.builder()
                .maxResults(10)
                .build();
        ListJobsResponse response = glueClient.listJobs(request);
        List<String> jobList = response.jobNames();
        jobList.forEach(job -> {
            System.out.println("Job Name: " + job);
        });
```

```
    }
}
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「ListJobs」を参照してください。

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
import { ListJobsCommand, GlueClient } from "@aws-sdk/client-glue";

const client = new GlueClient({});

export const main = async () => {
  const command = new ListJobsCommand({});

  const { JobNames } = await client.send(command);
  const formattedJobNames = JobNames.join("\n");
  console.log("Job names: ");
  console.log(formattedJobNames);
  return JobNames;
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「ListJobs」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
import boto3
from botocore.exceptions import ClientError


def hello_glue():
    """
    Lists the job definitions in your AWS Glue account, using the AWS SDK for
 Python (Boto3).
    """
    try:
        # Create the Glue client
        glue = boto3.client("glue")

        # List the jobs, limiting the results to 10 per page
        paginator = glue.get_paginator("get_jobs")
        response_iterator = paginator.paginate(
            PaginationConfig={"MaxItems": 10, "PageSize": 10}
        )

        # Print the job names
        print("Here are the jobs in your account:")
        for page in response_iterator:
            for job in page["Jobs"]:
                print(f"\t{job['Name']}")

    except ClientError as e:
        print(f"Error: {e}")


if __name__ == "__main__":
    hello_glue()
```

- API の詳細については、「AWS SDK for Python (Boto3) API リファレンス」の「ListJobs」を参照してください。

Ruby

SDK for Ruby

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
require 'aws-sdk-glue'
require 'logger'

# GlueManager is a class responsible for managing AWS Glue operations
# such as listing all Glue jobs in the current AWS account.
class GlueManager
  def initialize(client)
    @client = client
    @logger = Logger.new($stdout)
  end

  # Lists and prints all Glue jobs in the current AWS account.
  def list_jobs
    @logger.info('Here are the Glue jobs in your account:')

    paginator = @client.get_jobs(max_results: 10)
    jobs = []

    paginator.each_page do |page|
      jobs.concat(page.jobs)
    end

    if jobs.empty?
      @logger.info("You don't have any Glue jobs.")
```

```
    else
      jobs.each do |job|
        @logger.info("- #{job.name}")
      end
    end
  end
end

if $PROGRAM_NAME == __FILE__
  glue_client = Aws::Glue::Client.new
  manager = GlueManager.new(glue_client)
  manager.list_jobs
end
```

- API の詳細については、「AWS SDK for Ruby API リファレンス」の「ListJobs」を参照してください。

Rust

SDK for Rust

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
        let mut list_jobs = glue.list_jobs().into_paginator().send();
        while let Some(list_jobs_output) = list_jobs.next().await {
            match list_jobs_output {
                Ok(list_jobs) => {
                    let names = list_jobs.job_names();
                    info!(?names, "Found these jobs")
                }
                Err(err) => return Err(GlueMvpError::from_glue_sdk(err)),
            }
        }
```

- API の詳細については、「AWS SDK for Rust API リファレンス」の「ListJobs」を参照してください。

AWS Glue API コード例

- AWS SDK を使用した AWS Glue の基本的な例
  - Hello AWS Glue
  - AWS SDKを備えた AWS Glue の基本の詳細
  - AWS SDK を使用した AWS Glue 向けアクション
    - AWS SDK で CreateCrawler を使用する
    - AWS SDK または CLI で CreateJob を使用する
    - AWS SDK で DeleteCrawler を使用する
    - AWS SDK で DeleteDatabase を使用する
    - AWS SDK または CLI で DeleteJob を使用する
    - AWS SDK で DeleteTable を使用する
    - AWS SDK で GetCrawler を使用する
    - AWS SDK で GetDatabase を使用する
    - AWS SDK または CLI で GetDatabases を使用する
    - AWS SDK または CLI で GetJob を使用する
    - AWS SDK または CLI で GetJobRun を使用する
    - AWS SDK または CLI で GetJobRuns を使用する
    - AWS SDK または CLI で GetTables を使用する
    - AWS SDK で ListJobs を使用する
    - AWS SDK または CLI で StartCrawler を使用する
    - AWS SDK または CLI で StartJobRun を使用する

AWS SDK を使用した AWS Glue の基本的な例

次のコード例は、AWS SDK で AWS Glue の基本的な使用方法を示しています。

例

- Hello AWS Glue
- AWS SDKを備えた AWS Glue の基本の詳細

- AWS SDK を使用した AWS Glue 向けアクション
  - AWS SDK で CreateCrawler を使用する
  - AWS SDK または CLI で CreateJob を使用する
  - AWS SDK で DeleteCrawler を使用する
  - AWS SDK で DeleteDatabase を使用する
  - AWS SDK または CLI で DeleteJob を使用する
  - AWS SDK で DeleteTable を使用する
  - AWS SDK で GetCrawler を使用する
  - AWS SDK で GetDatabase を使用する
  - AWS SDK または CLI で GetDatabases を使用する
  - AWS SDK または CLI で GetJob を使用する
  - AWS SDK または CLI で GetJobRun を使用する
  - AWS SDK または CLI で GetJobRuns を使用する
  - AWS SDK または CLI で GetTables を使用する
  - AWS SDK で ListJobs を使用する
  - AWS SDK または CLI で StartCrawler を使用する
  - AWS SDK または CLI で StartJobRun を使用する

Hello AWS Glue

次のコード例は、AWS Glue の使用を開始する方法を示しています。

.NET

SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
namespace GlueActions;
```

```
public class HelloGlue
{
    private static ILogger logger = null!;

    static async Task Main(string[] args)
    {
        // Set up dependency injection for AWS Glue.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
 LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft",
 LogLevel.Trace))
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonGlue>()
                .AddTransient<GlueWrapper>()
            )
            .Build();

        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<HelloGlue>();
        var glueClient = host.Services.GetRequiredService<IAmazonGlue>();

        var request = new ListJobsRequest();

        var jobNames = new List<string>();

        do
        {
            var response = await glueClient.ListJobsAsync(request);
            jobNames.AddRange(response.JobNames);
            request.NextToken = response.NextToken;
        }
        while (request.NextToken is not null);

        Console.Clear();
        Console.WriteLine("Hello, Glue. Let's list your existing Glue Jobs:");
        if (jobNames.Count == 0)
        {
            Console.WriteLine("You don't have any AWS Glue jobs.");
        }
        else
        {
```

```
            jobNames.ForEach(Console.WriteLine);
        }
    }
}
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「ListJobs」を参照してください。

C++

SDK for C++

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

CMakeLists.txt CMake ファイルのコード。

```
# Set the minimum required version of CMake for this project.
cmake_minimum_required(VERSION 3.13)

# Set the AWS service components used by this project.
set(SERVICE_COMPONENTS glue)

# Set this project's name.
project("hello_glue")

# Set the C++ standard to use to build this target.
# At least C++ 11 is required for the AWS SDK for C++.
set(CMAKE_CXX_STANDARD 11)

# Use the MSVC variable to determine if this is a Windows build.
set(WINDOWS_BUILD ${MSVC})

if (WINDOWS_BUILD) # Set the location where CMake can find the installed
 libraries for the AWS SDK.
    string(REPLACE ";" "/aws-cpp-sdk-all;" SYSTEM_MODULE_PATH
 "${CMAKE_SYSTEM_PREFIX_PATH}/aws-cpp-sdk-all")
```

```
    list(APPEND CMAKE_PREFIX_PATH ${SYSTEM_MODULE_PATH})
endif ()

# Find the AWS SDK for C++ package.
find_package(AWSSDK REQUIRED COMPONENTS ${SERVICE_COMPONENTS})

if (WINDOWS_BUILD AND AWSSDK_INSTALL_AS_SHARED_LIBS)
     # Copy relevant AWS SDK for C++ libraries into the current binary directory
 for running and debugging.

     # set(BIN_SUB_DIR "/Debug") # if you are building from the command line you
 may need to uncomment this
                                 # and set the proper subdirectory to the
 executables' location.

     AWSSDK_CPY_DYN_LIBS(SERVICE_COMPONENTS ""
 ${CMAKE_CURRENT_BINARY_DIR}${BIN_SUB_DIR})
endif ()

add_executable(${PROJECT_NAME}
        hello_glue.cpp)

target_link_libraries(${PROJECT_NAME}
        ${AWSSDK_LINK_LIBRARIES})
```

hello_glue.cpp ソースファイルのコード。

```
#include <aws/core/Aws.h>
#include <aws/glue/GlueClient.h>
#include <aws/glue/model/ListJobsRequest.h>
#include <iostream>

/*
 *  A "Hello Glue" starter application which initializes an AWS Glue client and
 lists the
 *  AWS Glue job definitions.
 *
 *  main function
 *
 *  Usage: 'hello_glue'
 *
 */
```

```
int main(int argc, char **argv) {
    Aws::SDKOptions options;
    // Optionally change the log level for debugging.
//   options.loggingOptions.logLevel = Utils::Logging::LogLevel::Debug;
    Aws::InitAPI(options); // Should only be called once.
    int result = 0;
    {
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

        Aws::Glue::GlueClient glueClient(clientConfig);

        std::vector<Aws::String> jobs;

        Aws::String nextToken;  // Used for pagination.
        do {
            Aws::Glue::Model::ListJobsRequest listJobsRequest;
            if (!nextToken.empty()) {
                listJobsRequest.SetNextToken(nextToken);
            }

            Aws::Glue::Model::ListJobsOutcome listRunsOutcome =
 glueClient.ListJobs(
                    listJobsRequest);

            if (listRunsOutcome.IsSuccess()) {
                const std::vector<Aws::String> &jobNames =
 listRunsOutcome.GetResult().GetJobNames();
                jobs.insert(jobs.end(), jobNames.begin(), jobNames.end());

                nextToken = listRunsOutcome.GetResult().GetNextToken();
            } else {
                std::cerr << "Error listing jobs. "
                          << listRunsOutcome.GetError().GetMessage()
                          << std::endl;
                result = 1;
                break;
            }
        } while (!nextToken.empty());

        std::cout << "Your account has " << jobs.size() << " jobs."
                  << std::endl;
```

```
        for (size_t i = 0; i < jobs.size(); ++i) {
            std::cout << "   " << i + 1 << ". " << jobs[i] << std::endl;
        }
    }
    Aws::ShutdownAPI(options); // Should only be called once.
    return result;
}
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「ListJobs」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
package com.example.glue;

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.glue.GlueClient;
import software.amazon.awssdk.services.glue.model.ListJobsRequest;
import software.amazon.awssdk.services.glue.model.ListJobsResponse;
import java.util.List;

public class HelloGlue {
    public static void main(String[] args) {
        GlueClient glueClient = GlueClient.builder()
                .region(Region.US_EAST_1)
                .build();

        listJobs(glueClient);
    }

    public static void listJobs(GlueClient glueClient) {
        ListJobsRequest request = ListJobsRequest.builder()
```

```
                .maxResults(10)
                .build();
        ListJobsResponse response = glueClient.listJobs(request);
        List<String> jobList = response.jobNames();
        jobList.forEach(job -> {
            System.out.println("Job Name: " + job);
        });
    }
}
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「ListJobs」を参照してください。

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
import { ListJobsCommand, GlueClient } from "@aws-sdk/client-glue";

const client = new GlueClient({});

export const main = async () => {
  const command = new ListJobsCommand({});

  const { JobNames } = await client.send(command);
  const formattedJobNames = JobNames.join("\n");
  console.log("Job names: ");
  console.log(formattedJobNames);
  return JobNames;
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「ListJobs」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
import boto3
from botocore.exceptions import ClientError


def hello_glue():
    """
    Lists the job definitions in your AWS Glue account, using the AWS SDK for
 Python (Boto3).
    """
    try:
        # Create the Glue client
        glue = boto3.client("glue")

        # List the jobs, limiting the results to 10 per page
        paginator = glue.get_paginator("get_jobs")
        response_iterator = paginator.paginate(
            PaginationConfig={"MaxItems": 10, "PageSize": 10}
        )

        # Print the job names
        print("Here are the jobs in your account:")
        for page in response_iterator:
            for job in page["Jobs"]:
                print(f"\t{job['Name']}")

    except ClientError as e:
        print(f"Error: {e}")


if __name__ == "__main__":
    hello_glue()
```

- API の詳細については、「AWS SDK for Python (Boto3) API リファレンス」の「ListJobs」を参照してください。

Ruby

SDK for Ruby

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
require 'aws-sdk-glue'
require 'logger'

# GlueManager is a class responsible for managing AWS Glue operations
# such as listing all Glue jobs in the current AWS account.
class GlueManager
  def initialize(client)
    @client = client
    @logger = Logger.new($stdout)
  end

  # Lists and prints all Glue jobs in the current AWS account.
  def list_jobs
    @logger.info('Here are the Glue jobs in your account:')

    paginator = @client.get_jobs(max_results: 10)
    jobs = []

    paginator.each_page do |page|
      jobs.concat(page.jobs)
    end

    if jobs.empty?
      @logger.info("You don't have any Glue jobs.")
```

```
    else
      jobs.each do |job|
        @logger.info("- #{job.name}")
      end
    end
  end
end

if $PROGRAM_NAME == __FILE__
  glue_client = Aws::Glue::Client.new
  manager = GlueManager.new(glue_client)
  manager.list_jobs
end
```

- API の詳細については、「AWS SDK for Ruby API リファレンス」の「ListJobs」を参照してください。

Rust

SDK for Rust

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
        let mut list_jobs = glue.list_jobs().into_paginator().send();
        while let Some(list_jobs_output) = list_jobs.next().await {
            match list_jobs_output {
                Ok(list_jobs) => {
                    let names = list_jobs.job_names();
                    info!(?names, "Found these jobs")
                }
                Err(err) => return Err(GlueMvpError::from_glue_sdk(err)),
            }
        }
```

- API の詳細については、「AWS SDK for Rust API リファレンス」の「ListJobs」を参照してください。

AWS SDK デベロッパーガイドとコード例の詳細なリストについては、「このサービスを AWS SDK で使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDKを備えた AWS Glue の基本の詳細

次のコード例は、以下を実行する方法を示しています。

- パブリック Amazon S3 バケットをクロールし、CSV 形式のメタデータのデータベースを生成するクローラーを作成する。
- AWS Glue Data Catalog 内のデータベースとテーブルに関する情報を一覧表示します。
- S3 バケットから CSV 形式のデータを抽出するジョブを作成し、そのデータを変換して JSON 形式の出力を別の S3 バケットにロードする。
- ジョブ実行に関する情報を一覧表示し、変換されたデータを表示してリソースをクリーンアップする。

詳細については、「チュートリアル: AWS Glue Studio の開始方法」をご参照ください。

.NET

SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

シナリオで使用される AWS Glue 関数をラップするクラスを作成します。

```
using System.Net;

namespace GlueActions;
```

```
public class GlueWrapper
{
    private readonly IAmazonGlue _amazonGlue;

    /// <summary>
    /// Constructor for the AWS Glue actions wrapper.
    /// </summary>
    /// <param name="amazonGlue"></param>
    public GlueWrapper(IAmazonGlue amazonGlue)
    {
        _amazonGlue = amazonGlue;
    }

    /// <summary>
    /// Create an AWS Glue crawler.
    /// </summary>
    /// <param name="crawlerName">The name for the crawler.</param>
    /// <param name="crawlerDescription">A description of the crawler.</param>
    /// <param name="role">The AWS Identity and Access Management (IAM) role to
    /// be assumed by the crawler.</param>
    /// <param name="schedule">The schedule on which the crawler will be
 executed.</param>
    /// <param name="s3Path">The path to the Amazon Simple Storage Service
 (Amazon S3)
    /// bucket where the Python script has been stored.</param>
    /// <param name="dbName">The name to use for the database that will be
    /// created by the crawler.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> CreateCrawlerAsync(
        string crawlerName,
        string crawlerDescription,
        string role,
        string schedule,
        string s3Path,
        string dbName)
    {
        var s3Target = new S3Target
        {
            Path = s3Path,
        };

        var targetList = new List<S3Target>
        {
            s3Target,
```

```
        };

        var targets = new CrawlerTargets
        {
            S3Targets = targetList,
        };

        var crawlerRequest = new CreateCrawlerRequest
        {
            DatabaseName = dbName,
            Name = crawlerName,
            Description = crawlerDescription,
            Targets = targets,
            Role = role,
            Schedule = schedule,
        };

        var response = await _amazonGlue.CreateCrawlerAsync(crawlerRequest);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
    /// Create an AWS Glue job.
    /// </summary>
    /// <param name="jobName">The name of the job.</param>
    /// <param name="roleName">The name of the IAM role to be assumed by
    /// the job.</param>
    /// <param name="description">A description of the job.</param>
    /// <param name="scriptUrl">The URL to the script.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> CreateJobAsync(string dbName, string tableName,
string bucketUrl, string jobName, string roleName, string description, string
scriptUrl)
    {
        var command = new JobCommand
        {
            PythonVersion = "3",
            Name = "glueetl",
            ScriptLocation = scriptUrl,
        };

        var arguments = new Dictionary<string, string>
        {
```

```
            { "--input_database", dbName },
            { "--input_table", tableName },
            { "--output_bucket_url", bucketUrl }
        };

        var request = new CreateJobRequest
        {
            Command = command,
            DefaultArguments = arguments,
            Description = description,
            GlueVersion = "3.0",
            Name = jobName,
            NumberOfWorkers = 10,
            Role = roleName,
            WorkerType = "G.1X"
        };

        var response = await _amazonGlue.CreateJobAsync(request);
        return response.HttpStatusCode == HttpStatusCode.OK;
    }


    /// <summary>
    /// Delete an AWS Glue crawler.
    /// </summary>
    /// <param name="crawlerName">The name of the crawler.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteCrawlerAsync(string crawlerName)
    {
        var response = await _amazonGlue.DeleteCrawlerAsync(new
DeleteCrawlerRequest { Name = crawlerName });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }


    /// <summary>
    /// Delete the AWS Glue database.
    /// </summary>
    /// <param name="dbName">The name of the database.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteDatabaseAsync(string dbName)
    {
        var response = await _amazonGlue.DeleteDatabaseAsync(new
DeleteDatabaseRequest { Name = dbName });
```

```
        return response.HttpStatusCode == HttpStatusCode.OK;
    }


    /// <summary>
    /// Delete an AWS Glue job.
    /// </summary>
    /// <param name="jobName">The name of the job.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteJobAsync(string jobName)
    {
        var response = await _amazonGlue.DeleteJobAsync(new DeleteJobRequest
{ JobName = jobName });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }


    /// <summary>
    /// Delete a table from an AWS Glue database.
    /// </summary>
    /// <param name="tableName">The table to delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteTableAsync(string dbName, string tableName)
    {
        var response = await _amazonGlue.DeleteTableAsync(new DeleteTableRequest
{ Name = tableName, DatabaseName = dbName });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }


    /// <summary>
    /// Get information about an AWS Glue crawler.
    /// </summary>
    /// <param name="crawlerName">The name of the crawler.</param>
    /// <returns>A Crawler object describing the crawler.</returns>
    public async Task<Crawler?> GetCrawlerAsync(string crawlerName)
    {
        var crawlerRequest = new GetCrawlerRequest
        {
            Name = crawlerName,
        };

        var response = await _amazonGlue.GetCrawlerAsync(crawlerRequest);
        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
```

```
        {
            var databaseName = response.Crawler.DatabaseName;
            Console.WriteLine($"{crawlerName} has the database {databaseName}");
            return response.Crawler;
        }

        Console.WriteLine($"No information regarding {crawlerName} could be
 found.");
        return null;
    }


    /// <summary>
    /// Get information about the state of an AWS Glue crawler.
    /// </summary>
    /// <param name="crawlerName">The name of the crawler.</param>
    /// <returns>A value describing the state of the crawler.</returns>
    public async Task<CrawlerState> GetCrawlerStateAsync(string crawlerName)
    {
        var response = await _amazonGlue.GetCrawlerAsync(
            new GetCrawlerRequest { Name = crawlerName });
        return response.Crawler.State;
    }


    /// <summary>
    /// Get information about an AWS Glue database.
    /// </summary>
    /// <param name="dbName">The name of the database.</param>
    /// <returns>A Database object containing information about the database.</
returns>
    public async Task<Database> GetDatabaseAsync(string dbName)
    {
        var databasesRequest = new GetDatabaseRequest
        {
            Name = dbName,
        };

        var response = await _amazonGlue.GetDatabaseAsync(databasesRequest);
        return response.Database;
    }


    /// <summary>
```

```
    /// Get information about a specific AWS Glue job run.
    /// </summary>
    /// <param name="jobName">The name of the job.</param>
    /// <param name="jobRunId">The Id of the job run.</param>
    /// <returns>A JobRun object with information about the job run.</returns>
    public async Task<JobRun> GetJobRunAsync(string jobName, string jobRunId)
    {
        var response = await _amazonGlue.GetJobRunAsync(new GetJobRunRequest
{ JobName = jobName, RunId = jobRunId });
        return response.JobRun;
    }


    /// <summary>
    /// Get information about all AWS Glue runs of a specific job.
    /// </summary>
    /// <param name="jobName">The name of the job.</param>
    /// <returns>A list of JobRun objects.</returns>
    public async Task<List<JobRun>> GetJobRunsAsync(string jobName)
    {
        var jobRuns = new List<JobRun>();

        var request = new GetJobRunsRequest
        {
            JobName = jobName,
        };

        // No need to loop to get all the log groups--the SDK does it for us
behind the scenes
        var paginatorForJobRuns =
            _amazonGlue.Paginators.GetJobRuns(request);

        await foreach (var response in paginatorForJobRuns.Responses)
        {
            response.JobRuns.ForEach(jobRun =>
            {
                jobRuns.Add(jobRun);
            });
        }

        return jobRuns;
    }
```

```
    /// <summary>
    /// Get a list of tables for an AWS Glue database.
    /// </summary>
    /// <param name="dbName">The name of the database.</param>
    /// <returns>A list of Table objects.</returns>
    public async Task<List<Table>> GetTablesAsync(string dbName)
    {
        var request = new GetTablesRequest { DatabaseName = dbName };
        var tables = new List<Table>();

        // Get a paginator for listing the tables.
        var tablePaginator = _amazonGlue.Paginators.GetTables(request);

        await foreach (var response in tablePaginator.Responses)
        {
            tables.AddRange(response.TableList);
        }

        return tables;
    }


    /// <summary>
    /// List AWS Glue jobs using a paginator.
    /// </summary>
    /// <returns>A list of AWS Glue job names.</returns>
    public async Task<List<string>> ListJobsAsync()
    {
        var jobNames = new List<string>();

        var listJobsPaginator = _amazonGlue.Paginators.ListJobs(new
ListJobsRequest { MaxResults = 10 });
        await foreach (var response in listJobsPaginator.Responses)
        {
            jobNames.AddRange(response.JobNames);
        }

        return jobNames;
    }


    /// <summary>
    /// Start an AWS Glue crawler.
    /// </summary>
```

```
    /// <param name="crawlerName">The name of the crawler.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> StartCrawlerAsync(string crawlerName)
    {
        var crawlerRequest = new StartCrawlerRequest
        {
            Name = crawlerName,
        };

        var response = await _amazonGlue.StartCrawlerAsync(crawlerRequest);

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
    /// Start an AWS Glue job run.
    /// </summary>
    /// <param name="jobName">The name of the job.</param>
    /// <returns>A string representing the job run Id.</returns>
    public async Task<string> StartJobRunAsync(
        string jobName,
        string inputDatabase,
        string inputTable,
        string bucketName)
    {
        var request = new StartJobRunRequest
        {
            JobName = jobName,
            Arguments = new Dictionary<string, string>
            {
                {"--input_database", inputDatabase},
                {"--input_table", inputTable},
                {"--output_bucket_url", $"s3://{bucketName}/"}
            }
        };

        var response = await _amazonGlue.StartJobRunAsync(request);
        return response.JobRunId;
    }

}
```

シナリオを実行するクラスを作成します。

```
global using Amazon.Glue;
global using GlueActions;
global using Microsoft.Extensions.Configuration;
global using Microsoft.Extensions.DependencyInjection;
global using Microsoft.Extensions.Hosting;
global using Microsoft.Extensions.Logging;
global using Microsoft.Extensions.Logging.Console;
global using Microsoft.Extensions.Logging.Debug;



using Amazon.Glue.Model;
using Amazon.S3;
using Amazon.S3.Model;

namespace GlueBasics;

public class GlueBasics
{
    private static ILogger logger = null!;
    private static IConfiguration _configuration = null!;

    static async Task Main(string[] args)
    {
        // Set up dependency injection for AWS Glue.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
 LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft",
 LogLevel.Trace))
            .ConfigureServices((_, services) =>
            services.AddAWSService<IAmazonGlue>()
            .AddTransient<GlueWrapper>()
            .AddTransient<UiWrapper>()
            )
            .Build();
```

```
        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
        .CreateLogger<GlueBasics>();

        _configuration = new ConfigurationBuilder()
            .SetBasePath(Directory.GetCurrentDirectory())
            .AddJsonFile("settings.json") // Load settings from .json file.
            .AddJsonFile("settings.local.json",
                true) // Optionally load local settings.
            .Build();

        // These values are stored in settings.json
        // Once you have run the CDK script to deploy the resources,
        // edit the file to set "BucketName", "RoleName", and "ScriptURL"
        // to the appropriate values. Also set "CrawlerName" to the name
        // you want to give the crawler when it is created.
        string bucketName = _configuration["BucketName"]!;
        string bucketUrl = _configuration["BucketUrl"]!;
        string crawlerName = _configuration["CrawlerName"]!;
        string roleName = _configuration["RoleName"]!;
        string sourceData = _configuration["SourceData"]!;
        string dbName = _configuration["DbName"]!;
        string cron = _configuration["Cron"]!;
        string scriptUrl = _configuration["ScriptURL"]!;
        string jobName = _configuration["JobName"]!;

        var wrapper = host.Services.GetRequiredService<GlueWrapper>();
        var uiWrapper = host.Services.GetRequiredService<UiWrapper>();

        uiWrapper.DisplayOverview();
        uiWrapper.PressEnter();

        // Create the crawler and wait for it to be ready.
        uiWrapper.DisplayTitle("Create AWS Glue crawler");
        Console.WriteLine("Let's begin by creating the AWS Glue crawler.");

        var crawlerDescription = "Crawler created for the AWS Glue Basics
scenario.";
        var crawlerCreated = await wrapper.CreateCrawlerAsync(crawlerName,
crawlerDescription, roleName, cron, sourceData, dbName);
        if (crawlerCreated)
        {
            Console.WriteLine($"The crawler: {crawlerName} has been created. Now
let's wait until it's ready.");
            CrawlerState crawlerState;
```

```
            do
            {
                crawlerState = await wrapper.GetCrawlerStateAsync(crawlerName);
            }
            while (crawlerState != "READY");
            Console.WriteLine($"The crawler {crawlerName} is now ready for
use.");
        }
        else
        {
            Console.WriteLine($"Couldn't create crawler {crawlerName}.");
            return; // Exit the application.
        }

        uiWrapper.DisplayTitle("Start AWS Glue crawler");
        Console.WriteLine("Now let's wait until the crawler has successfully
started.");
        var crawlerStarted = await wrapper.StartCrawlerAsync(crawlerName);
        if (crawlerStarted)
        {
            CrawlerState crawlerState;
            do
            {
                crawlerState = await wrapper.GetCrawlerStateAsync(crawlerName);
            }
            while (crawlerState != "READY");
            Console.WriteLine($"The crawler {crawlerName} is now ready for
use.");
        }
        else
        {
            Console.WriteLine($"Couldn't start the crawler {crawlerName}.");
            return; // Exit the application.
        }

        uiWrapper.PressEnter();

        Console.WriteLine($"\nLet's take a look at the database: {dbName}");
        var database = await wrapper.GetDatabaseAsync(dbName);

        if (database != null)
        {
            uiWrapper.DisplayTitle($"{database.Name} Details");
```

```
            Console.WriteLine($"{database.Name} created on
{database.CreateTime}");
            Console.WriteLine(database.Description);
        }

        uiWrapper.PressEnter();

        var tables = await wrapper.GetTablesAsync(dbName);
        if (tables.Count > 0)
        {
            tables.ForEach(table =>
            {
                Console.WriteLine($"{table.Name}\tCreated:
{table.CreateTime}\tUpdated: {table.UpdateTime}");
            });
        }

        uiWrapper.PressEnter();

        uiWrapper.DisplayTitle("Create AWS Glue job");
        Console.WriteLine("Creating a new AWS Glue job.");
        var description = "An AWS Glue job created using the AWS SDK for .NET";
        await wrapper.CreateJobAsync(dbName, tables[0].Name, bucketUrl, jobName,
roleName, description, scriptUrl);

        uiWrapper.PressEnter();

        uiWrapper.DisplayTitle("Starting AWS Glue job");
        Console.WriteLine("Starting the new AWS Glue job...");
        var jobRunId = await wrapper.StartJobRunAsync(jobName, dbName,
tables[0].Name, bucketName);
        var jobRunComplete = false;
        var jobRun = new JobRun();
        do
        {
            jobRun = await wrapper.GetJobRunAsync(jobName, jobRunId);
            if (jobRun.JobRunState == "SUCCEEDED" || jobRun.JobRunState ==
"STOPPED" ||
                jobRun.JobRunState == "FAILED" || jobRun.JobRunState ==
"TIMEOUT")
            {
                jobRunComplete = true;
            }
        } while (!jobRunComplete);
```

```
        uiWrapper.DisplayTitle($"Data in {bucketName}");

        // Get the list of data stored in the S3 bucket.
        var s3Client = new AmazonS3Client();

        var response = await s3Client.ListObjectsAsync(new ListObjectsRequest
{ BucketName = bucketName });
        response.S3Objects.ForEach(s3Object =>
        {
            Console.WriteLine(s3Object.Key);
        });

        uiWrapper.DisplayTitle("AWS Glue jobs");
        var jobNames = await wrapper.ListJobsAsync();
        jobNames.ForEach(jobName =>
        {
            Console.WriteLine(jobName);
        });

        uiWrapper.PressEnter();

        uiWrapper.DisplayTitle("Get AWS Glue job run information");
        Console.WriteLine("Getting information about the AWS Glue job.");
        var jobRuns = await wrapper.GetJobRunsAsync(jobName);

        jobRuns.ForEach(jobRun =>
        {

Console.WriteLine($"{jobRun.JobName}\t{jobRun.JobRunState}\t{jobRun.CompletedOn}");
        });

        uiWrapper.PressEnter();

        uiWrapper.DisplayTitle("Deleting resources");
        Console.WriteLine("Deleting the AWS Glue job used by the example.");
        await wrapper.DeleteJobAsync(jobName);

        Console.WriteLine("Deleting the tables from the database.");
        tables.ForEach(async table =>
        {
            await wrapper.DeleteTableAsync(dbName, table.Name);
        });
```

```
        Console.WriteLine("Deleting the database.");
        await wrapper.DeleteDatabaseAsync(dbName);

        Console.WriteLine("Deleting the AWS Glue crawler.");
        await wrapper.DeleteCrawlerAsync(crawlerName);

        Console.WriteLine("The AWS Glue scenario has completed.");
        uiWrapper.PressEnter();
    }
}


namespace GlueBasics;

public class UiWrapper
{
    public readonly string SepBar = new string('-', Console.WindowWidth);

    /// <summary>
    /// Show information about the scenario.
    /// </summary>
    public void DisplayOverview()
    {
        Console.Clear();
        DisplayTitle("Amazon Glue: get started with crawlers and jobs");

        Console.WriteLine("This example application does the following:");
        Console.WriteLine("\t 1. Create a crawler, pass it the IAM role and the
 URL to the public S3 bucket that contains the source data");
        Console.WriteLine("\t 2. Start the crawler.");
        Console.WriteLine("\t 3. Get the database created by the crawler and the
 tables in the database.");
        Console.WriteLine("\t 4. Create a job.");
        Console.WriteLine("\t 5. Start a job run.");
        Console.WriteLine("\t 6. Wait for the job run to complete.");
        Console.WriteLine("\t 7. Show the data stored in the bucket.");
        Console.WriteLine("\t 8. List jobs for the account.");
        Console.WriteLine("\t 9. Get job run details for the job that was run.");
        Console.WriteLine("\t10. Delete the demo job.");
        Console.WriteLine("\t11. Delete the database and tables created for the
 demo.");
        Console.WriteLine("\t12. Delete the crawler.");
    }
```

```
    /// <summary>
    /// Display a message and wait until the user presses enter.
    /// </summary>
    public void PressEnter()
    {
        Console.Write("\nPlease press <Enter> to continue. ");
        _ = Console.ReadLine();
    }

    /// <summary>
    /// Pad a string with spaces to center it on the console display.
    /// </summary>
    /// <param name="strToCenter">The string to center on the screen.</param>
    /// <returns>The string padded to make it center on the screen.</returns>
    public string CenterString(string strToCenter)
    {
        var padAmount = (Console.WindowWidth - strToCenter.Length) / 2;
        var leftPad = new string(' ', padAmount);
        return $"{leftPad}{strToCenter}";
    }

    /// <summary>
    /// Display a line of hyphens, the centered text of the title and another
    /// line of hyphens.
    /// </summary>
    /// <param name="strTitle">The string to be displayed.</param>
    public void DisplayTitle(string strTitle)
    {
        Console.WriteLine(SepBar);
        Console.WriteLine(CenterString(strTitle));
        Console.WriteLine(SepBar);
    }
}
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の以下のトピックを参照してください。
  - CreateCrawler
  - CreateJob
  - DeleteCrawler
  - DeleteDatabase

- DeleteJob
- DeleteTable
- GetCrawler
- GetDatabase
- GetDatabases
- GetJob
- GetJob
- GetJobRuns
- GetTables
- ListJobs
- StartCrawler
- StartJobRun

C++

SDK for C++

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
//! Scenario which demonstrates using AWS Glue to add a crawler and run a job.
/*!
 \\sa runGettingStartedWithGlueScenario()
 \param bucketName: An S3 bucket created in the setup.
 \param roleName: An AWS Identity and Access Management (IAM) role created in the
 setup.
 \param clientConfig: AWS client configuration.
 \return bool: Successful completion.
 */

bool AwsDoc::Glue::runGettingStartedWithGlueScenario(const Aws::String
 &bucketName,
                                                     const Aws::String &roleName,
```

```
                                                    const
Aws::Client::ClientConfiguration &clientConfig) {
   Aws::Glue::GlueClient client(clientConfig);

   Aws::String roleArn;
   if (!getRoleArn(roleName, roleArn, clientConfig)) {
       std::cerr << "Error getting role ARN for role." << std::endl;
       return false;
   }

   // 1. Upload the job script to the S3 bucket.
   {
       std::cout << "Uploading the job script '"
                 << AwsDoc::Glue::PYTHON_SCRIPT
                 << "'." << std::endl;

       if (!AwsDoc::Glue::uploadFile(bucketName,
                                     AwsDoc::Glue::PYTHON_SCRIPT_PATH,
                                     AwsDoc::Glue::PYTHON_SCRIPT,
                                     clientConfig)) {
           std::cerr << "Error uploading the job file." << std::endl;
           return false;
       }
   }

   // 2. Create a crawler.
   {
       Aws::Glue::Model::S3Target s3Target;
       s3Target.SetPath("s3://crawler-public-us-east-1/flight/2016/csv");
       Aws::Glue::Model::CrawlerTargets crawlerTargets;
       crawlerTargets.AddS3Targets(s3Target);

       Aws::Glue::Model::CreateCrawlerRequest request;
       request.SetTargets(crawlerTargets);
       request.SetName(CRAWLER_NAME);
       request.SetDatabaseName(CRAWLER_DATABASE_NAME);
       request.SetTablePrefix(CRAWLER_DATABASE_PREFIX);
       request.SetRole(roleArn);

       Aws::Glue::Model::CreateCrawlerOutcome outcome =
client.CreateCrawler(request);

       if (outcome.IsSuccess()) {
           std::cout << "Successfully created the crawler." << std::endl;
```

```
        }
        else {
            std::cerr << "Error creating a crawler. " <<
outcome.GetError().GetMessage()
                      << std::endl;
            deleteAssets("", CRAWLER_DATABASE_NAME, "", bucketName,
clientConfig);
            return false;
        }
    }

    // 3. Get a crawler.
    {
        Aws::Glue::Model::GetCrawlerRequest request;
        request.SetName(CRAWLER_NAME);

        Aws::Glue::Model::GetCrawlerOutcome outcome = client.GetCrawler(request);

        if (outcome.IsSuccess()) {
            Aws::Glue::Model::CrawlerState crawlerState =
outcome.GetResult().GetCrawler().GetState();
            std::cout << "Retrieved crawler with state " <<

Aws::Glue::Model::CrawlerStateMapper::GetNameForCrawlerState(
                              crawlerState)
                      << "." << std::endl;
        }
        else {
            std::cerr << "Error retrieving a crawler.  "
                      << outcome.GetError().GetMessage() << std::endl;
            deleteAssets(CRAWLER_NAME, CRAWLER_DATABASE_NAME, "", bucketName,
                         clientConfig);
            return false;
        }
    }

    // 4. Start a crawler.
    {
        Aws::Glue::Model::StartCrawlerRequest request;
        request.SetName(CRAWLER_NAME);

        Aws::Glue::Model::StartCrawlerOutcome outcome =
client.StartCrawler(request);
```

```
       if (outcome.IsSuccess() || (Aws::Glue::GlueErrors::CRAWLER_RUNNING ==
                                   outcome.GetError().GetErrorType())) {
           if (!outcome.IsSuccess()) {
               std::cout << "Crawler was already started." << std::endl;
           }
           else {
               std::cout << "Successfully started crawler." << std::endl;
           }

           std::cout << "This may take a while to run." << std::endl;

           Aws::Glue::Model::CrawlerState crawlerState =
Aws::Glue::Model::CrawlerState::NOT_SET;
           int iterations = 0;
           while (Aws::Glue::Model::CrawlerState::READY != crawlerState) {
               std::this_thread::sleep_for(std::chrono::seconds(1));
               ++iterations;
               if ((iterations % 10) == 0) { // Log status every 10 seconds.
                   std::cout << "Crawler status " <<

Aws::Glue::Model::CrawlerStateMapper::GetNameForCrawlerState(
                                       crawlerState)
                             << ". After " << iterations
                             << " seconds elapsed."
                             << std::endl;
               }
               Aws::Glue::Model::GetCrawlerRequest getCrawlerRequest;
               getCrawlerRequest.SetName(CRAWLER_NAME);

               Aws::Glue::Model::GetCrawlerOutcome getCrawlerOutcome =
client.GetCrawler(
                       getCrawlerRequest);

               if (getCrawlerOutcome.IsSuccess()) {
                   crawlerState =
getCrawlerOutcome.GetResult().GetCrawler().GetState();
               }
               else {
                   std::cerr << "Error getting crawler.  "
                             << getCrawlerOutcome.GetError().GetMessage() <<
std::endl;
                   break;
               }
```

```
            }

            if (Aws::Glue::Model::CrawlerState::READY == crawlerState) {
                std::cout << "Crawler finished running after " << iterations
                          << " seconds."
                          << std::endl;
            }
        }
        else {
            std::cerr << "Error starting a crawler.  "
                      << outcome.GetError().GetMessage()
                      << std::endl;

            deleteAssets(CRAWLER_NAME, CRAWLER_DATABASE_NAME, "", bucketName,
                         clientConfig);
            return false;
        }
    }

    // 5. Get a database.
    {
        Aws::Glue::Model::GetDatabaseRequest request;
        request.SetName(CRAWLER_DATABASE_NAME);

        Aws::Glue::Model::GetDatabaseOutcome outcome =
client.GetDatabase(request);

        if (outcome.IsSuccess()) {
            const Aws::Glue::Model::Database &database =
outcome.GetResult().GetDatabase();

            std::cout << "Successfully retrieve the database\n" <<
                      database.Jsonize().View().WriteReadable() << "'." <<
std::endl;
        }
        else {
            std::cerr << "Error getting the database.  "
                      << outcome.GetError().GetMessage() << std::endl;
            deleteAssets(CRAWLER_NAME, CRAWLER_DATABASE_NAME, "", bucketName,
                         clientConfig);
            return false;
        }
    }
```

```
   // 6. Get tables.
   Aws::String tableName;
   {
       Aws::Glue::Model::GetTablesRequest request;
       request.SetDatabaseName(CRAWLER_DATABASE_NAME);
       std::vector<Aws::Glue::Model::Table> all_tables;
       Aws::String nextToken; // Used for pagination.
       do {
           Aws::Glue::Model::GetTablesOutcome outcome =
client.GetTables(request);

           if (outcome.IsSuccess()) {
               const std::vector<Aws::Glue::Model::Table> &tables =
outcome.GetResult().GetTableList();
               all_tables.insert(all_tables.end(), tables.begin(),
tables.end());
               nextToken = outcome.GetResult().GetNextToken();
           }
           else {
               std::cerr << "Error getting the tables. "
                         << outcome.GetError().GetMessage()
                         << std::endl;
               deleteAssets(CRAWLER_NAME, CRAWLER_DATABASE_NAME, "", bucketName,
                            clientConfig);
               return false;
           }
       } while (!nextToken.empty());

       std::cout << "The database contains " << all_tables.size()
                 << (all_tables.size() == 1 ?
                     " table." : "tables.") << std::endl;
       std::cout << "Here is a list of the tables in the database.";
       for (size_t index = 0; index < all_tables.size(); ++index) {
           std::cout << "    " << index + 1 << ":  " <<
all_tables[index].GetName()
                     << std::endl;
       }

       if (!all_tables.empty()) {
           int tableIndex = askQuestionForIntRange(
                   "Enter an index to display the database detail ",
                   1, static_cast<int>(all_tables.size()));
           std::cout << all_tables[tableIndex -
1].Jsonize().View().WriteReadable()
```

```
                    << std::endl;

            tableName = all_tables[tableIndex - 1].GetName();
        }
    }

    // 7. Create a job.
    {
        Aws::Glue::Model::CreateJobRequest request;
        request.SetName(JOB_NAME);
        request.SetRole(roleArn);
        request.SetGlueVersion(GLUE_VERSION);

        Aws::Glue::Model::JobCommand command;
        command.SetName(JOB_COMMAND_NAME);
        command.SetPythonVersion(JOB_PYTHON_VERSION);
        command.SetScriptLocation(
                Aws::String("s3://") + bucketName + "/" + PYTHON_SCRIPT);
        request.SetCommand(command);

        Aws::Glue::Model::CreateJobOutcome outcome = client.CreateJob(request);

        if (outcome.IsSuccess()) {
            std::cout << "Successfully created the job." << std::endl;
        }
        else {
            std::cerr << "Error creating the job. " <<
outcome.GetError().GetMessage()
                      << std::endl;
            deleteAssets(CRAWLER_NAME, CRAWLER_DATABASE_NAME, "", bucketName,
                         clientConfig);
            return false;
        }
    }

    // 8. Start a job run.
    {
        Aws::Glue::Model::StartJobRunRequest request;
        request.SetJobName(JOB_NAME);

        Aws::Map<Aws::String, Aws::String> arguments;
        arguments["--input_database"] = CRAWLER_DATABASE_NAME;
        arguments["--input_table"] = tableName;
```

```
        arguments["--output_bucket_url"] = Aws::String("s3://") + bucketName +
"/";
        request.SetArguments(arguments);

        Aws::Glue::Model::StartJobRunOutcome outcome =
client.StartJobRun(request);

        if (outcome.IsSuccess()) {
            std::cout << "Successfully started the job." << std::endl;

            Aws::String jobRunId = outcome.GetResult().GetJobRunId();

            int iterator = 0;
            bool done = false;
            while (!done) {
                ++iterator;
                std::this_thread::sleep_for(std::chrono::seconds(1));
                Aws::Glue::Model::GetJobRunRequest jobRunRequest;
                jobRunRequest.SetJobName(JOB_NAME);
                jobRunRequest.SetRunId(jobRunId);

                Aws::Glue::Model::GetJobRunOutcome jobRunOutcome =
client.GetJobRun(
                        jobRunRequest);

                if (jobRunOutcome.IsSuccess()) {
                    const Aws::Glue::Model::JobRun &jobRun =
jobRunOutcome.GetResult().GetJobRun();
                    Aws::Glue::Model::JobRunState jobRunState =
jobRun.GetJobRunState();

                    if ((jobRunState == Aws::Glue::Model::JobRunState::STOPPED)
||
                        (jobRunState == Aws::Glue::Model::JobRunState::FAILED) ||
                        (jobRunState == Aws::Glue::Model::JobRunState::TIMEOUT))
{
                        std::cerr << "Error running job. "
                                  << jobRun.GetErrorMessage()
                                  << std::endl;
                        deleteAssets(CRAWLER_NAME, CRAWLER_DATABASE_NAME,
JOB_NAME,
                                     bucketName,
                                     clientConfig);
                        return false;
```

```
                    }
                    else if (jobRunState ==
                             Aws::Glue::Model::JobRunState::SUCCEEDED) {
                        std::cout << "Job run succeeded after  " << iterator <<
                                  " seconds elapsed." << std::endl;
                        done = true;
                    }
                    else if ((iterator % 10) == 0) { // Log status every 10
seconds.
                        std::cout << "Job run status " <<

Aws::Glue::Model::JobRunStateMapper::GetNameForJobRunState(
                                        jobRunState) <<
                                  ". " << iterator <<
                                  " seconds elapsed." << std::endl;
                    }
                }
                else {
                    std::cerr << "Error retrieving job run state. "
                              << jobRunOutcome.GetError().GetMessage()
                              << std::endl;
                    deleteAssets(CRAWLER_NAME, CRAWLER_DATABASE_NAME, JOB_NAME,
                                 bucketName, clientConfig);
                    return false;
                }
            }
        }
        else {
            std::cerr << "Error starting a job. " <<
outcome.GetError().GetMessage()
                      << std::endl;
            deleteAssets(CRAWLER_NAME, CRAWLER_DATABASE_NAME, JOB_NAME,
bucketName,
                         clientConfig);
            return false;
        }
    }

    // 9. List the output data stored in the S3 bucket.
    {
        Aws::S3::S3Client s3Client;
        Aws::S3::Model::ListObjectsV2Request request;
        request.SetBucket(bucketName);
        request.SetPrefix(OUTPUT_FILE_PREFIX);
```

```
        Aws::String continuationToken; // Used for pagination.
        std::vector<Aws::S3::Model::Object> allObjects;
        do {
            if (!continuationToken.empty()) {
                request.SetContinuationToken(continuationToken);
            }
            Aws::S3::Model::ListObjectsV2Outcome outcome =
s3Client.ListObjectsV2(
                    request);

            if (outcome.IsSuccess()) {
                const std::vector<Aws::S3::Model::Object> &objects =
                        outcome.GetResult().GetContents();
                allObjects.insert(allObjects.end(), objects.begin(),
objects.end());
                continuationToken =
outcome.GetResult().GetNextContinuationToken();
            }
            else {
                std::cerr << "Error listing objects. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                break;
            }
        } while (!continuationToken.empty());

        std::cout << "Data from your job is in " << allObjects.size() <<
                  " files in the S3 bucket, " << bucketName << "." << std::endl;

        for (size_t i = 0; i < allObjects.size(); ++i) {
            std::cout << "    " << i + 1 << ". " << allObjects[i].GetKey()
                      << std::endl;
        }

        int objectIndex = askQuestionForIntRange(
                std::string(
                        "Enter the number of a block to download it and see the
first ") +
                std::to_string(LINES_OF_RUN_FILE_TO_DISPLAY) +
                " lines of JSON output in the block: ", 1,
                static_cast<int>(allObjects.size()));

        Aws::String objectKey = allObjects[objectIndex - 1].GetKey();
```

```
        std::stringstream stringStream;
        if (getObjectFromBucket(bucketName, objectKey, stringStream,
                                clientConfig)) {
            for (int i = 0; i < LINES_OF_RUN_FILE_TO_DISPLAY && stringStream; +
+i) {
                std::string line;
                std::getline(stringStream, line);
                std::cout << "    " << line << std::endl;
            }
        }
        else {
            deleteAssets(CRAWLER_NAME, CRAWLER_DATABASE_NAME, JOB_NAME,
 bucketName,
                         clientConfig);
            return false;
        }
    }

    // 10. List all the jobs.
    Aws::String jobName;
    {
        Aws::Glue::Model::ListJobsRequest listJobsRequest;

        Aws::String nextToken;
        std::vector<Aws::String> allJobNames;

        do {
            if (!nextToken.empty()) {
                listJobsRequest.SetNextToken(nextToken);
            }
            Aws::Glue::Model::ListJobsOutcome listRunsOutcome = client.ListJobs(
                    listJobsRequest);

            if (listRunsOutcome.IsSuccess()) {
                const std::vector<Aws::String> &jobNames =
 listRunsOutcome.GetResult().GetJobNames();
                allJobNames.insert(allJobNames.end(), jobNames.begin(),
 jobNames.end());
                nextToken = listRunsOutcome.GetResult().GetNextToken();
            }
            else {
                std::cerr << "Error listing jobs. "
                          << listRunsOutcome.GetError().GetMessage()
```

```
                        << std::endl;
            }
        } while (!nextToken.empty());
        std::cout << "Your account has " << allJobNames.size() << " jobs."
                  << std::endl;
        for (size_t i = 0; i < allJobNames.size(); ++i) {
            std::cout << "   " << i + 1 << ". " << allJobNames[i] << std::endl;
        }
        int jobIndex = askQuestionForIntRange(
                Aws::String("Enter a number between 1 and ") +
                std::to_string(allJobNames.size()) +
                " to see the list of runs for a job: ",
                1, static_cast<int>(allJobNames.size()));

        jobName = allJobNames[jobIndex - 1];
    }

    // 11. Get the job runs for a job.
    Aws::String jobRunID;
    if (!jobName.empty()) {
        Aws::Glue::Model::GetJobRunsRequest getJobRunsRequest;
        getJobRunsRequest.SetJobName(jobName);

        Aws::String nextToken; // Used for pagination.
        std::vector<Aws::Glue::Model::JobRun> allJobRuns;
        do {
            if (!nextToken.empty()) {
                getJobRunsRequest.SetNextToken(nextToken);
            }
            Aws::Glue::Model::GetJobRunsOutcome jobRunsOutcome =
client.GetJobRuns(
                    getJobRunsRequest);

            if (jobRunsOutcome.IsSuccess()) {
                const std::vector<Aws::Glue::Model::JobRun> &jobRuns =
jobRunsOutcome.GetResult().GetJobRuns();
                allJobRuns.insert(allJobRuns.end(), jobRuns.begin(),
jobRuns.end());

                nextToken = jobRunsOutcome.GetResult().GetNextToken();
            }
            else {
                std::cerr << "Error getting job runs. "
                          << jobRunsOutcome.GetError().GetMessage()
```

```
                        << std::endl;
                break;
            }
        } while (!nextToken.empty());

        std::cout << "There are " << allJobRuns.size() << " runs in the job '"
                  <<
                  jobName << "'." << std::endl;

        for (size_t i = 0; i < allJobRuns.size(); ++i) {
            std::cout << "   " << i + 1 << ". " << allJobRuns[i].GetJobName()
                      << std::endl;
        }

        int runIndex = askQuestionForIntRange(
                Aws::String("Enter a number between 1 and ") +
                std::to_string(allJobRuns.size()) +
                " to see details for a run: ",
                1, static_cast<int>(allJobRuns.size()));
        jobRunID = allJobRuns[runIndex - 1].GetId();
    }

    // 12. Get a single job run.
    if (!jobRunID.empty()) {
        Aws::Glue::Model::GetJobRunRequest jobRunRequest;
        jobRunRequest.SetJobName(jobName);
        jobRunRequest.SetRunId(jobRunID);

        Aws::Glue::Model::GetJobRunOutcome jobRunOutcome = client.GetJobRun(
                jobRunRequest);

        if (jobRunOutcome.IsSuccess()) {
            std::cout << "Displaying the job run JSON description." << std::endl;
            std::cout
                    <<
jobRunOutcome.GetResult().GetJobRun().Jsonize().View().WriteReadable()
                    << std::endl;
        }
        else {
            std::cerr << "Error get a job run. "
                      << jobRunOutcome.GetError().GetMessage()
                      << std::endl;
        }
    }
```

```
    return deleteAssets(CRAWLER_NAME, CRAWLER_DATABASE_NAME, JOB_NAME,
 bucketName,
                        clientConfig);
}

//! Cleanup routine to delete created assets.
/*!
 \\sa deleteAssets()
 \param crawler: Name of an AWS Glue crawler.
 \param database: The name of an AWS Glue database.
 \param job: The name of an AWS Glue job.
 \param bucketName: The name of an S3 bucket.
 \param clientConfig: AWS client configuration.
 \return bool: Successful completion.
 */
bool AwsDoc::Glue::deleteAssets(const Aws::String &crawler, const Aws::String
 &database,
                                const Aws::String &job, const Aws::String
 &bucketName,
                                const Aws::Client::ClientConfiguration
 &clientConfig) {
    const Aws::Glue::GlueClient client(clientConfig);
    bool result = true;

    // 13. Delete a job.
    if (!job.empty()) {
        Aws::Glue::Model::DeleteJobRequest request;
        request.SetJobName(job);

        Aws::Glue::Model::DeleteJobOutcome outcome = client.DeleteJob(request);


        if (outcome.IsSuccess()) {
            std::cout << "Successfully deleted the job." << std::endl;
        }
        else {
            std::cerr << "Error deleting the job. " <<
 outcome.GetError().GetMessage()
                      << std::endl;
            result = false;
        }
    }
```

```
    // 14. Delete a database.
    if (!database.empty()) {
        Aws::Glue::Model::DeleteDatabaseRequest request;
        request.SetName(database);

        Aws::Glue::Model::DeleteDatabaseOutcome outcome = client.DeleteDatabase(
                request);

        if (outcome.IsSuccess()) {
            std::cout << "Successfully deleted the database." << std::endl;
        }
        else {
            std::cerr << "Error deleting database. " <<
 outcome.GetError().GetMessage()
                      << std::endl;
            result = false;
        }
    }

    // 15. Delete a crawler.
    if (!crawler.empty()) {
        Aws::Glue::Model::DeleteCrawlerRequest request;
        request.SetName(crawler);

        Aws::Glue::Model::DeleteCrawlerOutcome outcome =
 client.DeleteCrawler(request);

        if (outcome.IsSuccess()) {
            std::cout << "Successfully deleted the crawler." << std::endl;
        }
        else {
            std::cerr << "Error deleting the crawler. "
                      << outcome.GetError().GetMessage() << std::endl;
            result = false;
        }
    }

    // 16. Delete the job script and run data from the S3 bucket.
    result &= AwsDoc::Glue::deleteAllObjectsInS3Bucket(bucketName,
                                                       clientConfig);
    return result;
}

//! Routine which uploads a file to an S3 bucket.
```

```
/*!
 \\sa uploadFile()
 \param bucketName: An S3 bucket created in the setup.
 \param filePath: The path of the file to upload.
 \param fileName The name for the uploaded file.
 \param clientConfig: AWS client configuration.
 \return bool: Successful completion.
 */
bool
AwsDoc::Glue::uploadFile(const Aws::String &bucketName,
                         const Aws::String &filePath,
                         const Aws::String &fileName,
                         const Aws::Client::ClientConfiguration &clientConfig) {
    Aws::S3::S3Client s3_client(clientConfig);

    Aws::S3::Model::PutObjectRequest request;
    request.SetBucket(bucketName);
    request.SetKey(fileName);

    std::shared_ptr<Aws::IOStream> inputData =
            Aws::MakeShared<Aws::FStream>("SampleAllocationTag",
                                          filePath.c_str(),
                                          std::ios_base::in |
 std::ios_base::binary);

    if (!*inputData) {
        std::cerr << "Error unable to read file " << filePath << std::endl;
        return false;
    }

    request.SetBody(inputData);

    Aws::S3::Model::PutObjectOutcome outcome =
            s3_client.PutObject(request);

    if (!outcome.IsSuccess()) {
        std::cerr << "Error: PutObject: " <<
                  outcome.GetError().GetMessage() << std::endl;
    }
    else {
        std::cout << "Added object '" << filePath << "' to bucket '"
                  << bucketName << "'." << std::endl;
    }
```

```
    return outcome.IsSuccess();
}

//! Routine which deletes all objects in an S3 bucket.
/*!
 \\sa deleteAllObjectsInS3Bucket()
 \param bucketName: The S3 bucket name.
 \param clientConfig: AWS client configuration.
 \return bool: Successful completion.
 */
bool AwsDoc::Glue::deleteAllObjectsInS3Bucket(const Aws::String &bucketName,
                                              const
 Aws::Client::ClientConfiguration &clientConfig) {
    Aws::S3::S3Client client(clientConfig);
    Aws::S3::Model::ListObjectsV2Request listObjectsRequest;
    listObjectsRequest.SetBucket(bucketName);

    Aws::String continuationToken; // Used for pagination.
    bool result = true;
    do {
        if (!continuationToken.empty()) {
            listObjectsRequest.SetContinuationToken(continuationToken);
        }

        Aws::S3::Model::ListObjectsV2Outcome listObjectsOutcome =
 client.ListObjectsV2(
                listObjectsRequest);

        if (listObjectsOutcome.IsSuccess()) {
            const std::vector<Aws::S3::Model::Object> &objects =
 listObjectsOutcome.GetResult().GetContents();
            if (!objects.empty()) {
                Aws::S3::Model::DeleteObjectsRequest deleteObjectsRequest;
                deleteObjectsRequest.SetBucket(bucketName);

                std::vector<Aws::S3::Model::ObjectIdentifier> objectIdentifiers;
                for (const Aws::S3::Model::Object &object: objects) {
                    objectIdentifiers.push_back(
                            Aws::S3::Model::ObjectIdentifier().WithKey(
                                    object.GetKey()));
                }
                Aws::S3::Model::Delete objectsDelete;
                objectsDelete.SetObjects(objectIdentifiers);
                objectsDelete.SetQuiet(true);
```

```
                deleteObjectsRequest.SetDelete(objectsDelete);

                Aws::S3::Model::DeleteObjectsOutcome deleteObjectsOutcome =
                        client.DeleteObjects(deleteObjectsRequest);

                if (!deleteObjectsOutcome.IsSuccess()) {
                    std::cerr << "Error deleting objects. " <<
                              deleteObjectsOutcome.GetError().GetMessage() <<
 std::endl;
                    result = false;
                    break;
                }
                else {
                    std::cout << "Successfully deleted the objects." <<
 std::endl;

                }
            }
            else {
                std::cout << "No objects to delete in '" << bucketName << "'."
                          << std::endl;
            }

            continuationToken =
 listObjectsOutcome.GetResult().GetNextContinuationToken();
        }
        else {
            std::cerr << "Error listing objects. "
                      << listObjectsOutcome.GetError().GetMessage() << std::endl;
            result = false;
            break;
        }
    } while (!continuationToken.empty());

    return result;
}

//! Routine which retrieves an object from an S3 bucket.
/*!
 \\sa getObjectFromBucket()
 \param bucketName: The S3 bucket name.
 \param objectKey: The object's name.
 \param objectStream: A stream to receive the retrieved data.
 \param clientConfig: AWS client configuration.
```

```
 \return bool: Successful completion.
 */
bool AwsDoc::Glue::getObjectFromBucket(const Aws::String &bucketName,
                                       const Aws::String &objectKey,
                                       std::ostream &objectStream,
                                       const Aws::Client::ClientConfiguration
 &clientConfig) {
    Aws::S3::S3Client client(clientConfig);
    Aws::S3::Model::GetObjectRequest request;
    request.SetBucket(bucketName);
    request.SetKey(objectKey);

    Aws::S3::Model::GetObjectOutcome outcome = client.GetObject(request);


    if (outcome.IsSuccess()) {
        std::cout << "Successfully retrieved '" << objectKey << "'." <<
 std::endl;
        auto &body = outcome.GetResult().GetBody();
        objectStream << body.rdbuf();
    }
    else {
        std::cerr << "Error retrieving object. " <<
 outcome.GetError().GetMessage()
                  << std::endl;
    }

    return outcome.IsSuccess();
}
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の以下のトピックを参照してください。
  - CreateCrawler
  - CreateJob
  - DeleteCrawler
  - DeleteDatabase
  - DeleteJob
  - DeleteTable

- GetCrawler
- GetDatabase
- GetDatabases
- GetJob
- GetJobRun
- GetJobRuns
- GetTables
- ListJobs
- StartCrawler
- StartJobRun

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 * <p>
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * To set up the resources, see this documentation topic:
 *
 * https://docs.aws.amazon.com/glue/latest/ug/tutorial-add-crawler.html
 *
 * This example performs the following tasks:
 *
```

```
 * 1. Create a database.
 * 2. Create a crawler.
 * 3. Get a crawler.
 * 4. Start a crawler.
 * 5. Get a database.
 * 6. Get tables.
 * 7. Create a job.
 * 8. Start a job run.
 * 9. List all jobs.
 * 10. Get job runs.
 * 11. Delete a job.
 * 12. Delete a database.
 * 13. Delete a crawler.
 */

public class GlueScenario {
    public static final String DASHES = new String(new char[80]).replace("\0",
 "-");

    public static void main(String[] args) throws InterruptedException {
        final String usage = """

            Usage:
                <iam> <s3Path> <cron> <dbName> <crawlerName> <jobName>
 <scriptLocation> <locationUri> <bucketNameSc>\s

            Where:
                iam - The ARN of the IAM role that has AWS Glue and S3
 permissions.\s
                s3Path - The Amazon Simple Storage Service (Amazon S3) target
 that contains data (for example, s3://<bucket name>/read).
                cron - A cron expression used to specify the schedule  (i.e.,
 cron(15 12 * * ? *).
                dbName - The database name.\s
                crawlerName - The name of the crawler.\s
                jobName - The name you assign to this job definition.
                scriptLocation - The Amazon S3 path to a script that runs a job.
                locationUri - The location of the database (you can find this
 file in resources folder).
                bucketNameSc - The Amazon S3 bucket name used when creating a job
                """;

        if (args.length != 9) {
            System.out.println(usage);
```

```
            return;
        }
        Scanner scanner = new Scanner(System.in);
        String iam = args[0];
        String s3Path = args[1];
        String cron = args[2];
        String dbName = args[3];
        String crawlerName = args[4];
        String jobName = args[5];
        String scriptLocation = args[6];
        String locationUri = args[7];
        String bucketNameSc = args[8];

        Region region = Region.US_EAST_1;
        GlueClient glueClient = GlueClient.builder()
            .region(region)
            .build();
        System.out.println(DASHES);
        System.out.println("Welcome to the AWS Glue scenario.");
        System.out.println("""
            AWS Glue is a fully managed extract, transform, and load (ETL)
service provided by Amazon
            Web Services (AWS). It is designed to simplify the process of
building, running, and maintaining
            ETL pipelines, which are essential for data integration and data
warehousing tasks.

            One of the key features of AWS Glue is its ability to automatically
discover and catalog data
            stored in various sources, such as Amazon S3, Amazon RDS, Amazon
Redshift, and other databases.
            This cataloging process creates a central metadata repository, known
as the AWS Glue Data Catalog,
            which provides a unified view of an organization's data assets. This
metadata can then be used to
            create ETL jobs, which can be scheduled and run on-demand or on a
regular basis.

            Lets get started.

            """);
        waitForInputToContinue(scanner);
        System.out.println(DASHES);
```

```
        System.out.println(DASHES);
        System.out.println("1. Create a database.");
        try {
            createDatabase(glueClient, dbName, locationUri);
        } catch (GlueException e) {
            if (e.awsErrorDetails().errorMessage().equals("Database already
exists.")) {
                System.out.println("Database " + dbName + " already exists.
Skipping creation.");
            } else {
                System.err.println(e.awsErrorDetails().errorMessage());
                return;
            }
        }

        waitForInputToContinue(scanner);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("2. Create a crawler.");
        try {
            createGlueCrawler(glueClient, iam, s3Path, cron, dbName,
crawlerName);
        } catch (GlueException e) {
            if (e.awsErrorDetails().errorMessage().contains("already exists")) {
                System.out.println("Crawler " + crawlerName + " already exists.
Skipping creation.");
            } else {
                System.err.println(e.awsErrorDetails().errorMessage());
                System.exit(1);
            }
        }
        waitForInputToContinue(scanner);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("3. Get a crawler.");
        try {
            getSpecificCrawler(glueClient, crawlerName);
        } catch (GlueException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            return;
        }
        waitForInputToContinue(scanner);
```

```
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("4. Start a crawler.");
        try {
            startSpecificCrawler(glueClient, crawlerName);
        } catch (GlueException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            return;
        }
        waitForInputToContinue(scanner);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("5. Get a database.");
        try {
            getSpecificDatabase(glueClient, dbName);
        } catch (GlueException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            return;
        }
        waitForInputToContinue(scanner);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("*** Wait 5 min for the tables to become available");
        TimeUnit.MINUTES.sleep(5);
        System.out.println("6. Get tables.");
        String myTableName;
        try {
            myTableName = getGlueTables(glueClient, dbName);
        } catch (GlueException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            return;
        }
        waitForInputToContinue(scanner);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("7. Create a job.");
        try {
            createJob(glueClient, jobName, iam, scriptLocation);
        } catch (GlueException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
```

```
            return;
        }
        waitForInputToContinue(scanner);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("8. Start a Job run.");
        try {
            startJob(glueClient, jobName, dbName, myTableName, bucketNameSc);
        } catch (GlueException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            return;
        }
        waitForInputToContinue(scanner);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("9. List all jobs.");
        try {
            getAllJobs(glueClient);
        } catch (GlueException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            return;
        }
        waitForInputToContinue(scanner);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("10. Get job runs.");
        try {
            getJobRuns(glueClient, jobName);
        } catch (GlueException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            return;
        }
        waitForInputToContinue(scanner);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("11. Delete a job.");
        try {
            deleteJob(glueClient, jobName);
        } catch (GlueException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
```

```
            return;
        }
        System.out.println("*** Wait 5 MIN for the " + crawlerName + " to stop");
        TimeUnit.MINUTES.sleep(5);
        waitForInputToContinue(scanner);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("12. Delete a database.");
        try {
            deleteDatabase(glueClient, dbName);
        } catch (GlueException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            return;
        }
        waitForInputToContinue(scanner);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("Delete a crawler.");
        try {
            deleteSpecificCrawler(glueClient, crawlerName);
        } catch (GlueException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            return;
        }
        waitForInputToContinue(scanner);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("Successfully completed the AWS Glue Scenario");
        System.out.println(DASHES);
    }


    /**
     * Creates a Glue database with the specified name and location URI.
     *
     * @param glueClient  The Glue client to use for the database creation.
     * @param dbName      The name of the database to create.
     * @param locationUri The location URI for the database.
     */
    public static void createDatabase(GlueClient glueClient, String dbName,
String locationUri) {
```

```
        try {
            DatabaseInput input = DatabaseInput.builder()
                .description("Built with the AWS SDK for Java V2")
                .name(dbName)
                .locationUri(locationUri)
                .build();

            CreateDatabaseRequest request = CreateDatabaseRequest.builder()
                .databaseInput(input)
                .build();

            glueClient.createDatabase(request);
            System.out.println(dbName + " was successfully created");

        } catch (GlueException e) {
            throw e;
        }
    }


    /**
     * Creates a new AWS Glue crawler using the AWS Glue Java API.
     *
     * @param glueClient  the AWS Glue client used to interact with the AWS Glue
service
     * @param iam         the IAM role that the crawler will use to access the
data source
     * @param s3Path      the S3 path that the crawler will scan for data
     * @param cron        the cron expression that defines the crawler's schedule
     * @param dbName      the name of the AWS Glue database where the crawler
will store the metadata
     * @param crawlerName the name of the crawler to be created
     */
    public static void createGlueCrawler(GlueClient glueClient,
                                          String iam,
                                          String s3Path,
                                          String cron,
                                          String dbName,
                                          String crawlerName) {

        try {
            S3Target s3Target = S3Target.builder()
                .path(s3Path)
                .build();
```

```
            List<S3Target> targetList = new ArrayList<>();
            targetList.add(s3Target);
            CrawlerTargets targets = CrawlerTargets.builder()
                .s3Targets(targetList)
                .build();

            CreateCrawlerRequest crawlerRequest = CreateCrawlerRequest.builder()
                .databaseName(dbName)
                .name(crawlerName)
                .description("Created by the AWS Glue Java API")
                .targets(targets)
                .role(iam)
                .schedule(cron)
                .build();

            glueClient.createCrawler(crawlerRequest);
            System.out.println(crawlerName + " was successfully created");

        } catch (GlueException e) {
            throw e;
        }
    }

    /**
     * Retrieves a specific crawler from the AWS Glue service and waits for it to
be in the "READY" state.
     *
     * @param glueClient  the AWS Glue client used to interact with the Glue
service
     * @param crawlerName the name of the crawler to be retrieved
     */
    public static void getSpecificCrawler(GlueClient glueClient, String
crawlerName) throws InterruptedException {
        try {
            GetCrawlerRequest crawlerRequest = GetCrawlerRequest.builder()
                .name(crawlerName)
                .build();

            boolean ready = false;
            while (!ready) {
                GetCrawlerResponse response =
glueClient.getCrawler(crawlerRequest);
                String status = response.crawler().stateAsString();
```

```
                if (status.compareTo("READY") == 0) {
                    ready = true;
                }
                Thread.sleep(3000);
            }

            System.out.println("The crawler is now ready");

        } catch (GlueException | InterruptedException e) {
            throw e;
        }
    }

    /**
     * Starts a specific AWS Glue crawler.
     *
     * @param glueClient  the AWS Glue client to use for the crawler operation
     * @param crawlerName the name of the crawler to start
     * @throws GlueException if there is an error starting the crawler
     */
    public static void startSpecificCrawler(GlueClient glueClient, String
crawlerName) {
        try {
            StartCrawlerRequest crawlerRequest = StartCrawlerRequest.builder()
                .name(crawlerName)
                .build();

            glueClient.startCrawler(crawlerRequest);
            System.out.println(crawlerName + " was successfully started!");

        } catch (GlueException e) {
            throw e;
        }
    }

    /**
     * Retrieves the specific database from the AWS Glue service.
     *
     * @param glueClient   an instance of the AWS Glue client used to interact
with the service
     * @param databaseName the name of the database to retrieve
     * @throws GlueException if there is an error retrieving the database from
the AWS Glue service
     */
```

```
   public static void getSpecificDatabase(GlueClient glueClient, String
databaseName) {
       try {
           GetDatabaseRequest databasesRequest = GetDatabaseRequest.builder()
               .name(databaseName)
               .build();

           GetDatabaseResponse response =
glueClient.getDatabase(databasesRequest);
           Instant createDate = response.database().createTime();

           // Convert the Instant to readable date.
           DateTimeFormatter formatter =
DateTimeFormatter.ofLocalizedDateTime(FormatStyle.SHORT)
               .withLocale(Locale.US)
               .withZone(ZoneId.systemDefault());

           formatter.format(createDate);
           System.out.println("The create date of the database is " +
createDate);

       } catch (GlueException e) {
           throw e;
       }
   }


   /**
    * Retrieves the names of the tables in the specified Glue database.
    *
    * @param glueClient the Glue client to use for the operation
    * @param dbName     the name of the Glue database to retrieve the table
names from
    * @return the name of the first table retrieved, or an empty string if no
tables were found
    */
   public static String getGlueTables(GlueClient glueClient, String dbName) {
       String myTableName = "";
       try {
           GetTablesRequest tableRequest = GetTablesRequest.builder()
               .databaseName(dbName)
               .build();

           GetTablesResponse response = glueClient.getTables(tableRequest);
```

```
            List<Table> tables = response.tableList();
            if (tables.isEmpty()) {
                System.out.println("No tables were returned");
            } else {
                for (Table table : tables) {
                    myTableName = table.name();
                    System.out.println("Table name is: " + myTableName);
                }
            }

        } catch (GlueException e) {
            throw e;
        }
        return myTableName;
    }


    /**
     * Starts a job run in AWS Glue.
     *
     * @param glueClient    the AWS Glue client to use for the job run
     * @param jobName       the name of the Glue job to run
     * @param inputDatabase the name of the input database
     * @param inputTable    the name of the input table
     * @param outBucket     the URL of the output S3 bucket
     * @throws GlueException if there is an error starting the job run
     */
    public static void startJob(GlueClient glueClient, String jobName, String
inputDatabase, String inputTable,
                                String outBucket) {
        try {
            Map<String, String> myMap = new HashMap<>();
            myMap.put("--input_database", inputDatabase);
            myMap.put("--input_table", inputTable);
            myMap.put("--output_bucket_url", outBucket);

            StartJobRunRequest runRequest = StartJobRunRequest.builder()
                .workerType(WorkerType.G_1_X)
                .numberOfWorkers(10)
                .arguments(myMap)
                .jobName(jobName)
                .build();

            StartJobRunResponse response = glueClient.startJobRun(runRequest);
```

```
            System.out.println("The request Id of the job is " +
response.responseMetadata().requestId());

        } catch (GlueException e) {
            throw e;
        }
    }


    /**
     * Creates a new AWS Glue job.
     *
     * @param glueClient     the AWS Glue client to use for the operation
     * @param jobName        the name of the job to create
     * @param iam            the IAM role to associate with the job
     * @param scriptLocation the location of the script to be used by the job
     * @throws GlueException if there is an error creating the job
     */
    public static void createJob(GlueClient glueClient, String jobName, String
iam, String scriptLocation) {
        try {
            JobCommand command = JobCommand.builder()
                .pythonVersion("3")
                .name("glueetl")
                .scriptLocation(scriptLocation)
                .build();

            CreateJobRequest jobRequest = CreateJobRequest.builder()
                .description("A Job created by using the AWS SDK for Java V2")
                .glueVersion("2.0")
                .workerType(WorkerType.G_1_X)
                .numberOfWorkers(10)
                .name(jobName)
                .role(iam)
                .command(command)
                .build();

            glueClient.createJob(jobRequest);
            System.out.println(jobName + " was successfully created.");

        } catch (GlueException e) {
            throw e;
        }
    }
```

```
   /**
    * Retrieves and prints information about all the jobs in the Glue data
catalog.
    *
    * @param glueClient the Glue client used to interact with the AWS Glue
service
    */
   public static void getAllJobs(GlueClient glueClient) {
       try {
           GetJobsRequest jobsRequest = GetJobsRequest.builder()
               .maxResults(10)
               .build();

           GetJobsResponse jobsResponse = glueClient.getJobs(jobsRequest);
           List<Job> jobs = jobsResponse.jobs();
           for (Job job : jobs) {
               System.out.println("Job name is : " + job.name());
               System.out.println("The job worker type is : " +
job.workerType().name());
           }

       } catch (GlueException e) {
           throw e;
       }
   }

   /**
    * Retrieves the job runs for a given Glue job and prints the status of the
job runs.
    *
    * @param glueClient the Glue client used to make API calls
    * @param jobName    the name of the Glue job to retrieve the job runs for
    */
   public static void getJobRuns(GlueClient glueClient, String jobName) {
       try {
           GetJobRunsRequest runsRequest = GetJobRunsRequest.builder()
               .jobName(jobName)
               .maxResults(20)
               .build();

           boolean jobDone = false;
           while (!jobDone) {
```

```
                GetJobRunsResponse response = glueClient.getJobRuns(runsRequest);
                List<JobRun> jobRuns = response.jobRuns();
                for (JobRun jobRun : jobRuns) {
                    String jobState = jobRun.jobRunState().name();
                    if (jobState.compareTo("SUCCEEDED") == 0) {
                        System.out.println(jobName + " has succeeded");
                        jobDone = true;

                    } else if (jobState.compareTo("STOPPED") == 0) {
                        System.out.println("Job run has stopped");
                        jobDone = true;

                    } else if (jobState.compareTo("FAILED") == 0) {
                        System.out.println("Job run has failed");
                        jobDone = true;

                    } else if (jobState.compareTo("TIMEOUT") == 0) {
                        System.out.println("Job run has timed out");
                        jobDone = true;

                    } else {
                        System.out.println("*** Job run state is " +
jobRun.jobRunState().name());
                        System.out.println("Job run Id is " + jobRun.id());
                        System.out.println("The Glue version is " +
jobRun.glueVersion());
                    }
                    TimeUnit.SECONDS.sleep(5);
                }
            }

        } catch (GlueException e) {
            throw e;
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }
    }


    /**
     * Deletes a Glue job.
     *
     * @param glueClient the Glue client to use for the operation
     * @param jobName    the name of the job to be deleted
```

```
     * @throws GlueException if there is an error deleting the job
     */
    public static void deleteJob(GlueClient glueClient, String jobName) {
        try {
            DeleteJobRequest jobRequest = DeleteJobRequest.builder()
                .jobName(jobName)
                .build();

            glueClient.deleteJob(jobRequest);
            System.out.println(jobName + " was successfully deleted");

        } catch (GlueException e) {
            throw e;
        }
    }

    /**
     * Deletes a AWS Glue Database.
     *
     * @param glueClient   An instance of the AWS Glue client used to interact
with the AWS Glue service.
     * @param databaseName The name of the database to be deleted.
     * @throws GlueException If an error occurs while deleting the database.
     */
    public static void deleteDatabase(GlueClient glueClient, String databaseName)
{
        try {
            DeleteDatabaseRequest request = DeleteDatabaseRequest.builder()
                .name(databaseName)
                .build();

            glueClient.deleteDatabase(request);
            System.out.println(databaseName + " was successfully deleted");

        } catch (GlueException e) {
            throw e;
        }
    }


    /**
     * Deletes a specific AWS Glue crawler.
     *
     * @param glueClient  the AWS Glue client object
```

```
     * @param crawlerName the name of the crawler to be deleted
     * @throws GlueException if an error occurs during the deletion process
     */
    public static void deleteSpecificCrawler(GlueClient glueClient, String
 crawlerName) {
        try {
            DeleteCrawlerRequest deleteCrawlerRequest =
 DeleteCrawlerRequest.builder()
                .name(crawlerName)
                .build();

            glueClient.deleteCrawler(deleteCrawlerRequest);
            System.out.println(crawlerName + " was deleted");

        } catch (GlueException e) {
            throw e;
        }
    }

    private static void waitForInputToContinue(Scanner scanner) {
        while (true) {
            System.out.println("");
            System.out.println("Enter 'c' followed by <ENTER> to continue:");
            String input = scanner.nextLine();

            if (input.trim().equalsIgnoreCase("c")) {
                System.out.println("Continuing with the program...");
                System.out.println("");
                break;
            } else {
                // Handle invalid input.
                System.out.println("Invalid input. Please try again.");
            }
        }
    }
}
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の以下のトピックを参照してください。
  - CreateCrawler
  - CreateJob

- DeleteCrawler
- DeleteDatabase
- DeleteJob
- DeleteTable
- GetCrawler
- GetDatabase
- GetDatabases
- GetJob
- GetJobRun
- GetJobRuns
- GetTables
- ListJobs
- StartCrawler
- StartJobRun

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

パブリックの Amazon Simple Storage Service (Amazon S3) バケットをクロールし、検出した CSV 形式データを記述するメタデータデータベースを生成するクローラーを作成して実行します。

```
const createCrawler = (name, role, dbName, tablePrefix, s3TargetPath) => {
  const client = new GlueClient({});

  const command = new CreateCrawlerCommand({
    Name: name,
    Role: role,
    DatabaseName: dbName,
```

```
    TablePrefix: tablePrefix,
    Targets: {
      S3Targets: [{ Path: s3TargetPath }],
    },
  });

  return client.send(command);
};

const getCrawler = (name) => {
  const client = new GlueClient({});

  const command = new GetCrawlerCommand({
    Name: name,
  });

  return client.send(command);
};

const startCrawler = (name) => {
  const client = new GlueClient({});

  const command = new StartCrawlerCommand({
    Name: name,
  });

  return client.send(command);
};

const crawlerExists = async ({ getCrawler }, crawlerName) => {
  try {
    await getCrawler(crawlerName);
    return true;
  } catch {
    return false;
  }
};

/**
 * @param {{ createCrawler: import('../../../actions/create-
crawler.js').createCrawler}} actions
 */
const makeCreateCrawlerStep = (actions) => async (context) => {
  if (await crawlerExists(actions, process.env.CRAWLER_NAME)) {
```

```
    log("Crawler already exists. Skipping creation.");
  } else {
    await actions.createCrawler(
      process.env.CRAWLER_NAME,
      process.env.ROLE_NAME,
      process.env.DATABASE_NAME,
      process.env.TABLE_PREFIX,
      process.env.S3_TARGET_PATH,
    );

    log("Crawler created successfully.", { type: "success" });
  }

  return { ...context };
};

/**
 * @param {(name: string) => Promise<import('@aws-sdk/client-
glue').GetCrawlerCommandOutput>} getCrawler
 * @param {string} crawlerName
 */
const waitForCrawler = async (getCrawler, crawlerName) => {
  const waitTimeInSeconds = 30;
  const { Crawler } = await getCrawler(crawlerName);

  if (!Crawler) {
    throw new Error(`Crawler with name ${crawlerName} not found.`);
  }

  if (Crawler.State === "READY") {
    return;
  }

  log(`Crawler is ${Crawler.State}. Waiting ${waitTimeInSeconds} seconds...`);
  await wait(waitTimeInSeconds);
  return waitForCrawler(getCrawler, crawlerName);
};

const makeStartCrawlerStep =
  ({ startCrawler, getCrawler }) =>
  async (context) => {
    log("Starting crawler.");
    await startCrawler(process.env.CRAWLER_NAME);
    log("Crawler started.", { type: "success" });
```

```
    log("Waiting for crawler to finish running. This can take a while.");
    await waitForCrawler(getCrawler, process.env.CRAWLER_NAME);
    log("Crawler ready.", { type: "success" });

    return { ...context };
  };
```

AWS Glue Data Catalog 内のデータベースとテーブルに関する情報を一覧表示します。

```
const getDatabase = (name) => {
  const client = new GlueClient({});

  const command = new GetDatabaseCommand({
    Name: name,
  });

  return client.send(command);
};

const getTables = (databaseName) => {
  const client = new GlueClient({});

  const command = new GetTablesCommand({
    DatabaseName: databaseName,
  });

  return client.send(command);
};

const makeGetDatabaseStep =
  ({ getDatabase }) =>
  async (context) => {
    const {
      Database: { Name },
    } = await getDatabase(process.env.DATABASE_NAME);
    log(`Database: ${Name}`);
    return { ...context };
  };

/**
```

```
 * @param {{ getTables: () => Promise<import('@aws-sdk/client-
glue').GetTablesCommandOutput}} config
 */
const makeGetTablesStep =
  ({ getTables }) =>
  async (context) => {
    const { TableList } = await getTables(process.env.DATABASE_NAME);
    log("Tables:");
    log(TableList.map((table) => `  • ${table.Name}\n`));
    return { ...context };
  };
```

ソース Amazon S3 バケットから CSV 形式データを抽出し、フィールドを削除して名前を変更することで変換し、JSON 形式の出力を別の Amazon S3 バケットにロードするジョブを作成して実行します。

```
const createJob = (name, role, scriptBucketName, scriptKey) => {
  const client = new GlueClient({});

  const command = new CreateJobCommand({
    Name: name,
    Role: role,
    Command: {
      Name: "glueetl",
      PythonVersion: "3",
      ScriptLocation: `s3://${scriptBucketName}/${scriptKey}`,
    },
    GlueVersion: "3.0",
  });

  return client.send(command);
};

const startJobRun = (jobName, dbName, tableName, bucketName) => {
  const client = new GlueClient({});

  const command = new StartJobRunCommand({
    JobName: jobName,
    Arguments: {
      "--input_database": dbName,
      "--input_table": tableName,
      "--output_bucket_url": `s3://${bucketName}/`,
```

```
    },
  });

  return client.send(command);
};

const makeCreateJobStep =
  ({ createJob }) =>
  async (context) => {
    log("Creating Job.");
    await createJob(
      process.env.JOB_NAME,
      process.env.ROLE_NAME,
      process.env.BUCKET_NAME,
      process.env.PYTHON_SCRIPT_KEY,
    );
    log("Job created.", { type: "success" });

    return { ...context };
  };

/**
 * @param {(name: string, runId: string) => Promise<import('@aws-sdk/client-
glue').GetJobRunCommandOutput> }  getJobRun
 * @param {string} jobName
 * @param {string} jobRunId
 */
const waitForJobRun = async (getJobRun, jobName, jobRunId) => {
  const waitTimeInSeconds = 30;
  const { JobRun } = await getJobRun(jobName, jobRunId);

  if (!JobRun) {
    throw new Error(`Job run with id ${jobRunId} not found.`);
  }

  switch (JobRun.JobRunState) {
    case "FAILED":
    case "TIMEOUT":
    case "STOPPED":
    case "ERROR":
      throw new Error(
        `Job ${JobRun.JobRunState}. Error: ${JobRun.ErrorMessage}`,
      );
    case "SUCCEEDED":
```

```
      return;
    default:
      break;
  }

  log(
    `Job ${JobRun.JobRunState}. Waiting ${waitTimeInSeconds} more seconds...`,
  );
  await wait(waitTimeInSeconds);
  return waitForJobRun(getJobRun, jobName, jobRunId);
};

/**
 * @param {{ prompter: { prompt: () => Promise<{ shouldOpen: boolean }>} }}
 context
 */
const promptToOpen = async (context) => {
  const { shouldOpen } = await context.prompter.prompt({
    name: "shouldOpen",
    type: "confirm",
    message: "Open the output bucket in your browser?",
  });

  if (shouldOpen) {
    return open(
      `https://s3.console.aws.amazon.com/s3/buckets/${process.env.BUCKET_NAME} to
 view the output.`,
    );
  }
};

const makeStartJobRunStep =
  ({ startJobRun, getJobRun }) =>
  async (context) => {
    log("Starting job.");
    const { JobRunId } = await startJobRun(
      process.env.JOB_NAME,
      process.env.DATABASE_NAME,
      process.env.TABLE_NAME,
      process.env.BUCKET_NAME,
    );
    log("Job started.", { type: "success" });

    log("Waiting for job to finish running. This can take a while.");
```

```
    await waitForJobRun(getJobRun, process.env.JOB_NAME, JobRunId);
    log("Job run succeeded.", { type: "success" });

    await promptToOpen(context);

    return { ...context };
  };
```

ジョブ実行に関する情報を一覧表示し、変換されたデータの一部を表示します。

```
const getJobRuns = (jobName) => {
  const client = new GlueClient({});
  const command = new GetJobRunsCommand({
    JobName: jobName,
  });

  return client.send(command);
};

const getJobRun = (jobName, jobRunId) => {
  const client = new GlueClient({});
  const command = new GetJobRunCommand({
    JobName: jobName,
    RunId: jobRunId,
  });

  return client.send(command);
};

/**
 * @typedef {{ prompter: { prompt: () => Promise<{jobName: string}> } }} Context
 */

/**
 * @typedef {() => Promise<import('@aws-sdk/client-
glue').GetJobRunCommandOutput>} getJobRun
 */

/**
 * @typedef {() => Promise<import('@aws-sdk/client-
glue').GetJobRunsCommandOutput} getJobRuns
 */
```

```
/**
 *
 * @param {getJobRun} getJobRun
 * @param {string} jobName
 * @param {string} jobRunId
 */
const logJobRunDetails = async (getJobRun, jobName, jobRunId) => {
  const { JobRun } = await getJobRun(jobName, jobRunId);
  log(JobRun, { type: "object" });
};

/**
 *
 * @param {{getJobRuns: getJobRuns, getJobRun: getJobRun }} funcs
 */
const makePickJobRunStep =
  ({ getJobRuns, getJobRun }) =>
  async (/** @type { Context } */ context) => {
    if (context.selectedJobName) {
      const { JobRuns } = await getJobRuns(context.selectedJobName);

      const { jobRunId } = await context.prompter.prompt({
        name: "jobRunId",
        type: "list",
        message: "Select a job run to see details.",
        choices: JobRuns.map((run) => run.Id),
      });

      logJobRunDetails(getJobRun, context.selectedJobName, jobRunId);
    }

    return { ...context };
  };
```

デモによって作成されたすべてのリソースを削除します。

```
const deleteJob = (jobName) => {
  const client = new GlueClient({});

  const command = new DeleteJobCommand({
    JobName: jobName,
```

```
  });

  return client.send(command);
};

const deleteTable = (databaseName, tableName) => {
  const client = new GlueClient({});

  const command = new DeleteTableCommand({
    DatabaseName: databaseName,
    Name: tableName,
  });

  return client.send(command);
};

const deleteDatabase = (databaseName) => {
  const client = new GlueClient({});

  const command = new DeleteDatabaseCommand({
    Name: databaseName,
  });

  return client.send(command);
};

const deleteCrawler = (crawlerName) => {
  const client = new GlueClient({});

  const command = new DeleteCrawlerCommand({
    Name: crawlerName,
  });

  return client.send(command);
};

/**
 *
 * @param {import('../../../actions/delete-job.js').deleteJob} deleteJobFn
 * @param {string[]} jobNames
 * @param {{ prompter: { prompt: () => Promise<any> }}} context
 */
const handleDeleteJobs = async (deleteJobFn, jobNames, context) => {
  /**
```

```
   * @type {{ selectedJobNames: string[] }}
   */
  const { selectedJobNames } = await context.prompter.prompt({
    name: "selectedJobNames",
    type: "checkbox",
    message: "Let's clean up jobs. Select jobs to delete.",
    choices: jobNames,
  });

  if (selectedJobNames.length === 0) {
    log("No jobs selected.");
  } else {
    log("Deleting jobs.");
    await Promise.all(
      selectedJobNames.map((n) => deleteJobFn(n).catch(console.error)),
    );
    log("Jobs deleted.", { type: "success" });
  }
};

/**
 * @param {{
 *   listJobs: import('../../../actions/list-jobs.js').listJobs,
 *   deleteJob: import('../../../actions/delete-job.js').deleteJob
 * }} config
 */
const makeCleanUpJobsStep =
  ({ listJobs, deleteJob }) =>
  async (context) => {
    const { JobNames } = await listJobs();
    if (JobNames.length > 0) {
      await handleDeleteJobs(deleteJob, JobNames, context);
    }

    return { ...context };
  };

/**
 * @param {import('../../../actions/delete-table.js').deleteTable} deleteTable
 * @param {string} databaseName
 * @param {string[]} tableNames
 */
const deleteTables = (deleteTable, databaseName, tableNames) =>
  Promise.all(
```

```
    tableNames.map((tableName) =>
      deleteTable(databaseName, tableName).catch(console.error),
    ),
  );

/**
 * @param {{
 *  getTables: import('../../../actions/get-tables.js').getTables,
 *  deleteTable: import('../../../actions/delete-table.js').deleteTable
 * }} config
 */
const makeCleanUpTablesStep =
  ({ getTables, deleteTable }) =>
  /**
   * @param {{ prompter: { prompt: () => Promise<any>}}} context
   */
  async (context) => {
    const { TableList } = await getTables(process.env.DATABASE_NAME).catch(
      () => ({ TableList: null }),
    );

    if (TableList && TableList.length > 0) {
      /**
       * @type {{ tableNames: string[] }}
       */
      const { tableNames } = await context.prompter.prompt({
        name: "tableNames",
        type: "checkbox",
        message: "Let's clean up tables. Select tables to delete.",
        choices: TableList.map((t) => t.Name),
      });

      if (tableNames.length === 0) {
        log("No tables selected.");
      } else {
        log("Deleting tables.");
        await deleteTables(deleteTable, process.env.DATABASE_NAME, tableNames);
        log("Tables deleted.", { type: "success" });
      }
    }

    return { ...context };
  };
```

```
/**
 * @param {import('../../../actions/delete-database.js').deleteDatabase}
 deleteDatabase
 * @param {string[]} databaseNames
 */
const deleteDatabases = (deleteDatabase, databaseNames) =>
  Promise.all(
    databaseNames.map((dbName) => deleteDatabase(dbName).catch(console.error)),
  );

/**
 * @param {{
 *   getDatabases: import('../../../actions/get-databases.js').getDatabases
 *   deleteDatabase: import('../../../actions/delete-database.js').deleteDatabase
 * }} config
 */
const makeCleanUpDatabasesStep =
  ({ getDatabases, deleteDatabase }) =>
  /**
   * @param {{ prompter: { prompt: () => Promise<any>}} context
   */
  async (context) => {
    const { DatabaseList } = await getDatabases();

    if (DatabaseList.length > 0) {
      /** @type {{ dbNames: string[] }} */
      const { dbNames } = await context.prompter.prompt({
        name: "dbNames",
        type: "checkbox",
        message: "Let's clean up databases. Select databases to delete.",
        choices: DatabaseList.map((db) => db.Name),
      });

      if (dbNames.length === 0) {
        log("No databases selected.");
      } else {
        log("Deleting databases.");
        await deleteDatabases(deleteDatabase, dbNames);
        log("Databases deleted.", { type: "success" });
      }
    }

    return { ...context };
  };
```

```
const cleanUpCrawlerStep = async (context) => {
  log("Deleting crawler.");

  try {
    await deleteCrawler(process.env.CRAWLER_NAME);
    log("Crawler deleted.", { type: "success" });
  } catch (err) {
    if (err.name === "EntityNotFoundException") {
      log("Crawler is already deleted.");
    } else {
      throw err;
    }
  }

  return { ...context };
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の以下のトピックを参照してください。
  - CreateCrawler
  - CreateJob
  - DeleteCrawler
  - DeleteDatabase
  - DeleteJob
  - DeleteTable
  - GetCrawler
  - GetDatabase
  - GetDatabases
  - GetJob
  - GetJob
  - GetJobRuns
  - GetTables
  - ListJobs
  - StartCrawler

- StartJobRun

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
suspend fun main(args: Array<String>) {
    val usage = """
        Usage:
            <iam> <s3Path> <cron> <dbName> <crawlerName> <jobName>
 <scriptLocation> <locationUri>

        Where:
            iam - The Amazon Resource Name (ARN) of the AWS Identity and Access
 Management (IAM) role that has AWS Glue and Amazon Simple Storage Service
 (Amazon S3) permissions.
            s3Path - The Amazon Simple Storage Service (Amazon S3) target that
 contains data (for example, CSV data).
            cron - A cron expression used to specify the schedule (for example,
 cron(15 12 * * ? *).
            dbName - The database name.
            crawlerName - The name of the crawler.
            jobName - The name you assign to this job definition.
            scriptLocation - Specifies the Amazon S3 path to a script that runs a
 job.
            locationUri - Specifies the location of the database
        """

    if (args.size != 8) {
        println(usage)
        exitProcess(1)
    }

    val iam = args[0]
    val s3Path = args[1]
```

```
    val cron = args[2]
    val dbName = args[3]
    val crawlerName = args[4]
    val jobName = args[5]
    val scriptLocation = args[6]
    val locationUri = args[7]

    println("About to start the AWS Glue Scenario")
    createDatabase(dbName, locationUri)
    createCrawler(iam, s3Path, cron, dbName, crawlerName)
    getCrawler(crawlerName)
    startCrawler(crawlerName)
    getDatabase(dbName)
    getGlueTables(dbName)
    createJob(jobName, iam, scriptLocation)
    startJob(jobName)
    getJobs()
    getJobRuns(jobName)
    deleteJob(jobName)
    println("*** Wait for 5 MIN so the $crawlerName is ready to be deleted")
    TimeUnit.MINUTES.sleep(5)
    deleteMyDatabase(dbName)
    deleteCrawler(crawlerName)
}

suspend fun createDatabase(
    dbName: String?,
    locationUriVal: String?,
) {
    val input =
        DatabaseInput {
            description = "Built with the AWS SDK for Kotlin"
            name = dbName
            locationUri = locationUriVal
        }

    val request =
        CreateDatabaseRequest {
            databaseInput = input
        }

    GlueClient { region = "us-east-1" }.use { glueClient ->
        glueClient.createDatabase(request)
        println("The database was successfully created")
```

```
    }
}

suspend fun createCrawler(
    iam: String?,
    s3Path: String?,
    cron: String?,
    dbName: String?,
    crawlerName: String,
) {
    val s3Target =
        S3Target {
            path = s3Path
        }

    val targetList = ArrayList<S3Target>()
    targetList.add(s3Target)

    val targetOb =
        CrawlerTargets {
            s3Targets = targetList
        }

    val crawlerRequest =
        CreateCrawlerRequest {
            databaseName = dbName
            name = crawlerName
            description = "Created by the AWS Glue Java API"
            targets = targetOb
            role = iam
            schedule = cron
        }

    GlueClient { region = "us-east-1" }.use { glueClient ->
        glueClient.createCrawler(crawlerRequest)
        println("$crawlerName was successfully created")
    }
}

suspend fun getCrawler(crawlerName: String?) {
    val request =
        GetCrawlerRequest {
            name = crawlerName
        }
```

```
    GlueClient { region = "us-east-1" }.use { glueClient ->
        val response = glueClient.getCrawler(request)
        val role = response.crawler?.role
        println("The role associated with this crawler is $role")
    }
}

suspend fun startCrawler(crawlerName: String) {
    val crawlerRequest =
        StartCrawlerRequest {
            name = crawlerName
        }

    GlueClient { region = "us-east-1" }.use { glueClient ->
        glueClient.startCrawler(crawlerRequest)
        println("$crawlerName was successfully started.")
    }
}

suspend fun getDatabase(databaseName: String?) {
    val request =
        GetDatabaseRequest {
            name = databaseName
        }

    GlueClient { region = "us-east-1" }.use { glueClient ->
        val response = glueClient.getDatabase(request)
        val dbDesc = response.database?.description
        println("The database description is $dbDesc")
    }
}

suspend fun getGlueTables(dbName: String?) {
    val tableRequest =
        GetTablesRequest {
            databaseName = dbName
        }

    GlueClient { region = "us-east-1" }.use { glueClient ->
        val response = glueClient.getTables(tableRequest)
        response.tableList?.forEach { tableName ->
            println("Table name is ${tableName.name}")
        }
```

```
    }
}

suspend fun startJob(jobNameVal: String?) {
    val runRequest =
        StartJobRunRequest {
            workerType = WorkerType.G1X
            numberOfWorkers = 10
            jobName = jobNameVal
        }

    GlueClient { region = "us-east-1" }.use { glueClient ->
        val response = glueClient.startJobRun(runRequest)
        println("The job run Id is ${response.jobRunId}")
    }
}

suspend fun createJob(
    jobName: String,
    iam: String?,
    scriptLocationVal: String?,
) {
    val commandOb =
        JobCommand {
            pythonVersion = "3"
            name = "MyJob1"
            scriptLocation = scriptLocationVal
        }

    val jobRequest =
        CreateJobRequest {
            description = "A Job created by using the AWS SDK for Java V2"
            glueVersion = "2.0"
            workerType = WorkerType.G1X
            numberOfWorkers = 10
            name = jobName
            role = iam
            command = commandOb
        }

    GlueClient { region = "us-east-1" }.use { glueClient ->
        glueClient.createJob(jobRequest)
        println("$jobName was successfully created.")
    }
```

```
}

suspend fun getJobs() {
    val request =
        GetJobsRequest {
            maxResults = 10
        }

    GlueClient { region = "us-east-1" }.use { glueClient ->
        val response = glueClient.getJobs(request)
        response.jobs?.forEach { job ->
            println("Job name is ${job.name}")
        }
    }
}

suspend fun getJobRuns(jobNameVal: String?) {
    val request =
        GetJobRunsRequest {
            jobName = jobNameVal
        }

    GlueClient { region = "us-east-1" }.use { glueClient ->
        val response = glueClient.getJobRuns(request)
        response.jobRuns?.forEach { job ->
            println("Job name is ${job.jobName}")
        }
    }
}

suspend fun deleteJob(jobNameVal: String) {
    val jobRequest =
        DeleteJobRequest {
            jobName = jobNameVal
        }

    GlueClient { region = "us-east-1" }.use { glueClient ->
        glueClient.deleteJob(jobRequest)
        println("$jobNameVal was successfully deleted")
    }
}

suspend fun deleteMyDatabase(databaseName: String) {
    val request =
```

```
        DeleteDatabaseRequest {
            name = databaseName
        }

    GlueClient { region = "us-east-1" }.use { glueClient ->
        glueClient.deleteDatabase(request)
        println("$databaseName was successfully deleted")
    }
}

suspend fun deleteCrawler(crawlerName: String) {
    val request =
        DeleteCrawlerRequest {
            name = crawlerName
        }
    GlueClient { region = "us-east-1" }.use { glueClient ->
        glueClient.deleteCrawler(request)
        println("$crawlerName was deleted")
    }
}
```

- API の詳細については、「AWS SDK for Kotlin API リファレンス」の以下のトピックを参照してください。
  - CreateCrawler
  - CreateJob
  - DeleteCrawler
  - DeleteDatabase
  - DeleteJob
  - DeleteTable
  - GetCrawler
  - GetDatabase
  - GetDatabases
  - GetJob
  - GetJob
  - GetJobRuns
  - GetTables

- ListJobs
- StartCrawler
- StartJobRun

PHP

SDK for PHP

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

```
namespace Glue;

use Aws\Glue\GlueClient;
use Aws\S3\S3Client;
use AwsUtilities\AWSServiceClass;
use GuzzleHttp\Psr7\Stream;
use Iam\IAMService;

class GettingStartedWithGlue
{
    public function run()
    {
        echo("\n");
        echo("--------------------------------------\n");
        print("Welcome to the AWS Glue getting started demo using PHP!\n");
        echo("--------------------------------------\n");

        $clientArgs = [
            'region' => 'us-west-2',
            'version' => 'latest',
            'profile' => 'default',
        ];
        $uniqid = uniqid();

        $glueClient = new GlueClient($clientArgs);
        $glueService = new GlueService($glueClient);
        $iamService = new IAMService();
```

```
        $crawlerName = "example-crawler-test-" . $uniqid;

        AWSServiceClass::$waitTime = 5;
        AWSServiceClass::$maxWaitAttempts = 20;

        $role = $iamService->getRole("AWSGlueServiceRole-DocExample");

        $databaseName = "doc-example-database-$uniqid";
        $path = 's3://crawler-public-us-east-1/flight/2016/csv';
        $glueService->createCrawler($crawlerName, $role['Role']['Arn'],
$databaseName, $path);
        $glueService->startCrawler($crawlerName);

        echo "Waiting for crawler";
        do {
            $crawler = $glueService->getCrawler($crawlerName);
            echo ".";
            sleep(10);
        } while ($crawler['Crawler']['State'] != "READY");
        echo "\n";

        $database = $glueService->getDatabase($databaseName);
        echo "Found a database named " . $database['Database']['Name'] . "\n";

        //Upload job script
        $s3client = new S3Client($clientArgs);
        $bucketName = "test-glue-bucket-" . $uniqid;
        $s3client->createBucket([
            'Bucket' => $bucketName,
            'CreateBucketConfiguration' => ['LocationConstraint' => 'us-west-2'],
        ]);

        $s3client->putObject([
            'Bucket' => $bucketName,
            'Key' => 'run_job.py',
            'SourceFile' => __DIR__ . '/flight_etl_job_script.py'
        ]);
        $s3client->putObject([
            'Bucket' => $bucketName,
            'Key' => 'setup_scenario_getting_started.yaml',
            'SourceFile' => __DIR__ . '/setup_scenario_getting_started.yaml'
        ]);

        $tables = $glueService->getTables($databaseName);
```

```
        $jobName = 'test-job-' . $uniqid;
        $scriptLocation = "s3://$bucketName/run_job.py";
        $job = $glueService->createJob($jobName, $role['Role']['Arn'],
$scriptLocation);

        $outputBucketUrl = "s3://$bucketName";
        $runId = $glueService->startJobRun($jobName, $databaseName, $tables,
$outputBucketUrl)['JobRunId'];

        echo "waiting for job";
        do {
            $jobRun = $glueService->getJobRun($jobName, $runId);
            echo ".";
            sleep(10);
        } while (!array_intersect([$jobRun['JobRun']['JobRunState']],
['SUCCEEDED', 'STOPPED', 'FAILED', 'TIMEOUT']));
        echo "\n";

        $jobRuns = $glueService->getJobRuns($jobName);

        $objects = $s3client->listObjects([
            'Bucket' => $bucketName,
        ])['Contents'];

        foreach ($objects as $object) {
            echo $object['Key'] . "\n";
        }

        echo "Downloading " . $objects[1]['Key'] . "\n";
        /** @var Stream $downloadObject */
        $downloadObject = $s3client->getObject([
            'Bucket' => $bucketName,
            'Key' => $objects[1]['Key'],
        ])['Body']->getContents();
        echo "Here is the first 1000 characters in the object.";
        echo substr($downloadObject, 0, 1000);

        $jobs = $glueService->listJobs();
        echo "Current jobs:\n";
        foreach ($jobs['JobNames'] as $jobsName) {
            echo "{$jobsName}\n";
        }
```

```
        echo "Delete the job.\n";
        $glueClient->deleteJob([
            'JobName' => $job['Name'],
        ]);

        echo "Delete the tables.\n";
        foreach ($tables['TableList'] as $table) {
            $glueService->deleteTable($table['Name'], $databaseName);
        }

        echo "Delete the databases.\n";
        $glueClient->deleteDatabase([
            'Name' => $databaseName,
        ]);

        echo "Delete the crawler.\n";
        $glueClient->deleteCrawler([
            'Name' => $crawlerName,
        ]);

        $deleteObjects = $s3client->listObjectsV2([
            'Bucket' => $bucketName,
        ]);
        echo "Delete all objects in the bucket.\n";
        $deleteObjects = $s3client->deleteObjects([
            'Bucket' => $bucketName,
            'Delete' => [
                'Objects' => $deleteObjects['Contents'],
            ]
        ]);
        echo "Delete the bucket.\n";
        $s3client->deleteBucket(['Bucket' => $bucketName]);

        echo "This job was brought to you by the number $uniqid\n";
    }
}

namespace Glue;

use Aws\Glue\GlueClient;
use Aws\Result;

use function PHPUnit\Framework\isEmpty;
```

```
class GlueService extends \AwsUtilities\AWSServiceClass
{
    protected GlueClient $glueClient;

    public function __construct($glueClient)
    {
        $this->glueClient = $glueClient;
    }

    public function getCrawler($crawlerName)
    {
        return $this->customWaiter(function () use ($crawlerName) {
            return $this->glueClient->getCrawler([
                'Name' => $crawlerName,
            ]);
        });
    }

    public function createCrawler($crawlerName, $role, $databaseName, $path):
 Result
    {
        return $this->customWaiter(function () use ($crawlerName, $role,
 $databaseName, $path) {
            return $this->glueClient->createCrawler([
                'Name' => $crawlerName,
                'Role' => $role,
                'DatabaseName' => $databaseName,
                'Targets' => [
                    'S3Targets' =>
                        [[
                            'Path' => $path,
                        ]]
                ],
            ]);
        });
    }

    public function startCrawler($crawlerName): Result
    {
        return $this->glueClient->startCrawler([
            'Name' => $crawlerName,
        ]);
    }
```

```
    public function getDatabase(string $databaseName): Result
    {
        return $this->customWaiter(function () use ($databaseName) {
            return $this->glueClient->getDatabase([
                'Name' => $databaseName,
            ]);
        });
    }

    public function getTables($databaseName): Result
    {
        return $this->glueClient->getTables([
            'DatabaseName' => $databaseName,
        ]);
    }

    public function createJob($jobName, $role, $scriptLocation, $pythonVersion =
'3', $glueVersion = '3.0'): Result
    {
        return $this->glueClient->createJob([
            'Name' => $jobName,
            'Role' => $role,
            'Command' => [
                'Name' => 'glueetl',
                'ScriptLocation' => $scriptLocation,
                'PythonVersion' => $pythonVersion,
            ],
            'GlueVersion' => $glueVersion,
        ]);
    }

    public function startJobRun($jobName, $databaseName, $tables,
$outputBucketUrl): Result
    {
        return $this->glueClient->startJobRun([
            'JobName' => $jobName,
            'Arguments' => [
                'input_database' => $databaseName,
                'input_table' => $tables['TableList'][0]['Name'],
                'output_bucket_url' => $outputBucketUrl,
                '--input_database' => $databaseName,
                '--input_table' => $tables['TableList'][0]['Name'],
                '--output_bucket_url' => $outputBucketUrl,
            ],
```

```
        ]);
    }

    public function listJobs($maxResults = null, $nextToken = null, $tags = []):
Result
    {
        $arguments = [];
        if ($maxResults) {
            $arguments['MaxResults'] = $maxResults;
        }
        if ($nextToken) {
            $arguments['NextToken'] = $nextToken;
        }
        if (!empty($tags)) {
            $arguments['Tags'] = $tags;
        }
        return $this->glueClient->listJobs($arguments);
    }

    public function getJobRuns($jobName, $maxResults = 0, $nextToken = ''):
Result
    {
        $arguments = ['JobName' => $jobName];
        if ($maxResults) {
            $arguments['MaxResults'] = $maxResults;
        }
        if ($nextToken) {
            $arguments['NextToken'] = $nextToken;
        }
        return $this->glueClient->getJobRuns($arguments);
    }

    public function getJobRun($jobName, $runId, $predecessorsIncluded = false):
Result
    {
        return $this->glueClient->getJobRun([
            'JobName' => $jobName,
            'RunId' => $runId,
            'PredecessorsIncluded' => $predecessorsIncluded,
        ]);
    }

    public function deleteJob($jobName)
    {
```

```
        return $this->glueClient->deleteJob([
            'JobName' => $jobName,
        ]);
    }

    public function deleteTable($tableName, $databaseName)
    {
        return $this->glueClient->deleteTable([
            'DatabaseName' => $databaseName,
            'Name' => $tableName,
        ]);
    }

    public function deleteDatabase($databaseName)
    {
        return $this->glueClient->deleteDatabase([
            'Name' => $databaseName,
        ]);
    }

    public function deleteCrawler($crawlerName)
    {
        return $this->glueClient->deleteCrawler([
            'Name' => $crawlerName,
        ]);
    }
}
```

- API の詳細については、「AWS SDK for PHP API リファレンス」の以下のトピックを参照してください。
  - CreateCrawler
  - CreateJob
  - DeleteCrawler
  - DeleteDatabase
  - DeleteJob
  - DeleteTable
  - GetCrawler
  - GetDatabase

- GetDatabases
- GetJob
- GetJobRun
- GetJobRuns
- GetTables
- ListJobs
- StartCrawler
- StartJobRun

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

シナリオで使用される AWS Glue 関数をラップするクラスを作成します。

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""

    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 Glue client.
        """
        self.glue_client = glue_client


    def get_crawler(self, name):
        """
        Gets information about a crawler.

        :param name: The name of the crawler to look up.
        :return: Data about the crawler.
        """
        crawler = None
```

```
        try:
            response = self.glue_client.get_crawler(Name=name)
            crawler = response["Crawler"]
        except ClientError as err:
            if err.response["Error"]["Code"] == "EntityNotFoundException":
                logger.info("Crawler %s doesn't exist.", name)
            else:
                logger.error(
                    "Couldn't get crawler %s. Here's why: %s: %s",
                    name,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        return crawler


    def create_crawler(self, name, role_arn, db_name, db_prefix, s3_target):
        """
        Creates a crawler that can crawl the specified target and populate a
        database in your AWS Glue Data Catalog with metadata that describes the
data
        in the target.

        :param name: The name of the crawler.
        :param role_arn: The Amazon Resource Name (ARN) of an AWS Identity and
Access
                         Management (IAM) role that grants permission to let AWS
Glue
                         access the resources it needs.
        :param db_name: The name to give the database that is created by the
crawler.
        :param db_prefix: The prefix to give any database tables that are created
by
                          the crawler.
        :param s3_target: The URL to an S3 bucket that contains data that is
                          the target of the crawler.
        """
        try:
            self.glue_client.create_crawler(
                Name=name,
                Role=role_arn,
                DatabaseName=db_name,
                TablePrefix=db_prefix,
```

```
                Targets={"S3Targets": [{"Path": s3_target}]},
            )
        except ClientError as err:
            logger.error(
                "Couldn't create crawler. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise


    def start_crawler(self, name):
        """
        Starts a crawler. The crawler crawls its configured target and creates
        metadata that describes the data it finds in the target data source.

        :param name: The name of the crawler to start.
        """
        try:
            self.glue_client.start_crawler(Name=name)
        except ClientError as err:
            logger.error(
                "Couldn't start crawler %s. Here's why: %s: %s",
                name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise


    def get_database(self, name):
        """
        Gets information about a database in your Data Catalog.

        :param name: The name of the database to look up.
        :return: Information about the database.
        """
        try:
            response = self.glue_client.get_database(Name=name)
        except ClientError as err:
            logger.error(
                "Couldn't get database %s. Here's why: %s: %s",
                name,
                err.response["Error"]["Code"],
```

```
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["Database"]


    def get_tables(self, db_name):
        """
        Gets a list of tables in a Data Catalog database.

        :param db_name: The name of the database to query.
        :return: The list of tables in the database.
        """
        try:
            response = self.glue_client.get_tables(DatabaseName=db_name)
        except ClientError as err:
            logger.error(
                "Couldn't get tables %s. Here's why: %s: %s",
                db_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["TableList"]


    def create_job(self, name, description, role_arn, script_location):
        """
        Creates a job definition for an extract, transform, and load (ETL) job
that can
        be run by AWS Glue.

        :param name: The name of the job definition.
        :param description: The description of the job definition.
        :param role_arn: The ARN of an IAM role that grants AWS Glue the
permissions
                         it requires to run the job.
        :param script_location: The Amazon S3 URL of a Python ETL script that is
run as
                                part of the job. The script defines how the data
is
                                transformed.
```

```
        """
        try:
            self.glue_client.create_job(
                Name=name,
                Description=description,
                Role=role_arn,
                Command={
                    "Name": "glueetl",
                    "ScriptLocation": script_location,
                    "PythonVersion": "3",
                },
                GlueVersion="3.0",
            )
        except ClientError as err:
            logger.error(
                "Couldn't create job %s. Here's why: %s: %s",
                name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise


    def start_job_run(self, name, input_database, input_table,
output_bucket_name):
        """
        Starts a job run. A job run extracts data from the source, transforms it,
        and loads it to the output bucket.

        :param name: The name of the job definition.
        :param input_database: The name of the metadata database that contains
tables
                               that describe the source data. This is typically
created
                               by a crawler.
        :param input_table: The name of the table in the metadata database that
                            describes the source data.
        :param output_bucket_name: The S3 bucket where the output is written.
        :return: The ID of the job run.
        """
        try:
            # The custom Arguments that are passed to this function are used by
the
```

```
            # Python ETL script to determine the location of input and output
data.
            response = self.glue_client.start_job_run(
                JobName=name,
                Arguments={
                    "--input_database": input_database,
                    "--input_table": input_table,
                    "--output_bucket_url": f"s3://{output_bucket_name}/",
                },
            )
        except ClientError as err:
            logger.error(
                "Couldn't start job run %s. Here's why: %s: %s",
                name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["JobRunId"]


    def list_jobs(self):
        """
        Lists the names of job definitions in your account.

        :return: The list of job definition names.
        """
        try:
            response = self.glue_client.list_jobs()
        except ClientError as err:
            logger.error(
                "Couldn't list jobs. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["JobNames"]


    def get_job_runs(self, job_name):
        """
        Gets information about runs that have been performed for a specific job
```

```
        definition.

        :param job_name: The name of the job definition to look up.
        :return: The list of job runs.
        """
        try:
            response = self.glue_client.get_job_runs(JobName=job_name)
        except ClientError as err:
            logger.error(
                "Couldn't get job runs for %s. Here's why: %s: %s",
                job_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["JobRuns"]


    def get_job_run(self, name, run_id):
        """
        Gets information about a single job run.

        :param name: The name of the job definition for the run.
        :param run_id: The ID of the run.
        :return: Information about the run.
        """
        try:
            response = self.glue_client.get_job_run(JobName=name, RunId=run_id)
        except ClientError as err:
            logger.error(
                "Couldn't get job run %s/%s. Here's why: %s: %s",
                name,
                run_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["JobRun"]


    def delete_job(self, job_name):
        """
```

```
        Deletes a job definition. This also deletes data about all runs that are
        associated with this job definition.

        :param job_name: The name of the job definition to delete.
        """
        try:
            self.glue_client.delete_job(JobName=job_name)
        except ClientError as err:
            logger.error(
                "Couldn't delete job %s. Here's why: %s: %s",
                job_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise


    def delete_table(self, db_name, table_name):
        """
        Deletes a table from a metadata database.

        :param db_name: The name of the database that contains the table.
        :param table_name: The name of the table to delete.
        """
        try:
            self.glue_client.delete_table(DatabaseName=db_name, Name=table_name)
        except ClientError as err:
            logger.error(
                "Couldn't delete table %s. Here's why: %s: %s",
                table_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise


    def delete_database(self, name):
        """
        Deletes a metadata database from your Data Catalog.

        :param name: The name of the database to delete.
        """
        try:
            self.glue_client.delete_database(Name=name)
```

```
        except ClientError as err:
            logger.error(
                "Couldn't delete database %s. Here's why: %s: %s",
                name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise


    def delete_crawler(self, name):
        """
        Deletes a crawler.

        :param name: The name of the crawler to delete.
        """
        try:
            self.glue_client.delete_crawler(Name=name)
        except ClientError as err:
            logger.error(
                "Couldn't delete crawler %s. Here's why: %s: %s",
                name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

シナリオを実行するクラスを作成します。

```
class GlueCrawlerJobScenario:
    """
    Encapsulates a scenario that shows how to create an AWS Glue crawler and job
 and use
    them to transform data from CSV to JSON format.
    """

    def __init__(self, glue_client, glue_service_role, glue_bucket):
        """
        :param glue_client: A Boto3 AWS Glue client.
```

```
        :param glue_service_role: An AWS Identity and Access Management (IAM)
role
                                  that AWS Glue can assume to gain access to the
                                  resources it requires.
        :param glue_bucket: An S3 bucket that can hold a job script and output
data
                            from AWS Glue job runs.
        """
        self.glue_client = glue_client
        self.glue_service_role = glue_service_role
        self.glue_bucket = glue_bucket

    @staticmethod
    def wait(seconds, tick=12):
        """
        Waits for a specified number of seconds, while also displaying an
animated
        spinner.

        :param seconds: The number of seconds to wait.
        :param tick: The number of frames per second used to animate the spinner.
        """
        progress = "|/-\\"
        waited = 0
        while waited < seconds:
            for frame in range(tick):
                sys.stdout.write(f"\r{progress[frame % len(progress)]}")
                sys.stdout.flush()
                time.sleep(1 / tick)
            waited += 1

    def upload_job_script(self, job_script):
        """
        Uploads a Python ETL script to an S3 bucket. The script is used by the
AWS Glue
        job to transform data.

        :param job_script: The relative path to the job script.
        """
        try:
            self.glue_bucket.upload_file(Filename=job_script, Key=job_script)
            print(f"Uploaded job script '{job_script}' to the example bucket.")
        except S3UploadFailedError as err:
            logger.error("Couldn't upload job script. Here's why: %s", err)
```

```
            raise

    def run(self, crawler_name, db_name, db_prefix, data_source, job_script,
job_name):
        """
        Runs the scenario. This is an interactive experience that runs at a
command
        prompt and asks you for input throughout.

        :param crawler_name: The name of the crawler used in the scenario. If the
                             crawler does not exist, it is created.
        :param db_name: The name to give the metadata database created by the
crawler.
        :param db_prefix: The prefix to give tables added to the database by the
                          crawler.
        :param data_source: The location of the data source that is targeted by
the
                            crawler and extracted during job runs.
        :param job_script: The job script that is used to transform data during
job
                           runs.
        :param job_name: The name to give the job definition that is created
during the
                         scenario.
        """
        wrapper = GlueWrapper(self.glue_client)
        print(f"Checking for crawler {crawler_name}.")
        crawler = wrapper.get_crawler(crawler_name)
        if crawler is None:
            print(f"Creating crawler {crawler_name}.")
            wrapper.create_crawler(
                crawler_name,
                self.glue_service_role.arn,
                db_name,
                db_prefix,
                data_source,
            )
            print(f"Created crawler {crawler_name}.")
            crawler = wrapper.get_crawler(crawler_name)
        pprint(crawler)
        print("-" * 88)

        print(
```

```
            f"When you run the crawler, it crawls data stored in {data_source}
and "
            f"creates a metadata database in the AWS Glue Data Catalog that
describes "
            f"the data in the data source."
        )
        print("In this example, the source data is in CSV format.")
        ready = False
        while not ready:
            ready = Question.ask_question(
                "Ready to start the crawler? (y/n) ", Question.is_yesno
            )
        wrapper.start_crawler(crawler_name)
        print("Let's wait for the crawler to run. This typically takes a few
minutes.")
        crawler_state = None
        while crawler_state != "READY":
            self.wait(10)
            crawler = wrapper.get_crawler(crawler_name)
            crawler_state = crawler["State"]
            print(f"Crawler is {crawler['State']}.")
        print("-" * 88)

        database = wrapper.get_database(db_name)
        print(f"The crawler created database {db_name}:")
        pprint(database)
        print(f"The database contains these tables:")
        tables = wrapper.get_tables(db_name)
        for index, table in enumerate(tables):
            print(f"\t{index + 1}. {table['Name']}")
        table_index = Question.ask_question(
            f"Enter the number of a table to see more detail: ",
            Question.is_int,
            Question.in_range(1, len(tables)),
        )
        pprint(tables[table_index - 1])
        print("-" * 88)

        print(f"Creating job definition {job_name}.")
        wrapper.create_job(
            job_name,
            "Getting started example job.",
            self.glue_service_role.arn,
            f"s3://{self.glue_bucket.name}/{job_script}",
```

```
        )
        print("Created job definition.")
        print(
            f"When you run the job, it extracts data from {data_source},
transforms it "
            f"by using the {job_script} script, and loads the output into "
            f"S3 bucket {self.glue_bucket.name}."
        )
        print(
            "In this example, the data is transformed from CSV to JSON, and only
a few "
            "fields are included in the output."
        )
        job_run_status = None
        if Question.ask_question(f"Ready to run? (y/n) ", Question.is_yesno):
            job_run_id = wrapper.start_job_run(
                job_name, db_name, tables[0]["Name"], self.glue_bucket.name
            )
            print(f"Job {job_name} started. Let's wait for it to run.")
            while job_run_status not in ["SUCCEEDED", "STOPPED", "FAILED",
"TIMEOUT"]:
                self.wait(10)
                job_run = wrapper.get_job_run(job_name, job_run_id)
                job_run_status = job_run["JobRunState"]
                print(f"Job {job_name}/{job_run_id} is {job_run_status}.")
        print("-" * 88)

        if job_run_status == "SUCCEEDED":
            print(
                f"Data from your job run is stored in your S3 bucket
'{self.glue_bucket.name}':"
            )
            try:
                keys = [
                    obj.key for obj in
self.glue_bucket.objects.filter(Prefix="run-")
                ]
                for index, key in enumerate(keys):
                    print(f"\t{index + 1}: {key}")
                lines = 4
                key_index = Question.ask_question(
                    f"Enter the number of a block to download it and see the
first {lines} "
                    f"lines of JSON output in the block: ",
```

```
                    Question.is_int,
                    Question.in_range(1, len(keys)),
                )
                job_data = io.BytesIO()
                self.glue_bucket.download_fileobj(keys[key_index - 1], job_data)
                job_data.seek(0)
                for _ in range(lines):
                    print(job_data.readline().decode("utf-8"))
            except ClientError as err:
                logger.error(
                    "Couldn't get job run data. Here's why: %s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
            print("-" * 88)

        job_names = wrapper.list_jobs()
        if job_names:
            print(f"Your account has {len(job_names)} jobs defined:")
            for index, job_name in enumerate(job_names):
                print(f"\t{index + 1}. {job_name}")
            job_index = Question.ask_question(
                f"Enter a number between 1 and {len(job_names)} to see the list
of runs for "
                f"a job: ",
                Question.is_int,
                Question.in_range(1, len(job_names)),
            )
            job_runs = wrapper.get_job_runs(job_names[job_index - 1])
            if job_runs:
                print(f"Found {len(job_runs)} runs for job {job_names[job_index -
1]}:")
                for index, job_run in enumerate(job_runs):
                    print(
                        f"\t{index + 1}. {job_run['JobRunState']} on "
                        f"{job_run['CompletedOn']:%Y-%m-%d %H:%M:%S}"
                    )
                run_index = Question.ask_question(
                    f"Enter a number between 1 and {len(job_runs)} to see details
for a run: ",
                    Question.is_int,
                    Question.in_range(1, len(job_runs)),
                )
```

```
                pprint(job_runs[run_index - 1])
            else:
                print(f"No runs found for job {job_names[job_index - 1]}")
        else:
            print("Your account doesn't have any jobs defined.")
        print("-" * 88)

        print(
            f"Let's clean up. During this example we created job definition
 '{job_name}'."
        )
        if Question.ask_question(
            "Do you want to delete the definition and all runs? (y/n) ",
            Question.is_yesno,
        ):
            wrapper.delete_job(job_name)
            print(f"Job definition '{job_name}' deleted.")
        tables = wrapper.get_tables(db_name)
        print(f"We also created database '{db_name}' that contains these
 tables:")
        for table in tables:
            print(f"\t{table['Name']}")
        if Question.ask_question(
            "Do you want to delete the tables and the database? (y/n) ",
            Question.is_yesno,
        ):
            for table in tables:
                wrapper.delete_table(db_name, table["Name"])
                print(f"Deleted table {table['Name']}.")
            wrapper.delete_database(db_name)
            print(f"Deleted database {db_name}.")
        print(f"We also created crawler '{crawler_name}'.")
        if Question.ask_question(
            "Do you want to delete the crawler? (y/n) ", Question.is_yesno
        ):
            wrapper.delete_crawler(crawler_name)
            print(f"Deleted crawler {crawler_name}.")
        print("-" * 88)


def parse_args(args):
    """
    Parse command line arguments.
```

```
    :param args: The command line arguments.
    :return: The parsed arguments.
    """
    parser = argparse.ArgumentParser(
        description="Runs the AWS Glue getting started with crawlers and jobs
 scenario. "
        "Before you run this scenario, set up scaffold resources by running "
        "'python scaffold.py deploy'."
    )
    parser.add_argument(
        "role_name",
        help="The name of an IAM role that AWS Glue can assume. This role must
 grant access "
        "to Amazon S3 and to the permissions granted by the AWSGlueServiceRole "
        "managed policy.",
    )
    parser.add_argument(
        "bucket_name",
        help="The name of an S3 bucket that AWS Glue can access to get the job
 script and "
        "put job results.",
    )
    parser.add_argument(
        "--job_script",
        default="flight_etl_job_script.py",
        help="The name of the job script file that is used in the scenario.",
    )
    return parser.parse_args(args)


def main():
    args = parse_args(sys.argv[1:])
    try:
        print("-" * 88)
        print(
            "Welcome to the AWS Glue getting started with crawlers and jobs
 scenario."
        )
        print("-" * 88)
        scenario = GlueCrawlerJobScenario(
            boto3.client("glue"),
            boto3.resource("iam").Role(args.role_name),
            boto3.resource("s3").Bucket(args.bucket_name),
        )
```

```
        scenario.upload_job_script(args.job_script)
        scenario.run(
            "doc-example-crawler",
            "doc-example-database",
            "doc-example-",
            "s3://crawler-public-us-east-1/flight/2016/csv",
            args.job_script,
            "doc-example-job",
        )
        print("-" * 88)
        print(
            "To destroy scaffold resources, including the IAM role and S3 bucket
 "
            "used in this scenario, run 'python scaffold.py destroy'."
        )
        print("\nThanks for watching!")
        print("-" * 88)
    except Exception:
        logging.exception("Something went wrong with the example.")
```

ジョブの実行中に AWS Glue がデータの抽出、変換、ロードに使用する ETL スクリプトを作成します。

```
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job

"""
These custom arguments must be passed as Arguments to the StartJobRun request.
    --input_database    The name of a metadata database that is contained in
 your
                        AWS Glue Data Catalog and that contains tables that
 describe
                        the data to be processed.
    --input_table       The name of a table in the database that describes the
 data to
                        be processed.
```

```
    --output_bucket_url An S3 bucket that receives the transformed output data.
"""
args = getResolvedOptions(
    sys.argv, ["JOB_NAME", "input_database", "input_table", "output_bucket_url"]
)
sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
job.init(args["JOB_NAME"], args)

# Script generated for node S3 Flight Data.
S3FlightData_node1 = glueContext.create_dynamic_frame.from_catalog(
    database=args["input_database"],
    table_name=args["input_table"],
    transformation_ctx="S3FlightData_node1",
)

# This mapping performs two main functions:
# 1. It simplifies the output by removing most of the fields from the data.
# 2. It renames some fields. For example, `fl_date` is renamed to `flight_date`.
ApplyMapping_node2 = ApplyMapping.apply(
    frame=S3FlightData_node1,
    mappings=[
        ("year", "long", "year", "long"),
        ("month", "long", "month", "tinyint"),
        ("day_of_month", "long", "day", "tinyint"),
        ("fl_date", "string", "flight_date", "string"),
        ("carrier", "string", "carrier", "string"),
        ("fl_num", "long", "flight_num", "long"),
        ("origin_city_name", "string", "origin_city_name", "string"),
        ("origin_state_abr", "string", "origin_state_abr", "string"),
        ("dest_city_name", "string", "dest_city_name", "string"),
        ("dest_state_abr", "string", "dest_state_abr", "string"),
        ("dep_time", "long", "departure_time", "long"),
        ("wheels_off", "long", "wheels_off", "long"),
        ("wheels_on", "long", "wheels_on", "long"),
        ("arr_time", "long", "arrival_time", "long"),
        ("mon", "string", "mon", "string"),
    ],
    transformation_ctx="ApplyMapping_node2",
)

# Script generated for node Revised Flight Data.
```

```
RevisedFlightData_node3 = glueContext.write_dynamic_frame.from_options(
    frame=ApplyMapping_node2,
    connection_type="s3",
    format="json",
    connection_options={"path": args["output_bucket_url"], "partitionKeys": []},
    transformation_ctx="RevisedFlightData_node3",
)

job.commit()
```

- API の詳細については、「AWS SDK for Python (Boto3) API リファレンス」の以下のトピックを参照してください。
  - CreateCrawler
  - CreateJob
  - DeleteCrawler
  - DeleteDatabase
  - DeleteJob
  - DeleteTable
  - GetCrawler
  - GetDatabase
  - GetDatabases
  - GetJob
  - GetJobRun
  - GetJobRuns
  - GetTables
  - ListJobs
  - StartCrawler
  - StartJobRun

Ruby

SDK for Ruby

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

シナリオで使用される AWS Glue 関数をラップするクラスを作成します。

```
# The `GlueWrapper` class serves as a wrapper around the AWS Glue API, providing
 a simplified interface for common operations.
# It encapsulates the functionality of the AWS SDK for Glue and provides methods
 for interacting with Glue crawlers, databases, tables, jobs, and S3 resources.
# The class initializes with a Glue client and a logger, allowing it to make API
 calls and log any errors or informational messages.
class GlueWrapper
  def initialize(glue_client, logger)
    @glue_client = glue_client
    @logger = logger
  end

  # Retrieves information about a specific crawler.
  #
  # @param name [String] The name of the crawler to retrieve information about.
  # @return [Aws::Glue::Types::Crawler, nil] The crawler object if found, or nil
 if not found.
  def get_crawler(name)
    @glue_client.get_crawler(name: name)
  rescue Aws::Glue::Errors::EntityNotFoundException
    @logger.info("Crawler #{name} doesn't exist.")
    false
  rescue Aws::Glue::Errors::GlueException => e
    @logger.error("Glue could not get crawler #{name}: \n#{e.message}")
    raise
  end

  # Creates a new crawler with the specified configuration.
  #
  # @param name [String] The name of the crawler.
```

```
 # @param role_arn [String] The ARN of the IAM role to be used by the crawler.
 # @param db_name [String] The name of the database where the crawler stores its
metadata.
 # @param db_prefix [String] The prefix to be added to the names of tables that
the crawler creates.
 # @param s3_target [String] The S3 path that the crawler will crawl.
 # @return [void]
 def create_crawler(name, role_arn, db_name, _db_prefix, s3_target)
   @glue_client.create_crawler(
     name: name,
     role: role_arn,
     database_name: db_name,
     targets: {
       s3_targets: [
         {
           path: s3_target
         }
       ]
     }
   )
 rescue Aws::Glue::Errors::GlueException => e
   @logger.error("Glue could not create crawler: \n#{e.message}")
   raise
 end

 # Starts a crawler with the specified name.
 #
 # @param name [String] The name of the crawler to start.
 # @return [void]
 def start_crawler(name)
   @glue_client.start_crawler(name: name)
 rescue Aws::Glue::Errors::ServiceError => e
   @logger.error("Glue could not start crawler #{name}: \n#{e.message}")
   raise
 end

 # Deletes a crawler with the specified name.
 #
 # @param name [String] The name of the crawler to delete.
 # @return [void]
 def delete_crawler(name)
   @glue_client.delete_crawler(name: name)
 rescue Aws::Glue::Errors::ServiceError => e
   @logger.error("Glue could not delete crawler #{name}: \n#{e.message}")
```

```
    raise
  end

  # Retrieves information about a specific database.
  #
  # @param name [String] The name of the database to retrieve information about.
  # @return [Aws::Glue::Types::Database, nil] The database object if found, or
nil if not found.
  def get_database(name)
    response = @glue_client.get_database(name: name)
    response.database
  rescue Aws::Glue::Errors::GlueException => e
    @logger.error("Glue could not get database #{name}: \n#{e.message}")
    raise
  end

  # Retrieves a list of tables in the specified database.
  #
  # @param db_name [String] The name of the database to retrieve tables from.
  # @return [Array<Aws::Glue::Types::Table>]
  def get_tables(db_name)
    response = @glue_client.get_tables(database_name: db_name)
    response.table_list
  rescue Aws::Glue::Errors::GlueException => e
    @logger.error("Glue could not get tables #{db_name}: \n#{e.message}")
    raise
  end

  # Creates a new job with the specified configuration.
  #
  # @param name [String] The name of the job.
  # @param description [String] The description of the job.
  # @param role_arn [String] The ARN of the IAM role to be used by the job.
  # @param script_location [String] The location of the ETL script for the job.
  # @return [void]
  def create_job(name, description, role_arn, script_location)
    @glue_client.create_job(
      name: name,
      description: description,
      role: role_arn,
      command: {
        name: 'glueetl',
        script_location: script_location,
        python_version: '3'
```

```
      },
      glue_version: '3.0'
    )
  rescue Aws::Glue::Errors::GlueException => e
    @logger.error("Glue could not create job #{name}: \n#{e.message}")
    raise
  end

  # Starts a job run for the specified job.
  #
  # @param name [String] The name of the job to start the run for.
  # @param input_database [String] The name of the input database for the job.
  # @param input_table [String] The name of the input table for the job.
  # @param output_bucket_name [String] The name of the output S3 bucket for the
job.
  # @return [String] The ID of the started job run.
  def start_job_run(name, input_database, input_table, output_bucket_name)
    response = @glue_client.start_job_run(
      job_name: name,
      arguments: {
        '--input_database': input_database,
        '--input_table': input_table,
        '--output_bucket_url': "s3://#{output_bucket_name}/"
      }
    )
    response.job_run_id
  rescue Aws::Glue::Errors::GlueException => e
    @logger.error("Glue could not start job run #{name}: \n#{e.message}")
    raise
  end

  # Retrieves a list of jobs in AWS Glue.
  #
  # @return [Aws::Glue::Types::ListJobsResponse]
  def list_jobs
    @glue_client.list_jobs
  rescue Aws::Glue::Errors::GlueException => e
    @logger.error("Glue could not list jobs: \n#{e.message}")
    raise
  end

  # Retrieves a list of job runs for the specified job.
  #
  # @param job_name [String] The name of the job to retrieve job runs for.
```

```
  # @return [Array<Aws::Glue::Types::JobRun>]
  def get_job_runs(job_name)
    response = @glue_client.get_job_runs(job_name: job_name)
    response.job_runs
  rescue Aws::Glue::Errors::GlueException => e
    @logger.error("Glue could not get job runs: \n#{e.message}")
  end

  # Retrieves data for a specific job run.
  #
  # @param job_name [String] The name of the job run to retrieve data for.
  # @return [Glue::Types::GetJobRunResponse]
  def get_job_run(job_name, run_id)
    @glue_client.get_job_run(job_name: job_name, run_id: run_id)
  rescue Aws::Glue::Errors::GlueException => e
    @logger.error("Glue could not get job runs: \n#{e.message}")
  end

  # Deletes a job with the specified name.
  #
  # @param job_name [String] The name of the job to delete.
  # @return [void]
  def delete_job(job_name)
    @glue_client.delete_job(job_name: job_name)
  rescue Aws::Glue::Errors::ServiceError => e
    @logger.error("Glue could not delete job: \n#{e.message}")
  end

  # Deletes a table with the specified name.
  #
  # @param database_name [String] The name of the catalog database in which the
table resides.
  # @param table_name [String] The name of the table to be deleted.
  # @return [void]
  def delete_table(database_name, table_name)
    @glue_client.delete_table(database_name: database_name, name: table_name)
  rescue Aws::Glue::Errors::ServiceError => e
    @logger.error("Glue could not delete job: \n#{e.message}")
  end

  # Removes a specified database from a Data Catalog.
  #
  # @param database_name [String] The name of the database to delete.
  # @return [void]
```

```
  def delete_database(database_name)
    @glue_client.delete_database(name: database_name)
  rescue Aws::Glue::Errors::ServiceError => e
    @logger.error("Glue could not delete database: \n#{e.message}")
  end

  # Uploads a job script file to an S3 bucket.
  #
  # @param file_path [String] The local path of the job script file.
  # @param bucket_resource [Aws::S3::Bucket] The S3 bucket resource to upload the
 file to.
  # @return [void]
  def upload_job_script(file_path, bucket_resource)
    File.open(file_path) do |file|
      bucket_resource.client.put_object({
                                          body: file,
                                          bucket: bucket_resource.name,
                                          key: file_path
                                        })
    end
  rescue Aws::S3::Errors::S3UploadFailedError => e
    @logger.error("S3 could not upload job script: \n#{e.message}")
    raise
  end
end
```

シナリオを実行するクラスを作成します。

```
class GlueCrawlerJobScenario
  def initialize(glue_client, glue_service_role, glue_bucket, logger)
    @glue_client = glue_client
    @glue_service_role = glue_service_role
    @glue_bucket = glue_bucket
    @logger = logger
  end

  def run(crawler_name, db_name, db_prefix, data_source, job_script, job_name)
    wrapper = GlueWrapper.new(@glue_client, @logger)
    setup_crawler(wrapper, crawler_name, db_name, db_prefix, data_source)
    query_database(wrapper, crawler_name, db_name)
    create_and_run_job(wrapper, job_script, job_name, db_name)
  end
```

```
  private

  def setup_crawler(wrapper, crawler_name, db_name, db_prefix, data_source)
    new_step(1, 'Create a crawler')
    crawler = wrapper.get_crawler(crawler_name)
    unless crawler
      puts "Creating crawler #{crawler_name}."
      wrapper.create_crawler(crawler_name, @glue_service_role.arn, db_name,
 db_prefix, data_source)
      puts "Successfully created #{crawler_name}."
    end
    wrapper.start_crawler(crawler_name)
    monitor_crawler(wrapper, crawler_name)
  end

  def monitor_crawler(wrapper, crawler_name)
    new_step(2, 'Monitor Crawler')
    crawler_state = nil
    until crawler_state == 'READY'
      custom_wait(15)
      crawler = wrapper.get_crawler(crawler_name)
      crawler_state = crawler[0]['state']
      print "Crawler status: #{crawler_state}".yellow
    end
  end

  def query_database(wrapper, _crawler_name, db_name)
    new_step(3, 'Query the database.')
    wrapper.get_database(db_name)
    puts "The crawler created database #{db_name}:"
    puts "Database contains tables: #{wrapper.get_tables(db_name).map { |t|
 t['name'] }}"
  end

  def create_and_run_job(wrapper, job_script, job_name, db_name)
    new_step(4, 'Create and run job.')
    wrapper.upload_job_script(job_script, @glue_bucket)
    wrapper.create_job(job_name, 'ETL Job', @glue_service_role.arn, "s3://
#{@glue_bucket.name}/#{job_script}")
    run_job(wrapper, job_name, db_name)
  end

  def run_job(wrapper, job_name, db_name)
```

```
    new_step(5, 'Run the job.')
    wrapper.start_job_run(job_name, db_name, wrapper.get_tables(db_name)[0]
['name'], @glue_bucket.name)
    job_run_status = nil
    until %w[SUCCEEDED FAILED STOPPED].include?(job_run_status)
      custom_wait(10)
      job_run = wrapper.get_job_runs(job_name)
      job_run_status = job_run[0]['job_run_state']
      print "Job #{job_name} status: #{job_run_status}".yellow
    end
  end
end

def main
  banner('../../helpers/banner.txt')
  puts 'Starting AWS Glue demo...'

  # Load resource names from YAML.
  resource_names = YAML.load_file('resource_names.yaml')

  # Setup services and resources.
  iam_role = Aws::IAM::Resource.new(region: 'us-
east-1').role(resource_names['glue_service_role'])
  s3_bucket = Aws::S3::Resource.new(region: 'us-
east-1').bucket(resource_names['glue_bucket'])

  # Instantiate scenario and run.
  scenario = GlueCrawlerJobScenario.new(Aws::Glue::Client.new(region: 'us-
east-1'), iam_role, s3_bucket, @logger)
  random_suffix = rand(10**4)
  scenario.run("crawler-#{random_suffix}", "db-#{random_suffix}", "prefix-
#{random_suffix}-", 's3://data_source',
               'job_script.py', "job-#{random_suffix}")

  puts 'Demo complete.'
end
```

ジョブの実行中に AWS Glue がデータの抽出、変換、ロードに使用する ETL スクリプトを作成します。

```
import sys
from awsglue.transforms import *
```

```
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job

"""
These custom arguments must be passed as Arguments to the StartJobRun request.
    --input_database    The name of a metadata database that is contained in
 your
                        AWS Glue Data Catalog and that contains tables that
 describe
                        the data to be processed.
    --input_table       The name of a table in the database that describes the
 data to
                        be processed.
    --output_bucket_url An S3 bucket that receives the transformed output data.
"""
args = getResolvedOptions(
    sys.argv, ["JOB_NAME", "input_database", "input_table", "output_bucket_url"]
)
sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
job.init(args["JOB_NAME"], args)

# Script generated for node S3 Flight Data.
S3FlightData_node1 = glueContext.create_dynamic_frame.from_catalog(
    database=args["input_database"],
    table_name=args["input_table"],
    transformation_ctx="S3FlightData_node1",
)

# This mapping performs two main functions:
# 1. It simplifies the output by removing most of the fields from the data.
# 2. It renames some fields. For example, `fl_date` is renamed to `flight_date`.
ApplyMapping_node2 = ApplyMapping.apply(
    frame=S3FlightData_node1,
    mappings=[
        ("year", "long", "year", "long"),
        ("month", "long", "month", "tinyint"),
        ("day_of_month", "long", "day", "tinyint"),
        ("fl_date", "string", "flight_date", "string"),
        ("carrier", "string", "carrier", "string"),
```

```
        ("fl_num", "long", "flight_num", "long"),
        ("origin_city_name", "string", "origin_city_name", "string"),
        ("origin_state_abr", "string", "origin_state_abr", "string"),
        ("dest_city_name", "string", "dest_city_name", "string"),
        ("dest_state_abr", "string", "dest_state_abr", "string"),
        ("dep_time", "long", "departure_time", "long"),
        ("wheels_off", "long", "wheels_off", "long"),
        ("wheels_on", "long", "wheels_on", "long"),
        ("arr_time", "long", "arrival_time", "long"),
        ("mon", "string", "mon", "string"),
    ],
    transformation_ctx="ApplyMapping_node2",
)

# Script generated for node Revised Flight Data.
RevisedFlightData_node3 = glueContext.write_dynamic_frame.from_options(
    frame=ApplyMapping_node2,
    connection_type="s3",
    format="json",
    connection_options={"path": args["output_bucket_url"], "partitionKeys": []},
    transformation_ctx="RevisedFlightData_node3",
)

job.commit()
```

- API の詳細については、「AWS SDK for Ruby API リファレンス」の以下のトピックを参照してください。
  - CreateCrawler
  - CreateJob
  - DeleteCrawler
  - DeleteDatabase
  - DeleteJob
  - DeleteTable
  - GetCrawler
  - GetDatabase
  - GetDatabases
  - GetJob
  - GetJob

- GetJobRuns
- GetTables
- ListJobs
- StartCrawler
- StartJobRun

Rust

SDK for Rust

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

パブリックの Amazon Simple Storage Service (Amazon S3) バケットをクロールし、検出した CSV 形式データを記述するメタデータデータベースを生成するクローラーを作成して実行します。

```
        let create_crawler = glue
            .create_crawler()
            .name(self.crawler())
            .database_name(self.database())
            .role(self.iam_role.expose_secret())
            .targets(
                CrawlerTargets::builder()
                    .s3_targets(S3Target::builder().path(CRAWLER_TARGET).build())
                    .build(),
            )
            .send()
            .await;

        match create_crawler {
            Err(err) => {
                let glue_err: aws_sdk_glue::Error = err.into();
                match glue_err {
                    aws_sdk_glue::Error::AlreadyExistsException(_) => {
                        info!("Using existing crawler");
```

```
                        Ok(())
                    }
                    _ => Err(GlueMvpError::GlueSdk(glue_err)),
                }
            }
            Ok(_) => Ok(()),
        }?;

        let start_crawler =
glue.start_crawler().name(self.crawler()).send().await;

        match start_crawler {
            Ok(_) => Ok(()),
            Err(err) => {
                let glue_err: aws_sdk_glue::Error = err.into();
                match glue_err {
                    aws_sdk_glue::Error::CrawlerRunningException(_) => Ok(()),
                    _ => Err(GlueMvpError::GlueSdk(glue_err)),
                }
            }
        }?;
```

AWS Glue Data Catalog 内のデータベースとテーブルに関する情報を一覧表示します。

```
        let database = glue
            .get_database()
            .name(self.database())
            .send()
            .await
            .map_err(GlueMvpError::from_glue_sdk)?
            .to_owned();
        let database = database
            .database()
            .ok_or_else(|| GlueMvpError::Unknown("Could not find
database".into()))?;

        let tables = glue
            .get_tables()
            .database_name(self.database())
            .send()
            .await
            .map_err(GlueMvpError::from_glue_sdk)?;
```

```
        let tables = tables.table_list();
```

ソース Amazon S3 バケットから CSV 形式データを抽出し、フィールドを削除して名前を変更することで変換し、JSON 形式の出力を別の Amazon S3 バケットにロードするジョブを作成して実行します。

```
        let create_job = glue
            .create_job()
            .name(self.job())
            .role(self.iam_role.expose_secret())
            .command(
                JobCommand::builder()
                    .name("glueetl")
                    .python_version("3")
                    .script_location(format!("s3://{}/job.py", self.bucket()))
                    .build(),
            )
            .glue_version("3.0")
            .send()
            .await
            .map_err(GlueMvpError::from_glue_sdk)?;

        let job_name = create_job.name().ok_or_else(|| {
            GlueMvpError::Unknown("Did not get job name after creating
job".into())
        })?;

        let job_run_output = glue
            .start_job_run()
            .job_name(self.job())
            .arguments("--input_database", self.database())
            .arguments(
                "--input_table",
                self.tables
                    .first()
                    .ok_or_else(|| GlueMvpError::Unknown("Missing crawler
table".into()))?
                    .name(),
            )
            .arguments("--output_bucket_url", self.bucket())
            .send()
```

```
            .await
            .map_err(GlueMvpError::from_glue_sdk)?;

        let job = job_run_output
            .job_run_id()
            .ok_or_else(|| GlueMvpError::Unknown("Missing run id from just
started job".into()))?
            .to_string();
```

デモによって作成されたすべてのリソースを削除します。

```
        glue.delete_job()
            .job_name(self.job())
            .send()
            .await
            .map_err(GlueMvpError::from_glue_sdk)?;

        for t in &self.tables {
            glue.delete_table()
                .name(t.name())
                .database_name(self.database())
                .send()
                .await
                .map_err(GlueMvpError::from_glue_sdk)?;
        }

        glue.delete_database()
            .name(self.database())
            .send()
            .await
            .map_err(GlueMvpError::from_glue_sdk)?;

        glue.delete_crawler()
            .name(self.crawler())
            .send()
            .await
            .map_err(GlueMvpError::from_glue_sdk)?;
```

- API の詳細については、「AWS SDK for Rust API リファレンス」の以下のトピックを参照してください。

- CreateCrawler
- CreateJob
- DeleteCrawler
- DeleteDatabase
- DeleteJob
- DeleteTable
- GetCrawler
- GetDatabase
- GetDatabases
- GetJob
- GetJob
- GetJobRuns
- GetTables
- ListJobs
- StartCrawler
- StartJobRun

Swift

SDK for Swift

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

`Package.swift` ファイル。

```
// swift-tools-version: 5.9
//
// The swift-tools-version declares the minimum version of Swift required to
// build this package.

import PackageDescription
```

```
let package = Package(
    name: "glue-scenario",
    // Let Xcode know the minimum Apple platforms supported.
    platforms: [
        .macOS(.v13),
        .iOS(.v15)
    ],
    dependencies: [
        // Dependencies declare other packages that this package depends on.
        .package(
            url: "https://github.com/awslabs/aws-sdk-swift",
            from: "1.0.0"),
        .package(
            url: "https://github.com/apple/swift-argument-parser.git",
            branch: "main"
        )
    ],
    targets: [
        // Targets are the basic building blocks of a package, defining a module
 or a test suite.
        // Targets can depend on other targets in this package and products
        // from dependencies.
        .executableTarget(
            name: "glue-scenario",
            dependencies: [
                .product(name: "AWSGlue", package: "aws-sdk-swift"),
                .product(name: "AWSS3", package: "aws-sdk-swift"),
                .product(name: "ArgumentParser", package: "swift-argument-
parser")
            ],
            path: "Sources")

    ]
)
```

Swift コードファイル、`entry.swift`。

```
// An example that shows how to use the AWS SDK for Swift to demonstrate
// creating and using crawlers and jobs using AWS Glue.
//
// 0. Upload the Python job script to Amazon S3 so it can be used when
```

```
//    calling `startJobRun()` later.
// 1. Create a crawler, pass it the IAM role and the URL of the public Amazon
//    S3 bucket that contains the source data:
//    s3://crawler-public-us-east-1/flight/2016/csv.
// 2. Start the crawler. This takes time, so after starting it, use a loop
//    that calls `getCrawler()` until the state is "READY".
// 3. Get the database created by the crawler, and the tables in the
//    database. Display them to the user.
// 4. Create a job. Pass it the IAM role and the URL to a Python ETL script
//    previously uploaded to the user's S3 bucket.
// 5. Start a job run, passing the following custom arguments. These are
//    expected by the ETL script, so must exactly match.
//    * `--input_database: <name of the database created by the crawler>`
//    * `--input_table: <name of the table created by the crawler>`
//    * `--output_bucket_url: <URL to the scaffold bucket created for the
//      user>`
// 6. Loop and get the job run until it returns one of the following states:
//    "SUCCEEDED", "STOPPED", "FAILED", or "TIMEOUT".
// 7. Output data is stored in a group of files in the user's S3 bucket.
//    Either direct the user to their location or download a file and display
//    the results inline.
// 8. List the jobs for the user's account.
// 9. Get job run details for a job run.
// 10. Delete the demo job.
// 11. Delete the database and tables created by the example.
// 12. Delete the crawler created by the example.

import ArgumentParser
import AWSS3
import Foundation
import Smithy

import AWSClientRuntime
import AWSGlue

struct ExampleCommand: ParsableCommand {
    @Option(help: "The AWS IAM role to use for AWS Glue calls.")
    var role: String

    @Option(help: "The Amazon S3 bucket to use for this example.")
    var bucket: String

    @Option(help: "The Amazon S3 URL of the data to crawl.")
    var s3url: String = "s3://crawler-public-us-east-1/flight/2016/csv"
```

```
    @Option(help: "The Python script to run as a job with AWS Glue.")
    var script: String = "./flight_etl_job_script.py"

    @Option(help: "The AWS Region to run AWS API calls in.")
    var awsRegion = "us-east-1"

    @Option(help: "A prefix string to use when naming tables.")
    var tablePrefix = "swift-glue-basics-table"

    @Option(
        help: ArgumentHelp("The level of logging for the Swift SDK to perform."),
        completion: .list([
            "critical",
            "debug",
            "error",
            "info",
            "notice",
            "trace",
            "warning"
        ])
    )
    var logLevel: String = "error"

    static var configuration = CommandConfiguration(
        commandName: "glue-scenario",
        abstract: """
        Demonstrates various features of AWS Glue.
        """,
        discussion: """
        An example showing how to use AWS Glue to create, run, and monitor
        crawlers and jobs.
        """
    )

    /// Generate and return a unique file name that begins with the specified
    /// string.
    ///
    /// - Parameters:
    ///   - prefix: Text to use at the beginning of the returned name.
    ///
    /// - Returns: A string containing a unique filename that begins with the
    ///   specified `prefix`.
    ///
```

```
    /// The returned name uses a random number between 1 million and 1 billion to
    /// provide reasonable certainty of uniqueness for the purposes of this
    /// example.
    func tempName(prefix: String) -> String {
        return "\(prefix)-\(Int.random(in: 1000000..<1000000000))"
    }

    /// Upload a file to an Amazon S3 bucket.
    ///
    /// - Parameters:
    ///   - s3Client: The S3 client to use when uploading the file.
    ///   - path: The local path of the source file to upload.
    ///   - toBucket: The name of the S3 bucket into which to upload the file.
    ///   - key: The key (name) to give the file in the S3 bucket.
    ///
    /// - Returns: `true` if the file is uploaded successfully, otherwise
`false`.
    func uploadFile(s3Client: S3Client, path: String, toBucket: String, key:
String) async -> Bool {
        do {
            let fileData: Data = try Data(contentsOf: URL(fileURLWithPath: path))
            let dataStream = ByteStream.data(fileData)
            _ = try await s3Client.putObject(
                input: PutObjectInput(
                    body: dataStream,
                    bucket: toBucket,
                    key: key
                )
            )
        } catch {
            print("*** An unexpected error occurred uploading the script to the
Amazon S3 bucket \"\(bucket)\".")
            return false
        }

        return true
    }

    /// Create a new AWS Glue crawler.
    ///
    /// - Parameters:
    ///   - glueClient: An AWS Glue client to use for the crawler.
    ///   - crawlerName: A name for the new crawler.
    ///   - iamRole: The name of an Amazon IAM role for the crawler to use.
```

```
    ///   - s3Path: The path of an Amazon S3 folder to use as a target location.
    ///   - cronSchedule: A `cron` schedule indicating when to run the crawler.
    ///   - databaseName: The name of an AWS Glue database to operate on.
    ///
    /// - Returns: `true` if the crawler is created successfully, otherwise
`false`.
    func createCrawler(glueClient: GlueClient, crawlerName: String, iamRole:
String,
                       s3Path: String, cronSchedule: String, databaseName:
String) async -> Bool {
        let s3Target = GlueClientTypes.S3Target(path: s3url)
        let targetList = GlueClientTypes.CrawlerTargets(s3Targets: [s3Target])

        do {
            _ = try await glueClient.createCrawler(
                input: CreateCrawlerInput(
                    databaseName: databaseName,
                    description: "Created by the AWS SDK for Swift Scenario
Example for AWS Glue.",
                    name: crawlerName,
                    role: iamRole,
                    schedule: cronSchedule,
                    tablePrefix: tablePrefix,
                    targets: targetList
                )
            )
        } catch _ as AlreadyExistsException {
            print("*** A crawler named \"\(crawlerName)\" already exists.")
            return false
        } catch _ as OperationTimeoutException {
            print("*** The attempt to create the AWS Glue crawler timed out.")
            return false
        } catch {
            print("*** An unexpected error occurred creating the AWS Glue
crawler: \(error.localizedDescription)")
            return false
        }

        return true
    }

    /// Delete an AWS Glue crawler.
    ///
    /// - Parameters:
```

```
    ///   - glueClient: The AWS Glue client to use.
    ///   - name: The name of the crawler to delete.
    ///
    /// - Returns: `true` if successful, otherwise `false`.
    func deleteCrawler(glueClient: GlueClient, name: String) async -> Bool {
        do {
            _ = try await glueClient.deleteCrawler(
                input: DeleteCrawlerInput(name: name)
            )
        } catch {
            return false
        }
        return true
    }

    /// Start running an AWS Glue crawler.
    ///
    /// - Parameters:
    ///   - glueClient: The AWS Glue client to use when starting the crawler.
    ///   - name: The name of the crawler to start running.
    ///
    /// - Returns: `true` if the crawler is started successfully, otherwise
`false`.
    func startCrawler(glueClient: GlueClient, name: String) async -> Bool {
        do {
            _ = try await glueClient.startCrawler(
                input: StartCrawlerInput(name: name)
            )
        } catch {
            print("*** An unexpected error occurred starting the crawler.")
            return false
        }

        return true
    }

    /// Get the state of the specified AWS Glue crawler.
    ///
    /// - Parameters:
    ///   - glueClient: The AWS Glue client to use.
    ///   - name: The name of the crawler whose state should be returned.
    ///
    /// - Returns: A `GlueClientTypes.CrawlerState` value describing the
    ///   state of the crawler.
```

```
   func getCrawlerState(glueClient: GlueClient, name: String) async ->
GlueClientTypes.CrawlerState {
       do {
           let output = try await glueClient.getCrawler(
               input: GetCrawlerInput(name: name)
           )

           // If the crawler or its state is `nil`, report that the crawler
           // is stopping. This may not be what you want for your
           // application but it works for this one!

           guard let crawler = output.crawler else {
               return GlueClientTypes.CrawlerState.stopping
           }
           guard let state = crawler.state else {
               return GlueClientTypes.CrawlerState.stopping
           }
           return state
       } catch {
           return GlueClientTypes.CrawlerState.stopping
       }
   }

   /// Wait until the specified crawler is ready to run.
   ///
   /// - Parameters:
   ///   - glueClient: The AWS Glue client to use.
   ///   - name: The name of the crawler to wait for.
   ///
   /// - Returns: `true` if the crawler is ready, `false` if the client is
   ///   stopping (and will therefore never be ready).
   func waitUntilCrawlerReady(glueClient: GlueClient, name: String) async ->
Bool {
       while true {
           let state = await getCrawlerState(glueClient: glueClient, name: name)

           if state == .ready {
               return true
           } else if state == .stopping {
               return false
           }
           Thread.sleep(forTimeInterval: 4)
       }
   }
```

```
    /// Create a new AWS Glue job.
    ///
    /// - Parameters:
    ///   - glueClient: The AWS Glue client to use.
    ///   - jobName: The name to give the new job.
    ///   - role: The IAM role for the job to use when accessing AWS services.
    ///   - scriptLocation: The AWS S3 URI of the script to be run by the job.
    ///
    /// - Returns: `true` if the job is created successfully, otherwise `false`.
    func createJob(glueClient: GlueClient, name jobName: String, role: String,
                   scriptLocation: String) async -> Bool {
        let command = GlueClientTypes.JobCommand(
            name: "glueetl",
            pythonVersion: "3",
            scriptLocation: scriptLocation
        )

        do {
            _ = try await glueClient.createJob(
                input: CreateJobInput(
                    command: command,
                    description: "Created by the AWS SDK for Swift Glue basic
scenario example.",
                    glueVersion: "3.0",
                    name: jobName,
                    numberOfWorkers: 10,
                    role: role,
                    workerType: .g1x
                )
            )
        } catch {
            return false
        }
        return true
    }

    /// Return a list of the AWS Glue jobs listed on the user's account.
    ///
    /// - Parameters:
    ///   - glueClient: The AWS Glue client to use.
    ///   - maxJobs: The maximum number of jobs to return (default: 100).
    ///
    /// - Returns: An array of strings listing the names of all available AWS
```

```
    ///   Glue jobs.
    func listJobs(glueClient: GlueClient, maxJobs: Int = 100) async -> [String] {
        var jobList: [String] = []
        var nextToken: String?

        repeat {
            do {
                let output = try await glueClient.listJobs(
                    input: ListJobsInput(
                        maxResults: maxJobs,
                        nextToken: nextToken
                    )
                )

                guard let jobs = output.jobNames else {
                    return jobList
                }

                jobList = jobList + jobs
                nextToken = output.nextToken
            } catch {
                return jobList
            }
        } while (nextToken != nil)

        return jobList
    }

    /// Delete an AWS Glue job.
    ///
    /// - Parameters:
    ///   - glueClient: The AWS Glue client to use.
    ///   - jobName: The name of the job to delete.
    ///
    /// - Returns: `true` if the job is successfully deleted, otherwise `false`.
    func deleteJob(glueClient: GlueClient, name jobName: String) async -> Bool {
        do {
            _ = try await glueClient.deleteJob(
                input: DeleteJobInput(jobName: jobName)
            )
        } catch {
            return false
        }
        return true
```

```
    }

    /// Create an AWS Glue database.
    ///
    /// - Parameters:
    ///   - glueClient: The AWS Glue client to use.
    ///   - databaseName: The name to give the new database.
    ///   - location: The URL of the source data to use with AWS Glue.
    ///
    /// - Returns: `true` if the database is created successfully, otherwise
`false`.
    func createDatabase(glueClient: GlueClient, name databaseName: String,
location: String) async -> Bool {
        let databaseInput = GlueClientTypes.DatabaseInput(
            description: "Created by the AWS SDK for Swift Glue basic scenario
example.",
            locationUri: location,
            name: databaseName
        )

        do {
            _ = try await glueClient.createDatabase(
                input: CreateDatabaseInput(
                    databaseInput: databaseInput
                )
            )
        } catch {
            return false
        }

        return true
    }

    /// Get the AWS Glue database with the specified name.
    ///
    /// - Parameters:
    ///   - glueClient: The AWS Glue client to use.
    ///   - name: The name of the database to return.
    ///
    /// - Returns: The `GlueClientTypes.Database` object describing the
    ///   specified database, or `nil` if an error occurs or the database
    ///   isn't found.
    func getDatabase(glueClient: GlueClient, name: String) async ->
GlueClientTypes.Database? {
```

```
        do {
            let output = try await glueClient.getDatabase(
                input: GetDatabaseInput(name: name)
            )

            return output.database
        } catch {
            return nil
        }
    }

    /// Returns a list of the tables in the specified database.
    ///
    /// - Parameters:
    ///   - glueClient: The AWS Glue client to use.
    ///   - databaseName: The name of the database whose tables are to be
    ///     returned.
    ///
    /// - Returns: An array of `GlueClientTypes.Table` objects, each
    ///   describing one table in the named database. An empty array indicates
    ///   that there are either no tables in the database, or an error
    ///   occurred before any tables could be found.
    func getTablesInDatabase(glueClient: GlueClient, databaseName: String) async
-> [GlueClientTypes.Table] {
        var tables: [GlueClientTypes.Table] = []
        var nextToken: String?

        repeat {
            do {
                let output = try await glueClient.getTables(
                    input: GetTablesInput(
                        databaseName: databaseName,
                        nextToken: nextToken
                    )
                )

                guard let tableList = output.tableList else {
                    return tables
                }

                tables = tables + tableList
                nextToken = output.nextToken
            } catch {
                return tables
```

```
            }
        } while nextToken != nil

        return tables
    }

    /// Delete the specified database.
    ///
    /// - Parameters:
    ///   - glueClient: The AWS Glue client to use.
    ///   - databaseName: The name of the database to delete.
    ///   - deleteTables: A Bool indicating whether or not to delete the
    ///     tables in the database before attempting to delete the database.
    ///
    /// - Returns: `true` if the database (and optionally its tables) are
    ///   deleted, otherwise `false`.
    func deleteDatabase(glueClient: GlueClient, name databaseName: String,
                        withTables deleteTables: Bool = false) async -> Bool {
        if deleteTables {
            var tableNames: [String] = []

            // Get a list of the names of all of the tables in the database.

            let tableList = await self.getTablesInDatabase(glueClient:
glueClient, databaseName: databaseName)
            for table in tableList {
                guard let name = table.name else {
                    continue
                }
                tableNames.append(name)
            }

            // Delete the tables.

            do {
                _ = try await glueClient.batchDeleteTable(
                    input: BatchDeleteTableInput(
                        databaseName: databaseName,
                        tablesToDelete: tableNames
                    )
                )
            } catch {
                print("*** Unable to delete the tables.")
            }
```

```
            return true
        }

        // Delete the database itself.

        do {
            _ = try await glueClient.deleteDatabase(
                input: DeleteDatabaseInput(name: databaseName)
            )
        } catch {
            print("*** Unable to delete the database.")
        }
        return true
    }

    /// Start an AWS Glue job run.
    ///
    /// - Parameters:
    ///   - glueClient: The AWS Glue client to use.
    ///   - jobName: The name of the job to run.
    ///   - databaseName: The name of the AWS Glue database to run the job
against.
    ///   - tableName: The name of the table in the database to run the job
against.
    ///   - outputURL: The AWS S3 URI of the bucket location into which to
    ///     write the resulting output.
    ///
    /// - Returns: `true` if the job run is started successfully, otherwise
`false`.
    func startJobRun(glueClient: GlueClient, name jobName: String, databaseName:
String,
                     tableName: String, outputURL: String) async -> String? {
        do {
            let output = try await glueClient.startJobRun(
                input: StartJobRunInput(
                    arguments: [
                        "--input_database": databaseName,
                        "--input_table": tableName,
                        "--output_bucket_url": outputURL
                    ],
                    jobName: jobName,
                    numberOfWorkers: 10,
                    workerType: .g1x
                )
```

```
            )

            guard let id = output.jobRunId else {
                return nil
            }

            return id
        } catch {
            return nil
        }
    }

    /// Get information about a specific AWS Glue job run.
    ///
    /// - Parameters:
    ///   - glueClient: The AWS Glue client to use.
    ///   - jobName: The name of the job to return job run data for.
    ///   - id: The run ID of the specific job run to return.
    ///
    /// - Returns: A `GlueClientTypes.JobRun` object describing the state of
    ///   the job run, or `nil` if an error occurs.
    func getJobRun(glueClient: GlueClient, name jobName: String, id: String)
async -> GlueClientTypes.JobRun? {
        do {
            let output = try await glueClient.getJobRun(
                input: GetJobRunInput(
                    jobName: jobName,
                    runId: id
                )
            )

            return output.jobRun
        } catch {
            return nil
        }
    }

    /// Called by ``main()`` to run the bulk of the example.
    func runAsync() async throws {
        // A name to give the Python script upon upload to the Amazon S3
        // bucket.
        let scriptName = "jobscript.py"

        // Schedule string in `cron` format, as described here:
```

```
        // https://docs.aws.amazon.com/glue/latest/dg/monitor-data-warehouse-
schedule.html
        let cron = "cron(15 12 * * ? *)"

        let glueConfig = try await GlueClient.GlueClientConfiguration(region:
 awsRegion)
        let glueClient = GlueClient(config: glueConfig)

        let s3Config = try await S3Client.S3ClientConfiguration(region:
 awsRegion)
        let s3Client = S3Client(config: s3Config)

        // Create random names for things that need them.

        let crawlerName = tempName(prefix: "swift-glue-basics-crawler")
        let databaseName = tempName(prefix: "swift-glue-basics-db")

        // Create a name for the AWS Glue job.

        let jobName = tempName(prefix: "scenario-job")

        // The URL of the Python script on S3.

        let scriptURL = "s3://\(bucket)/\(scriptName)"

        print("Welcome to the AWS SDK for Swift basic scenario for AWS Glue!")

        //===========================================================================
        // 0. Upload the Python script to the target bucket so it's available
        //    for use by the Amazon Glue service.
        //===========================================================================

        print("Uploading the Python script: \(script) as key \(scriptName)")
        print("Destination bucket: \(bucket)")
        if !(await uploadFile(s3Client: s3Client, path: script, toBucket: bucket,
 key: scriptName)) {
            return
        }

        //===========================================================================
        // 1. Create the database and crawler using the randomized names
        //    generated previously.
        //===========================================================================
```

```
        print("Creating database \"\(databaseName)\"...")
        if !(await createDatabase(glueClient: glueClient, name: databaseName,
location: s3url)) {
            print("*** Unable to create the database.")
            return
        }

        print("Creating crawler \"\(crawlerName)\"...")
        if !(await createCrawler(glueClient: glueClient, crawlerName:
crawlerName,
                                 iamRole: role, s3Path: s3url, cronSchedule:
cron,
                                 databaseName: databaseName)) {
            return
        }

        //=========================================================================
        // 2. Start the crawler, then wait for it to be ready.
        //=========================================================================

        print("Starting the crawler and waiting until it's ready...")
        if !(await startCrawler(glueClient: glueClient, name: crawlerName)) {
            _ = await deleteCrawler(glueClient: glueClient, name: crawlerName)
            return
        }

        if !(await waitUntilCrawlerReady(glueClient: glueClient, name:
crawlerName)) {
            _ = await deleteCrawler(glueClient: glueClient, name: crawlerName)
        }

        //=========================================================================
        // 3. Get the database and table created by the crawler.
        //=========================================================================

        print("Getting the crawler's database...")
        let database = await getDatabase(glueClient: glueClient, name:
databaseName)
        let tableList = await getTablesInDatabase(glueClient: glueClient,
databaseName: databaseName)

        print("Found \(tableList.count) table(s):")
        for table in tableList {
            print("  \(table.name ?? "<unnamed>")")
```

```
        }

        if tableList.count != 1 {
            print("*** Incorrect number of tables found. There should only be
one.")
            _ = await deleteDatabase(glueClient: glueClient, name: databaseName,
withTables: true)
            _ = await deleteCrawler(glueClient: glueClient, name: crawlerName)
            return
        }

        guard let tableName = tableList[0].name else {
            print("*** Table is unnamed.")
            _ = await deleteDatabase(glueClient: glueClient, name: databaseName,
withTables: true)
            _ = await deleteCrawler(glueClient: glueClient, name: crawlerName)
            return
        }

        //====================================================================
        // 4. Create a job.
        //====================================================================

        print("Creating a job...")
        if !(await createJob(glueClient: glueClient, name: jobName, role: role,
                             scriptLocation: scriptURL)) {
            _ = await deleteDatabase(glueClient: glueClient, name: databaseName,
withTables: true)
            _ = await deleteCrawler(glueClient: glueClient, name: crawlerName)
            return
        }

        //====================================================================
        // 5. Start a job run.
        //====================================================================

        print("Starting the job...")

        // Construct the Amazon S3 URL for the job run's output. This is in
        // the bucket specified on the command line, with a folder name that's
        // unique for this job run.

        let timeStamp = Date().timeIntervalSince1970
        let jobPath = "\(jobName)-\(Int(timeStamp))"
```

```
        let outputURL = "s3://\(bucket)/\(jobPath)"

        // Start the job run.

        let jobRunID = await startJobRun(glueClient: glueClient, name: jobName,
                                         databaseName: databaseName,
                                         tableName: tableName,
                                         outputURL: outputURL)

        guard let jobRunID else {
            print("*** Job run ID is invalid.")
            _ = await deleteJob(glueClient: glueClient, name: jobName)
            _ = await deleteDatabase(glueClient: glueClient, name: databaseName,
withTables: true)
            _ = await deleteCrawler(glueClient: glueClient, name: crawlerName)
            return
        }

        //=====================================================================
        // 6. Wait for the job run to indicate that the run is complete.
        //=====================================================================

        print("Waiting for job run to end...")

        var jobRunFinished = false
        var jobRunState: GlueClientTypes.JobRunState

        repeat {
            let jobRun = await getJobRun(glueClient: glueClient, name: jobName,
id: jobRunID)
            guard let jobRun else {
                print("*** Unable to get the job run.")
                _ = await deleteJob(glueClient: glueClient, name: jobName)
                _ = await deleteDatabase(glueClient: glueClient, name:
databaseName, withTables: true)
                _ = await deleteCrawler(glueClient: glueClient, name:
crawlerName)
                return
            }
            jobRunState = jobRun.jobRunState ?? .failed

            //
=====================================================================
            // 7. Output where to find the data if the job run was successful.
```

```
            //    If the job run failed for any reason, output an appropriate
            //    error message.
            //
================================================================================

            switch jobRunState {
                case .succeeded:
                    print("Job run succeeded. JSON files are in the Amazon S3
 path:")
                    print("    \(outputURL)")
                    jobRunFinished = true
                case .stopped:
                    jobRunFinished = true
                case .error:
                    print("*** Error: Job run ended in an error.
 \(jobRun.errorMessage ?? "")")
                    jobRunFinished = true
                case .failed:
                    print("*** Error: Job run failed. \(jobRun.errorMessage ??
 "")")
                    jobRunFinished = true
                case .timeout:
                    print("*** Warning: Job run timed out.")
                    jobRunFinished = true
                default:
                    Thread.sleep(forTimeInterval: 0.25)
            }
        } while jobRunFinished != true

        //====================================================================
        // 8. List the jobs for the user's account.
        //====================================================================

        print("\nThe account has the following jobs:")
        let jobs = await listJobs(glueClient: glueClient)

        if jobs.count == 0 {
            print("    <no jobs found>")
        } else {
            for job in jobs {
                print("    \(job)")
            }
        }
```

```
        //=======================================================================
        // 9. Get the job run details for a job run.
        //=======================================================================

        print("Information about the job run:")
        let jobRun = await getJobRun(glueClient: glueClient, name: jobName, id:
jobRunID)

        guard let jobRun else {
            print("*** Unable to retrieve the job run.")
            _ = await deleteJob(glueClient: glueClient, name: jobName)
            _ = await deleteDatabase(glueClient: glueClient, name: databaseName,
withTables: true)
            _ = await deleteCrawler(glueClient: glueClient, name: crawlerName)
            return
        }

        let startDate = jobRun.startedOn ?? Date(timeIntervalSince1970: 0)
        let endDate = jobRun.completedOn ?? Date(timeIntervalSince1970: 0)
        let dateFormatter: DateFormatter = DateFormatter()
        dateFormatter.dateStyle = .long
        dateFormatter.timeStyle = .long

        print("    Started at: \(dateFormatter.string(from: startDate))")
        print("  Completed at: \(dateFormatter.string(from: endDate))")

        //=======================================================================
        // 10. Delete the job.
        //=======================================================================

        print("\nDeleting the job...")
        _ = await deleteJob(glueClient: glueClient, name: jobName)

        //=======================================================================
        // 11. Delete the database and tables created by this example.
        //=======================================================================

        print("Deleting the database...")
        _ = await deleteDatabase(glueClient: glueClient, name: databaseName,
withTables: true)

        //=======================================================================
        // 12. Delete the crawler.
        //=======================================================================
```

```
        print("Deleting the crawler...")
        if !(await deleteCrawler(glueClient: glueClient, name: crawlerName)) {
            return
        }
    }
}

/// The program's asynchronous entry point.
@main
struct Main {
    static func main() async {
        let args = Array(CommandLine.arguments.dropFirst())

        do {
            let command = try ExampleCommand.parse(args)
            try await command.runAsync()
        } catch {
            ExampleCommand.exit(withError: error)
        }
    }
}
```

- API の詳細については、AWS SDK for Swift API リファレンスの以下のトピックを参照してください。
  - CreateCrawler
  - CreateJob
  - DeleteCrawler
  - DeleteDatabase
  - DeleteJob
  - DeleteTable
  - GetCrawler
  - GetDatabase
  - GetDatabases
  - GetJob
  - GetJob
  - GetJobRuns

- GetTables
- ListJobs
- StartCrawler
- StartJobRun

AWS SDK デベロッパーガイドとコード例の詳細なリストについては、「このサービスを AWS SDK で使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK を使用した AWS Glue 向けアクション

以下は、AWS SDK を使用して個々の AWS Glue アクションを実行する方法を説明するコード例です。それぞれの例には、GitHub へのリンクがあり、そこにはコードの設定と実行に関する説明が記載されています。

以下の例には、最も一般的に使用されるアクションのみ含まれています。詳細な一覧については、「AWS Glue API リファレンス」を参照してください。

例

- AWS SDK で CreateCrawler を使用する
- AWS SDK または CLI で CreateJob を使用する
- AWS SDK で DeleteCrawler を使用する
- AWS SDK で DeleteDatabase を使用する
- AWS SDK または CLI で DeleteJob を使用する
- AWS SDK で DeleteTable を使用する
- AWS SDK で GetCrawler を使用する
- AWS SDK で GetDatabase を使用する
- AWS SDK または CLI で GetDatabases を使用する
- AWS SDK または CLI で GetJob を使用する
- AWS SDK または CLI で GetJobRun を使用する
- AWS SDK または CLI で GetJobRuns を使用する
- AWS SDK または CLI で GetTables を使用する
- AWS SDK で ListJobs を使用する

- AWS SDK または CLI で StartCrawler を使用する
- AWS SDK または CLI で StartJobRun を使用する

AWS SDK で **CreateCrawler** を使用する

次のサンプルコードは、CreateCrawler を使用する方法を説明しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- 基本を学ぶ

.NET

SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Create an AWS Glue crawler.
    /// </summary>
    /// <param name="crawlerName">The name for the crawler.</param>
    /// <param name="crawlerDescription">A description of the crawler.</param>
    /// <param name="role">The AWS Identity and Access Management (IAM) role to
    /// be assumed by the crawler.</param>
    /// <param name="schedule">The schedule on which the crawler will be
executed.</param>
    /// <param name="s3Path">The path to the Amazon Simple Storage Service
(Amazon S3)
    /// bucket where the Python script has been stored.</param>
    /// <param name="dbName">The name to use for the database that will be
    /// created by the crawler.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> CreateCrawlerAsync(
        string crawlerName,
        string crawlerDescription,
```

```
        string role,
        string schedule,
        string s3Path,
        string dbName)
    {
        var s3Target = new S3Target
        {
            Path = s3Path,
        };

        var targetList = new List<S3Target>
        {
            s3Target,
        };

        var targets = new CrawlerTargets
        {
            S3Targets = targetList,
        };

        var crawlerRequest = new CreateCrawlerRequest
        {
            DatabaseName = dbName,
            Name = crawlerName,
            Description = crawlerDescription,
            Targets = targets,
            Role = role,
            Schedule = schedule,
        };

        var response = await _amazonGlue.CreateCrawlerAsync(crawlerRequest);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「CreateCrawler」を参照してください。

C++

SDK for C++

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
       Aws::Client::ClientConfiguration clientConfig;
       // Optional: Set to the AWS Region in which the bucket was created
(overrides config file).
       // clientConfig.region = "us-east-1";

   Aws::Glue::GlueClient client(clientConfig);

       Aws::Glue::Model::S3Target s3Target;
       s3Target.SetPath("s3://crawler-public-us-east-1/flight/2016/csv");
       Aws::Glue::Model::CrawlerTargets crawlerTargets;
       crawlerTargets.AddS3Targets(s3Target);

       Aws::Glue::Model::CreateCrawlerRequest request;
       request.SetTargets(crawlerTargets);
       request.SetName(CRAWLER_NAME);
       request.SetDatabaseName(CRAWLER_DATABASE_NAME);
       request.SetTablePrefix(CRAWLER_DATABASE_PREFIX);
       request.SetRole(roleArn);

       Aws::Glue::Model::CreateCrawlerOutcome outcome =
client.CreateCrawler(request);

       if (outcome.IsSuccess()) {
           std::cout << "Successfully created the crawler." << std::endl;
       }
       else {
           std::cerr << "Error creating a crawler. " <<
outcome.GetError().GetMessage()
                     << std::endl;
           deleteAssets("", CRAWLER_DATABASE_NAME, "", bucketName,
clientConfig);
           return false;
```

```
        }
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「CreateCrawler」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
   /**
    * Creates a new AWS Glue crawler using the AWS Glue Java API.
    *
    * @param glueClient  the AWS Glue client used to interact with the AWS Glue
service
    * @param iam         the IAM role that the crawler will use to access the
data source
    * @param s3Path      the S3 path that the crawler will scan for data
    * @param cron        the cron expression that defines the crawler's schedule
    * @param dbName      the name of the AWS Glue database where the crawler
will store the metadata
    * @param crawlerName the name of the crawler to be created
    */
   public static void createGlueCrawler(GlueClient glueClient,
                                         String iam,
                                         String s3Path,
                                         String cron,
                                         String dbName,
                                         String crawlerName) {

       try {
           S3Target s3Target = S3Target.builder()
               .path(s3Path)
               .build();
```

```
            List<S3Target> targetList = new ArrayList<>();
            targetList.add(s3Target);
            CrawlerTargets targets = CrawlerTargets.builder()
                .s3Targets(targetList)
                .build();

            CreateCrawlerRequest crawlerRequest = CreateCrawlerRequest.builder()
                .databaseName(dbName)
                .name(crawlerName)
                .description("Created by the AWS Glue Java API")
                .targets(targets)
                .role(iam)
                .schedule(cron)
                .build();

            glueClient.createCrawler(crawlerRequest);
            System.out.println(crawlerName + " was successfully created");

        } catch (GlueException e) {
            throw e;
        }
    }
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「CreateCrawler」を参照してください。

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
const createCrawler = (name, role, dbName, tablePrefix, s3TargetPath) => {
  const client = new GlueClient({});
```

```
  const command = new CreateCrawlerCommand({
    Name: name,
    Role: role,
    DatabaseName: dbName,
    TablePrefix: tablePrefix,
    Targets: {
      S3Targets: [{ Path: s3TargetPath }],
    },
  });

  return client.send(command);
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「CreateCrawler」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
suspend fun createGlueCrawler(
    iam: String?,
    s3Path: String?,
    cron: String?,
    dbName: String?,
    crawlerName: String,
) {
    val s3Target =
        S3Target {
            path = s3Path
        }

    // Add the S3Target to a list.
    val targetList = mutableListOf<S3Target>()
```

```
    targetList.add(s3Target)

    val targetOb =
        CrawlerTargets {
            s3Targets = targetList
        }

    val request =
        CreateCrawlerRequest {
            databaseName = dbName
            name = crawlerName
            description = "Created by the AWS Glue Kotlin API"
            targets = targetOb
            role = iam
            schedule = cron
        }

    GlueClient { region = "us-west-2" }.use { glueClient ->
        glueClient.createCrawler(request)
        println("$crawlerName was successfully created")
    }
}
```

- API の詳細については、「AWS SDK for Kotlin API リファレンス」の「CreateCrawler」を参照してください。

PHP

SDK for PHP

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

```
        $crawlerName = "example-crawler-test-" . $uniqid;

        $role = $iamService->getRole("AWSGlueServiceRole-DocExample");
```

```
        $path = 's3://crawler-public-us-east-1/flight/2016/csv';
        $glueService->createCrawler($crawlerName, $role['Role']['Arn'],
$databaseName, $path);

    public function createCrawler($crawlerName, $role, $databaseName, $path):
Result
    {
        return $this->customWaiter(function () use ($crawlerName, $role,
$databaseName, $path) {
            return $this->glueClient->createCrawler([
                'Name' => $crawlerName,
                'Role' => $role,
                'DatabaseName' => $databaseName,
                'Targets' => [
                    'S3Targets' =>
                        [[
                            'Path' => $path,
                        ]]
                ],
            ]);
        });
    }
```

- API の詳細については、「AWS SDK for PHP API リファレンス」の「CreateCrawler」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""

    def __init__(self, glue_client):
```

```
        """
        :param glue_client: A Boto3 Glue client.
        """
        self.glue_client = glue_client


    def create_crawler(self, name, role_arn, db_name, db_prefix, s3_target):
        """
        Creates a crawler that can crawl the specified target and populate a
        database in your AWS Glue Data Catalog with metadata that describes the
data
        in the target.

        :param name: The name of the crawler.
        :param role_arn: The Amazon Resource Name (ARN) of an AWS Identity and
Access
                         Management (IAM) role that grants permission to let AWS
Glue
                         access the resources it needs.
        :param db_name: The name to give the database that is created by the
crawler.
        :param db_prefix: The prefix to give any database tables that are created
by
                          the crawler.
        :param s3_target: The URL to an S3 bucket that contains data that is
                          the target of the crawler.
        """
        try:
            self.glue_client.create_crawler(
                Name=name,
                Role=role_arn,
                DatabaseName=db_name,
                TablePrefix=db_prefix,
                Targets={"S3Targets": [{"Path": s3_target}]},
            )
        except ClientError as err:
            logger.error(
                "Couldn't create crawler. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- API の詳細については、「AWS SDK for Python (Boto3) API リファレンス」の「CreateCrawler」を参照してください。

Ruby

SDK for Ruby

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
# The `GlueWrapper` class serves as a wrapper around the AWS Glue API, providing
 a simplified interface for common operations.
# It encapsulates the functionality of the AWS SDK for Glue and provides methods
 for interacting with Glue crawlers, databases, tables, jobs, and S3 resources.
# The class initializes with a Glue client and a logger, allowing it to make API
 calls and log any errors or informational messages.
class GlueWrapper
  def initialize(glue_client, logger)
    @glue_client = glue_client
    @logger = logger
  end

  # Creates a new crawler with the specified configuration.
  #
  # @param name [String] The name of the crawler.
  # @param role_arn [String] The ARN of the IAM role to be used by the crawler.
  # @param db_name [String] The name of the database where the crawler stores its
 metadata.
  # @param db_prefix [String] The prefix to be added to the names of tables that
 the crawler creates.
  # @param s3_target [String] The S3 path that the crawler will crawl.
  # @return [void]
  def create_crawler(name, role_arn, db_name, _db_prefix, s3_target)
    @glue_client.create_crawler(
      name: name,
```

```
      role: role_arn,
      database_name: db_name,
      targets: {
        s3_targets: [
          {
            path: s3_target
          }
        ]
      }
    )
  rescue Aws::Glue::Errors::GlueException => e
    @logger.error("Glue could not create crawler: \n#{e.message}")
    raise
  end
```

- API の詳細については、「AWS SDK for Ruby API リファレンス」の「CreateCrawler」を参照してください。

Rust

SDK for Rust

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
        let create_crawler = glue
            .create_crawler()
            .name(self.crawler())
            .database_name(self.database())
            .role(self.iam_role.expose_secret())
            .targets(
                CrawlerTargets::builder()
                    .s3_targets(S3Target::builder().path(CRAWLER_TARGET).build())
                    .build(),
            )
            .send()
            .await;
```

```
        match create_crawler {
            Err(err) => {
                let glue_err: aws_sdk_glue::Error = err.into();
                match glue_err {
                    aws_sdk_glue::Error::AlreadyExistsException(_) => {
                        info!("Using existing crawler");
                        Ok(())
                    }
                    _ => Err(GlueMvpError::GlueSdk(glue_err)),
                }
            }
            Ok(_) => Ok(()),
        }?;
```

- API の詳細については、「AWS SDK for Rust API リファレンス」の「CreateCrawler」を参照してください。

Swift

SDK for Swift

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
import AWSClientRuntime
import AWSGlue

    /// Create a new AWS Glue crawler.
    ///
    /// - Parameters:
    ///   - glueClient: An AWS Glue client to use for the crawler.
    ///   - crawlerName: A name for the new crawler.
    ///   - iamRole: The name of an Amazon IAM role for the crawler to use.
    ///   - s3Path: The path of an Amazon S3 folder to use as a target location.
    ///   - cronSchedule: A `cron` schedule indicating when to run the crawler.
    ///   - databaseName: The name of an AWS Glue database to operate on.
```

```
    ///
    /// - Returns: `true` if the crawler is created successfully, otherwise
`false`.
    func createCrawler(glueClient: GlueClient, crawlerName: String, iamRole:
String,
                       s3Path: String, cronSchedule: String, databaseName:
String) async -> Bool {
        let s3Target = GlueClientTypes.S3Target(path: s3url)
        let targetList = GlueClientTypes.CrawlerTargets(s3Targets: [s3Target])

        do {
            _ = try await glueClient.createCrawler(
                input: CreateCrawlerInput(
                    databaseName: databaseName,
                    description: "Created by the AWS SDK for Swift Scenario
Example for AWS Glue.",
                    name: crawlerName,
                    role: iamRole,
                    schedule: cronSchedule,
                    tablePrefix: tablePrefix,
                    targets: targetList
                )
            )
        } catch _ as AlreadyExistsException {
            print("*** A crawler named \"\(crawlerName)\" already exists.")
            return false
        } catch _ as OperationTimeoutException {
            print("*** The attempt to create the AWS Glue crawler timed out.")
            return false
        } catch {
            print("*** An unexpected error occurred creating the AWS Glue
crawler: \(error.localizedDescription)")
            return false
        }

        return true
    }
```

- API の詳細については、「AWS SDK for Swift API リファレンス」の「CreateCrawler」を参照してください。

AWS SDK デベロッパーガイドとコード例の詳細なリストについては、「このサービスを AWS SDK で使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI で **CreateJob** を使用する

次のサンプルコードは、CreateJob を使用する方法を説明しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- 基本を学ぶ

.NET

SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Create an AWS Glue job.
    /// </summary>
    /// <param name="jobName">The name of the job.</param>
    /// <param name="roleName">The name of the IAM role to be assumed by
    /// the job.</param>
    /// <param name="description">A description of the job.</param>
    /// <param name="scriptUrl">The URL to the script.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> CreateJobAsync(string dbName, string tableName,
string bucketUrl, string jobName, string roleName, string description, string
scriptUrl)
    {
        var command = new JobCommand
        {
            PythonVersion = "3",
            Name = "glueetl",
            ScriptLocation = scriptUrl,
```

```
        };

        var arguments = new Dictionary<string, string>
        {
            { "--input_database", dbName },
            { "--input_table", tableName },
            { "--output_bucket_url", bucketUrl }
        };

        var request = new CreateJobRequest
        {
            Command = command,
            DefaultArguments = arguments,
            Description = description,
            GlueVersion = "3.0",
            Name = jobName,
            NumberOfWorkers = 10,
            Role = roleName,
            WorkerType = "G.1X"
        };

        var response = await _amazonGlue.CreateJobAsync(request);
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「CreateJob」を参照してください。

C++

SDK for C++

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
        Aws::Client::ClientConfiguration clientConfig;
```

```
        // Optional: Set to the AWS Region in which the bucket was created
(overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::Glue::GlueClient client(clientConfig);

        Aws::Glue::Model::CreateJobRequest request;
        request.SetName(JOB_NAME);
        request.SetRole(roleArn);
        request.SetGlueVersion(GLUE_VERSION);

        Aws::Glue::Model::JobCommand command;
        command.SetName(JOB_COMMAND_NAME);
        command.SetPythonVersion(JOB_PYTHON_VERSION);
        command.SetScriptLocation(
                Aws::String("s3://") + bucketName + "/" + PYTHON_SCRIPT);
        request.SetCommand(command);

        Aws::Glue::Model::CreateJobOutcome outcome = client.CreateJob(request);

        if (outcome.IsSuccess()) {
            std::cout << "Successfully created the job." << std::endl;
        }
        else {
            std::cerr << "Error creating the job. " <<
outcome.GetError().GetMessage()
                      << std::endl;
            deleteAssets(CRAWLER_NAME, CRAWLER_DATABASE_NAME, "", bucketName,
                         clientConfig);
            return false;
        }
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「CreateJob」を参照してください。

CLI

AWS CLI

データを変換するジョブを作成するには

次の create-job 例では、S3 に保存されているスクリプトを実行するストリーミングジョブを作成します。

```
aws glue create-job \
    --name my-testing-job \
    --role AWSGlueServiceRoleDefault \
    --command '{ \
        "Name": "gluestreaming", \
        "ScriptLocation": "s3://amzn-s3-demo-bucket/folder/" \
    }' \
    --region us-east-1 \
    --output json \
    --default-arguments '{ \
        "--job-language":"scala", \
        "--class":"GlueApp" \
    }' \
    --profile my-profile \
    --endpoint https://glue.us-east-1.amazonaws.com
```

test_script.scala の内容:

```
import com.amazonaws.services.glue.ChoiceOption
import com.amazonaws.services.glue.GlueContext
import com.amazonaws.services.glue.MappingSpec
import com.amazonaws.services.glue.ResolveSpec
import com.amazonaws.services.glue.errors.CallSite
import com.amazonaws.services.glue.util.GlueArgParser
import com.amazonaws.services.glue.util.Job
import com.amazonaws.services.glue.util.JsonOptions
import org.apache.spark.SparkContext
import scala.collection.JavaConverters._

object GlueApp {
    def main(sysArgs: Array[String]) {
        val spark: SparkContext = new SparkContext()
        val glueContext: GlueContext = new GlueContext(spark)
        // @params: [JOB_NAME]
        val args = GlueArgParser.getResolvedOptions(sysArgs,
 Seq("JOB_NAME").toArray)
        Job.init(args("JOB_NAME"), glueContext, args.asJava)
        // @type: DataSource
        // @args: [database = "tempdb", table_name = "s3-source",
 transformation_ctx = "datasource0"]
```

```
        // @return: datasource0
        // @inputs: []
        val datasource0 = glueContext.getCatalogSource(database = "tempdb",
 tableName = "s3-source", redshiftTmpDir = "", transformationContext =
 "datasource0").getDynamicFrame()
        // @type: ApplyMapping
        // @args: [mapping = [("sensorid", "int", "sensorid", "int"),
 ("currenttemperature", "int", "currenttemperature", "int"), ("status", "string",
 "status", "string")], transformation_ctx = "applymapping1"]
        // @return: applymapping1
        // @inputs: [frame = datasource0]
        val applymapping1 = datasource0.applyMapping(mappings = Seq(("sensorid",
 "int", "sensorid", "int"), ("currenttemperature", "int", "currenttemperature",
 "int"), ("status", "string", "status", "string")), caseSensitive = false,
 transformationContext = "applymapping1")
        // @type: SelectFields
        // @args: [paths = ["sensorid", "currenttemperature", "status"],
 transformation_ctx = "selectfields2"]
        // @return: selectfields2
        // @inputs: [frame = applymapping1]
        val selectfields2 = applymapping1.selectFields(paths = Seq("sensorid",
 "currenttemperature", "status"), transformationContext = "selectfields2")
        // @type: ResolveChoice
        // @args: [choice = "MATCH_CATALOG", database = "tempdb", table_name =
 "my-s3-sink", transformation_ctx = "resolvechoice3"]
        // @return: resolvechoice3
        // @inputs: [frame = selectfields2]
        val resolvechoice3 = selectfields2.resolveChoice(choiceOption =
 Some(ChoiceOption("MATCH_CATALOG")), database = Some("tempdb"), tableName =
 Some("my-s3-sink"), transformationContext = "resolvechoice3")
        // @type: DataSink
        // @args: [database = "tempdb", table_name = "my-s3-sink",
 transformation_ctx = "datasink4"]
        // @return: datasink4
        // @inputs: [frame = resolvechoice3]
        val datasink4 = glueContext.getCatalogSink(database = "tempdb",
 tableName = "my-s3-sink", redshiftTmpDir = "", transformationContext =
 "datasink4").writeDynamicFrame(resolvechoice3)
        Job.commit()
    }
}
```

出力:

```
{
    "Name": "my-testing-job"
}
```

詳細については、「AWS デベロッパーガイド」の「AWS Glue でのジョブの作成」を参照してください。

- API の詳細については、AWS CLI コマンドリファレンスの「CreateJob」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
   /**
    * Creates a new AWS Glue job.
    *
    * @param glueClient     the AWS Glue client to use for the operation
    * @param jobName        the name of the job to create
    * @param iam            the IAM role to associate with the job
    * @param scriptLocation the location of the script to be used by the job
    * @throws GlueException if there is an error creating the job
    */
   public static void createJob(GlueClient glueClient, String jobName, String
iam, String scriptLocation) {
       try {
           JobCommand command = JobCommand.builder()
               .pythonVersion("3")
               .name("glueetl")
               .scriptLocation(scriptLocation)
               .build();

           CreateJobRequest jobRequest = CreateJobRequest.builder()
               .description("A Job created by using the AWS SDK for Java V2")
```

```
                .glueVersion("2.0")
                .workerType(WorkerType.G_1_X)
                .numberOfWorkers(10)
                .name(jobName)
                .role(iam)
                .command(command)
                .build();

            glueClient.createJob(jobRequest);
            System.out.println(jobName + " was successfully created.");

        } catch (GlueException e) {
            throw e;
        }
    }
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「CreateJob」を参照してください。

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
const createJob = (name, role, scriptBucketName, scriptKey) => {
  const client = new GlueClient({});

  const command = new CreateJobCommand({
    Name: name,
    Role: role,
    Command: {
      Name: "glueetl",
      PythonVersion: "3",
      ScriptLocation: `s3://${scriptBucketName}/${scriptKey}`,
    },
```

```
    GlueVersion: "3.0",
  });

  return client.send(command);
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「CreateJob」を参照してください。

PHP

SDK for PHP

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

```
        $role = $iamService->getRole("AWSGlueServiceRole-DocExample");

        $jobName = 'test-job-' . $uniqid;

        $scriptLocation = "s3://$bucketName/run_job.py";
        $job = $glueService->createJob($jobName, $role['Role']['Arn'],
$scriptLocation);

    public function createJob($jobName, $role, $scriptLocation, $pythonVersion =
'3', $glueVersion = '3.0'): Result
    {
        return $this->glueClient->createJob([
            'Name' => $jobName,
            'Role' => $role,
            'Command' => [
                'Name' => 'glueetl',
                'ScriptLocation' => $scriptLocation,
                'PythonVersion' => $pythonVersion,
            ],
            'GlueVersion' => $glueVersion,
        ]);
```

```
    }
```

- API の詳細については、「AWS SDK for PHP API リファレンス」の「CreateJob」を参照してください。

PowerShell

Tools for PowerShell V4

例 1: この例では、AWS Glue で新しいジョブを作成します。コマンド名の値は常に **glueetl** です。 AWSGlue は、Python または Scala で記述されたジョブスクリプトの実行をサポートします。この例では、ジョブスクリプト (MyTestGlueJob.py) は Python で記述されています。Python パラメータは **$DefArgs** 変数で指定され、ハッシュテーブルを受け入れる **DefaultArguments** パラメータで PowerShell コマンドに渡されます。**$JobParams** 変数のパラメータは CreateJob API から取得され、AWS Glue API リファレンスの「ジョブ (https://docs.aws.amazon.com/glue/latest/dg/aws-glue-api-jobs-job.html)」トピックに記載されています。

```
$Command = New-Object Amazon.Glue.Model.JobCommand
$Command.Name = 'glueetl'
$Command.ScriptLocation = 's3://amzn-s3-demo-source-bucket/admin/
MyTestGlueJob.py'
$Command

$Source = "source_test_table"
$Target = "target_test_table"
$Connections = $Source, $Target

$DefArgs = @{
     '--TempDir' = 's3://amzn-s3-demo-bucket/admin'
     '--job-bookmark-option' = 'job-bookmark-disable'
     '--job-language' = 'python'
     }
$DefArgs

$ExecutionProp = New-Object Amazon.Glue.Model.ExecutionProperty
$ExecutionProp.MaxConcurrentRuns = 1
$ExecutionProp

$JobParams = @{
```

```
    "AllocatedCapacity"    = "5"
    "Command"              = $Command
    "Connections_Connection" = $Connections
    "DefaultArguments"  = $DefArgs
    "Description"       = "This is a test"
    "ExecutionProperty" = $ExecutionProp
    "MaxRetries"        = "1"
    "Name"              = "MyOregonTestGlueJob"
    "Role"              = "Amazon-GlueServiceRoleForSSM"
    "Timeout"           = "20"
     }

New-GlueJob @JobParams
```

- API の詳細については、「AWS Tools for PowerShell Cmdlet リファレンス (V4)」の「CreateJob」を参照してください。

Tools for PowerShell V5

例 1: この例では、AWS Glue で新しいジョブを作成します。コマンド名の値は常に **glueetl** です。 AWSGlue は、Python または Scala で記述されたジョブスクリプトの実行をサポートします。この例では、ジョブスクリプト (MyTestGlueJob.py) は Python で記述されています。Python パラメータは **$DefArgs** 変数で指定され、ハッシュテーブルを受け入れる **DefaultArguments** パラメータで PowerShell コマンドに渡されます。**$JobParams** 変数のパラメータは CreateJob API から取得され、AWS Glue API リファレンスの「ジョブ (https://docs.aws.amazon.com/glue/latest/dg/aws-glue-api-jobs-job.html)」トピックに記載されています。

```
$Command = New-Object Amazon.Glue.Model.JobCommand
$Command.Name = 'glueetl'
$Command.ScriptLocation = 's3://amzn-s3-demo-source-bucket/admin/
MyTestGlueJob.py'
$Command

$Source = "source_test_table"
$Target = "target_test_table"
$Connections = $Source, $Target

$DefArgs = @{
     '--TempDir' = 's3://amzn-s3-demo-bucket/admin'
     '--job-bookmark-option' = 'job-bookmark-disable'
     '--job-language' = 'python'
     }
```

```
$DefArgs

$ExecutionProp = New-Object Amazon.Glue.Model.ExecutionProperty
$ExecutionProp.MaxConcurrentRuns = 1
$ExecutionProp

$JobParams = @{
    "AllocatedCapacity"    = "5"
    "Command"              = $Command
    "Connections_Connection" = $Connections
    "DefaultArguments"  = $DefArgs
    "Description"       = "This is a test"
    "ExecutionProperty" = $ExecutionProp
    "MaxRetries"        = "1"
    "Name"              = "MyOregonTestGlueJob"
    "Role"              = "Amazon-GlueServiceRoleForSSM"
    "Timeout"           = "20"
     }

New-GlueJob @JobParams
```

- API の詳細については、「AWS Tools for PowerShell Cmdlet リファレンス (V5)」の「CreateJob」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""

    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 Glue client.
        """
```

```
        self.glue_client = glue_client


    def create_job(self, name, description, role_arn, script_location):
        """
        Creates a job definition for an extract, transform, and load (ETL) job
that can
        be run by AWS Glue.

        :param name: The name of the job definition.
        :param description: The description of the job definition.
        :param role_arn: The ARN of an IAM role that grants AWS Glue the
permissions
                         it requires to run the job.
        :param script_location: The Amazon S3 URL of a Python ETL script that is
run as
                                part of the job. The script defines how the data
is
                                transformed.
        """
        try:
            self.glue_client.create_job(
                Name=name,
                Description=description,
                Role=role_arn,
                Command={
                    "Name": "glueetl",
                    "ScriptLocation": script_location,
                    "PythonVersion": "3",
                },
                GlueVersion="3.0",
            )
        except ClientError as err:
            logger.error(
                "Couldn't create job %s. Here's why: %s: %s",
                name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- API の詳細については、「AWS SDK for Python (Boto3) API リファレンス」の「CreateJob」を参照してください。

Ruby

SDK for Ruby

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
# The `GlueWrapper` class serves as a wrapper around the AWS Glue API, providing
 a simplified interface for common operations.
# It encapsulates the functionality of the AWS SDK for Glue and provides methods
 for interacting with Glue crawlers, databases, tables, jobs, and S3 resources.
# The class initializes with a Glue client and a logger, allowing it to make API
 calls and log any errors or informational messages.
class GlueWrapper
  def initialize(glue_client, logger)
    @glue_client = glue_client
    @logger = logger
  end

  # Creates a new job with the specified configuration.
  #
  # @param name [String] The name of the job.
  # @param description [String] The description of the job.
  # @param role_arn [String] The ARN of the IAM role to be used by the job.
  # @param script_location [String] The location of the ETL script for the job.
  # @return [void]
  def create_job(name, description, role_arn, script_location)
    @glue_client.create_job(
      name: name,
      description: description,
      role: role_arn,
      command: {
        name: 'glueetl',
        script_location: script_location,
```

```
        python_version: '3'
      },
      glue_version: '3.0'
    )
  rescue Aws::Glue::Errors::GlueException => e
    @logger.error("Glue could not create job #{name}: \n#{e.message}")
    raise
  end
```

- API の詳細については、「AWS SDK for Ruby API リファレンス」の「CreateJob」を参照してください。

Rust

SDK for Rust

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
        let create_job = glue
            .create_job()
            .name(self.job())
            .role(self.iam_role.expose_secret())
            .command(
                JobCommand::builder()
                    .name("glueetl")
                    .python_version("3")
                    .script_location(format!("s3://{}/job.py", self.bucket()))
                    .build(),
            )
            .glue_version("3.0")
            .send()
            .await
            .map_err(GlueMvpError::from_glue_sdk)?;

        let job_name = create_job.name().ok_or_else(|| {
```

```
            GlueMvpError::Unknown("Did not get job name after creating
job".into())
        })?;
```

- API の詳細については、「AWS SDK for Rust API リファレンス」の「CreateJob」を参照してください。

Swift

SDK for Swift

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
import AWSClientRuntime
import AWSGlue

    /// Create a new AWS Glue job.
    ///
    /// - Parameters:
    ///   - glueClient: The AWS Glue client to use.
    ///   - jobName: The name to give the new job.
    ///   - role: The IAM role for the job to use when accessing AWS services.
    ///   - scriptLocation: The AWS S3 URI of the script to be run by the job.
    ///
    /// - Returns: `true` if the job is created successfully, otherwise `false`.
    func createJob(glueClient: GlueClient, name jobName: String, role: String,
                   scriptLocation: String) async -> Bool {
        let command = GlueClientTypes.JobCommand(
            name: "glueetl",
            pythonVersion: "3",
            scriptLocation: scriptLocation
        )

        do {
            _ = try await glueClient.createJob(
                input: CreateJobInput(
```

```
                    command: command,
                    description: "Created by the AWS SDK for Swift Glue basic
scenario example.",
                    glueVersion: "3.0",
                    name: jobName,
                    numberOfWorkers: 10,
                    role: role,
                    workerType: .g1x
                )
            )
        } catch {
            return false
        }
        return true
    }
```

- API の詳細については、「AWS SDK for Swift API リファレンス」の「CreateJob」を参照してください。

AWS SDK デベロッパーガイドとコード例の詳細なリストについては、「このサービスを AWS SDK で使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK で **DeleteCrawler** を使用する

次のサンプルコードは、DeleteCrawler を使用する方法を説明しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- 基本を学ぶ

.NET

SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Delete an AWS Glue crawler.
    /// </summary>
    /// <param name="crawlerName">The name of the crawler.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteCrawlerAsync(string crawlerName)
    {
        var response = await _amazonGlue.DeleteCrawlerAsync(new
DeleteCrawlerRequest { Name = crawlerName });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「DeleteCrawler」を参照してください。

C++

SDK for C++

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region in which the bucket was created
(overrides config file).
```

```
        // clientConfig.region = "us-east-1";

    Aws::Glue::GlueClient client(clientConfig);

        Aws::Glue::Model::DeleteCrawlerRequest request;
        request.SetName(crawler);

        Aws::Glue::Model::DeleteCrawlerOutcome outcome =
client.DeleteCrawler(request);

        if (outcome.IsSuccess()) {
            std::cout << "Successfully deleted the crawler." << std::endl;
        }
        else {
            std::cerr << "Error deleting the crawler. "
                      << outcome.GetError().GetMessage() << std::endl;
            result = false;
        }
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「DeleteCrawler」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /**
     * Deletes a specific AWS Glue crawler.
     *
     * @param glueClient  the AWS Glue client object
     * @param crawlerName the name of the crawler to be deleted
     * @throws GlueException if an error occurs during the deletion process
     */
```

```
    public static void deleteSpecificCrawler(GlueClient glueClient, String
 crawlerName) {
        try {
            DeleteCrawlerRequest deleteCrawlerRequest =
 DeleteCrawlerRequest.builder()
                .name(crawlerName)
                .build();

            glueClient.deleteCrawler(deleteCrawlerRequest);
            System.out.println(crawlerName + " was deleted");

        } catch (GlueException e) {
            throw e;
        }
    }
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「DeleteCrawler」を参照してください。

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
const deleteCrawler = (crawlerName) => {
  const client = new GlueClient({});

  const command = new DeleteCrawlerCommand({
    Name: crawlerName,
  });

  return client.send(command);
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DeleteCrawler」を参照してください。

PHP

SDK for PHP

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

```
        echo "Delete the crawler.\n";
        $glueClient->deleteCrawler([
            'Name' => $crawlerName,
        ]);

    public function deleteCrawler($crawlerName)
    {
        return $this->glueClient->deleteCrawler([
            'Name' => $crawlerName,
        ]);
    }
```

- API の詳細については、「AWS SDK for PHP API リファレンス」の「DeleteCrawler」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""

    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 Glue client.
        """
        self.glue_client = glue_client


    def delete_crawler(self, name):
        """
        Deletes a crawler.

        :param name: The name of the crawler to delete.
        """
        try:
            self.glue_client.delete_crawler(Name=name)
        except ClientError as err:
            logger.error(
                "Couldn't delete crawler %s. Here's why: %s: %s",
                name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- API の詳細については、「AWS SDK for Python (Boto3) API リファレンス」の「DeleteCrawler」を参照してください。

Ruby

SDK for Ruby

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
# The `GlueWrapper` class serves as a wrapper around the AWS Glue API, providing
 a simplified interface for common operations.
# It encapsulates the functionality of the AWS SDK for Glue and provides methods
 for interacting with Glue crawlers, databases, tables, jobs, and S3 resources.
# The class initializes with a Glue client and a logger, allowing it to make API
 calls and log any errors or informational messages.
class GlueWrapper
  def initialize(glue_client, logger)
    @glue_client = glue_client
    @logger = logger
  end

  # Deletes a crawler with the specified name.
  #
  # @param name [String] The name of the crawler to delete.
  # @return [void]
  def delete_crawler(name)
    @glue_client.delete_crawler(name: name)
  rescue Aws::Glue::Errors::ServiceError => e
    @logger.error("Glue could not delete crawler #{name}: \n#{e.message}")
    raise
  end
```

- API の詳細については、「AWS SDK for Ruby API リファレンス」の「DeleteCrawler」を参照してください。

Rust

SDK for Rust

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
        glue.delete_crawler()
            .name(self.crawler())
            .send()
```

```
            .await
            .map_err(GlueMvpError::from_glue_sdk)?;
```

- API の詳細については、「AWS SDK for Rust API リファレンス」の「DeleteCrawler」を参照してください。

Swift

SDK for Swift

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
import AWSClientRuntime
import AWSGlue

    /// Delete an AWS Glue crawler.
    ///
    /// - Parameters:
    ///   - glueClient: The AWS Glue client to use.
    ///   - name: The name of the crawler to delete.
    ///
    /// - Returns: `true` if successful, otherwise `false`.
    func deleteCrawler(glueClient: GlueClient, name: String) async -> Bool {
        do {
            _ = try await glueClient.deleteCrawler(
                input: DeleteCrawlerInput(name: name)
            )
        } catch {
            return false
        }
        return true
    }
```

- API の詳細については、「AWS SDK for Swift API リファレンス」の「DeleteCrawler」を参照してください。

AWS SDK デベロッパーガイドとコード例の詳細なリストについては、「このサービスを AWS SDK で使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK で **DeleteDatabase** を使用する

次のサンプルコードは、DeleteDatabase を使用する方法を説明しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- 基本を学ぶ

.NET

SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Delete the AWS Glue database.
    /// </summary>
    /// <param name="dbName">The name of the database.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteDatabaseAsync(string dbName)
    {
        var response = await _amazonGlue.DeleteDatabaseAsync(new
DeleteDatabaseRequest { Name = dbName });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「DeleteDatabase」を参照してください。

C++

SDK for C++

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region in which the bucket was created
(overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::Glue::GlueClient client(clientConfig);

        Aws::Glue::Model::DeleteDatabaseRequest request;
        request.SetName(database);

        Aws::Glue::Model::DeleteDatabaseOutcome outcome = client.DeleteDatabase(
                request);

        if (outcome.IsSuccess()) {
            std::cout << "Successfully deleted the database." << std::endl;
        }
        else {
            std::cerr << "Error deleting database. " <<
outcome.GetError().GetMessage()
                      << std::endl;
            result = false;
        }
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「DeleteDatabase」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
   /**
    * Deletes a AWS Glue Database.
    *
    * @param glueClient   An instance of the AWS Glue client used to interact
with the AWS Glue service.
    * @param databaseName The name of the database to be deleted.
    * @throws GlueException If an error occurs while deleting the database.
    */
   public static void deleteDatabase(GlueClient glueClient, String databaseName)
{
       try {
           DeleteDatabaseRequest request = DeleteDatabaseRequest.builder()
               .name(databaseName)
               .build();

           glueClient.deleteDatabase(request);
           System.out.println(databaseName + " was successfully deleted");

       } catch (GlueException e) {
           throw e;
       }
   }
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「DeleteDatabase」を参照してください。

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
const deleteDatabase = (databaseName) => {
  const client = new GlueClient({});

  const command = new DeleteDatabaseCommand({
    Name: databaseName,
  });

  return client.send(command);
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DeleteDatabase」を参照してください。

PHP

SDK for PHP

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

```
        echo "Delete the databases.\n";
        $glueClient->deleteDatabase([
            'Name' => $databaseName,
        ]);

    public function deleteDatabase($databaseName)
```

```
    {
        return $this->glueClient->deleteDatabase([
            'Name' => $databaseName,
        ]);
    }
```

- API の詳細については、「AWS SDK for PHP API リファレンス」の「DeleteDatabase」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""

    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 Glue client.
        """
        self.glue_client = glue_client


    def delete_database(self, name):
        """
        Deletes a metadata database from your Data Catalog.

        :param name: The name of the database to delete.
        """
        try:
            self.glue_client.delete_database(Name=name)
        except ClientError as err:
            logger.error(
                "Couldn't delete database %s. Here's why: %s: %s",
```

```
                name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- API の詳細については、「AWS SDK for Python (Boto3) API リファレンス」で「DeleteDatabase」を参照してください。

Ruby

SDK for Ruby

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
# The `GlueWrapper` class serves as a wrapper around the AWS Glue API, providing
 a simplified interface for common operations.
# It encapsulates the functionality of the AWS SDK for Glue and provides methods
 for interacting with Glue crawlers, databases, tables, jobs, and S3 resources.
# The class initializes with a Glue client and a logger, allowing it to make API
 calls and log any errors or informational messages.
class GlueWrapper
  def initialize(glue_client, logger)
    @glue_client = glue_client
    @logger = logger
  end

  # Removes a specified database from a Data Catalog.
  #
  # @param database_name [String] The name of the database to delete.
  # @return [void]
  def delete_database(database_name)
    @glue_client.delete_database(name: database_name)
  rescue Aws::Glue::Errors::ServiceError => e
```

```
    @logger.error("Glue could not delete database: \n#{e.message}")
  end
```

- API の詳細については、「AWS SDK for Ruby API リファレンス」の「DeleteDatabase」を参照してください。

Rust

SDK for Rust

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
        glue.delete_database()
            .name(self.database())
            .send()
            .await
            .map_err(GlueMvpError::from_glue_sdk)?;
```

- API の詳細については、「AWS SDK for Rust API リファレンス」の「DeleteDatabase」を参照してください。

Swift

SDK for Swift

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
import AWSClientRuntime
import AWSGlue
```

```
    /// Delete the specified database.
    ///
    /// - Parameters:
    ///   - glueClient: The AWS Glue client to use.
    ///   - databaseName: The name of the database to delete.
    ///   - deleteTables: A Bool indicating whether or not to delete the
    ///     tables in the database before attempting to delete the database.
    ///
    /// - Returns: `true` if the database (and optionally its tables) are
    ///   deleted, otherwise `false`.
    func deleteDatabase(glueClient: GlueClient, name databaseName: String,
                        withTables deleteTables: Bool = false) async -> Bool {
        if deleteTables {
            var tableNames: [String] = []

            // Get a list of the names of all of the tables in the database.

            let tableList = await self.getTablesInDatabase(glueClient:
glueClient, databaseName: databaseName)
            for table in tableList {
                guard let name = table.name else {
                    continue
                }
                tableNames.append(name)
            }

            // Delete the tables.

            do {
                _ = try await glueClient.batchDeleteTable(
                    input: BatchDeleteTableInput(
                        databaseName: databaseName,
                        tablesToDelete: tableNames
                    )
                )
            } catch {
                print("*** Unable to delete the tables.")
            }
            return true
        }

        // Delete the database itself.
```

```
        do {
            _ = try await glueClient.deleteDatabase(
                input: DeleteDatabaseInput(name: databaseName)
            )
        } catch {
            print("*** Unable to delete the database.")
        }
        return true
    }
```

- API の詳細については、「AWS SDK for Swift API リファレンス」の「DeleteDatabase」を参照してください。

AWS SDK デベロッパーガイドとコード例の詳細なリストについては、「このサービスを AWS SDK で使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI で **DeleteJob** を使用する

次のサンプルコードは、DeleteJob を使用する方法を説明しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- 基本を学ぶ

.NET

SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Delete an AWS Glue job.
    /// </summary>
```

```
    /// <param name="jobName">The name of the job.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteJobAsync(string jobName)
    {
        var response = await _amazonGlue.DeleteJobAsync(new DeleteJobRequest
{ JobName = jobName });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「DeleteJob」を参照してください。

C++

SDK for C++

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region in which the bucket was created
(overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::Glue::GlueClient client(clientConfig);

        Aws::Glue::Model::DeleteJobRequest request;
        request.SetJobName(job);

        Aws::Glue::Model::DeleteJobOutcome outcome = client.DeleteJob(request);


        if (outcome.IsSuccess()) {
            std::cout << "Successfully deleted the job." << std::endl;
        }
        else {
```

```
            std::cerr << "Error deleting the job. " <<
outcome.GetError().GetMessage()
                      << std::endl;
            result = false;
        }
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「DeleteJob」を参照してください。

CLI

AWS CLI

ジョブを削除するには

次の delete-job 例では、不要になったジョブを削除します。

```
aws glue delete-job \
    --job-name my-testing-job
```

出力:

```
{
    "JobName": "my-testing-job"
}
```

詳細については、「AWS Glue デベロッパーガイド」の「AWS Glue コンソールでのジョブの操作」を参照してください。

- API の詳細については、「AWS CLI コマンドリファレンス」の「DeleteJob」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
/**
 * Deletes a Glue job.
 *
 * @param glueClient the Glue client to use for the operation
 * @param jobName    the name of the job to be deleted
 * @throws GlueException if there is an error deleting the job
 */
public static void deleteJob(GlueClient glueClient, String jobName) {
    try {
        DeleteJobRequest jobRequest = DeleteJobRequest.builder()
            .jobName(jobName)
            .build();

        glueClient.deleteJob(jobRequest);
        System.out.println(jobName + " was successfully deleted");

    } catch (GlueException e) {
        throw e;
    }
}
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「DeleteJob」を参照してください。

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
const deleteJob = (jobName) => {
  const client = new GlueClient({});

  const command = new DeleteJobCommand({
    JobName: jobName,
  });

  return client.send(command);
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DeleteJob」を参照してください。

PHP

SDK for PHP

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

```
        echo "Delete the job.\n";
        $glueClient->deleteJob([
            'JobName' => $job['Name'],
        ]);

    public function deleteJob($jobName)
```

```
    {
        return $this->glueClient->deleteJob([
            'JobName' => $jobName,
        ]);
    }
```

- API の詳細については、「AWS SDK for PHP API リファレンス」の「DeleteJob」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""

    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 Glue client.
        """
        self.glue_client = glue_client


    def delete_job(self, job_name):
        """
        Deletes a job definition. This also deletes data about all runs that are
        associated with this job definition.

        :param job_name: The name of the job definition to delete.
        """
        try:
            self.glue_client.delete_job(JobName=job_name)
        except ClientError as err:
            logger.error(
```

```
                "Couldn't delete job %s. Here's why: %s: %s",
                job_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- API の詳細については、「AWS SDK for Python (Boto3) API リファレンス」の「DeleteJob」を参照してください。

Ruby

SDK for Ruby

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
# The `GlueWrapper` class serves as a wrapper around the AWS Glue API, providing
 a simplified interface for common operations.
# It encapsulates the functionality of the AWS SDK for Glue and provides methods
 for interacting with Glue crawlers, databases, tables, jobs, and S3 resources.
# The class initializes with a Glue client and a logger, allowing it to make API
 calls and log any errors or informational messages.
class GlueWrapper
  def initialize(glue_client, logger)
    @glue_client = glue_client
    @logger = logger
  end

  # Deletes a job with the specified name.
  #
  # @param job_name [String] The name of the job to delete.
  # @return [void]
  def delete_job(job_name)
    @glue_client.delete_job(job_name: job_name)
```

```
  rescue Aws::Glue::Errors::ServiceError => e
    @logger.error("Glue could not delete job: \n#{e.message}")
  end
```

- API の詳細については、「AWS SDK for Ruby API リファレンス」の「DeleteJob」を参照してください。

Rust

SDK for Rust

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
        glue.delete_job()
            .job_name(self.job())
            .send()
            .await
            .map_err(GlueMvpError::from_glue_sdk)?;
```

- API の詳細については、「AWS SDK for Rust API リファレンス」の「DeleteJob」を参照してください。

Swift

SDK for Swift

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
import AWSClientRuntime
```

```
import AWSGlue

    /// Delete an AWS Glue job.
    ///
    /// - Parameters:
    ///   - glueClient: The AWS Glue client to use.
    ///   - jobName: The name of the job to delete.
    ///
    /// - Returns: `true` if the job is successfully deleted, otherwise `false`.
    func deleteJob(glueClient: GlueClient, name jobName: String) async -> Bool {
        do {
            _ = try await glueClient.deleteJob(
                input: DeleteJobInput(jobName: jobName)
            )
        } catch {
            return false
        }
        return true
    }
```

- API の詳細については、「AWS SDK for Swift API リファレンス」の「DeleteJob」を参照してください。

AWS SDK デベロッパーガイドとコード例の詳細なリストについては、「このサービスを AWS SDK で使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK で **DeleteTable** を使用する

次のサンプルコードは、DeleteTable を使用する方法を説明しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- 基本を学ぶ

.NET

SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Delete a table from an AWS Glue database.
    /// </summary>
    /// <param name="tableName">The table to delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteTableAsync(string dbName, string tableName)
    {
        var response = await _amazonGlue.DeleteTableAsync(new DeleteTableRequest
{ Name = tableName, DatabaseName = dbName });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「DeleteTable」を参照してください。

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
const deleteTable = (databaseName, tableName) => {
  const client = new GlueClient({});
```

```
  const command = new DeleteTableCommand({
    DatabaseName: databaseName,
    Name: tableName,
  });

  return client.send(command);
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DeleteTable」を参照してください。

PHP

SDK for PHP

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

```
        echo "Delete the tables.\n";
        foreach ($tables['TableList'] as $table) {
            $glueService->deleteTable($table['Name'], $databaseName);
        }

    public function deleteTable($tableName, $databaseName)
    {
        return $this->glueClient->deleteTable([
            'DatabaseName' => $databaseName,
            'Name' => $tableName,
        ]);
    }
```

- API の詳細については、「AWS SDK for PHP API リファレンス」の「DeleteTable」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""

    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 Glue client.
        """
        self.glue_client = glue_client


    def delete_table(self, db_name, table_name):
        """
        Deletes a table from a metadata database.

        :param db_name: The name of the database that contains the table.
        :param table_name: The name of the table to delete.
        """
        try:
            self.glue_client.delete_table(DatabaseName=db_name, Name=table_name)
        except ClientError as err:
            logger.error(
                "Couldn't delete table %s. Here's why: %s: %s",
                table_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- API の詳細については、「AWS SDK for Python (Boto3) API リファレンス」の「DeleteTable」を参照してください。

Ruby

SDK for Ruby

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
# The `GlueWrapper` class serves as a wrapper around the AWS Glue API, providing
 a simplified interface for common operations.
# It encapsulates the functionality of the AWS SDK for Glue and provides methods
 for interacting with Glue crawlers, databases, tables, jobs, and S3 resources.
# The class initializes with a Glue client and a logger, allowing it to make API
 calls and log any errors or informational messages.
class GlueWrapper
  def initialize(glue_client, logger)
    @glue_client = glue_client
    @logger = logger
  end

  # Deletes a table with the specified name.
  #
  # @param database_name [String] The name of the catalog database in which the
 table resides.
  # @param table_name [String] The name of the table to be deleted.
  # @return [void]
  def delete_table(database_name, table_name)
    @glue_client.delete_table(database_name: database_name, name: table_name)
  rescue Aws::Glue::Errors::ServiceError => e
    @logger.error("Glue could not delete job: \n#{e.message}")
  end
```

- API の詳細については、「AWS SDK for Ruby API リファレンス」の「DeleteTable」を参照してください。

Rust

SDK for Rust

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
        for t in &self.tables {
            glue.delete_table()
                .name(t.name())
                .database_name(self.database())
                .send()
                .await
                .map_err(GlueMvpError::from_glue_sdk)?;
        }
```

- API の詳細については、「AWS SDK for Rust API リファレンス」の「DeleteTable」を参照してください。

AWS SDK デベロッパーガイドとコード例の詳細なリストについては、「このサービスを AWS SDK で使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK で **GetCrawler** を使用する

次のサンプルコードは、GetCrawler を使用する方法を説明しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- 基本を学ぶ

.NET

SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Get information about an AWS Glue crawler.
    /// </summary>
    /// <param name="crawlerName">The name of the crawler.</param>
    /// <returns>A Crawler object describing the crawler.</returns>
    public async Task<Crawler?> GetCrawlerAsync(string crawlerName)
    {
        var crawlerRequest = new GetCrawlerRequest
        {
            Name = crawlerName,
        };

        var response = await _amazonGlue.GetCrawlerAsync(crawlerRequest);
        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            var databaseName = response.Crawler.DatabaseName;
            Console.WriteLine($"{crawlerName} has the database {databaseName}");
            return response.Crawler;
        }

        Console.WriteLine($"No information regarding {crawlerName} could be
found.");
        return null;
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「GetCrawler」を参照してください。

C++

SDK for C++

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
       Aws::Client::ClientConfiguration clientConfig;
       // Optional: Set to the AWS Region in which the bucket was created
(overrides config file).
       // clientConfig.region = "us-east-1";

   Aws::Glue::GlueClient client(clientConfig);

       Aws::Glue::Model::GetCrawlerRequest request;
       request.SetName(CRAWLER_NAME);

       Aws::Glue::Model::GetCrawlerOutcome outcome = client.GetCrawler(request);

       if (outcome.IsSuccess()) {
           Aws::Glue::Model::CrawlerState crawlerState =
outcome.GetResult().GetCrawler().GetState();
           std::cout << "Retrieved crawler with state " <<

Aws::Glue::Model::CrawlerStateMapper::GetNameForCrawlerState(
                               crawlerState)
                   << "." << std::endl;
       }
       else {
           std::cerr << "Error retrieving a crawler.  "
                     << outcome.GetError().GetMessage() << std::endl;
           deleteAssets(CRAWLER_NAME, CRAWLER_DATABASE_NAME, "", bucketName,
                        clientConfig);
           return false;
       }
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「GetCrawler」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
   /**
    * Retrieves a specific crawler from the AWS Glue service and waits for it to
be in the "READY" state.
    *
    * @param glueClient  the AWS Glue client used to interact with the Glue
service
    * @param crawlerName the name of the crawler to be retrieved
    */
   public static void getSpecificCrawler(GlueClient glueClient, String
crawlerName) throws InterruptedException {
       try {
           GetCrawlerRequest crawlerRequest = GetCrawlerRequest.builder()
               .name(crawlerName)
               .build();

           boolean ready = false;
           while (!ready) {
               GetCrawlerResponse response =
glueClient.getCrawler(crawlerRequest);
               String status = response.crawler().stateAsString();
               if (status.compareTo("READY") == 0) {
                   ready = true;
               }
               Thread.sleep(3000);
           }

           System.out.println("The crawler is now ready");

       } catch (GlueException | InterruptedException e) {
           throw e;
       }
   }
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「GetCrawler」を参照してください。

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
const getCrawler = (name) => {
  const client = new GlueClient({});

  const command = new GetCrawlerCommand({
    Name: name,
  });

  return client.send(command);
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「GetCrawler」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
suspend fun getSpecificCrawler(crawlerName: String?) {
    val request =
        GetCrawlerRequest {
            name = crawlerName
        }
    GlueClient { region = "us-east-1" }.use { glueClient ->
        val response = glueClient.getCrawler(request)
        val role = response.crawler?.role
        println("The role associated with this crawler is $role")
    }
}
```

- API の詳細については、「AWS SDK for Kotlin API リファレンス」の「GetCrawler」を参照してください。

PHP

SDK for PHP

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

```
        echo "Waiting for crawler";
        do {
            $crawler = $glueService->getCrawler($crawlerName);
            echo ".";
            sleep(10);
        } while ($crawler['Crawler']['State'] != "READY");
        echo "\n";

    public function getCrawler($crawlerName)
    {
        return $this->customWaiter(function () use ($crawlerName) {
            return $this->glueClient->getCrawler([
                'Name' => $crawlerName,
            ]);
        });
```

```
    }
```

- API の詳細については、「AWS SDK for PHP API リファレンス」の「GetCrawler」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""

    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 Glue client.
        """
        self.glue_client = glue_client


    def get_crawler(self, name):
        """
        Gets information about a crawler.

        :param name: The name of the crawler to look up.
        :return: Data about the crawler.
        """
        crawler = None
        try:
            response = self.glue_client.get_crawler(Name=name)
            crawler = response["Crawler"]
        except ClientError as err:
            if err.response["Error"]["Code"] == "EntityNotFoundException":
                logger.info("Crawler %s doesn't exist.", name)
            else:
```

```
                logger.error(
                    "Couldn't get crawler %s. Here's why: %s: %s",
                    name,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        return crawler
```

- API の詳細については、「AWS SDK for Python (Boto3) API リファレンス」の「GetCrawler」を参照してください。

Ruby

SDK for Ruby

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
# The `GlueWrapper` class serves as a wrapper around the AWS Glue API, providing
 a simplified interface for common operations.
# It encapsulates the functionality of the AWS SDK for Glue and provides methods
 for interacting with Glue crawlers, databases, tables, jobs, and S3 resources.
# The class initializes with a Glue client and a logger, allowing it to make API
 calls and log any errors or informational messages.
class GlueWrapper
  def initialize(glue_client, logger)
    @glue_client = glue_client
    @logger = logger
  end

  # Retrieves information about a specific crawler.
  #
  # @param name [String] The name of the crawler to retrieve information about.
```

```
  # @return [Aws::Glue::Types::Crawler, nil] The crawler object if found, or nil
if not found.
 def get_crawler(name)
   @glue_client.get_crawler(name: name)
 rescue Aws::Glue::Errors::EntityNotFoundException
   @logger.info("Crawler #{name} doesn't exist.")
   false
 rescue Aws::Glue::Errors::GlueException => e
   @logger.error("Glue could not get crawler #{name}: \n#{e.message}")
   raise
 end
```

- API の詳細については、「AWS SDK for Ruby API リファレンス」の「GetCrawler」を参照してください。

Rust

SDK for Rust

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
            let tmp_crawler = glue
                .get_crawler()
                .name(self.crawler())
                .send()
                .await
                .map_err(GlueMvpError::from_glue_sdk)?;
```

- API の詳細については、「AWS SDK for Rust API リファレンス」の「GetCrawler」を参照してください。

Swift

SDK for Swift

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
import AWSClientRuntime
import AWSGlue

    /// Get the state of the specified AWS Glue crawler.
    ///
    /// - Parameters:
    ///   - glueClient: The AWS Glue client to use.
    ///   - name: The name of the crawler whose state should be returned.
    ///
    /// - Returns: A `GlueClientTypes.CrawlerState` value describing the
    ///   state of the crawler.
    func getCrawlerState(glueClient: GlueClient, name: String) async ->
 GlueClientTypes.CrawlerState {
        do {
            let output = try await glueClient.getCrawler(
                input: GetCrawlerInput(name: name)
            )

            // If the crawler or its state is `nil`, report that the crawler
            // is stopping. This may not be what you want for your
            // application but it works for this one!

            guard let crawler = output.crawler else {
                return GlueClientTypes.CrawlerState.stopping
            }
            guard let state = crawler.state else {
                return GlueClientTypes.CrawlerState.stopping
            }
            return state
        } catch {
            return GlueClientTypes.CrawlerState.stopping
        }
```

```
    }
```

- API の詳細については、「AWS SDK for Swift API リファレンス」の「GetCrawler」を参照してください。

AWS SDK デベロッパーガイドとコード例の詳細なリストについては、「このサービスを AWS SDK で使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK で **GetDatabase** を使用する

次のサンプルコードは、GetDatabase を使用する方法を説明しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- 基本を学ぶ

.NET

SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Get information about an AWS Glue database.
    /// </summary>
    /// <param name="dbName">The name of the database.</param>
    /// <returns>A Database object containing information about the database.</
returns>
    public async Task<Database> GetDatabaseAsync(string dbName)
    {
        var databasesRequest = new GetDatabaseRequest
        {
            Name = dbName,
```

```
        };

        var response = await _amazonGlue.GetDatabaseAsync(databasesRequest);
        return response.Database;
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「GetDatabase」を参照してください。

C++

SDK for C++

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
       Aws::Client::ClientConfiguration clientConfig;
       // Optional: Set to the AWS Region in which the bucket was created
(overrides config file).
       // clientConfig.region = "us-east-1";

   Aws::Glue::GlueClient client(clientConfig);

       Aws::Glue::Model::GetDatabaseRequest request;
       request.SetName(CRAWLER_DATABASE_NAME);

       Aws::Glue::Model::GetDatabaseOutcome outcome =
client.GetDatabase(request);

       if (outcome.IsSuccess()) {
           const Aws::Glue::Model::Database &database =
outcome.GetResult().GetDatabase();

           std::cout << "Successfully retrieve the database\n" <<
                     database.Jsonize().View().WriteReadable() << "'." <<
std::endl;
```

```
        }
        else {
            std::cerr << "Error getting the database.  "
                      << outcome.GetError().GetMessage() << std::endl;
            deleteAssets(CRAWLER_NAME, CRAWLER_DATABASE_NAME, "", bucketName,
                         clientConfig);
            return false;
        }
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「GetDatabase」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
   /**
    * Retrieves the specific database from the AWS Glue service.
    *
    * @param glueClient   an instance of the AWS Glue client used to interact
with the service
    * @param databaseName the name of the database to retrieve
    * @throws GlueException if there is an error retrieving the database from
the AWS Glue service
    */
   public static void getSpecificDatabase(GlueClient glueClient, String
databaseName) {
        try {
            GetDatabaseRequest databasesRequest = GetDatabaseRequest.builder()
                .name(databaseName)
                .build();

            GetDatabaseResponse response =
glueClient.getDatabase(databasesRequest);
```

```
            Instant createDate = response.database().createTime();

            // Convert the Instant to readable date.
            DateTimeFormatter formatter =
 DateTimeFormatter.ofLocalizedDateTime(FormatStyle.SHORT)
                .withLocale(Locale.US)
                .withZone(ZoneId.systemDefault());

            formatter.format(createDate);
            System.out.println("The create date of the database is " +
 createDate);

        } catch (GlueException e) {
            throw e;
        }
    }
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「GetDatabase」を参照してください。

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
const getDatabase = (name) => {
  const client = new GlueClient({});

  const command = new GetDatabaseCommand({
    Name: name,
  });

  return client.send(command);
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「GetDatabase」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
suspend fun getSpecificDatabase(databaseName: String?) {
    val request =
        GetDatabaseRequest {
            name = databaseName
        }

    GlueClient { region = "us-east-1" }.use { glueClient ->
        val response = glueClient.getDatabase(request)
        val dbDesc = response.database?.description
        println("The database description is $dbDesc")
    }
}
```

- API の詳細については、「AWS SDK for Kotlin API リファレンス」の「GetDatabase」を参照してください。

PHP

SDK for PHP

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

```
        $databaseName = "doc-example-database-$uniqid";

        $database = $glueService->getDatabase($databaseName);
        echo "Found a database named " . $database['Database']['Name'] . "\n";

    public function getDatabase(string $databaseName): Result
    {
        return $this->customWaiter(function () use ($databaseName) {
            return $this->glueClient->getDatabase([
                'Name' => $databaseName,
            ]);
        });
    }
```

- API の詳細については、「AWS SDK for PHP API リファレンス」の「GetDatabase」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""

    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 Glue client.
        """
        self.glue_client = glue_client


    def get_database(self, name):
        """
        Gets information about a database in your Data Catalog.
```

```
        :param name: The name of the database to look up.
        :return: Information about the database.
        """
        try:
            response = self.glue_client.get_database(Name=name)
        except ClientError as err:
            logger.error(
                "Couldn't get database %s. Here's why: %s: %s",
                name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["Database"]
```

- API の詳細については、「AWS SDK for Python (Boto3) API リファレンス」の「GetDatabase」を参照してください。

Ruby

SDK for Ruby

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
# The `GlueWrapper` class serves as a wrapper around the AWS Glue API, providing
 a simplified interface for common operations.
# It encapsulates the functionality of the AWS SDK for Glue and provides methods
 for interacting with Glue crawlers, databases, tables, jobs, and S3 resources.
# The class initializes with a Glue client and a logger, allowing it to make API
 calls and log any errors or informational messages.
class GlueWrapper
  def initialize(glue_client, logger)
```

```
    @glue_client = glue_client
    @logger = logger
  end

  # Retrieves information about a specific database.
  #
  # @param name [String] The name of the database to retrieve information about.
  # @return [Aws::Glue::Types::Database, nil] The database object if found, or
nil if not found.
  def get_database(name)
    response = @glue_client.get_database(name: name)
    response.database
  rescue Aws::Glue::Errors::GlueException => e
    @logger.error("Glue could not get database #{name}: \n#{e.message}")
    raise
  end
```

- API の詳細については、「AWS SDK for Ruby API リファレンス」の「GetDatabase」を参照してください。

Rust

SDK for Rust

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
        let database = glue
            .get_database()
            .name(self.database())
            .send()
            .await
            .map_err(GlueMvpError::from_glue_sdk)?
            .to_owned();
        let database = database
            .database()
```

```
            .ok_or_else(|| GlueMvpError::Unknown("Could not find
database".into()))?;
```

- API の詳細については、「AWS SDK for Rust API リファレンス」の「GetDatabase」を参照してください。

Swift

SDK for Swift

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
import AWSClientRuntime
import AWSGlue

    /// Get the AWS Glue database with the specified name.
    ///
    /// - Parameters:
    ///   - glueClient: The AWS Glue client to use.
    ///   - name: The name of the database to return.
    ///
    /// - Returns: The `GlueClientTypes.Database` object describing the
    ///   specified database, or `nil` if an error occurs or the database
    ///   isn't found.
    func getDatabase(glueClient: GlueClient, name: String) async ->
GlueClientTypes.Database? {
        do {
            let output = try await glueClient.getDatabase(
                input: GetDatabaseInput(name: name)
            )

            return output.database
        } catch {
            return nil
        }
    }
```

- API の詳細については、「AWS SDK for Swift API リファレンス」の「GetDatabase」を参照してください。

AWS SDK デベロッパーガイドとコード例の詳細なリストについては、「このサービスを AWS SDK で使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI で **GetDatabases** を使用する

次のサンプルコードは、GetDatabases を使用する方法を説明しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- 基本を学ぶ

CLI

AWS CLI

AWS Glue データカタログの一部またはすべてのデータベースの定義を一覧表示するには

次の get-databases の例では、データカタログのデータベースに関する情報を返します。

```
aws glue get-databases
```

出力:

```
{
    "DatabaseList": [
        {
            "Name": "default",
            "Description": "Default Hive database",
            "LocationUri": "file:/spark-warehouse",
            "CreateTime": 1602084052.0,
            "CreateTableDefaultPermissions": [
                {
                    "Principal": {
                        "DataLakePrincipalIdentifier": "IAM_ALLOWED_PRINCIPALS"
```

```
                    },
                    "Permissions": [
                        "ALL"
                    ]
                }
            ],
            "CatalogId": "111122223333"
        },
        {
            "Name": "flights-db",
            "CreateTime": 1587072847.0,
            "CreateTableDefaultPermissions": [
                {
                    "Principal": {
                        "DataLakePrincipalIdentifier": "IAM_ALLOWED_PRINCIPALS"
                    },
                    "Permissions": [
                        "ALL"
                    ]
                }
            ],
            "CatalogId": "111122223333"
        },
        {
            "Name": "legislators",
            "CreateTime": 1601415625.0,
            "CreateTableDefaultPermissions": [
                {
                    "Principal": {
                        "DataLakePrincipalIdentifier": "IAM_ALLOWED_PRINCIPALS"
                    },
                    "Permissions": [
                        "ALL"
                    ]
                }
            ],
            "CatalogId": "111122223333"
        },
        {
            "Name": "tempdb",
            "CreateTime": 1601498566.0,
            "CreateTableDefaultPermissions": [
                {
                    "Principal": {
```

```
                        "DataLakePrincipalIdentifier": "IAM_ALLOWED_PRINCIPALS"
                    },
                    "Permissions": [
                        "ALL"
                    ]
                }
            ],
            "CatalogId": "111122223333"
        }
    ]
}
```

詳細については、「AWS Glue デベロッパーガイド」の「データカタログにデータベースを定義する」を参照してください。

- API の詳細については、「AWS CLI コマンドリファレンス」の「GetDatabases」を参照してください。

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
const getDatabases = () => {
  const client = new GlueClient({});

  const command = new GetDatabasesCommand({});

  return client.send(command);
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「GetDatabases」を参照してください。

AWS SDK デベロッパーガイドとコード例の詳細なリストについては、「このサービスを AWS SDK で使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI で **GetJob** を使用する

次のサンプルコードは、GetJob を使用する方法を説明しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- 基本を学ぶ

CLI

AWS CLI

ジョブに関する情報を取得するには

次の get-job の例では、ジョブに関する情報を取得します。

```
aws glue get-job \
    --job-name my-testing-job
```

出力:

```
{
    "Job": {
        "Name": "my-testing-job",
        "Role": "Glue_DefaultRole",
        "CreatedOn": 1602805698.167,
        "LastModifiedOn": 1602805698.167,
        "ExecutionProperty": {
            "MaxConcurrentRuns": 1
        },
        "Command": {
            "Name": "gluestreaming",
            "ScriptLocation": "s3://janetst-bucket-01/Scripts/test_script.scala",
            "PythonVersion": "2"
        },
        "DefaultArguments": {
```

```
            "--class": "GlueApp",
            "--job-language": "scala"
        },
        "MaxRetries": 0,
        "AllocatedCapacity": 10,
        "MaxCapacity": 10.0,
        "GlueVersion": "1.0"
    }
}
```

詳細については、「AWS Glue デベロッパーガイド」の「ジョブ」を参照してください。

- API の詳細については、AWS CLI コマンドリファレンスの「GetJob」を参照してください。

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
const getJob = (jobName) => {
  const client = new GlueClient({});

  const command = new GetJobCommand({
    JobName: jobName,
  });

  return client.send(command);
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「GetJob」を参照してください。

AWS SDK デベロッパーガイドとコード例の詳細なリストについては、「このサービスを AWS SDK で使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI で **GetJobRun** を使用する

次のサンプルコードは、GetJobRun を使用する方法を説明しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- 基本を学ぶ

.NET

SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Get information about a specific AWS Glue job run.
    /// </summary>
    /// <param name="jobName">The name of the job.</param>
    /// <param name="jobRunId">The Id of the job run.</param>
    /// <returns>A JobRun object with information about the job run.</returns>
    public async Task<JobRun> GetJobRunAsync(string jobName, string jobRunId)
    {
        var response = await _amazonGlue.GetJobRunAsync(new GetJobRunRequest
{ JobName = jobName, RunId = jobRunId });
        return response.JobRun;
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「GetJobRun」を参照してください。

C++

SDK for C++

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region in which the bucket was created
(overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::Glue::GlueClient client(clientConfig);

        Aws::Glue::Model::GetJobRunRequest jobRunRequest;
        jobRunRequest.SetJobName(jobName);
        jobRunRequest.SetRunId(jobRunID);

        Aws::Glue::Model::GetJobRunOutcome jobRunOutcome = client.GetJobRun(
                jobRunRequest);

        if (jobRunOutcome.IsSuccess()) {
            std::cout << "Displaying the job run JSON description." << std::endl;
            std::cout
                    <<
jobRunOutcome.GetResult().GetJobRun().Jsonize().View().WriteReadable()
                    << std::endl;
        }
        else {
            std::cerr << "Error get a job run. "
                      << jobRunOutcome.GetError().GetMessage()
                      << std::endl;
        }
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「GetJobRun」を参照してください。

CLI

AWS CLI

ジョブの実行に関する情報を取得するには

次の get-job-run の例では、ジョブ実行に関する情報を取得します。

```
aws glue get-job-run \
    --job-name "Combine legistators data" \
    --run-id jr_012e176506505074d94d761755e5c62538ee1aad6f17d39f527e9140cf0c9a5e
```

出力:

```
{
    "JobRun": {
        "Id":
 "jr_012e176506505074d94d761755e5c62538ee1aad6f17d39f527e9140cf0c9a5e",
        "Attempt": 0,
        "JobName": "Combine legistators data",
        "StartedOn": 1602873931.255,
        "LastModifiedOn": 1602874075.985,
        "CompletedOn": 1602874075.985,
        "JobRunState": "SUCCEEDED",
        "Arguments": {
            "--enable-continuous-cloudwatch-log": "true",
            "--enable-metrics": "",
            "--enable-spark-ui": "true",
            "--job-bookmark-option": "job-bookmark-enable",
            "--spark-event-logs-path": "s3://aws-glue-assets-111122223333-us-
east-1/sparkHistoryLogs/"
        },
        "PredecessorRuns": [],
        "AllocatedCapacity": 10,
        "ExecutionTime": 117,
        "Timeout": 2880,
        "MaxCapacity": 10.0,
        "WorkerType": "G.1X",
        "NumberOfWorkers": 10,
        "LogGroupName": "/aws-glue/jobs",
        "GlueVersion": "2.0"
    }
}
```

詳細については、「AWS Glue デベロッパーガイド」の「ジョブの実行」を参照してください。

- API の詳細については、「AWS CLI コマンドリファレンス」の「GetJobRun」を参照してください。

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
const getJobRun = (jobName, jobRunId) => {
  const client = new GlueClient({});
  const command = new GetJobRunCommand({
    JobName: jobName,
    RunId: jobRunId,
  });

  return client.send(command);
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「GetJobRun」を参照してください。

PHP

SDK for PHP

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

```
        $jobName = 'test-job-' . $uniqid;

        $outputBucketUrl = "s3://$bucketName";
        $runId = $glueService->startJobRun($jobName, $databaseName, $tables,
$outputBucketUrl)['JobRunId'];

        echo "waiting for job";
        do {
            $jobRun = $glueService->getJobRun($jobName, $runId);
            echo ".";
            sleep(10);
        } while (!array_intersect([$jobRun['JobRun']['JobRunState']],
['SUCCEEDED', 'STOPPED', 'FAILED', 'TIMEOUT']));
        echo "\n";

    public function getJobRun($jobName, $runId, $predecessorsIncluded = false):
Result
    {
        return $this->glueClient->getJobRun([
            'JobName' => $jobName,
            'RunId' => $runId,
            'PredecessorsIncluded' => $predecessorsIncluded,
        ]);
    }
```

- API の詳細については、「AWS SDK for PHP API リファレンス」の「GetJobRun」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""
```

```
    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 Glue client.
        """
        self.glue_client = glue_client


    def get_job_run(self, name, run_id):
        """
        Gets information about a single job run.

        :param name: The name of the job definition for the run.
        :param run_id: The ID of the run.
        :return: Information about the run.
        """
        try:
            response = self.glue_client.get_job_run(JobName=name, RunId=run_id)
        except ClientError as err:
            logger.error(
                "Couldn't get job run %s/%s. Here's why: %s: %s",
                name,
                run_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["JobRun"]
```

- API の詳細については、「AWS SDK for Python (Boto3) API リファレンス」の「GetJobRun」を参照してください。

Ruby

SDK for Ruby

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
# The `GlueWrapper` class serves as a wrapper around the AWS Glue API, providing
 a simplified interface for common operations.
# It encapsulates the functionality of the AWS SDK for Glue and provides methods
 for interacting with Glue crawlers, databases, tables, jobs, and S3 resources.
# The class initializes with a Glue client and a logger, allowing it to make API
 calls and log any errors or informational messages.
class GlueWrapper
  def initialize(glue_client, logger)
    @glue_client = glue_client
    @logger = logger
  end

  # Retrieves data for a specific job run.
  #
  # @param job_name [String] The name of the job run to retrieve data for.
  # @return [Glue::Types::GetJobRunResponse]
  def get_job_run(job_name, run_id)
    @glue_client.get_job_run(job_name: job_name, run_id: run_id)
  rescue Aws::Glue::Errors::GlueException => e
    @logger.error("Glue could not get job runs: \n#{e.message}")
  end
```

- API の詳細については、「AWS SDK for Ruby API リファレンス」の「GetJobRun」を参照してください。

Rust

SDK for Rust

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
        let get_job_run = || async {
            Ok::<JobRun, GlueMvpError>(
                glue.get_job_run()
                    .job_name(self.job())
                    .run_id(job_run_id.to_string())
                    .send()
                    .await
                    .map_err(GlueMvpError::from_glue_sdk)?
                    .job_run()
                    .ok_or_else(|| GlueMvpError::Unknown("Failed to get
job_run".into()))?
                    .to_owned(),
            )
        };

        let mut job_run = get_job_run().await?;
        let mut state =
job_run.job_run_state().unwrap_or(&unknown_state).to_owned();

        while matches!(
            state,
            JobRunState::Starting | JobRunState::Stopping | JobRunState::Running
        ) {
            info!(?state, "Waiting for job to finish");
            tokio::time::sleep(self.wait_delay).await;

            job_run = get_job_run().await?;
            state = job_run.job_run_state().unwrap_or(&unknown_state).to_owned();
        }
```

- API の詳細については、「AWS SDK for Rust API リファレンス」の「GetJobRun」を参照してください。

Swift

SDK for Swift

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
import AWSClientRuntime
import AWSGlue

    /// Get information about a specific AWS Glue job run.
    ///
    /// - Parameters:
    ///   - glueClient: The AWS Glue client to use.
    ///   - jobName: The name of the job to return job run data for.
    ///   - id: The run ID of the specific job run to return.
    ///
    /// - Returns: A `GlueClientTypes.JobRun` object describing the state of
    ///   the job run, or `nil` if an error occurs.
    func getJobRun(glueClient: GlueClient, name jobName: String, id: String)
 async -> GlueClientTypes.JobRun? {
        do {
            let output = try await glueClient.getJobRun(
                input: GetJobRunInput(
                    jobName: jobName,
                    runId: id
                )
            )

            return output.jobRun
        } catch {
            return nil
        }
    }
```

- API の詳細については、「AWS SDK for Swift API リファレンス」の「GetJobRun」を参照してください。

AWS SDK デベロッパーガイドとコード例の詳細なリストについては、「このサービスを AWS SDK で使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI で **GetJobRuns** を使用する

次のサンプルコードは、GetJobRuns を使用する方法を説明しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- 基本を学ぶ

.NET

SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
/// <summary>
/// Get information about all AWS Glue runs of a specific job.
/// </summary>
/// <param name="jobName">The name of the job.</param>
/// <returns>A list of JobRun objects.</returns>
public async Task<List<JobRun>> GetJobRunsAsync(string jobName)
{
    var jobRuns = new List<JobRun>();

    var request = new GetJobRunsRequest
    {
        JobName = jobName,
    };
```

```
        // No need to loop to get all the log groups--the SDK does it for us
behind the scenes
        var paginatorForJobRuns =
            _amazonGlue.Paginators.GetJobRuns(request);

        await foreach (var response in paginatorForJobRuns.Responses)
        {
            response.JobRuns.ForEach(jobRun =>
            {
                jobRuns.Add(jobRun);
            });
        }

        return jobRuns;
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「GetJobRuns」を参照してください。

C++

SDK for C++

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region in which the bucket was created
(overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::Glue::GlueClient client(clientConfig);

        Aws::Glue::Model::GetJobRunsRequest getJobRunsRequest;
        getJobRunsRequest.SetJobName(jobName);
```

```
        Aws::String nextToken; // Used for pagination.
        std::vector<Aws::Glue::Model::JobRun> allJobRuns;
        do {
            if (!nextToken.empty()) {
                getJobRunsRequest.SetNextToken(nextToken);
            }
            Aws::Glue::Model::GetJobRunsOutcome jobRunsOutcome =
client.GetJobRuns(
                    getJobRunsRequest);

            if (jobRunsOutcome.IsSuccess()) {
                const std::vector<Aws::Glue::Model::JobRun> &jobRuns =
jobRunsOutcome.GetResult().GetJobRuns();
                allJobRuns.insert(allJobRuns.end(), jobRuns.begin(),
jobRuns.end());

                nextToken = jobRunsOutcome.GetResult().GetNextToken();
            }
            else {
                std::cerr << "Error getting job runs. "
                          << jobRunsOutcome.GetError().GetMessage()
                          << std::endl;
                break;
            }
        } while (!nextToken.empty());
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「GetJobRuns」を参照してください。

CLI

AWS CLI

ジョブのすべてのジョブ実行に関する情報を取得するには

次の `get-job-runs` の例では、ジョブのジョブ実行に関する情報を取得します。

```
aws glue get-job-runs \
    --job-name "my-testing-job"
```

出力:

```
{
    "JobRuns": [
        {
            "Id":
 "jr_012e176506505074d94d761755e5c62538ee1aad6f17d39f527e9140cf0c9a5e",
            "Attempt": 0,
            "JobName": "my-testing-job",
            "StartedOn": 1602873931.255,
            "LastModifiedOn": 1602874075.985,
            "CompletedOn": 1602874075.985,
            "JobRunState": "SUCCEEDED",
            "Arguments": {
                "--enable-continuous-cloudwatch-log": "true",
                "--enable-metrics": "",
                "--enable-spark-ui": "true",
                "--job-bookmark-option": "job-bookmark-enable",
                "--spark-event-logs-path": "s3://aws-glue-assets-111122223333-us-
east-1/sparkHistoryLogs/"
            },
            "PredecessorRuns": [],
            "AllocatedCapacity": 10,
            "ExecutionTime": 117,
            "Timeout": 2880,
            "MaxCapacity": 10.0,
            "WorkerType": "G.1X",
            "NumberOfWorkers": 10,
            "LogGroupName": "/aws-glue/jobs",
            "GlueVersion": "2.0"
        },
        {
            "Id":
 "jr_03cc19ddab11c4e244d3f735567de74ff93b0b3ef468a713ffe73e53d1aec08f_attempt_2",
            "Attempt": 2,
            "PreviousRunId":
 "jr_03cc19ddab11c4e244d3f735567de74ff93b0b3ef468a713ffe73e53d1aec08f_attempt_1",
            "JobName": "my-testing-job",
            "StartedOn": 1602811168.496,
            "LastModifiedOn": 1602811282.39,
            "CompletedOn": 1602811282.39,
            "JobRunState": "FAILED",
            "ErrorMessage": "An error occurred while calling
 o122.pyWriteDynamicFrame.
```

```
                Access Denied (Service: Amazon S3; Status Code: 403; Error Code:
 AccessDenied;
                Request ID: 021AAB703DB20A2D;
                S3 Extended Request ID: teZk24Y09TkXzBvMPG502L5VJBhe9DJuWA9/
TXtuGOqfByajkfL/Tlqt5JBGdEGpigAqzdMDM/U=)",
            "PredecessorRuns": [],
            "AllocatedCapacity": 10,
            "ExecutionTime": 110,
            "Timeout": 2880,
            "MaxCapacity": 10.0,
            "WorkerType": "G.1X",
            "NumberOfWorkers": 10,
            "LogGroupName": "/aws-glue/jobs",
            "GlueVersion": "2.0"
        },
        {
            "Id":
 "jr_03cc19ddab11c4e244d3f735567de74ff93b0b3ef468a713ffe73e53d1aec08f_attempt_1",
            "Attempt": 1,
            "PreviousRunId":
 "jr_03cc19ddab11c4e244d3f735567de74ff93b0b3ef468a713ffe73e53d1aec08f",
            "JobName": "my-testing-job",
            "StartedOn": 1602811020.518,
            "LastModifiedOn": 1602811138.364,
            "CompletedOn": 1602811138.364,
            "JobRunState": "FAILED",
            "ErrorMessage": "An error occurred while calling
 o122.pyWriteDynamicFrame.
                Access Denied (Service: Amazon S3; Status Code: 403; Error Code:
 AccessDenied;
                Request ID: 2671D37856AE7ABB;
                S3 Extended Request ID: RLJCJw20brV
+PpC6GpORahyF2fp9flB5SSb2bTGPnUSPVizLXRl1PN3QZldb+v1o9qRVktNYbW8=)",
            "PredecessorRuns": [],
            "AllocatedCapacity": 10,
            "ExecutionTime": 113,
            "Timeout": 2880,
            "MaxCapacity": 10.0,
            "WorkerType": "G.1X",
            "NumberOfWorkers": 10,
            "LogGroupName": "/aws-glue/jobs",
            "GlueVersion": "2.0"
        }
    ]
```

```
}
```

詳細については、「AWS Glue デベロッパーガイド」の「ジョブの実行」を参照してください。

- API の詳細については、「AWS CLI コマンドリファレンス」の「GetJobRuns」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /**
     * Retrieves the job runs for a given Glue job and prints the status of the
job runs.
     *
     * @param glueClient the Glue client used to make API calls
     * @param jobName    the name of the Glue job to retrieve the job runs for
     */
    public static void getJobRuns(GlueClient glueClient, String jobName) {
        try {
            GetJobRunsRequest runsRequest = GetJobRunsRequest.builder()
                .jobName(jobName)
                .maxResults(20)
                .build();

            boolean jobDone = false;
            while (!jobDone) {
                GetJobRunsResponse response = glueClient.getJobRuns(runsRequest);
                List<JobRun> jobRuns = response.jobRuns();
                for (JobRun jobRun : jobRuns) {
                    String jobState = jobRun.jobRunState().name();
                    if (jobState.compareTo("SUCCEEDED") == 0) {
                        System.out.println(jobName + " has succeeded");
                        jobDone = true;
```

```
                    } else if (jobState.compareTo("STOPPED") == 0) {
                        System.out.println("Job run has stopped");
                        jobDone = true;

                    } else if (jobState.compareTo("FAILED") == 0) {
                        System.out.println("Job run has failed");
                        jobDone = true;

                    } else if (jobState.compareTo("TIMEOUT") == 0) {
                        System.out.println("Job run has timed out");
                        jobDone = true;

                    } else {
                        System.out.println("*** Job run state is " +
jobRun.jobRunState().name());
                        System.out.println("Job run Id is " + jobRun.id());
                        System.out.println("The Glue version is " +
jobRun.glueVersion());
                    }
                    TimeUnit.SECONDS.sleep(5);
                }
            }

        } catch (GlueException e) {
            throw e;
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }
    }
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「GetJobRuns」を参照してください。

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
const getJobRuns = (jobName) => {
  const client = new GlueClient({});
  const command = new GetJobRunsCommand({
    JobName: jobName,
  });

  return client.send(command);
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「GetJobRuns」を参照してください。

PHP

SDK for PHP

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

```
        $jobName = 'test-job-' . $uniqid;

        $jobRuns = $glueService->getJobRuns($jobName);

    public function getJobRuns($jobName, $maxResults = 0, $nextToken = ''):
 Result
    {
```

```
        $arguments = ['JobName' => $jobName];
        if ($maxResults) {
            $arguments['MaxResults'] = $maxResults;
        }
        if ($nextToken) {
            $arguments['NextToken'] = $nextToken;
        }
        return $this->glueClient->getJobRuns($arguments);
    }
```

- API の詳細については、「AWS SDK for PHP API リファレンス」の「GetJobRuns」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""

    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 Glue client.
        """
        self.glue_client = glue_client


    def get_job_runs(self, job_name):
        """
        Gets information about runs that have been performed for a specific job
        definition.

        :param job_name: The name of the job definition to look up.
        :return: The list of job runs.
```

```
        """
        try:
            response = self.glue_client.get_job_runs(JobName=job_name)
        except ClientError as err:
            logger.error(
                "Couldn't get job runs for %s. Here's why: %s: %s",
                job_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["JobRuns"]
```

- API の詳細については、「AWS SDK for Python (Boto3) API リファレンス」の「GetJobRuns」を参照してください。

Ruby

SDK for Ruby

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
# The `GlueWrapper` class serves as a wrapper around the AWS Glue API, providing
 a simplified interface for common operations.
# It encapsulates the functionality of the AWS SDK for Glue and provides methods
 for interacting with Glue crawlers, databases, tables, jobs, and S3 resources.
# The class initializes with a Glue client and a logger, allowing it to make API
 calls and log any errors or informational messages.
class GlueWrapper
  def initialize(glue_client, logger)
    @glue_client = glue_client
    @logger = logger
  end
```

```
# Retrieves a list of job runs for the specified job.
#
# @param job_name [String] The name of the job to retrieve job runs for.
# @return [Array<Aws::Glue::Types::JobRun>]
def get_job_runs(job_name)
  response = @glue_client.get_job_runs(job_name: job_name)
  response.job_runs
rescue Aws::Glue::Errors::GlueException => e
  @logger.error("Glue could not get job runs: \n#{e.message}")
end
```

- API の詳細については、「AWS SDK for Ruby API リファレンス」の「GetJobRuns」を参照してください。

AWS SDK デベロッパーガイドとコード例の詳細なリストについては、「このサービスを AWS SDK で使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI で **GetTables** を使用する

次のサンプルコードは、GetTables を使用する方法を説明しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- 基本を学ぶ

.NET

SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
/// <summary>
```

```
    /// Get a list of tables for an AWS Glue database.
    /// </summary>
    /// <param name="dbName">The name of the database.</param>
    /// <returns>A list of Table objects.</returns>
    public async Task<List<Table>> GetTablesAsync(string dbName)
    {
        var request = new GetTablesRequest { DatabaseName = dbName };
        var tables = new List<Table>();

        // Get a paginator for listing the tables.
        var tablePaginator = _amazonGlue.Paginators.GetTables(request);

        await foreach (var response in tablePaginator.Responses)
        {
            tables.AddRange(response.TableList);
        }

        return tables;
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「GetTables」を参照してください。

C++

SDK for C++

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region in which the bucket was created
(overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::Glue::GlueClient client(clientConfig);
```

```
        Aws::Glue::Model::GetTablesRequest request;
        request.SetDatabaseName(CRAWLER_DATABASE_NAME);
        std::vector<Aws::Glue::Model::Table> all_tables;
        Aws::String nextToken; // Used for pagination.
        do {
            Aws::Glue::Model::GetTablesOutcome outcome =
client.GetTables(request);

            if (outcome.IsSuccess()) {
                const std::vector<Aws::Glue::Model::Table> &tables =
outcome.GetResult().GetTableList();
                all_tables.insert(all_tables.end(), tables.begin(),
tables.end());
                nextToken = outcome.GetResult().GetNextToken();
            }
            else {
                std::cerr << "Error getting the tables. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                deleteAssets(CRAWLER_NAME, CRAWLER_DATABASE_NAME, "", bucketName,
                             clientConfig);
                return false;
            }
        } while (!nextToken.empty());

        std::cout << "The database contains " << all_tables.size()
                  << (all_tables.size() == 1 ?
                      " table." : "tables.") << std::endl;
        std::cout << "Here is a list of the tables in the database.";
        for (size_t index = 0; index < all_tables.size(); ++index) {
            std::cout << "    " << index + 1 << ":  " <<
all_tables[index].GetName()
                      << std::endl;
        }

        if (!all_tables.empty()) {
            int tableIndex = askQuestionForIntRange(
                    "Enter an index to display the database detail ",
                    1, static_cast<int>(all_tables.size()));
            std::cout << all_tables[tableIndex -
1].Jsonize().View().WriteReadable()
                      << std::endl;
```

```
            tableName = all_tables[tableIndex - 1].GetName();
        }
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「GetTables」を参照してください。

CLI

AWS CLI

指定されたデータベース内で、一部またはすべてのテーブルの定義を一覧表示するには

次の get-tables の例では、指定されたデータベース内のテーブルに関する情報を返します。

```
aws glue get-tables --database-name 'tempdb'
```

出力:

```
{
    "TableList": [
        {
            "Name": "my-s3-sink",
            "DatabaseName": "tempdb",
            "CreateTime": 1602730539.0,
            "UpdateTime": 1602730539.0,
            "Retention": 0,
            "StorageDescriptor": {
                "Columns": [
                    {
                        "Name": "sensorid",
                        "Type": "int"
                    },
                    {
                        "Name": "currenttemperature",
                        "Type": "int"
                    },
                    {
                        "Name": "status",
                        "Type": "string"
                    }
```

```
                ],
                "Location": "s3://janetst-bucket-01/test-s3-output/",
                "Compressed": false,
                "NumberOfBuckets": 0,
                "SerdeInfo": {
                    "SerializationLibrary": "org.openx.data.jsonserde.JsonSerDe"
                },
                "SortColumns": [],
                "StoredAsSubDirectories": false
            },
            "Parameters": {
                "classification": "json"
            },
            "CreatedBy": "arn:aws:iam::007436865787:user/JRSTERN",
            "IsRegisteredWithLakeFormation": false,
            "CatalogId": "007436865787"
        },
        {
            "Name": "s3-source",
            "DatabaseName": "tempdb",
            "CreateTime": 1602730658.0,
            "UpdateTime": 1602730658.0,
            "Retention": 0,
            "StorageDescriptor": {
                "Columns": [
                    {
                        "Name": "sensorid",
                        "Type": "int"
                    },
                    {
                        "Name": "currenttemperature",
                        "Type": "int"
                    },
                    {
                        "Name": "status",
                        "Type": "string"
                    }
                ],
                "Location": "s3://janetst-bucket-01/",
                "Compressed": false,
                "NumberOfBuckets": 0,
                "SortColumns": [],
                "StoredAsSubDirectories": false
            },
```

```
            "Parameters": {
                "classification": "json"
            },
            "CreatedBy": "arn:aws:iam::007436865787:user/JRSTERN",
            "IsRegisteredWithLakeFormation": false,
            "CatalogId": "007436865787"
        },
        {
            "Name": "test-kinesis-input",
            "DatabaseName": "tempdb",
            "CreateTime": 1601507001.0,
            "UpdateTime": 1601507001.0,
            "Retention": 0,
            "StorageDescriptor": {
                "Columns": [
                    {
                        "Name": "sensorid",
                        "Type": "int"
                    },
                    {
                        "Name": "currenttemperature",
                        "Type": "int"
                    },
                    {
                        "Name": "status",
                        "Type": "string"
                    }
                ],
                "Location": "my-testing-stream",
                "Compressed": false,
                "NumberOfBuckets": 0,
                "SerdeInfo": {
                    "SerializationLibrary": "org.openx.data.jsonserde.JsonSerDe"
                },
                "SortColumns": [],
                "Parameters": {
                    "kinesisUrl": "https://kinesis.us-east-1.amazonaws.com",
                    "streamName": "my-testing-stream",
                    "typeOfData": "kinesis"
                },
                "StoredAsSubDirectories": false
            },
            "Parameters": {
                "classification": "json"
```

```
            },
            "CreatedBy": "arn:aws:iam::007436865787:user/JRSTERN",
            "IsRegisteredWithLakeFormation": false,
            "CatalogId": "007436865787"
        }
    ]
}
```

詳細については、「AWS Glue デベロッパーガイド」の「AWS Glue データカタログでのテーブルの定義」を参照してください。

- API の詳細については、「AWS CLI コマンドリファレンス」の「GetTables」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /**
     * Retrieves the names of the tables in the specified Glue database.
     *
     * @param glueClient the Glue client to use for the operation
     * @param dbName     the name of the Glue database to retrieve the table
names from
     * @return the name of the first table retrieved, or an empty string if no
tables were found
     */
    public static String getGlueTables(GlueClient glueClient, String dbName) {
        String myTableName = "";
        try {
            GetTablesRequest tableRequest = GetTablesRequest.builder()
                .databaseName(dbName)
                .build();
```

```
            GetTablesResponse response = glueClient.getTables(tableRequest);
            List<Table> tables = response.tableList();
            if (tables.isEmpty()) {
                System.out.println("No tables were returned");
            } else {
                for (Table table : tables) {
                    myTableName = table.name();
                    System.out.println("Table name is: " + myTableName);
                }
            }

        } catch (GlueException e) {
            throw e;
        }
        return myTableName;
    }
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「GetTables」を参照してください。

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
const getTables = (databaseName) => {
  const client = new GlueClient({});

  const command = new GetTablesCommand({
    DatabaseName: databaseName,
  });

  return client.send(command);
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「GetTables」を参照してください。

PHP

SDK for PHP

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

```
        $databaseName = "doc-example-database-$uniqid";

        $tables = $glueService->getTables($databaseName);

    public function getTables($databaseName): Result
    {
        return $this->glueClient->getTables([
            'DatabaseName' => $databaseName,
        ]);
    }
```

- API の詳細については、「AWS SDK for PHP API リファレンス」の「GetTables」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
class GlueWrapper:
```

```
    """Encapsulates AWS Glue actions."""

    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 Glue client.
        """
        self.glue_client = glue_client


    def get_tables(self, db_name):
        """
        Gets a list of tables in a Data Catalog database.

        :param db_name: The name of the database to query.
        :return: The list of tables in the database.
        """
        try:
            response = self.glue_client.get_tables(DatabaseName=db_name)
        except ClientError as err:
            logger.error(
                "Couldn't get tables %s. Here's why: %s: %s",
                db_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["TableList"]
```

- API の詳細については、「AWS SDK for Python (Boto3) API リファレンス」の「GetTables」を参照してください。

Ruby

SDK for Ruby

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
# The `GlueWrapper` class serves as a wrapper around the AWS Glue API, providing
 a simplified interface for common operations.
# It encapsulates the functionality of the AWS SDK for Glue and provides methods
 for interacting with Glue crawlers, databases, tables, jobs, and S3 resources.
# The class initializes with a Glue client and a logger, allowing it to make API
 calls and log any errors or informational messages.
class GlueWrapper
  def initialize(glue_client, logger)
    @glue_client = glue_client
    @logger = logger
  end

  # Retrieves a list of tables in the specified database.
  #
  # @param db_name [String] The name of the database to retrieve tables from.
  # @return [Array<Aws::Glue::Types::Table>]
  def get_tables(db_name)
    response = @glue_client.get_tables(database_name: db_name)
    response.table_list
  rescue Aws::Glue::Errors::GlueException => e
    @logger.error("Glue could not get tables #{db_name}: \n#{e.message}")
    raise
  end
```

- API の詳細については、「AWS SDK for Ruby API リファレンス」の「GetTables」を参照してください。

Rust

SDK for Rust

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
        let tables = glue
            .get_tables()
            .database_name(self.database())
            .send()
            .await
            .map_err(GlueMvpError::from_glue_sdk)?;

        let tables = tables.table_list();
```

- API の詳細については、「AWS SDK for Rust API リファレンス」の「GetTables」を参照してください。

Swift

SDK for Swift

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
import AWSClientRuntime
import AWSGlue

    /// Returns a list of the tables in the specified database.
    ///
    /// - Parameters:
    ///   - glueClient: The AWS Glue client to use.
```

```
    ///   - databaseName: The name of the database whose tables are to be
    ///     returned.
    ///
    /// - Returns: An array of `GlueClientTypes.Table` objects, each
    ///   describing one table in the named database. An empty array indicates
    ///   that there are either no tables in the database, or an error
    ///   occurred before any tables could be found.
    func getTablesInDatabase(glueClient: GlueClient, databaseName: String) async
-> [GlueClientTypes.Table] {
        var tables: [GlueClientTypes.Table] = []
        var nextToken: String?

        repeat {
            do {
                let output = try await glueClient.getTables(
                    input: GetTablesInput(
                        databaseName: databaseName,
                        nextToken: nextToken
                    )
                )

                guard let tableList = output.tableList else {
                    return tables
                }

                tables = tables + tableList
                nextToken = output.nextToken
            } catch {
                return tables
            }
        } while nextToken != nil

        return tables
    }
```

- API の詳細については、「AWS SDK for Swift API リファレンス」の「GetTables」を参照してください。

AWS SDK デベロッパーガイドとコード例の詳細なリストについては、「このサービスを AWS SDK で使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK で **ListJobs** を使用する

次のサンプルコードは、ListJobs を使用する方法を説明しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- 基本を学ぶ

.NET

SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// List AWS Glue jobs using a paginator.
    /// </summary>
    /// <returns>A list of AWS Glue job names.</returns>
    public async Task<List<string>> ListJobsAsync()
    {
        var jobNames = new List<string>();

        var listJobsPaginator = _amazonGlue.Paginators.ListJobs(new
ListJobsRequest { MaxResults = 10 });
        await foreach (var response in listJobsPaginator.Responses)
        {
            jobNames.AddRange(response.JobNames);
        }

        return jobNames;
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「ListJobs」を参照してください。

C++

SDK for C++

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
       Aws::Client::ClientConfiguration clientConfig;
       // Optional: Set to the AWS Region in which the bucket was created
(overrides config file).
       // clientConfig.region = "us-east-1";

   Aws::Glue::GlueClient client(clientConfig);

       Aws::Glue::Model::ListJobsRequest listJobsRequest;

       Aws::String nextToken;
       std::vector<Aws::String> allJobNames;

       do {
           if (!nextToken.empty()) {
               listJobsRequest.SetNextToken(nextToken);
           }
           Aws::Glue::Model::ListJobsOutcome listRunsOutcome = client.ListJobs(
                   listJobsRequest);

           if (listRunsOutcome.IsSuccess()) {
               const std::vector<Aws::String> &jobNames =
listRunsOutcome.GetResult().GetJobNames();
               allJobNames.insert(allJobNames.end(), jobNames.begin(),
jobNames.end());
               nextToken = listRunsOutcome.GetResult().GetNextToken();
           }
           else {
               std::cerr << "Error listing jobs. "
                         << listRunsOutcome.GetError().GetMessage()
                         << std::endl;
           }
       } while (!nextToken.empty());
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「ListJobs」を参照してください。

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
const listJobs = () => {
  const client = new GlueClient({});

  const command = new ListJobsCommand({});

  return client.send(command);
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「ListJobs」を参照してください。

PHP

SDK for PHP

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

```
        $jobs = $glueService->listJobs();
        echo "Current jobs:\n";
```

```
        foreach ($jobs['JobNames'] as $jobsName) {
            echo "{$jobsName}\n";
        }

    public function listJobs($maxResults = null, $nextToken = null, $tags = []):
Result
    {
        $arguments = [];
        if ($maxResults) {
            $arguments['MaxResults'] = $maxResults;
        }
        if ($nextToken) {
            $arguments['NextToken'] = $nextToken;
        }
        if (!empty($tags)) {
            $arguments['Tags'] = $tags;
        }
        return $this->glueClient->listJobs($arguments);
    }
```

- API の詳細については、「AWS SDK for PHP API リファレンス」の「ListJobs」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""

    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 Glue client.
        """
```

```
        self.glue_client = glue_client


    def list_jobs(self):
        """
        Lists the names of job definitions in your account.

        :return: The list of job definition names.
        """
        try:
            response = self.glue_client.list_jobs()
        except ClientError as err:
            logger.error(
                "Couldn't list jobs. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["JobNames"]
```

- API の詳細については、「AWS SDK for Python (Boto3) API リファレンス」の「ListJobs」を参照してください。

Ruby

SDK for Ruby

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
# The `GlueWrapper` class serves as a wrapper around the AWS Glue API, providing
 a simplified interface for common operations.
# It encapsulates the functionality of the AWS SDK for Glue and provides methods
 for interacting with Glue crawlers, databases, tables, jobs, and S3 resources.
```

```
# The class initializes with a Glue client and a logger, allowing it to make API
 calls and log any errors or informational messages.
class GlueWrapper
  def initialize(glue_client, logger)
    @glue_client = glue_client
    @logger = logger
  end

  # Retrieves a list of jobs in AWS Glue.
  #
  # @return [Aws::Glue::Types::ListJobsResponse]
  def list_jobs
    @glue_client.list_jobs
  rescue Aws::Glue::Errors::GlueException => e
    @logger.error("Glue could not list jobs: \n#{e.message}")
    raise
  end
```

- API の詳細については、「AWS SDK for Ruby API リファレンス」の「ListJobs」を参照してください。

Rust

SDK for Rust

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
        let mut list_jobs = glue.list_jobs().into_paginator().send();
        while let Some(list_jobs_output) = list_jobs.next().await {
            match list_jobs_output {
                Ok(list_jobs) => {
                    let names = list_jobs.job_names();
                    info!(?names, "Found these jobs")
                }
                Err(err) => return Err(GlueMvpError::from_glue_sdk(err)),
            }
```

```
        }
```

- API の詳細については、「AWS SDK for Rust API リファレンス」の「ListJobs」を参照してください。

Swift

SDK for Swift

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
import AWSClientRuntime
import AWSGlue

    /// Return a list of the AWS Glue jobs listed on the user's account.
    ///
    /// - Parameters:
    ///   - glueClient: The AWS Glue client to use.
    ///   - maxJobs: The maximum number of jobs to return (default: 100).
    ///
    /// - Returns: An array of strings listing the names of all available AWS
    ///   Glue jobs.
    func listJobs(glueClient: GlueClient, maxJobs: Int = 100) async -> [String] {
        var jobList: [String] = []
        var nextToken: String?

        repeat {
            do {
                let output = try await glueClient.listJobs(
                    input: ListJobsInput(
                        maxResults: maxJobs,
                        nextToken: nextToken
                    )
                )

                guard let jobs = output.jobNames else {
```

```
                return jobList
            }

            jobList = jobList + jobs
            nextToken = output.nextToken
        } catch {
            return jobList
        }
    } while (nextToken != nil)

    return jobList
}
```

- API の詳細については、「AWS SDK for Swift API リファレンス」の「ListJobs」を参照してください。

AWS SDK デベロッパーガイドとコード例の詳細なリストについては、「このサービスを AWS SDK で使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI で **StartCrawler** を使用する

次のサンプルコードは、StartCrawler を使用する方法を説明しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- 基本を学ぶ

.NET

SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Start an AWS Glue crawler.
    /// </summary>
    /// <param name="crawlerName">The name of the crawler.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> StartCrawlerAsync(string crawlerName)
    {
        var crawlerRequest = new StartCrawlerRequest
        {
            Name = crawlerName,
        };

        var response = await _amazonGlue.StartCrawlerAsync(crawlerRequest);

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「StartCrawler」を参照してください。

C++

SDK for C++

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region in which the bucket was created
(overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::Glue::GlueClient client(clientConfig);

        Aws::Glue::Model::StartCrawlerRequest request;
        request.SetName(CRAWLER_NAME);
```

```
       Aws::Glue::Model::StartCrawlerOutcome outcome =
client.StartCrawler(request);


       if (outcome.IsSuccess() || (Aws::Glue::GlueErrors::CRAWLER_RUNNING ==
                                   outcome.GetError().GetErrorType())) {
           if (!outcome.IsSuccess()) {
               std::cout << "Crawler was already started." << std::endl;
           }
           else {
               std::cout << "Successfully started crawler." << std::endl;
           }

           std::cout << "This may take a while to run." << std::endl;

           Aws::Glue::Model::CrawlerState crawlerState =
Aws::Glue::Model::CrawlerState::NOT_SET;
           int iterations = 0;
           while (Aws::Glue::Model::CrawlerState::READY != crawlerState) {
               std::this_thread::sleep_for(std::chrono::seconds(1));
               ++iterations;
               if ((iterations % 10) == 0) { // Log status every 10 seconds.
                   std::cout << "Crawler status " <<

Aws::Glue::Model::CrawlerStateMapper::GetNameForCrawlerState(
                                       crawlerState)
                             << ". After " << iterations
                             << " seconds elapsed."
                             << std::endl;
               }
               Aws::Glue::Model::GetCrawlerRequest getCrawlerRequest;
               getCrawlerRequest.SetName(CRAWLER_NAME);

               Aws::Glue::Model::GetCrawlerOutcome getCrawlerOutcome =
client.GetCrawler(
                       getCrawlerRequest);

               if (getCrawlerOutcome.IsSuccess()) {
                   crawlerState =
getCrawlerOutcome.GetResult().GetCrawler().GetState();
               }
               else {
                   std::cerr << "Error getting crawler.  "
```

```
                                << getCrawlerOutcome.GetError().GetMessage() <<
std::endl;
                    break;
                }
            }

            if (Aws::Glue::Model::CrawlerState::READY == crawlerState) {
                std::cout << "Crawler finished running after " << iterations
                          << " seconds."
                          << std::endl;
            }
        }
        else {
            std::cerr << "Error starting a crawler.  "
                      << outcome.GetError().GetMessage()
                      << std::endl;

            deleteAssets(CRAWLER_NAME, CRAWLER_DATABASE_NAME, "", bucketName,
                         clientConfig);
            return false;
        }
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「StartCrawler」を参照してください。

CLI

AWS CLI

クローラーを開始するには

次の `start-crawler` の例では、クローラーを開始します。

```
aws glue start-crawler --name my-crawler
```

出力:

```
None
```

詳細については、「AWS Glue デベロッパーガイド」の「クローラーの定義」を参照してください。

- API の詳細については、「AWS CLI コマンドリファレンス」の「StartCrawler」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /**
     * Starts a specific AWS Glue crawler.
     *
     * @param glueClient  the AWS Glue client to use for the crawler operation
     * @param crawlerName the name of the crawler to start
     * @throws GlueException if there is an error starting the crawler
     */
    public static void startSpecificCrawler(GlueClient glueClient, String
crawlerName) {
        try {
            StartCrawlerRequest crawlerRequest = StartCrawlerRequest.builder()
                .name(crawlerName)
                .build();

            glueClient.startCrawler(crawlerRequest);
            System.out.println(crawlerName + " was successfully started!");

        } catch (GlueException e) {
            throw e;
        }
    }
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「StartCrawler」を参照してください。

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
const startCrawler = (name) => {
  const client = new GlueClient({});

  const command = new StartCrawlerCommand({
    Name: name,
  });

  return client.send(command);
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「StartCrawler」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
suspend fun startSpecificCrawler(crawlerName: String?) {
    val request =
        StartCrawlerRequest {
            name = crawlerName
        }
```

```
    GlueClient { region = "us-west-2" }.use { glueClient ->
        glueClient.startCrawler(request)
        println("$crawlerName was successfully started.")
    }
}
```

- API の詳細については、「AWS SDK for Kotlin API リファレンス」の「StartCrawler」を参照してください。

PHP

SDK for PHP

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

```
        $crawlerName = "example-crawler-test-" . $uniqid;

        $databaseName = "doc-example-database-$uniqid";

        $glueService->startCrawler($crawlerName);

    public function startCrawler($crawlerName): Result
    {
        return $this->glueClient->startCrawler([
            'Name' => $crawlerName,
        ]);
    }
```

- API の詳細については、「AWS SDK for PHP API リファレンス」の「StartCrawler」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""

    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 Glue client.
        """
        self.glue_client = glue_client


    def start_crawler(self, name):
        """
        Starts a crawler. The crawler crawls its configured target and creates
        metadata that describes the data it finds in the target data source.

        :param name: The name of the crawler to start.
        """
        try:
            self.glue_client.start_crawler(Name=name)
        except ClientError as err:
            logger.error(
                "Couldn't start crawler %s. Here's why: %s: %s",
                name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- API の詳細については、「AWS SDK for Python (Boto3) API リファレンス」の「StartCrawler」を参照してください。

Ruby

SDK for Ruby

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
# The `GlueWrapper` class serves as a wrapper around the AWS Glue API, providing
 a simplified interface for common operations.
# It encapsulates the functionality of the AWS SDK for Glue and provides methods
 for interacting with Glue crawlers, databases, tables, jobs, and S3 resources.
# The class initializes with a Glue client and a logger, allowing it to make API
 calls and log any errors or informational messages.
class GlueWrapper
  def initialize(glue_client, logger)
    @glue_client = glue_client
    @logger = logger
  end

  # Starts a crawler with the specified name.
  #
  # @param name [String] The name of the crawler to start.
  # @return [void]
  def start_crawler(name)
    @glue_client.start_crawler(name: name)
  rescue Aws::Glue::Errors::ServiceError => e
    @logger.error("Glue could not start crawler #{name}: \n#{e.message}")
    raise
  end
```

- API の詳細については、「AWS SDK for Ruby API リファレンス」の「StartCrawler」を参照してください。

Rust

SDK for Rust

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
        let start_crawler =
glue.start_crawler().name(self.crawler()).send().await;

        match start_crawler {
            Ok(_) => Ok(()),
            Err(err) => {
                let glue_err: aws_sdk_glue::Error = err.into();
                match glue_err {
                    aws_sdk_glue::Error::CrawlerRunningException(_) => Ok(()),
                    _ => Err(GlueMvpError::GlueSdk(glue_err)),
                }
            }
        }?;
```

- API の詳細については、「AWS SDK for Rust API リファレンス」の「StartCrawler」を参照してください。

Swift

SDK for Swift

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
import AWSClientRuntime
```

```
import AWSGlue

    /// Start running an AWS Glue crawler.
    ///
    /// - Parameters:
    ///   - glueClient: The AWS Glue client to use when starting the crawler.
    ///   - name: The name of the crawler to start running.
    ///
    /// - Returns: `true` if the crawler is started successfully, otherwise
 `false`.
    func startCrawler(glueClient: GlueClient, name: String) async -> Bool {
        do {
            _ = try await glueClient.startCrawler(
                input: StartCrawlerInput(name: name)
            )
        } catch {
            print("*** An unexpected error occurred starting the crawler.")
            return false
        }

        return true
    }
```

- API の詳細については、「AWS SDK for Swift API リファレンス」の「StartCrawler」を参照してください。

AWS SDK デベロッパーガイドとコード例の詳細なリストについては、「このサービスを AWS SDK で使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI で **StartJobRun** を使用する

次のサンプルコードは、StartJobRun を使用する方法を説明しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- 基本を学ぶ

.NET

SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
/// <summary>
/// Start an AWS Glue job run.
/// </summary>
/// <param name="jobName">The name of the job.</param>
/// <returns>A string representing the job run Id.</returns>
public async Task<string> StartJobRunAsync(
    string jobName,
    string inputDatabase,
    string inputTable,
    string bucketName)
{
    var request = new StartJobRunRequest
    {
        JobName = jobName,
        Arguments = new Dictionary<string, string>
        {
            {"--input_database", inputDatabase},
            {"--input_table", inputTable},
            {"--output_bucket_url", $"s3://{bucketName}/"}
        }
    };

    var response = await _amazonGlue.StartJobRunAsync(request);
    return response.JobRunId;
}
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「StartJobRun」を参照してください。

C++

SDK for C++

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
       Aws::Client::ClientConfiguration clientConfig;
       // Optional: Set to the AWS Region in which the bucket was created
(overrides config file).
       // clientConfig.region = "us-east-1";

   Aws::Glue::GlueClient client(clientConfig);

       Aws::Glue::Model::StartJobRunRequest request;
       request.SetJobName(JOB_NAME);

       Aws::Map<Aws::String, Aws::String> arguments;
       arguments["--input_database"] = CRAWLER_DATABASE_NAME;
       arguments["--input_table"] = tableName;
       arguments["--output_bucket_url"] = Aws::String("s3://") + bucketName +
"/";
       request.SetArguments(arguments);

       Aws::Glue::Model::StartJobRunOutcome outcome =
client.StartJobRun(request);

       if (outcome.IsSuccess()) {
           std::cout << "Successfully started the job." << std::endl;

           Aws::String jobRunId = outcome.GetResult().GetJobRunId();

           int iterator = 0;
           bool done = false;
           while (!done) {
               ++iterator;
               std::this_thread::sleep_for(std::chrono::seconds(1));
               Aws::Glue::Model::GetJobRunRequest jobRunRequest;
               jobRunRequest.SetJobName(JOB_NAME);
```

```
                jobRunRequest.SetRunId(jobRunId);

                Aws::Glue::Model::GetJobRunOutcome jobRunOutcome =
client.GetJobRun(
                        jobRunRequest);

                if (jobRunOutcome.IsSuccess()) {
                    const Aws::Glue::Model::JobRun &jobRun =
jobRunOutcome.GetResult().GetJobRun();
                    Aws::Glue::Model::JobRunState jobRunState =
jobRun.GetJobRunState();

                    if ((jobRunState == Aws::Glue::Model::JobRunState::STOPPED)
||
                        (jobRunState == Aws::Glue::Model::JobRunState::FAILED) ||
                        (jobRunState == Aws::Glue::Model::JobRunState::TIMEOUT))
{
                        std::cerr << "Error running job. "
                                  << jobRun.GetErrorMessage()
                                  << std::endl;
                        deleteAssets(CRAWLER_NAME, CRAWLER_DATABASE_NAME,
JOB_NAME,
                                     bucketName,
                                     clientConfig);
                        return false;
                    }
                    else if (jobRunState ==
                             Aws::Glue::Model::JobRunState::SUCCEEDED) {
                        std::cout << "Job run succeeded after  " << iterator <<
                                  " seconds elapsed." << std::endl;
                        done = true;
                    }
                    else if ((iterator % 10) == 0) { // Log status every 10
seconds.
                        std::cout << "Job run status " <<

Aws::Glue::Model::JobRunStateMapper::GetNameForJobRunState(
                                         jobRunState) <<
                                  ". " << iterator <<
                                  " seconds elapsed." << std::endl;
                    }
                }
                else {
                    std::cerr << "Error retrieving job run state. "
```

```
                                << jobRunOutcome.GetError().GetMessage()
                                << std::endl;
                        deleteAssets(CRAWLER_NAME, CRAWLER_DATABASE_NAME, JOB_NAME,
                                     bucketName, clientConfig);
                        return false;
                    }
                }
            }
            else {
                std::cerr << "Error starting a job. " <<
outcome.GetError().GetMessage()
                          << std::endl;
                deleteAssets(CRAWLER_NAME, CRAWLER_DATABASE_NAME, JOB_NAME,
bucketName,
                             clientConfig);
                return false;
            }
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「StartJobRun」を参照してください。

CLI

AWS CLI

ジョブの実行を開始するには

次の `start-job-run` の例ではジョブを開始します。

```
aws glue start-job-run \
    --job-name my-job
```

出力:

```
{
    "JobRunId":
 "jr_22208b1f44eb5376a60569d4b21dd20fcb8621e1a366b4e7b2494af764b82ded"
}
```

詳細については、「AWS Glue デベロッパーガイド」の「ジョブの作成」を参照してください。

- API の詳細については、「AWS CLI コマンドリファレンス」の「StartJobRun」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
   /**
    * Starts a job run in AWS Glue.
    *
    * @param glueClient    the AWS Glue client to use for the job run
    * @param jobName       the name of the Glue job to run
    * @param inputDatabase the name of the input database
    * @param inputTable    the name of the input table
    * @param outBucket     the URL of the output S3 bucket
    * @throws GlueException if there is an error starting the job run
    */
   public static void startJob(GlueClient glueClient, String jobName, String
inputDatabase, String inputTable,
                               String outBucket) {
       try {
           Map<String, String> myMap = new HashMap<>();
           myMap.put("--input_database", inputDatabase);
           myMap.put("--input_table", inputTable);
           myMap.put("--output_bucket_url", outBucket);

           StartJobRunRequest runRequest = StartJobRunRequest.builder()
               .workerType(WorkerType.G_1_X)
               .numberOfWorkers(10)
               .arguments(myMap)
               .jobName(jobName)
               .build();

           StartJobRunResponse response = glueClient.startJobRun(runRequest);
```

```
            System.out.println("The request Id of the job is " +
 response.responseMetadata().requestId());

        } catch (GlueException e) {
            throw e;
        }
    }
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「StartJobRun」を参照してください。

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
const startJobRun = (jobName, dbName, tableName, bucketName) => {
  const client = new GlueClient({});

  const command = new StartJobRunCommand({
    JobName: jobName,
    Arguments: {
      "--input_database": dbName,
      "--input_table": tableName,
      "--output_bucket_url": `s3://${bucketName}/`,
    },
  });

  return client.send(command);
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「StartJobRun」を参照してください。

PHP

SDK for PHP

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コードサンプルリポジトリでの設定と実行の方法を確認してください。

```
        $jobName = 'test-job-' . $uniqid;

        $databaseName = "doc-example-database-$uniqid";

        $tables = $glueService->getTables($databaseName);

        $outputBucketUrl = "s3://$bucketName";
        $runId = $glueService->startJobRun($jobName, $databaseName, $tables,
$outputBucketUrl)['JobRunId'];

    public function startJobRun($jobName, $databaseName, $tables,
$outputBucketUrl): Result
    {
        return $this->glueClient->startJobRun([
            'JobName' => $jobName,
            'Arguments' => [
                'input_database' => $databaseName,
                'input_table' => $tables['TableList'][0]['Name'],
                'output_bucket_url' => $outputBucketUrl,
                '--input_database' => $databaseName,
                '--input_table' => $tables['TableList'][0]['Name'],
                '--output_bucket_url' => $outputBucketUrl,
            ],
        ]);
    }
```

- API の詳細については、「AWS SDK for PHP API リファレンス」の「StartJobRun」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""

    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 Glue client.
        """
        self.glue_client = glue_client


    def start_job_run(self, name, input_database, input_table,
 output_bucket_name):
        """
        Starts a job run. A job run extracts data from the source, transforms it,
        and loads it to the output bucket.

        :param name: The name of the job definition.
        :param input_database: The name of the metadata database that contains
 tables
                               that describe the source data. This is typically
 created
                               by a crawler.
        :param input_table: The name of the table in the metadata database that
                            describes the source data.
        :param output_bucket_name: The S3 bucket where the output is written.
        :return: The ID of the job run.
        """
        try:
            # The custom Arguments that are passed to this function are used by
 the
            # Python ETL script to determine the location of input and output
 data.
```

```
            response = self.glue_client.start_job_run(
                JobName=name,
                Arguments={
                    "--input_database": input_database,
                    "--input_table": input_table,
                    "--output_bucket_url": f"s3://{output_bucket_name}/",
                },
            )
        except ClientError as err:
            logger.error(
                "Couldn't start job run %s. Here's why: %s: %s",
                name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["JobRunId"]
```

- API の詳細については、「AWS SDK for Python (Boto3) API リファレンス」の「StartJobRun」を参照してください。

Ruby

SDK for Ruby

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
# The `GlueWrapper` class serves as a wrapper around the AWS Glue API, providing
 a simplified interface for common operations.
# It encapsulates the functionality of the AWS SDK for Glue and provides methods
 for interacting with Glue crawlers, databases, tables, jobs, and S3 resources.
# The class initializes with a Glue client and a logger, allowing it to make API
 calls and log any errors or informational messages.
```

```
class GlueWrapper
  def initialize(glue_client, logger)
    @glue_client = glue_client
    @logger = logger
  end

  # Starts a job run for the specified job.
  #
  # @param name [String] The name of the job to start the run for.
  # @param input_database [String] The name of the input database for the job.
  # @param input_table [String] The name of the input table for the job.
  # @param output_bucket_name [String] The name of the output S3 bucket for the
 job.
  # @return [String] The ID of the started job run.
  def start_job_run(name, input_database, input_table, output_bucket_name)
    response = @glue_client.start_job_run(
      job_name: name,
      arguments: {
        '--input_database': input_database,
        '--input_table': input_table,
        '--output_bucket_url': "s3://#{output_bucket_name}/"
      }
    )
    response.job_run_id
  rescue Aws::Glue::Errors::GlueException => e
    @logger.error("Glue could not start job run #{name}: \n#{e.message}")
    raise
  end
```

- API の詳細については、「AWS SDK for Ruby API リファレンス」の「StartJobRun」を参照してください。

Rust

SDK for Rust

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
        let job_run_output = glue
            .start_job_run()
            .job_name(self.job())
            .arguments("--input_database", self.database())
            .arguments(
                "--input_table",
                self.tables
                    .first()
                    .ok_or_else(|| GlueMvpError::Unknown("Missing crawler
table".into()))?
                    .name(),
            )
            .arguments("--output_bucket_url", self.bucket())
            .send()
            .await
            .map_err(GlueMvpError::from_glue_sdk)?;

        let job = job_run_output
            .job_run_id()
            .ok_or_else(|| GlueMvpError::Unknown("Missing run id from just
started job".into()))?
            .to_string();
```

- API の詳細については、「AWS SDK for Rust API リファレンス」の「StartJobRun」を参照してください。

Swift

SDK for Swift

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
import AWSClientRuntime
import AWSGlue

    /// Start an AWS Glue job run.
```

```
    ///
    /// - Parameters:
    ///   - glueClient: The AWS Glue client to use.
    ///   - jobName: The name of the job to run.
    ///   - databaseName: The name of the AWS Glue database to run the job
against.
    ///   - tableName: The name of the table in the database to run the job
against.
    ///   - outputURL: The AWS S3 URI of the bucket location into which to
    ///     write the resulting output.
    ///
    /// - Returns: `true` if the job run is started successfully, otherwise
`false`.
    func startJobRun(glueClient: GlueClient, name jobName: String, databaseName:
String,
                     tableName: String, outputURL: String) async -> String? {
        do {
            let output = try await glueClient.startJobRun(
                input: StartJobRunInput(
                    arguments: [
                        "--input_database": databaseName,
                        "--input_table": tableName,
                        "--output_bucket_url": outputURL
                    ],
                    jobName: jobName,
                    numberOfWorkers: 10,
                    workerType: .g1x
                )
            )

            guard let id = output.jobRunId else {
                return nil
            }

            return id
        } catch {
            return nil
        }
    }
```

- API の詳細については、「AWS SDK for Swift API リファレンス」の「StartJobRun」を参照してください。

AWS SDK デベロッパーガイドとコード例の詳細なリストについては、「このサービスを AWS SDK で使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

「AWS Glue」 のセキュリティ

AWS では、クラウドのセキュリティが最優先事項です。セキュリティを最も重視する組織の要件を満たすために構築された 「AWS」 のデータセンターとネットワークアーキテクチャは、お客様に大きく貢献します。

セキュリティは、「AWS」 と顧客の間の責任共有です。責任共有モデルでは、この責任がクラウドのセキュリティおよびクラウド内のセキュリティとして説明されています。

- クラウドのセキュリティ - 「AWS」 は、「AWS」 クラウドで 「AWS」 のサービスを実行するインフラストラクチャを保護する責任を負います。また、「AWS」 は、使用するサービスを安全に提供します。「AWS」 コンプライアンスプログラムの一環として、サードパーティーの監査が定期的にセキュリティの有効性をテストおよび検証しています。「AWS Glue」 に適用されるコンプライアンスプログラムの詳細については、「コンプライアンスプログラムによる対象範囲内の 「AWS」 のサービス」を参照してください。
- クラウド内のセキュリティ - ユーザーの責任は、使用する 「AWS」 のサービスに応じて異なります。また、お客様は、データの機密性、お客様の会社の要件、および適用される法律および規制など、その他の要因についても責任を負います。

このドキュメントは、AWS Glue を使用する際に責任共有モデルを適用する方法を理解するのに役立ちます。以下のトピックでは、セキュリティおよびコンプライアンスの目的を達成するために AWS Glue を設定する方法を示します。また、AWS Glue リソースのモニタリングや保護に役立つ、その他 AWS サービスの使用方法についても説明します。

トピック

- AWS Glue でのデータ保護
- AWS Glue のアイデンティティとアクセスの管理
- AWS Lake Formation アクセスコントロールモデル
- AWS Glue での Amazon S3 Access Grants の使用
- AWS Glue でのログ記録とモニタリング
- AWS Glue のコンプライアンス検証
- AWS Glue での耐障害性
- AWS Glue 内のインフラストラクチャセキュリティ

AWS Glue でのデータ保護

AWS Glue には、データを保護するために設計された機能がいくつか用意されています。

トピック

- 保管中のデータの暗号化
- Encrypting data in transit
- FIPS 準拠
- キーの管理
- AWS の他のサービスへの AWS Glue の依存関係
- 開発エンドポイント

保管中のデータの暗号化

AWS Glue は、AWS Glue Studio でビジュアル ETL ジョブを作成する と 開発エンドポイントを使用してスクリプトを開発する のために保管時のデータの暗号化をサポートしています。AWS Key Management Service (AWS KMS) キーを使用して保管時の暗号化されたデータを書き込むように ETL (抽出、変換、ロード) ジョブと開発エンドポイントを設定できます。また、AWS KMS で管理するキーを使用して、AWS Glue Data Catalog に格納されているメタデータを暗号化することもできます。さらに、AWS KMS キーを使用して、ジョブのブックマークや、クローラーおよび ETL ジョブで生成されたログを暗号化することもできます。

ジョブ、クローラー、開発エンドポイントによって Amazon Simple Storage Service (Amazon S3) と Amazon CloudWatch Logs に書き込まれたデータに加えて、AWS Glue Data Catalog でメタデータオブジェクトを暗号化できます。ジョブ、クローラーそして開発エンドポイントを AWS Glue で作成する場合、セキュリティ設定をアタッチすることで暗号化設定を指定します。セキュリティ設定には、Amazon S3 で管理されるサーバー側の暗号化キー (SSE-S3)、または AWS KMS (SSE-KMS) で保管されるカスタマーマスターキー (CMK) があります。AWS Glue コンソールを使用してセキュリティ設定を作成できます。

アカウント内の Data Catalog 全体の暗号化を有効にすることもできます。そのためには、AWS KMS に格納されている CMK を指定します。

> **Important**
>
> AWS Glue は、対称カスタマーマネージドキーのみをサポートしています。詳細については、「AWS Key Management Service デベロッパーガイド」の「カスタマーマネージドキー (CMK)」を参照してください。

暗号化を有効にすると、データカタログオブジェクトを追加するとき、クローラーやジョブを実行するとき、または開発エンドポイントを開始するときに、SSE-S3 または SSE-KMS キーを使用して、保管中のデータの書き込みが行われます。さらに、信頼された Transport Layer Security (TLS) プロトコルを使用して、Java Database Connectivity (JDBC) データストアのみにアクセスするように AWS Glue を設定できます。

AWS Glue では、以下の場所で暗号化設定をコントロールします。

- Data Catalog の設定。
- 作成したセキュリティ設定。
- AWS Glue ETL (抽出、変換、ロード) ジョブにパラメータとして渡されるサーバー側の暗号化設定 (SSE-S3 または SSE-KMS)。

暗号化を設定する方法の詳細については、「AWS Glue での暗号化のセットアップ」を参照してください。

トピック

- Data Catalog の暗号化
- 接続パスワードの暗号化
- AWS Glue によって書き込まれたデータの暗号化

Data Catalog の暗号化

AWS Glue Data Catalog 暗号化により、機密データのセキュリティが強化されます。AWS Glue は AWS Key Management Service (AWS KMS) と統合し、Data Catalog に保存されているメタデータを暗号化します。AWS Glue コンソールまたは AWS CLI を使用し、Data Catalog でリソースの暗号化設定を有効または無効にできます。

Data Catalog の暗号化を有効にすると、新しく作成するオブジェクトはすべて暗号化されます。暗号化を無効にすると、新しく作成するオブジェクトは暗号化されませんが、既存の暗号化オブジェクトは暗号化された状態が維持されます。

AWS マネージド暗号化キーまたはカスタマーマネージド暗号化キーを使用し、Data Catalog 全体を暗号化できます。キーの種類と状態の詳細については、「AWS Key Management Service デベロッパーガイド」の「AWS Key Management Service の概念」を参照してください。

> **Note**
>
> 暗号化されたデータカタログをクローラーで使用する場合、暗号化設定を維持する必要があります。クローラーが暗号化されたカタログを処理した後に暗号化設定を削除すると、エラーが発生します。暗号化設定を削除する必要がある場合は、既存のクローラを変更するのではなく、新しいクローラを作成してください。

AWS マネージドキー

AWS マネージドキーはお客様のアカウントにある KMS キーであり、AWS KMS と統合された AWS サービスがお客様に代わって作成、管理、使用します。アカウントの AWS マネージドキーの表示、キーポリシーの表示し、AWS CloudTrail ログでその使用の監査を行うことができます。ただし、キーを管理したり、許可を変更したりすることはできません。

保管時の暗号化は、メタデータの暗号化に使用される AWS Glue の AWS マネージドキーを管理するため、AWS KMS と自動的に統合されます。メタデータの暗号化を有効にしたときに AWS マネージドキーが存在しない場合、AWS KMS は自動的に新しいキーを作成します。

詳細については、「AWS マネージドキー」を参照してください。

カスタマーマネージドキー

カスタマーマネージドキーは AWS アカウント内の KMS キーで、ユーザーが作成、所有、および管理します。ユーザーは、この KMS キーに関する完全なコントロール権を持ちます。次のようにできます：

- キーポリシー、IAM ポリシー、許可付与の確立と維持
- 有効化と無効化
- 暗号化マテリアルのローテーション

- タグを追加する
- 参照するエイリアスの作成
- 削除のスケジュール設定

カスタマーマネージドキーの許可を管理する方法の詳細については、「カスタマーマネージドキー」を参照してください。

> **Important**
>
> AWS Glue は、対称カスタマーマネージドキーのみをサポートしています。KMS キー リストには、対称キーのみが表示されます。ただし、[KMS キー ARN の選択] を選択した場合、任意のキータイプの ARN をコンソールで入力します。対称キーには ARN だけを入力するようにしてください。
> 対称カスタマーマネージドキーを作成するには、「AWS Key Management Service デベロッパーガイド」の「対称カスタマーマネージドキーの作成」の手順に従ってください。

保存時に Data Catalog の暗号化を有効にすると、KMS キーを使用して次のリソースタイプが暗号化されます。

- データベース
- テーブル
- パーティション
- テーブルのバージョン
- 列統計
- ユーザー定義関数
- Data Catalog のビュー

AWS Glue の暗号化コンテキスト

暗号化コンテキストとは、データに関する追加のコンテキスト情報を含むために、使用する (オプションの) キーと値のペアのセットです。AWS KMS は、暗号化コンテキストを追加の認証済みデータとして使用し、暗号化の認証をサポートします。データの暗号化リクエストに暗号化コンテキストを組み込むと、AWS KMS は暗号化コンテキストを暗号化後のデータにバインドします。データを復号化するには、リクエストに同じ暗号化コンテキストを含めます。AWS Glue はすべての AWS

KMS の暗号化処理に同じ暗号化コンテキストを使用する際、キーは glue_catalog_id で値は catalogId を指定します。

```
"encryptionContext": {
    "glue_catalog_id": "111122223333"
}
```

AWS マネージドキーまたは対称カスタマーマネージドキーを使用して Data Catalog を暗号化するとき、監査レコードとログで暗号化コンテキストも使用することができ、キーがどのように使用されているか特定することができます。暗号化コンテキストは、AWS CloudTrail または Amazon CloudWatch ログが生成したログにも表示されます。

暗号化の有効化

AWS Glue コンソールの [データカタログの設定] または AWS CLI を使用し、AWS Glue Data Catalog オブジェクトの暗号化を有効にできます。

Console

コンソールを使用して暗号化を有効にするには

1. AWS Management Consoleにサインインし、AWS Glue コンソール (https://console.aws.amazon.com/glue/) を開きます。
2. ナビゲーションペインで、[データカタログ] を選択します。
3. [データカタログの設定] ページで、[メタデータの暗号化] チェックボックスを選択し、AWS KMS キーを選択します。

   暗号化を有効にするとき、カスタマーマネージドキーを指定しない場合、暗号化設定では AWS マネージド KMS キーが使用されます。
4. (オプション) カスタマーマネージドキーを使用して Data Catalog を暗号化すると、リソースを暗号化および復号化するため、Data Catalog は IAM ロールを登録するオプションを提供します。お客様に代わって AWS Glue が引き受ける IAM ロールの許可を付与する必要があります。データを暗号化および復号する AWS KMS アクセス許可が含まれます。

   Data Catalog に新しいリソースを作成すると、AWS Glue はデータを暗号化するために提供される IAM ロールを引き受けます。同様に、コンシューマがリソースにアクセスすると、AWS Glue はデータを復号化する IAM ロールを引き受けます。必要な許可を持つ IAM ロールを登録した場合、呼び出し元のプリンシパルはキーにアクセスしてデータを復号化する許可の必要がなくなります。

> **Important**
>
> カスタマーマネージドキーを使用して Data Catalog リソースを暗号化するときに限り、KMS オペレーションを IAM ロールに委任できます。KMS ロール委任の機能は、現時点では Data Catalog リソースの暗号化に AWS マネージドキーの使用をサポートしていません。

> **Warning**
>
> IAM ロールを有効にして KMS の操作を委任すると、以前に AWS マネージドキーで暗号化された Data Catalog リソースにアクセスできなくなります。

a. お客様に代わって AWS Glue が引き受けられる IAM ロールを有効にし、データの暗号化および復号化できるようにするには、[KMS 操作を IAM ロールに委任する] オプションを選択します。

b. 次に、IAM ロールを選択します。

   IAM ロールを作成するには、「AWS Glue の IAM ロールを作成する」を参照してください。

   Data Catalog にアクセスするために AWS Glue が引き受ける IAM ロールには、Data Catalog のメタデータを暗号化および復号化する許可が必要です。IAM ロールを作成し、次のインラインポリシーをアタッチできます。

   - 次のポリシーを追加し、Data Catalog を暗号化および復号化する AWS KMS 許可を含めてください。

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

```
            ],
            "Resource": "arn:aws:kms:<region>:<account-id>:key/<key-id>"
        }
    ]
}
```

- 次に、AWS Glue サービスが IAM ロールを引き受けるには、ロールに次の信頼ポリシーを追加します。

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

- 次に、IAM ロールに `iam:PassRole` 許可を追加します。

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

暗号化を有効にするとき、AWS Glue が引き受ける IAM ロールを指定していない場合、Data Catalog にアクセスするプリンシパルには、次の API 操作を実行する許可が必要です。

- `kms:Decrypt`
- `kms:Encrypt`
- `kms:GenerateDataKey`

AWS CLI

SDK または AWS CLI を使用して暗号化を有効にするには

- `PutDataCatalogEncryptionSettings` API オペレーションを使用します。キーが指定されていない場合、AWS Glue はカスタマーアカウントに AWS マネージド暗号化キーを使用し、Data Catalog を暗号化します。

```
aws glue put-data-catalog-encryption-settings \
  --data-catalog-encryption-settings '{
      "EncryptionAtRest": {
      "CatalogEncryptionMode": "SSE-KMS-WITH-SERVICE-ROLE",
      "SseAwsKmsKeyId": "arn:aws:kms:<region>:<account-id>:key/<key-id>",
      "CatalogEncryptionServiceRole":"arn:aws:iam::<account-
id>:role/<encryption-role-name>"
    }

  }'
```

暗号化を有効にすると、Data Catalog オブジェクトで作成するすべてのオブジェクトが暗号化されます。この設定をオフにすると、Data Catalog で作成するオブジェクトが暗号化されなくなります。必要な KMS 許可を使用して、Data Catalog の既存の暗号化オブジェクトに引き続きアクセスできます。

> **Important**
>
> AWS KMS キーは、Data Catalog でそのキーを使用して暗号化されたすべてのオブジェクトの AWS KMS キーストアに、使用可能な状態で保持する必要があります。キーを削除

すると、オブジェクトは復号化できなくなります。ある種のシナリオでは、このようにして、Data Catalog メタデータにアクセスできないようにする場合があります。

AWS Glue 用に KMS キーの監視

Data Catalog リソースに KMS キーを使用すると、AWS CloudTrail または Amazon CloudWatch ログを使用して AWS Glue が AWS KMS に送信するリクエストを追跡できます。AWS CloudTrail は、KMS キーで暗号化されたデータにアクセスするために AWS Glue が呼び出す KMS 操作を監視および記録します。

次の例は、`Decrypt` および `GenerateDataKey` オペレーションの AWS CloudTrail イベントです。

Decrypt

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROAXPHTESTANDEXAMPLE:Sampleuser01",
        "arn": "arn:aws:sts::111122223333:assumed-role/Admin/Sampleuser01",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROAXPHTESTANDEXAMPLE",
                "arn": "arn:aws:iam::111122223333:role/Admin",
                "accountId": "111122223333",
                "userName": "Admin"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2024-01-10T14:33:56Z",
                "mfaAuthenticated": "false"
            }
        },
        "invokedBy": "glue.amazonaws.com"
    },
    "eventTime": "2024-01-10T15:18:11Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "Decrypt",
```

```
    "awsRegion": "eu-west-2",
    "sourceIPAddress": "glue.amazonaws.com",
    "userAgent": "glue.amazonaws.com",
    "requestParameters": {
        "encryptionContext": {
            "glue_catalog_id": "111122223333"
        },
        "encryptionAlgorithm": "SYMMETRIC_DEFAULT"
    },
    "responseElements": null,
    "requestID": "43b019aa-34b8-4798-9b98-ee968b2d63df",
    "eventID": "d7614763-d3fe-4f84-a1e1-3ca4d2a5bbd5",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:<region>:111122223333:key/<key-id>"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management",
    "sessionCredentialFromConsole": "true"
}
```

GenerateDataKey

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId":
 "AROAXPHTESTANDEXAMPLE:V_00_GLUE_KMS_GENERATE_DATA_KEY_111122223333",
        "arn": "arn:aws:sts::111122223333:assumed-role/Admin/
V_00_GLUE_KMS_GENERATE_DATA_KEY_111122223333",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
```

```
                "principalId": "AROAXPHTESTANDEXAMPLE",
                "arn": "arn:aws:iam::111122223333:role/Admin",
                "accountId": "AKIAIOSFODNN7EXAMPLE",
                "userName": "Admin"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2024-01-05T21:15:47Z",
                "mfaAuthenticated": "false"
            }
        },
        "invokedBy": "glue.amazonaws.com"
    },
    "eventTime": "2024-01-05T21:15:47Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "GenerateDataKey",
    "awsRegion": "eu-west-2",
    "sourceIPAddress": "glue.amazonaws.com",
    "userAgent": "glue.amazonaws.com",
    "requestParameters": {
        "keyId": "arn:aws:kms:eu-west-2:AKIAIOSFODNN7EXAMPLE:key/
AKIAIOSFODNN7EXAMPLE",
        "encryptionContext": {
            "glue_catalog_id": "111122223333"
        },
        "keySpec": "AES_256"
    },
    "responseElements": null,
    "requestID": "64d1783a-4b62-44ba-b0ab-388b50188070",
    "eventID": "1c73689b-2ef2-443b-aed7-8c126585ca5e",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:eu-west-2:111122223333:key/AKIAIOSFODNN7EXAMPLE"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```

接続パスワードの暗号化

AWS Glue Data Catalog の接続パスワードを取得するには、`GetConnection` および `GetConnections` の API オペレーションを使用します。これらのパスワードは Data Catalog 接続に保存され、AWS Glue から Java Database Connectivity (JDBC) データストアに接続される際に使用されます。接続が作成または更新された際、Data Catalog 設定のオプションによって、パスワードが暗号化されたかどうかが判断され、暗号化されている場合には、指定された AWS Key Management Service (AWS KMS) キーが判断されます。

AWS Glue コンソールの [Data catalog settings] (データカタログ設定) ページで、このオプションを有効にできます。

接続パスワードを暗号化するには

1. AWS Management Consoleにサインインし、AWS Glue コンソール (https://console.aws.amazon.com/glue/) を開きます。
2. ナビゲーションペインで [設定] を選択します。
3. [Data catalog settings] (データカタログ設定) ページで、[Encrypt connection passwords] (接続パスワードの暗号化) を選択し、AWS KMS を選択します。

> **Important**
>
> AWS Glue は、対称カスタマーマスターキー (CMK) のみをサポートしています。[AWS KMS key] (KMS キー) リストには、対称キーのみが表示されます。ただし、[Choose a AWS KMS key ARN] (KMS キー ARN を選択します) を選択した場合、任意のキータイプの ARN をコンソールで入力します。対称キーには ARN だけを入力するようにしてください。

詳細については、「データカタログの設定 」を参照してください。

AWS Glue によって書き込まれたデータの暗号化

セキュリティ設定は、AWS Glue が使用できるセキュリティプロパティのセットです。セキュリティ構成を使用して、保管時にデータを暗号化できます。以下のシナリオでは、セキュリティ設定を使用できるいくつかの方法を示します。

- 暗号化された Amazon CloudWatch Logs を書き込む AWS Glue クローラーにセキュリティ設定をアタッチします。クローラーにセキュリティ設定のアタッチの詳細については、「the section called “セキュリティ設定の構成”」を参照してください。
- 抽出、変換、ロード (ETL) ジョブにセキュリティ設定をアタッチして、暗号化された Amazon Simple Storage Service (Amazon S3) ターゲット、および暗号化された CloudWatch Logs を書き込みます。
- ETL ジョブにセキュリティ設定をアタッチして、暗号化された Amazon S3 データとしてジョブブックマークを書き込みます。
- 開発エンドポイントにセキュリティ設定をアタッチして、暗号化された Amazon S3 ターゲットを書き込みます。

> **Important**
>
> 現時点では、セキュリティ設定は、ETL ジョブのパラメータとして渡されるサーバー側暗号化 (SSE-S3) 設定をすべてオーバーライドします。したがって、ジョブにセキュリティ設定と SSE-S3 の両方が関連付けられている場合、SSE-S3 パラメータは無視されます。

セキュリティ設定の詳細については、「AWS Glue コンソールでのセキュリティ設定の管理」を参照してください。

トピック

- セキュリティ設定を使用するための AWS Glue のセットアップ
- VPC ジョブとクローラーの AWS KMS へのルートを作成する
- AWS Glue コンソールでのセキュリティ設定の管理

セキュリティ設定を使用するための AWS Glue のセットアップ

以下の手順を実行して、セキュリティ設定を使用するように AWS Glue 環境をセットアップします。

1. AWS Glue クローラーに渡される IAM ロールおよび CloudWatch Logs を暗号化するジョブへの AWS KMS アクセス許可を付与するため、AWS Key Management Service (AWS KMS) キーを作成または更新します。詳しくは、Amazon CloudWatch Logs ユーザーガイドの「AWS KMS を使用して CloudWatch Logs のログデータを暗号化する」を参照してください。

   次の例では、*"role1"*、*"role2"*、および *"role3"* が、クローラーおよびジョブに渡される IAM ロールです。

```
{
       "Effect": "Allow",
       "Principal": { "Service": "logs.region.amazonaws.com",
       "AWS": [
                "role1",
                "role2",
                "role3"
             ] },
                    "Action": [
                           "kms:Encrypt*",
                           "kms:Decrypt*",
                           "kms:ReEncrypt*",
                           "kms:GenerateDataKey*",
                           "kms:Describe*"
                    ],
                    "Resource": "*"
}
```

   キーを使用して CloudWatch Logs を暗号化する場合、"Service": "logs.*region*.amazonaws.com" と表示される Service ステートメントが必要です。

2. 使用する前に、AWS KMS キーが ENABLED であることを確認します。

> **Note**
>
> データレイクフレームワークとして Iceberg を使用している場合、Iceberg テーブルにはサーバー側の暗号化を有効にする独自のメカニズムがあります。AWS Glue のセキュリティ設定に加えて、これらの設定を有効にする必要があります。Iceberg テーブルでサーバー側の暗号化を有効にするには、「Iceberg ドキュメント」のガイダンスを確認してください。

VPC ジョブとクローラーの AWS KMS へのルートを作成する

インターネットを介さずに、仮想プライベートクラウド (VPC) のプライベートエンドポイントから AWS KMS に直接接続できます。VPC エンドポイントを使用すると、VPC と AWS KMS の間の通信は完全に AWS ネットワーク内で実施されます。

VPC 内に AWS KMS VPC エンドポイントを作成することができます。この手順を実行しないと、ジョブまたはクローラーがジョブの `kms timeout` またはクローラーの `internal service exception` で失敗する可能性があります。詳細な手順については、AWS Key Management Service デベロッパーガイドの「VPC エンドポイントを介した AWS KMS への接続」を参照してください。

これらの手順に従って、VPC コンソールで次の操作を行う必要があります。

- [Enable Private DNS name] (プライベート DNS 名を有効にする) を選択します。
- Java Database Connectivity (JDBC) にアクセスするジョブまたはクローラーに使用する [Security group] (セキュリティグループ) (自己参照ルールを持つ) を選択します。AWS Glue 接続の詳細については、「データへの接続」を参照してください。

JDBC データストアにアクセスするクローラーまたはジョブにセキュリティ設定を追加する場合、AWS Glue には AWS KMS エンドポイントへのルートが必要です。ネットワークアドレス変換 (NAT) ゲートウェイまたは AWS KMS VPC エンドポイントを使用してルートを指定できます。NAT ゲートウェイを作成するには、Amazon VPC ユーザーガイドの「NAT ゲートウェイ」を参照してください。

AWS Glue コンソールでのセキュリティ設定の管理

> **Warning**
>
> AWS Glue セキュリティ設定は、現在、Ray ジョブではサポートされていません。

AWS Glue の [security configuration] (セキュリティ設定) には、暗号化されたデータを書き込むときに必要なプロパティが含まれています。AWS Glue コンソールでセキュリティ設定を作成し、クローラ、ジョブ、および開発エンドポイントによって使用される暗号化プロパティを指定します。

作成したすべてのセキュリティ設定を表示するには、https://console.aws.amazon.com/glue/ で AWS Glue コンソールを開き、ナビゲーションペインで [Security configurations] (セキュリティ設定) をクリックします。

[セキュリティ設定] には、設定に関する次のプロパティが表示されます。

名前

設定を作成したときに指定した一意の名前です。名前に使用できるのは、文字 (A ~ Z)、数字 (0 ~ 9)、ハイフン (-)、またはアンダースコア (_) で、長さは 255 文字までです。

Amazon S3 暗号化を有効にする

オンの場合、データカタログのメタデータ保存に Amazon Simple Storage Service (Amazon S3) 暗号化モード (SSE-KMS、SSE-S3 など) が有効になります。

Amazon CloudWatch Logs 暗号化を有効にする

オンの場合、Amazon CloudWatch にログを書き込む際に Amazon S3 暗号化モード (SSE-KMS など) が有効になります。

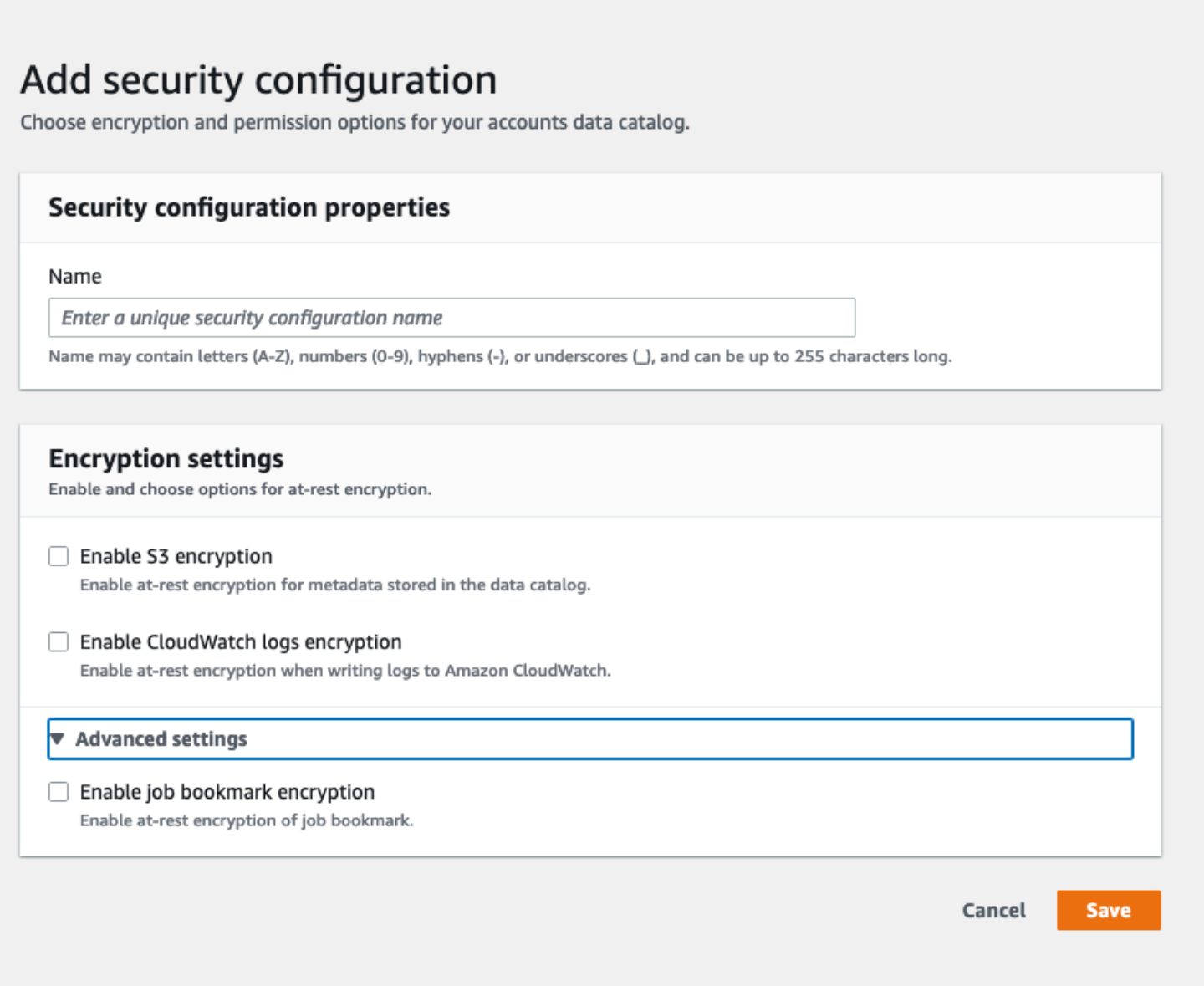
詳細設定: ジョブのブックマーク暗号化を有効にする

オンの場合、ジョブがブックマークされた際に Amazon S3 暗号化モード (CSE-KMS など) が有効になります。

コンソールの [セキュリティ設定] セクションで設定を追加または削除できます。設定の詳細を表示するには、リスト内の設定名を選択します。詳細には、設定の作成時に定義した情報が含まれます。

セキュリティ設定の追加

セキュリティ設定を追加するには、AWS Glue コンソールの [セキュリティ設定] ページで [セキュリティ設定の追加] を選択します。



セキュリティ設定プロパティ

一意のセキュリティ設定名を入力します。名前に使用できるのは、文字 (A ~ Z)、数字 (0 ~ 9)、ハイフン (-)、またはアンダースコア (_) で、長さは 255 文字までです。

暗号化設定

Amazon CloudWatch のログと Amazon S3 のデータカタログに保存されているメタデータの保管時の暗号化を有効にできます。AWS Glue コンソールで AWS Key Management Service (AWS KMS) キーを使用したデータとメタデータの暗号化を設定するには、コンソールユーザーにポリシーを追加します。このポリシーでは次の例のように、許可されているリソースを、Amazon S3 データストア

を暗号化するために使用されるキーの Amazon リソースネーム (ARN) として、指定する必要があります。

```
{
"Version": "2012-10-17",
  "Statement": {
  "Effect": "Allow",
  "Action": [
    "kms:GenerateDataKey",
    "kms:Decrypt",
    "kms:Encrypt"],
  "Resource": "arn:aws:kms:region:account-id:key/key-id"}
}
```

> **Important**
>
> セキュリティ設定がクローラまたはジョブにアタッチされている場合、渡される IAM ロールには AWS KMS アクセス許可が必要です。詳細については、AWS Glue によって書き込まれたデータの暗号化 を参照してください。

設定を定義する場合、次のプロパティの値を指定できます。

S3 暗号化を有効にする

Amazon S3 データを書き込む場合は、Amazon S3 管理キー (SSE-S3) によるサーバー側の暗号化、または AWS KMS 管理キー (SSE-KMS) によるサーバー側の暗号化を使用します。このフィールドはオプションです。Amazon S3 へのアクセス許可を有効にするには、AWS KMS キーを選択するか、または [Enter a key ARN] (キーの ARN を入力) を選択してキーの ARN を指定します。`arn:aws:kms:region:account-id:key/key-id` の形式で ARN を入力します。`arn:aws:kms:region:account-id:alias/alias-name` など、キーのエイリアスで ARN を指定することもできます。

ジョブで Spark UI を有効にすると、Amazon S3 にアップロードされた Spark UI ログファイルに同じ暗号化が適用されます。

> **Important**
>
> AWS Glue は、対称カスタマーマスターキー (CMK) のみをサポートしています。[AWS KMS key] (KMS キー) リストには、対称キーのみが表示されます。ただし、[Choose a

AWS KMS key ARN] (KMS キー ARN を選択します) を選択した場合、任意のキータイプの ARN をコンソールで入力します。対称キーには ARN だけを入力するようにしてください。

CloudWatch Logs 暗号化を有効にする

サーバー側 (SSE-KMS) 暗号化が CloudWatch Logs の暗号化に使用されます。このフィールドはオプションです。有効にするには、AWS KMS キーを選択するか、または [Enter a key ARN] (キーの ARN を入力)を選択してキーの ARN を指定します。`arn:aws:kms:`*`region`*`:`*`account-id`*`:key/`*`key-id`* の形式で ARN を入力します。`arn:aws:kms:`*`region`*`:`*`account-id`*`:alias/`*`alias-name`* など、キーのエイリアスで ARN を指定することもできます。

詳細設定: ジョブブックマークの暗号化

クライアント側 (CSE-KMS) 暗号化は、ジョブのブックマークを暗号化するために使用されます。このフィールドはオプションです。ブックマークデータは、Amazon S3 に保存するために送信される前に暗号化されます。有効にするには、AWS KMS キーを選択するか、または [Enter a key ARN] (キーの ARN を入力)を選択してキーの ARN を指定します。`arn:aws:kms:`*`region`*`:`*`account-id`*`:key/`*`key-id`* の形式で ARN を入力します。`arn:aws:kms:`*`region`*`:`*`account-id`*`:alias/`*`alias-name`* など、キーのエイリアスで ARN を指定することもできます。

詳細については、Amazon Simple Storage Service コンソールユーザーガイドの以下のトピックを参照してください。

- SSE-S3 の詳細については、「Amazon S3 で管理された暗号化キーによるサーバー側の暗号化 (SSE-S3) を使用したデータの保護」を参照してください。
- SSE-KMS の詳細については、「AWS KMS keys によるサーバー側の暗号化を使用したデータの保護」を参照してください。
- CSE-KMS の詳細については、「AWS KMS に保存されている KMS キーの使用」を参照してください。

Encrypting data in transit

AWS は転送中のデータに対して Transport Layer Security (TLS) 暗号化を提供します。AWS Glue の [security configurations] (セキュリティ設定) を使用して、クローラー、ETL ジョブ、開発エンドポイ

ントの暗号化を設定できます。Data Catalog の設定を介して、AWS Glue Data Catalog 暗号化を有効にできます。

2018 年 9 月 4 日現在、 (AWS KMSbring your own keyとserver-side encryption) が AWS Glue ETLおよび AWS Glue Data Catalog がサポートされています。

FIPS 準拠

コマンドラインインターフェイスまたは API を使用して AWS にアクセスするときに FIPS 140-2 検証済みの暗号化モジュールが必要な場合は、FIPS エンドポイントを使用します。利用可能な FIPS エンドポイントの詳細については、「連邦情報処理規格 (FIPS) 140-2」を参照してください。

キーの管理

AWS Glue で AWS Identity and Access Management (IAM) を使用して、ユーザー、AWS リソース、グループ、ロール、およびアクセスや拒否などに関する詳細なポリシーを定義できます。

組織のニーズに応じて、リソースベースのポリシーとアイデンティティベースのポリシーの両方を使用してメタデータへのアクセスを定義できます。リソースベースのポリシーでは、リソースへのアクセスを許可または拒否するプリンシパルを一覧表示し、クロスアカウントアクセスなどのポリシーを設定できます。アイデンティティポリシーは、IAM 内の特定のユーザー、グループ、およびロールにアタッチします。

詳細な例については、AWS ビッグデータブログ「リソースレベルの IAM アクセス許可とリソースベースのポリシーで、AWS Glue Data Catalog へのアクセスを制限する」を参照してください。

ポリシーの詳細なアクセス部分については、`Resource` 句内で定義します。この部分では、アクションを実行する対象の AWS Glue Data Catalog オブジェクトと、このオペレーションで返される結果のオブジェクトの両方を定義します。

開発エンドポイントは、AWS Glue スクリプトの開発およびテストに使用できる環境です。開発エンドポイントの SSH キーを追加、削除、または変更できます。

2018 年 9 月 4 日現在、AWS KMS ETLおよび AWS Glue の (AWS Glue Data Catalogbring your own key と server-side encryption) がサポートされています。

AWS の他のサービスへの AWS Glue の依存関係

ユーザーが AWS Glue コンソールを使用するには、AWS アカウントで AWS Glue リソースの使用を許可する最小限のアクセス許可セットが必要です。これらの AWS Glue アクセス許可に加えて、コンソールでは次のサービスからのアクセス許可が必要になります。

- ログを表示するための Amazon CloudWatch Logs のアクセス許可。
- ロールをリスト化して渡すための AWS Identity and Access Management (IAM) のアクセス許可。
- スタックを操作する AWS CloudFormation のアクセス許可。
- Virtual Private Cloud (VPC)、サブネット、セキュリティグループ、インスタンスなどのオブジェクトをリストする Amazon Elastic Compute Cloud (Amazon EC2) のアクセス許可 (ジョブ、クローラーの実行時や、開発エンドポイントの作成時の VPC などの Amazon EC2 項目をセットアップするため)。
- バケットとオブジェクトをリストし、スクリプトを取得して保存する Amazon Simple Storage Service (Amazon S3) のアクセス許可。
- クラスターでの作業に必要な Amazon Redshift のアクセス許可。
- インスタンスをリスト化するための Amazon Relational Database Service (Amazon RDS) のアクセス許可。

開発エンドポイント

開発エンドポイントは、AWS Glue スクリプトの作成およびテストに使用できる環境です。AWS Glue を使用して、開発エンドポイントを作成、編集、削除できます。作成されたすべての開発エンドポイントを一覧表示できます。開発エンドポイントの SSH キーを追加、削除、または変更できます。開発エンドポイントで使用するノートブックを作成することもできます。

開発環境をプロビジョニングするための設定値を提供します。これらの値は、開発エンドポイントに安全にアクセスし、エンドポイントからデータストアにアクセスできるようにネットワークを設定する方法を AWS Glue に指定します。次に、開発エンドポイントに接続するノートブックを作成できます。ノートブックを使用して ETL スクリプトを作成し、テストします。

AWS Glue の ETL ジョブの実行に使用する IAM ロールと同様のアクセス許可を持つ AWS Identity and Access Management (IAM) ロールを使用します。Virtual Private Cloud (VPC)、サブネット、セキュリティグループを使用して、データリソースに安全に接続できる開発エンドポイントを作成します。SSH を使用して開発環境に接続するための SSH キーペアを生成します。

JDBC を通じてデータセットにアクセスするために使用できる VPC 内で、Amazon S3 のデータ用の開発エンドポイントを作成できます。

ローカルマシンに Jupyter Notebook クライアントをインストールし、これを使用して開発エンドポイントで ETL スクリプトをデバッグおよびテストできます。または Sagemaker ノートブックを使用して、AWS 上の JupyterLab で ETL スクリプトを記述できます。「開発エンドポイントで Amazon SageMaker ノートブックを使用する」を参照してください。

AWS Glue は、aws-glue-dev-endpoint をプレフィックスとする名前を Amazon EC2 インスタンスにタグ付けします。

開発エンドポイントにノートブックサーバーをセットアップして、AWS Glue 拡張機能で PySpark を実行できます。

AWS Glue のアイデンティティとアクセスの管理

AWS Identity and Access Management (IAM) は、管理者が AWS リソースへのアクセスを安全に制御するために役立つ AWS のサービスです。IAM 管理者は、誰を認証 (サインイン) し、誰に AWS Glue リソースの使用を許可する (アクセス許可を持たせる) かを制御します。IAM は、追加費用なしで使用できる AWS のサービスです。

> **Note**
>
> AWS Glue データカタログ内のデータへのアクセスを許可するときは、AWS Glue メソッドか AWS Lake Formation の付与を使用します。AWS Glue メソッドを使用したきめ細かなアクセスコントロールを設定するときは、AWS Identity and Access Management (IAM) ポリシーを使用します。Lake Formation では、リレーショナルデータベースシステムの GRANT/REVOKE コマンドに似た、よりシンプルな GRANT/REVOKE アクセス許可モデルを使用します。
> このセクションでは、AWS Glue メソッドの使用方法を説明します Lake Formation の付与の詳細については、AWS Lake Formation デベロッパーガイドの「Granting Lake Formation Permissions」(Lake Formation 許可の付与) を参照してください。

トピック

- 対象者
- アイデンティティによる認証
- ポリシーを使用したアクセス権の管理
- AWS Glue の仕組み IAM
- AWS Glue の IAM アクセス許可の設定
- AWS Glue のアクセスコントロールポリシーの例
- AWS Glue の AWS マネージドポリシーの付与

- AWS Glue リソース ARN の指定
- クロスアカウントアクセス許可の付与
- AWS Glue アイデンティティとアクセスに関するトラブルシューティング

対象者

AWS Identity and Access Management (IAM) の用途は、AWS Glue で行う作業によって異なります。

サービスユーザー - AWS Glue サービスを使用してジョブを実行するときは、管理者から必要な認証情報とアクセス許可が提供されます。作業を実行するために他の AWS Glue の機能を使用するときは、追加の許可が必要になる場合があります。アクセスの管理方法を理解すると、管理者から適切な権限をリクエストするのに役に立ちます。AWS Glue の機能にアクセスできないときは、「AWS Glue アイデンティティとアクセスに関するトラブルシューティング」を参照してください。

サービス管理者 - 社内の AWS Glue リソースを管理しているユーザーは、通常は AWS Glue に完全にアクセスできます。サービスを利用するユーザーがどの AWS Glue 機能やリソースにアクセスできるかを決めるのは、管理者の仕事です。その後、IAM 管理者にリクエストを送信して、サービスユーザーの権限を変更する必要があります。このページの情報を点検して、IAM の基本概念を理解してください。会社が AWS Glueで IAM を使用する方法の詳細については、「AWS Glue の仕組み IAM」を参照してください。

IAM 管理者 - IAM 管理者は、AWS Glue へのアクセスを管理するポリシーの、作成方法の詳細を確認しておく必要があります。IAM で使用できる AWS Glue アイデンティティベースのポリシーの例については、「AWS Glue のアイデンティティベースポリシーの例」を参照してください。

アイデンティティによる認証

認証とは、アイデンティティ認証情報を使用して AWSにサインインする方法です。ユーザーは、AWS アカウントのルートユーザー、IAM ユーザーとして、または IAM ロールを引き受けることによって、認証される (AWSにサインインする) 必要があります。

ID ソースから提供された認証情報を使用して、フェデレーティッドアイデンティティとして AWS にサインインできます。AWS IAM Identity Centerフェデレーティッドアイデンティティの例としては、(IAM アイデンティティセンター) ユーザー、貴社のシングルサインオン認証、Google または Facebook の認証情報などがあります。フェデレーティッド ID としてサインインする場合、IAM ロールを使用して、前もって管理者により ID フェデレーションが設定されています。フェデレーションを使用して AWSにアクセスする場合、間接的にロールを引き受けることになります。

ユーザーのタイプに応じて、AWS Management Consoleまたは AWS アクセスポータルにサインインできます。AWS へのサインインの詳細については、AWS サインインユーザーガイドの「AWS アカウントにサインインする方法」を参照してください。

プログラムを使用して AWS にアクセスする場合、AWSは Software Development Kit (SDK) とコマンドラインインターフェイス (CLI) を提供し、認証情報を使用してリクエストに暗号で署名します。AWS ツールを使用しない場合は、リクエストに自分で署名する必要があります。リクエストに署名する推奨方法の使用については、「IAM ユーザーガイド」の「API リクエスト向け AWS Signature Version 4」を参照してください。

使用する認証方法を問わず、追加セキュリティ情報の提供をリクエストされる場合もあります。例えば、AWSは、アカウントのセキュリティを強化するために多要素認証 (MFA) を使用することをお勧めします。詳細については、「AWS IAM Identity Center ユーザーガイド 」の「多要素認証」、および「IAM ユーザーガイド」の「IAM での AWS 多要素認証」を参照してください。

AWS アカウント のルートユーザー

AWS アカウント を作成する場合は、このアカウントのすべての AWS のサービス とリソースに対して完全なアクセス権を持つ 1 つのサインインアイデンティティから始めます。このアイデンティティは AWS アカウント ルートユーザーと呼ばれ、アカウントの作成に使用した E メールアドレスとパスワードでサインインすることによってアクセスできます。日常的なタスクには、ルートユーザーを使用しないことを強くお勧めします。ルートユーザーの認証情報は保護し、ルートユーザーでしか実行できないタスクを実行するときに使用します。ルートユーザーとしてサインインする必要があるタスクの完全なリストについては、IAM ユーザーガイドのルートユーザー認証情報が必要なタスクを参照してください。

フェデレーティッドアイデンティティ

ベストプラクティスとして、管理者アクセスを必要とするユーザーを含む人間のユーザーに対し、ID プロバイダーとのフェデレーションを使用して、一時的な認証情報の使用により、AWS のサービスにアクセスすることを要求します。

フェデレーティッド ID は、エンタープライズユーザーディレクトリ、ウェブ ID プロバイダ、AWS Directory Service、Identity Center ディレクトリのユーザーか、または ID ソースから提供された認証情報を使用して AWS のサービス にアクセスするユーザーです。フェデレーティッド ID が AWS アカウント にアクセスすると、ロールが継承され、ロールは一時的な認証情報を提供します。

アクセスを一元管理する場合は、AWS IAM Identity Centerを使用することをお勧めします。IAM アイデンティティセンター でユーザーとグループを作成するか、すべての AWS アカウント とアプリ

ケーションで使用するために、独自の ID ソースで一連のユーザーとグループに接続して同期することもできます。IAM Identity Center の詳細については、「AWS IAM Identity Centerユーザーガイド」の「What is IAM Identity Center?」(IAM Identity Center とは) を参照してください。

IAM ユーザーとグループ

IAM ユーザーは、1 人のユーザーまたは 1 つのアプリケーションに対して特定の許可を持つ AWS アカウント 内のアイデンティティです。可能であれば、パスワードやアクセスキーなどの長期的な認証情報を保有する IAM ユーザーを作成する代わりに、一時的な認証情報を使用することをお勧めします。ただし、IAM ユーザーでの長期的な認証情報が必要な特定のユースケースがある場合は、アクセスキーをローテーションすることをお勧めします。詳細については、「IAM ユーザーガイド」の「Rotate access keys regularly for use cases that require long-term credentials」を参照してください。

IAM グループは、IAM ユーザーの集団を指定するアイデンティティです。グループとしてサインインすることはできません。グループを使用して、複数のユーザーに対して一度に権限を指定できます。多数のユーザーグループがある場合、グループを使用することで権限の管理が容易になります。例えば、IAMAdmins という名前のグループを設定して、そのグループに IAM リソースを管理する許可を与えることができます。

ユーザーは、ロールとは異なります。ユーザーは 1 人の人または 1 つのアプリケーションに一意に関連付けられますが、ロールはそれを必要とする任意の人が引き受けるようになっています。ユーザーには永続的な長期の認証情報がありますが、ロールでは一時認証情報が提供されます。詳細については、「IAM ユーザーガイド」の「IAM ユーザー向けユースケース」を参照してください。

IAM ロール

IAM ロールは、特定の許可を持つ、AWS アカウント内のアイデンティティです。これは IAM ユーザーに似ていますが、特定のユーザーには関連付けられていません。AWS Management Console で IAM ロールを一時的に引き受けたい場合、ユーザーから IAM ロール に切り替える(コンソール操作)ことができます。ロールを引き受けるには、AWS CLI または AWS API オペレーションを呼び出すか、カスタム URL を使用します。ロールを使用する方法の詳細については、「IAM ユーザーガイド」の「ロールを引き受けるための各種方法」を参照してください。

IAM ロールと一時的な認証情報は、次の状況で役立ちます:

- フェデレーションユーザーアクセス – フェデレーティッド ID に許可を割り当てるには、ロールを作成してそのロールの許可を定義します。フェデレーティッド ID が認証されると、その ID はロールに関連付けられ、ロールで定義されている許可が付与されます。フェデレーションのロール

の詳細については、「IAM ユーザーガイド」の「サードパーティー ID プロバイダー (フェデレーション) 用のロールを作成する」を参照してください。IAM Identity Center を使用する場合は、許可セットを設定します。アイデンティティが認証後にアクセスできるものを制御するため、IAM Identity Center は、権限セットを IAM のロールに関連付けます。アクセス許可セットの詳細については、「AWS IAM Identity Center User Guide」の「Permission sets」を参照してください。

- 一時的な IAM ユーザー権限 - IAM ユーザーまたはロールは、特定のタスクに対して複数の異なる権限を一時的に IAM ロールで引き受けることができます。
- クロスアカウントアクセス - IAM ロールを使用して、自分のアカウントのリソースにアクセスすることを、別のアカウントの人物 (信頼済みプリンシパル) に許可できます。クロスアカウントアクセス権を付与する主な方法は、ロールを使用することです。ただし、一部の AWS のサービス では、(ロールをプロキシとして使用する代わりに) リソースにポリシーを直接アタッチできます。クロスアカウントアクセスにおけるロールとリソースベースのポリシーの違いについては、「IAM ユーザーガイド」の「IAM でのクロスアカウントのリソースへのアクセス」を参照してください。
- クロスサービスアクセス権 - 一部の AWS のサービス では、他の AWS のサービス の機能を使用します。例えば、あるサービスで呼び出しを行うと、通常そのサービスによって Amazon EC2 でアプリケーションが実行されたり、Amazon S3 にオブジェクトが保存されたりします。サービスでは、呼び出し元プリンシパルの許可、サービスロール、またはサービスリンクロールを使用してこれを行う場合があります。
  - 転送アクセスセッション (FAS) – IAM ユーザーまたはロールを使用して AWS でアクションを実行するユーザーは、プリンシパルと見なされます。一部のサービスを使用する際に、アクションを実行することで、別のサービスの別のアクションがトリガーされることがあります。FAS は、AWS のサービスを呼び出すプリンシパルの権限を、AWS のサービスのリクエストと合わせて使用し、ダウンストリームのサービスに対してリクエストを行います。FAS リクエストは、サービスが、完了するために他の AWS のサービス またはリソースとのやりとりを必要とするリクエストを受け取ったときにのみ行われます。この場合、両方のアクションを実行するためのアクセス許可が必要です。FAS リクエストを行う際のポリシーの詳細については、「転送アクセスセッション」を参照してください。
  - サービスロール - サービスがユーザーに代わってアクションを実行するために引き受ける IAM ロールです。IAM 管理者は、IAM 内からサービスロールを作成、変更、削除することができます。詳細については、「IAM ユーザーガイド」の「AWS のサービス にアクセス許可を委任するロールの作成」を参照してください。
  - サービスにリンクされたロール - サービスにリンクされたロールは、AWS のサービス にリンクされたサービスロールの一種です。サービスがロールを引き受け、ユーザーに代わってアクションを実行できるようになります。サービスにリンクされたロールは、AWS アカウント に表示さ

れ、サービスによって所有されます。IAM 管理者は、サービスリンクロールの許可を表示できますが、編集することはできません。

- Amazon EC2 で実行されているアプリケーション - EC2 インスタンスで実行され、AWS CLI または AWS API 要求を行っているアプリケーションの一時的な認証情報を管理するには、IAM ロールを使用できます。これは、EC2 インスタンス内でのアクセスキーの保存に推奨されます。AWS ロールを EC2 インスタンスに割り当て、そのすべてのアプリケーションで使用できるようにするには、インスタンスに添付されたインスタンスプロファイルを作成します。インスタンスプロファイルにはロールが含まれ、EC2 インスタンスで実行されるプログラムは一時的な認証情報を取得できます。詳細については、「IAM ユーザーガイド」の「Amazon EC2 インスタンスで実行されるアプリケーションに IAM ロールを使用してアクセス許可を付与する」を参照してください。

ポリシーを使用したアクセス権の管理

AWS でアクセスを制御するには、ポリシーを作成して AWS ID またはリソースにアタッチします。ポリシーは AWS のオブジェクトであり、アイデンティティやリソースに関連付けて、これらのアクセス許可を定義します。AWS は、プリンシパル (ユーザー、ルートユーザー、またはロールセッション) がリクエストを行うと、これらのポリシーを評価します。ポリシーでの権限により、リクエストが許可されるか拒否されるかが決まります。大半のポリシーは JSON ドキュメントとして AWS に保存されます。JSON ポリシードキュメントの構造と内容の詳細については、「IAM ユーザーガイド」の「JSON ポリシー概要」を参照してください。

管理者は AWSJSON ポリシーを使用して、だれが何にアクセスできるかを指定できます。つまり、どのプリンシパルがどんなリソースにどんな条件でアクションを実行できるかということです。

デフォルトでは、ユーザーやロールに権限はありません。IAM 管理者は、リソースで必要なアクションを実行するための権限をユーザーに付与する IAM ポリシーを作成できます。その後、管理者はロールに IAM ポリシーを追加し、ユーザーはロールを引き継ぐことができます。

IAM ポリシーは、オペレーションの実行方法を問わず、アクションの許可を定義します。例えば、`iam:GetRole` アクションを許可するポリシーがあるとします。このポリシーがあるユーザーは、AWS Management Console、AWS CLI、または AWS API からロール情報を取得できます。

アイデンティティベースポリシー

アイデンティティベースポリシーは、IAM ユーザーグループ、ユーザーのグループ、ロールなど、アイデンティティにアタッチできる JSON 許可ポリシードキュメントです。これらのポリシーは、ユーザーとロールが実行できるアクション、リソース、および条件をコントロールします。ID ベー

スのポリシーを作成する方法については、「IAM ユーザーガイド」の「カスタマーマネージドポリシーによるカスタム IAM アクセス許可の定義」を参照してください。

アイデンティティベースのポリシーは、さらにインラインポリシーまたはマネージドポリシーに分類できます。インラインポリシーは、単一のユーザー、グループ、またはロールに直接埋め込まれます。マネージドポリシーは、AWS アカウント内の複数のユーザー、グループ、およびロールにアタッチできるスタンドアロンポリシーです。マネージドポリシーには、AWS マネージドポリシーとカスタマー管理ポリシーがあります。マネージドポリシーまたはインラインポリシーの使い分けについては、「IAM ユーザーガイド」の「マネージドポリシーとインラインポリシーの比較」を参照してください。

リソースベースのポリシー

リソースベースのポリシーは、リソースに添付する JSON ポリシードキュメントです。リソースベースのポリシーには例として、IAM ロールの信頼ポリシー や Amazon S3 バケットポリシー があげられます。リソースベースのポリシーをサポートするサービスでは、サービス管理者はポリシーを使用して特定のリソースへのアクセスを制御できます。ポリシーがアタッチされているリソースの場合、指定されたプリンシパルがそのリソースに対して実行できるアクションと条件は、ポリシーによって定義されます。リソースベースのポリシーでは、プリンシパルを指定する必要があります。プリンシパルには、アカウント、ユーザー、ロール、フェデレーションユーザー、または AWS のサービス を含めることができます。

リソースベースのポリシーは、そのサービス内にあるインラインポリシーです。リソースベースのポリシーでは IAM の AWSマネージドポリシーは使用できません。

アクセスコントロールリスト (ACL)

アクセスコントロールリスト (ACL) は、どのプリンシパル (アカウントメンバー、ユーザー、またはロール) がリソースにアクセスするための許可を持つかを制御します。ACL はリソースベースのポリシーに似ていますが、JSON ポリシードキュメント形式は使用しません。

Amazon S3、AWS WAF、および Amazon VPC は、ACL をサポートするサービスの例です。ACL の詳細については、「Amazon Simple Storage Service デベロッパーガイド」の「アクセスコントロールリスト (ACL) の概要」を参照してください。

その他のポリシータイプ

AWS では、他の一般的ではないポリシータイプをサポートしています。これらのポリシータイプでは、より一般的なポリシータイプで付与された最大の権限を設定できます。

- アクセス許可の境界 - アクセス許可の境界は、アイデンティティベースポリシーによって IAM エンティティ (IAM ユーザーまたはロール) に付与できる権限の上限を設定する高度な機能です。エンティティにアクセス許可の境界を設定できます。結果として得られる権限は、エンティティのアイデンティティベースポリシーとそのアクセス許可の境界の共通部分になります。`Principal` フィールドでユーザーまたはロールを指定するリソースベースのポリシーでは、アクセス許可の境界は制限されません。これらのポリシーのいずれかを明示的に拒否した場合、権限は無効になります。アクセス許可の境界の詳細については、「IAM ユーザーガイド」の「IAM エンティティのアクセス許可境界」を参照してください。
- サービスコントロールポリシー (SCP) - SCP は、AWS Organizations で組織や組織単位 (OU) の最大許可を指定する JSON ポリシーです。AWS Organizations は、顧客のビジネスが所有する複数の AWS アカウント をグループ化し、一元的に管理するサービスです。組織内のすべての機能を有効にすると、サービスコントロールポリシー (SCP) を一部またはすべてのアカウントに適用できます。SCP はメンバーアカウントのエンティティに対する権限を制限します (各 AWS アカウントのルートユーザー など)。Organizations と SCP の詳細については、「AWS Organizations ユーザーガイド」の「サービスコントロールポリシー (SCP)」を参照してください。
- リソースコントロールポリシー (RCP) – RCP は、所有する各リソースにアタッチされた IAM ポリシーを更新することなく、アカウント内のリソースで利用可能な最大のアクセス許可を設定するために使用できる JSON ポリシーです。RCP は、メンバーアカウントのリソースの許可を制限し、組織に属するかどうかにかかわらず、AWS アカウントのルートユーザー を含む ID のための有効な許可に影響を及ぼす可能性があります。Organizations と RCP の詳細 (RCP をサポートする AWS のサービスのリストを含む) については、「AWS Organizations ユーザーガイド」の「リソースコントロールポリシー (RCP)」を参照してください。
- セッションポリシー - セッションポリシーは、ロールまたはフェデレーションユーザーの一時的なセッションをプログラムで作成する際にパラメータとして渡す高度なポリシーです。結果としてセッションの権限は、ユーザーまたはロールのアイデンティティベースポリシーとセッションポリシーの共通部分になります。また、リソースベースのポリシーから権限が派生する場合もあります。これらのポリシーのいずれかを明示的に拒否した場合、権限は無効になります。詳細については、「IAM ユーザーガイド」の「セッションポリシー」を参照してください。

複数のポリシータイプ

1 つのリクエストに複数のタイプのポリシーが適用されると、結果として作成される権限を理解するのがさらに難しくなります。複数のポリシータイプが関連するとき、リクエストを許可するかどうかを AWS が決定する方法の詳細については、IAM ユーザーガイドのポリシーの評価ロジックを参照してください。

AWS Glue の仕組み IAM

IAM を使用して AWS Glue へのアクセスを管理する前に、 AWS Glue で使用できるIAM機能について説明します。

IAM AWS Glue で使用できる機能

IAM 機能	AWS Glue サポート
アイデンティティベースポリシー	あり
リソースベースのポリシー	部分的
ポリシーアクション	あり
ポリシーリソース	はい
ポリシー条件キー (サービス固有)	はい
ACLs	なし
ABAC (ポリシーのタグ)	部分的
一時的な認証情報	あり
プリンシパル権限	いいえ
サービスロール	あり
サービスリンクロール	なし

AWS Glue やその他の AWS サービスがほとんどのIAM機能でどのように機能するかの概要については、 IAM ユーザーガイドのAWS 「 で と連携する のサービスIAM」を参照してください。

AWS Glue のアイデンティティベースのポリシー

アイデンティティベースのポリシーのサポート: あり

ID ベースのポリシーは、IAMユーザー、ユーザーのグループ、ロールなどの ID にアタッチできるJSONアクセス許可ポリシードキュメントです。これらのポリシーは、ユーザーとロールが実行でき

るアクション、リソース、および条件をコントロールします。ID ベースのポリシーを作成する方法については、IAM「 ユーザーガイド」の「カスタマーマネージドポリシーを使用したカスタムIAMアクセス許可の定義」を参照してください。

IAM ID ベースのポリシーでは、許可または拒否されたアクションとリソース、およびアクションが許可または拒否される条件を指定できます。プリンシパルは、それが添付されているユーザーまたはロールに適用されるため、アイデンティティベースのポリシーでは指定できません。JSON ポリシーで使用できるすべての要素については、 IAM ユーザーガイドのIAMJSON「ポリシー要素リファレンス」を参照してください。

AWS Glue は、すべての の ID ベースのポリシー (IAM ポリシー) をサポートします。AWS Glue オペレーション。ポリシーをアタッチすることで、AWS Glue のテーブルなどの リソース AWS Glue Data Catalog.

AWS Glue のアイデンティティベースのポリシーの例

AWS Glue アイデンティティベースのポリシーの例を表示するには、「」を参照してくださいAWS Glue のアイデンティティベースポリシーの例。

AWS Glue 内のリソースベースのポリシー

リソースベースのポリシーのサポート: 一部

リソースベースのポリシーは、リソースにアタッチするJSONポリシードキュメントです。リソースベースのポリシーの例としては、IAMロール信頼ポリシーと Amazon S3 バケットポリシー があります。リソースベースのポリシーをサポートするサービスでは、サービス管理者はポリシーを使用して特定のリソースへのアクセスをコントロールできます。ポリシーがアタッチされているリソースの場合、指定されたプリンシパルがそのリソースに対して実行できるアクションと条件は、ポリシーによって定義されます。リソースベースのポリシーでは、プリンシパルを指定する必要があります。プリンシパルには、アカウント、ユーザー、ロール、フェデレーティッドユーザー、 を含めることができます AWS のサービス。

クロスアカウントアクセスを有効にするには、リソースベースのポリシーのプリンシパルとして、別のアカウントのアカウントまたはIAMエンティティ全体を指定できます。リソースベースのポリシーにクロスアカウントのプリンシパルを追加しても、信頼関係は半分しか確立されない点に注意してください。プリンシパルとリソースが異なる にある場合 AWS アカウント、信頼されたアカウントのIAM管理者は、プリンシパルエンティティ (ユーザーまたはロール) にリソースへのアクセス許可も付与する必要があります。IAM 管理者は、アイデンティティベースのポリシーをエンティティにアタッチすることで権限を付与します。ただし、リソースベースのポリシーで、同じアカウントのプリ

ンシパルへのアクセス権が付与されている場合は、アイデンティティベースのポリシーをさらに付与する必要はありません。詳細については、「 ユーザーガイド」の「 のクロスアカウントリソースアクセスIAM」を参照してください。 IAM

> **Note**
>
> を使用できるのは、AWS Glue Data Catalog リソースのアクセス許可を管理するリソースポリシー。他の にアタッチすることはできません AWS Glue ジョブ、トリガー、開発エンドポイント、クローラー、分類子などの リソース。
> リソースポリシーは 1 カタログにつき 1 つのみ許可されます。サイズの上限は 10 KB です。

AWS Glue では、リソースポリシーがカタログ にアタッチされます。カタログ は、前述のあらゆる種類の Data Catalog リソースの仮想コンテナです。各 AWS アカウントは、カタログ ID が AWS アカウント ID と同じ AWS リージョンで 1 つのカタログを所有しています。カタログを削除したり変更したりすることはできません。

リソースポリシーは、呼び出し元プリンシパルがポリシードキュメントの `"Principal"`ブロックに含まれているカタログへのすべてのAPI呼び出しについて評価されます。

AWS Glue リソースベースのポリシーの例を表示するには、「」を参照してくださいAWS Glue のリソースベースのポリシーの例。

AWS Glue のポリシーアクション

ポリシーアクションのサポート: あり

管理者はポリシーを使用して AWS JSON、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースに対してどのような条件下でアクションを実行できるかということです。

JSON ポリシーの `Action`要素は、ポリシー内のアクセスを許可または拒否するために使用できるアクションについて説明します。ポリシーアクションは通常、関連付けられた AWS APIオペレーションと同じ名前を持ちます。一致するAPIオペレーションを持たないアクセス許可のみのアクションなど、いくつかの例外があります。また、ポリシーに複数のアクションが必要なオペレーションもあります。これらの追加アクションは、依存アクションと呼ばれます。

このアクションは、関連付けられたオペレーションを実行するための権限を付与するポリシーで使用されます。

AWS Glue アクションのリストを確認するには、「サービス認可リファレンス」のAWS 「Glue で定義されるアクション」を参照してください。

AWS Glue のポリシーアクションでは、アクションの前に次のプレフィックスを使用します。

```
glue
```

単一のステートメントで複数のアクションを指定するには、アクションをカンマで区切ります。

```
"Action": [
      "glue:action1",
      "glue:action2"
         ]
```

ワイルドカード (*) を使用して複数アクションを指定できます。例えば、Get という単語で始まるすべてのアクションを指定するには、次のアクションを含めます。

```
"Action": "glue:Get*"
```

ポリシーの例を表示するには、「AWS Glue のアクセスコントロールポリシーの例」を参照してください。

AWS Glue のポリシーリソース

ポリシーリソースのサポート: あり

管理者はポリシーを使用して AWS JSON、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースに対してどのような条件下でアクションを実行できるかということです。

Resource JSON ポリシー要素は、アクションが適用されるオブジェクトを指定します。ステートメントには、Resource または NotResource 要素を含める必要があります。ベストプラクティスとして、Amazon リソースネーム (ARN) を使用してリソースを指定します。これは、リソースレベルの許可と呼ばれる特定のリソースタイプをサポートするアクションに対して実行できます。

オペレーションのリスト化など、リソースレベルの権限をサポートしないアクションの場合は、ステートメントがすべてのリソースに適用されることを示すために、ワイルドカード (*) を使用します。

```
"Resource": "*"
```

を使用して AWS Glue リソースへのアクセスを制御する方法の詳細についてはARNs、「」を参照してくださいAWS Glue リソース ARN の指定。

AWS Glue リソースタイプとその のリストを確認するにはARNs、「 サービス認可リファレンス」のAWS 「Glue で定義されるリソース」を参照してください。各リソースARNの を指定するために使用できるアクションについては、AWS 「Glue で定義されるアクション」を参照してください。

AWS Glue のポリシー条件キー

サービス固有のポリシー条件キーのサポート: あり

管理者はポリシーを使用して AWS JSON、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルが、どのリソースに対してどのような条件下でアクションを実行できるかということです。

Condition 要素 (または Condition ブロック) を使用すると、ステートメントが有効な条件を指定できます。Condition 要素はオプションです。イコールや未満などの条件演算子を使用して条件式を作成することで、ポリシーの条件とリクエスト内の値を一致させることができます。

1 つのステートメントに複数の Condition 要素を指定する場合、または 1 つの Condition 要素に複数のキーを指定する場合、 AWS では AND 論理演算子を使用してそれらを評価します。1 つの条件キーに複数の値を指定すると、 は論理ORオペレーションを使用して条件 AWS を評価します。ステートメントの権限が付与される前にすべての条件が満たされる必要があります。

条件を指定する際にプレースホルダー変数も使用できます。例えば、リソースにIAMユーザー名がタグ付けされている場合にのみ、リソースにアクセスするアクセス許可をIAMユーザーに付与できます。詳細については、「 ユーザーガイド」のIAM「ポリシー要素: 変数とタグ」を参照してください。 IAM

AWS は、グローバル条件キーとサービス固有の条件キーをサポートしています。すべての AWS グローバル条件キーを表示するには、 ユーザーガイドのAWS 「グローバル条件コンテキストキー」を参照してください。 IAM

AWS Glue 条件キーのリストを確認するには、「サービス認可リファレンス」のAWS 「Glue の条件キー」を参照してください。条件キーを使用できるアクションとリソースについては、AWS 「Glue で定義されるアクション」を参照してください。

ポリシーの例を表示するには、「条件キーまたはコンテキストキーを使用して設定を制御する」を参照してください。

ACLs AWS Glue で

サポートACLs： なし

アクセスコントロールリスト (ACLs) は、どのプリンシパル (アカウントメンバー、ユーザー、またはロール) がリソースにアクセスするアクセス許可を持っているかを制御します。ACLs はリソースベースのポリシーに似ていますが、JSONポリシードキュメント形式は使用していません。

ABAC AWS Glue を使用

サポート ABAC (ポリシーのタグ): 部分的

属性ベースのアクセスコントロール (ABAC) は、属性に基づいてアクセス許可を定義する認可戦略です。では AWS、これらの属性はタグ と呼ばれます。タグは、IAMエンティティ (ユーザーまたはロール) および多くの AWS リソースにアタッチできます。エンティティとリソースのタグ付けは、の最初のステップですABAC。次に、プリンシパルのタグがアクセスしようとしているリソースのタグと一致する場合に、オペレーションを許可するABACポリシーを設計します。

ABAC は、急速に成長している環境や、ポリシー管理が煩雑になる状況で役立ちます。

タグに基づいてアクセスを管理するには、`aws:ResourceTag/key-name`、`aws:RequestTag/key-name`、または `aws:TagKeys` の条件キーを使用して、ポリシーの条件要素でタグ情報を提供します。

サービスがすべてのリソースタイプに対して 3 つの条件キーすべてをサポートする場合、そのサービスの値はありです。サービスが一部のリソースタイプに対してのみ 3 つの条件キーのすべてをサポートする場合、値は「部分的」になります。

の詳細についてはABAC、「 ユーザーガイド」のABAC「認可によるアクセス許可の定義」を参照してください。 IAM のセットアップ手順を含むチュートリアルを表示するにはABAC、 IAM ユーザーガイドの「属性ベースのアクセスコントロールを使用する (ABAC) 」を参照してください。

> **Important**
>
> 条件コンテキストキーは、 にのみ適用されます。AWS Glue API クローラー、ジョブ、トリガー、開発エンドポイントに対する アクション。影響を受けるAPIオペレーションの詳細については、AWS Glue の条件キーを参照してください。

- AWS Glue データカタログAPIオペレーションは現在、 `aws:referer`および `aws:UserAgent` グローバル条件コンテキストキーをサポートしていません。

リソースのタグに基づいてリソースへのアクセスを制限するためのアイデンティティベースポリシーの例を表示するには、「タグを使用してアクセスを許可する」を参照してください。

AWS Glue での一時的な認証情報の使用

一時的な認証情報のサポート: あり

一部の AWS のサービス は、一時的な認証情報を使用してサインインすると機能しません。一時的な認証情報 AWS のサービス を使用する方法などの詳細については、 IAM ユーザーガイドの AWS のサービス で を使用するIAM方法を参照してください。

ユーザー名とパスワード以外の AWS Management Console 方法で にサインインする場合は、一時的な認証情報を使用します。例えば、会社のシングルサインオン (SSO) リンク AWS を使用して にアクセスすると、そのプロセスによって一時的な認証情報が自動的に作成されます。また、ユーザーとしてコンソールにサインインしてからロールを切り替える場合も、一時的な認証情報が自動的に作成されます。ロールの切り替えの詳細については、IAM「 ユーザーガイド」の「ユーザーからIAMロールへの切り替え (コンソール)」を参照してください。

AWS CLI または を使用して、一時的な認証情報を手動で作成できます AWS API。その後、これらの一時的な認証情報を使用して にアクセスできます AWS。長期的なアクセスキーを使用する代わりに、一時的な認証情報を動的に生成 AWS することをお勧めします。詳細については、「」の「一時的なセキュリティ認証情報IAM」を参照してください。

AWS Glue のクロスサービスプリンシパルアクセス許可

転送アクセスセッションをサポート (FAS): いいえ

ユーザーIAMまたはロールを使用して でアクションを実行する場合 AWS、プリンシパルと見なされます。一部のサービスを使用する際に、アクションを実行することで、別のサービスの別のアクションがトリガーされることがあります。FAS は、 を呼び出すプリンシパルのアクセス許可を AWS のサービス、ダウンストリームサービス AWS のサービス へのリクエストリクエストと組み合わせて使用します。FAS リクエストは、サービスが他の AWS のサービス または リソースとのやり取りを完了する必要があるリクエストを受け取った場合にのみ行われます。この場合、両方のアクションを実行するための権限が必要です。FAS リクエストを行う際のポリシーの詳細については、「アクセスセッションの転送」を参照してください。

AWS Glue のサービスロール

サービスロールのサポート: あり

サービスロールは、ユーザーに代わってアクションを実行するためにサービスが引き受けるIAMロールです。IAM 管理者は、 内からサービスロールを作成、変更、削除できますIAM。詳細については、「 ユーザーガイド」の「アクセス許可を に委任するロールを作成する AWS のサービス」を参照してください。 IAM

> **Warning**
>
> サービスロールのアクセス許可を変更すると、 AWS Glue 機能が壊れる可能性があります。サービスロールは、 AWS Glue がガイダンスを提供する場合にのみ編集します。

AWS Glue のサービスロールの作成方法の詳細については、ステップ 1: AWS Glue サービスの IAM ポリシーを作成する「」および「」を参照してくださいステップ 2: AWS Glue 用の IAM ロールを作成する。

AWS Glue のサービスにリンクされたロール

サービスにリンクされたロールのサポート: なし

サービスにリンクされたロールは、 にリンクされたサービスロールの一種です AWS のサービス。サービスは、ユーザーに代わってアクションを実行するロールを引き受けることができます。サービスにリンクされたロールは に表示され AWS アカウント 、 サービスによって所有されます。IAM 管理者はサービスにリンクされたロールのアクセス許可を表示できますが、編集することはできません。

サービスにリンクされたロールの作成または管理の詳細については、AWS 「 と連携するサービス IAM」を参照してください。表の「サービスリンクロール」列に Yes と記載されたサービスを見つけます。サービスにリンクされたロールに関するドキュメントをサービスで表示するには、[はい] リンクを選択します。

AWS Glue の IAM アクセス許可の設定

AWS Identity and Access Management (IAM) を使用して、リソースにアクセスするために AWS Glue が使用するポリシーとロールを定義します。次の手順では、AWS Glue のアクセス許可を設定するためのさまざまなオプションについて説明します。ビジネスニーズに応じて、リソースへのアクセスを追加または削減できます。

> **Note**
>
> オプションを設定せず、基本的な IAM アクセス許可で AWS Glue の使用を開始する場合は、「AWS Glue 用の IAM アクセス許可のセットアップ」を参照してください。

1. AWS Glue サービスの IAM ポリシーを作成する: AWS Glue リソースへのアクセスを許可するサービスポリシーを作成します。
2. AWS Glue の IAM ロールを作成する: IAM ロールを作成して、AWS Glue が使用する Amazon Simple Storage Service (Amazon S3) リソース用の AWS Glue サービスポリシーおよびポリシーをアタッチします。
3. AWS Glue にアクセスするユーザーまたはグループにポリシーをアタッチする: AWS Glue コンソールにサインインするユーザーまたはグループにポリシーをアタッチします。
4. ノートブックの IAM ポリシーを作成する: 開発エンドポイント上のノートブックサーバーの作成に使用する、ノートブックサーバーポリシーを作成します。
5. ノートブックの IAM ロールを作成する: IAM ロールを作成し、ノートブックサーバーポリシーをアタッチします。
6. Amazon SageMaker AI ノートブックの IAM ポリシーを作成する: 開発エンドポイント上に Amazon SageMaker AI ノートブックを作成する際に使用する IAM ポリシーを作成します。
7. Amazon SageMaker AI ノートブックの IAM ロールを作成する: IAM ロールを作成し、開発エンドポイント上に Amazon SageMaker AI ノートブックを作成する際のアクセス許可を付与するポリシーをアタッチします。

ステップ 1: AWS Glue サービスの IAM ポリシーを作成する

Amazon S3 のオブジェクトにアクセスするなど、別の AWS リソース上のデータにアクセスするオペレーションの場合、AWS Glue には、ユーザーの代わりにリソースにアクセスするためのアクセス許可が必要です。AWS Identity and Access Management (IAM) を使用してアクセス権限を提供できます。

> **Note**
>
> AWS 管理ポリシー **`AWSGlueServiceRole`** を使用する場合は、このステップをスキップできます。

このステップでは、AWSGlueServiceRole に似たポリシーを作成します。AWSGlueServiceRole の最新バージョンは IAM コンソールで取得できます。

AWS Glue の IAM ポリシーを作成するには

このポリシーは、AWS Glue がこのポリシーを使用してロールを引き受ける際に必要となる、アカウント内のリソースを管理する Amazon S3 アクションの一部を許可します。このポリシーで指定されているリソースの一部は、Amazon S3 バケット、Amazon S3 ETL スクリプト、CloudWatch Logs、および Amazon EC2 リソースのために AWS Glue で使用される、デフォルト名を参照しています。簡素化のため AWS Glue のデフォルトでは、aws-glue-* のプレフィックスが付いた Amazon S3 オブジェクトをアカウント内のバケットに書き込みます。

1. AWS Management Console にサインインして、IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. 左のナビゲーションペインの [ポリシー] を選択します。
3. [ポリシーの作成] を選択します。
4. [ポリシーの作成] 画面で、JSON 編集のためのタブに移動します。次の JSON ステートメントを使用してポリシードキュメントを作成して、[ポリシーの確認] を選択します。

> **Note**
>
> Amazon S3 リソースに必要なアクセス許可を追加します。アクセスポリシーのリソースセクションを必要なリソースだけに絞り込みたい場合があるかもしれません。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:*",
                "s3:GetBucketLocation",
                "s3:ListBucket",
                "s3:ListAllMyBuckets",
                "s3:GetBucketAcl",
                "ec2:DescribeVpcEndpoints",
                "ec2:DescribeRouteTables",
                "ec2:CreateNetworkInterface",
```

```
                "ec2:DeleteNetworkInterface",
                "ec2:DescribeNetworkInterfaces",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeSubnets",
                "ec2:DescribeVpcAttribute",
                "iam:ListRolePolicies",
                "iam:GetRole",
                "iam:GetRolePolicy",
                "cloudwatch:PutMetricData"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:CreateBucket",
                "s3:PutBucketPublicAccessBlock"
            ],
            "Resource": [
                "arn:aws:s3:::aws-glue-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:PutObject",
                "s3:DeleteObject"
            ],
            "Resource": [
                "arn:aws:s3:::aws-glue-*/*",
                "arn:aws:s3:::*/*aws-glue-*/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::crawler-public*",
                "arn:aws:s3:::aws-glue-*"
```

```
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:AssociateKmsKey"
            ],
            "Resource": [
                "arn:aws:logs:*:*:log-group:/aws-glue/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateTags",
                "ec2:DeleteTags"
            ],
            "Condition": {
                "ForAllValues:StringEquals": {
                    "aws:TagKeys": [
                        "aws-glue-service-resource"
                    ]
                }
            },
            "Resource": [
                "arn:aws:ec2:*:*:network-interface/*",
                "arn:aws:ec2:*:*:security-group/*",
                "arn:aws:ec2:*:*:instance/*"
            ]
        }
    ]
}
```

次の表は、このポリシーによって付与されたアクセス権限を示しています。

[アクション]	リソース	説明
"glue:*"	"*"	すべての AWS Glue API オペレーションを実行する権限を付与します。
"s3:GetBucketLocation", "s3:ListBucket", "s3:ListAllMyBuckets", "s3:GetBucketAcl",	"*"	クローラ、ジョブ、開発エンドポイント、ノートブックサーバーからの Amazon S3 バケットの一覧表示を許可します。
"ec2:DescribeVpcEndpoints", "ec2:DescribeRouteTables", "ec2:CreateNetworkInterface", "ec2:DeleteNetworkInterface", "ec2:DescribeNetworkInterfaces", "ec2:DescribeSecurityGroups", "ec2:DescribeSubnets", "ec2:DescribeVpcAttribute",	"*"	Virtual Private Cloud (VPC) などの Amazon EC2 ネットワーク項目を、ジョブ、クローラ、開発エンドポイントの実行時に設定することを許可します。
"iam:ListRolePolicies", "iam:GetRole", "iam:GetRolePolicy"	"*"	クローラ、ジョブ、開発エンドポイント、ノートブックサーバーからの IAM ロールの一覧表示を許可します。
"cloudwatch:PutMetricData"	"*"	ジョブの CloudWatch メトリクスの書き込みを許可します。

[アクション]	リソース	説明
"s3:CreateBucket", "s3:PutBucketPublicAccessBlock"	"arn:aws:s3:::aws-glue-*"	ジョブおよびノートブックサーバーからアカウントに Amazon S3 バケットを作成できます。 命名規則: [aws-glue-] という Amazon S3 フォルダを使用します。 AWS Glue による、パブリックアクセスをブロックするバケットの作成を有効化します。
"s3:GetObject", "s3:PutObject", "s3:DeleteObject"	"arn:aws:s3:::aws-glue-*/*", "arn:aws:s3:::*/*aws-glue-*/*"	ETL スクリプトやノートブックサーバーの口ケーションなどのオブジェクトを格納するために、アカウントに対する Amazon S3 オブジェクトの取得、配置、および削除を許可します。 命名規則: 名前に aws-glue- のプレフィックスが付いている Amazon S3 バケットまたはフォルダにアクセス許可を付与します。

[アクション]	リソース	説明
`"s3:GetObject"`	`"arn:aws:s3:::crawler-public*", "arn:aws:s3:::aws-glue-*"`	クローラやジョブに関するサンプル、もしくはチュートリアルで使用されている、Amazon S3 オブジェクトの取得を許可します。 命名規則: Amazon S3 バケット名は 「crawler-public」および「aws-glue-」で始まります。
`"logs:CreateLogGroup", "logs:CreateLogStream", "logs:PutLogEvents"`	`"arn:aws:logs:*:*:log-group:/aws-glue/*"`	ログを CloudWatch Logs に書き込むことを許可します。 命名規則: AWS Glue は名前が [aws-glue] で始まるロググループにログを書き込みます。
`"ec2:CreateTags", "ec2:DeleteTags"`	`"arn:aws:ec2:*:*:network-interface/*", "arn:aws:ec2:*:*:security-group/*", "arn:aws:ec2:*:*:instance/*"`	開発エンドポイント用に作成された Amazon EC2 リソースのタグ付けを許可します。 命名規則: AWS Glue は [aws-glue-service-resource] を使用して、Amazon EC2 ネットワークインターフェイス、セキュリティグループ、およびインスタンスにタグ付けします。

5. [Review Policy (ポリシーの確認)] 画面で、[Policy Name (ポリシー名)] ([GlueServiceRolePolicy] など) を入力します。オプションの説明を入力し、ポリシーが適切であることを確認したら、[Create policy] を選択します。

ステップ 2: AWS Glue 用の IAM ロールを作成する

ユーザーに代わり他のサービスを呼び出す際に AWS Glue が引き受けることができる、IAM ロールのアクセス許可を付与する必要があります。これには、AWS Glue で使用するすべてのソース、ターゲット、スクリプト、および一時ディレクトリでの Amazon S3 へのアクセスが含まれます。クローラ、ジョブ、および開発エンドポイントによって許可が必要です。

AWS Identity and Access Management (IAM) を使用してアクセス権限を提供できます。AWS Glue に渡す IAM ロールにポリシーを追加します。

用に IAM ロールを作成するには AWS Glue

1. AWS Management Console にサインインして、IAM コンソール https://console.aws.amazon.com/iam/ を開きます。
2. 左のナビゲーションペインで、[ロール] を選択してください。
3. [ロールの作成] を選択してください。
4. 信頼されたエンティティの種類として [AWS サービス] を選択します。次に、サービスまたはユースケースに対して [AWS Glue] を検索して選択します。[次へ] を選択します。
5. [許可の追加] のページで、必要な許可を含むポリシーを選択します。例えば、一般的な AWS Glue 許可の AWS マネージドポリシー `AWSGlueServiceRole` や Amazon S3 リソースにアクセスするための AWS マネージドポリシー [AmazonS3FullAccess] などがあります。次いで、[次へ] を選択します。

> **Note**
>
> Amazon S3 のソースとターゲットに対するアクセス許可を、このロールのポリシーの 1 つにより付与してください。独自のポリシーを、特定の Amazon S3 リソースにアクセスするために指定します。データソースには、`s3:ListBucket` および `s3:GetObject` アクセス許可が必要です。データターゲットには、`s3:ListBucket`、`s3:PutObject`、および `s3:DeleteObject` アクセス許可が必要です。リソースの Amazon S3 ポリシーの作成については、「Specifying Resources in a Policy」を参照してください。Amazon S3 ポリシーの例については、「Writing IAM Policies: How to Grant Access to an Amazon S3 Bucket」を参照してください。
> SSE-KMS で暗号化された Amazon S3 のソースとターゲットにアクセスする予定がある場合は、AWS Glue のクローラ、ジョブ、開発エンドポイントに、データを復号化するためのポリシーをアタッチしてください。詳細については、「Protecting Data Using

Server-Side Encryption with AWS KMS-Managed Keys (SSE-KMS)」を参照してください。
以下に例を示します。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "kms:Decrypt"
         ],
         "Resource":[
            "arn:aws:kms:*:account-id-without-hyphens:key/key-id"
         ]
      }
   ]
}
```

6. ロールに名前を付けて説明 (オプション) を追加したら、信頼ポリシーおよび許可を確認します。[Role name] (ロール名) に、AWSGlueServiceRoleDefault などのロール名を入力します。コンソールユーザーからサービスにロールを渡すには、名前に文字列 AWSGlueServiceRole のプレフィックスが付けられたロールを作成します。AWSGlueServiceRole が提供するポリシーでは、IAM サービスロールが AWS Glue で始まることを想定しています。それ以外の場合は、ユーザーにポリシーを追加して、IAM ロールの iam:PassRole アクセス許可をユーザーの命名規則に一致させる必要があります。[ロールの作成] を選択します。

> **Note**
>
> ロールを使用してノートブックを作成すると、そのロールがインタラクティブセッションに渡され、その両方で同じロールを使用できるようになります。このように、iam:PassRole のアクセス許可がロールのポリシーの一部として必要です。
> 次の例を使用して、ロール用の新しいポリシーを作成します。アカウント番号とロール名を自分のものに置き換えます。

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

```
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "arn:aws:iam::090000000210:role/<role_name>"
        }
    ]
}
```

7. ロールにタグを追加します (オプション)。タグは AWS リソースに追加できるキーと値のペアであり、リソースの識別、整理、検索するために役立ちます。続いて、[ロールの作成] を選択します。

ステップ 3: AWS Glue にアクセスするユーザーまたはグループにポリシーをアタッチする

管理者は、AWS Glue コンソールまたは AWS Command Line Interface (AWS CLI) を使用して、すべてのユーザー、グループ、ロールにアクセス許可を割り当てる必要があります。これらのアクセス許可は、AWS Identity and Access Management (IAM) を使用しながらポリシーを介して提供します。このステップでは、ユーザーまたはグループにアクセス許可を割り当てる方法について説明します。

このステップを完了すると、ユーザーまたはグループに次のポリシーがアタッチされます。

- AWS 管理ポリシー [AWSGlueConsoleFullAccess]、またはカスタムポリシー [GlueConsoleAccessPolicy]。
- **AWSGlueConsoleSageMakerNotebookFullAccess**
- **CloudWatchLogsReadOnlyAccess**
- **AWSCloudFormationReadOnlyAccess**
- **AmazonAthenaFullAccess**

インラインポリシーをアタッチしてこれをユーザーまたはグループに埋め込むには

AWS Glue コンソールにアクセスするユーザーまたはグループに、AWS マネージドポリシーまたはインラインポリシーをアタッチすることができます。このポリシーで指定されているリソースの中には、AWS Glue で (Amazon S3 バケット、Amazon S3 ETL スクリプト、CloudWatch Logs、AWS

CloudFormation、および Amazon EC2 リソース用として) 使用されるデフォルトの名前があります。簡素化のため AWS Glue のデフォルトでは、`aws-glue-*` のプレフィックスが付いた Amazon S3 オブジェクトをアカウント内のバケットに書き込みます。

> **Note**
>
> AWS 管理ポリシー **`AWSGlueConsoleFullAccess`** を使用する場合は、このステップをスキップできます。

> **Important**
>
> AWS Glue には、ユーザーの代理操作を実行するために使用されるロールを引き受けるアクセス権限が必要です。そのためには、**`iam:PassRole`** アクセス権限を AWS Glue ユーザーまたはグループに追加します。このポリシーは、AWS Glue サービスロールの `AWSGlueServiceRole` で始まるロール、およびノートブックサーバーの作成に必要なロール `AWSGlueServiceNotebookRole` にアクセス権限を与えます。また、命名規則に従った `iam:PassRole` アクセス権限の独自のポリシーを作成することもできます。
> セキュリティのベストプラクティスに従って、ポリシーを強化して、Amazon S3 バケットおよび Amazon CloudWatch ロググループへのアクセスをさらに制限することが推奨されます。Amazon S3 ポリシーの例については、「Writing IAM Policies: How to Grant Access to an Amazon S3 Bucket」を参照してください。

このステップでは、`AWSGlueConsoleFullAccess` に似たポリシーを作成します。`AWSGlueConsoleFullAccess` の最新バージョンは IAM コンソールで取得できます。

1. AWS Management Console にサインインして、IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. ナビゲーションペインで、[Users] (ユーザー) または [User groups] (ユーザーグループ) を選択します。
3. 一覧から、ポリシーを埋め込むユーザーまたはグループの名前を選択します。
4. [Permissions (アクセス許可)] タブを選択して、必要であれば [Permissions policies (アクセス許可ポリシー)] セクションを展開します。
5. [Add Inline policy] (インラインポリシーの追加) リンクを選択します。

6. [ポリシーの作成] 画面で、JSON 編集のためのタブに移動します。次の JSON ステートメントを使用してポリシードキュメントを作成して、[ポリシーの確認] を選択します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:*",
                "redshift:DescribeClusters",
                "redshift:DescribeClusterSubnetGroups",
                "iam:ListRoles",
                "iam:ListUsers",
                "iam:ListGroups",
                "iam:ListRolePolicies",
                "iam:GetRole",
                "iam:GetRolePolicy",
                "iam:ListAttachedRolePolicies",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeSubnets",
                "ec2:DescribeVpcs",
                "ec2:DescribeVpcEndpoints",
                "ec2:DescribeRouteTables",
                "ec2:DescribeVpcAttribute",
                "ec2:DescribeKeyPairs",
                "ec2:DescribeInstances",
                "rds:DescribeDBInstances",
                "rds:DescribeDBClusters",
                "rds:DescribeDBSubnetGroups",
                "s3:ListAllMyBuckets",
                "s3:ListBucket",
                "s3:GetBucketAcl",
                "s3:GetBucketLocation",
                "cloudformation:DescribeStacks",
                "cloudformation:GetTemplateSummary",
                "dynamodb:ListTables",
                "kms:ListAliases",
                "kms:DescribeKey",
                "cloudwatch:GetMetricData",
                "cloudwatch:ListDashboards"
            ],
            "Resource": [
```

```
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::*/*aws-glue-*/*",
                "arn:aws:s3:::aws-glue-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "tag:GetResources"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:CreateBucket",
                "s3:PutBucketPublicAccessBlock"            ],
            "Resource": [
                "arn:aws:s3:::aws-glue-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:GetLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:*:*:/aws-glue/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
```

```
                "cloudformation:CreateStack",
                "cloudformation:DeleteStack"
            ],
            "Resource": "arn:aws:cloudformation:*:*:stack/aws-glue*/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:RunInstances"
            ],
            "Resource": [
                "arn:aws:ec2:*:*:instance/*",
                "arn:aws:ec2:*:*:key-pair/*",
                "arn:aws:ec2:*:*:image/*",
                "arn:aws:ec2:*:*:security-group/*",
                "arn:aws:ec2:*:*:network-interface/*",
                "arn:aws:ec2:*:*:subnet/*",
                "arn:aws:ec2:*:*:volume/*"
            ]
        },
        {
            "Action": [
                "iam:PassRole"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:iam::*:role/AWSGlueServiceRole*",
            "Condition": {
                "StringLike": {
                    "iam:PassedToService": [
                        "glue.amazonaws.com"
                    ]
                }
            }
        },
        {
            "Action": [
                "iam:PassRole"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:iam::*:role/AWSGlueServiceNotebookRole*",
            "Condition": {
                "StringLike": {
                    "iam:PassedToService": [
                        "ec2.amazonaws.com"
```

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

次の表は、このポリシーによって付与されたアクセス権限を示しています。

Action	リソース	説明
`"glue:*"`	`"*"`	すべての AWS Glue API オペレーションを実行する権限を付与します。 `"glue:*"` アクションを使用せずに以前にポリシーを作成したことがある場合は、ポリシーに次の個別のアクセス許可を追加する必要があります。 • "glue:ListCrawlers" • "glue:BatchGetCrawlers" • "glue:ListTriggers" • "glue:BatchGetTriggers" • "glue:ListDevEndpoints" • "glue:BatchGetDevEndpoints" • "glue:ListJobs" • "glue:BatchGetJobs"
`"redshift:DescribeClusters", "redshift:DescribeClusterSubnetGroups"`	`"*"`	Amazon Redshift に対する接続の作成を許可します。

Action	リソース	説明
"iam:ListRoles", "iam:ListRolePolicies", "iam:GetRole", "iam:GetRolePolicy", "iam:ListAttachedRolePolicies"	"*"	クローラ、ジョブ、開発エンドポイント、ノートブックサーバーを使用する際の IAM ロールの一覧表示を許可します。
"ec2:DescribeSecurityGroups", "ec2:DescribeSubnets", "ec2:DescribeVpcs", "ec2:DescribeVpcEndpoints", "ec2:DescribeRouteTables", "ec2:DescribeVpcAttribute", "ec2:DescribeKeyPairs", "ec2:DescribeInstances"	"*"	ジョブ、クローラ、開発エンドポイントを実行する場合に、VPC などの Amazon EC2 ネットワーク項目を設定することを許可します。
"rds:DescribeDBInstances"	"*"	Amazon RDS に対する接続の作成を許可します。
"s3:ListAllMyBuckets", "s3:ListBucket", "s3:GetBucketAcl", "s3:GetBucketLocation"	"*"	クローラ、ジョブ、開発エンドポイント、ノートブックサーバーを使用する際に、Amazon S3 バケットを一覧表示することを許可します。
"dynamodb:ListTables"	"*"	DynamoDB テーブルの一覧表示を許可します。
"kms:ListAliases", "kms:DescribeKey"	"*"	KMS キーを使用できます。

Action	リソース	説明
"cloudwatch:GetMetricData", "cloudwatch:ListDashboards"	"*"	CloudWatch メトリクスを使用できます。
"s3:GetObject", "s3:PutObject"	"arn:aws:s3::: aws-glue-*/*", "arn:aws:s3::: */*aws-glue-*/*", "arn:aws:s3::: aws-glue-*"	ETL スクリプトやノートブックサーバーの口ケーションなどのオブジェクトを格納する際に、アカウントで Amazon S3 オブジェクトを取得および配置することを許可します。 命名規則: 名前に aws-glue- のプレフィックスが付いている Amazon S3 バケットまたはフォルダにアクセス許可を付与します。
"tag:GetResources"	"*"	AWS タグの取得を許可します。

Action	リソース	説明
`"s3:CreateBucket", "s3:PutBucketPublicAccessBlock"`	`"arn:aws:s3::: aws-glue-*"`	ETL スクリプトやノートブックサーバーのロケーションなどのオブジェクトを格納する際に、アカウントで Amazon S3 バケットを作成することを許可します。 命名規則: 名前に aws-glue- のプレフィックスが付いている Amazon S3 バケットにアクセス許可を付与します。 AWS Glue による、パブリックアクセスをブロックするバケットの作成を有効化します。
`"logs:GetLogEvents"`	`"arn:aws:logs:*:*: /aws-glue/*"`	CloudWatch Logs の取得を許可します。 命名規則: AWS Glue は名前が [aws-glue-] で始まるロググループにログを書き込みます。

Action	リソース	説明
"cloudformation:CreateStack", "cloudformation:DeleteStack"	"arn:aws:cloudformation:*:*:stack/aws-glue*/*"	ノートブックサーバーで作業するときに AWS CloudFormation スタックの管理を許可します。 命名規則: AWS Glue は名前が [aws-glue] で始まるスタックを作成します。
"ec2:RunInstances"	"arn:aws:ec2:*:*:instance/*", "arn:aws:ec2:*:*:key-pair/*", "arn:aws:ec2:*:*:image/*", "arn:aws:ec2:*:*:security-group/*", "arn:aws:ec2:*:*:network-interface/*", "arn:aws:ec2:*:*:subnet/*", "arn:aws:ec2:*:*:volume/*"	開発エンドポイントとノートブックサーバーの実行を許可します。
"iam:PassRole"	"arn:aws:iam::*:role/ AWSGlueServiceRole*"	AWSGlueServiceRole で始まるロールに対する PassRole の許可を、AWS Glue が引き受けられるようにします。

Action	リソース	説明
"iam:PassRole"	"arn:aws:iam::*:role/ AWSGlueServiceNotebookRole*"	AWSGlueServiceNotebookRole で始まるロールに関するアクセス許可 PassRole を引き受けることを、Amazon EC2 に 許可します。
"iam:PassRole"	"arn:aws:iam::*:role/service-role/AWSGlueServiceRole*"	service-role/AWSGlueServiceRole で始まるロールに対する PassRole の許可を、AWS Glue が引き受けられるようにします。

7. [ポリシーの確認] セクションで、ポリシー名を入力します (例: GlueConsoleAccessPolicy) 。ポリシーが完成したら、[Create policy (ポリシーの作成)] を選択します。画面上部の赤いボックスにエラーが表示されていないことを確認します。報告されたエラーがあれば、修正します。

> **Note**
>
> [Use autoformatting (自動フォーマットを使用する)] を選択した場合は、ポリシーを開いたときおよび [Validate Policy (ポリシーの検証)] を選択したときに毎回、ポリシーが再フォーマットされます。

AWSGlueConsoleFullAccess 管理ポリシーをアタッチするには

AWSGlueConsoleFullAccess ポリシーをアタッチすると、AWS Glue コンソールユーザーが必要とするアクセス許可を提供できます。

> **Note**
>
> AWS Glue コンソールアクセス用に独自のポリシーを作成した場合は、このステップをスキップできます。

1. AWS Management Console にサインインして、IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. ナビゲーションペインで、[ポリシー] を選択します。
3. ポリシーのリストで、[AWSGlueConsoleFullAccess] ポリシーの横にあるチェックボックスを選択します。[Filter (フィルター)] メニューと検索ボックスを使用して、ポリシーのリストをフィルタリングできます。
4. [Policy actions] を選択して、[Attach] を選択します。
5. ポリシーをアタッチするユーザーを選択します。[Filter] メニューと検索ボックスを使用して、プリンシパルエンティティのリストをフィルタリングできます。ポリシーをアタッチするユーザーを選択し、[ポリシーのアタッチ] を選択します。

`AWSGlueConsoleSageMakerNotebookFullAccess` 管理ポリシーをアタッチするには

`AWSGlueConsoleSageMakerNotebookFullAccess` ポリシーをユーザーにアタッチすることで、AWS Glue コンソールで作成した SageMaker AI ノートブックを管理できます。AWS Glue コンソールのその他の必要なアクセス許可に加え、このポリシーは SageMaker AI ノートブックの管理に必要なリソースへのアクセス許可を付与します。

1. AWS Management Console にサインインして、IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. ナビゲーションペインで、[ポリシー] を選択します。
3. ポリシーのリストで、[AWSGlueConsoleSageMakerNotebookFullAccess] の横にあるチェックボックスを選択します。[Filter (フィルター)] メニューと検索ボックスを使用して、ポリシーのリストをフィルタリングできます。
4. [Policy actions] を選択して、[Attach] を選択します。
5. ポリシーをアタッチするユーザーを選択します。[Filter] メニューと検索ボックスを使用して、プリンシパルエンティティのリストをフィルタリングできます。ポリシーをアタッチするユーザーを選択し、[ポリシーのアタッチ] を選択します。

CloudWatchLogsReadOnlyAccess 管理ポリシーをアタッチするには

[CloudWatchLogsReadOnlyAccess] ポリシーをユーザーにアタッチして、CloudWatch Logs コンソールで AWS Glue によって作成されたログを表示できます。

1. AWS Management Console にサインインして、IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. ナビゲーションペインで、[ポリシー] を選択します。
3. ポリシーのリストで、[CloudWatchLogsReadOnlyAccess] ポリシーの横にあるチェックボックスを選択します。[Filter (フィルター)] メニューと検索ボックスを使用して、ポリシーのリストをフィルタリングできます。
4. [Policy actions] を選択して、[Attach] を選択します。
5. ポリシーをアタッチするユーザーを選択します。[Filter] メニューと検索ボックスを使用して、プリンシパルエンティティのリストをフィルタリングできます。ポリシーをアタッチするユーザーを選択し、[ポリシーのアタッチ] を選択します。

AWSCloudFormationReadOnlyAccess 管理ポリシーをアタッチするには

[AWSCloudFormationReadOnlyAccess] ポリシーをユーザーにアタッチして、AWS CloudFormation コンソールで AWS Glue が使用する AWS CloudFormation スタックを表示できます。

1. AWS Management Console にサインインして、IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. ナビゲーションペインで、[ポリシー] を選択します。
3. ポリシーのリストで、[AWSCloudFormationReadOnlyAccess] ポリシーの横にあるチェックボックスをオンにします。[Filter (フィルター)] メニューと検索ボックスを使用して、ポリシーのリストをフィルタリングできます。
4. [Policy actions] を選択して、[Attach] を選択します。
5. ポリシーをアタッチするユーザーを選択します。[Filter] メニューと検索ボックスを使用して、プリンシパルエンティティのリストをフィルタリングできます。ポリシーをアタッチするユーザーを選択し、[ポリシーのアタッチ] を選択します。

AmazonAthenaFullAccess 管理ポリシーをアタッチするには

[AmazonAthenaFullAccess] ポリシーをユーザーにアタッチすると、Amazon S3 データを Athena コンソールで表示できます。

1. AWS Management Console にサインインして、IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. ナビゲーションペインで、[ポリシー] を選択します。
3. ポリシーのリストで、[AmazonAthenaFullAccess] ポリシーの横にあるチェックボックスを選択します。[Filter (フィルター)] メニューと検索ボックスを使用して、ポリシーのリストをフィルタリングできます。
4. [Policy actions] を選択して、[Attach] を選択します。
5. ポリシーをアタッチするユーザーを選択します。[Filter] メニューと検索ボックスを使用して、プリンシパルエンティティのリストをフィルタリングできます。ポリシーをアタッチするユーザーを選択し、[ポリシーのアタッチ] を選択します。

ステップ 4: ノートブックサービス用に IAM ポリシーを作成する

開発エンドポイントでノートブックを使用する予定の場合は、ノートブックサーバーの作成時にアクセス許可を指定する必要があります。AWS Identity and Access Management (IAM) を使用してアクセス権限を提供できます。

このポリシーは、AWS Glue がこのポリシーを使用してロールを引き受ける際に必要となる、アカウント内のリソースを管理する Amazon S3 アクションの一部を許可します。このポリシーで指定されているリソースの一部は、 AWS Glue で (Amazon S3 バケット、Amazon S3 ETL スクリプト、および Amazon EC2 リソース用として) 使用されるデフォルトの名前を参照しています。分かりやすさのために、AWS Glue のデフォルトでは一部の Amazon S3 オブジェクトを、アカウント内の `aws-glue-*` のプレフィックスが付いたバケットに書き込んでいます。

> **Note**
>
> AWS 管理ポリシー **`AWSGlueServiceNotebookRole`** を使用する場合は、このステップをスキップできます。

このステップでは、`AWSGlueServiceNotebookRole` に似たポリシーを作成します。`AWSGlueServiceNotebookRole` の最新バージョンは IAM コンソールで取得できます。

ノートブック用に IAM ポリシーを作成するには

1. AWS Management Console にサインインして、IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。

2. 左のナビゲーションペインの [ポリシー] を選択します。
3. [ポリシーの作成] を選択します。
4. [ポリシーの作成] 画面で、JSON 編集のためのタブに移動します。次の JSON ステートメントを使用してポリシードキュメントを作成して、[ポリシーの確認] を選択します。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "glue:CreateDatabase",
            "glue:CreatePartition",
            "glue:CreateTable",
            "glue:DeleteDatabase",
            "glue:DeletePartition",
            "glue:DeleteTable",
            "glue:GetDatabase",
            "glue:GetDatabases",
            "glue:GetPartition",
            "glue:GetPartitions",
            "glue:GetTable",
            "glue:GetTableVersions",
            "glue:GetTables",
            "glue:UpdateDatabase",
            "glue:UpdatePartition",
            "glue:UpdateTable",
            "glue:GetJobBookmark",
            "glue:ResetJobBookmark",
            "glue:CreateConnection",
            "glue:CreateJob",
            "glue:DeleteConnection",
            "glue:DeleteJob",
            "glue:GetConnection",
            "glue:GetConnections",
            "glue:GetDevEndpoint",
            "glue:GetDevEndpoints",
            "glue:GetJob",
            "glue:GetJobs",
            "glue:UpdateJob",
            "glue:BatchDeleteConnection",
            "glue:UpdateConnection",
```

```
            "glue:GetUserDefinedFunction",
            "glue:UpdateUserDefinedFunction",
            "glue:GetUserDefinedFunctions",
            "glue:DeleteUserDefinedFunction",
            "glue:CreateUserDefinedFunction",
            "glue:BatchGetPartition",
            "glue:BatchDeletePartition",
            "glue:BatchCreatePartition",
            "glue:BatchDeleteTable",
            "glue:UpdateDevEndpoint",
            "s3:GetBucketLocation",
            "s3:ListBucket",
            "s3:ListAllMyBuckets",
            "s3:GetBucketAcl"
         ],
         "Resource":[
            "*"
         ]
      },
      {
         "Effect":"Allow",
         "Action":[
            "s3:GetObject"
         ],
         "Resource":[
            "arn:aws:s3:::crawler-public*",
            "arn:aws:s3:::aws-glue*"
         ]
      },
      {
         "Effect":"Allow",
         "Action":[
            "s3:PutObject",
            "s3:DeleteObject"
         ],
         "Resource":[
            "arn:aws:s3:::aws-glue*"
         ]
      },
      {
         "Effect":"Allow",
         "Action":[
            "ec2:CreateTags",
            "ec2:DeleteTags"
```

```
         ],
         "Condition":{
            "ForAllValues:StringEquals":{
               "aws:TagKeys":[
                  "aws-glue-service-resource"
               ]
            }
         },
         "Resource":[
            "arn:aws:ec2:*:*:network-interface/*",
            "arn:aws:ec2:*:*:security-group/*",
            "arn:aws:ec2:*:*:instance/*"
         ]
      }
   ]
}
```

次の表は、このポリシーによって付与されたアクセス権限を示しています。

[アクション]	リソース	説明
"glue:*"	"*"	すべての AWS Glue API オペレーションを実行する権限を付与します。
"s3:GetBucketLocation", "s3:ListBucket", "s3:ListAllMyBuckets", "s3:GetBucketAcl"	"*"	ノートブックサーバーからの Amazon S3 バケットの一覧表示を許可します。

[アクション]	リソース	説明
`"s3:GetObject"`	`"arn:aws:s3:::crawler-public*", "arn:aws:s3:::aws-glue-*"`	ノートブックのサンプルやチュートリアルで使用されている Amazon S3 オブジェクトを取得できます。 命名規則: Amazon S3 バケット名は 「crawler-public」および「aws-glue-」で始まります。
`"s3:PutObject", "s3:DeleteObject"`	`"arn:aws:s3:::aws-glue*"`	ノートブックからアカウントに対する、Amazon S3 オブジェクトの配置と削除を許可します。 命名規則: [aws-glue] という Amazon S3 フォルダを使用します。
`"ec2:CreateTags", "ec2:DeleteTags"`	`"arn:aws:ec2:*:*:network-interface/*", "arn:aws:ec2:*:*:security-group/*", "arn:aws:ec2:*:*:instance/*"`	ノートブックサーバー用に作成された Amazon EC2 リソースのタグ付けを許可します。 命名規則: AWS Glue は Amazon EC2 インスタンスを、[aws-glue-service-resource] によりタグ付けします。

5. [Review Policy (ポリシーの確認)] 画面で、[Policy Name (ポリシー名)] （[GlueServiceNotebookPolicyDefault] など）を入力します。オプションの説明を入力し、ポリシーが適切であることを確認したら、[Create policy] を選択します。

ステップ 5: ノートブックサービス用に IAM ロールを作成する

開発エンドポイントでノートブックを使用する予定がある場合は、IAM ロールのアクセス許可を付与する必要があります。AWS Identity and Access Management IAM を使用しながら IAM ロールを介して、これらのアクセス許可を付与できます。

> **Note**
>
> IAM コンソールを使用して IAM ロールを作成すると、コンソールによりインスタンスプロファイルが自動的に作成され、対応するロールと同じ名前が付けられます。

ノートブック用に IAM ロールを作成するには

1. AWS Management Console にサインインして、IAM コンソール https://console.aws.amazon.com/iam/ を開きます。
2. 左のナビゲーションペインで、[ロール] を選択してください。
3. [ロールの作成] を選択してください。
4. ロールタイプについては、[AWS Service] (サービス) を選択した後に [EC2] を見つけて選択し、ユースケースで [EC2] を選択した後、[Next: Permissions] (次へ: アクセス許可) をクリックします。
5. [アクセス権限ポリシーを添付する] のページで、必要なアクセス権を含むポリシーを選択します。例えば、一般的な AWS Glue 許可には AWSGlueServiceNotebookRole、Amazon S3 リソースへのアクセスには AWS マネージドポリシー AmazonS3FullAccess を選択します。続いて、[Next: Review] をクリックします。

   > **Note**
   >
   > Amazon S3 のソースとターゲットに対するアクセス許可を、このロールのポリシーの 1 つにより付与してください。また、ノートブックサーバーの作成時にノートブックを保管する場所へのフルアクセスがポリシーで許可されていることを確認してください。独自のポリシーを、特定の Amazon S3 リソースにアクセスするために指定します。リソースの Amazon S3 ポリシーの作成については、「Specifying Resources in a Policy」を参照してください。
   > SSE-KMS で暗号化された Amazon S3 のソースとターゲットにアクセスする予定がある場合は、ノートブックがデータを復号化できるようにポリシーをアタッチします。詳細

については、「Protecting Data Using Server-Side Encryption with AWS KMS-Managed Keys (SSE-KMS)」を参照してください。
以下に例を示します。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "kms:Decrypt"
         ],
         "Resource":[
            "arn:aws:kms:*:account-id-without-hyphens:key/key-id"
         ]
      }
   ]
}
```

6. ロール名 に、ロールの名前を入力します。コンソールユーザーからノートブックサーバーにロールを渡すには、名前に文字列 AWSGlueServiceNotebookRole のプレフィックスが付けられたロールを作成します。AWS Glue が提供するポリシーでは、IAM サービスロールが AWSGlueServiceNotebookRole で始まることを想定しています。それ以外の場合は、ユーザーにポリシーを追加して、IAM ロールに対する iam:PassRole の許可を命名規則に一致させる必要があります。たとえば、AWSGlueServiceNotebookRoleDefault と入力します。次に、[Create role (ロールの作成)] を選択します。

ステップ 6: SageMaker AI ノートブック用の IAM ポリシーを作成する

開発エンドポイントで SageMaker AI ノートブックを使用する予定の場合は、ノートブックの作成時にアクセス許可を指定する必要があります。AWS Identity and Access Management (IAM) を使用してアクセス権限を提供できます。

SageMaker AI ノートブック用の IAM ポリシーを作成するには

1. AWS Management Console にサインインして、IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. 左のナビゲーションペインの [ポリシー] を選択します。
3. [ポリシーの作成] を選択します。

4. [Create Policy] ページで、JSON を編集するタブに移動します。次の JSON ステートメントを使用して、ポリシードキュメントを作成します。環境の *bucket-name*、*region-code*、*account-id* を編集します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "s3:ListBucket"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:s3:::bucket-name"
            ]
        },
        {
            "Action": [
                "s3:GetObject"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:s3:::bucket-name*"
            ]
        },
        {
            "Action": [
                "logs:CreateLogStream",
                "logs:DescribeLogStreams",
                "logs:PutLogEvents",
                "logs:CreateLogGroup"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:logs:region-code:account-id:log-group:/aws/sagemaker/*",
                "arn:aws:logs:region-code:account-id:log-group:/aws/sagemaker/
*:log-stream:aws-glue-*"
            ]
        },
        {
            "Action": [
                "glue:UpdateDevEndpoint",
                "glue:GetDevEndpoint",
```

```
                "glue:GetDevEndpoints"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:glue:region-code:account-id:devEndpoint/*"
            ]
        },
        {
            "Action": [
                "sagemaker:ListTags"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:sagemaker:region-code:account-id:notebook-instance/*"
            ]
         }
    ]
}
```

次に [ポリシーの確認] を選択します。

次の表は、このポリシーによって付与されたアクセス権限を示しています。

Action	リソース	説明
"s3:ListBucket*"	"arn:aws:s3::: *bucket-name* "	Amazon S3 バケットを一覧表示するアクセス許可を付与します
"s3:GetObject"	"arn:aws:s3::: *bucket-name* *"	SageMaker AI ノートブックで使用される Amazon S3 オブジェクトを取得するためのアクセス許可を付与します。

Action	リソース	説明
"logs:CreateLogStream", "logs:DescribeLogStreams", "logs:PutLogEvents", "logs:CreateLogGroup"	"arn:aws:logs: *region-code* :*account-id* :log-group:/aws/sagemaker/*", "arn:aws:logs: *region-code* :*account-id* :log-group:/aws/sagemaker/*:log-stream:aws-glue-*"	ノートブックから Amazon CloudWatch Logs にログを書き込むためのアクセス許可を付与します。 命名規則: 名前が [aws-glue] で始まるロググループに書き込みます。
"glue:UpdateDevEndpoint", "glue:GetDevEndpoint", "glue:GetDevEndpoints"	"arn:aws:glue: *region-code* :*account-id* :devEndpoint/*"	SageMaker AI ノートブックから開発エンドポイントを使用するためのアクセス許可を付与します。
"sagemaker:ListTags"	"arn:aws:sagemaker: *region-code* :*account-id* :notebook-instance/*"	SageMaker AI リソースのタグを返すためのアクセス許可を付与します。この aws-glue-dev-endpoint タグは、ノートブックを開発エンドポイントに接続するために SageMaker AI ノートブック上で必要になります。

5. [Review Policy] (ポリシーの確認) 画面で、[Policy Name] (ポリシー名)に AWSGlueSageMakerNotebook (など)を入力します。オプションの説明を入力し、ポリシーが適切であることを確認したら、[Create policy] を選択します。

ステップ 7: SageMaker AI ノートブック用の IAM ロールを作成する

開発エンドポイントで SageMaker AI ノートブックを使用する予定がある場合は、IAM ロールのアクセス許可を付与する必要があります。AWS Identity and Access Management (IAM) を使用しながら IAM ロールを介して、アクセス許可を付与できます。

SageMaker AI ノートブック用の IAM ロールを作成するには

1. AWS Management Console にサインインして、IAM コンソール https://console.aws.amazon.com/iam/ を開きます。
2. 左のナビゲーションペインで、[ロール] を選択してください。
3. [ロールの作成] を選択してください。
4. ロールタイプについては、[AWS Service] を選択し、[SageMaker] を見つけて選択してから、[SageMaker - Execution] ユースケースを選択します。その後、[Next] (次へ) を選択します。
5. [Attach permissions policy] ページで、必要なアクセス許可を含むポリシー ([AmazonSageMakerFullAccess] など) を選択します。[次へ: レビュー] を選択します。

   SSE-KMS で暗号化された Amazon S3 のソースとターゲットにアクセスする予定がある場合は、次の例に示すように、ノートブックによるデータの復号化を許可するポリシーをアタッチしてください。詳細については、「Protecting Data Using Server-Side Encryption with AWS KMS-Managed Keys (SSE-KMS)」を参照してください。

   ```
   {
      "Version":"2012-10-17",
      "Statement":[
         {
            "Effect":"Allow",
            "Action":[
               "kms:Decrypt"
            ],
            "Resource":[
               "arn:aws:kms:*:account-id-without-hyphens:key/key-id"
            ]
         }
      ]
   }
   ```

6. ロール名 に、ロールの名前を入力します。コンソールユーザーから SageMaker AI にロールを渡すには、文字列 `AWSGlueServiceSageMakerNotebookRole` のプレフィックスが付けられた名前を使用します。AWS Glue が提供するポリシーでは、IAM ロールが `AWSGlueServiceSageMakerNotebookRole` で始まることを想定しています。それ以外の場合は、ユーザーにポリシーを追加して、IAM ロールに対する `iam:PassRole` の許可を命名規則に一致させる必要があります。

例えば、AWSGlueServiceSageMakerNotebookRole-Default を入力した上で、[Create role] (ロールを作成) をクリックします。

7. ロールを作成した後に、AWS Glue から SageMaker AI ノートブックを作成するために必要な、追加のアクセス許可を付与するポリシーをアタッチします。

   作成したロール AWSGlueServiceSageMakerNotebookRole-Default を開き、[Attach policies] (ポリシーのアタッチ) をクリックします。作成した AWSGlueSageMakerNotebook ポリシーを、ロールにアタッチします。

AWS Glue のアクセスコントロールポリシーの例

このセクションでは、アイデンティティベースの (IAM) アクセスコントロールポリシーと、AWS Glue リソースポリシーの両方の例を示します。

目次

- AWS Glue のアイデンティティベースポリシーの例
  - ポリシーに関するベストプラクティス
  - 特定の AWS Glue オブジェクトにのみ適用されるリソースレベルのアクセス許可
  - AWS Glue コンソールの使用
  - 自分の権限の表示をユーザーに許可する
  - テーブルへの読み取り専用のアクセスを許可する
  - GetTables アクセス許可によるテーブルのフィルタリング
  - テーブルとすべてのパーティションへのフルアクセスを付与する
  - 名前のプレフィックスと明示的な拒否によりアクセスを制御する
  - タグを使用してアクセスを許可する
  - タグを使用してアクセスを拒否する
  - リストとバッチ API オペレーションでタグを使用する
  - 条件キーまたはコンテキストキーを使用して設定を制御する
    - 条件キーを使って設定を制御するポリシーを制御する
    - 条件キーを使って設定を制御するポリシーを制御する
  - ID によるデータプレビューセッションの作成を拒否する
- AWS Glue のリソースベースのポリシーの例

- AWS Glue でリソースベースのポリシーを使用する際の考慮事項
- 同じアカウント内のアクセスをコントロールするリソースポリシーの使用

AWS Glue のアイデンティティベースポリシーの例

デフォルトでは、 ユーザーおよびロールには、AWS Glue リソースを作成または変更するアクセス許可は付与されていません。また、AWS Management Console、AWS Command Line Interface (AWS CLI)、または AWS API を使用してタスクを実行することもできません。IAM 管理者は、リソースで必要なアクションを実行するための権限をユーザーに付与する IAM ポリシーを作成できます。その後、管理者はロールに IAM ポリシーを追加し、ユーザーはロールを引き継ぐことができます。

これらサンプルの JSON ポリシードキュメントを使用して、IAM アイデンティティベースのポリシーを作成する方法については、「IAM ユーザーガイド」の「IAM ポリシーを作成する (コンソール)」を参照してください。

AWS Glue で定義されるアクションとリソースタイプ (リソースタイプごとの ARN の形式を含む) の詳細については、「サービス認可リファレンス」の「AWS Glue のアクション、リソース、および条件キー」を参照してください。

> **Note**
>
> このセクションで取り上げる例は、すべて `us-west-2` リージョンを使用しています。このリージョンは、ご自身が使用する AWS リージョンに置き換えることが可能です。

トピック

- ポリシーに関するベストプラクティス
- 特定の AWS Glue オブジェクトにのみ適用されるリソースレベルのアクセス許可
- AWS Glue コンソールの使用
- 自分の権限の表示をユーザーに許可する
- テーブルへの読み取り専用のアクセスを許可する
- GetTables アクセス許可によるテーブルのフィルタリング
- テーブルとすべてのパーティションへのフルアクセスを付与する
- 名前のプレフィックスと明示的な拒否によりアクセスを制御する

- タグを使用してアクセスを許可する
- タグを使用してアクセスを拒否する
- リストとバッチ API オペレーションでタグを使用する
- 条件キーまたはコンテキストキーを使用して設定を制御する
- ID によるデータプレビューセッションの作成を拒否する

ポリシーに関するベストプラクティス

アイデンティティベースのポリシーは、ユーザーのアカウントで誰が AWS Glue リソースを作成し、これにアクセスし、これを削除できるかを決定します。これらのアクションを実行すると、AWS アカウントに料金が発生する可能性があります。アイデンティティベースポリシーを作成したり編集したりする際には、以下のガイドラインと推奨事項に従ってください:

- AWS マネージドポリシーを使用して開始し、最小特権の許可に移行する – ユーザーとワークロードへの許可の付与を開始するには、多くの一般的なユースケースのために許可を付与する AWS マネージドポリシーを使用します。これらは AWS アカウントで使用できます。ユースケース別に AWS カスタマーマネージドポリシーを定義して、マネージドポリシーを絞り込むことをお勧めします。詳細については、「IAM ユーザーガイド」の「AWS マネージドポリシー」または「ジョブ機能の AWS マネージドポリシー」を参照してください。
- 最小特権を適用する – IAM ポリシーで許可を設定する場合は、タスクの実行に必要な許可のみを付与します。これを行うには、特定の条件下で特定のリソースに対して実行できるアクションを定義します。これは、最小特権アクセス許可とも呼ばれています。IAM を使用して許可を適用する方法の詳細については、「IAM ユーザーガイド」の「IAM でのポリシーとアクセス許可」を参照してください。
- IAM ポリシーで条件を使用してアクセスをさらに制限する - ポリシーに条件を追加して、アクションやリソースへのアクセスを制限できます。例えば、ポリシー条件を記述して、すべてのリクエストを SSL を使用して送信するように指定できます。また、AWS CloudFormation などの特定の AWS のサービス を介して使用する場合、条件を使用してサービスアクションへのアクセスを許可することもできます。詳細については、「IAM ユーザーガイド」の「IAM JSON ポリシー要素:条件」を参照してください。
- IAM Access Analyzer を使用して IAM ポリシーを検証し、安全で機能的な権限を確保する - IAM Access Analyzer は、新規および既存のポリシーを検証して、ポリシーが IAM ポリシー言語 (JSON) および IAM のベストプラクティスに準拠するようにします。IAM アクセスアナライザーは 100 を超えるポリシーチェックと実用的な推奨事項を提供し、安全で機能的なポリシーの作成をサポートします。詳細については、「IAM ユーザーガイド」の「IAM Access Analyzer でポリシーを検証する」を参照してください。

- 多要素認証 (MFA) を要求する – AWS アカウントで IAM ユーザーまたはルートユーザーを要求するシナリオがある場合は、セキュリティを強化するために MFA をオンにします。API オペレーションが呼び出されるときに MFA を必須にするには、ポリシーに MFA 条件を追加します。詳細については、「IAM ユーザーガイド」の「MFA を使用した安全な API アクセス」を参照してください。

IAM でのベストプラクティスの詳細については、「IAM ユーザーガイド」の「IAM でのセキュリティのベストプラクティス」を参照してください。

特定の AWS Glue オブジェクトにのみ適用されるリソースレベルのアクセス許可

AWS Glue の特定のオブジェクトに限り、詳細なコントロールを定義できます。したがって、Resource ステートメントの Amazon リソースネーム (ARN) を許可する API オペレーションと ARN を許可しない API オペレーションが混在しないようにクライアントの IAM ポリシーを記述する必要があります。

たとえば、次の IAM ポリシーは GetClassifier と GetJobRun の API オペレーションを許可します。AWS Glue は分類子とジョブ実行の ARN を許可しないため、このポリシーでは Resource を * として定義します。ARN は GetDatabase や GetTable などの特定の API オペレーションに対して許可されるため、ARN はポリシーの後半で指定できます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:GetClassifier*",
                "glue:GetJobRun*"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "glue:Get*"
            ],
            "Resource": [
                "arn:aws:glue:us-east-1:123456789012:catalog",
                "arn:aws:glue:us-east-1:123456789012:database/default",
                "arn:aws:glue:us-east-1:123456789012:table/default/e*1*",
```

```
                "arn:aws:glue:us-east-1:123456789012:connection/connection2"
            ]
        }
    ]
}
```

ARN を許可する AWS Glue オブジェクトのリストについては、「AWS Glue リソース ARN の指定」を参照してください。

AWS Glue コンソールの使用

AWS Glue コンソールにアクセスするには、最小限のアクセス許可が必要になります。これらのアクセス許可により、AWS アカウント にある AWS Glue リソースの詳細を一覧で表示できます。最小限必要なアクセス許可よりも制限が厳しいアイデンティティベースのポリシーを作成すると、そのポリシーを持つエンティティ (ユーザーまたはロール) ではコンソールが意図したとおりに機能しません。

AWS CLI または AWS API のみを呼び出すユーザーには、最小限のコンソール権限を付与する必要はありません。代わりに、実行しようとしている API オペレーションに一致するアクションのみへのアクセスを許可します。

ユーザーとロールが引き続き AWS Glue コンソールを使用できるようにするには、これらのエンティティに AWS Glue *ConsoleAccess* または *ReadOnly* AWS マネージドポリシーもアタッチします。詳細については、「IAM ユーザーガイド」の「ユーザーへのアクセス許可の追加」を参照してください。

ユーザーが AWS Glue コンソールを使用するには、AWS アカウントで AWS Glue リソースの使用を許可する最小限のアクセス許可セットが必要です。これらの AWS Glue アクセス許可に加えて、コンソールでは次のサービスからのアクセス許可が必要になります。

- ログを表示するための Amazon CloudWatch Logs のアクセス許可。
- ロールをリスト化して渡すための AWS Identity and Access Management (IAM) のアクセス許可。
- スタックを操作する AWS CloudFormation のアクセス許可。
- Amazon Elastic Compute Cloud (Amazon EC2) が VPC、サブネット、セキュリティグループ、インスタンス、およびその他のオブジェクトをリストするためのアクセス許可。
- Amazon Simple Storage Service (Amazon S3) が バケットとオブジェクトをリストし、スクリプトを取得および保存するためのアクセス許可。
- クラスターでの作業に必要な Amazon Redshift のアクセス許可。

- インスタンスをリスト化するための Amazon Relational Database Service (Amazon RDS) のアクセス許可。

ユーザーが AWS Glue コンソールを表示して操作するために必要なアクセス許可の詳細については、「ステップ 3: AWS Glue にアクセスするユーザーまたはグループにポリシーをアタッチする」を参照してください。

これらの最小限必要なアクセス許可よりも制限された IAM ポリシーを作成している場合、その IAM ポリシーを使用するユーザーに対してコンソールは意図したとおりには機能しません。これらのユーザーが引き続き AWS Glue コンソールを使用できるようにするには、AWS Glue の AWS マネージド (事前定義) ポリシー で述べたとおり AWSGlueConsoleFullAccess マネージドポリシーもアタッチします。

自分の権限の表示をユーザーに許可する

この例では、ユーザーアイデンティティにアタッチされたインラインおよびマネージドポリシーの表示を IAM ユーザーに許可するポリシーの作成方法を示します。このポリシーには、コンソールで、または AWS CLI か AWS API を使用してプログラム的に、このアクションを完了する権限が含まれています。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
```

```
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

テーブルへの読み取り専用のアクセスを許可する

次のポリシーは、データベース db1 の books テーブルに対する読み取り専用アクセス許可を付与します。Amazon リソースネーム (ARN) の詳細については、「データカタログの ARN」を参照してください。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "GetTablesActionOnBooks",
      "Effect": "Allow",
      "Action": [
        "glue:GetTables",
        "glue:GetTable"
      ],
      "Resource": [
        "arn:aws:glue:us-west-2:123456789012:catalog",
        "arn:aws:glue:us-west-2:123456789012:database/db1",
        "arn:aws:glue:us-west-2:123456789012:table/db1/books"
      ]
    }
  ]
}
```

このポリシーでは db1 という名前のデータベースにある books という名前のテーブルへの読み取り専用許可を付与します。テーブルへの Get アクセスを許可するには、カタログとデータベースリソースへのアクセス許可も必要になります。

次のポリシーは、データベース db1 にテーブル tb1 を作成するための必要最小限のアクセス許可を付与します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "glue:CreateTable"
      ],
      "Resource": [
        "arn:aws:glue:us-west-2:123456789012:table/db1/tbl1",
        "arn:aws:glue:us-west-2:123456789012:database/db1",
        "arn:aws:glue:us-west-2:123456789012:catalog"
      ]
    }
  ]
}
```

GetTables アクセス許可によるテーブルのフィルタリング

データベース db1 内に、customers、stores、および store_sales の 3 つのテーブルがあるとします。次のポリシーは、GetTables アクセス許可を stores および store_sales に付与しますが、customers には付与しません。このポリシーで GetTables を呼び出すと、結果には 2 つの認可されたテーブルのみが含まれます (customers テーブルは返されません)。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "GetTablesExample",
            "Effect": "Allow",
            "Action": [
                "glue:GetTables"
            ],
            "Resource": [
                "arn:aws:glue:us-west-2:123456789012:catalog",
                "arn:aws:glue:us-west-2:123456789012:database/db1",
                "arn:aws:glue:us-west-2:123456789012:table/db1/store_sales",
                "arn:aws:glue:us-west-2:123456789012:table/db1/stores"
            ]
```

```
        }
    ]
}
```

store* を使用して store で始まる任意のテーブル名に一致させることで、前述のポリシーを簡素化できます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "GetTablesExample2",
            "Effect": "Allow",
            "Action": [
                "glue:GetTables"
            ],
            "Resource": [
                "arn:aws:glue:us-west-2:123456789012:catalog",
                "arn:aws:glue:us-west-2:123456789012:database/db1",
                "arn:aws:glue:us-west-2:123456789012:table/db1/store*"
            ]
        }
    ]
}
```

同様に、/db1/* を使用して db1 内のすべてのテーブルと一致させることで、次のポリシーによって db1 内のすべてのテーブルへのアクセスが GetTables に付与されます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "GetTablesReturnAll",
            "Effect": "Allow",
            "Action": [
                "glue:GetTables"
            ],
            "Resource": [
                "arn:aws:glue:us-west-2:123456789012:catalog",
                "arn:aws:glue:us-west-2:123456789012:database/db1",
                "arn:aws:glue:us-west-2:123456789012:table/db1/*"
            ]
```

```
        }
    ]
}
```

テーブルの ARN を指定しない場合、GetTables の呼び出しは成功しますが、空のリストが返されます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "GetTablesEmptyResults",
            "Effect": "Allow",
            "Action": [
                "glue:GetTables"
            ],
            "Resource": [
                "arn:aws:glue:us-west-2:123456789012:catalog",
                "arn:aws:glue:us-west-2:123456789012:database/db1"
            ]
        }
    ]
}
```

ポリシーでデータベースの ARN が欠落している場合は、 GetTables の呼び出しが失敗して AccessDeniedException が発生します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "GetTablesAccessDeny",
            "Effect": "Allow",
            "Action": [
                "glue:GetTables"
            ],
            "Resource": [
                "arn:aws:glue:us-west-2:123456789012:catalog",
                "arn:aws:glue:us-west-2:123456789012:table/db1/*"
            ]
        }
    ]
```

```
}
```

テーブルとすべてのパーティションへのフルアクセスを付与する

次のポリシーは、データベース db1 で books という名前のテーブルのすべての許可を付与します。これには、テーブル自体、テーブルのアーカイブされたバージョン、テーブルのすべてのパーティションでの読み込みおよび書き込みアクセス許可が含まれます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "FullAccessOnTable",
            "Effect": "Allow",
            "Action": [
                "glue:CreateTable",
                "glue:GetTable",
                "glue:GetTables",
                "glue:UpdateTable",
                "glue:DeleteTable",
                "glue:BatchDeleteTable",
                "glue:GetTableVersion",
                "glue:GetTableVersions",
                "glue:DeleteTableVersion",
                "glue:BatchDeleteTableVersion",
                "glue:CreatePartition",
                "glue:BatchCreatePartition",
                "glue:GetPartition",
                "glue:GetPartitions",
                "glue:BatchGetPartition",
                "glue:UpdatePartition",
                "glue:DeletePartition",
                "glue:BatchDeletePartition"
            ],
            "Resource": [
                        "arn:aws:glue:us-west-2:123456789012:catalog",
                        "arn:aws:glue:us-west-2:123456789012:database/db1",
                        "arn:aws:glue:us-west-2:123456789012:table/db1/books"
                        ]
        }
    ]
}
```

前述のポリシーは、実際に次のように簡素化できます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "FullAccessOnTable",
            "Effect": "Allow",
            "Action": [
                "glue:*Table*",
                "glue:*Partition*"
            ],
            "Resource": [
                "arn:aws:glue:us-west-2:123456789012:catalog",
                "arn:aws:glue:us-west-2:123456789012:database/db1",
                "arn:aws:glue:us-west-2:123456789012:table/db1/books"
            ]
        }
    ]
}
```

きめ細かなアクセスコントロールの最小粒度はテーブルレベルであることに注意してください。つまり、テーブル内の一部のパーティションへのアクセス許可をユーザーに付与できませんが、他のパーティションには付与できます。また、一部のテーブルの列には付与できませんが、他のテーブルの列には付与できます。ユーザーはすべてのテーブルへのアクセス許可を持っているか、どのアクセス許可も持っていません。

名前のプレフィックスと明示的な拒否によりアクセスを制御する

ここでは、AWS Glue Data Catalog 内にあるデータベースとテーブルが、名前のプレフィックスを使用して整理されていると想定します。開発ステージのデータベースには、名前のプレフィックス dev- があります。実稼働環境のデータベースには、名前のプレフィックス prod- があります。次のポリシーを使用して、開発者に dev- プレフィックスがある、すべてのデータベース、テーブル、UDF などへのフルアクセスを付与できます。ただし、prod- プレフィックスを使用して、すべての読み取り専用アクセス許可を付与します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DevAndProdFullAccess",
            "Effect": "Allow",
```

```
            "Action": [
                "glue:*Database*",
                "glue:*Table*",
                "glue:*Partition*",
                "glue:*UserDefinedFunction*",
                "glue:*Connection*"
            ],
            "Resource": [
                "arn:aws:glue:us-west-2:123456789012:catalog",
                "arn:aws:glue:us-west-2:123456789012:database/dev-*",
                "arn:aws:glue:us-west-2:123456789012:database/prod-*",
                "arn:aws:glue:us-west-2:123456789012:table/dev-*/*",
                "arn:aws:glue:us-west-2:123456789012:table/*/dev-*",
                "arn:aws:glue:us-west-2:123456789012:table/prod-*/*",
                "arn:aws:glue:us-west-2:123456789012:table/*/prod-*",
                "arn:aws:glue:us-west-2:123456789012:userDefinedFunction/dev-*/*",
                "arn:aws:glue:us-west-2:123456789012:userDefinedFunction/*/dev-*",
                "arn:aws:glue:us-west-2:123456789012:userDefinedFunction/prod-*/*",
                "arn:aws:glue:us-west-2:123456789012:userDefinedFunction/*/prod-*",
                "arn:aws:glue:us-west-2:123456789012:connection/dev-*",
                "arn:aws:glue:us-west-2:123456789012:connection/prod-*"
            ]
        },
        {
            "Sid": "ProdWriteDeny",
            "Effect": "Deny",
            "Action": [
                "glue:*Create*",
                "glue:*Update*",
                "glue:*Delete*"
            ],
            "Resource": [
                "arn:aws:glue:us-west-2:123456789012:database/prod-*",
                "arn:aws:glue:us-west-2:123456789012:table/prod-*/*",
                "arn:aws:glue:us-west-2:123456789012:table/*/prod-*",
                "arn:aws:glue:us-west-2:123456789012:userDefinedFunction/prod-*/*",
                "arn:aws:glue:us-west-2:123456789012:userDefinedFunction/*/prod-*",
                "arn:aws:glue:us-west-2:123456789012:connection/prod-*"
            ]
        }
    ]
}
```

前述のポリシーの 2 番目のステートメントは、明示的な deny を使用します。明示的な deny を使用して、プリンシパルに付与された任意の allow アクセス許可を上書きできます。これにより、重要なリソースへのアクセスをロックダウンして、別のポリシーによって重要なリソースへのアクセスが誤って付与されることを防ぐことができます。

前述の例では、最初のステートメントによって prod- リソースへのフルアクセスが付与されていますが、2 番目のステートメントによって明示的にリソースへの書き込みアクセスが取り消され、prod- リソースへの読み取りアクセスのみが残されています。

タグを使用してアクセスを許可する

たとえば、お使いのアカウントの Tom という名前の特定のユーザーに対して、トリガー t2 へのアクセスを制限するとします。その他すべてのユーザー (Sam など) は、トリガー t1 にアクセスできます。トリガー t1 と t2 には、以下のプロパティがあります。

```
aws glue get-triggers
{
    "Triggers": [
        {
            "State": "CREATED",
            "Type": "SCHEDULED",
            "Name": "t1",
            "Actions": [
                {
                    "JobName": "j1"
                }
            ],
            "Schedule": "cron(0 0/1 * * ? *)"
        },
        {
            "State": "CREATED",
            "Type": "SCHEDULED",
            "Name": "t2",
            "Actions": [
                {
                    "JobName": "j1"
                }
            ],
            "Schedule": "cron(0 0/1 * * ? *)"
        }
    ]
}
```

AWS Glue 管理者は、トリガー t2にタグ値 Tom (aws:ResourceTag/Name": "Tom") を添付しました。AWS Glue 管理者は、タグに基づく条件ステートメントを使用して Tom に IAM ポリシーも提供しました。その結果、Tom はタグ値 Tom を持つリソースを対象とした AWS Glue オペレーションのみ使用できます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
         "Effect": "Allow",
         "Action": "glue:*",
         "Resource": "*",
         "Condition": {
             "StringEquals": {
                "aws:ResourceTag/Name": "Tom"
              }
          }
         }
    ]
}
```

Tom がトリガー t1 にアクセスしようとすると、アクセス拒否メッセージが返されます。一方、トリガー t2 は正常に取得できます。

```
aws glue get-trigger --name t1

An error occurred (AccessDeniedException) when calling the GetTrigger operation:
 User: Tom is not authorized to perform: glue:GetTrigger on resource: arn:aws:glue:us-
east-1:123456789012:trigger/t1

aws glue get-trigger --name t2
{
    "Trigger": {
        "State": "CREATED",
        "Type": "SCHEDULED",
        "Name": "t2",
        "Actions": [
            {
                "JobName": "j1"
            }
        ],
        "Schedule": "cron(0 0/1 * * ? *)"
```

```
    }
}
```

この API オペレーションではタグのフィルタリングがサポートされていないため、Tom は、複数の GetTriggers API オペレーションを使用してトリガーを一覧表示することができません。

Tom に GetTriggers へのアクセス権を与えるには、AWS Glue 管理者はアクセス許可を 2 つのセクションに分割するポリシーを作成します。1 つのセクションでは、GetTriggers API オペレーションを使用してすべてのトリガーにアクセスすることを Tom に許可します。2 番目のセクションでは、Tom という値でタグ付けされている API オペレーションにアクセスすることを Tom に許可します。このポリシーでは、Tom はトリガー t2 に対する GetTriggers と GetTrigger の両方のアクセスが許可されます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "glue:GetTriggers",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "glue:*",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/Name": "Tom"
                }
            }
        }
    ]
}
```

タグを使用してアクセスを拒否する

リソースポリシーのもう 1 つの方法は、リソースへのアクセスを明示的に拒否することです。

> **Important**
>
> 明示的な拒否ポリシーは、GetTriggers など複数の API オペレーションに使用することはできません。

次のポリシー例では、すべての AWS Glue ジョブ操作が許可されています。ただし、2 番目の Effect ステートメントでは、Team キーと Special 値がタグ付けされたジョブへのアクセスが明示的に拒否されています。

管理者が次のポリシーをアイデンティティにアタッチすると、このアイデンティティは、Team キーと Special 値がタグ付けされたジョブを除くすべてのジョブにアクセスできます。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "glue:*",
      "Resource": "arn:aws:glue:us-east-1:123456789012:job/*"
    },
    {
      "Effect": "Deny",
      "Action": "glue:*",
      "Resource": "arn:aws:glue:us-east-1:123456789012:job/*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/Team": "Special"
        }
      }
    }
  ]
}
```

リストとバッチ API オペレーションでタグを使用する

リソースポリシーを書き込む 3 つ目の方法としては、List API オペレーションを使用してリソースにアクセスし、タグ値のリソースを一覧表示することを許可します。次に、対応する Batch API オペレーションを使用して、特定のリソースの詳細にアクセスすることを許可します。この方法では、管理者は複数形の GetCrawlers、GetDevEndpoints、GetJobs、または GetTriggers API オ

ペレーションへのアクセスを許可する必要がありません。代わりに、次の API オペレーションを使用してリソースを一覧表示することを許可できます。

- ListCrawlers
- ListDevEndpoints
- ListJobs
- ListTriggers

また、以下の API オペレーションを使用して、個々のリソースに関する詳細を取得することを許可できます。

- BatchGetCrawlers
- BatchGetDevEndpoints
- BatchGetJobs
- BatchGetTriggers

管理者は、この方法を使用して、次の操作を実行できます。

1. クローラ、開発エンドポイント、ジョブ、およびトリガーにタグを追加します。
2. GetCrawlers、GetDevEndponts、GetJobs、および GetTriggers などの Get API オペレーションへのユーザーアクセスを拒否します。
3. アクセスできるタグ付きリソースをユーザーが見つけられるようにするには、ListCrawlers、ListDevEndponts、ListJobs、および ListTriggers などの List API オペレーションへのユーザーアクセスを許可します。
4. TagResource や UntagResource などの AWS Glue タグ付け API へのユーザーアクセスを拒否します。
5. BatchGetCrawlers、BatchGetDevEndponts、BatchGetJobs、および BatchGetTriggers などの BatchGet API オペレーションを使用して、リソースの詳細に対するアクセスをユーザーに許可します。

たとえば、ListCrawlers オペレーションを呼び出すときに、ユーザー名と一致するタグ値を指定します。結果として、指定したタグ値と一致するクローラのリストが返されます。指定されたタグを持つ各クローラーの詳細を取得するには、BatchGetCrawlers に名前のリストを提供します。

例えば、Tom に、Tom でタグ付けされたトリガーの詳細の取得のみを許可する場合、管理者は、Tom のトリガーにタグを追加し、すべてのユーザーに対して GetTriggers API オペレーションへのアクセスを拒否し、ListTriggers および BatchGetTriggers へのすべてのユーザーのアクセスを許可することができます。

次に示すのは、AWS Glue 管理者が Tom に付与するリソースポリシーです。ポリシーの最初のセクションでは、AWS Glue API オペレーションは GetTriggers に対して拒否されています。ポリシーの 2 番目のセクションでは、ListTriggers がすべてのリソースに対して許可されています。ただし、3 番目のセクションでは、Tom でタグ付けされたリソースは BatchGetTriggers アクセスでアクセスが許可されています。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": "glue:GetTriggers",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "glue:ListTriggers"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "glue:BatchGetTriggers"
            ],
            "Resource": [
                "*"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/Name": "Tom"
                }
            }
        }
```

```
    ]
}
```

前の例と同じトリガーを使用して、Tom はトリガー t2 にはアクセスできますが、トリガー t1 にはアクセスできません。次の例は、Tom が BatchGetTriggers で t1 と t2 にアクセスしようとしたときの結果を示しています。

```
aws glue batch-get-triggers --trigger-names t2
{
    "Triggers": {
        "State": "CREATED",
        "Type": "SCHEDULED",
        "Name": "t2",
        "Actions": [
            {
                "JobName": "j2"
            }
        ],
        "Schedule": "cron(0 0/1 * * ? *)"
    }
}

aws glue batch-get-triggers  --trigger-names t1

An error occurred (AccessDeniedException) when calling the BatchGetTriggers operation:
 No access to any requested resource.
```

次の例は、Tom が同じ BatchGetTriggers 呼び出しでトリガー t2 とトリガー t3 (存在しない) の両方にアクセスしようとしたときの結果を示しています。Tom は t2 をトリガーするアクセス権を持っており、それが存在するため、t2 のみが返されることに注意してください。Tom はトリガー t3 へのアクセスを許可されていますが、トリガー t3 は存在しないため、t3 はレスポンスの "TriggersNotFound": [] のリストで返されます。

```
aws glue batch-get-triggers --trigger-names t2 t3
{
    "Triggers": {
        "State": "CREATED",
        "Type": "SCHEDULED",
        "Name": "t2",
        "Actions": [
            {
                "JobName": "j2"
```

```
            }
        ],
        "TriggersNotFound": ["t3"],
        "Schedule": "cron(0 0/1 * * ? *)"
    }
}
```

条件キーまたはコンテキストキーを使用して設定を制御する

ジョブを作成および更新する権限を付与する場合は、条件キーまたはコンテキストキーを使用できます。ここでは、キーについて説明します。

- 条件キーを使って設定を制御するポリシーを制御する
- 条件キーを使って設定を制御するポリシーを制御する

条件キーを使って設定を制御するポリシーを制御する

AWS Glue は glue:VpcIds、glue:SubnetIds、glue:SecurityGroupIds の3 つの IAM 条件キーを提供します。ジョブを作成および更新する権限を付与する場合は、IAM ポリシーで条件キーを使用できます。この設定を使用して、目的の VPC 環境外で実行するジョブまたはセッションが作成されない (または更新されない) ようにすることができます。VPC 設定の情報は、CreateJob リクエストから直接入力されるのではなく、AWS Glue 接続を指すジョブの「接続」フィールドから推測されます。

使用例

希望する VpcId「vpc-id1234」、SubnetIds、SecurityGroupIds を持つ「traffic-monitored-connection」という名前の AWS Glue ネットワーク型接続を作成します。

IAM ポリシーのアクション CreateJob および UpdateJob の条件キーを指定します。

```
{
  "Effect": "Allow",
  "Action": [
    "glue:CreateJob",
    "glue:UpdateJob"
  ],
  "Resource": [
    "*"
  ],
  "Condition": {
    "ForAnyValue:StringLike": {
```

```
      "glue:VpcIds": [
        "vpc-id1234"
      ]
    }
  }
}
```

同様の IAM ポリシーを作成して、接続情報を指定しない AWS Glue ジョブの作成を禁止する事ができます。

VPC でのセッションの制限

指定した VPC 内で実行するように作成済みセッションを強制するには、glue:vpc-id が vpc-<123> と等しくないという条件付きで glue:CreateSession アクションに Deny effect を追加することで、ロールのアクセス許可を制限します。例：

```
"Effect": "Deny",
"Action": [
    "glue:CreateSession"
 ],
"Condition": {
    "StringNotEquals" : {"glue:VpcIds" : ["vpc-123"]}
}
```

また、glue:vpc-id が NULL であるという条件付きで glue:CreateSession アクションに Deny 効果を追加しても、VPC 内で実行するように作成済みセッションを強制できます。例：

```
{
    "Effect": "Deny",
    "Action": [
        "glue:CreateSession"
    ],
      "Condition": {
        "Null": {"glue:VpcIds": true}
    }
},
{
    "Effect": "Allow",
    "Action": [
        "glue:CreateSession"
    ],
```

```
    "Resource": ["*"]
}
```

条件キーを使って設定を制御するポリシーを制御する

AWS Glue は、glue:CredentialIssuingService= glue.amazonaws.com でジョブや開発者のエンドポイントから利用できるように、各ロールセッションにコンテキストキー(AWS Glue)を提供します。これにより、AWS Glue スクリプトが実行するアクションに対し、セキュリティ制御を実装できます。AWS Glue は、別のコンテキストキー (glue:RoleAssumedBy=glue.amazonaws.com) を各ロールセッションに提供し、そこで AWS Glue が顧客に代わって別のサービス AWS を (ジョブ/デバイスのエンドポイントではなく AWS Glue サービスにより直接) 呼び出します。

使用例

IAM ポリシーで条件付きアクセス許可を指定し、これを AWS Glue ジョブが使用するロールにアタッチします。これで、特定のアクションが、ロールセッションが AWS Glue ジョブランタイム環境で使用されているかどうかに基づいて、許可/拒否されます。

```
{
    "Effect": "Allow",
    "Action": "s3:GetObject",
    "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*",
    "Condition": {
        "StringEquals": {
            "glue:CredentialIssuingService": "glue.amazonaws.com"
        }
    }
}
```

ID によるデータプレビューセッションの作成を拒否する

このセクションには、ID によるデータプレビューセッションの作成を拒否するために使用される IAM ポリシーの例が含まれています。このポリシーを ID にアタッチします。ID は、データプレビューセッションの実行中に使用するロールとは別のものです。

```
{
    "Sid": "DatapreviewDeny",
    "Effect": "Deny",
     "Action": [
           "glue:CreateSession"
```

```
    ],
    "Resource": [
        "arn:aws:glue:*:*:session/glue-studio-datapreview*"
     ]
}
```

AWS Glue のリソースベースのポリシーの例

このセクションでは、リソースベースのポリシーの例を紹介します。これには、クロスアカウントアクセスを許可するポリシーが含まれます。

次の例では、AWS Command Line Interface (AWS CLI) を使用して AWS Glue サービス API オペレーションを操作します。AWS Glue コンソール、または AWS SDK のいずれかを使用して、これと同じオペレーションを実行できます。

> **Important**
>
> AWS Glue リソースポリシーを変更することで、アカウント内の既存の AWS Glue ユーザーのアクセス許可を誤って取り消したり、予期しない障害が発生したりする場合があります。これらの例は開発やテストアカウントでのみ試して、変更を行う前に既存のワークフローを壊さないことを確認します。

トピック

- AWS Glue でリソースベースのポリシーを使用する際の考慮事項
- 同じアカウント内のアクセスをコントロールするリソースポリシーの使用

AWS Glue でリソースベースのポリシーを使用する際の考慮事項

> **Note**
>
> IAM ポリシーと AWS Glue リソースポリシーが、それぞれ反映されるまでには数秒かかります。新しいポリシーをアタッチした後、新しいポリシーがシステム全体に反映されるまで古いポリシーが有効なままの場合があります。

JSON 形式で記述されたポリシードキュメントを使用して、リソースポリシーを作成または変更します。ポリシー構文は、以下の例外を除き、アイデンティティベースの IAM ポリシーと同じです (IAM JSON ポリシーリファレンスを参照のこと)。

- ポリシーステートメントごとに "Principal" または "NotPrincipal" ブロックが必要です。
- "Principal" または "NotPrincipal" では、有効な既存のプリンシパルを特定する必要があります。ワイルドカードパターン (arn:aws:iam::*account-id*:user/* など) は使用できません。
- ポリシー内の "Resource" ブロックでは、すべてのリソース ARN が、次の正規表現構文と一致している必要があります (最初の %s は *region*、2 番目の %s は *account-id*)。

```
*arn:aws:glue:%s:%s:(\*|[a-zA-Z\*]+\/?.*)
```

  たとえば、arn:aws:glue:us-west-2:*account-id*:* と arn:aws:glue:us-west-2:*account-id*:database/default はいずれも許可されますが、* は許可されません。
- アイデンティティベースのポリシーとは異なり、AWS Glue リソースポリシーには、このポリシーがアタッチされているカタログに属するリソースの、Amazon リソースネーム (ARN) のみを含めることができます。このような ARN は常に arn:aws:glue: で始まります。
- ポリシーは、それを作成しているアイデンティティを、それ以降のポリシーの作成または変更から排除することはできません。
- リソースポリシーの JSON ドキュメントのサイズは 10 KB を超えることはできません。

同じアカウント内のアクセスをコントロールするリソースポリシーの使用

この例では、Account A の管理者ユーザーが、Account A 内の IAM ユーザー Alice にカタログへの完全なアクセス許可を付与するリソースポリシーを作成します。Alice に添付されている IAM ポリシーはありません。

これを行うには、管理者ユーザーは次の AWS CLI コマンドを実行します。

```
# Run as admin of Account A
$ aws glue put-resource-policy --profile administrator-name --region us-west-2 --
policy-in-json '{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Principal": {
        "AWS": [
          "arn:aws:iam::account-A-id:user/Alice"
        ]
      },
      "Effect": "Allow",
```

```
      "Action": [
        "glue:*"
      ],
      "Resource": [
        "arn:aws:glue:us-west-2:account-A-id:*"
      ]
    }
  ]
}'
```

AWS CLI コマンドの一部として JSON ポリシードキュメントを入力する代わりに、ポリシードキュメントをファイルに保存して、file:// でプレフィックスが付けられた AWS CLI コマンドのファイルパスをリファレンスできます。その方法の例は次のとおりです。

```
$ echo '{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Principal": {
        "AWS": [
          "arn:aws:iam::account-A-id:user/Alice"
        ]
      },
      "Effect": "Allow",
      "Action": [
        "glue:*"
      ],
      "Resource": [
        "arn:aws:glue:us-west-2:account-A-id:*"
      ]
    }
  ]
}' > /temp/policy.json

$ aws glue put-resource-policy --profile admin1 \
    --region us-west-2 --policy-in-json file:///temp/policy.json
```

このリソースポリシーが反映されると、Alice は Account A のすべての AWS Glue リソースにアクセスできます。

```
# Run as user Alice
$ aws glue create-database --profile alice --region us-west-2 --database-input '{
```

```
    "Name": "new_database",
    "Description": "A new database created by Alice",
    "LocationUri": "s3://amzn-s3-demo-bucket"
}'

$ aws glue get-table --profile alice --region us-west-2 --database-name "default" --
table-name "tbl1"}
```

Alice の `get-table` 呼び出しに応じて、AWS Glue サービスは次を返します。

```
{
  "Table": {
    "Name": "tbl1",
    "PartitionKeys": [],
    "StorageDescriptor": {
        ......
    },
    ......
  }
}
```

AWS Glue の AWS マネージドポリシーの付与

AWS マネージドポリシーは、AWS が作成および管理するスタンドアロンポリシーです。AWS マネージドポリシーは、多くの一般的なユースケースに対してアクセス許可を提供するように設計されているため、ユーザー、グループ、ロールへのアクセス権の割り当てを開始できます。

AWS マネージドポリシーは、ご利用の特定のユースケースに対して最小特権のアクセス許可を付与しない場合があることにご注意ください。これは、すべての AWS ユーザーが使用できるようになるのを避けるためです。ユースケースに固有のカスタマー管理ポリシーを定義して、アクセス許可を絞り込むことをお勧めします。

AWS マネージドポリシーで定義されたアクセス許可は変更できません。AWS が AWS マネージドポリシーに定義されている権限を更新すると、更新はポリシーがアタッチされているすべてのプリンシパルアイデンティティ (ユーザー、グループ、ロール) に影響します。新しい AWS のサービス を起動するか、既存のサービスで新しい API オペレーションが使用可能になると、AWS が AWS マネージドポリシーを更新する可能性が最も高くなります。

詳細については、「IAM ユーザーガイド」の「AWS マネージドポリシー」を参照してください。

AWS Glue の AWS マネージド (事前定義) ポリシー

AWS は、AWS によって作成され管理されるスタンドアロンの IAM ポリシーを提供することで、多くの一般的ユースケースに対応します。これらの AWS 管理ポリシーは、一般的ユースケースに必要なアクセス権限を付与することで、どの権限が必要なのかをユーザーが調査する必要をなくすことができます。詳細については、IAM ユーザーガイドの AWS管理ポリシーを参照してください。

次の AWS マネージドポリシーはアカウントのアイデンティティにアタッチすることができ、AWS Glue に固有で、ユースケースのシナリオ別にグループ化されています。

- AWSGlueConsoleFullAccess — ポリシーがアタッチされているアイデンティティが AWS Management Console を使用するときは、AWS Glue リソースへのフルアクセスを許可します。このポリシーで指定されたリソースの命名規則に従った場合、ユーザーは完全なコンソール機能を使用できます。このポリシーは、通常 AWS Glue コンソールのユーザーにアタッチされています。
- AWSGlueServiceRole – さまざまな AWS Glue プロセスを実行するために必要なリソースへのアクセス権をユーザーに代わって付与します。これらのリソースには AWS Glue 、Amazon S3、IAM、CloudWatch Logs、および Amazon EC2 が含まれます。このポリシーで指定されたリソースの命名規則に従った場合、AWS Glue プロセスは必要なアクセス権限を使用できます。このポリシーは、通常、クローラ、ジョブ、開発エンドポイントを定義するときに指定されたロールにアタッチされます。
- AwsGlueSessionUserRestrictedServiceRole — セッションを除くすべての AWS Glue リソースへのフルアクセスを許可します。ユーザーに、自分に関連付けられたインタラクティブセッションのみ、作成と使用を許可します。このポリシーには、AWS Glue が他の AWS サービスで AWS Glue リソースを管理する際に必要になる他のアクセス権限が含まれています。このポリシーでは、その他の AWS Glue リソースの AWS サービスへのタグの追加も許可されます。

  > **Note**
  >
  > セキュリティ上のメリットを最大限に引き出すには、`AWSGlueServiceRole`、`AWSGlueConsoleFullAccess`、または `AWSGlueConsoleSageMakerNotebookFullAccess` のポリシーが割り当てられたユーザーにはこのポリシーを付与しないでください。

- AwsGlueSessionUserRestrictedPolicy - 使用者の AWS ユーザー ID と一致するタグキー「所有者」および値が提供されている場合のみ、`CreateSession` API オペレーションを使って AWS Glue インタラクティブセッションを作成するためのアクセスを提供します。この ID ポリシーは、`CreateSession` API オペレーションを呼び出す IAM ユーザーにアタッチされます。またこのポリシーは、使用者が、自分の AWS Glue ユーザー ID と一致する「所有者」タグと値とで作成

された、AWS インタラクティブセッションリソースを操作することも許可します。このポリシーは、セッションが作成された後に、AWS Glue セッションリソースから「所有者」タグを変更または削除する許可を拒否します。

> **Note**
>
> セキュリティ上のメリットを最大限に引き出すには、`AWSGlueServiceRole`、`AWSGlueConsoleFullAccess`、または `AWSGlueConsoleSageMakerNotebookFullAccess` のポリシーが割り当てられたユーザーにはこのポリシーを付与しないでください。

- AWSGlueSessionUserRestrictedNotebookServiceRole — AWS Glue Studio ノートブックセッションへの十分なアクセスを提供し、特定の AWS Glue インタラクティブセッションリソースのやり取りを行えるようにします。これらは、ノートブックを作成したプリンシパルの AWS ユーザー ID (IAM ユーザーまたはロール) と一致する「所有者」タグ値で作成されたリソースです。これらのタグの詳細については、IAM ユーザーガイドにあるプリンシパルキーの値の表を参照してください。

  このサービスロールポリシーは、ノートブック内でマジックコマンドにより指定されたロールと、`CreateSession` API オペレーションにロールとして渡されたロールにアタッチされます。またこのポリシーは、タグキー「所有者」と値とがプリンシパルの AWS ユーザー ID に一致する場合のみ、プリンシパルが AWS Glue Studio ノートブックインターフェースから AWS Glue インタラクティブセッションを作成することを許可します。このポリシーは、セッションが作成された後に、AWS Glue セッションリソースから「所有者」タグを変更または削除する許可を拒否します。このポリシーには、Amazon S3 バケットからの書き込みと読み取り、CloudWatch ログの書き込み、AWS Glue が使用する Amazon EC2 リソースのタグの作成と削除、のためのアクセス権限も含まれます。

  > **Note**
  >
  > セキュリティ上のメリットを最大限に引き出すには、`AWSGlueServiceRole`、`AWSGlueConsoleFullAccess`、または `AWSGlueConsoleSageMakerNotebookFullAccess` のポリシーが割り当てられたユーザーにはこのポリシーを付与しないでください。

- AwsGlueSessionUserRestrictedNotebookPolicy - ノートブックを作成するプリンシパルの AWS ユーザー ID (IAM ユーザーまたはロール) と一致する、タグキー「所有者」と値とがある場合にのみ、AWS Glue Studio ノートブックインターフェースから AWS Glue インタラクティブセッ

ションを作成するアクセス権限を付与します。これらのタグの詳細については、IAM ユーザーガイドにあるプリンシパルキーの値の表を参照してください。

このポリシーは、AWS Glue Studio ノートブックインターフェイスからセッションを作成するプリンシパル (IAM ユーザーまたはロール) にアタッチされます。このポリシーでは、特定の AWS Glue インタラクティブセッションリソースのやり取りを行うための、AWS Glue Studio ノートブックへの十分なアクセス権限も付与されます。これらは、プリンシパルの AWS ユーザー ID と一致する「所有者」タグ値で作成されたリソースです。このポリシーは、セッションが作成された後に、AWS Glue セッションリソースから「所有者」タグを変更または削除する許可を拒否します。

- AWSGlueServiceNotebookRole — AWS Glue Studio ノートブックで開始された AWS Glue セッションへのアクセスを許可します。このポリシーでは、すべてのセッションのセッション情報の一覧表示と取得を許可しますが、ユーザーが行えるのは、自分の AWS ユーザー ID でタグ付けされたセッションの作成と使用のみです。このポリシーは、ユーザーの AWS ID でタグ付けされた AWS Glue セッションリソースから、「所有者」タグを変更または削除しようとするアクセス権限は拒否します。

  このポリシーを、AWS のノートブックインターフェイスを使用してジョブを作成する AWS Glue Studio ユーザーに割り当ててください。

- AWSGlueConsoleSageMakerNotebookFullAccess - ポリシーがアタッチされた ID が AWS Management Consoleを使用するときに、AWS Glue と SageMaker AI リソースへの完全なアクセスを付与します。このポリシーで指定されたリソースの命名規則に従った場合、ユーザーは完全なコンソール機能を使用できます。このポリシーは、通常 SageMaker AI ノートブックを管理する AWS Glue コンソールのユーザーにアタッチされています。

- AWSGlueSchemaRegistryFullAccess – ポリシーがアタッチされた ID が AWS Management Console または AWS CLI を使用するときに、AWS Glue Schema Registry リソースへの完全なアクセスを付与します。このポリシーで指定されたリソースの命名規則に従った場合、ユーザーは完全なコンソール機能を使用できます。このポリシーは通常、AWS Glue Schema Registry を管理する AWS Glue コンソールまたは AWS CLI のユーザーにアタッチされます。

- AWSGlueSchemaRegistryReadonlyAccess – ポリシーがアタッチされた ID が AWS Management Console または AWS CLI を使用するときに、AWS Glue Schema Registry リソースへの読み取り専用アクセスを許可します。このポリシーで指定されたリソースの命名規則に従った場合、ユーザーは完全なコンソール機能を使用できます。このポリシーは通常、AWS Glue Schema Registry を使用する AWS Glue コンソールまたは AWS CLI のユーザーにアタッチされます。

> **Note**
>
> IAM コンソールにサインインし、特定のポリシーを検索することで、これらの許可ポリシーを確認できます。

これに加え、独自のカスタム IAM ポリシーを作成して、AWS Glue アクションとリソースにアクセス許可を付与することもできます。これらのカスタムポリシーは、それらのアクセス許可が必要な IAM ユーザーまたはグループにアタッチできます。

カスタム IAM ロールの使用中に VPC 設定を使用した接続を作成するには、次の VPC アクセスのアクションが必要です。

- secretsmanager:GetSecretValue
- secretsmanager:PutSecretValue
- secretsmanager:DescribeSecret
- ec2:CreateNetworkInterface
- ec2:DeleteNetworkInterface
- ec2:DescribeNetworkInterfaces
- ec2:DescribeSubnets

AWS Glueでの AWS マネージドポリシーの更新

AWS Glue の AWS 管理ポリシーの更新に関する詳細を、このサービスがこれらの変更の追跡を開始した以降の分について表示します。このページの変更を自動通知するには、AWS Glue の [Document history] (ドキュメントの履歴) ページで RSS フィードをサブスクライブします。

変更	説明	日付
AwsGlueSessionUserRestrictedNotebookPolicy – 既存のポリシーのマイナーアップデート。	所有者タグキーに `glue:TagResource` アクションの許可を追加します。所有者タグキーを持つセッションの tag-on-create のサポートに必要です。	2024 年 8 月 30 日

変更	説明	日付
AwsGlueSessionUserRestrictedNotebookServiceRole – 既存のポリシーのマイナーアップデート。	所有者タグキーに glue:TagResource アクションの許可を追加します。所有者タグキーを持つセッションの tag-on-create のサポートに必要です。	2024 年 8 月 30 日
AwsGlueSessionUserRestrictedPolicy – 既存のポリシーのマイナーアップデート。	所有者タグキーに glue:TagResource アクションの許可を追加します。所有者タグキーを持つセッションの tag-on-create のサポートに必要です。	2024 年 8 月 5 日
AwsGlueSessionUserRestrictedServiceRole – 既存のポリシーのマイナーアップデート。	所有者タグキーに glue:TagResource アクションの許可を追加します。所有者タグキーを持つセッションの tag-on-create のサポートに必要です。	2024 年 8 月 5 日
AwsGlueSessionUserRestrictedPolicy – 既存のポリシーのマイナーアップデート。	glue:StartCompletion と glue:GetCompletion をポリシーに追加します。AWS Glue の Amazon Q データ統合に必要です。	2024 年 4 月 30 日
AwsGlueSessionUserRestrictedNotebookServiceRole – 既存のポリシーのマイナーアップデート。	glue:StartCompletion と glue:GetCompletion をポリシーに追加します。AWS Glue の Amazon Q データ統合に必要です。	2024 年 4 月 30 日

変更	説明	日付
AwsGlueSessionUserRestrictedServiceRole – 既存のポリシーのマイナーアップデート。	`glue:StartCompletion` と `glue:GetCompletion` をポリシーに追加します。AWS Glue の Amazon Q データ統合に必要です。	2024 年 4 月 30 日
AWSGlueServiceNotebookRole – 既存のポリシーのマイナーアップデート。	`glue:StartCompletion` と `glue:GetCompletion` をポリシーに追加します。AWS Glue の Amazon Q データ統合に必要です。	2024 年 1 月 30 日
AwsGlueSessionUserRestrictedNotebookPolicy – 既存のポリシーのマイナーアップデート。	`glue:StartCompletion` と `glue:GetCompletion` をポリシーに追加します。AWS Glue の Amazon Q データ統合に必要です。	2023 年 11 月 29 日
AWSGlueServiceNotebookRole – -既存のポリシーのマイナーアップデート。	`codewhisperer:GenerateRecommendations` をポリシーに追加します。AWS Glue が CodeWhisperer の推奨事項を生成する新機能に必要です。	2023 年 10 月 9 日
AWSGlueServiceRole - 既存のポリシーのマイナーアップデート。	AWS Glue の口グ記録をより適切に反映できるように CloudWatch のアクセス許可の範囲を厳しくしてください。	2023 年 8 月 4 日

変更	説明	日付
AWSGlueConsoleFullAccess – 既存のポリシーのマイナーアップデート。	`databrew` レシピのリストと説明のアクセス許可をポリシーに追加します。AWS Glue がレシピにアクセスできる新機能への完全な管理アクセスを提供するために必要です。	2023 年 5 月 9 日
AWSGlueConsoleFullAccess – 既存のポリシーのマイナーアップデート。	`cloudformation:ListStacks` をポリシーに追加します。AWS CloudFormation 認可要件が変更された後も、既存の機能が維持されます。	2023 年 3 月 28 日
インタラクティブセッション機能で新たに追加されたマネージドポリシー: • AwsGlueSessionUserRestrictedServiceRole • AwsGlueSessionUserRestrictedPolicy • AwsGlueSessionUserRestrictedNotebookServiceRole • AwsGlueSessionUserRestrictedNotebookPolicy	これらのポリシーは、AWS Glue Studio のインタラクティブセッションとノートブックのセキュリティを強化するように設計されています。これらのポリシーは、所有者だけがアクセスできるよう、`CreateSession` API オペレーションへのアクセスを制限します。	2021 年 11 月 30 日

変更	説明	日付
AWSGlueConsoleSageMakerNotebookFullAccess – 既存のポリシーの更新	`arn:aws:s3:::aws-glue-*/*` がスクリプトと一時ファイルを保存する際に使用する、Amazon S3 バケットに対する読み込み/書き込みのアクセス許可を付与するアクションで、冗長していたリソース ARN (AWS Glue) を削除しました。 `"StringEquals"` を `"ForAnyValue:StringLike"` に変更し、順序に誤りがあった (各所の) `"Effect": "Allow"` 行を `"Action":` 行の前に移動することにより、構文の問題を修正しました。	2021 年 7 月 15 日
AWSGlueConsoleFullAccess – 既存のポリシーの更新	`arn:aws:s3:::aws-glue-*/*` がスクリプトと一時ファイルを保存する際に使用する、Amazon S3 バケットに対する読み込み/書き込みのアクセス許可を付与するアクションで、冗長していたリソース ARN (AWS Glue) を削除しました。	2021 年 7 月 15 日
AWS Glue は変更の追跡を開始しました	AWS Glue は AWS マネージドポリシーの変更の追跡を開始しました。	2021 年 6 月 10 日

AWS Glue リソース ARN の指定

AWS Glue では、AWS Identity and Access Management (IAM) ポリシーを使用してリソースへのアクセスを制御できます。ポリシーで Amazon リソースネーム (ARN) を使用して、ポリシーを適用するリソースを識別します。AWS Glue のすべてのリソースで ARN がサポートされているわけではありません。

トピック

- データカタログの ARN
- AWS Glue にあるカタログ以外のオブジェクトの ARN
- AWS Glue のカタログ以外の単一の API オペレーション
- 複数の項目を取得する AWS Glue のカタログ以外の API オペレーションのアクセスコントロール
- AWS Glue のカタログ以外のバッチ取得 API オペレーションのアクセス制御

データカタログの ARN

Data Catalog リソースには、`catalog` をルートとする階層構造があります。

```
arn:aws:glue:region:account-id:catalog
```

各 AWS アカウントは、カタログ ID として 12 桁のアカウント ID を使用する単一の Data Catalog を、AWS リージョン内に持っています。リソースには、次の表に示すように、一意の ARN が関連付けられています。

リソースタイプ:	ARN 形式
カタログ	`arn:aws:glue: region:account-id :catalog` 例: `arn:aws:glue:us-east-1:123456789012:catalog`
データベース	`arn:aws:glue: region:account-id :database/ database name` 例: `arn:aws:glue:us-east-1:123456789012:database/db1` 。
[テーブル]	`arn:aws:glue: region:account-id :table/database name/table name`

リソースタイプ:	ARN 形式
	例: arn:aws:glue:us-east-1:123456789012:table/db1/tbl1。
フェデレーティッド S3 テーブルカタログ (すべてのテーブルバケット)	arn:aws:glue: *region*:*account-id* :catalog/s3tablescatalog 例: arn:aws:glue:us-east-1:123456789012:catalog/s3tablescatalog
フェデレーティッド S3 テーブルバケットカタログ (子カタログ)	arn:aws:glue: *region*:*account-id* :catalog/s3tablescatalog/ *bucket name* 例: arn:aws:glue:us-east-1:123456789012:catalog/s3tablescatalog/amzn-s3-demo-bucket1
フェデレーティッド S3 テーブルデータベース	arn:aws:glue: *region*:*account-id* :database/s3tablescatalog/ *child catalog name*/*database name* 例: arn:aws:glue:us-east-1:123456789012:database/s3tablescatalog/amzn-s3-demo-bucket1/nsdb1
フェデレーティッド S3 テーブル	arn:aws:glue: *region*:*account-id* :table/s3tablescatalog/ *child catalog name*/*database name*/*table name* 例: arn:aws:glue:us-east-1:123456789012:table/s3tablescatalog/amzn-s3-demo-bucket1/nsdb1/s3tbl1
フェデレーティッド S3 テーブルカタログ (Lake Formation に登録された単一のテーブルバケット)	arn:aws:glue: *region*:*account-id* :catalog/*catalog name* 例: arn:aws:glue:us-east-1:123456789012:catalog/amzn-s3-demo-bucket1
フェデレーティッド S3 テーブルデータベース	arn:aws:glue: *region*:*account-id* :catalog/*catalog name*/*database name* 例: arn:aws:glue:us-east-1:123456789012:database/amzn-s3-demo-bucket1/nsdb1

リソースタイプ:	ARN 形式
フェデレーティッド S3 テーブル	arn:aws:glue: *region*:*account-id* :catalog/*catalog name*/*database name*/*table name* 例: arn:aws:glue:us-east-1:123456789012:table/amzn-s3-demo-bucket1/ *nsdb1*/s3tbl1
フェデレーティッドカタログまたはマネージドカタログ (マルチカタログの最上位カタログ)	arn:aws:glue: *region*:*account-id* :catalog/ *top-level catalog name* 例: arn:aws:glue:us-east-1:123456789012:catalog/nscatalog マネージドカタログの ARN 形式は、同じ構造に従います。
フェデレーティッドマルチレベルカタログ (マルチレベルカタログ内の子カタログ)	arn:aws:glue: *region*:*account-id* :catalog/ *top-level catalog name*/*child catalog name* 例: arn:aws:glue:us-east-1:123456789012:catalog/nscatalog/dbcatalog
フェデレーティッドデータベース	arn:aws:glue: *region*:*account-id* :database/ *name space catalog name*/*child catalog name*/*database name* 例: arn:aws:glue:us-east-1:123456789012:database/nscatalog/dbcatalog/schemadb
フェデレーティッドテーブル	arn:aws:glue: *region*:*account-id* :table/*name space catalog name*/*child catalog name*/*database name*/*table name* 例: arn:aws:glue:us-east-1:123456789012:table/nscatalog/dbcatalog/schemadb/rstbl1
カタログリンクコンテナ	arn:aws:glue: *region*:*account-id* :catalog/*link container name* 例: arn:aws:glue:glue:us-east-1:123456789012:catalog /linkcontainer-example

リソースタイプ:	ARN 形式
データベース	arn:aws:glue: *region*:*account-id* :catalog/*linkcontainer name*/*database name* 例: arn:aws:glue:glue:us-east-1:123456789012:database /linkcontainer-example/link-db
ユーザー定義関数	arn:aws:glue: *region*:*account-id* :userDefinedFunction/ *database name*/*user-defined function name* 例: arn:aws:glue:us-east-1:123456789012:userDefinedFunction/db1/func1 。
接続	arn:aws:glue: *region*:*account-id* :connection/ *connection name* 例: arn:aws:glue:us-east-1:123456789012:connection/connection1 。
インタラクティブセッション	arn:aws:glue: *region*:*account-id* :session/ *interactive session id* 例: arn:aws:glue:us-east-1:123456789012:session/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111 。

きめ細かなアクセスコントロールを有効にするには、IAM ポリシーとリソースポリシーでこれらの ARN を使用して、特定のリソースへのアクセス許可を付与または拒否します。ポリシーではワイルドカードを使用できます。たとえば、次の ARN はデータベース default のすべてのテーブルと一致します。

```
arn:aws:glue:us-east-1:123456789012:table/default/*
```

> **Important**
>
> Data Catalog リソースで実行されるすべてのオペレーションには、リソースとその継承元すべてに対するアクセス許可が必要です。たとえば、テーブルのパーティションを作成するには、テーブル、データベース、およびテーブルが存在するカタログに対するアクセス

許可が必要です。以下に、Data Catalog のデータベース PrivateDatabase 内のテーブル PrivateTable で、パーティションを作成するのに必要なアクセス許可の例を示します。

```
{
   "Sid": "GrantCreatePartitions",
   "Effect": "Allow",
   "Action": [
       "glue:BatchCreatePartitions"
   ],
   "Resource": [
       "arn:aws:glue:us-east-1:123456789012:table/PrivateDatabase/PrivateTable",
       "arn:aws:glue:us-east-1:123456789012:database/PrivateDatabase",
       "arn:aws:glue:us-east-1:123456789012:catalog"
   ]
}
```

リソースとそのすべての祖先のアクセス許可に加えて、すべての削除オペレーションにはそのリソースのすべての子のアクセス許可が必要です。たとえば、データベースを削除するには、データベースおよびデータベースが存在するカタログに加えて、データベース内のすべてのテーブルおよびユーザー定義関数に対するアクセス許可が必要です。以下は、Data Catalog でデータベース PrivateDatabase を削除するのに必要なアクセス許可の例です。

```
{
   "Sid": "GrantDeleteDatabase",
   "Effect": "Allow",
   "Action": [
       "glue:DeleteDatabase"
   ],
   "Resource": [
       "arn:aws:glue:us-east-1:123456789012:table/PrivateDatabase/*",
       "arn:aws:glue:us-east-1:123456789012:userDefinedFunction/PrivateDatabase/
*",
       "arn:aws:glue:us-east-1:123456789012:database/PrivateDatabase",
       "arn:aws:glue:us-east-1:123456789012:catalog"
   ]
}
```

要約すると、Data Catalog リソースに対するアクションは、以下のアクセス許可のルールに従います。

- カタログに対するアクションには、カタログのみに対するアクセス許可が必要です。

- データベースに対するアクションには、データベースおよびカタログに対するアクセス許可が必要です。
- データベースに対する削除アクションには、データベースとカタログに対するアクセス許可に加えて、データベース内のすべてのテーブルおよびユーザー定義関数に対するアクセス許可も必要です。
- テーブル、パーティション、またはテーブルバージョンに対するアクションには、テーブル、データベース、およびカタログに対するアクセス許可が必要です。
- ユーザー定義関数に対するアクションには、ユーザー定義関数、データベース、およびカタログに対するアクセス許可が必要です。
- 接続に対するアクションには、接続およびカタログに対するアクセス許可が必要です。

AWS Glue にあるカタログ以外のオブジェクトの ARN

一部の AWS Glue リソースによって、ARN を使用してアクセスをコントロールするリソースレベルの許可が付与されます。IAM ポリシーでこれらの ARN を使用して、きめ細かなアクセスコントロールを有効にします。次の表は、リソース ARN を含むことができるリソースの一覧です。

リソースタイプ:	ARN 形式
Crawler	`arn:aws:glue:` *`region`*`:`*`account-id`* `:crawler/` *`crawler-name`* 例: `arn:aws:glue:us-east-1:123456789012:crawler/mycrawler` 。
ジョブ	`arn:aws:glue:` *`region`*`:`*`account-id`* `:job/`*`job-name`* 例: `arn:aws:glue:us-east-1:123456789012:job/testjob` 。
Trigger トリガー)	`arn:aws:glue:` *`region`*`:`*`account-id`* `:trigger/` *`trigger-name`* 例: `arn:aws:glue:us-east-1:123456789012:trigger/sampletrigger`

リソースタイプ:	ARN 形式
開発エンドポイント	arn:aws:glue: *region*:*account-id* :devEndpoint/ *development-endpoint-name* 例: arn:aws:glue:us-east-1:123456789012:devEndpoint/temporarydevendpoint
機械学習変換	arn:aws:glue: *region*:*account-id* :mlTransform/ *transform-id* 例: arn:aws:glue:us-east-1:123456789012:mlTransform/tfm-1234567890

AWS Glue のカタログ以外の単一の API オペレーション

AWS Glue のカタログ以外の単一の API オペレーションは、単一の項目 (開発エンドポイント) で動作します。例は、GetDevEndpoint, CreateUpdateDevEndpoint および UpdateDevEndpoint です。これらのオペレーションでは、ポリシーは API 名を "action" ブロックに、リソース ARN を "resource" ブロックに置く必要があります。

GetDevEndpoint オペレーションの呼び出しをユーザーに許可するとします。次のポリシーは、myDevEndpoint-1 という名前のエンドポイントに必要最小限のアクセス許可を付与します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "MinimumPermissions",
            "Effect": "Allow",
            "Action": "glue:GetDevEndpoint",
            "Resource": "arn:aws:glue:us-east-1:123456789012:devEndpoint/
myDevEndpoint-1"
        }
    ]
}
```

次のポリシーは、ワイルドカード (*) を使用して myDevEndpoint- と一致するリソースへのアクセスを UpdateDevEndpoint に許可します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "PermissionWithWildcard",
            "Effect": "Allow",
            "Action": "glue:UpdateDevEndpoint",
            "Resource": "arn:aws:glue:us-east-1:123456789012:devEndpoint/myDevEndpoint-
*"
        }
    ]
}
```

次の例のように 2 つのポリシーを組み合わせることができます。名前が A で始まるすべての開発エンドポイントに対して EntityNotFoundException が表示される場合があります。一方、その他の開発エンドポイントにアクセスしようとすると、アクセス拒否エラーが表示されます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CombinedPermissions",
            "Effect": "Allow",
            "Action": [
                "glue:UpdateDevEndpoint",
                "glue:GetDevEndpoint"
            ],
            "Resource": "arn:aws:glue:us-east-1:123456789012:devEndpoint/A*"
        }
    ]
}
```

複数の項目を取得する AWS Glue のカタログ以外の API オペレーションのアクセスコントロール

一部の AWS Glue API オペレーション (GetDevEndpoints など) は、複数の項目 (複数の開発エンドポイントなど) を取得します。このオペレーションでは、特定の ARN ではなく、ワイルドカード (*) リソースのみを指定できます。

たとえば、GetDevEndpoints をポリシーに含めるには、リソースの範囲をワイルドカード (*) に限定する必要があります。単一のオペレーション (GetDevEndpoint、CreateDevEndpoint、および DeleteDevendpoint) の範囲も、例のすべての (*) リソースに限定されます。

```
{
            "Sid": "PluralAPIIncluded",
            "Effect": "Allow",
            "Action": [
                "glue:GetDevEndpoints",
                "glue:GetDevEndpoint",
                "glue:CreateDevEndpoint",
                "glue:UpdateDevEndpoint"
            ],
            "Resource": [
                "*"
            ]
}
```

AWS Glue のカタログ以外のバッチ取得 API オペレーションのアクセス制御

一部の AWS Glue API オペレーション (BatchGetDevEndpoints など) は、複数の項目 (複数の開発エンドポイントなど) を取得します。このオペレーションでは、アクセス可能なリソースの範囲を制限するために ARN を指定できます。

たとえば、特定の開発エンドポイントへのアクセスを許可するには、そのリソース ARN を指定して BatchGetDevEndpoints をポリシーに含めます。

```
{
            "Sid": "BatchGetAPIIncluded",
            "Effect": "Allow",
            "Action": [
                "glue:BatchGetDevEndpoints"
            ],
            "Resource": [
                "arn:aws:glue:us-east-1:123456789012:devEndpoint/de1"
            ]
}
```

このポリシーでは、de1 という名前の開発エンドポイントに正常にアクセスできます。ただし、de2 という名前の開発エンドポイントにアクセスしようとすると、エラーが返されます。

```
An error occurred (AccessDeniedException) when calling the BatchGetDevEndpoints
 operation: No access to any requested resource.
```

> **Important**
>
> List や BatchGet API オペレーションを使用するなど、IAM ポリシーを設定するための別の方法については、「AWS Glue のアイデンティティベースポリシーの例」を参照してください。

クロスアカウントアクセス許可の付与

アカウント間で Data Catalog のリソースへのアクセスを許可することで、抽出、変換、ロード (ETL) ジョブで、異なるアカウントからデータをクエリし、そのデータを結合できるようになります。

トピック

- クロスアカウントのアクセスを許可する AWS Glue のメソッド
- Data Catalog リソースポリシーの追加または更新
- クロスアカウント API コールの実行
- クロスアカウント ETL コールの実行
- CloudTrail のクロスアカウントロギング
- クロスアカウントリソースの所有権および請求
- クロスアカウントアクセスの制約事項

クロスアカウントのアクセスを許可する AWS Glue のメソッド

自分のデータに対するアクセスを外部の AWS アカウントに許可するためには AWS Glue メソッドまたは AWS Lake Formation クロスアカウント付与を使用する必要があります。AWS Glue メソッドは AWS Identity and Access Management (IAM) ポリシーを使用して、きめ細かなアクセスコントロールを実現します。Lake Formation では、リレーショナルデータベースシステムでの GRANT/REVOKE コマンドに類似した、よりシンプルなアクセス許可 (GRANT/REVOKE) のモデルを使用します。

このセクションでは、AWS Glue メソッドの使用について説明します。Lake Formation のクロスアカウント付与の使用については、AWS Lake Formation デベロッパーガイドの「Granting Lake Formation Permissions」を参照してください。

リソースにクロスアカウントアクセスを許可するためには、2 つの AWS Glue メソッドが用意されています。

- Data Catalog のリソースポリシーを使用
- IAM ロール を使用

リソースポリシーを使用してのクロスアカウントアクセスの許可

Data Catalog リソースポリシーを使用してクロスアカウントアクセスを許可するための、一般的な手順を以下に示します。

1. アカウント A の管理者 (または権限付与されたアイデンティティ) が、リソースポリシーをアカウント A 内の Data Catalog にアタッチします。このポリシーは、アカウント A のカタログにあるリソースに対してオペレーションを実行するための、特定のクロスアカウントアクセス許可をアカウント B に付与します。
2. アカウント B の管理者は、アカウント A から受け取ったアクセス許可を委任する、アカウント B の IAM アイデンティティに IAM ポリシーをアタッチします。

   これで、アカウント B のアイデンティティは、アカウント A にある指定されたリソースにアクセスできるようになります。

   このアイデンティティがこのリソースにアクセスするには、リソースの所有者 (アカウント A) およびその親アカウント (アカウントB) の両方からアクセスを許可される必要があります。

IAM ロールを使用したクロスアカウントアクセス許可の付与

以下に、IAM ロールを使用してクロスアカウントアクセスを許可する一般的な手順を示します。

1. リソースを所有するアカウント (アカウント A) 内の管理者 (または権限許可された他のアイデンティティ) が IAM ロールを作成します。
2. アカウント A 内の管理者は、対象のリソースにアクセスするためのクロスアカウントアクセス許可を付与するロールにポリシーをアタッチします。

3. アカウント A 内の管理者により、ロールへの信頼ポリシーのアタッチが行われます。このロールは、別のアカウント (アカウント B) 内の IAM アイデンティティを、そのロールを引き受けることのできるプリンシパルとして識別します。

   信頼ポリシーのプリンシパルに、ロールを引き受けるための AWS サービスへのアクセス許可を付与する場合は、このプリンシパルを AWS サービスのプリンシパルとすることができます。

4. これで、アカウント B の管理者はアカウント B の 1 つ以上の IAM アイデンティティにアクセス許可を委任して、ロールを引き受けることを許可します。これにより、アカウント B のこれらのアイデンティティに対してアカウント A のリソースに対するアクセス権が付与されます。

IAM を使用したアクセス許可の委任の詳細については、「IAM ユーザーガイド」の「アクセス管理」を参照してください。ユーザー、グループ、ロール、アクセス権限の詳細については、「IAM ユーザーガイド」の「ID (ユーザー、グループ、ロール)」を参照してください。

この 2 つの方法の比較については、IAM ユーザーガイド の「IAM ロールとリソースベースのポリシーとの相違点」を参照してください。AWS Glue は両方の方法をサポートしていますが、リソースポリシーが許可できるのはデータカタログリソースへのアクセスのみ、という制限があります。

例えば、アカウント B の Dev ロールにアカウント A のデータベース db1 へのアクセスを許可するときは、次のリソースポリシーをアカウント A のカタログにアタッチします。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "glue:GetDatabase"
      ],
      "Principal": {"AWS": [
        "arn:aws:iam::account-B-id:role/Dev"
      ]},
      "Resource": [
        "arn:aws:glue:us-east-1:account-A-id:catalog",
        "arn:aws:glue:us-east-1:account-A-id:database/db1"
      ]
    }
  ]
}
```

さらに、アカウント B がアカウント A の db1 に実際にアクセスする前に、アカウント B で次の IAM ポリシーを Dev ロールにアタッチする必要があります。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "glue:GetDatabase"
      ],
      "Resource": [
        "arn:aws:glue:us-east-1:account-A-id:catalog",
        "arn:aws:glue:us-east-1:account-A-id:database/db1"
      ]
    }
  ]
}
```

Data Catalog リソースポリシーの追加または更新

コンソール、API、または AWS Glue (AWS Command Line Interface) により、AWS CLI Data Catalog のリソースポリシーを追加したり、更新したりできます。

> **Important**
>
> 自分のアカウントの AWS Lake Formation で、既にクロスアカウントのアクセス許可付与を作成済みの場合は、Data Catalog のリソースポリシーを追加または更新するには追加の手順が必要となります。詳細については、AWS Lake Formation デベロッパーガイドの「AWS Glue Glue と Lake Formation の両方を使用したクロスアカウント許可の管理」を参照してください。
>
> Lake Formation クロスアカウント許可が存在するかどうかを確認するときは、`glue:GetResourcePolicies` API オペレーションまたは AWS CLI を使用します。この `glue:GetResourcePolicies` が既存のデータカタログポリシー以外のポリシーを返した場合、Lake Formation の許可は存在します。詳細については、AWS Lake Formation デベロッパーガイドの「Viewing All Cross-Account Grants Using the GetResourcePolicies API」(GetResourcePolicies API を使用したすべてのクロスアカウント許可の表示) を参照してください。

Data Catalog のリソースポリシーを追加または更新するには (コンソール)

1. https://console.aws.amazon.com/glue/ で AWS Glue コンソール を開きます。

   glue:PutResourcePolicy アクセス許可を持つ AWS Identity and Access Management (IAM) 管理ユーザーとしてサインインします。
2. ナビゲーションペインで [設定] を選択します。
3. [Data catalog settings] (データカタログ設定) ページの [Permissions] (アクセス許可) の下にあるテキスト欄に、リソースポリシーを貼り付けます。次に、[Save] (保存) を選択します。

   ポリシー内のアクセス許可が、Lake Formation を使用して付与されたアクセス許可に追加されることを示す警告がコンソールに表示される場合は、[Proceed] (続行する) をクリックします。

Data Catalog のリソースポリシーを追加または更新するには (AWS CLI)

- `aws glue put-resource-policy` コマンドを送信します。Lake Formation· グラントが既に存在する場合には、`--enable-hybrid` オプションに値 `'TRUE'` を設定する必要があります。

  このコマンドの使用例については、「AWS Glue のリソースベースのポリシーの例」を参照してください。

クロスアカウント API コールの実行

すべての AWS Glue Data Catalog オペレーションに `CatalogId` フィールドがあります。クロスアカウントアクセスを有効にするために必要なアクセス許可が付与済みであれば、複数のアカウントをまたいで Data Catalog API を呼び出すことが可能です。呼び出し元は、ターゲット AWS アカウント ID を `CatalogId` で渡すことで、ターゲットアカウントのリソースに対するアクセスを実行します。

`CatalogId` 値を指定しない場合、AWS Glue はデフォルトで発信者自身のアカウント ID を使用するため、コールはクロスアカウントにはなりません。

クロスアカウント ETL コールの実行

一部の AWS Glue PySpark および Scala API には、カタログ ID フィールドがあります。クロスアカウントアクセスを有効にする、すべての必要なアクセス許可が付与されていれば、ETL ジョブはアカウントをまたがって、PySpark および Scala による API オペレーション呼び出しを行えます。

これには、ターゲットアカウント内の Data Catalog リソースにアクセスするように、ターゲットの AWS アカウント ID をカタログ ID フィールドで指定し受け渡す必要があります、

カタログ ID 値を指定しない場合、AWS Glue はデフォルトで発信者自身のアカウント ID を使用するため、コールはクロスアカウントにはなりません。

`catalog_id` をサポートする PySpark API については、「GlueContext クラス」を参照してください。`catalogId` をサポートする Scala API については、「AWS Glue Scala GlueContext API」を参照してください。

次の例は、ETL ジョブを実行するために被付与者に必要となるアクセス許可を示しています。この例で、*`grantee-account-id`* はジョブを実行するクライアントの `catalog-id`、*`grantor-account-id`* はリソースの所有者です。この例では、権限付与者のアカウントにあるすべてのカタログリソースに対するアクセス許可を付与します。付与されたリソースの範囲を制限するには、カタログ、データベース、テーブル、および接続の特定の ARN を指定できます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:GetConnection",
                "glue:GetDatabase",
                "glue:GetTable",
                "glue:GetPartition"
            ],
            "Principal": {"AWS": ["arn:aws:iam::grantee-account-id:root"]},
            "Resource": [
                "arn:aws:glue:us-east-1:grantor-account-id:*"
            ]
        }
    ]
}
```

> **Note**
>
> 権限付与者のアカウントのテーブルで Amazon S3 の場所を指しており、この場所が権限付与者のアカウントにも含まれる場合、被付与者のアカウントで ETL ジョブを実行するために

使用する IAM ロールには、権限付与者のアカウントからオブジェクトをリスト化し取得するための、アクセス許可が必要となります。

アカウント A のクライアントが ETL ジョブを作成して実行するアクセス許可をすでに付与されている場合、クロスアカウントアクセスのために ETL ジョブをセットアップする基本的な手順は次のとおりです。

1. クロスアカウントのデータアクセスを許可します (Amazon S3 クロスアカウントアクセスがセットアップ済みである場合は、このステップをスキップします)。
   a. アカウント B の Amazon S3 バケットポリシーを更新して、アカウント A からのクロスアカウントアクセスを許可します。
   b. アカウント A の IAM ポリシーを更新して、アカウント B のバケットへのアクセスを許可します。
2. Data Catalog へのクロスアカウントアクセスを許可します
   a. アカウント B の Data Catalog にアタッチされたリソースポリシーを作成または更新して、アカウント A からのアクセスを許可します。
   b. アカウント A の IAM ポリシーを更新して、アカウント B の Data Catalog へのアクセスを許可します。

CloudTrail のクロスアカウントロギング

AWS Glue の抽出、変換、ロード (ETL) ジョブが、AWS Lake Formation クロスアカウント付与を介して共有された Data Catalog テーブルの基板となるデータにアクセスする際には、AWS CloudTrail ログ記録に追加的な動作が発生します。

この点を考えるために、テーブルを共有する AWS アカウントは所有者アカウントであり、テーブルの共有先アカウントは受け取りアカウントとします。受取人アカウントの ETL ジョブが、所有者アカウントのテーブルにあるデータにアクセスすると、受取人アカウントのログに追加された CloudTrail のデータアクセスイベントが所有者アカウントの CloudTrail ログにコピーされます。これは、所有者アカウントが、さまざまな受け取りアカウントによるデータアクセスを追跡できるようにするためです。デフォルトでは、CloudTrail イベントには、人間が読める形式のプリンシパル ID (プリンシパル ARN) は含まれていません。受け取りアカウントの管理者は、ログにプリンシパル ARN を含めるようにオプトインできます。

詳細については、AWS Lake Formationデベロッパーガイドの「CloudTrail のクロスアカウントロギング」を参照してください。

以下の資料も参照してください。

- the section called “ログ記録とモニタリング”

クロスアカウントリソースの所有権および請求

例えば、AWS アカウント (アカウント A) のユーザーが、別のアカウント (アカウント B) にデータベースなどの新しいリソースを作成すると、このリソースは、リソースの作成先のアカウントであるアカウント B に所有されます。アカウント B の管理者は、新しいリソースにアクセスする完全なアクセス許可を自動的に取得します。これには、読み取り、書き込み、および第 3 のアカウントに対するアクセス許可の付与が含まれます。アカウント A のユーザーは、アカウント B によって付与された適切なアクセス許可を持っている場合のみ、前の手順で作成したリソースにアクセスできます。

ストレージコストおよび新しいリソースに直接関連するその他のコストは、リソース所有者であるアカウント B に請求されます。リソースを作成したユーザーからのリクエストのコストは、リクエスタのアカウントであるアカウント A に請求されます。

AWS Glue での請求と料金の詳細については、「How AWS Pricing Works」を参照してください。

クロスアカウントアクセスの制約事項

AWS Glue クロスアカウントアクセスには、次の制約事項があります。

- リージョンが AWS Glue をサポートする前に、Amazon Athena または Amazon Redshift Spectrum を使用してデータベースとテーブルを作成し、リソースオーナーアカウントが Amazon Athena データカタログを AWS Glue に移行していない場合、AWS Glue へのクロスアカウントアクセスは許可されません。現在の移行ステータスは GetCatalogImportStatus (get_catalog_import_status) を使用して確認できます。Athena カタログを AWS Glue に移行する方法の詳細については、「Amazon Athena ユーザーガイド」の「ステップバイステップの AWS Glue Data Catalog へのアップグレード」を参照してください。
- クロスアカウントアクセスは、データベース、テーブル、ユーザー定義関数、接続などの Data Catalog リソースでのみサポートされています。
- Athena `DataCatalog` リソースからデータカタログへのクロスアカウントアクセスには、カタログを Athena として登録する必要があります。手順については、Amazon Athena ユーザーガイドの「別のアカウントからの AWS Glue Data Catalog の登録」を参照してください。

AWS Glue アイデンティティとアクセスに関するトラブルシューティング

次の情報は、AWS Glue と IAM を使用する際に遭遇しがちなよくある問題を診断し修復するのに役立ちます。

トピック

- AWS Glue でアクションを実行するための権限がない
- iam:PassRole を実行する権限がありません
- 自分の AWS アカウント 以外のユーザーに AWS Glue リソースへのアクセスを許可したい

AWS Glue でアクションを実行するための権限がない

「I am not authorized to perform an action in Amazon Bedrock」というエラーが表示された場合、そのアクションを実行できるようにポリシーを更新する必要があります。

次のエラー例は、mateojackson IAM ユーザーがコンソールを使用して、ある *my-example-widget* リソースに関する詳細情報を表示しようとしたことを想定して、その際に必要なglue:*GetWidget* アクセス許可を持っていない場合に発生するものです。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 glue:GetWidget on resource: my-example-widget
```

この場合、glue:*GetWidget* アクションを使用して *my-example-widget*リソースへのアクセスを許可するように、mateojackson ユーザーのポリシーを更新する必要があります。

サポートが必要な場合は、AWS 管理者に問い合わせてください。サインイン認証情報を提供した担当者が管理者です。

iam:PassRole を実行する権限がありません

iam:PassRole アクションを実行する権限がありません、というエラーが表示された場合は、ロールを AWS Glue に渡せるように使用しているポリシーを更新します。

一部の AWS のサービスでは、新しいサービスロールやサービスリンクロールを作成せずに、既存のロールをサービスに渡すことができます。そのためには、サービスにロールを渡す権限が必要です。

次のエラー例は、 marymajor という名前の IAM ユーザーが、コンソールから AWS Glue でアクションを実行しようとすると発生します。ただし、このアクションをサービスが実行するには、サー

ビスロールから付与された権限が必要です。メアリーには、ロールをサービスに渡す許可がありません。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

この場合、Mary のポリシーを更新してメアリーに `iam:PassRole` アクションの実行を許可する必要があります。

サポートが必要な場合は、AWS 管理者に問い合わせてください。サインイン認証情報を提供した担当者が管理者です。

自分の AWS アカウント 以外のユーザーに AWS Glue リソースへのアクセスを許可したい

他のアカウントのユーザーや組織外の人が、リソースにアクセスするために使用できるロールを作成できます。ロールの引き受けを委託するユーザーを指定できます。リソースベースのポリシーまたはアクセスコントロールリスト (ACL) をサポートするサービスの場合、それらのポリシーを使用して、リソースへのアクセスを付与できます。

詳細については、以下を参照してください。

- AWS Glue でこれらの機能がサポートされているかどうかを確認するには、「AWS Glue の仕組み IAM」を参照してください。
- 所有している AWS アカウント 全体のリソースへのアクセス権を提供する方法については、IAM ユーザーガイドの所有している別の AWS アカウント アカウントへのアクセス権を IAM ユーザーに提供を参照してください。
- サードパーティーの AWS アカウント にリソースへのアクセス権を提供する方法については、「IAM ユーザーガイド」の「サードパーティが所有する AWS アカウント へのアクセス権を付与する」を参照してください。
- ID フェデレーションを介してアクセスを提供する方法については、IAM ユーザーガイドの外部で認証されたユーザー (ID フェデレーション) へのアクセスの許可を参照してください。
- クロスアカウントアクセスにおけるロールとリソースベースのポリシーの使用方法の違いについては、「IAM ユーザーガイド」の「IAM でのクロスアカウントのリソースへのアクセス」を参照してください。

AWS Lake Formation アクセスコントロールモデル

AWS Glue 5.0 は、AWS Lake Formation を介してデータにアクセスする 2 つのモデルをサポートしています。

トピック

- AWS Lake Formation で AWS Glue を使用したフルテーブルアクセス
- AWS Glue と AWS Lake Formation を併用したきめ細かなアクセスコントロール

AWS Lake Formation で AWS Glue を使用したフルテーブルアクセス

フルテーブルアクセスの概要

AWS Glue 5.0 は、AWS Lake Formation で定義されたポリシーに基づいて、Apache Spark でのフルテーブルアクセス (FTA) コントロールをサポートします。この機能により、ジョブロールにフルテーブルアクセスがある場合は、AWS Glue Spark ジョブからの AWS Lake Formation 登録済みテーブルに対する読み取りおよび書き込み操作が有効になります。FTA は、テーブルレベルでセキュリティ規制に準拠する必要があるユースケースに最適です。また、障害耐性分散データセット (RDD)、カスタムライブラリ、AWS Lake Formation テーブルを含むユーザー定義関数 (UDF) などの Spark 機能をサポートしています。

AWS Glue Spark ジョブがフルテーブルアクセス (FTA) に対応するように設定されている場合、AWS Lake Formation 認証情報は AWS Lake Formation 登録済みテーブルの Amazon S3 データの読み取り/書き込みに使用され、ジョブのランタイムロール認証情報は AWS Lake Formation に登録されていないテーブルの読み取り/書き込みに使用されます。この機能を使用すると、Apache Hive テーブルと Iceberg テーブルで CREATE、ALTER、DELETE、UPDATE、MERGE INTO ステートメントを含むデータ操作言語 (DML) 操作が可能になります。

> **Note**
>
> 要件を確認し、きめ細かなアクセスコントロール (FGAC) またはフルテーブルアクセス (FTA) がニーズに応じているかどうかを判断してください。特定の AWS Glue ジョブに対して有効にできる AWS Lake Formation アクセス許可メソッドは 1 つだけです。ジョブは、フルテーブルアクセス (FTA) ときめ細かなアクセスコントロール (FGAC) を同時に実行することはできません。

AWS Glue でのフルテーブルアクセス (FTA) の仕組み

AWS Lake Formation には、きめ細かなアクセスコントロール (FGAC) とフルテーブルアクセス (FTA) の 2 つのデータアクセスコントロールアプローチがあります。FGAC は、列、行、セルレベルのフィルタリングを通じてセキュリティを強化するため、きめ細かなアクセス許可を必要とするシナリオに最適です。FTA は、テーブルレベルのアクセス許可が必要となる簡単なアクセスコントロールシナリオに最適です。これにより、きめ細かなアクセスモードを有効にする必要がなくなり、システムドライバーとシステムエグゼキューターを回避することでパフォーマンスを向上させ、コストを削減できます。また、読み取りと書き込み操作 (CREATE、ALTER、DELETE、UPDATE、MERGE INTO コマンドを含む) の両方がサポートされます。

AWS Glue 4.0 では、AWS Lake Formation ベースのデータアクセスは、AWS Glue によって提供されるユーティリティクラスである GlueContext クラスを通じて機能していました。AWS Glue 5.0 では、AWS Lake Formation ベースのデータアクセスはネイティブ Spark SQL、Spark DataFrames を通じて利用でき、GlueContext クラスを通じて引き続きサポートされます。

フルテーブルアクセスの実装

ステップ 1: AWS Lake Formation でテーブルへのフルアクセスを有効にする

フルテーブルアクセス (FTA) モードを使用するには、AWS Lake Formation で IAM セッションタグの検証を行わずに、サードパーティーのクエリエンジンがデータにアクセスすることを許可する必要があります。有効にするには、「フルテーブルアクセスのためのアプリケーション統合」のステップに従います。

ステップ 2: ジョブランタイムロールの IAM アクセス許可を設定する

基盤となるデータへの読み取りまたは書き込みアクセスには、AWS Lake Formation アクセス許可に加えて、ジョブランタイムロールに `lakeformation:GetDataAccess` IAM アクセス許可が必要です。このアクセス許可よって、AWS Lake Formation がデータにアクセスするための一時的な認証情報のリクエストを承諾します。

以下は、Amazon S3 のスクリプトにアクセスするための IAM アクセス許可、Amazon S3 へのログのアップロード、AWS Glue API アクセス許可、AWS Lake Formation へのアクセス許可を指定する方法のポリシー例です。

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

```
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": [
        "arn:aws:s3:::amzn-s3-demo-bucket/scripts/*"
      ]
    },
    {
      "Sid": "LoggingAccess",
      "Effect": "Allow",
      "Action": [
        "s3:PutObject"
      ],
      "Resource": [
        "arn:aws:s3:::amzn-s3-demo-bucket/logs/*"
      ]
    },
    {
      "Sid": "GlueCatalogAccess",
      "Effect": "Allow",
      "Action": [
        "glue:GetDatabase",
        "glue:GetDatabases",
        "glue:GetTable",
        "glue:GetTables",
        "glue:GetPartition",
        "glue:GetPartitions",
        "glue:CreateTable",
        "glue:UpdateTable"
      ],
      "Resource": [
        "arn:aws:glue:us-east-1:123456789012:catalog",
        "arn:aws:glue:us-east-1:123456789012:database/default",
        "arn:aws:glue:us-east-1:123456789012:table/default/*"
      ]
    },
    {
      "Sid": "LakeFormationAccess",
      "Effect": "Allow",
      "Action": [
        "lakeformation:GetDataAccess"
      ],
      "Resource": "*"
```

```
    }
  ]
}
```

ステップ 2.1: AWS Lake Formation アクセス許可を設定する

Amazon S3 からデータを読み取る AWS Glue Spark ジョブには、AWS Lake Formation SELECT アクセス許可が必要です。

Amazon S3 でデータを書き込み/削除する AWS Glue Spark ジョブには、AWS Lake Formation ALL アクセス許可が必要です。

AWS Glue データカタログを操作する AWS Glue Spark ジョブには、必要に応じて DESCRIBE、ALTER、DROP アクセス許可が必要です。

ステップ 3: AWS Lake Formation を使用してフルテーブルアクセスのための Spark セッションを初期化する

AWS Lake Formation に登録されたテーブルにアクセスするには、Spark の初期化中に以下の設定を行い、AWS Lake Formation 認証情報を使用するように Spark を設定する必要があります。

AWS Lake Formation に登録されているテーブルにアクセスするには、AWS Lake Formation 認証情報を使用するように Spark セッションを明示的に設定する必要があります。Spark セッションを初期化する際に、次の設定を追加します。

```
from pyspark.sql import SparkSession

# Initialize Spark session with Lake Formation configurations
spark = SparkSession.builder \
    .appName("Lake Formation Full Table Access") \
    .config("spark.sql.catalog.glue_catalog",
 "org.apache.spark.sql.catalog.hive.GlueCatalog") \
    .config("spark.sql.catalog.glue_catalog.glue.lakeformation-enabled", "true") \
    .config("spark.sql.defaultCatalog", "glue_catalog") \
    .getOrCreate()
```

設定キー:

- `spark.sql.catalog.glue_catalog`: GlueCatalog 実装を使用する「glue_catalog」という名前のカタログを登録します

- `spark.sql.catalog.glue_catalog.glue.lakeformation-enabled`: このカタログの AWS Lake Formation 統合を明示的に有効にします
- カタログ名 (この例では「glue_catalog」) はカスタマイズできますが、両方の設定で一貫している必要があります

Hive

```
--conf spark.hadoop.fs.s3.credentialsResolverClass=com.amazonaws.AWS
 Glue.accesscontrol.AWSLakeFormationCredentialResolver
--conf spark.hadoop.fs.s3.useDirectoryHeaderAsFolderObject=true
--conf spark.hadoop.fs.s3.folderObject.autoAction.disabled=true
--conf spark.sql.catalog.skipLocationValidationOnCreateTable.enabled=true
--conf spark.sql.catalog.createDirectoryAfterTable.enabled=true
--conf spark.sql.catalog.dropDirectoryBeforeTable.enabled=true
```

Iceberg

```
--conf spark.hadoop.fs.s3.credentialsResolverClass=com.amazonaws.AWS
 Glue.accesscontrol.AWSLakeFormationCredentialResolver
--conf spark.hadoop.fs.s3.useDirectoryHeaderAsFolderObject=true
--conf spark.hadoop.fs.s3.folderObject.autoAction.disabled=true
--conf spark.sql.catalog.skipLocationValidationOnCreateTable.enabled=true
--conf spark.sql.catalog.createDirectoryAfterTable.enabled=true
--conf spark.sql.catalog.dropDirectoryBeforeTable.enabled=true
--conf spark.sql.catalog.<catalog>.AWS Glue.lakeformation-enabled=true
```

- `spark.hadoop.fs.s3.credentialsResolverClass=com.amazonaws.AWS Glue.accesscontrol.AWSLakeFormationCredentialResolver`: AWS Lake Formation に登録されたテーブルに AWS Lake Formation S3 認証情報を使用するように EMR ファイルシステム (EMRFS) を設定します。テーブルが登録されていない場合は、ジョブのランタイムロール認証情報を使用します。
- `spark.hadoop.fs.s3.useDirectoryHeaderAsFolderObject=true` および `spark.hadoop.fs.s3.folderObject.autoAction.disabled=true`: S3 フォルダの作成時に $folder$ サフィックスの代わりにコンテンツタイプヘッダー application/x-directory を使用するように EMRFS を設定します。AWS Lake Formation 認証情報は $folder$ サフィックスを持つテー

ブルフォルダの読み取りを許可しないため、この設定は AWS Lake Formation テーブルを読み取る際に必要です。

- `spark.sql.catalog.skipLocationValidationOnCreateTable.enabled=true`: 作成前に空のテーブルの場所を検証するステップをスキップするよう Spark を設定します。空の場所を検証するための AWS Lake Formation 認証情報は AWS Glue Data Catalog テーブルの作成後にしか使用できないため、この設定は AWS Lake Formation に登録されたテーブルに対して必要です。この設定がない場合、ジョブのランタイムロール認証情報によって空のテーブルの場所が検証されます。
- `spark.sql.catalog.createDirectoryAfterTable.enabled=true`: Hive メタストアでテーブルを作成した後に Amazon S3 フォルダを作成するよう Spark を設定します。Amazon S3 フォルダを作成するための AWS Lake Formation 認証情報は AWS Glue Data Catalog テーブルの作成後にしか使用できないため、この設定は AWS Lake Formation に登録されたテーブルに対して必要です。
- `spark.sql.catalog.dropDirectoryBeforeTable.enabled=true`: Hive メタストアでテーブルを削除する前に Amazon S3 フォルダを削除するよう Spark を設定します。AWS Glue Data Catalog からテーブルを削除した後は、S3 フォルダを削除するための AWS Lake Formation 認証情報を使用できないため、この設定は AWS Lake Formation に登録されたテーブルに対して必要です。
- `spark.sql.catalog.<catalog>.AWS Glue.lakeformation-enabled=true`: AWS Lake Formation に登録されたテーブルに AWS Lake Formation Amazon S3 認証情報を使用するように Iceberg カタログを設定します。テーブルが登録されていない場合は、デフォルトの環境認証情報を使用します。

使用パターン

DataFrames での FTA の使用

Spark に精通しているユーザーは、DataFrames を AWS Lake Formation フルテーブルアクセスで使用できます。

AWS Glue 5.0 では、Lake Formation フルテーブルアクセスのネイティブ Spark サポートが追加されており、保護されたテーブルの操作方法が簡素化されています。この機能により、フルテーブルアクセスが付与されると AWS Glue 5.0 AWS Glue Spark ジョブはデータを直接読み書きできるため、以前に特定の抽出、変換、ロード (ETL) 操作の制限がなくなります。障害耐性分散データセット (RDD)、カスタムライブラリ、ユーザー定義関数 (UDF) などの高度な Spark 機能を AWS Lake Formation テーブルで活用できるようになりました。

AWS Glue 5.0 のネイティブ Spark FTA

AWS Glue 5.0 は、AWS Lake Formation で定義されたポリシーに基づいて、Apache Spark でのフルテーブルアクセス (FTA) コントロールをサポートします。このレベルのコントロールは、テーブルレベルでセキュリティ規制に準拠する必要があるユースケースに最適です。

Apache Iceberg テーブルの例

```
from pyspark.sql import SparkSession

catalog_name = "spark_catalog"
aws_region = "us-east-1"
aws_account_id = "123456789012"
warehouse_path = "s3://amzn-s3-demo-bucket/warehouse/"

spark = SparkSession.builder \

 .config("spark.sql.extensions","org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensi
 \
    .config(f"spark.sql.catalog.{catalog_name}",
 "org.apache.iceberg.spark.SparkSessionCatalog") \
    .config(f"spark.sql.catalog.{catalog_name}.warehouse", f"{warehouse_path}") \
    .config(f"spark.sql.catalog.{catalog_name}.client.region",f"{aws_region}") \
    .config(f"spark.sql.catalog.{catalog_name}.glue.account-id",f"{aws_account_id}") \
    .config(f"spark.sql.catalog.{catalog_name}.glue.lakeformation-enabled","true") \
    .config(f"spark.sql.catalog.dropDirectoryBeforeTable.enabled", "true") \
    .config(f"spark.sql.catalog.{catalog_name}.catalog-impl",
 "org.apache.iceberg.aws.glue.GlueCatalog") \
    .config(f"spark.sql.catalog.{catalog_name}.io-impl",
 "org.apache.iceberg.aws.s3.S3FileIO") \
    .config("spark.sql.defaultCatalog", catalog_name) \  # Add this line
    .getOrCreate()

database_name = "your_database"
table_name = "your_table"

df = spark.sql(f"select * from {database_name}.{table_name}")
df.show()
```

必要な IAM アクセス許可

AWS Glue ジョブ実行ロールには、以下が必要です。

```
{
    "Action": "lakeformation:GetDataAccess",
    "Resource": "*",
    "Effect": "Allow"
}
```

さらに、データロケーションに対する適切な S3 アクセス許可。

Lake Formation の設定

AWS Glue 5.0 でネイティブ Spark FTA を使用する前に:

1. AWS Lake Formation で IAM セッションタグの検証を行わずに、サードパーティーのクエリエンジンがデータにアクセスできるようにする
2. AWS Lake Formation コンソールを使用して AWS Glue ジョブ実行ロールに適切なテーブルアクセス許可を付与する
3. 上記の例に示す必須パラメータを使用して Spark セッションを設定する

DynamicFrames での FTA の使用

AWS Glue のネイティブ DynamicFrames は、ETL 操作を最適化するために AWS Lake Formation フルテーブルアクセスで使用できます。フルテーブルアクセス (FTA) は、テーブルレベルでアクセス許可を付与するセキュリティモデルを提供します。これにより、行および列レベルのアクセス許可チェックのオーバーヘッドを回避できるため、きめ細かなアクセスコントロール (FGAC) と比較してデータ処理が高速化されます。このアプローチは、セキュリティ要件を満たすテーブル全体とテーブルレベルのアクセス許可を処理する必要がある場合に役立ちます。

AWS Glue 4.0 では、FTA を使用する DynamicFrames では特定の GlueContext 設定が必要でした。FTA を使用する既存の AWS Glue 4.0 DynamicFrame コードは 5.0 AWS Glue でも引き続き動作しますが、新しいバージョンではネイティブ Spark FTA サポートも提供され、柔軟性が向上しています。新規の開発で、特に、障害耐性分散データセット (RDD)、カスタムライブラリ、AWS Lake Formation テーブルを含むユーザー定義関数 (UDF) などの追加機能が必要になる場合は、DataFrames セクションで説明されているネイティブの Spark アプローチを使用することを検討してください。

必要なアクセス許可

Glue ジョブを実行する IAM ロールには次のものが必要です。

- `lakeformation:GetDataAccess` アクセス許可
- Lake Formation コンソールを通じて付与される適切な Lake Formation テーブルのアクセス許可

AWS Glue 5.0 での DynamicFrame 実装例

```
from awsglue.context import GlueContext
from pyspark.context import SparkContext

# Initialize Glue context
sc = SparkContext()
glueContext = GlueContext(sc)

# Configure catalog for Iceberg tables
catalog_name = "glue_catalog"
aws_region = "us-east-1"
aws_account_id = "123456789012"
warehouse_path = "s3://amzn-s3-demo-bucket/warehouse/"

spark = glueContext.spark_session
spark.conf.set(f"spark.sql.catalog.{catalog_name}",
 "org.apache.iceberg.spark.SparkCatalog")
spark.conf.set(f"spark.sql.catalog.{catalog_name}.warehouse", f"{warehouse_path}")
spark.conf.set(f"spark.sql.catalog.{catalog_name}.catalog-impl",
 "org.apache.iceberg.aws.glue.GlueCatalog")
spark.conf.set(f"spark.sql.catalog.{catalog_name}.io-impl",
 "org.apache.iceberg.aws.s3.S3FileIO")
spark.conf.set(f"spark.sql.catalog.{catalog_name}.glue.lakeformation-enabled","true")
spark.conf.set(f"spark.sql.catalog.{catalog_name}.client.region",f"{aws_region}")
spark.conf.set(f"spark.sql.catalog.{catalog_name}.glue.id", f"{aws_account_id}")

# Read Lake Formation-protected table with DynamicFrame
df = glueContext.create_data_frame.from_catalog(
    database="your_database",
    table_name="your_table"
)
```

追加設定

AWS Glue Studio ノートブックでフルテーブルアクセスモードを設定する

AWS Glue Studio ノートブックのインタラクティブ Spark セッションから AWS Lake Formation に登録されたテーブルにアクセスするには、互換性アクセス許可モードを使用する必要があります。イ

ンタラクティブセッションを開始する前に、%%configure マジックコマンドを使用して Spark 設定をセットアップします。この設定は、セッション開始後に適用できないため、ノートブックの最初のコマンドである必要があります。次のテーブルタイプに基づいて設定を選択します。

Hive テーブルの場合

```
%%configure
--conf
 spark.hadoop.fs.s3.credentialsResolverClass=com.amazonaws.glue.accesscontrol.AWSLakeFormationC
--conf spark.hadoop.fs.s3.useDirectoryHeaderAsFolderObject=true
--conf spark.hadoop.fs.s3.folderObject.autoAction.disabled=true
--conf spark.sql.catalog.skipLocationValidationOnCreateTable.enabled=true
--conf spark.sql.catalog.createDirectoryAfterTable.enabled=true
--conf spark.sql.catalog.dropDirectoryBeforeTable.enabled=true
```

Iceberg テーブルの場合

```
%%configure
--conf
 spark.hadoop.fs.s3.credentialsResolverClass=com.amazonaws.glue.accesscontrol.AWSLakeFormationC
--conf spark.hadoop.fs.s3.useDirectoryHeaderAsFolderObject=true
--conf spark.hadoop.fs.s3.folderObject.autoAction.disabled=true
--conf spark.sql.catalog.skipLocationValidationOnCreateTable.enabled=true
--conf spark.sql.catalog.createDirectoryAfterTable.enabled=true
--conf spark.sql.catalog.dropDirectoryBeforeTable.enabled=true
--conf spark.sql.catalog.glue_catalog.glue.lakeformation-enabled=true
--conf spark.sql.catalog.glue_catalog.warehouse=s3://example-s3-bucket_DATA_LOCATION
--conf spark.sql.catalog.glue_catalog.catalog-
impl=org.apache.iceberg.aws.glue.GlueCatalog
--conf spark.sql.catalog.glue_catalog.io-impl=org.apache.iceberg.aws.s3.S3FileIO
--conf spark.sql.catalog.glue_catalog.glue.account-id=ACCOUNT_ID
--conf spark.sql.catalog.glue_catalog.glue.region=REGION
```

プレースホルダーを置き換えます:

- S3_DATA_LOCATION: *s3://amzn-s3-demo-bucket*
- REGION: *AWS Region (#: us-east-1)*
- ACCOUNT_ID: *#### AWS ##### ID*

> **Note**
>
> ノートブックで Spark 操作を実行する前に、これらの設定を行う必要があります。

サポートされているオペレーション

これらの操作では、AWS Lake Formation 認証情報を使用してテーブルデータにアクセスします。

> **Note**
>
> AWS Lake Formation を有効にする場合:
>
> - FTA の場合: Spark 設定 `spark.sql.catalog.{catalog_name}.glue.lakeformation-enabled` を有効にする

- CREATE TABLE
- ALTER TABLE
- INSERT INTO
- INSERT OVERWRITE
- SELECT
- UPDATE
- MERGE INTO
- DELETE FROM
- ANALYZE TABLE
- REPAIR TABLE
- DROP TABLE
- Spark データソースクエリ
- Spark データソース書き込み

> **Note**
>
> 上記に示されていない操作は、引き続き IAM アクセス許可を使用してテーブルデータにアクセスします。

AWS Glue 4.0 から AWS Glue 5.0 FTA への移行

AWS Glue 4.0 GlueContext FTA から AWS Glue 5.0 ネイティブ Spark FTA に移行する場合:

1. AWS Lake Formation で IAM セッションタグの検証を行わずに、サードパーティーのクエリエンジンがデータにアクセスできるようにします。ステップ 1: AWS Lake Formation でテーブルへのフルアクセスを有効にする に従います。
2. ジョブのランタイムロールを変更する必要はありません。ただし、AWS Glue ジョブ実行ロールに lakeformation:GetDataAccess IAM アクセス許可があることを確認します。
3. スクリプトで Spark セッション設定を変更します。次の Spark 設定が存在することを確認します。

```
--conf spark.sql.catalog.spark_catalog=org.apache.iceberg.spark.SparkSessionCatalog
--conf spark.sql.catalog.spark_catalog.warehouse=s3://<bucket-name>/warehouse/
--conf spark.sql.catalog.spark_catalog.client.region=<REGION>
--conf spark.sql.catalog.spark_catalog.glue.account-id=ACCOUNT_ID
--conf spark.sql.catalog.spark_catalog.glue.lakeformation-enabled=true
--conf spark.sql.catalog.dropDirectoryBeforeTable.enabled=true
```

4. GlueContext DataFrames がネイティブ Spark DataFrames に変更されるようにスクリプトを更新します。
5. AWS Glue 5.0 を使用するように AWS Glue ジョブを更新します。

考慮事項と制約事項

- フルテーブルアクセスが有効になっていないジョブを使用して Hive テーブルが作成され、レコードが挿入されていない場合、フルテーブルアクセスを持つジョブからの後続の読み取りまたは書き込みは失敗します。これは、フルテーブルアクセスを持たない AWS Glue Spark がテーブルフォルダ名に $folder$ サフィックスを追加するためです。この問題は、以下のいずれかの方法で解決できます。
  - FTA が有効になっていないジョブからテーブルに少なくとも 1 つの行を挿入します。
  - S3 のフォルダ名に $folder$ サフィックスが使用されないようにするため、FTA が有効になっていないジョブを設定します。そのために、Spark の設定を `spark.hadoop.fs.s3.useDirectoryHeaderAsFolderObject=true` にします。
  - Amazon S3 コンソールまたは Amazon S3 CLI を使用して、テーブルの場所 `s3://path/to/table/table_name` に Amazon S3 フォルダを作成します。

- フルテーブルアクセスは EMR ファイルシステム (EMRFS) でのみ機能します。S3A ファイルシステムは互換性がありません。
- フルテーブルアクセスは、Hive テーブルと Iceberg テーブルでサポートされています。Hudi と Delta テーブルのサポートはまだ追加されていません。
- AWS Lake Formation きめ細かなアクセスコントロール (FGAC) ルールまたは AWS Glue Data Catalog ビューを持つテーブルを参照するジョブは失敗します。FGAC ルールまたは AWS Glue Data Catalog ビューを使用してテーブルをクエリするには、FGAC モードを使用する必要があります。FGAC モードを有効にするには、AWS ドキュメント「AWS Glue と AWS Lake Formation を併用したきめ細かなアクセスコントロール」で説明されている手順に従います。
- フルテーブルアクセスは Spark ストリーミングをサポートしていません。
- FGAC と同時に使用することはできません。

AWS Glue と AWS Lake Formation を併用したきめ細かなアクセスコントロール

概要

AWS Glue バージョン 5.0 以降では、AWS Lake Formation を活用して、S3 でサポートされているデータカタログテーブルにきめ細かなアクセスコントロールを適用できます。この機能を使用すると、AWS Glue for Apache Spark ジョブ内の read クエリのテーブル、行、列、セルレベルのアクセスコントロールを設定できます。Lake Formation の詳細と AWS Glue での使用方法については、以下のセクションを参照してください。

Glue 4.0 以前でサポートされていた、AWS Lake Formation のアクセス許可を持つ `GlueContext` ベースのテーブルレベルのアクセスコントロールは、Glue 5.0 ではサポートされません。Glue 5.0 では新しい Spark ネイティブのきめ細かなアクセスコントロール (FGAC) を使用します。次の詳細情報に注意してください。

- row/column/cell のアクセスコントロールにきめ細かなアクセスコントロール (FGAC) が必要な場合は、 Glue 4.0 の `GlueContext`/Glue DynamicFrame から、 Glue 5.0 の Spark DataFrame より前に移行する必要があります。例については、「GlueContext/Glue DynamicFrame から Spark DataFrame への移行」を参照してください。
- フルテーブルアクセスコントロール (FTA) が必要な場合は、Glue 5.0 の DynamicFrames を使って FTA を活用できます。また、ネイティブ Spark アプローチに移行して、障害耐性分散データセット (RDD)、カスタムライブラリ、AWS Lake Formation テーブルを含むユーザー定義関数 (UDF)

などの追加機能を利用することもできます。例については、「AWS Glue 4.0 から AWS Glue 5.0 FTA への移行」を参照してください。

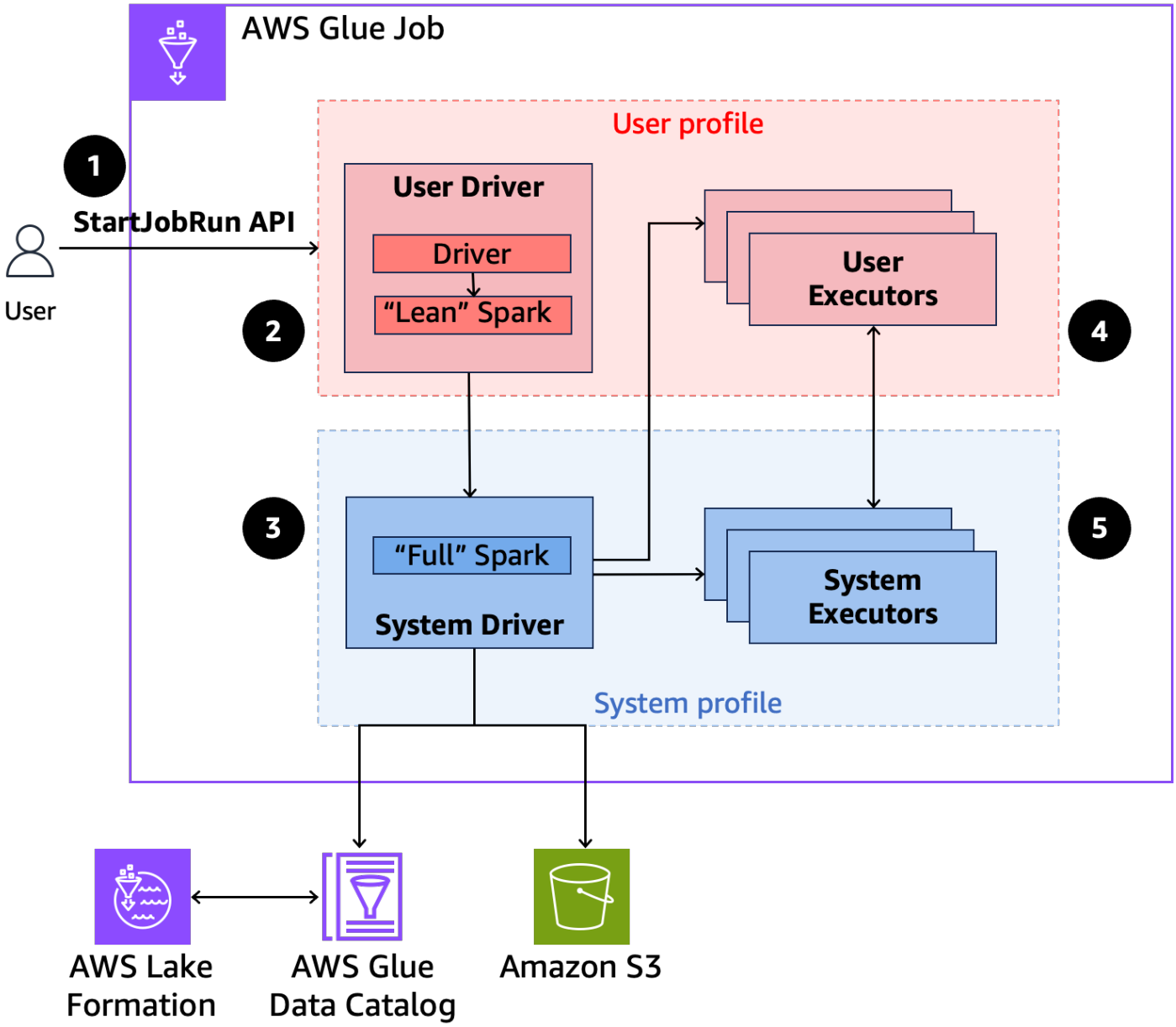
- FGAC が必要ない場合、Spark DataFrame への移行は不要で、ジョブのブックマークやプッシュダウン述語などの `GlueContext` 機能は引き続き機能します。
- FGAC を使用するジョブには、1 つのユーザードライバー、1 つのシステムドライバー、1 つのシステムエグゼキューター、1 つのスタンバイユーザーエグゼキューターの最低 4 つのワーカーが必要です。

AWS Lake Formation で AWS Glue を使用すると、追加料金が発生します。

AWS Glue と AWS Lake Formation の連携方法

AWS Glue と Lake Formation を使用すると、各 Spark ジョブにアクセス許可のレイヤーを適用して、AWS Glue がジョブを実行するときに Lake Formation アクセス許可コントロールを適用できます。AWSGlue は、Spark リソースプロファイルを使用して 2 つのプロファイルを作成し、ジョブを効果的に実行します。ユーザープロファイルはユーザー提供のコードを実行し、システムプロファイルは Lake Formation ポリシーを適用します。詳細については、「AWS Lake Formation とは」および「考慮事項と制限事項」を参照してください。

以下は、AWS Glue が Lake Formation セキュリティポリシーで保護されたデータにアクセスする方法の概要を示します。



1. ユーザーは AWS Lake Formation が有効な AWS Glue ジョブで `StartJobRun` API を呼び出します。
2. AWS Glue はジョブをユーザードライバーに送信し、ユーザープロファイルでジョブを実行します。ユーザードライバーは、タスクの起動、エグゼキュターのリクエスト、S3 または Glue カタログへのアクセスができない Spark のリーンバージョンを実行します。ジョブプランを構築します。
3. AWS Glue は、システムドライバーと呼ばれる 2 番目のドライバーを設定し、システムプロファイルで (特権 ID を使用して) 実行します。AWSGlue は、通信用の 2 つのドライバー間に暗号化された TLS チャネルを設定します。ユーザードライバーはチャネルを使用して、ジョブプ

ランをシステムドライバーに送信します。システムドライバーは、ユーザーが送信したコードを実行しません。フル Spark を実行して、データアクセスのために S3 およびデータカタログと通信します。エグゼキュターをリクエストし、ジョブプランを一連の実行ステージにコンパイルします。

4. 次に、AWS Glue はユーザードライバーまたはシステムドライバーを使用してエグゼキュターでステージを実行します。どのステージのユーザーコードも、ユーザープロファイルのエグゼキュターでのみ実行されます。
5. AWS Lake Formation で保護されたデータカタログテーブルからデータを読み取るステージ、またはセキュリティフィルターを適用するステージは、システムエグゼキュターに委任されます。

ワーカーの最小要件

AWS Glue の Lake Formation 対応ジョブには、1 つのユーザードライバー、1 つのシステムドライバー、1 つのシステムエグゼキューター、1 つのスタンバイユーザーエグゼキューターの最低 4 つのワーカーが必要です。これは、標準の AWS Glue ジョブに必要な最低 2 つのワーカーより多いです。

AWS Glue の Lake Formation 対応ジョブごとに 2 つの Spark ドライバー (1 つはユーザープロファイル用、もう 1 つはシステムプロファイル用) を使用します。同様に、エグゼキューターも 2 つのプロファイルに分割されます。

- システムエグゼキューター: Lake Formation データフィルターが適用されるタスクを処理します。
- ユーザーエグゼキューター: 必要に応じてシステムドライバーによってリクエストされます。

Spark ジョブは遅延する性質があるため、AWS Glue は 2 つのドライバーを差し引いたワーカー総数の 10% (最小 1) をユーザーエグゼキューター用に予約します。

すべての Lake Formation 対応ジョブで、自動スケーリングが有効になっています。つまり、ユーザーエグゼキューターは必要なときにのみ起動します。

設定例については、「考慮事項と制限」を参照してください。

ジョブランタイムロールの IAM アクセス許可

Lake Formation アクセス許可は、AWS Glue Data Catalog リソース、Amazon S3 ロケーション、およびこれらのロケーションにある基盤データへのアクセスを制御します。IAM アクセス許可は、Lake Formation および AWS Glue API とリソースへのアクセスを制御します。データカタ

ログ内のテーブルにアクセスするための Lake Formation アクセス許可 (SELECT) を持っていても、glue:Get* API オペレーションに対する IAM アクセス許可がない場合、操作は失敗します。

以下は、S3 のスクリプトにアクセスするための IAM アクセス許可、S3 AWS へのログのアップロード、Glue API アクセス許可、Lake Formation へのアクセス許可を指定する方法のポリシー例です。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ScriptAccess",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:ListBucket"
            ],
            "Resource": [
            "arn:aws:s3:::*.amzn-s3-demo-bucket/scripts",
            "arn:aws:s3:::*.amzn-s3-demo-bucket/*" ]
        },
        {
         "Sid": "LoggingAccess",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject"
            ],
            "Resource": [
               "arn:aws:s3:::amzn-s3-demo-bucket/logs/*"
            ]
        },
        {
            "Sid": "GlueCatalogAccess",
            "Effect": "Allow",
            "Action": [
                 "glue:Get*",
                 "glue:Create*",
                 "glue:Update*"
            ],
            "Resource": ["*"]
        },
        {
            "Sid": "LakeFormationAccess",
            "Effect": "Allow",
```

```
            "Action": [
             "lakeformation:GetDataAccess"
             ],
            "Resource": ["*"]
        }
    ]
}
```

ジョブランタイムロールの Lake Formation アクセス許可の設定

まず、Hive テーブルの場所を Lake Formation に登録します。次に、目的のテーブルにジョブランタイムロールのアクセス許可を作成します。Lake Formation の詳細については、「AWS Lake Formation デベロッパーガイド」の「AWS Lake Formation とは」を参照してください。

Lake Formation のアクセス許可を設定したら、AWS Glue で Spark ジョブを送信できます。

ジョブ実行の送信

Lake Formation のアクセス許可の設定が完了したら、AWS Glue で Spark ジョブを送信できます。Iceberg ジョブを実行するには、次の Spark 設定を指定する必要があります。Glue ジョブパラメータを使用して設定するには、次のパラメータを入力します。

- キー:

```
--conf
```

- 値:

```
spark.sql.catalog.spark_catalog=org.apache.iceberg.spark.SparkSessionCatalog
       --conf spark.sql.catalog.spark_catalog.warehouse=<S3_DATA_LOCATION>
       --conf spark.sql.catalog.spark_catalog.glue.account-id=<ACCOUNT_ID>
       --conf spark.sql.catalog.spark_catalog.client.region=<REGION>
       --conf spark.sql.catalog.spark_catalog.glue.endpoint=https://
glue.<REGION>.amazonaws.com
```

インタラクティブセッションの使用

AWS Lake Formation 許可の設定が完了したら、AWS Glue でインタラクティブセッションを使用できます。コードを実行する前に、%%configure マジックを使用して次の Spark 設定を提供する必要があります。

```
%%configure
{
    "--enable-lakeformation-fine-grained-access": "true",
    "--conf":
 "spark.sql.catalog.spark_catalog=org.apache.iceberg.spark.SparkSessionCatalog
 --conf spark.sql.catalog.spark_catalog.warehouse=<S3_DATA_LOCATION> --conf
 spark.sql.catalog.spark_catalog.catalog-impl=org.apache.iceberg.aws.glue.GlueCatalog
 --conf spark.sql.catalog.spark_catalog.io-impl=org.apache.iceberg.aws.s3.S3FileIO --
conf
 spark.sql.extensions=org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions
 --conf spark.sql.catalog.spark_catalog.client.region=<REGION> --conf
 spark.sql.catalog.spark_catalog.glue.account-id=<ACCOUNT_ID> --conf
 spark.sql.catalog.spark_catalog.glue.endpoint=https://glue.<REGION>.amazonaws.com"
}
```

AWS Glue 5.0 Notebook またはインタラクティブセッション用の FGAC

AWS Glue で「きめ細かなアクセスコントロール」 (FGAC) を有効にするには、最初のセルを作成する前に、%%configure マジックの一部として Lake Formation に必要な Spark confs を指定する必要があります。

後で呼び出し `SparkSession.builder().conf("").get()` または `SparkSession.builder().conf("").create()` を使用して指定するだけでは不十分です。これは AWS Glue 4.0 からの挙動変更となります。

オープンテーブル形式のサポート

AWS Glue バージョン 5.0 以降には、Lake Formation に基づくきめ細かなアクセスコントロールのサポートが含まれています。AWSGlue は、Hive テーブルタイプと Iceberg テーブルタイプをサポートしています。次の表で、サポートされているすべてのオペレーションについて説明します。

オペレーション	Hive	Iceberg
DDL コマンド	IAM ロールのアクセス許可のみ	IAM ロールのアクセス許可のみ
増分クエリ	該当しない	完全サポートされています
タイムトラベルクエリ	このテーブル形式には適用されません	完全サポートされています

オペレーション	Hive	Iceberg
メタデータテーブル	このテーブル形式には適用されません	サポートされていますが、特定のテーブルは非表示になっています。詳細については、「考慮事項と制限事項」を参照してください。
DML INSERT	IAM アクセス許可のみ	IAM アクセス許可のみ
DML UPDATE	このテーブル形式には適用されません	IAM アクセス許可のみ
DML DELETE	このテーブル形式には適用されません	IAM アクセス許可のみ
読み込みオペレーション	完全サポートされています	完全サポートされています
ストアドプロシージャ	該当しない	register_table および migrate の例外でサポートされます。詳細については、「考慮事項と制限事項」を参照してください。

GlueContext/Glue DynamicFrame から Spark DataFrame への移行

以下は、 Glue 4.0 の GlueContext/Glue DynamicFrame を Glue 5.0 の Spark DataFrame に移行する Python および Scala の例です。

Python

変更前:

```
escaped_table_name= '`<dbname>`.`<table_name>`'

additional_options = {
  "query": f'select * from {escaped_table_name} WHERE column1 = 1 AND column7 = 7'
}

# DynamicFrame example
```

```
dataset = glueContext.create_data_frame_from_catalog(
    database="<dbname>",
    table_name=escaped_table_name,
    additional_options=additional_options)
```

変更後:

```
table_identifier= '`<catalogname>`.`<dbname>`.`<table_name>`"' #catalogname is optional

# DataFrame example
dataset = spark.sql(f'select * from {table_identifier} WHERE column1 = 1 AND column7 =
 7')
```

Scala

変更前:

```
val escapedTableName = "`<dbname>`.`<table_name>`"

val additionalOptions = JsonOptions(Map(
    "query" -> s"select * from $escapedTableName WHERE column1 = 1 AND column7 = 7"
    )
)

# DynamicFrame example
val datasource0 = glueContext.getCatalogSource(
    database="<dbname>",
    tableName=escapedTableName,
    additionalOptions=additionalOptions).getDataFrame()
```

変更後:

```
val tableIdentifier = "`<catalogname>`.`<dbname>`.`<table_name>`" //catalogname is
 optional

# DataFrame example
val datasource0 = spark.sql(s"select * from $tableIdentifier WHERE column1 = 1 AND
 column7 = 7")
```

考慮事項と制限事項

AWS Glue で Lake Formation を使用する場合は、次の考慮事項と制限事項を検討してください。

Lake Formation を使用した AWS Glue は、AWS GovCloud (米国東部) と AWS GovCloud (米国西部) を除く、サポートされているすべてのリージョンで使用できます。

- AWS Glue は、Apache Hive テーブルおよび Apache Iceberg テーブルに対してのみ Lake Formation を介したきめ細かなアクセスコントロールをサポートします。Apache Hive 形式には、Parquet、ORC、および CSV が含まれます。
- Lake Formation は Spark ジョブでのみ使用できます。
- Lake Formation を使用した AWS Glue は、ジョブ全体で 1 つの Spark セッションのみをサポートします。
- Lake Formation が有効になっている場合、1 つのシステムドライバー、システムエグゼキュター、1 つのユーザードライバー、およびオプションでユーザーエグゼキュター (ジョブに UDF または `spark.createDataFrame` がある場合に必要) が必要なため、AWS Glue にはより多くのワーカーが必要です。
- Lake Formation を使用した AWS Glue は、リソースリンクを介して共有されるクロスアカウントテーブルクエリのみをサポートします。resource-link には、ソースアカウントのリソースと同じ名前を付ける必要があります。
- Glue AWS ジョブのきめ細かなアクセスコントロールを有効にするには、`--enable-lakeformation-fine-grained-access` ジョブパラメータを渡します。
- AWS Glue マルチカタログ階層で動作するように AWS Glue ジョブを設定できます。AWS Glue `StartJobRun` API で使用する設定パラメータの詳細については、「Working with AWS Glue multi-catalog hierarchy on EMR Serverless」を参照してください。
- 次のサポートはありません。
  - 耐障害性のある分散データセット (RDD)
  - Spark ストリーミング
  - Lake Formation に付与されたアクセス許可による書き込み
  - ネストされた列のアクセスコントロール
- AWS Glue は、以下を含むシステムドライバーの完全な分離を損なわせる可能性のある機能をブロックします。
  - UDT、HiveUDF およびカスタムクラスを含むユーザー定義関数
  - カスタムデータソース
  - Spark 拡張機能、コネクタ、メタストア用の追加の jar の提供
  - `ANALYZE TABLE` コマンド

- アクセスコントロールを適用するために、EXPLAIN PLAN および DESCRIBE TABLE などの DDL オペレーションは、限定された情報を公開しません。
- AWS Glue は、Lake Formation 対応アプリケーションのシステムドライバー Spark ログへのアクセスを制限します。システムドライバーはより多くのアクセス権を使用して実行されるため、システムドライバーが生成するイベントとログには機密情報が含まれる可能性があります。許可されていないユーザーまたはコードがこの機密データにアクセスできないように、AWS Glue はシステムドライバーログへのアクセスを無効にしました。トラブルシューティングについては、AWS のサポートにお問い合わせください。
- テーブルの場所を Lake Formation に登録した場合、データアクセスパスは、AWS Glue ジョブのランタイムロールの IAM アクセス許可に関係なく、Lake Formation に保存された認証情報を経由します。テーブルの場所に登録されたロールを誤って設定すると、テーブルの場所に対して S3 IAM アクセス許可を持つロールを使用して送信されたジョブは失敗します。
- Lake Formation テーブルへの書き込みでは、Lake Formation に付与されたアクセス許可ではなく、IAM アクセス許可が使用されます。ジョブランタイムロールに必要な S3 アクセス許可がある場合は、それを使用して書き込みオペレーションを実行できます。

Apache Iceberg を使用する場合の考慮事項と制限事項を次に示します。

- Apache Iceberg はセッションカタログでのみ使用でき、任意の名前のカタログでは使用できません。
- Lake Formation に登録されている Iceberg テーブルは、メタデータテーブル `history`、`metadata_log_entries`、`snapshots`、`files`、`manifests`、`refs` のみをサポートします。AWSGlue は `partitions`、`path`、`summaries` などの機密データを持つ可能性のある列を非表示にします。この制限は、Lake Formation に登録されていない Iceberg テーブルには適用されません。
- Lake Formation に登録していないテーブルは、すべての Iceberg ストアドプロシージャをサポートしています。`register_table` プロシージャと `migrate` プロシージャは、どのテーブルでもサポートされていません。
- V1 の代わりに Iceberg DataFrameWriterV2 を使用することをお勧めします。

ワーカー割り当ての例

以下のパラメーターを使用して設定されたジョブの場合:

```
--enable-lakeformation-fine-grained-access=true
```

```
--number-of-workers=20
```

ワーカーの割り当ては次のとおりです。

- ユーザードライバーの 1 つのワーカー。
- システムドライバーの 1 つのワーカー。
- ユーザーエグゼキューター用に予約されている残りの 18 個のワーカー (つまり 2 個のワーカー) の 10%。
- システムエグゼキューターに割り当てられるワーカーは最大 16 個です。

自動スケーリングを有効にすると、ユーザーエグゼキューターは必要に応じてシステムエグゼキューターから未割り当ての容量を利用できます。

ユーザーエグゼキューターの割り当ての制御

次の設定を使用して、ユーザーエグゼキューターの予約率を調整できます。

```
--conf spark.dynamicAllocation.maxExecutorsRatio=<value between 0 and 1>
```

この設定では、使用可能なキャパシティの合計に対して予約されているユーザーエグゼキュターの数を微調整制御できます。

トラブルシューティング

トラブルシューティングソリューションについては、以下のセクションを参照してください。

ログ記録

AWS Glue は、Spark リソースプロファイルを使用してジョブ実行を分割します。AWSGlue は、ユーザープロファイルを使用して指定したコードを実行し、システムプロファイルは Lake Formation ポリシーを適用します。ユーザープロファイルとして実行されたタスクのログにアクセスできます。

Live UI と Spark 履歴サーバー

Live UI と Spark 履歴サーバーには、ユーザープロファイルから生成されたすべての Spark イベントと、システムドライバーから生成された編集済みイベントがあります。

[エグゼキュター] タブには、ユーザードライバーとシステムドライバーの両方からすべてのタスクが表示されます。ただし、ログリンクはユーザープロファイルでのみ使用できます。また、出力レコードの数など、一部の情報は Live UI から編集されます。

Lake Formation のアクセス許可が不十分なためジョブが失敗した

ジョブランタイムロールには、アクセスしているテーブルで SELECT と DESCRIBE を実行するアクセス許可があることを確認してください。

RDD の実行が失敗したジョブ

AWS Glue は現在、Lake Formation 対応ジョブで耐障害性のある分散データセット (RDD) オペレーションをサポートしていません。

Amazon S3 のデータファイルにアクセスできない

Lake Formation にデータレイクの場所が登録されていることを確認します。

セキュリティ検証の例外

AWS Glue がセキュリティ検証エラーを検出しました。AWS のサポートにご連絡ください。

アカウント間での AWS Glue データカタログとテーブルの共有

データベースとテーブルをアカウント間で共有し、Lake Formation を引き続き使用できます。詳細については、「Cross-account data sharing in Lake Formation」と「How do I share AWS Glue Data Catalog and tables cross-account using ?」を参照してください。

次の表は、ワークロードのきめ細かなアクセスコントロール (FGAC) とフルテーブルアクセス (FTA) を選択する方法の概要を説明しています。

機能	きめ細かなアクセス制御 (FGAC)	フルテーブルアクセス (FTA)
Access Level	Column/row level	Full table
Use Case	Queries and ETL with limited permissions	ETL
Performance Impact	Requires system/user space transitions for access control evaluation, adding latency	Optimized performance

AWS Glue での Amazon S3 Access Grants の使用

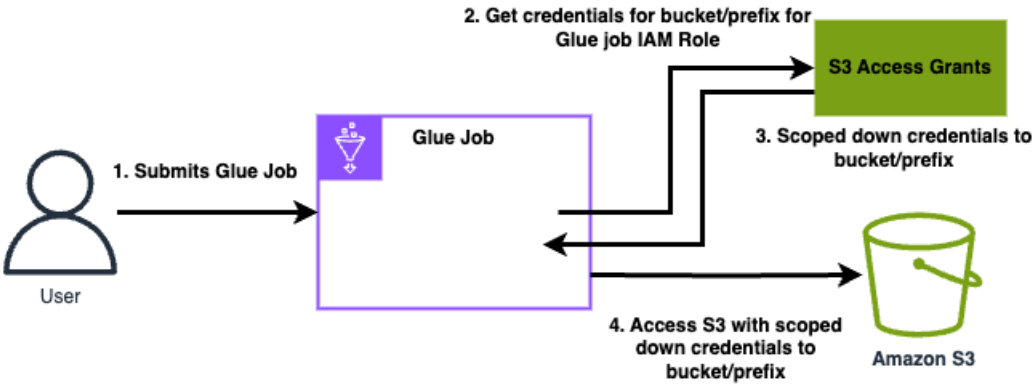
Glue バージョン 5.0 では、Amazon S3 Access Grants が提供するスケーラブルなアクセスコントロールソリューションにより、AWS Glue から Amazon S3 データへのアクセスが拡張されています。S3 データのアクセス許可設定が複雑または大規模である場合は、S3 Access Grants を使用して、ユーザーとロールの S3 データのアクセス許可をスケールできます。

S3 Access Grants を使用すると、AWS Glue ジョブにアクセスできる ID に関連付けられたランタイムロールまたは IAM ロールによって付与されるアクセス許可を超えたレベルまで、Amazon S3 データへのアクセスを拡張することができます。詳細については、「Amazon S3 ユーザーガイド」の「S3 Access Grants によるアクセス管理」を参照してください。

AWS Glue が S3 Access Grants と連携する仕組み

AWS Glue バージョン 5.0 以降では、S3 Access Grants がネイティブ統合されています。AWS Glue で S3 Access Grants を有効にし、Spark ジョブを実行することができます。Spark ジョブが S3 データをリクエストすると、Amazon S3 は特定のバケット、プレフィックス、またはオブジェクトを対象とする一時的な認証情報を提供します。

以下に、S3 Access Grants でアクセス管理されたデータに AWS Glue がアクセスする方法の概要を示します。



1. ユーザーが、Amazon S3 に保存されているデータを使用する AWS Glue Spark ジョブを送信します。
2. AWS Glue は、バケット、プレフィックス、またはオブジェクトへのアクセスを許可するユーザーの一時的な認証情報を S3 Access Grants に渡すようにリクエストします。

3. AWS Glue は一時的な認証情報を、そのユーザーの AWS セキュリティトークンサービス (STS) のトークン形式で返します。トークンのスコープは、S3 バケット、プレフィックス、またはオブジェクトへのアクセスを対象としています。
4. AWS Glue は STS トークンを使用して S3 からデータを取得します。
5. AWS Glue は S3 からデータを受け取り、結果をユーザーに返します。

AWS Glue での S3 Access Grants の使用に関する考慮事項

AWS Glue で S3 Access Grants を使用するときは、以下の動作と制限に注意してください。

機能のサポート

- S3 Access Grants は、AWS Glue バージョン 5.0 以降でサポートされています。
- AWS Glue で S3 Access Grants を使用する際にサポートされるジョブタイプは Spark のみです。
- AWS Glue で S3 Access Grants を使用する際にサポートされるオープンテーブル形式は、Delta Lake と Hudi だけです。
- 以下の機能は、S3 Access Grants との使用においてはサポートされません。
  - Apache Iceberg テーブル
  - IAM ロールを使用する Amazon S3 に対する AWS CLI のリクエスト
  - オープンソースの S3A プロトコルによる S3 アクセス

動作に関する注意事項

- AWS Glue は、ユーザーが Spark ジョブ内で同じ認証情報を繰り返しリクエストする必要がないように、認証情報キャッシュを提供します。そのため、AWS Glue は認証情報をリクエストするときは常にデフォルトレベルの権限を要求します。詳細については、「Amazon S3 ユーザーガイド」の「S3 データへのアクセスをリクエストする」を参照してください。

AWS Glue での S3 Access Grants の設定

前提条件

発信者または管理者が S3 Access Grants インスタンスを作成しました。

AWS Glue ポリシーとジョブ設定のセットアップ

AWS Glue で S3 Access Grants を設定するには、信頼ポリシーと IAM ポリシーを設定し、ジョブパラメータを介して設定を渡す必要があります。

1. 付与に使用されるロール (セッションまたはジョブを実行する AWS Glue ロール) に、以下の最小限の信頼ポリシーと IAM ポリシーを設定します。

   信頼ポリシー:

```
{
            "Sid": "Stmt1234567891011",
            "Action": [
                "sts:AssumeRole",
                "sts:SetSourceIdentity",
                "sts:SetContext"
            ],
            "Effect": "Allow",
            "Principal": {
                "Service": "access-grants.s3.amazonaws.com"
            },
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "123456789012",
                    "aws:SourceArn": "arn:aws:s3:<region>:123456789012:access-grants/
default"
                }
            }
        }
```

   IAM ポリシー:

```
{
            "Sid": "S3Grants",
            "Effect": "Allow",
            "Action": [
                "s3:GetDataAccess",
                "s3:GetAccessGrantsInstanceForPrefix"
            ],
            "Resource": "arn:aws:s3:<region>:123456789012:access-grants/default"
        },
        {
            "Sid": "BucketLevelReadPermissions",
            "Effect": "Allow",
```

```
            "Action": [
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::*"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:ResourceAccount": "123456789012"
                },
                "ArnEquals": {
                    "s3:AccessGrantsInstanceArn": [
                        "arn:aws:s3:<region>:123456789012:access-grants/default"
                    ]
                }
            }
        },
        {
            "Sid": "ObjectLevelReadPermissions",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion",
                "s3:GetObjectAcl",
                "s3:GetObjectVersionAcl",
                "s3:ListMultipartUploadParts"
            ],
            "Resource": [
                "arn:aws:s3:::*"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:ResourceAccount": "123456789012"
                },
                "ArnEquals": {
                    "s3:AccessGrantsInstanceArn": [
                        "arn:aws:s3:<region>:123456789012:access-grants/default"
                    ]
                }
            }
        }
```

2. AWS Glue ジョブで、AWS Glue ジョブパラメータまたは SparkConf を使用して次の Spark 設定を渡します。

```
--conf spark.hadoop.fs.s3.s3AccessGrants.enabled=true \
--conf spark.hadoop.fs.s3.s3AccessGrants.fallbackToIAM=false
```

AWS Glue でのログ記録とモニタリング

ETL (抽出、変換、ロード) ジョブの実行を自動化できます。AWS Glue は、モニタリングできるクローラとジョブのメトリクスを提供します。必要なメタデータを使用して AWS Glue Data Catalog を設定すると、AWS Glue は環境のヘルスに関する統計を提供します。クローラとジョブの呼び出しを、cron に基づく時間ベースのスケジュールで自動化することができます。イベントベースのトリガーが発生したときにジョブをトリガーすることもできます。

AWS Glue は、AWS Glue で ユーザー、ロール、または AWS のサービスによって実行されたアクションを記録するサービスである AWS CloudTrail と統合されています。証跡を作成する場合は、Amazon Simple Storage Service (Amazon S3) バケット、Amazon CloudWatch Logs、および Amazon CloudWatch Events への CloudTrail イベントの継続的な配信を有効にします。各イベントまたはログエントリには、リクエストの生成者に関する情報が含まれます。

Amazon CloudWatch Events を使用して、AWS のサービスを自動化し、アプリケーションの可用性の問題やリソースの変更などのシステムイベントに自動的に応答します。AWS のサービスからのイベントは、ほぼリアルタイムで CloudWatch Events に配信されます。簡単なルールを記述して、注目するイベントと、イベントがルールに一致した場合に自動的に実行するアクションを指定できます。

以下も参照してください。

- EventBridge を使用した AWS Glue の自動化
- CloudTrail のクロスアカウントロギング

クラウドのセキュリティの重要な側面に、ログ記録があります。クラウドインフラストラクチャのデバッグと保護に必要な情報をキャプチャしながら、シークレットや機密情報をキャプチャしない方法でログを設定する必要があります。ログに記録されているものを十分に理解してください。

AWS Glue のコンプライアンス検証

AWS のサービス が特定のコンプライアンスプログラムの対象であるかどうかを確認するには、コンプライアンスプログラムによる対象範囲内の AWS のサービス のサービスをご覧いただき、関心のあるコンプライアンスプログラムを選択してください。一般的な情報については、「AWSコンプライアンスプログラム」を参照してください。

AWS Artifact を使用して、サードパーティーの監査レポートをダウンロードできます。詳細については、「AWS Artifact でレポートをダウンロードする」を参照してください。

AWS のサービス を使用する際のユーザーのコンプライアンス責任は、ユーザーのデータの機密性や貴社のコンプライアンス目的、適用される法律および規制によって決まります。AWS では、コンプライアンスに役立つ次のリソースを提供しています。

- セキュリティのコンプライアンスとガバナンス – これらのソリューション実装ガイドでは、アーキテクチャ上の考慮事項について説明し、セキュリティとコンプライアンスの機能をデプロイする手順を示します。
- HIPAA 対象サービスのリファレンス – HIPAA 対象サービスのリストです。すべての AWS のサービスが HIPAA 適格であるわけではありません。
- 「AWS コンプライアンスのリソース」 – このワークブックおよびガイドのコレクションは、顧客の業界と拠点に適用されるものである場合があります。
- AWS Customer Compliance Guide - コンプライアンスの観点から見た責任共有モデルを理解できます。このガイドは、AWS のサービスを保護するためのベストプラクティスを要約したものであり、複数のフレームワーク (米国標準技術研究所 (NIST)、ペイメントカード業界セキュリティ標準評議会 (PCI)、国際標準化機構 (ISO) など) にわたるセキュリティ統制へのガイダンスがまとめられています。
- 「AWS Config デベロッパーガイド」の「ルールでのリソースの評価」 - AWS Config サービスは、自社のプラクティス、業界ガイドライン、および規制に対するリソースの設定の準拠状態を評価します。
- AWS Security Hub – この AWS のサービス は、AWS 内のセキュリティ状態の包括的なビューを提供します。Security Hub では、セキュリティコントロールを使用して AWS リソースを評価し、セキュリティ業界標準とベストプラクティスに対するコンプライアンスをチェックします。サポートされているサービスとコントロールの一覧については、Security Hub のコントロールリファレンスを参照してください。
- Amazon GuardDuty – この AWS のサービスは、環境をモニタリングして、疑わしいアクティビティや悪意のあるアクティビティがないか調べることで、AWS アカウント、ワークロード、コンテナ、データに対する潜在的な脅威を検出します。GuardDuty を使用すると、特定のコンプライ

アンスフレームワークで義務付けられている侵入検知要件を満たすことで、PCI DSS などのさまざまなコンプライアンス要件に対応できます。

- AWS Audit Manager - この AWS のサービスは、AWS の使用状況を継続的に監査して、リスクの管理方法や、規制および業界標準へのコンプライアンスの管理方法を簡素化するために役立ちます。

AWS Glue での耐障害性

AWS のグローバルインフラストラクチャは AWS リージョンとアベイラビリティーゾーンを中心に構築されます。AWSリージョンには、低レイテンシー、高いスループット、そして高度の冗長ネットワークで接続されている複数の物理的に独立し隔離されたアベイラビリティーゾーンがあります。アベイラビリティーゾーンでは、アベイラビリティーゾーン間で中断せずに、自動的にフェイルオーバーするアプリケーションとデータベースを設計および運用することができます。アベイラビリティーゾーンは、従来の単一または複数のデータセンターインフラストラクチャよりも可用性、耐障害性、およびスケーラビリティが優れています。

AWS リージョンとアベイラビリティーゾーンの詳細については、「AWS グローバルインフラストラクチャ」を参照してください。

AWS Glue ジョブの復元力の詳細については、「エラー: AWS Glue で VPC 間のフェイルオーバー動作」を参照してください。

AWS Glue 内のインフラストラクチャセキュリティ

マネージドサービスである AWS Glue は、Amazon Web Services のセキュリティプロセスの概要ホワイトペーパーに記載されている AWS グローバルネットワークセキュリティの手順で保護されています。

AWS 公開版 API コールを使用して、ネットワーク経由で AWS Glue にアクセスします。クライアントで Transport Layer Security (TLS) 1.0 以降がサポートされている必要があります。TLS 1.2 以降が推奨されています。また、Ephemeral Diffie-Hellman (DHE) や Elliptic Curve Ephemeral Diffie-Hellman (ECDHE) などの Perfect Forward Secrecy (PFS) を使用した暗号スイートもクライアントでサポートされている必要があります。これらのモードは、Java 7 以降など、最近のほとんどのシステムでサポートされています。

また、リクエストには、アクセスキー ID と、IAM プリンシパルに関連付けられているシークレットアクセスキーを使用して署名する必要があります。または、AWS Security Token ServiceAWS STS を使用して、一時的なセキュリティ認証情報を生成し、リクエストに署名することもできます。

トピック

- AWS Glue (AWS PrivateLink) のインターフェイス VPC エンドポイントの設定 (AWS PrivateLink)
- 共有 Amazon VPCs の設定

AWS Glue (AWS PrivateLink) のインターフェイス VPC エンドポイントの設定 (AWS PrivateLink)

VPC と AWS Glue とのプライベート接続を確立するには、インターフェイス VPC エンドポイントを作成します。インターフェイスエンドポイントは、インターネットゲートウェイ、NAT デバイス、VPN 接続、AWS Direct Connect 接続のいずれも必要とせずに AWS Glue API にプライベートにアクセスできるテクノロジーである AWS PrivateLink を利用しています。VPC のインスタンスは、パブリック IP アドレスがなくても AWS Glue API と通信できます。VPC と AWS Glue 間のトラフィックは、Amazon ネットワークを離れません。

各インターフェースエンドポイントは、サブネット内の 1 つ以上の Elastic Network Interface によって表されます。

詳細については、「Amazon VPC ユーザーガイド」の「インターフェイス VPC エンドポイント (AWS PrivateLink)」を参照してください。

AWS Glue VPC エンドポイントに関する考慮事項

AWS Glue のインターフェイスVPC エンドポイントを設定する前に、Amazon VPC ユーザーガイドのインターフェイスエンドポイントのプロパティと制限を確認してください。

AWS Glue は、VPC からのすべての API アクションの呼び出しをサポートしています。

AWS Glue 用のインターフェイス VPC エンドポイントの作成

AWS Glue サービス用の VPC エンドポイントは、Amazon VPC コンソールまたは AWS Command Line Interface (AWS CLI) を使用して作成できます。詳細については、「Amazon VPC ユーザーガイド」の「インターフェイスエンドポイントの作成」を参照してください。

AWS Glue 用の VPC エンドポイントは、以下のサービス名を使用して作成します。

- com.amazonaws.*region*glue

エンドポイントのプライベート DNS を有効にすると、リージョンのデフォルト DNS 名 (AWS Glue など) を使用して、`glue.us-east-1.amazonaws.com` への API リクエストを実行できます。

詳細については、「Amazon VPC ユーザーガイド」の「インターフェイスエンドポイントを介したサービスへのアクセス」を参照してください。

AWS Glue 用の VPC エンドポイントポリシーの作成

VPC エンドポイントには、AWS Glue へのアクセスを制御するエンドポイントポリシーをアタッチできます。このポリシーでは、以下の情報を指定します。

- アクションを実行できるプリンシパル。
- 実行可能なアクション。
- このアクションを実行できるリソース。

詳細については、「Amazon VPC ユーザーガイド」の「VPC エンドポイントでサービスへのアクセスを制御する」を参照してください。

例: AWS Glue の VPC エンドポイントポリシージョブの作成と更新を許可する

以下は、AWS Glue 用のエンドポイントポリシーの例です。エンドポイントにアタッチされると、このポリシーは、すべてのリソースですべてのプリンシパルに、リストされている AWS Glue アクションへのアクセス権を付与します。

```
{
   "Statement":[
      {
         "Principal":"*",
         "Effect":"Allow",
         "Action":[
             "glue:CreateJob",
             "glue:UpdateJob",
             "iam:PassRole"
         ],
         "Resource":"*"
      }
   ]
}
```

例: VPC エンドポイントポリシーでデータカタログアクセスを読み取り専用で許可する場合

以下は、AWS Glue 用のエンドポイントポリシーの例です。エンドポイントにアタッチされると、このポリシーは、すべてのリソースですべてのプリンシパルに、リストされている AWS Glue アクションへのアクセス権を付与します。

```
{
    "Statement": [
        {
            "Principal": "*",
            "Effect": "Allow",
            "Action": [
                "glue:GetDatabase",
                "glue:GetDatabases",
                "glue:GetTable",
                "glue:GetTables",
                "glue:GetTableVersion",
                "glue:GetTableVersions",
                "glue:GetPartition",
                "glue:GetPartitions",
                "glue:BatchGetPartition",
                "glue:SearchTables"
            ],
            "Resource": "*"
        }
    ]
}
```

共有 Amazon VPCs の設定

AWS Glue は、Amazon Virtual Private Cloud において、共有Virtual Private Cloud (VPC) をサポートしています。Amazon VPC の共有では、複数の AWS アカウントが、Amazon EC2 インスタンスや Amazon Relational Database Service (Amazon RDS) データベースなどのアプリケーションリソースを、共有の一元管理された Amazon VPC に作成できます。このモデルでは、VPC を所有するアカウント (所有者) は、 で同じ組織に属する他のアカウント (参加者) と 1 つまたは複数のサブネットを共有しますAWS Organizations サブネットが共有されると、参加者は共有しているサブネット内にある自分のアプリケーションリソースを表示、作成、変更、および削除できます。

AWS Glue で、共有サブネットとの接続を作成するには、アカウント内にセキュリティグループを作成し、そのセキュリティグループを共有サブネットにアタッチする必要があります。

詳細については、以下のトピックを参照してください。

- 詳細については、Amazon VPC ユーザーガイドの「Working with Shared VPCs」を参照してください。
- AWS Organizations ユーザーガイドの「What Is AWS Organizations? 」

AWS Glue のトラブルシューティング

AWS Glue の操作中に問題が発生した場合は、このセクションのトピックを参照してください。

トピック

- AWS Glue トラブルシューティング情報の収集
- エラーのトラブルシューティング Spark エラー
- クローラーが Lake Formation アクセス許可を使用している場合のクローラーエラー
- ログから AWS Glue for Ray エラーをトラブルシューティングする
- AWS Glue 機械学習の例外
- AWS Glue のクォータ

AWS Glue トラブルシューティング情報の収集

AWS Glue でエラーまたは予期しない動作が発生して AWS サポート に問い合わせる必要がある場合、名前、ID、および失敗したアクションに関連したログに関する情報をまず収集する必要があります。この情報が用意してあると、サポート は発生している問題を解決しやすくなります。

アカウント ID に加えて、次の失敗のタイプそれぞれについて以下の情報を収集します。

クローラが失敗した場合、以下の情報を収集します。

- クローラの名前

  クローラ実行からのログは `/aws-glue/crawlers` の下の CloudWatch Logs にあります。

テスト接続が失敗した場合、以下の情報を収集します。

- 接続名
- 接続 ID
- `jdbc:protocol://host:port/database-name` フォームの JDBC 接続文字列。

  テスト接続からのログは `/aws-glue/testconnection` の下の CloudWatch Logs にあります。

ジョブが失敗した場合、以下の情報を収集します。

- ジョブ名

- `jr_xxxxx` フォームのジョブ実行 ID。

  ジョブ実行からのログは `/aws-glue/jobs` の下の CloudWatch Logs にあります。

エラーのトラブルシューティング Spark エラー

AWS Glue でエラーが発生した場合は、次の情報を使用して問題の原因を突き止めて修正してください。

> **Note**
>
> AWS Glue GitHub リポジトリには AWS Glue に関するよくある質問のトラブルシューティングガイダンスの詳細が含まれています。

トピック

- エラー: リソースを利用できません。
- エラー: VPC の subnetId に S3 エンドポイントまたは NAT ゲートウェイが見つかりませんでした
- エラー: 必要なセキュリティグループのインバウンドルール
- エラー: 必要なセキュリティグループのアウトバウンドルール
- エラー: ジョブの実行に失敗しました、渡されたロールには AWS Glue サービスでロールを引き受けるためのアクセス許可が必要です
- エラー:DescribeVpcEndpoints アクションは許可されていません。VPC ID を検証できません vpc-id
- エラー:DescribeRouteTables アクションは許可されていません。VPC id (vpc-id) の Subnet Id (Subnet id) を検証できません。
- エラー: ec2:DescribeSubnets の呼び出しに失敗しました
- エラー: ec2:DescribeSecurityGroups の呼び出しに失敗しました
- エラー: AZ のサブネットが見つかりませんでした
- エラー: JDBC ターゲットへの書き込み時のジョブ実行の例外
- Error: Amazon S3: The operation is not valid for the object's storage class
- エラー:Amazon S3 タイムアウト
- エラー: Amazon S3 へのアクセスが拒否されました

- エラー: Amazon S3 のアクセスキー ID が存在しません
- エラー: URI に s3a:// を使用しながら Amazon S3 にアクセスするとジョブ実行が失敗します
- エラー: Amazon S3 のサービストークンが有効期限切れです
- エラー: ネットワークインターフェイスのプライベート DNS が見つかりません
- エラー: 開発エンドポイントのプロビジョニングに失敗しました
- エラー: ノートブックサーバー CREATE_FAILED
- エラー: ローカルノートブックの起動に失敗する
- エラー: クローラの実行に失敗しました
- エラー: パーティションが更新されませんでした
- Error: Job bookmark update failed due to version mismatch (エラー: バージョンの不一致のため、ジョブのブックマークの更新に失敗しました)
- エラー : ジョブのブックマークが有効なときにジョブがデータを再処理しています
- エラー: AWS Glue 内の VPC 間のフェイルオーバー動作

エラー: リソースを利用できません。

AWS Glue がリソース使用不可メッセージを返す場合は、エラーメッセージやログを表示して、問題の詳細を確認することができます。ここでは、トラブルシューティングするための一般的な方法について説明します。

- 使用する接続および開発エンドポイントについては、クラスターに Elastic Network Interface が不足していないかどうかを確認してください。

エラー: VPC の subnetId に S3 エンドポイントまたは NAT ゲートウェイが見つかりませんでした

問題の診断に役立つメッセージのサブネット ID と VPC ID を確認します。

- AWS Glue に必要な Amazon S3 VPC エンドポイントがセットアップ済みであることを確認してください。さらに、設定の一部である場合には、NAT ゲートウェイを確認します。詳細については、Amazon S3 用の VPC エンドポイント を参照してください。

エラー: 必要なセキュリティグループのインバウンドルール

少なくとも 1 つのセキュリティグループのすべての受信ポートを開く必要があります。トラフィックを制限するために、インバウンドルールのソースセキュリティグループを同じセキュリティグループに制限できます。

- 使用するすべての接続について、セキュリティグループで自己参照のインバウンドルールを確認します。詳細については、データストアへのネットワークアクセスを設定する を参照してください。
- 開発エンドポイントを使用している場合は、自己参照のインバウンドルールのセキュリティグループを確認します。詳細については、データストアへのネットワークアクセスを設定する を参照してください。

エラー: 必要なセキュリティグループのアウトバウンドルール

少なくとも 1 つのセキュリティグループのすべての発信ポートを開く必要があります。トラフィックを制限するために、アウトバウンドルールのソースセキュリティグループを同じセキュリティグループに制限できます。

- 使用するすべての接続について、セキュリティグループで自己参照のアウトバウンドルールを確認します。詳細については、データストアへのネットワークアクセスを設定する を参照してください。
- 開発エンドポイントを使用している場合は、自己参照のアウトバウンドルールのセキュリティグループを確認します。詳細については、データストアへのネットワークアクセスを設定する を参照してください。

エラー: ジョブの実行に失敗しました、渡されたロールには AWS Glue サービスでロールを引き受けるためのアクセス許可が必要です

ジョブを定義するユーザーは AWS Glue の `iam:PassRole` のアクセス権限を持っている必要があります。

- ユーザーが AWS Glue ジョブを作成するとき、そのユーザーのロールに AWS Glue の `iam:PassRole` を含むポリシーが含まれていることを確認します。詳細については、ステップ 3: AWS Glue にアクセスするユーザーまたはグループにポリシーをアタッチする を参照してください。

エラー:DescribeVpcEndpoints アクションは許可されていません。VPC ID を検証できません vpc-id

- `ec2:DescribeVpcEndpoints` アクセス権限用に、AWS Glue に渡されたポリシーを確認してください。

エラー:DescribeRouteTables アクションは許可されていません。VPC id (vpc-id) の Subnet Id (Subnet id) を検証できません。

- `ec2:DescribeRouteTables` アクセス権限用に、AWS Glue に渡されたポリシーを確認してください。

エラー: ec2:DescribeSubnets の呼び出しに失敗しました

- `ec2:DescribeSubnets` アクセス権限用に、AWS Glue に渡されたポリシーを確認してください。

エラー: ec2:DescribeSecurityGroups の呼び出しに失敗しました

- `ec2:DescribeSecurityGroups` アクセス権限用に、AWS Glue に渡されたポリシーを確認してください。

エラー: AZ のサブネットが見つかりませんでした

- アベイラビリティーゾーンは、AWS Glue では使用できない場合があります。メッセージで指定されているものとは異なるアベイラビリティーゾーンに新しいサブネットを作成して使用します。

エラー: JDBC ターゲットへの書き込み時のジョブ実行の例外

JDBC ターゲットに書き込むジョブを実行すると、以下のシナリオのようにジョブでエラーが発生する可能性があります。

- ジョブが Microsoft SQL Server テーブルに書き込む場合、テーブルに `Boolean` 型として定義された列がある場合には、SQL Server データベースでテーブルを事前に定義する必要があります。SQL Server ターゲットを使用して AWS Glue コンソールでジョブを定義し、[Create tables in

your data target (データターゲットでテーブルを作成する)] オプションを使用する場合は、データ型が Boolean であるターゲット列にソース列をマッピングしないでください。ジョブの実行時にエラーが発生する可能性があります。

次のようにして、エラーを回避できます。

- [Boolean (ブール)] 列を使用して既存のテーブルを選択します。
- ApplyMapping 変換を編集し、ソース内の [Boolean] (ブール) 列をターゲット内の数値または文字列にマップします。
- ApplyMapping 変換を編集して、[Boolean] (ブール) 列をソースから削除します。

- ジョブが Oracle テーブルに書き込む場合は、Oracle オブジェクトの名前の長さを調整する必要があります。Oracle の一部のバージョンでは、識別子の最大長は 128 バイトまたは 30 バイトに制限されています。この制限は、Oracle ターゲットデータストアのテーブル名および列名に影響を与えます。

  次のようにして、エラーを回避できます。

  - ご使用のバージョンの制限内で Oracle ターゲットテーブルに名前を付けます。
  - デフォルトの列名は、データのフィールド名から生成されます。列名が制限よりも長い場合は処理のために、ApplyMapping または RenameField 変換を使用して列の名前を制限内に変更します。

Error: Amazon S3: The operation is not valid for the object's storage class

AWS Glue がこのエラーを返した場合、AWS Glue ジョブは Amazon S3 ストレージクラス層にまたがるパーティションを持つテーブルからデータを読み取っていた可能性があります。

- ストレージクラスの除外を使用することで、これらのストレージクラス層にまたがるパーティションを持つテーブルで AWS Glue ジョブを確実に動作させることができます。除外しないと、これらの階層からデータを読み取るジョブは AmazonS3Exception: The operation is not valid for the object's storage class のエラーで失敗します。

  詳細については、「Amazon S3 ストレージクラスの除外」を参照してください。

エラー:Amazon S3 タイムアウト

AWS Glue が接続タイムアウトエラーを返す場合は、別の AWS リージョンの Amazon S3 バケットにアクセスしようとしている可能性があります。

- Amazon S3 の VPC エンドポイントは、AWS リージョン内のバケットにのみトラフィックをルーティングできます。他のリージョンのバケットに接続する必要がある場合、考えられる回避策は NAT ゲートウェイを使用することです。詳細については、「NAT ゲートウェイ」を参照してください。

エラー: Amazon S3 へのアクセスが拒否されました

AWS Glue が Amazon S3 バケットまたはオブジェクトへのアクセス拒否エラーを返す場合は、提供された IAM ロールにデータストアへのアクセス許可を持つポリシーがない可能性があります。

- ETL ジョブは、ソースまたはターゲットとして使用される Amazon S3 データストアにアクセスできる必要があります。クローラは、クロールする Amazon S3 データストアにアクセスできる必要があります。詳細については、ステップ 2: AWS Glue 用の IAM ロールを作成する を参照してください。

エラー: Amazon S3 のアクセスキー ID が存在しません

ジョブの実行中に AWS Glue がアクセスキー ID が存在しないというエラーを返す場合は、次のいずれかの理由が考えられます。

- ETL ジョブは IAM ロールを使用してデータストアにアクセスし、IAM ロールがそのジョブの開始前に削除されていないことを確認します。
- IAM ロールには、データストアにアクセスするためのアクセス許可が含まれており、アタッチ済みの `s3:ListBucket` を含む Amazon S3 ポリシーが正しいことを確認します。

エラー: URI に **s3a://** を使用しながら Amazon S3 にアクセスするとジョブ実行が失敗します

ジョブ実行から「XML ドキュメントをハンドラクラスで解析できませんでした」などのエラーが返された場合は、 `s3a://` URI を使用して数百のファイルをリストアップしようとして失敗した可能があります。代わりに `s3://` URI を使用してデータストアにアクセスしてください。次の例外トレースを調べてエラーを確認できます。

```
1. com.amazonaws.SdkClientException: Failed to parse XML document with handler class
 com.amazonaws.services.s3.model.transform.XmlResponsesSaxParser$ListBucketHandler
```

```
2. at
 com.amazonaws.services.s3.model.transform.XmlResponsesSaxParser.parseXmlInputStream(XmlRespons
3. at
 com.amazonaws.services.s3.model.transform.XmlResponsesSaxParser.parseListBucketObjectsResponse
4. at com.amazonaws.services.s3.model.transform.Unmarshallers
$ListObjectsUnmarshaller.unmarshall(Unmarshallers.java:70)
5. at com.amazonaws.services.s3.model.transform.Unmarshallers
$ListObjectsUnmarshaller.unmarshall(Unmarshallers.java:59)
6. at
 com.amazonaws.services.s3.internal.S3XmlResponseHandler.handle(S3XmlResponseHandler.java:62)
7. at
 com.amazonaws.services.s3.internal.S3XmlResponseHandler.handle(S3XmlResponseHandler.java:31)
8. at
 com.amazonaws.http.response.AwsResponseHandlerAdapter.handle(AwsResponseHandlerAdapter.java:70
9. at com.amazonaws.http.AmazonHttpClient
$RequestExecutor.handleResponse(AmazonHttpClient.java:1554)
10. at com.amazonaws.http.AmazonHttpClient
$RequestExecutor.executeOneRequest(AmazonHttpClient.java:1272)
11. at com.amazonaws.http.AmazonHttpClient
$RequestExecutor.executeHelper(AmazonHttpClient.java:1056)
12. at com.amazonaws.http.AmazonHttpClient
$RequestExecutor.doExecute(AmazonHttpClient.java:743)
13. at com.amazonaws.http.AmazonHttpClient
$RequestExecutor.executeWithTimer(AmazonHttpClient.java:717)
14. at com.amazonaws.http.AmazonHttpClient
$RequestExecutor.execute(AmazonHttpClient.java:699)
15. at com.amazonaws.http.AmazonHttpClient$RequestExecutor.access
$500(AmazonHttpClient.java:667)
16. at com.amazonaws.http.AmazonHttpClient
$RequestExecutionBuilderImpl.execute(AmazonHttpClient.java:649)
17. at com.amazonaws.http.AmazonHttpClient.execute(AmazonHttpClient.java:513)
18. at com.amazonaws.services.s3.AmazonS3Client.invoke(AmazonS3Client.java:4325)
19. at com.amazonaws.services.s3.AmazonS3Client.invoke(AmazonS3Client.java:4272)
20. at com.amazonaws.services.s3.AmazonS3Client.invoke(AmazonS3Client.java:4266)
21. at com.amazonaws.services.s3.AmazonS3Client.listObjects(AmazonS3Client.java:834)
22. at org.apache.hadoop.fs.s3a.S3AFileSystem.getFileStatus(S3AFileSystem.java:971)
23. at
 org.apache.hadoop.fs.s3a.S3AFileSystem.deleteUnnecessaryFakeDirectories(S3AFileSystem.java:115
24. at org.apache.hadoop.fs.s3a.S3AFileSystem.finishedWrite(S3AFileSystem.java:1144)
25. at org.apache.hadoop.fs.s3a.S3AOutputStream.close(S3AOutputStream.java:142)
26. at org.apache.hadoop.fs.FSDataOutputStream
$PositionCache.close(FSDataOutputStream.java:74)
27. at org.apache.hadoop.fs.FSDataOutputStream.close(FSDataOutputStream.java:108)
28. at org.apache.parquet.hadoop.ParquetFileWriter.end(ParquetFileWriter.java:467)
```

```
29. at
 org.apache.parquet.hadoop.InternalParquetRecordWriter.close(InternalParquetRecordWriter.java:1
30. at
 org.apache.parquet.hadoop.ParquetRecordWriter.close(ParquetRecordWriter.java:112)
31. at
 org.apache.spark.sql.execution.datasources.parquet.ParquetOutputWriter.close(ParquetOutputWrit
32. at org.apache.spark.sql.execution.datasources.FileFormatWriter
$SingleDirectoryWriteTask.releaseResources(FileFormatWriter.scala:252)
33. at org.apache.spark.sql.execution.datasources.FileFormatWriter$$anonfun
$org$apache$spark$sql$execution$datasources$FileFormatWriter$$executeTask
$3.apply(FileFormatWriter.scala:191)
34. at org.apache.spark.sql.execution.datasources.FileFormatWriter$$anonfun
$org$apache$spark$sql$execution$datasources$FileFormatWriter$$executeTask
$3.apply(FileFormatWriter.scala:188)
35. at org.apache.spark.util.Utils
$.tryWithSafeFinallyAndFailureCallbacks(Utils.scala:1341)
36. at org.apache.spark.sql.execution.datasources.FileFormatWriter$.org$apache$spark
$sql$execution$datasources$FileFormatWriter$$executeTask(FileFormatWriter.scala:193)
37. at org.apache.spark.sql.execution.datasources.FileFormatWriter$$anonfun$write$1$
$anonfun$3.apply(FileFormatWriter.scala:129)
38. at org.apache.spark.sql.execution.datasources.FileFormatWriter$$anonfun$write$1$
$anonfun$3.apply(FileFormatWriter.scala:128)
39. at org.apache.spark.scheduler.ResultTask.runTask(ResultTask.scala:87)
40. at org.apache.spark.scheduler.Task.run(Task.scala:99)
41. at org.apache.spark.executor.Executor$TaskRunner.run(Executor.scala:282)
42. at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1149)
43. at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624)
44. at java.lang.Thread.run(Thread.java:748)
```

エラー: Amazon S3 のサービストークンが有効期限切れです

Amazon Redshift との間でデータを移行する際には、1 時間で有効期限が切れる、Amazon S3 の一時的な認証情報が使用されます。長時間実行するジョブがある場合は、失敗する可能性があります。長時間実行するジョブを設定して Amazon Redshift との間でデータを移動する方法については、「aws-glue-programming-etl-connect-redshift-home」を参照してください。

エラー: ネットワークインターフェイスのプライベート DNS が見つかりません

ジョブが失敗したり、開発エンドポイントがプロビジョニングに失敗した場合は、ネットワーク設定が問題の原因である可能性があります。

- Amazon が提供する DNS を使用している場合は、`enableDnsHostnames` の値を true に設定する必要があります。詳細については、「DNS」を参照してください。

エラー: 開発エンドポイントのプロビジョニングに失敗しました

AWS Glue が開発エンドポイントのプロビジョニングに失敗した場合は、ネットワーク設定が問題の原因である可能性があります。

- 開発エンドポイントを定義すると、VPC、サブネット、およびセキュリティグループが検証され、特定の要件を満たしていることが確認されます。
- オプションの SSH パブリックキーを指定した場合、有効な SSH パブリックキーになっていることを確認します。
- VPC コンソールで、VPC が有効な [DHCP option set (DHCP オプションセット)] を使用していることを確認します。詳細については、「DHCP オプションセット」を参照してください。
- クラスターが PROVISIONING 状態のままになっている場合は、AWS サポート にお問い合わせください。

エラー: ノートブックサーバー CREATE_FAILED

AWS Glue が開発エンドポイント用のノートブックサーバーの作成に失敗した場合は、次のいずれかの問題が原因である可能性があります。

- AWS Glue は、ノートブックサーバーのセットアップ時に、Amazon EC2 に対し IAM ロールを渡します。IAM ロールでは、Amazon EC2 との信頼関係が必要です。
- IAM ロールには、同じ名前のインスタンスプロファイルが必要です。IAM コンソールで Amazon EC2 のロールを作成すると、同じ名前のインスタンスプロファイルが自動的に作成されます。無効なインスタンスプロファイル名　`iamInstanceProfile.name` に関するログのエラーをチェックします。詳細については、「インスタンスプロファイルの使用」を参照してください。
- ロールにノートブックサーバーを作成するために渡したポリシーの `aws-glue*` バケットにアクセスする権限があることを確認します。

エラー: ローカルノートブックの起動に失敗する

ローカルノートブックの起動に失敗し、ディレクトリまたはフォルダが見つからないというエラーが報告された場合は、次のいずれかの問題が原因である可能性があります。

- Microsoft Windows で実行している場合は、JAVA_HOME 環境変数が適切な Java ディレクトリを指していることを確認します。この変数を更新せずに Java が更新される可能性があります。存在しなくなったフォルダを変数が指していると、Jupyter Notebook は起動に失敗します。

エラー: クローラの実行に失敗しました

AWS Glue がデータをカタログ化するためにクローラを正常に実行できなかった場合は、次のいずれかの理由が考えられます。まず、AWS Glue コンソールのクローラリストにエラーがあるかどうかを確認します。クローラ名の横に感嘆符 アイコンがあるかどうかを確認し、アイコンの上にカーソルを置いて関連するメッセージを確認します。

- /aws-glue/crawlers にある CloudWatch Logs で実行されるクローラのログを確認します。

エラー: パーティションが更新されませんでした

ETL ジョブの実行時に Data Catalog でパーティションが更新されなかった場合、CloudWatch の口グにある DataSink クラスの次のログステートメントが役に立ちます。

- 「Attempting to fast-forward updates to the Catalog - nameSpace:」 — このジョブによって変更しようとしているデータベース、テーブル、および catalogId を示します。このステートメントがここにない場合は、enableUpdateCatalog が true に設定され、getSink() パラメータとして、または　additional_options で正しく渡されるかどうかを確認します。
- 「Schema change policy behavior:」 — 渡したスキーマ　updateBehavior 値を示します。
- 「Schemas qualify (schema compare):」 — true または false になります。
- 「Schemas qualify (case-insensitive compare):」 — true または false になります。
- 両方が false であり、updateBehavior に　UPDATE_IN_DATABASE が設定されていない場合は、DynamicFrame スキーマが同一であるか、このスキーマには、Data Catalog テーブルスキーマに表示される列のサブセットが含まれている必要があります。

パーティションの更新の詳細については、「AWS Glue ETL ジョブを使用してデータカタログのスキーマを更新し、新規パーティションを追加する」を参照してください。

Error: Job bookmark update failed due to version mismatch (エラー: バージョンの不一致のため、ジョブのブックマークの更新に失敗しました)

AWS Glue ジョブをパラメータ化して、Amazon S3 の異なるデータセットに同じ変換/ロジックを適用しようとしている可能性があります。ユーザーは指定の場所で処理されたファイルを追跡したいと考えています。同じソースバケットで同じジョブを実行し、同じ/異なる送信先に同時に書き込んだ場合 (同時実行数 >1)、次のエラーによりジョブは失敗します。

```
py4j.protocol.Py4JJavaError: An error occurred while
 callingz:com.amazonaws.services.glue.util.Job.commit.:com.amazonaws.services.gluejobexecutor.m
 Continuation update failed due to version mismatch. Expected version 2 but found
 version 3
```

解決策として、同時実行数を 1 に設定するか、ジョブを同時に実行しないようにします。

現在、AWS Glue のブックマークは同時ジョブ実行をサポートしておらず、コミットは失敗します。

エラー : ジョブのブックマークが有効なときにジョブがデータを再処理しています

AWS Glue ジョブのブックマークを有効にしてあっても、以前の実行で処理したデータを ETL ジョブで再処理する場合があります。このエラーの一般的な原因を確認します。

最大同時実行数

ジョブの同時実行の最大数をデフォルト値の 1 より大きく設定すると、ジョブのブックマークを妨げる可能性があります。ジョブのブックマークがオブジェクトの最終更新日時を確認し、どのオブジェクトを再処理する必要があるか確認するときに発生します。詳細については、「AWS Glue で Spark ジョブに関するジョブプロパティの構成」の最大同時実行数に関する説明を参照してください。

Job オブジェクトが見つからない

ジョブの実行スクリプトが次のコメントで終わることを確認します。

```
job.commit()
```

このオブジェクトを含めると、AWS Glue はジョブ実行のタイムスタンプとパスを記録します。同じパスでジョブを再度実行すると、AWS Glue は新しいファイルのみを処理します。このオブジェクト

を含まず、ジョブのブックマークが有効になっている場合、ジョブはすでに処理されたファイルと共に新しいファイルを再処理して、ジョブのターゲットデータストアで冗長化されます。

変換コンテキストパラメータが見つからない

変換コンテキストは、GlueContext クラスのオプションのパラメータですが、ジョブのブックマークは含めなければ機能しません。このエラーを解決するには、以下に示すように DynamicFrame を作成するときに変換コンテキストパラメータを追加します。

```
sample_dynF=create_dynamic_frame_from_catalog(database,
 table_name,transformation_ctx="sample_dynF")
```

入力ソース

入力ソースのリレーショナルデータベース (JDBC 接続) を使用している場合は、テーブルのプライマリキーが順番になっている場合のみジョブのブックマークが機能します。ジョブのブックマークは新しい行に対して機能しますが、更新された行に対しては機能しません。これは、ジョブのブックマークで探すプライマリキーがすでに存在しているからです。これは、入力ソースが Amazon Simple Storage Service (Amazon S3) である場合には適用されません。

最終更新日時

Amazon S3 入力ソースの場合、ジョブのブックマークは、ファイル名ではなくオブジェクトの最終更新日時を確認して、どのオブジェクトを再処理する必要があるのかを確認します。入力ソースデータが最後のジョブ実行以降に変更されている場合、ジョブを再度実行すると、ファイルが再度処理されます。

エラー: AWS Glue 内の VPC 間のフェイルオーバー動作

次のプロセスは、AWS Glue 4.0 以前のバージョンにおけるジョブのフェイルオーバーに使用されます。

概要: AWS Glue 接続は、ジョブ実行の送信時に選択されます。ジョブ実行で何らかの問題 (IP アドレスの不足、ソースへの接続、ルーティングの問題) が発生した場合、ジョブ実行は失敗します。再試行が設定されている場合、AWS Glue は同じ接続で再度実行を試みます。

1. 実行の試行ごとに、AWS Glue は、使用できる接続が見つかるまで、ジョブ設定に記載されている順序で接続の状態をチェックします。アベイラビリティーゾーン (AZ) に障害が発生した場合、その AZ からの接続はチェックに失敗し、スキップされます。
2. AWS Glue は次のことを実行して接続を検証します。

- 有効な Amazon VPC ID とサブネットを確認する。
- NAT ゲートウェイまたは Amazon VPC エンドポイントが存在することを確認する。
- サブネットに 0 個を超える IP アドレスが割り当てられていることを確認する。
- AZ が正常であることを確認する。

AWS Glue は、ジョブ実行の送信時に接続を確認できません。

3. Amazon VPC を使用するジョブの場合、すべてのドライバーとエグゼキューターは、ジョブ実行の送信時に選択された接続を使用して同じ AZ に作成されます。
4. 再試行が設定されている場合、AWS Glue は同じ接続で再度実行を試みます。これは、この接続に関する問題が長時間継続することを保証できないためです。AZ に障害が発生すると、その AZ 内の既存のジョブ実行 (ジョブ実行の段階によって異なる) が失敗する可能性があります。再試行すると AZ の障害が検出され、新しい実行で別の AZ が選択されるはずです。

クローラーが Lake Formation アクセス許可を使用している場合のクローラーエラー

以下の情報を使用して、Lake Formation の認証情報を使用してクローラーを設定する際のさまざまな問題を診断し、修正します。

エラー: 「The S3 location: s3://examplepath is not registered」(S3 ロケーション: s3://examplepath は登録されていません)

Lake Formation の認証情報を使用してクローラーを実行する場合は、最初に Lake Formation のアクセス許可を設定する必要があります。このエラーを解決するには、ターゲットの Amazon S3 ロケーションを Lake Formation で登録してください。詳細については、「Amazon S3 ロケーションの登録」を参照してください。

エラー: 「User/Role is not authorized to perform: lakeformation:GetDataAccess on resource」(ユーザー/ロールは、リソースに対する lakeformation:GetDataAccess の実行が許可されていません)

IAM コンソールまたは AWS CLI を使用して、クローラーロールに `lakeformation:GetDataAccess` へのアクセス許可を追加してください。この許可があると、Lake Formation がデータにアクセスするための一時的な認証情報のリクエストを承諾します。以下のポリシーを参照してください。

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

エラー: 「Insufficient Lake Formation permission(s) on (Database name: exampleDatabase, Table Name: exampleTable)」(データベース名: exampleDatabase、テーブル名: exampleTable に対する Lake Formation へのアクセス許可が不十分です)

Lake Formation コンソール (https://console.aws.amazon.com/lakeformation/) で、出力データベースとして指定されたデータベースに対するクローラーロールへのアクセス許可 (`Create`、`Describe`、`Alter`) を付与します。テーブルに対するアクセス許可を付与することもできます。詳細については、「名前付きリソース方式を使用した Data Catalog 許可の付与」を参照してください。

エラー: 「Insufficient Lake Formation permission(s) on s3://examplepaths」(s3://examplepath に対する Lake Formation へのアクセス許可が不十分です)

1. クロスアカウントでのクローリング
   a. Amazon S3 バケットが登録されているアカウント (アカウント B) を使用して、Lake Formation コンソール (https://console.aws.amazon.com/lakeformation/) にログインします。クローラーを実行するアカウント (アカウント A) に、データロケーションへのアクセス許可を付与します。これにより、クローラーはターゲットの Amazon S3 ロケーションからデータを読み取ることができます。
   b. クローラーが作成されたアカウント (アカウント A) で、クローラーの実行に使用される IAM ロールに、ターゲットの Amazon S3 ロケーションのデータロケーションへのアクセス許可を付与し、クローラーが Lake Formation の送信先からデータを読み取れるようにします。詳細については、「データロケーション許可の付与 (外部アカウント)」を参照してください。

2. アカウント内 (クローラーと登録されている Amazon S3 ロケーションが同じアカウントにある) クローリング - Amazon S3 ロケーションで実行されるクローラーに使用される IAM ロールにデータロケーションへのアクセス許可を付与し、クローラーが Lake Formation のターゲットからデータを読み取れるようにします。詳細については、「データロケーション許可の付与 (同じアカウント)」を参照してください。

Lake Formation の認証情報を使用したクローラーの設定に関するよくある質問

1. AWS コンソールを使用して、Lake Formation の認証情報を使用して実行できるようにクローラーを設定するにはどうすればよいですか。

   AWS Glue コンソール (https://console.aws.amazon.com/glue/) で、クローラーを設定する際に、オプション [Use Lake Formation credentials for crawling Amazon S3 data source] (Amazon S3 データソースのクローリングに Lake Formation の認証情報を使用する) を選択してください。クロスアカウントクローリングの場合は、ターゲットの Amazon S3 ロケーションが Lake Formation で登録されている AWS アカウント ID を指定します。アカウント内クローリングでは、accountId フィールドはオプションです。

2. AWS CLI を使用して、Lake Formation の認証情報を使用できるようにクローラーを設定するにはどうすればよいですか。

   `CreateCrawler` API コール中に `LakeFormationConfiguration` を追加します。

   ```
   "LakeFormationConfiguration": {
       "UseLakeFormationCredentials": true,
       "AccountId": "111111111111" (AWS account ID where the target Amazon S3 location
    is registered with Lake Formation)
     }
   ```

3. Lake Formation の認証情報を使用するクローラーでサポートされているターゲットは何ですか。

   Lake Formation の認証情報を使用するクローラーは、Amazon S3 (アカウント内およびクロスアカウントのクローリング) と、アカウント内のデータカタログターゲット (基盤となるロケーションが Amazon S3 である)、および Apache Iceberg ターゲットでのみサポートされています。

4. Lake Formation の認証情報を使用して、複数の Amazon S3 バケットを単一のクローラーの一部としてクローリングできますか。

いいえ。Lake Formation の認証情報供給を使用するターゲットをクローリングするには、基盤となる Amazon S3 ロケーションが同じバケットに属している必要があります。例えば、複数のお客様が同じバケット (bucket1) に属している場合は、複数のターゲットロケーション (`s3://bucket1/folder1, s3://bucket1/folder2`) を使用できます。別のバケット (s3://bucket1/folder1、s3://bucket2/folder2) の指定はサポートされていません。

ログから AWS Glue for Ray エラーをトラブルシューティングする

AWS Glue では、ジョブの実行中に Ray プロセスが生成するログにアクセス可能です。Ray ジョブでエラーや予期しない動作が発生した場合は、まずログから情報を収集して失敗の原因を特定します。また、インタラクティブセッションについても同様のログが提供されています。セッションのログは、先頭に `/aws-glue/ray/sessions` が付きます。

ジョブの実行時、ログの行はリアルタイムで CloudWatch に送信されます。実行が完了すると、Print ステートメントが CloudWatch ログに追加されます。ログはジョブの実行後 2 週間保持されます。

Ray ジョブのログを検査する

ジョブが失敗した場合は、ジョブ名とジョブ実行 ID を収集します。これらは AWS Glue コンソールで確認できます。ジョブページに移動し、[Runs] (実行) タブに移動します。Ray ジョブのログは、次の専用の CloudWatch ロググループに保存されます。

- `/aws-glue/ray/jobs/script-log/` — メインの Ray スクリプトが生成するログを保存します。
- `/aws-glue/ray/jobs/ray-monitor-log/` — Ray のオートスケーラープロセスが生成するログを保存します。これらのログはヘッドノードについて生成され、他のワーカーノードでは生成されません。
- `/aws-glue/ray/jobs/ray-gcs-logs/` – GCS (グローバルコントロールストア) プロセスから出力されたログを保存します。これらのログはヘッドノードについて生成され、他のワーカーノードでは生成されません。
- `/aws-glue/ray/jobs/ray-process-logs/` – ヘッドノードで実行されている他の Ray プロセス (主にダッシュボードエージェント) から出力されたログを保存します。これらのログはヘッドノードについて生成され、他のワーカーノードでは生成されません。

- `/aws-glue/ray/jobs/ray-raylet-logs/` – 各 raylet プロセスから出力されたログを保存します。これらのログは、ワーカーノード (ヘッドノードを含む) ごとに 1 つのストリームにまとめられます。
- `/aws-glue/ray/jobs/ray-worker-out-logs/` – クラスター内の各ワーカーの stdout ログを保存します。これらのログは、ワーカーノード (ヘッドノードを含む) ごとに生成されます。
- `/aws-glue/ray/jobs/ray-worker-err-logs/` – クラスター内の各ワーカーの stderr ログを保存します。これらのログは、ワーカーノード (ヘッドノードを含む) ごとに生成されます。
- `/aws-glue/ray/jobs/ray-runtime-env-log/` – Ray のセットアッププロセスに関するログを保存します。これらのログは、ワーカーノード (ヘッドノードを含む) ごとに生成されます。

Ray ジョブのエラーをトラブルシューティングする

Ray のロググループの構成を理解し、エラーのトラブルシューティングの助けになるロググループを見つけるには、Ray のアーキテクチャに関する背景情報があると役立ちます。

AWS Glue ETL では、ワーカーはインスタンスに対応します。AWS Glue ジョブのワーカーを設定する際は、そのジョブ専用のインスタンスタイプおよびインスタンス数を設定します。Ray では、ワーカーという単語がさまざまな形で使われています。

Ray は、ヘッドノードとワーカーノードを使用して、Ray クラスター内のインスタンスの役割を区別します。Ray のワーカーノードは、分散計算の結果を得るため計算を実行する、複数のアクタープロセスをホストできます。関数のレプリカを実行するアクターはレプリカと呼ばれます。レプリカのアクターはワーカープロセスとも呼ばれます。レプリカはヘッドノードでも実行できます。ヘッドノードは、追加のプロセスを実行してクラスターを調整することから、ヘッドと呼ばれます。

計算に関与する各アクターは、独自のログストリームを生成します。これにより、いくつかのインサイトが得られます。

- ログを生成するプロセスの数は、ジョブに割り当てられるワーカーの数よりも多い場合があります。多くの場合、各インスタンスのコアには 1 つのアクターがあります。
- Ray のヘッドノードは、クラスターの管理および起動ログを生成します。それに対して Ray のワーカーノードは、そのノードで実行される処理のログのみを生成します。

Ray のアーキテクチャの詳細については、Ray のドキュメントの「Architecture Whitepapers」(アーキテクチャホワイトペーパー) を参照してください。

問題の領域: Amazon S3 へのアクセス

ジョブ実行のエラーメッセージを確認します。それでも十分な情報が得られない場合は、/aws-glue/ray/jobs/script-log/ を確認します。

問題領域: PIP の依存性の管理

/aws-glue/ray/jobs/ray-runtime-env-log/ をチェックします。

問題の領域: メインプロセスにおける中間値の検査

メインスクリプトから stderr または stdout に書き込みを行います。また、/aws-glue/ray/jobs/script-log/ からログを取得します。

問題の領域: 子プロセスにおける中間値の検査

remote 関数から stderr または stdout に書き込みを行います。その後、/aws-glue/ray/jobs/ray-worker-out-logs/ または /aws-glue/ray/jobs/ray-worker-err-logs/ からログを取得します。関数はどのレプリカでも実行されている可能性があるため、目的の出力を見つけるには複数のログを調べなければならない場合があります。

問題の領域: エラーメッセージ内の IP アドレスの解釈

特定のエラー状況では、ジョブは IP アドレスを含むエラーメッセージを出力することがあります。これらの IP アドレスは一時的なもので、クラスターがノードを識別し、ノード間で通信するために使用されます。ノードのログは、IP アドレスに基づく一意のサフィックスが付いたログストリームに発行されます。

CloudWatch では、このサフィックスを識別することでログを絞り込み、この IP アドレス固有のログを検査できます。例えば、*FAILED_IP* と *JOB_RUN_ID* が指定されている場合、サフィックスを以下のように識別できます。

```
filter @logStream like /JOB_RUN_ID/
| filter @message like /IP-/
| parse @message "IP-[*]" as ip
| filter ip like /FAILED_IP/
| fields replace(ip, ":", "_") as uIP
| stats count_distinct by uIP as logStreamSuffix
| display logStreamSuffix
```

AWS Glue 機械学習の例外

このトピックでは、機械学習に関連する AWS Glue 例外の HTTP エラーコードとエラー文字列について説明します。機械学習アクティビティごとに、オペレーションの実行時に発生する可能性があるエラーコードとエラー文字列が表示されます。また、エラーが発生したオペレーションを再試行できるかどうかも確認できます。

CancelMLTaskRunActivity

このアクティビティには、次の例外があります。

- EntityNotFoundException (400)
  - 「ハンドル [transformName] を持つアカウント [accountId] に ML 変換が見つかりません。」
  - 「[taskRunId] の ML タスク実行が見つかりません: 変換 [transformName] のアカウント [accountId] 内。」

  再試行してもいいですか: いいえ。

CreateMLTaskRunActivity

このアクティビティには、次の例外があります。

- InvalidInputException (400)
  - 「予期しない入力による内部サービスエラー。」
  - 「AWS Glue テーブルの入力ソースを変換に指定する必要があります。」
  - 「入力ソース列 [columnName] で、カタログに無効なデータ型が定義されています。」
  - 「入力レコードテーブルは 1 つだけ指定する必要があります。」
  - 「データベース名を指定する必要があります。」
  - 「テーブル名を指定する必要があります。」
  - 「スキーマが変換に定義されていません。」
  - 「スキーマには、特定の主キー [PrimaryKey] を含める必要があります。」
  - 「データカタログスキーマの取得中に問題が発生しました: [message]。」
  - 「最大容量とワーカー番号/タイプを同時に設定することはできません。」
  - 「WorkerType と NumberOfWorkers の両方を設定する必要があります。」
  - 「MaxCapacity は >= [maxCapacity] にする必要があります。」

- 「NumberOfWorkers は >= [maxCapacity] にする必要があります。」
- 「最大再試行数は負以外にする必要があります。」
- 「一致検索パラメータが設定されていません。」
- 「一致検索パラメータに主キーを指定する必要があります。」

再試行してもいいですか: いいえ。

- AlreadyExistsException (400)
  - 「[transformName] という名前の変換はすでに存在します。」

再試行してもいいですか: いいえ。

- IdempotentParameterMismatchException (400)
  - 「変換 [transformName] のべき等作成リクエストにパラメータの不一致がありました。」

再試行してもいいですか: いいえ。

- InternalServiceException (500)
  - 「依存関係エラー。」

再試行してもいいですか: はい。

- ResourceNumberLimitExceededException (400)
  - 「ML 変換数 ([count]) が [limit] 変換数の制限を超えています。」

再試行してもいいですか: はい。変換を削除して、この新しい変換用のスペースを作成してから再試行できます。

DeleteMLTransformActivity

このアクティビティには、次の例外があります。

- EntityNotFoundException (400)
  - 「ハンドル [transformName] を持つアカウント [accountId] に ML 変換が見つかりません」

再試行してもいいですか: いいえ。

GetMLTaskRunActivity

このアクティビティには、次の例外があります。

- EntityNotFoundException (400)
  - 「ハンドル [transformName] を持つアカウント [accountId] に ML 変換が見つかりません。」
  - 「[taskRunId] の ML タスク実行が見つかりません: 変換 [transformName] のアカウント [accountId] 内。」

  再試行してもいいですか: いいえ。

GetMLTaskRunsActivity

このアクティビティには、次の例外があります。

- EntityNotFoundException (400)
  - 「ハンドル [transformName] を持つアカウント [accountId] に ML 変換が見つかりません。」
  - 「[taskRunId] の ML タスク実行が見つかりません: 変換 [transformName] のアカウント [accountId] 内。」

  再試行してもいいですか: いいえ。

GetMLTransformActivity

このアクティビティには、次の例外があります。

- EntityNotFoundException (400)
  - 「ハンドル [transformName] を持つアカウント [accountId] に ML 変換が見つかりません。」

  再試行してもいいですか: いいえ。

GetMLTransformsActivity

このアクティビティには、次の例外があります。

- EntityNotFoundException (400)
  - 「ハンドル [transformName] を持つアカウント [accountId] に ML 変換が見つかりません。」

  再試行してもいいですか: いいえ。
- InvalidInputException (400)
  - 「アカウント ID を空白にすることはできません。」

- 「列 [column] での並べ替えはサポートされていません。」
- 「[column] を空白にすることはできません。」
- 「予期しない入力による内部サービスエラー。」

再試行してもいいですか: いいえ。

GetSaveLocationForTransformArtifactActivity

このアクティビティには、次の例外があります。

- EntityNotFoundException (400)
  - 「ハンドル [transformName] を持つアカウント [accountId] に ML 変換が見つかりません。」

  再試行してもいいですか: いいえ。
- InvalidInputException (400)
  - 「サポートされていないアーティファクトタイプ [artifactType]。」
  - 「予期しない入力による内部サービスエラー。」

  再試行してもいいですか: いいえ。

GetTaskRunArtifactActivity

このアクティビティには、次の例外があります。

- EntityNotFoundException (400)
  - 「ハンドル [transformName] を持つアカウント [accountId] に ML 変換が見つかりません。」
  - 「[taskRunId] の ML タスク実行が見つかりません: 変換 [transformName] のアカウント [accountId] 内。」

  再試行してもいいですか: いいえ。
- InvalidInputException (400)
  - 「ファイル名 '[fileName]' は発行に対して無効です。」
  - 「[taskType] タスクタイプのアーティファクトを取得できません。」
  - 「[artifactType] のアーティファクトを取得できません。」
  - 「予期しない入力による内部サービスエラー。」

再試行してもいいですか: いいえ。

PublishMLTransformModelActivity

このアクティビティには、次の例外があります。

- EntityNotFoundException (400)
  - 「ハンドル [transformName] を持つアカウント [accountId] に ML 変換が見つかりません。」
  - 「アカウント ID - [accountId] - および変換 ID - [transformId] で、バージョン - [version] を持つ既存のモデルが見つかりません。」

  再試行してもいいですか: いいえ。
- InvalidInputException (400)
  - 「ファイル名 '[fileName]' は発行に対して無効です。」
  - 「符号なし文字列 [string] の先頭のマイナス記号が不正です。」
  - 「[string] の末尾に不正な数字があります。」
  - 「文字列値 [string] が符号なし長整数の範囲を超えています。」
  - 「予期しない入力による内部サービスエラー。」

  再試行してもいいですか: いいえ。

PullLatestMLTransformModelActivity

このアクティビティには、次の例外があります。

- EntityNotFoundException (400)
  - 「ハンドル [transformName] を持つアカウント [accountId] に ML 変換が見つかりません。」

  再試行してもいいですか: いいえ。
- InvalidInputException (400)
  - 「予期しない入力による内部サービスエラー。」

  再試行してもいいですか: いいえ。
- ConcurrentModificationException (400)

- 「競合する挿入のパラメータが一致しないため、トレーニング対象のモデルバージョンを作成できません。」
- 「変換 ID [transformId] の ML 変換モデルが古いか、別のプロセスによって更新中です。再試行してください。」

再試行してもいいですか: はい。

PutJobMetadataForMLTransformActivity

このアクティビティには、次の例外があります。

- EntityNotFoundException (400)
  - 「ハンドル [transformName] を持つアカウント [accountId] に ML 変換が見つかりません。」
  - 「[taskRunId] の ML タスク実行が見つかりません: 変換 [transformName] のアカウント [accountId] 内。」

  再試行してもいいですか: いいえ。
- InvalidInputException (400)
  - 「予期しない入力による内部サービスエラー。」
  - 「不明なジョブメタデータタイプ [jobType]。」
  - 「更新するタスク実行 ID を指定する必要があります。」

  再試行してもいいですか: いいえ。

StartExportLabelsTaskRunActivity

このアクティビティには、次の例外があります。

- EntityNotFoundException (400)
  - 「ハンドル [transformName] を持つアカウント [accountId] に ML 変換が見つかりません。」
  - 「アカウント ID [accountId] に変換 ID [transformId] のラベルセットがありません。」

  再試行してもいいですか: いいえ。
- InvalidInputException (400)
  - 「[message]。」

- 「指定した S3 パスは、変換と同じリージョン内にありません。」 リージョン - [region] が必要ですが、- [region] が提供されました。」

再試行してもいいですか: いいえ。

StartImportLabelsTaskRunActivity

このアクティビティには、次の例外があります。

- EntityNotFoundException (400)
  - 「ハンドル [transformName] を持つアカウント [accountId] に ML 変換が見つかりません。」

  再試行してもいいですか: いいえ。
- InvalidInputException (400)
  - 「[message]。」
  - 「無効なラベルファイルパスです。」
  - 「[labelPath] のラベルファイルにアクセスできません。[message]。」
  - 「変換で提供された IAM ロールを使用できません。ロール: [role]。」
  - 「サイズ 0 の無効なラベルファイル。」
  - 「指定した S3 パスは、変換と同じリージョン内にありません。」 リージョン - [region] が必要ですが、- [region] が提供されました。」

  再試行してもいいですか: いいえ。
- ResourceNumberLimitExceededException (400)
  - 「ラベルファイルが [limit] MB の制限を超えています。」

  再試行してもいいですか: いいえ。ラベルファイルをいくつかの小さなファイルに分割することを検討してください。

StartMLEvaluationTaskRunActivity

このアクティビティには、次の例外があります。

- EntityNotFoundException (400)
  - 「ハンドル [transformName] を持つアカウント [accountId] に ML 変換が見つかりません。」

再試行してもいいですか: いいえ。

- InvalidInputException (400)
  - 「入力レコードテーブルは 1 つだけ指定する必要があります。」
  - 「データベース名を指定する必要があります。」
  - 「テーブル名を指定する必要があります。」
  - 「一致検索パラメータが設定されていません。」
  - 「一致検索パラメータに主キーを指定する必要があります。」

  再試行してもいいですか: いいえ。
- MLTransformNotReadyException (400)
  - 「このオペレーションは READY 状態の変換にのみ適用できます。」

  再試行してもいいですか: いいえ。
- InternalServiceException (500)
  - 「依存関係エラー。」

  再試行してもいいですか: はい。
- ConcurrentRunsExceededException (400)
  - 「ML タスク実行数 [count] が [limit] タスク実行の変換制限を超えています。」
  - 「ML タスク実行数 [count] が [limit] タスク実行数の制限を超えています。」

  再試行してもいいですか: はい。タスク実行が完了するまで待ってから再試行できます。

StartMLLabelingSetGenerationTaskRunActivity

このアクティビティには、次の例外があります。

- EntityNotFoundException (400)
  - 「ハンドル [transformName] を持つアカウント [accountId] に ML 変換が見つかりません。」

  再試行してもいいですか: いいえ。
- InvalidInputException (400)
  - 「入力レコードテーブルは 1 つだけ指定する必要があります。」
  - 「データベース名を指定する必要があります。」

- 「テーブル名を指定する必要があります。」
- 「一致検索パラメータが設定されていません。」
- 「一致検索パラメータに主キーを指定する必要があります。」

再試行してもいいですか: いいえ。

- InternalServiceException (500)
  - 「依存関係エラー。」

再試行してもいいですか: はい。

- ConcurrentRunsExceededException (400)
  - 「ML タスク実行数 [count] が [limit] タスク実行の変換制限を超えています。」

再試行してもいいですか: はい。タスク実行の完了後に再試行できます。

UpdateMLTransformActivity

このアクティビティには、次の例外があります。

- EntityNotFoundException (400)
  - 「ハンドル [transformName] を持つアカウント [accountId] に ML 変換が見つかりません。」

再試行してもいいですか: いいえ。

- InvalidInputException (400)
  - 「[TransformName] という名前の別の変換がすでに存在します。」
  - 「[message]。」
  - 「変換名を空白にすることはできません。」
  - 「最大容量とワーカー番号/タイプを同時に設定することはできません。」
  - 「WorkerType と NumberOfWorkers の両方を設定する必要があります。」
  - 「MaxCapacity は >= [minMaxCapacity] にする必要があります。」
  - 「NumberOfWorkers は >= [minNumWorkers] にする必要があります。」
  - 「最大再試行数は負以外にする必要があります。」
  - 「予期しない入力による内部サービスエラー。」
  - 「一致検索パラメータが設定されていません。」
  - 「一致検索パラメータに主キーを指定する必要があります。」

再試行してもいいですか: いいえ。

- AlreadyExistsException (400)
  - 「[transformName] という名前の変換はすでに存在します。」

  再試行してもいいですか: いいえ。
- IdempotentParameterMismatchException (400)
  - 「変換 [transformName] のべき等作成リクエストにパラメータの不一致がありました。」

  再試行してもいいですか: いいえ。

AWS Glue のクォータ

AWS サポート に連絡して、「AWS 全般のリファレンス」に記載されているサービスクォータに関し、クォータの引き上げをリクエストできます。特に明記されていない限り、クォータは地域固有です。詳細については、「AWS Glue エンドポイントとクォータ」を参照してください。

AWS Glue パフォーマンスの向上

パフォーマンスチューニングのベースライン戦略

AWS Glue のパフォーマンスを向上させるには、AWS Glue パラメータに関連する特定のパフォーマンスを更新することを検討してください。パラメータのチューニングを準備する際は、次のベストプラクティスを使用します。

- 問題の特定を始める前に、パフォーマンス目標を決定します。
- パラメータのチューニングを変更する前に、メトリクスを使用して問題を特定します。

ジョブのチューニングで一貫した結果を得るには、チューニング作業のベースライン戦略を立てます。

通常、パフォーマンスチューニングは次のワークフローで実施します。

1. パフォーマンス目標を決定します。
2. メトリクスを測定します。
3. ボトルネックを特定します。
4. ボトルネックの影響を軽減します。
5. 想定目標を達成するまで、ステップ 2 ~ 4 を繰り返します。

ジョブタイプのチューニング戦略

Spark ジョブ — 「AWS の規範的ガイダンス」の「AWS Glue for Apache Spark ジョブのパフォーマンスチューニングのベストプラクティス」のガイダンスに従ってください。

その他のジョブ – 他のランタイム環境で利用できる戦略を適応させることにより、Ray や AWS Glue Python のシェルジョブ用に AWS Glue をチューニングできます。

AWS Glue for Apache Spark ジョブのためのパフォーマンスの向上

AWS Glue for Spark のパフォーマンスを向上させるには、AWS Glue 関連のパフォーマンスおよび Spark パラメータを更新することを検討してください。

メトリクスを通じてボトルネックを特定し、その影響を軽減するための具体的な戦略に関する詳細は、「AWS 規範的ガイダンス」の「Best practices for performance tuning AWS Glue for Apache

Spark jobs」を参照してください。このガイドでは、Spark アーキテクチャや Resilient Distributed Datasets など、すべてのランタイム環境で Apache Spark に適用できる主なトピックを紹介します。これらのトピックを参考に、シャッフルの最適化やタスクの並列化など、特定のパフォーマンスチューニング戦略を実装する方法を案内します。

AWS Glue を設定して Spark UI を表示することにより、ボトルネックを特定できます。詳細については、「the section called “Spark UI を使用したモニタリング”」を参照してください。

さらに、AWS Glue はジョブが接続する特定タイプのデータストアに適用できるパフォーマンス機能も提供します。データストアのパフォーマンスパラメータに関する参照情報は、the section called “接続パラメータ” にあります。

AWS Glue ETL のプッシュダウンによる読み取りの最適化

プッシュダウンは、データのソースにより近いデータの取得に関するロジックをプッシュする最適化手法です。ソースは、データベースでも、Amazon S3 などのファイルシステムでもかまいません。特定の操作をソース上で直接実行する場合、ネットワーク上のすべてのデータを AWS Glue が管理する Spark エンジンに送らないようにすることで、時間と処理能力を節約できます。

つまり、プッシュダウンによってデータスキャンが減ります。この手法が適切であることを特定するプロセスの詳細については、「AWS 規範的ガイダンス」の「Best practices for performance tuning AWS Glue for Apache Spark jobs」ガイドにある「Reduce the amount of data scan」を参照してください。

Amazon S3 に保存されているファイルの述語プッシュダウン

プレフィックス別に整理された Amazon S3 上のファイルを操作する場合、プッシュダウン述語を定義することでターゲット Amazon S3 パスをフィルタリングできます。データセット全体を読み込み、`DynamicFrame` 内のフィルタを適用する代わりに、AWS Glue データカタログに保存されているパーティションメタデータにフィルタを直接適用できます。この方法では、必要なデータだけを選択的に一覧表示して読み込むことができます。パーティションごとのバケットへの書き込みなど、このプロセスの詳細については、「the section called “パーティションの管理”」を参照してください。

Amazon S3 で述語プッシュダウンを実現するには、`push_down_predicate` パラメータを使用します。Amazon S3 のバケットを年、月、日ごとに分割したとします。2022 年 6 月の顧客データを取得したい場合は、関連する Amazon S3 パスのみを読み込むように AWS Glue に指示できます。この場合の `push_down_predicate` は `year='2022' and month='06'` です。まとめると、読み込み操作は以下のようになります。

Python

```
customer_records = glueContext.create_dynamic_frame.from_catalog(
    database = "customer_db",
    table_name = "customer_tbl",
    push_down_predicate = "year='2022' and month='06'"
)
```

Scala

```
val customer_records = glueContext.getCatalogSource(
database="customer_db",
tableName="customer_tbl",
pushDownPredicate="year='2022' and month='06'"
).getDynamicFrame()
```

前のシナリオでは、push_down_predicate は AWS Glue データカタログからすべてのパーティションのリストを取得してフィルタリングしてから、基になる Amazon S3 ファイルを読み込みます。これはほとんどの場合に役立ちますが、数百万のパーティションがあるデータセットを扱う場合、パーティションを一覧表示するプロセスには時間がかかる場合があります。この問題に対処するには、サーバー側でパーティションをプルーニングすることでパフォーマンスを向上させることができます。そのためには、AWS Glue データカタログでデータのパーティションインデックスを作成します。パーティションインデックスについての詳細は、「the section called “パーティションインデックスの作成”」を参照してください。その後、catalogPartitionPredicate オプションを使用してインデックスを参照できます。catalogPartitionPredicate を使用してパーティションを取得する例については、「the section called “カタログパーティション述語”」を参照してください。

JDBC ソースを操作するときのプッシュダウン

GlueContext で使用されている AWS Glue JDBC リーダーは、ソース上で直接実行できるカスタム SQL クエリを提供することで、サポートされているデータベースへのプッシュダウンをサポートします。これは sampleQuery パラメータを設定することで実現できます。サンプルクエリでは、選択する列を指定したり、Spark エンジンに転送されるデータを制限するプッシュダウン述語を指定したりできます。

デフォルトでは、サンプルクエリは 1 つのノードで動作するため、大量のデータを処理するとジョブが失敗する可能性があります。この機能を使用してデータを大規模にクエリするに

は、enablePartitioningForSampleQuery を true に設定してクエリパーティショニングを設定する必要があります。これにより、選択したキーにまたがる複数のノードにクエリが分散されます。クエリパーティショニングには、他にもいくつかの必要な設定パラメータがあります。クエリパーティショニングについての詳細は、「the section called “JDBC からの並列読み取り”」を参照してください。

enablePartitioningForSampleQuery を設定すると、AWS Glue はデータベースをクエリするときに、プッシュダウン述語とパーティショニング述語を組み合わせます。パーティショニング条件を追加するには、AWS Glue の sampleQuery が AND で終わる必要があります (プッシュダウン述語を指定しない場合は、sampleQuery は WHERE で終わる必要があります)。以下の例を参照してください。ここでは、述語をプッシュダウンして id が 1000 を超える行のみを取得しています。この sampleQuery により、id が指定した値より大きい行の名前と場所の列だけが返されます。

Python

```
sample_query = "select name, location from customer_tbl WHERE id>=1000 AND"
customer_records = glueContext.create_dynamic_frame.from_catalog(
    database="customer_db",
    table_name="customer_tbl",
    sample_query = "select name, location from customer_tbl WHERE id>=1000 AND",

    additional_options = {
                            "hashpartitions": 36 ,
                            "hashfield":"id",
                            "enablePartitioningForSampleQuery":True,
                            "sampleQuery":sample_query
                           }
)
```

Scala

```
val additionalOptions = Map(
        "hashpartitions" -> "36",
        "hashfield" -> "id",
        "enablePartitioningForSampleQuery" -> "true",
        "sampleQuery" -> "select name, location from customer_tbl WHERE id >= 1000
 AND"
        )

    val customer_records = glueContext.getCatalogSource(
        database="customer_db",
        tableName="customer_tbl").getDynamicFrame()
```

> **Note**
>
> customer_tbl が Data Catalog と基になるデータストアで異なる名前がある場合、クエリは基になるデータストアに渡されるため、sample_query に基になるテーブル名を指定する必要があります。

AWS Glue データカタログと統合せずに JDBC テーブルに対してクエリを実行することもできます。ユーザー名とパスワードをパラメータとしてメソッドに提供する代わりに、useConnectionProperties と connectionName を指定することで既存の接続の認証情報を再利用できます。この例では、my_postgre_connection という接続から認証情報を取得します。

Python

```
connection_options_dict = {
    "useConnectionProperties": True,
    "connectionName": "my_postgre_connection",
    "dbtable":"customer_tbl",
    "sampleQuery":"select name, location from customer_tbl WHERE id>=1000 AND",
    "enablePartitioningForSampleQuery":True,
    "hashfield":"id",
    "hashpartitions":36
    }

customer_records = glueContext.create_dynamic_frame.from_options(
    connection_type="postgresql",
    connection_options=connection_options_dict
    )
```

Scala

```
val connectionOptionsJson = """
      {
        "useConnectionProperties": true,
        "connectionName": "my_postgre_connection",
        "dbtable": "customer_tbl",
        "sampleQuery": "select name, location from customer_tbl WHERE id>=1000 AND",
```

```
        "enablePartitioningForSampleQuery" : true,
        "hashfield" : "id",
        "hashpartitions" : 36
      }
    """

    val connectionOptions = new JsonOptions(connectionOptionsJson)

    val dyf = glueContext.getSource("postgresql",
connectionOptions).getDynamicFrame()
```

AWS Glue のプッシュダウンに関する注意事項と制限事項

プッシュダウンは概念上、ストリーミング以外のソースから読み込む場合にも適用できます。AWSGlue はさまざまなソースに対応しています。プッシュダウン機能はソースとコネクタによって異なります。

- Snowflake に接続するときは、query オプションを使用できます。AWS Glue 4.0 以降のバージョンの Redshift コネクタにも同様の機能があります。Snowflake から query を使用して読み込む方法については、「the section called “Snowflake から読み込む”」を参照してください。
- DynamoDB ETL リーダーでは、フィルタまたはプッシュダウン述語は使用できません。MongoDB と DocumentDB もこの種の機能をサポートしていません。
- Amazon S3 に保存されているデータをオープンテーブル形式で読み込む場合、Amazon S3 内のファイルのパーティショニング方法ではもはや十分ではありません。オープンテーブル形式を使用してパーティションから読み込みと書き込みを行うには、その形式のドキュメントを参照してください。
- DynamicFrame メソッドは Amazon S3 プロジェクションプッシュダウンを実行しません。述語フィルタを通過したファイルからすべての列が読み込まれます。
- AWS Glue で custom.jdbc コネクタを操作する場合、プッシュダウンできるかどうかはソースとコネクタによって異なります。該当するコネクタのマニュアルを参照して、AWS Glue のプッシュダウンをサポートしているかどうか、またどのようにサポートしているかを確認してください。

での自動スケーリングの使用 AWS Glue

Auto Scaling は、 で使用できます。AWS Glue ETLを使用した 、インタラクティブセッション、ストリーミングジョブ AWS Glue バージョン 3.0 以降。

Auto Scaling を有効にすると、次のようなメリットがあります。

- AWS Glue は、ジョブ実行の各ステージまたはマイクロバッチでの並列処理に応じて、クラスターからワーカーを自動的に追加および削除します。
- これにより、 に割り当てるワーカーの数を実験して決定する必要が減ります。AWS Glue ETL ジョブ。
- 指定されたワーカーの最大数では、AWS Glue は、ワークロードに適したサイズのリソースを選択します。
- ジョブの実行中にクラスターのサイズがどのように変化するかは、 のジョブ実行の詳細ページの CloudWatch メトリクスで確認できます。AWS Glue Studio。

の Auto Scaling AWS Glue ETL および ストリーミングジョブにより、 のコンピューティングリソースのオンデマンドスケーリングアウトとスケーリングインが可能になります。AWS Glue ジョブ。オンデマンドのスケールアップは、ジョブ実行のスタートアップに必要なコンピューティングリソースのみを最初に割り当てることができ、またジョブ実行中の需要に応じて必要なリソースをプロビジョニングするためにも役立ちます。

Auto Scaling は、 の動的スケールインもサポートします。AWS Glue ジョブ中のジョブリソース。ジョブ実行中に Spark アプリケーションからより多くのエグゼキューターが要求されると、より多くのワーカーがクラスターに追加さます。エグゼキュータがアクティブな計算タスクなしでアイドル状態になると、エグゼキュータと対応するワーカーが削除されます。

Auto Scaling が Spark アプリケーションのコストと使用率に役立つ一般的なシナリオは次のとおりです。

- Spark ドライバーがAmazon S3 内の多数のファイルを一覧表示している、またはエグゼキュターが非アクティブのときにロードを実行している場合
- Spark ステージがオーバープロビジョニングにより、少数のエグゼキュターのみで実行されている場合
- Spark ステージ間でのデータの偏りまたは不均一な計算需要

要件

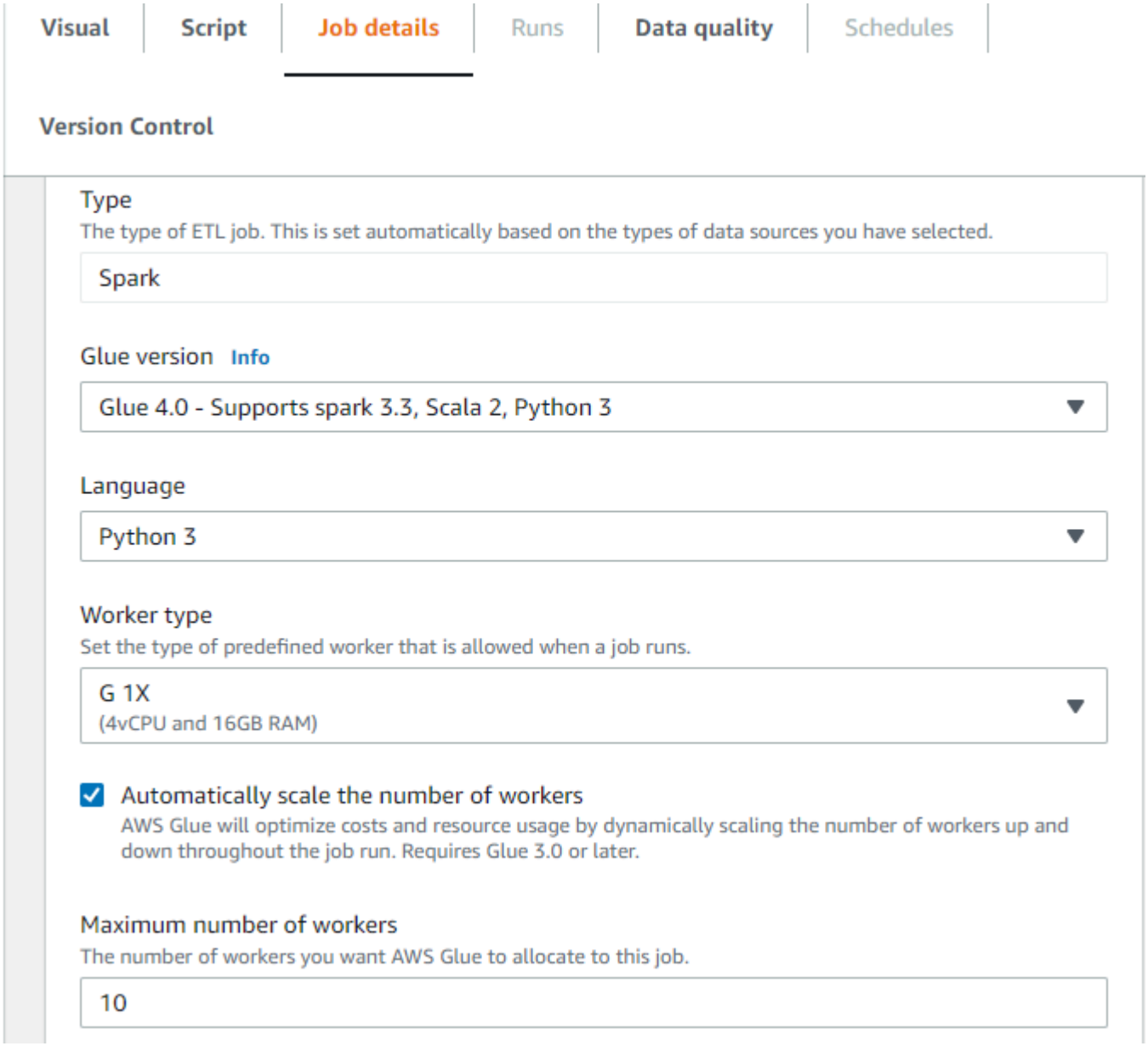
Auto Scaling は でのみ使用できます。AWS Glue バージョン 3.0 以降。Auto Scaling を使用するには、移行ガイドに従って既存のジョブを に移行します。AWS Glue バージョン 3.0 以降、または を使用して新しいジョブを作成する AWS Glue バージョン 3.0 以降。

Auto Scaling は で使用できます。AWS Glue 、G.1X、G.2XG.4X、G.8X、または G.025X (ストリーミングジョブのみ) ワーカータイプの ジョブ。標準DPUsはサポートされていません。

で Auto Scaling を有効にする AWS Glue Studio

のジョブの詳細タブで AWS Glue Studio で、タイプを Spark または Spark Streaming として選択し、Glue バージョンを **Glue 3.0**以降として選択します。次に、[ワーカータイプ] の下にチェックボックスが表示されます。

- [Automatically scale the number of workers] (ワーカー数を自動的にスケールする) を選択します。
- ワーカーの最大数を設定して、ジョブ実行に投入できるワーカーの最大数を定義します。



または AWS CLIで Auto Scaling を有効にする SDK

ジョブ実行のために AWS CLIから Auto Scaling を有効にするには、次の設定`start-job-run`で を実行します。

```
{
    "JobName": "<your job name>",
    "Arguments": {
        "--enable-auto-scaling": "true"
    },
    "WorkerType": "G.2X", // G.1X and G.2X are allowed for Auto Scaling Jobs
```

```
    "NumberOfWorkers": 20, // represents Maximum number of workers
    ...other job run configurations...
}
```

ETL ジョブ実行が完了したら、 を呼び出しget-job-runて、ジョブ実行の実際のリソース使用状況を DPU-秒で確認することもできます。注: 新しいフィールドDPUSecondsは、Auto Scaling で有効な AWS Glue 4.0 以降のバッチジョブに対してのみ表示されます。このフィールドは、ストリーミングジョブではサポートされません。

```
$ aws glue get-job-run --job-name your-job-name --run-id jr_xx --endpoint https://
glue.us-east-1.amazonaws.com --region us-east-1
{
    "JobRun": {
        ...
        "GlueVersion": "3.0",
        "DPUSeconds": 386.0
    }
}
```

Auto Scaling でジョブ実行を同じ設定AWS Glue SDKの を使用して設定することもできます。

インタラクティブセッションによる Auto Scaling の有効化

インタラクティブセッションを使用して AWS Glue ジョブを構築するときに Auto Scaling を有効にするには、「インタラクティブセッションの設定 AWS Glue」を参照してください。

ヒントと注意点

AWS Glue Auto Scaling を微調整するためのヒントと考慮事項：

- ワーカーの最大数の初期値について不明な場合は、推定 AWS Glue DPU で説明されている大まかな計算から始めることができます。データ量が非常に少ない場合、最大ワーカー数に極端に大きな値を設定しないでください。
- AWS Glue Auto Scaling は、ジョブで設定された最大数 DPU (ワーカーの最大数とワーカータイプで計算) spark.default.parallelismに基づいて spark.sql.shuffle.partitions と を設定します。これらの設定で固定値を使用する場合は、次のジョブパラメータでこれらのパラメータを上書きできます。
  - [Key] (キー): --conf
  - 値: spark.sql.shuffle.partitions=200 --conf spark.default.parallelism=200

- ストリーミングジョブの場合、デフォルトでは、マイクロバッチ内で自動スケーリング AWS Glue は行われず、自動スケーリングを開始するには複数のマイクロバッチが必要です。マイクロバッチ内で自動スケーリングを有効にする場合は、`--auto-scale-within-microbatch` を指定します。詳細については、「擬似パラメータのリファレンス」を参照してください。

Amazon CloudWatch メトリクスによる Auto Scaling のモニタリング

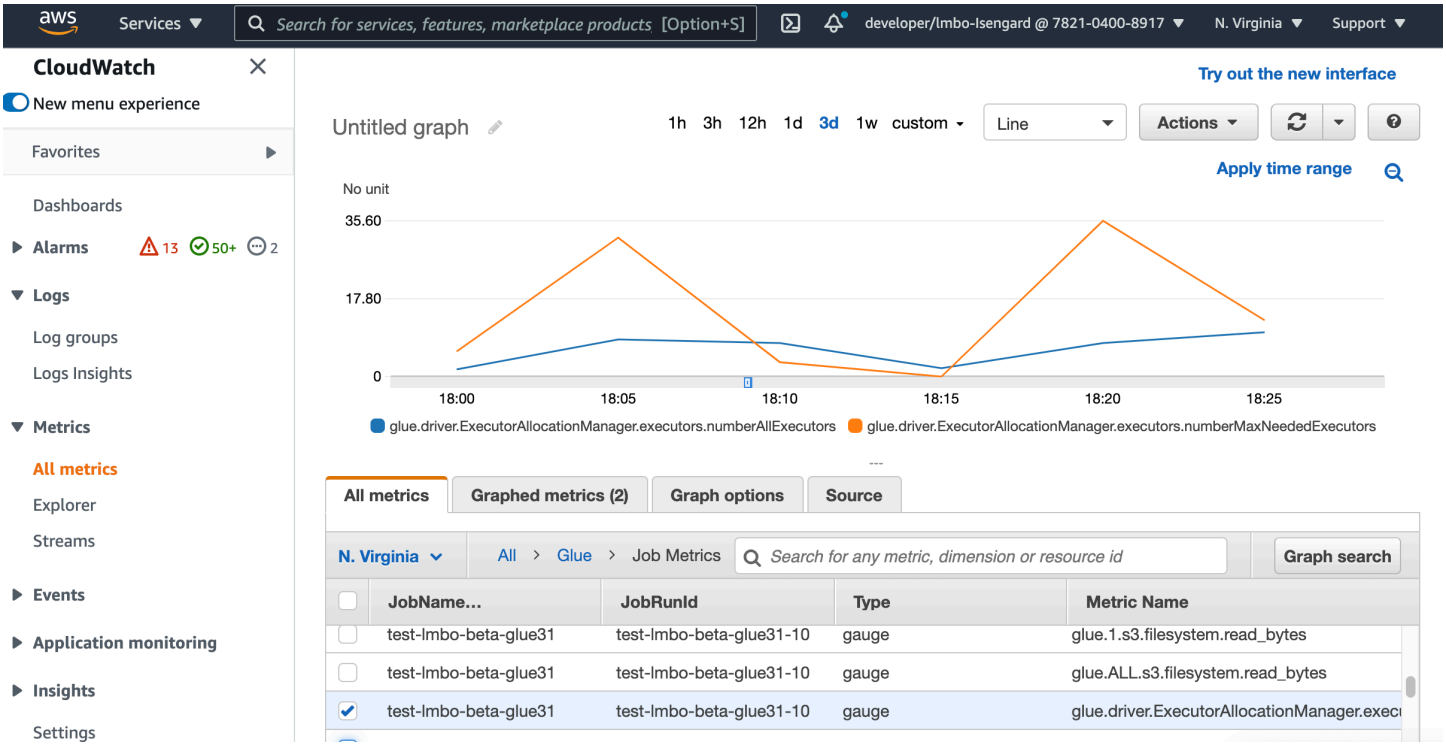
エグ CloudWatch ゼキュターメトリクスは、AWS Glue Auto Scaling を有効にする場合は、3.0 以降のジョブ。メトリクスを使用して、Auto Scaling で有効化された Spark アプリケーションでのエグゼキューターの需要と最適化された使用状況をモニタリングできます。詳細については、「Amazon CloudWatch メトリクスを使用した AWS Glue のモニタリング」を参照してください。

また、 AWS Glue オブザーバビリティメトリクスを使用して、リソース使用率に関するインサイトを取得することもできます。例えば、`glue.driver.workerUtilization` をモニタリングすることで、自動スケーリングの有無にかかわらず、実際に使用されたリソースの量をモニタリングできます。別の例として、`glue.driver.skewness.job` と `glue.driver.skewness.stage` をモニタリングすることで、データの歪みを確認できます。これらのインサイトは、自動スケーリングを有効にするかどうかを決定し、設定を微調整するのに役立ちます。詳細については、「AWS Glue オブザーバビリティメトリクスを使用したモニタリング でのモニタリング」を参照してください。

- glue.driverExecutorAllocationManager..executors。numberAllExecutors
- glue.driverExecutorAllocationManager..executors.numberMaxNeededExecutors

これらのメトリクスの詳細については、「DPU の容量計画のモニタリング」を参照してください。

> **Note**
>
> CloudWatch エグゼキューターメトリクスは、インタラクティブセッションでは使用できません。

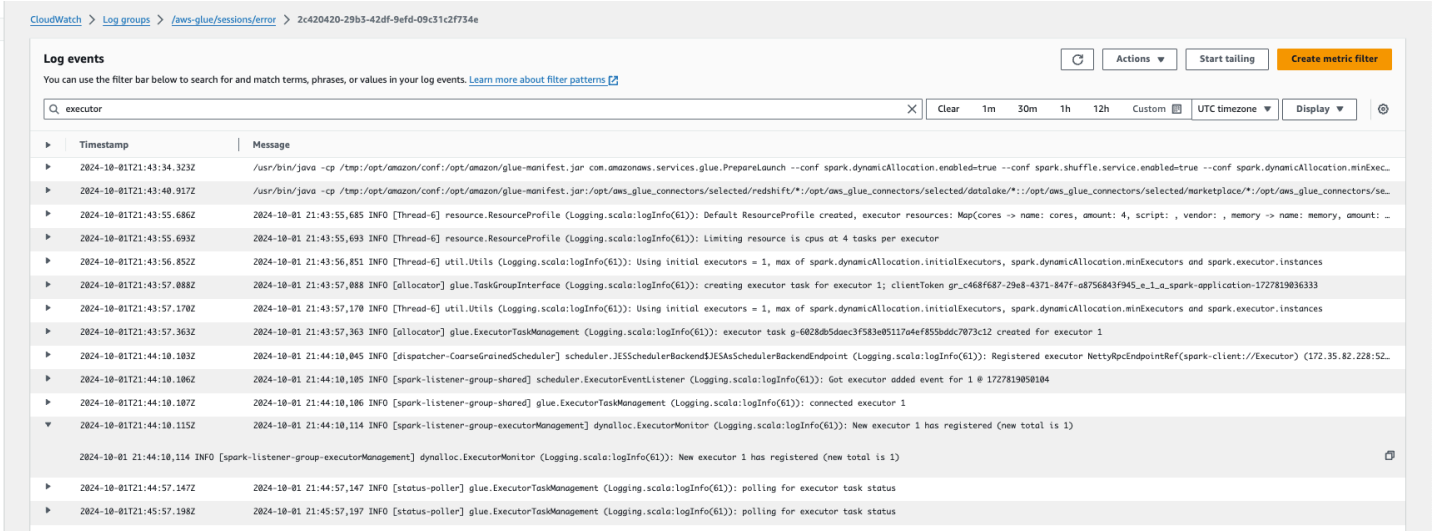


Amazon CloudWatch Logs による Auto Scaling のモニタリング

インタラクティブセッションを使用している場合は、継続的な Amazon CloudWatch Logs を有効にして ログで「エグゼキューター」を検索するか、Spark UI を使用してエグゼキューターの数をモニタリングできます。これを行うには、`%%configure` マジックを使用して `enable auto scaling` とともに継続的なログ記録を有効にします。

```
%%configure{
    "--enable-continuous-cloudwatch-log": "true",
    "--enable-auto-scaling": "true"
}
```

Amazon CloudWatch Logsevents で、ログで「実行」を検索します。

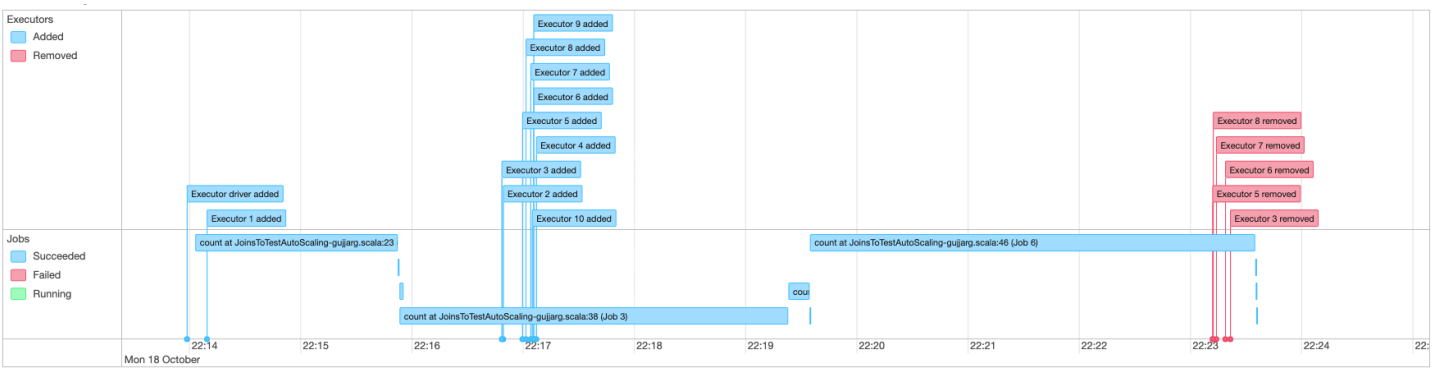


Spark UI による Auto Scaling のモニタリング

Auto Scaling を有効にすると、AWS Glue Glue Spark UI を使用する ジョブ。詳細については、「AWS Glue ジョブ用の Apache Spark ウェブ UI の有効化」を参照してください。

Jupyter Notebook のインタラクティブセッションを使用すると、次のマジックを実行して、Spark UI とともに自動スケーリングを有効にすることができます。

```
%%configure{
    "--enable-auto-scaling": "true",
    "--enable-continuous-cloudwatch-log": "true"
}
```



Auto Scaling ジョブの実行DPU使用状況のモニタリング

AWS Glue Studio ジョブ実行ビューを使用して、Auto Scaling ジョブのDPU使用状況を確認できます。

1. AWS Glue Studio ナビゲーションペインからモニタリングを選択します。Monitoring (モニタリング) ページが表示されます。
2. ジョブ実行チャートまでスクロールダウンします。
3. 目的のジョブ実行に移動し、DPU時間列にスクロールして、特定のジョブ実行の使用状況を確認します。

制限

AWS Glue 現在、ストリーミング Auto Scaling は、 の外部で DataFrame 作成された静的なストリーミング DataFrame 結合をサポートしていませんForEachBatch。内で DataFrame 作成された静的は、期待どおりに機能ForEachBatchします。

実行に上限を設定してワークロードをパーティション化する

通常、Spark アプリケーションのエラーは、Spark スクリプトが非効率的なこと、大規模な変換を分散メモリ内で実行すること、およびデータセットの異常によって発生します。ドライバーや エグゼキュターがメモリ不足の問題を引き起こす原因はたくさんあります。例えば、データの偏り、オブジェクトのリストが多すぎること、または大きなデータのシャッフルなどです。これらの問題は、Sparkで大量のバックログデータを処理しているときにしばしば発生します。

AWS Glue を使用すると、OOM の問題を解決し、ワークロードのパーティション化によって ETL 処理を容易に行うことができます。ワークロードのパーティション化を有効にすると、各 ETL ジョブ実行では、データセットサイズまたはこのジョブ実行で処理されるファイル数に上限を設定して、未処理のデータのみを選択します。残りのデータは、後で実行されるジョブで処理されます。例えば、1000 個のファイルを処理する必要がある場合、ファイル数を 500 に設定し、2 つのジョブ実行に分けることができます。

ワークロードのパーティション化は、Amazon S3 のデータソースでのみサポートされています。

ワークロードのパーティション化の有効化

スクリプトでオプションを手動で設定するか、カタログテーブルのプロパティを追加することで、実行に上限を設定できます。

スクリプトで実行に上限を設定してワークロードのパーティション化を有効にするには、次の手順に従います。

1. データの再処理を回避するには、新しいジョブまたは既存のジョブでジョブブックマークを有効にします。詳細については、「Tracking Processed Data Using Job Bookmarks」を参照してください。
2. スクリプトを修正し、AWS Glue getSource API の追加のオプションに上限の制限を設定します。また、ジョブブックマークの変換コンテキストを設定して、state 要素を保存します。例:

   Python

   ```
   glueContext.create_dynamic_frame.from_catalog(
       database = "database",
       table_name = "table_name",
       redshift_tmp_dir = "",
       transformation_ctx = "datasource0",
       additional_options = {
           "boundedFiles" : "500", # need to be string
         # "boundedSize" : "1000000000" unit is byte
       }
   )
   ```

   Scala

   ```
   val datasource0 = glueContext.getCatalogSource(
       database = "database", tableName = "table_name", redshiftTmpDir = "",
       transformationContext = "datasource0",
       additionalOptions = JsonOptions(
           Map("boundedFiles" -> "500") // need to be string
         //"boundedSize" -> "1000000000" unit is byte
       )
   ).getDynamicFrame()
   ```

   ```
   val connectionOptions = JsonOptions(
       Map("paths" -> List(baseLocation), "boundedFiles" -> "30")
   )
   val source = glueContext.getSource("s3", connectionOptions, "datasource0", "")
   ```

Data Catalog テーブルで実行に上限を設定してワークロードのパーティション化を有効にするには

1. Data Catalog のテーブル構造の `parameters` フィールドに、キーと値のペアを設定します。詳細については、「Viewing and Editing Table Details」を参照してください。
2. データセットサイズの上限または処理されるファイル数を次のように設定します。

   - `boundedSize` をデータセットのターゲットサイズ (バイト単位) に設定します。ジョブの実行は、テーブルからターゲットサイズに達すると停止します。
   - `boundedFiles` をターゲットファイル数に設定します。ジョブの実行は、ターゲット数のファイルを処理すると停止します。

> **Note**
>
> 単一の上限のみがサポートされているので、`boundedSize` または `boundedFiles` のどちらか一方だけを設定します。

ジョブを自動的に実行する AWS Glue トリガーの設定

実行に上限を設定すると、AWS Glue トリガーを設定してジョブを自動的に実行し、シーケンシャル実行でデータを段階的にロードすることができます。AWS Glue コンソールに移動してトリガーを作成し、スケジュール時刻を設定し、ジョブにアタッチします。その後、自動的に次のジョブ実行がトリガーされ、データの新しいバッチが処理されます。

また、AWS Glue ワークフローを使用して、複数のジョブを調整して、異なるパーティションのデータを並行して処理することもできます。詳細については、「AWS Glue Triggers」および「AWS Glue Workflows」を参照してください。

ユースケースとオプションの詳細については、ブログ「Optimizing Spark applications with workload partitioning in AWS Glue」を参照してください。

AWS Glue の既知の問題

AWS Glue には以下の既知の問題があります。

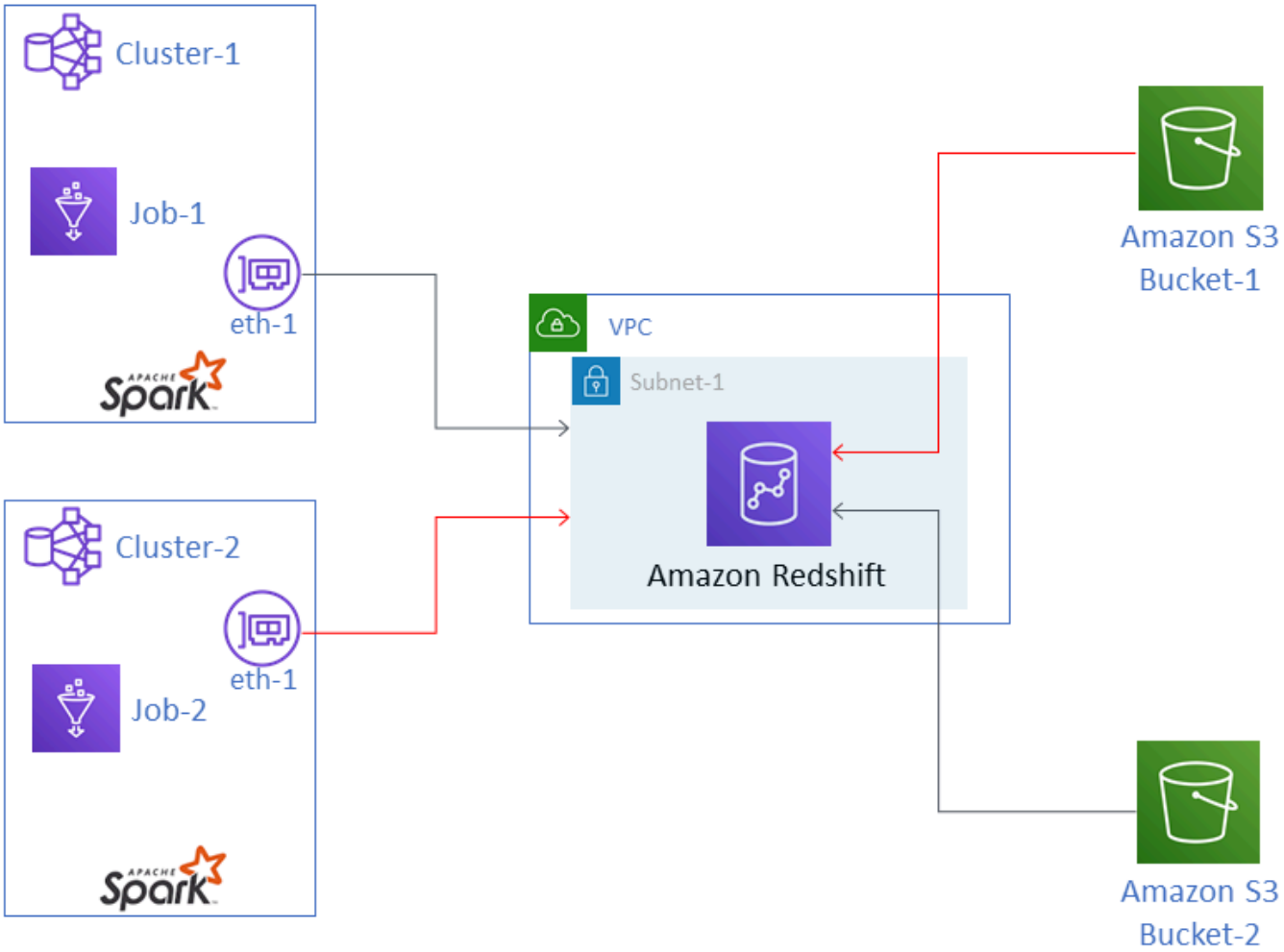
トピック

- クロスジョブデータアクセスの防止

クロスジョブデータアクセスの防止

単一の AWS アカウントに 2 つの AWS Glue Spark ジョブがあり、それぞれが別の AWS Glue Spark クラスターで実行されている状況を考慮します。ジョブは、AWS Glue 接続を使用して、同じ Virtual Private Cloud (VPC) 内のリソースにアクセスしています。この状況では、一方のクラスターで実行されているジョブが、もう一方のクラスターで実行されているジョブのデータにアクセスできる可能性があります。

次の図は、この状況の例を示しています。



この図では、AWS Glue Job-1 は Cluster-1 で実行され 、Job-2 は Cluster-2 で実行されています。どちらのジョブも、VPC の Subnet-1 に存在する Amazon Redshift の同じインスタンスを使用しています。Subnet-1 は、パブリックサブネットであることもプライベートサブネットであることもあります。

Job-1 は、Amazon Simple Storage Service (Amazon S3) Bucket-1 からデータを変換し、データを Amazon Redshift に書き込んでいます。Job-2 は Bucket-2 のデータで同じ処理を行っています。Job-1 は、Bucket-1 へのアクセスを許可する AWS Identity and Access Management (IAM) ロール Role-1 (非表示) を使用しています。Job-2 は、Bucket-2 へのアクセスを許可する Role-2 (非表示) を使用しています。

この 2 つのジョブにはネットワークパスがあり、相互のクラスターと通信し、相互のデータにアクセスできるようになっています。たとえば、Job-2 は Bucket-1 のデータにアクセスできます。この図では、これは赤色のパスとして示されています。

このような状況を回避するため、Job-1 および Job-2 に異なるセキュリティ設定をアタッチすることをお勧めします。セキュリティ設定をアタッチすることで、データへのクロスジョブアクセスは AWS Glue が作成する証明書によってブロックされます。セキュリティ設定は、ダミー設定にすることができます。つまり、Amazon S3 データ、Amazon CloudWatch データ、ジョブのブックマークの暗号化を有効にすることなく、セキュリティ設定を作成できます。3 つの暗号化オプションはすべて無効にできます。

セキュリティ設定の詳細については、「the section called “AWS Glue によって書き込まれたデータの暗号化”」を参照してください。

セキュリティ設定をジョブにアタッチするには

1. https://console.aws.amazon.com/glue/ で AWS Glue コンソール を開きます。
2. ジョブの [Configure the job properties (ジョブプロパティの設定)] ページで、[セキュリティ設定、スクリプトライブラリおよびジョブパラメータ] セクションを展開します。
3. リストでセキュリティ設定を選択します。

AWS Glue のドキュメント履歴

変更	説明	日付
AWS Glue 用に、さらに 14 つの新しい SaaS ネイティブコネクタをサポート	さらに 14 つの SaaS ネイティブコネクタが AWS Glue に追加されました。詳細については、「AWS Glue 接続の追加」を参照してください。	2025 年 1 月 30 日
AWS Glue 用のさらに 16 つの新しい SaaS ネイティブコネクタのサポート	さらに 16 個の SaaS ネイティブコネクタが AWS Glue に追加されました。詳細については、「AWS Glue 接続の追加」を参照してください。	2024 年 12 月 17 日
列統計の自動生成	AWS Glue Data Quality は、Data Catalog と ETL で Amazon SageMaker AI LakeHouse テーブルと AWS Lake Formation マネージド Iceberg、Delta、HUDI テーブルをサポートするようになりました。詳細については、「AWS Glue Data Quality」を参照してください。	2024 年 12 月 6 日
ゼロ ETL 統合のサポート	ゼロ ETL は、ETL データパイプラインを構築する必要性を最小限に抑える、AWS によるフルマネージド型の統合のセットです。詳細については、「ゼロ ETL 統合」を参照してください。	2024 年 12 月 3 日
再利用可能な接続のサポート	新しい AWS Glue 接続スキーマでは、AWS Glue、Amazon	2024 年 12 月 3 日

	Athena、Amazon SageMaker Unified Studio などの AWS サービスとアプリケーションとの間のデータ接続を統一された方法で管理できます。詳細については、「データへの接続」を参照してください。	
AWS Glue バージョン 5.0 のサポート。	AWS Glue バージョン 5.0 のサポートに関する情報を追加しました。機能には、Apache Spark の 3.52 への更新、Java の 17 への更新、オープンテーブルフォーマットの更新、Spark ネイティブのきめ細かなアクセスコントロール、Sagemaker Lakehouse とデータウェアハウスの抽象化統合、Sagemaker Unified Studio のサポートなどがあります。詳細については、「AWS Glue リリースノート」および「AWS Glue ジョブの AWS Glue バージョン 5.0 への移行」を参照してください。	2024 年 12 月 3 日

AWS Glue Iceberg REST エンドポイントを使用した AWS Glue Data Catalog への接続方法	AWS Glue の Iceberg REST エンドポイントは、Apache Iceberg REST 仕様で指定された API オペレーションに対応しています。Iceberg REST クライアントを使用すると、分析エンジンで実行されているアプリケーションをデータカタログでホストされている REST カタログに接続することができます。詳細については、「データカタログにアクセスする」を参照してください	2024 年 12 月 3 日
列統計の自動生成	AWS Glue Data Catalog の新しいテーブルで列統計を自動生成 詳細については、「Automatic column statistics generation」を参照してください。	2024 年 12 月 3 日
AWS Glue での Apache Spark の生成 AI アップグレードのサポート	AWS Glue の Spark アップグレードにより、データエンジニアとデベロッパーは、生成 AI を使用して、既存の AWS Glue Spark ジョブをアップグレードおよび移行し、最新の Spark リリースに移行できます。詳細については、「AI を使用したアップグレード分析」を参照してください。	2024 年 11 月 22 日

AWS Glue での Apache Spark の生成 AI トラブルシューティングのサポート	AWS Glue の Apache Spark ジョブ用の生成 AI トラブルシューティングは、データエンジニアや科学者が Spark アプリケーションの問題を簡単に診断して修正するのに役立ちます。詳細については、「AI を使用した Spark ジョブのトラブルシューティング」を参照してください。	2024 年 11 月 22 日
Iceberg オプティマイザーが VPC 内の Amazon S3 バケットにアクセスするためのサポート	AWS Glue Data Catalog は、Iceberg テーブルオプティマイザーをサポートし、AWS Glue ネットワーク接続を使用して特定の仮想プライベートクラウド (VPC) から Amazon S3 バケットにアクセスします。詳細については、「Iceberg テーブルの最適化」を参照してください。	2024 年 11 月 20 日
AWS Glue 用のさらに 9 つの新しい SaaS ネイティブコネクタのサポート	さらに 9 つの SaaS ネイティブコネクタが AWS Glue に追加されました。詳細については、「AWS Glue 接続の追加」を参照してください。	2024 年 11 月 19 日
AWS Glue 用の 10 個の新しい SaaS ネイティブコネクタのサポート	10 個の SaaS ネイティブコネクタが AWS Glue に追加されました。詳細については、「AWS Glue 接続の追加」を参照してください。	2024 年 11 月 15 日

AWS Glue ETL ジョブのジョブ実行キューイングのサポート	サービスクォータが原因でジョブをすぐに実行できないとき、ジョブ実行キューイングを有効にして後でジョブを実行できます。詳細については、「AWS Glue で Spark ジョブに関するジョブプロパティの構成」を参照してください。	2024 年 9 月 3 日
ポリシー変更の更新	[AwsGlueSessionUserRestrictedNotebookPolicy] および [AwsGlueSessionUserRestrictedNotebookServiceRole] ポリシーに対する文書化された変更は、所有者タグキーを持つセッションにタグオン作成をサポートするために必要です。詳細については、「AWS Glue での AWS マネージドポリシーの更新」を参照してください。	2024 年 8 月 30 日
異常検出と動的ルールの一般提供が開始	AWS Glue Data Quality は機械学習アルゴリズムを利用して過去の傾向から学習し、将来の値を予測して異常を検出します。動的ルールは、動的なしきい値を設定できます。詳細については、「Iceberg テーブルのクエリパフォーマンスの最適化」を参照してください。	2024 年 8 月 7 日

ポリシー変更の更新	[AwsGlueSessionUserRestrictedPolicy] および [AwsGlueSessionUserRestrictedServiceRole] ポリシーに対する文書化された変更は、所有者タグキーを持つセッションにタグオン作成をサポートするために必要です。詳細については、「AWS Glue での AWS マネージドポリシーの更新」を参照してください。	2024 年 8 月 5 日
Iceberg テーブルの列統計の生成が一般公開されました	AWS Glue は、Iceberg テーブルの各列における個別値数 (NDV) の計算および更新をサポートします。詳細については、「AWS Glue Data Quality での異常検出」および「動的ルール」を参照してください。	2024 年 7 月 9 日
AWS Glue 使用プロファイルのサポート	管理者は、デベロッパー、テスター、製品チームなど、アカウント内のさまざまなクラスのユーザーの AWS Glue 使用プロファイルを作成できます。この柔軟性により、管理者はユーザーのクラスごとに異なる使用状況とコストの制御を適用できます。詳細については、「AWS Glue 使用プロファイルの設定」を参照してください。	2024 年 6 月 18 日

AWS Glue for Spark 向け Salesforce コネクタのサポート	Salesforce 向けの新しい AWS Glue コネクタに関する情報を追加しました。この機能を使用すると、AWS Glue for Spark を使用して 4.0 以降のバージョンで Salesforce と AWS Glue の間で読み書きを行うことができます。詳細については、「Salesforce への接続」を参照してください。	2024 年 5 月 22 日

AWS Glue の Amazon Q データ統合 (GA)	AWS Glue の Amazon Q データ統合は、データエンジニアや ETL 開発者が自然言語を使用してデータ統合ジョブを構築できるようにする、AWS Glue の新しい生成 AI 機能です。エンジニアや開発者は Q に、ジョブの作成、問題のトラブルシューティング、AWS Glue やデータ統合に関する質問への回答を依頼できます。詳細については、「AWS Glue の Amazon Q データ統合」を参照してください。この機能には、`AwsGlueSessionUserRestrictedPolicy` 、`AwsGlueSessionUserRestrictedNotebookServiceRole` 、`AwsGlueSessionUserRestrictedServiceRole` および AWS のマネージドポリシーの更新が含まれます。詳細については、「AWS Glue での AWS マネージドポリシーの更新」を参照してください。	2024 年 4 月 30 日

AWS Glue で Amazon Q のデータ統合 (プレビュー)	AWS Glue の Amazon Q データ統合は、データエンジニアや ETL 開発者が自然言語を使用してデータ統合ジョブを構築できるようにする、AWS Glue の新しい生成 AI 機能です。エンジニアや開発者は Q に、ジョブの作成、問題のトラブルシューティング、AWS Glue やデータ統合に関する質問への回答を依頼できます。詳細については、「AWS Glue の Amazon Q データ統合」を参照してください。この機能には、`AwsGlueSessionUserRestrictedNotebookPolicy` AWS マネージドポリシーの更新が含まれます。詳細については、「AWS Glue での AWS マネージドポリシーの更新」を参照してください。	2024 年 1 月 30 日
AWS Glue Streaming のドキュメントの更新	新しい章が追加され、AWS Glue ストリーミングのコンテンツが新しくなって再編成されました。このコンテンツでは、AWS Glue でのストリーミングの仕組み、リアルタイムデータ処理の特徴、ストリーミングジョブのモニタリング方法について説明します。詳細については、「AWS Glue Streaming」を参照してください。	2023 年 12 月 27 日

詳細な機密データ検出の使用をサポート	Detect Sensitive Data は、定義したエンティティ、または AWS Glue によって事前定義されたエンティティを検出、マスキング、削除する機能を提供します。さらに、詳細なアクションを使用することで、エンティティごとに特定のアクションを適用できます。詳細については、「詳細な機密データ検出の使用」を参照してください。	2023 年 11 月 26 日
AWS Glue オブザーバビリティメトリクスによるジョブのモニタリングをサポート	AWS Glue オブザーバビリティメトリクスを使用して、AWS Glue for Apache Spark ジョブの内部で何が起こっているかに関するインサイトを生成し、問題の優先順位付けと分析を改善できます。詳細については、「AWS Glue オブザーバビリティメトリクスを使用したモニタリング」を参照してください。	2023 年 11 月 26 日
AWS Glue Data Quality での異常検出をサポート	AWS Glue Data Quality の異常検出は、時間の経過とともにデータ統計に機械学習 (ML) アルゴリズムを適用して、ルールでは検出が難しい異常パターンや隠れたデータ品質問題を検出します。詳細については、「AWS Glue Data Quality での異常検出」を参照してください。	2023 年 11 月 26 日

デフォルトの Spark UI ログ動作を更新	Spark UI ログを生成する Spark ジョブは、AWS Glue コンソールで Spark UI をサポートするために別のファイル名パターンで書き込まれるようになりました。これによって CloudWatch ログの動作に変更が生じることはありません。ジョブ設定を更新することで、従来の動作に戻すことができます。詳細については、「Apache Spark ウェブ UI を使用したジョブのモニタリング」を参照してください。	2023 年 11 月 17 日

AWS Glue for Spark での新しいデータソースをサポート	Amazon OpenSearch Service、Azure SQL、Azure Cosmos for NoSQL、SAP HANA Teradata Vantage、Vertica への接続が AWS Glue 内でネイティブにサポートされるようになりました。さらに、これらのデータソースへの接続は、MongoDB に加えて、AWS Glue Studio のビジュアルエディターでも使用できるようになりました。詳しくは、AWS Glue for Spark のサポートについては「AWS Glue for Spark での ETL の接続タイプとオプション」を、AWS Glue Studio のビジュアルエディターでの使用については「AWS Glue 接続の追加」を参照してください。	2023 年 11 月 17 日
カラム統計生成のサポート	追加のデータパイプラインを設定することなく、Parquet、ORC、JSON、ION、CSV、XML などのデータ形式で AWS Glue Data Catalog テーブルの列レベルの統計を計算できます。詳細については、「カラム統計の操作」を参照してください。	2023 年 11 月 16 日

Iceberg テーブルのデータ圧縮のサポート	Amazon Athena、Amazon EMR、AWS Glue ETL ジョブなどの AWS 分析サービスによる読み取りパフォーマンスを向上させるために、Data カタログは、Data カタログ内の Iceberg テーブル用にマネージド圧縮 (小さな Amazon S3 オブジェクトを圧縮してより大きなオブジェクトにコンパクト化するプロセス) を提供しています。詳細については、「Iceberg テーブルの最適化」を参照してください。	2023 年 11 月 13 日
ジョブ実行待機動作を更新する	標準の Spark および Python シェルのジョブ実行は、すぐに `FAILED` に移行するのではなく、特定の状況において `WAITING` に移行するようになりました。詳細については、「AWS Glue ジョブ実行ステータス」を参照してください。	2023 年 11 月 8 日
AWS Glue Studio ユーザーガイドを AWS Glue デベロッパーガイドに統合	AWS Glue Studio ユーザーガイドがデベロッパーガイドに移され、AWS Glue Studio、AWS Glue コンソール、および AWS Glue Studio プログラムによるアクセスに関する単一の統合ユーザーガイドが作成されました。	2023 年 10 月 25 日

AWSGlueServiceNotebookRole AWS マネージドポリシーのアップデート	AWSGlueServiceNotebookRole AWS マネージドポリシーのマイナーアップデートに関する情報を追加しました。詳細については、「AWS Glue での AWS マネージドポリシーの更新」を参照してください。	2023 年 10 月 9 日
AWS Glue Studio が 5 つの新しい組み込み変換をサポート	AWS Glue Studio は、レコードマッチング、null 行の削除、JSON 列の解析、JSON パスの抽出、正規表現抽出の 5 つの新しい組み込み変換をサポートしています。詳細については、「AWS Glue マネージドデータ変換ノードの編集」を参照してください。	2023 年 8 月 11 日
AWSGlueServiceRole AWS マネージドポリシーのアップデート	AWSGlueServiceRole AWS マネージドポリシーのマイナーアップデートに関する情報を追加しました。詳細については、「AWS Glue での AWS マネージドポリシーの更新」を参照してください。	2023 年 8 月 4 日
Apache Hudi テーブルのクローリングをサポート	AWS Glue を使用した Amazon S3 バケット内の Hudi テーブルのクローリングと、AWS Glue Data Catalog への Hudi テーブルの登録に関する情報を追加しました。詳細については、「クロール可能なデータストア」および「クローラーのプロパティ」を参照してください。	2023 年 7 月 21 日

AWSGlueConsoleFullAccess AWS マネージドポリシーのアップデート	AWSGlueConsoleFullAccess AWS マネージドポリシーのマイナーアップデートに関する情報を追加しました。詳細については、「AWS Glue での AWS マネージドポリシーの更新」を参照してください。	2023 年 7 月 14 日
Apache Iceberg テーブルのクローリングをサポート	AWS Glue を使用した Amazon S3 バケット内の Iceberg テーブルのクローリングと、AWS Glue Data Catalog への Iceberg テーブルの登録に関する情報を追加しました。詳細については、「クロール可能なデータストア」および「クローラーのプロパティ」を参照してください。	2023 年 7 月 7 日
Ray を使用した AWS Glue をサポート	AWS Glue ジョブを強化可能な新しいエンジンである Ray を使用した AWS Glue に関する情報を追加しました。あいまいさを解消するために、Spark を使用した AWS Glue に関する既存の内容を再編成しました。	2023 年 5 月 30 日

AWS Glue Data Quality をサポート (一般提供)	AWS Glue Data Quality の一般提供が開始されました。AWS GlueData Quality は、データの品質を評価およびモニタリングするのに役立ちます。データカタログで AWS Glue Data Quality を使用する方法については、「AWS Glue Data Quality」を参照してください。AWS Glue Studio での AWS Glue Data Quality の詳細については、「AWS Glue Studio によるデータ品質の評価」を参照してください。	2023 年 5 月 24 日
Apache Spark ジョブの大規模ワーカータイプのサポート	Apache Spark ジョブ用に G.4X と G.8X のワーカータイプの使用がサポートされるようになりました。これらのワーカータイプは、ワークロードに含まれる変換、集約、結合、クエリへの要求が非常に厳しいジョブに適しています。詳細については、「AWS Glue でジョブを追加する」を参照してください。	2023 年 5 月 8 日
テーブルをクロールするときのパーティションインデックス作成のサポート	クローラーが検出したテーブルのパーティションインデックスの作成をクローラーがどのようにサポートするかについての情報を追加しました。詳細については、「クローラー設定オプションの設定」を参照してください。	2023 年 4 月 24 日

リソースの使用状況メトリクスのサポート	Amazon CloudWatch でのサービスのリソース使用状況の表示と、アラームの設定に関する情報を追加しました。詳細については、「AWS Glue のモニタリング」を参照してください。	2023 年 4 月 7 日
AWSGlueConsoleFullAccess AWS マネージドポリシーのアップデート	AWSGlueConsoleFullAccess AWS マネージドポリシーのマイナーアップデートに関する情報を追加しました。詳細については、「AWS Glue での AWS マネージドポリシーの更新」を参照してください。	2023 年 3 月 28 日
AWS SDK での AWS Glue の使用に関するガイダンスを例付きで追加	AWS Glue デベロッパーガイドに、AWS SDK での AWS Glue の使用に役立つ情報が記載された 2 つのセクションが新たに追加されました。詳細については、「AWS SDK での AWS Glue の使用」および「AWS SDK を使用した AWS Glue のコード例」を参照してください。	2023 年 2 月 23 日
AWS Glue での IAM ドキュメントの更新	AWS Glue での IAM の使用に関する情報を再編成して追加しました。詳細については、「AWS Glue のIdentity and Access Management」を参照してください。	2023 年 2 月 15 日

AWS Glue バージョン 4.0 でストリーミング ETL ジョブをサポート	Glue バージョン 4.0 でのストリーミング ETL ジョブの実行のサポートに関する情報と、Kafka クラスターまたは Amazon Managed Streaming for Apache Kafka クラスター、および Amazon Kinesis Data Streams に接続するための新しいオプションに関する情報が追加されました。詳細については、「AWS Glue でストリーミング ETL ジョブを追加する」および「AWS Glue での ETL の接続タイプとオプション」を参照してください。	2023 年 2 月 8 日
MongoDB Atlas データソースのクローリングをサポート	MongoDB Atlas データソースのクロールに AWS Glue を使用する方法についての情報を追加しました。詳細については、「クロール可能なデータストア」、「MongoDB と MongoDB Atlas の接続プロパティ」、「MongoDB または MongoDB Atlas 接続の使用」を参照してください。	2023 年 2 月 6 日

ネイティブの Delta Lake コネクタを使用した Delta Lake テーブルのクローリングをサポート	AWS Glue での、ネイティブの Delta Lake コネクタを使用した Delta Lake テーブルのクロールリングについての情報を追加しました。この機能により、AWS クエリエンジンを使用して Delta トランザクションログを直接クエリしたり、タイムトラベルや ACID 保証などの機能を使用できます。また、Amazon S3 トランザクションファイルの Delta Lake メタデータをデータカタログに同期して、Lake Formation のクエリに対する列のアクセス許可を有効にできます。詳細については、「Delta Lake データストアの設定オプションを指定する方法」および「Delta Lake テーブルのクエリを実行する」を参照してください。	2022 年 12 月 15 日

AWS Glue Data Quality (プレビュー) のサポート	AWS Glue Data Quality (プレビュー) が利用可能になりました。 AWS GlueData Quality は、AWS Glue 3.0 を使用する際、データの品質の評価とモニタリングを行うのに役立ちます。データカタログで AWS Glue Data Quality を使用する方法については、「AWS Glue Data Quality (プレビュー)」を参照してください。AWS Glue Studio での AWS Glue Data Quality の詳細については、「AWS Glue Studio によるデータ品質の評価」を参照してください。	2022 年 11 月 30 日
新しい機能が追加されパフォーマンスが向上した、新しい Amazon Redshift Spark コネクタのサポート	AWS Glue ETL ジョブで使用できる、新しい Amazon Redshift Spark コネクタおよび JDBC ドライバーがサポートされるようになりました。これにより、データの取り込みおよび変換パイプラインの一部として Amazon Redshift のデータを読み書きする Apache Spark アプリケーションを構築できます。詳細については、「Amazon Redshift との間でのデータの移動」を参照してください。	2022 年 11 月 29 日

AWS Glue バージョン 4.0 のサポート。	AWS Glue バージョン 4.0 のサポートに関する情報を追加しました。機能には、Apache Hudi、Delta Lake、Apache Iceberg でのオープンデータレイクフレームワークのネイティブサポートや、Amazon S3 ベースの Cloud Shuffle Storage Plugin (Apache Spark プラグイン) のネイティブサポートが含まれます。これにより、Amazon S3 を使用してシャッフルと伸縮自在なストレージ容量を実現できます。詳細については、AWS Glue リリースノートおよび「AWS Glue ジョブの AWS Glue バージョン 4.0 への移行」を参照してください。	2022 年 11 月 28 日
AWS Glue Studio がカスタムビジュアル変換機能をリリースしました。	カスタムビジュアル変換機能により、お客様がビジネス固有の ETL ロジックをチーム間で定義、再利用、共有できるようになりました。詳細については、「カスタムビジュアル変更機能」を参照してください。	2022 年 11 月 28 日

AWS Glue クローラーを使用した JDBC データストアのメタデータの公開をサポート	AWS Glue クローラーを使用して、コメントや rawtype などのメタデータを JDBC データストアのデータカタログに公開することができるようになりました。詳細については、「クローラーによって設定されたデータカタログテーブルのパラメータ」、「クローラーのプロパティ」、および「JdbcTarget 構造」を参照してください。	2022 年 11 月 18 日
Snowflake データストアのクローリングをサポート	AWS Glue を使用して、Snowflake のテーブルとビューをクロールしたり、メタデータをテーブルエントリとしてデータカタログに公開できるようになりました。Amazon S3 での Snowflake の外部テーブルの場合、クローラーは Amazon S3 の場所と外部テーブルのファイル形式のタイプもクロールし、テーブルパラメータとして入力します。詳細については、「クロール可能なデータストア」、「AWS Glue 接続プロパティ」および「クローラーによって設定されたデータカタログテーブルのパラメータ」を参照してください。	2022 年 11 月 18 日

Spark アプリケーションのシャッフル管理が改善	Cloud Shuffle Storage Plugin for Apache Spark が利用可能になりました。詳細については、「AWS Glue Spark シャッフルマネージャーと Amazon S3」および「Cloud Shuffle Storage Plugin for Apache Spark」を参照してください。	2022 年 11 月 15 日
Amazon S3 イベント通知のクロールを高速化するときのデータカタログターゲットに対するサポートが追加されました	Amazon S3 ターゲットに対する既存のサポートに加えて、Amazon S3 イベント通知を使用したデータカタログターゲットのクロールの高速化に対するサポートが利用可能になりました。詳細については、「Amazon S3 イベント通知を使用したクロールの高速化」を参照してください。	2022 年 10 月 13 日
クローラーが作成できるテーブルの最大数の指定をサポート	クローラーが作成できるテーブルの最大数を指定できるようになりました。詳細については、「How to specify the maximum number of tables the crawler is allowed to create」(クローラーが作成できるテーブルの最大数を指定する方法)を参照してください。	2022 年 9 月 6 日

AWS Glue 内の Python シェルジョブでの Python 3.9 のサポート	AWS Glue 内の Python シェルジョブで Python 3.9 との互換性のあるスクリプトを実行したり、事前にパッケージ済みのライブラリセットを選択できるようになりました。詳細については、「AWS Glue の Python シェルジョブ」を参照してください。	2022 年 8 月 11 日
緊急性のない、または時間的な制約のない AWS Glue ジョブの、予備の容量上での実行をサポート	実稼働前のジョブ、テスト、1 回限りのデータの読み込みなど、緊急性のないジョブに対する柔軟なジョブ実行を設定できるようになりました。詳細については、「AWS Glue でジョブを追加する」を参照してください。	2022 年 8 月 9 日
ストリーミングジョブ向けの新しいワーカータイプのサポート	少量のストリーミングジョブ用に G.025X ワーカータイプを使用できるようになりました。詳細については、「AWS Glue でジョブを追加する」を参照してください。	2022 年 7 月 14 日
AWS Glue 接続での Kafka SASL の使用のサポート	AWS Glue 接続で Kafka SASL の使用がサポートされるようになりました。詳細については、「AWS Glue Kafka connection properties for client authentication」(クライアント認証用の Kafka 接続プロパティ) を参照してください。	2022 年 7 月 5 日

Protobuf スキーマ用の Apache kafka コネクタサポート	現在、Protobuf スキーマ用の Apache Kafka コネクタサポートがご利用可能です。詳細については、「AWS Glue Schema Registry」を参照してください。	2022 年 6 月 9 日
AWS Glue ジョブ (GA) の Auto Scaling のサポート	AWS Glue バージョン 3.0 のジョブに Auto Scaling を使用して、コンピューティングリソースを動的にスケーリングする方法に関する情報を追加しました。　詳細については、「AWS Glue の Auto Scaling を利用する」を参照してください。	2022 年 4 月 14 日
AWS Glue 開発とAWS Glue ジョブスクリプトのテストに関するドキュメントの更新	Docker での開発手順を含む、AWS Glue 向けとして利用可能な開発とテストの方法に関する情報を再編成して追加しました。詳細については、「AWS Glue ETL ライブラリを使用した ETL スクリプトのローカルでの開発とテスト」を参照してください。	2022 年 3 月 14 日
AWS Glue スキーマレジストリでサポートされるデータ形式に protocol buffers (protobuf) を追加	サポートされているデータ形式として (AVRO および JSON に加えて) Protobuf に関する情報を追加しました。詳細については、「AWS Glue Schema Registry」を参照してください。	2022 年 2 月 25 日

Delta Lake テーブルのクローリングのサポート	AWS Glue を使用して Delta Lake テーブルをクロールする情報を追加しました。詳細については、「Delta Lake データストアの設定オプションを指定する方法」を参照してください。	2022 年 2 月 24 日
AWS Glue ジョブインサイトのサポート	AWS Glue ジョブインサイトの使用に関する情報を追加して、AWS Glue ジョブのジョブのデバッグと最適化を簡素化します。詳細については、「AWS Glue ジョブインサイトを使用したモニタリング」を参照してください。	2022 年 2 月 8 日
VPC エンドポイントを使用した Amazon S3 backedデータカタログテーブルのクロールのサポート	セキュリティ、監査、またはコントロールのために、Amazon Virtual Private Cloud 環境 (Amazon VPC) でのみアクセスするように Amazon S3 backed データカタログテーブルを設定する方法に関する情報を追加しました。詳細については、「Crawling an Amazon S3 Data Store or Amazon S3 backedData Catalog tables using a VPC Endpoint」を参照してください。	2022 年 2 月 3 日

Lake Formation ガバメントテーブルのSupport	ACID トランザクション、自動データ圧縮、および時間移動クエリを AWS Glue サポートするLake Formation 管理テーブルのサポートについての情報を追加しました 詳細については、「AWS Glue API および AWS Lake Formation デベロッパーガイド」を参照してください。	2021 年 11 月 30 日
インタラクティブセッションおよびノートブックに追加された新規の AWS マネージドポリシー	IAM の新しいマネージドポリシーにより、AWS Glue インタラクティブなセッションとノートブックで使用するためのセキュリティが強化されました。詳細については、「AWS Glue の AWS マネージドポリシー」を参照してください。	2021 年 11 月 30 日
Glue スキーマレジストリがストリーミングジョブでサポートされるようになりました	Glue スキーマレジストリの一部であるテーブルにアクセスするストリーミングジョブを作成できます。詳細については、「AWS Glue Schema Registry とストリーミング ETL ジョブを追加 AWS Glue」を参照してください。	2021 年 11 月 15 日

新しい機械学習機能のサポート	インクリメンタルマッチングやマッチスコアリングなど、機械学習変換の新機能に関する情報を追加しました。詳細については、「インクリメンタルマッチの検索」と「マッチの信頼度スコアを使用した一致の質の推定」を参照してください。	2021 年 10 月 31 日
(プライベートプレビュー) AWS Glue フレックスジョブをサポート	柔軟な実行クラスを持つ AWS Glue Spark ジョブの設定に関する情報が追加されました。これは、開始時間と完了時間が異なる時間的な制約のあるジョブに適しています。詳細については、「AWS Glue でジョブを追加する」を参照してください。	2021 年 10 月 29 日
Amazon S3 イベント通知を使用したクロールの高速化のサポート	Amazon S3 イベント通知を使用したクロールの高速化のサポート 詳細については「クロールの高速化のサポートを使用した Amazon S3 イベント通知」を参照してください。	2021 年 10 月 15 日

アクセスコントロールおよび VPC に関連するセキュリティ設定オプションの追加	AWS Glue および VPC の設定で新しいアクセス制御アクセス許可を設定する方法についての情報を追加しました。詳細については「AWS のタグ AWS Glue」、「条件キーまたはコンテキストキーを使用して設定を制御する ID ベースのポリシー (IAM ポリシー)」、「すべての設定AWSVPC を通過するためのコール」を参照してください。	2021 年 10 月 13 日
VPC エンドポイントポリシーのサポートが追加されました	AWS Glue でVirtual Private Cloud (VPC) エンドポイントポリシーのサポートに関する情報を追加しました。詳細については、「AWS Glue とインターフェイス VPC エンドポイント (AWS PrivateLink)」を参照してください。	2021 年 10 月 11 日
Glue Studio が中国で利用可能になりました	AWS Glue Studio が北京および寧夏リージョンで利用可能になりました。	2021 年 10 月 11 日
AWS Glue Studio が、インタラクティブなジョブ編集におけるノートブックのオーサリング機能をリリースしました。	ノートブックはコードの記述と実行、結果の可視化、およびインサイトの共有に役立ちます。通常データサイエンティストは、ノートブックを使用して実験やデータ探索のタスクを行います。詳細については、「ノートブックの使用」を参照してください。	2021 年 10 月 1 日

ストリーミングソースへの直接アクセスが可能になりました	ビジュアルエディタで ETL ジョブにデータソースを追加する場合、データカタログデータベースとテーブルを使用せずに、データストリームにアクセスするための情報を指定できます。	2021 年 9 月 30 日
AWS Glue バージョンのサポートポリシーのドキュメント化	AWS Glue バージョンサポートポリシーと、特定の期間終了フェーズ AWS Glue バージョンに関する情報を追加しました。詳細については、「AWS Glue バージョンサポートのポリシー」を参照してください。	2021 年 9 月 24 日
カスタムコネクタをデータプレビューで使用できるようになりました	カスタムコネクタを使用してデータソースノードを編集する場合、[データプレビュー] タブを選択してデータセットをプレビューできます。詳細については、「カスタムコネクタ」を参照してください。	2021 年 9 月 24 日

サポート対象 AWS Glue インタラクティブセッション (プライベートプレビュー)	(プライベートプレビュー) の使用方法に関する情報を追加。任意の Jupyter Notebook から Spark ワークロードをクラウドで実行するための AWS Glue 対話型セッション。インタラクティブセッションは AWS Glue 2.0以降を使用する場合、AWS Glue 抽出、変換、ロード (ETL) のコードを開発するための好ましい方法です。詳細については、「Jupyter Notebook の AWS Glue インタラクティブセッションのセットアップと実行」を参照してください。	2021 年 8 月 24 日
ブループリントからのワークフロー作成のサポート (GA)	ブループリントでの一般的な抽出、変換、ロード (ETL) ユースケースのコーディングと、その後のブループリントからのワークフローの作成に関する情報を追加しました。データアナリストが複雑な ETL プロセスを簡単に作成して実行できるようにしています。詳細については、「Performing Complex ETL Activities Using blueprints and Workflows in AWS Glue」を参照してください。	2021 年 8 月 23 日

AWS Glue バージョン 3.0 のサポート。	AWS Glue バージョン 3.0 のサポートについての情報を追加しました。Apache Spark ETL ジョブを実行するための Apache Spark 3.0 エンジンのアップグレード、およびその他の最適化とアップグレードをサポートしています。詳細については、AWS Glue リリースノートおよび「Migrating AWS Glue jobs to AWS Glue version 3.0」を参照してください。このリリースの特徴には、他に、AWS Glue シャッフルマネージャー、SIMD ベクトル化された CSV リーダー、カタログパーティションの述語などがあります。詳細については、「AWS Glue Spark shuffle manager with Amazon S3」、「Format Options for ETL Inputs and Outputs in AWS Glue」、および「Server-side filtering using catalog partition predicates」を参照してください。	2021 年 8 月 18 日
AWS GovCloud (US) Region	AWS GovCloud (US) Region で、AWS Glue Studio が使用可能になりました	2021 年 8 月 18 日

AWS Glue Studio で Python シェルのオーサリングが使用可能になりました	作成する新しいジョブとして、Python シェルジョブが選択できるようになりました。詳細については、ジョブの作成プロセスの開始 および Editing Python shell jobs in AWS Glue Studio を参照してください。	2021 年 8 月 13 日
Amazon EventBridge イベントでワークフローを開始するためのサポート	イベント駆動型アーキテクチャで AWS Glue がイベントを使用する方法についての情報を追加しました。詳細については、「Starting an AWS Glue Workflow with an Amazon EventBridge Event」および「Viewing the EventBridge Events That Started a Workflow」を参照してください。	2021 年 7 月 14 日
AWS Glue スキーマレジストリでサポートされるデータ形式として JSON を追加	サポートされているデータ形式として JSON に関する情報を追加しました (AVRO に加えて)。詳細については、「AWS Glue Schema Registry」を参照してください。	2021 年 6 月 30 日
データカタログテーブルなしで AWS Glue ストリーミングジョブを作成	`create_data_frame_from_options` Python 関数または `getSource` for Scala スクリプトは、データカタログテーブルを必要とせず、データストリームを直接参照するストリーミング ETL ジョブの作成をサポートします。	2021 年 6 月 15 日

AWS Glue 機械学習変換で AWS Key Management Service キーのサポート開始	コンソール、CLI、または AWS Glue API で、AWS Glue 機械学習変換設定時に、セキュリティ設定または AWS KMS キーを指定できます。詳細については、「Using Data Encryption with Machine Learning Transforms」および「AWS Glue Machine Learning API」を参照してください。	2021 年 6 月 15 日
AWSGlueConsoleFullAccess AWS マネージドポリシーのアップデート	AWSGlueConsoleFullAccess AWS マネージドポリシーのマイナーアップデートに関する情報を追加しました。詳細については、「AWS Glue での AWS マネージドポリシーの更新」を参照してください。	2021 年 6 月 10 日
ジョブの作成および編集中にジョブのデータセットを表示する	ジョブダイアグラム内のノードで、新規に追加された [Data preview] (データのプレビュー) タブを使用すると、そのノードによって処理されるデータのサンプルを表示できます。詳細については、「Using data previews in the visual job editor」を参照してください。	2021 年 6 月 7 日
クローラー出力のテーブルの場所を示す値の指定をサポート	クローラーの出力設定時、テーブルの場所を示す値の指定に関する情報を追加しました。詳細については、「How to specify the table location」を参照してください。	2021 年 6 月 4 日

Amazon S3 データストアのクロール時、データセット内のファイルのサンプルのクロールをサポート	Amazon S3 をクロールするときにファイルのサンプルをクロールする方法に関する情報を追加しました。詳細については、「クローラーのプロパティ」を参照してください。	2021 年 5 月 10 日
AWS Glue 最適化された parquet ライターのサポート	DynamicFrames 用の AWS Glue 最適化された parquet ライターを使用して `parquet` 分類されたテーブルを作成または更新する方法についての情報を追加しました。詳細については、「Creating Tables, Updating Schema, and Adding New Partitions in theデータカタログfrom AWS Glue ETL Jobs」および「Format Options for ETL Inputs and Outputs in AWS Glue」を参照してください。	2021 年 5 月 4 日

Kafka クライアント認証パスワードのサポート	Apache Kafka ストリームプロデューサーを使用した AWS Glue でサポートされる SSL クライアント証明書認証での ETL ジョブのストリーミング方法に関する情報を追加しました。Apache Kafka クラスターへの AWS Glue 接続を定義する際に、カスタム証明書を指定できるようになりました。これは、認証時に AWS Glue で使用されます。詳細については、「AWS Glue Connection Properties」および「Connection API」を参照してください。	2021 年 4 月 28 日
ストリーミング ETL ジョブで、別のアカウントの Amazon Kinesis Data Streams からのデータの使用をサポート	ストリーミング ETL ジョブの作成による別のアカウントの Amazon Kinesis Data Streams からのデータの使用に関する情報を追加しました。詳細については、「AWS Glue でストリーミング ETL ジョブを追加する」を参照してください。	2021 年 3 月 30 日
SQL 変換が使用可能になりました	SQL の変換ノードを使用して、SQL クエリの形式で独自の変換を記述できます。詳細については、「Using a SQL query to transform data」を参照してください。	2021 年 3 月 23 日

ブループリントからのワークフロー作成のサポート (公開プレビュー)	(公開プレビュー) ブループリントでの一般的な抽出、変換、ロード (ETL) のユースケースのコーディングと、その後のブループリントからのワークフローの作成に関する情報を追加しました。データアナリストが複雑な ETL プロセスを簡単に作成して実行できるようにしています。詳細については、「Performing Complex ETL Activities Using blueprints and Workflows in AWS Glue」を参照してください。	2021 年 3 月 22 日
データターゲットでコネクタが使用可能になりました	カスタムまたは AWS Marketplace で提供されたコネクタを、データターゲット用に使用できるようになりました。詳細については、「Authoring jobs with custom connectors」を参照してください。	2021 年 3 月 15 日
AWS Glue 機械学習変換の列重要度メトリクスのサポート	AWS Glue 機械学習変換使用時の列重要度メトリクスの表示に関する情報が追加されました。詳細については、「Working with Machine Learning Transforms on the AWS Glue Console」を参照してください。	2021 年 2 月 5 日

AWS Glue Studio でジョブのスケジューリングが使用可能になりました	AWS Glue Studio で時間ベースのスケジュールをジョブ実行用に定義できます。コンソールを使用して基本的なスケジュールを作成したり、Unix ライクの cron 構文を使用して、より複雑なスケジュールを定義したりできます。詳細については、「Schedule job runs」を参照してください。	2020 年 12 月 21 日
AWS Glue カスタムコネクタがリリースされました	AWS Glue の Custom Connectors を使用すると、AWS Marketplace 内でコネクターを検索およびサブスクライブできます。また、Apache Spark データソース、Athena フェデレーティッドクエリ、および JDBC API 用に構築されたコネクタをプラグインするための、AWS Glue Spark ランタイムインターフェイスもリリースしました。詳細については、「Using connectors and connections with AWS Glue Studio」を参照してください。	2020 年 12 月 21 日

AWS Glue バージョン 2.0でストリーミング ETL ジョブをサポート	Glue バージョン 2.0 でのストリーミング ETL ジョブの実行に関するサポートに関する情報を追加しました。詳細については、「AWS Glue でストリーミング ETL ジョブを追加する」を参照してください。	2020 年 12 月 18 日
実行に上限を設定したワークロードのパーティション化のサポート	ワークロードのパーティション化を有効にして、データセットサイズまたは ETL ジョブ実行で処理されるファイル数の上限を設定する方法に関する情報を追加しました。詳細については、「Workload Partitioning with Bounded Execution」を参照してください。	2020 年 11 月 23 日
拡張パーティション管理のサポート	新しい API を使用して、既存のテーブルに対してパーティションインデックスを追加または削除する方法に関する情報を追加しました。詳細については、「Working with Partition Indexes」を参照してください。	2020 年 11 月 23 日
AWS Glue スキーマレジストリのサポート	AWS Glue スキーマレジストリを使用して、スキーマを一元的に検出し、コントロールし、進化させる方法に関する情報を追加しました。詳細については、「AWS Glue Schema Registry」を参照してください。	2020 年 11 月 19 日

ストリーミング ETL ジョブでの grok 入力フォーマットのサポート	ログファイルなどのストリーミングソースへの Grok パターンの適用に関する情報が追加されました。詳細については、「Applying Grok Patterns to Streaming Sources」を参照してください。	2020 年 11 月 17 日
AWS Glue コンソールでのワークフローへのタグの追加をサポート	AWS Glue コンソールを使用してワークフローを作成するときのタグの追加に関する情報を追加しました。詳細については、「Creating and Building Out a Workflow Using the AWS Glue Console」を参照してください。	2020 年 10 月 27 日
増分クローラー実行のサポート	前回の実行以降に追加された Amazon S3 フォルダのみをクロールする増分クローラー実行のサポートに関する情報を追加しました。詳細については、「Incremental Crawls」を参照してください。	2020 年 10 月 21 日

ストリーミング ETL データソースのスキーマ検出のサポート。Avro ストリーミング ETL データソースとセルフマネージド型の kafkのサポート。	AWS Glue でのストリーミング抽出、変換、ロード (ETL) ジョブで、受信レコードのスキーマを自動的に検出し、レコードごとにスキーマの変更を処理できるようになりました。自己管理型 Kafka データソースがサポートされるようになりました。ストリーミング ETL ジョブで、データソースの Avro 形式がサポートされるようになりました。詳細については、「Streaming ETL in AWS Glue」、「Defining Job Properties for a Streaming ETL Job」、および「Notes and Restrictions for Avro Streaming Sources」を参照してください。	2020 年 10 月 7 日
MongoDB および DocumentDB データソースのクロールのサポート	MongoDB および Amazon DocumentDB (MongoDB 互換) データソースをクロールするためのサポートに関する情報を追加しました。詳細については、「Defining Crawlers」を参照してください。	2020 年 10 月 5 日

FIPS コンプライアンスのサポート	AWS Glue を使用してデータにアクセスするときに FIPS 140-2 検証済みの暗号化モジュールを必要とするお客様のための FIPS エンドポイントに関する情報を追加しました。詳細については、「FIPS Compliance」を参照してください。	2020 年 9 月 23 日
AWS Glue Studio Glue Studio での、ジョブを作成およびモニタリングするための使いやすいビジュアルインターフェイスの提供	シンプルなグラフベースのインターフェイスを使用して、データを移動および変換するジョブを作成し、AWS Glue で実行できるようになりました。その後、AWS Glue Studio Glue Studio のジョブ実行ダッシュボードを使用して ETL の実行をモニタリングし、ジョブが意図したとおりに動作していることを確認できます。詳細については、「AWS Glue Studioユーザーガイド」を参照してください。	2020 年 9 月 23 日
クエリのパフォーマンスを向上させるためのテーブルインデックス作成のサポート	テーブルからパーティションのサブセットを取得できるようにするテーブルインデックスの作成に関する情報が追加されました。詳細については、「Working with Partition Indexes」を参照してください。	2020 年 9 月 9 日

Apache Spark ETL ジョブを実行する際のスタートアップ時間の削減を AWS Glue バージョン 2.0 でサポートしました。	AWS Glue バージョン 2.0 のサポートについての情報を追加しました。このバージョンでは、Apache Spark ETL ジョブを実行するインフラストラクチャがアップグレードされ、スタートアップ時間の短縮、ログ記録の変更、ジョブレベルでの追加の Python モジュール指定のサポートが行われます。詳細については、「AWS Glue Release Notes」および「Running Spark ETL Jobs with Reduced Startup Times」を参照してください。	2020 年 8 月 10 日
同時ワークフロー実行数の制限のサポート。	特定のワークフローの同時ワークフロー実行数を制限する方法に関する情報を追加しました。詳細については、「Creating and Building Out a Workflow Using the AWS Glue Console」を参照してください。	2020 年 8 月 10 日
VPC エンドポイントを使用した Amazon S3 データストアのクロールのサポート	セキュリティ、監査、またはコントロールのために、Amazon Virtual Private Cloud 環境 (Amazon VPC) でのみアクセスするように Amazon S3 データストアを設定する方法に関する情報を追加しました。詳細については、「Crawling an Amazon S3 Data Store using a VPC Endpoint」を参照してください。	2020 年 8 月 7 日

ワークフロー実行再開のサポート	1 つ以上のノード (ジョブまたはクローラー) が正常に完了しなかったために部分的にしか完了しなかったワークフロー実行を再開する方法に関する情報を追加しました。詳細については、「Repairing and Resuming a Workflow Run」を参照してください。	2020 年 7 月 27 日
の kafka 接続でのプライベート CA 証明書の有効化をサポート。AWS Glue	AWS Glue の Kafka 接続のプライベート CA 証明書を有効にすることをサポートする新しい接続オプションに関する情報を追加しました。詳細については、「Connection Types and Options for ETL in AWS Glue」および「Special Parameters Used by AWS Glue」を参照してください。	2020 年 7 月 20 日
別のアカウント内の DynamoDB データ読み取りのサポート	別の AWS アカウントの DynamoDB テーブルからのデータの読み取りの AWS Glue サポートに関する情報を追加しました。詳細については、「Reading from DynamoDB Data in Another Account」を参照してください。	2020 年 7 月 17 日

AWS Glue バージョン 1.0 以降での DynamoDB ライター接続のサポート	DynamoDB ライター、および DynamoDB の読み書き用の新しい接続オプションまたは更新された接続オプションのサポートに関する情報を追加しました。詳細については、「Connection Types and Options for ETL in AWS Glue」を参照してください。	2020 年 7 月 17 日
リソースリンクのサポート AWS Glue およびand Lake Formationを用いたアカウント横断アクセス制御のサポート	リソースリンクと呼ばれる新しいデータカタログオブジェクトに関するコンテンツ、および AWS Glue と AWS Lake Formation の両方でアカウント間のデータカタログリソース共有を管理する方法に関するコンテンツを追加しました。詳細については、「Granting Cross-Account Access」および「Table Resource Links」を参照してください。	2020 年 7 月 7 日
DynamoDB データストアをクロールするときのレコードのサンプリングのサポート	DynamoDB データストアのクロール時に設定できる新しいプロパティに関する情報を追加しました。詳細については、「クローラーのプロパティ」を参照してください。	2020 年 6 月 12 日
ワークフロー実行停止のサポート。	特定のワークフローのワークフロー実行を停止する方法に関する情報を追加しました。詳細については、「ワークフロー実行の停止」を参照してください。	2020 年 5月 14 日

Spark ストリーミング ETL ジョブのサポート	ストリーミングデータソースを使用した抽出/変換/ロード (ETL、Extract/Transform/Load) ジョブの作成に関する情報を追加しました。詳細については、「AWS Glue でストリーミング ETL ジョブを追加する」を参照してください。	2020 年 4 月 27 日
ETL ジョブの実行後のテーブルの作成、スキーマの更新、データカタログ での新規パーティション追加のサポート	テーブルの作成、スキーマの更新、およびデータカタログでの ETL ジョブの結果を確認するための新しいパーティションの追加を有効にする方法についての情報が追加されました。詳細については、「Creating Tables, Updating Schema, and Adding New Partitions in theデータカタログfrom AWS Glue ETL Jobs」を参照してください。	2020 年 4 月 2 日
AWS Glue で、ETL 入力および出力として Apache Avro データ形式のバージョン指定のサポート	Apache Avro データ形式のバージョンを AWS Glue の ETL 入力および出力として指定する方法についての情報を追加しました。デフォルトバージョン 1.7。`version` フォーマットオプションを使用して、Avro バージョン 1.8 を指定して、論理読み取り / 書き込みを有効にすることができます。詳細については、「Format Options for ETL Inputs and Outputs in AWS Glue」を参照してください。	2020 年 3 月 31 日

Parquet データを Amazon S3 に書き込むための EMRFS S3 最適化コミッターのサポート	AWS Glue ジョブの作成または更新時に、EMRFR S3 向けに最適化されたコミッターが Parquet データを Amazon S3 に書き込むための新しいフラグを設定する方法に関する情報を追加しました。詳細については、「AWS Glue で使用される特別なパラメータ」を参照してください。	2020 年 3 月 30 日
AWS リソースタグによって管理されるリソースとしての機械学習変換のサポート	AWS リソースタグを使用して AWS Glue での機械学習変換へのアクセスを管理およびコントロールする方法に関する情報を追加しました。AWS Glue で、AWS リソースタグをジョブ、トリガー、エンドポイント、クローラー、機械学習変換に割り当てることができます。詳細については、「AWS Tags in AWS Glue」を参照してください。	2020 年 3 月 2 日
上書きできないジョブ引数のサポート	トリガーまたはジョブの実行時に上書きできない特殊なジョブパラメータのサポートに関する情報を追加しました。詳細については、「AWS Glue でジョブを追加する」を参照してください。	2020 年 2 月 12 日

Amazon S3 でデータセットを操作するための新しい変換のサポート	Amazon S3 でデータセットを操作する Apache Spark アプリケーションの新しい変換 (マージ、パージ、遷移) と Amazon S3 ストレージクラスの除外に関する情報を追加しました。Python でのこれらの変換のサポートの詳細については、「mergeDynamicFrame」および「Working with Datasets in Amazon S3」を参照してください。Scala については、「mergeDynamicFrames」および「AWS Glue Scala GlueContext APIs」を参照してください。	2020 年 1 月 16 日
ETL ジョブからの新しいパーティション情報を使用したデータカタログ更新のサポート	新しいパーティション情報で AWS Glue Data Catalog を更新するために、抽出、変換、ロード (ETL) スクリプトをコーディングする方法についての情報を追加しました。この機能を使用すると、新しいパーティションを表示するためにジョブの完了後にクローラーを再実行する必要がなくなります。詳細については、「Updating the Data Catalog with New Partitions」を参照してください。	2020 年 1 月 15 日

新しいチュートリアル: SageMaker AI ノートブックの使用	Amazon SageMaker ノートブックを使用して ETL および機械学習スクリプトを開発する方法を示すチュートリアルを追加しました。「チュートリアル: 開発エンドポイントで Amazon SageMaker ノートブックを使用する」を参照してください。	2020 年 1 月 3 日
MongoDB および Amazon DocumentDB (MongoDB 互換) からの読み取りのサポート	MongoDB と Amazon DocumentDB (MongoDB 互換) との読み書き用の新しい接続タイプと接続オプションに関する情報を追加しました。詳細については、「Connection Types and Options for ETL in AWS Glue」を参照してください。	2019 年 12 月 17 日
さまざまな修正と説明	全体にわたって修正と説明を追加しました。「既知の問題」の章からエントリを削除しました。データカタログの暗号化設定を指定し、セキュリティ設定を作成するときに、AWS Glue は対称カスタマーマスターキー (CMK) のみをサポートすることを知らせる警告が追加されました。AWS Glue が Amazon DynamoDB への書き込みをサポートしていないことを示す注意を追加しました。	2019 年 12 月 9 日

カスタム JDBC ドライバーのサポート	MySQL バージョン 8 や Oracle Database バージョン 18 など、AWS Glue がネイティブでサポートしていない JDBC ドライバーを使用したデータソースおよびターゲットへの接続に関する情報を追加しました。詳細については、「JDBC connectionType の値」を参照してください。	2019 年 11 月 25 日
SageMaker AI ノートブックをさまざまな開発エンドポイント接続するためのサポート	SageMaker AI ノートブックをさまざまな開発エンドポイントに接続する方法に関する情報を追加しました。新しい開発エンドポイントに切り替えるための新しいコンソールアクションと新しい SageMaker AI IAM ポリシーに関する説明を更新しました。詳細については、「Working with Notebooks on the AWS Glue Console」および「Amazon SageMaker ノートブック用に IAM ポリシーを作成する」を参照してください。	2019 年 11 月 21 日

機械学習変換での AWS Glue バージョンのサポート	機械学習変換と互換性のある AWS Glue のバージョンを示すため、機械学習変換での AWS Glue バージョンの定義に関する情報を追加しました。詳細については、「Working with Machine Learning Transforms on the AWS Glue Console」を参照してください。	2019 年 11 月 21 日
ジョブブックマークの巻き戻しのサポート	ジョブのブックマークを以前のジョブ実行に巻き戻すことに関する情報を追加しました。その結果、後続のジョブ実行ではブックマークされたジョブ実行からのデータだけが再処理されます。2 つのブックマーク間でジョブを実行できる `job-bookmark-pause` オプションの 2 つの新しいサブオプションについて説明しました。詳細については、「Tracking Processed Data Using Job Bookmarks」および「Special Parameters Used by AWS Glue」を参照してください。	2019 年 10 月 22 日

データストアに接続するためのカスタム JDBC 証明書のサポート	AWS Glue データソースまたはターゲットへの SSL 接続用のカスタム JDBC 証明書の AWS Glue サポートに関する情報を追加しました。詳細については、「Working with Connections on the AWS Glue Console」を参照してください。	2019 年 10 月 10 日
Python Wheel のサポート	Python シェルジョブの依存関係として wheel ファイル (egg ファイルと併用) の AWS Glue サポートに関する情報を追加しました。詳細については、「独自の Python ライブラリの提供」を参照してください。	2019 年 9 月 26 日
AWS Glue での開発エンドポイントのバージョニングのサポート	開発エンドポイントでの `Glue version` の定義に関する情報を追加しました。`Glue version` により、AWS Glue がサポートする Apache Spark および Python のバージョンが決定されます。詳細については、「開発エンドポイントの追加」を参照してください。	2019 年 9 月 19 日

Spark UI を使用した AWS Glue モニタリングのサポート	Apache Spark UI を使用して、AWS Glue ジョブシステムで実行されている AWS Glue ETL ジョブと AWS Glue 開発エンドポイントの Spark アプリケーションのモニタリングとデバッグに関する情報を追加しました。詳細については、「Monitoring AWS Glue Using Spark UI」を参照してください。	2019 年 9 月 19 日
パブリック AWS Glue ETL ライブラリを使用したローカル ETL スクリプト開発のサポートの強化	AWS Glue ETL ライブラリのコンテンツを更新して、AWS Glue バージョン 1.0 がサポートされたことを反映しました。詳細については、「Developing and Testing ETL Scripts Locally Using the AWS Glue ETL Library」を参照してください。	2019 年 9 月 18 日
ジョブ実行時の Amazon S3 ストレージクラス除外のサポート	Amazon S3 からファイルまたはパーティションを読み取る AWS Glue ETL ジョブを実行する際の Amazon S3 ストレージクラスの除外に関する情報を追加しました。詳細については、「Excluding Amazon S3 Storage Classes」を参照してください。	2019 年 8 月 29 日

パブリック AWS Glue ETL ライブラリを使用したローカル ETL スクリプト開発のサポート	ネットワーク接続を必要とせずに Python および Scala ETL スクリプトをローカルで開発およびテストする方法に関する情報を追加しました。詳細については、「Developing and Testing ETL Scripts Locally Using the AWS Glue ETL Library」を参照してください。	2019 年 8 月 28 日
既知の問題	AWS Glue の既知の問題に関する情報を追加しました。詳細については、「AWS Glue の既知の問題」を参照してください。	2019 年 8 月 28 日
AWS Glueでの機械学習変換のサポート	カスタム変換を作成するために AWS Glue によって提供される機械学習機能に関する情報を追加しました。これらの変換は、ジョブの作成時に作成できます。機械学習変換の詳細については、「AWS Glue の機械学習変換」を参照してください。	2019 年 8 月 8 日
共有 Amazon Virtual Private Cloud のサポート	共有 Amazon Virtual Private Cloud の AWS Glue サポートに関する情報を追加しました。詳細については、「共有 Amazon VPC」を参照してください。	2019 年 8 月 6 日

AWS Glue でのバージョニングのサポート	ジョブプロパティでの `Glue version` の定義に関する情報を追加しました。AWS Glue バージョンにより、AWS Glue がサポートする Apache Spark および Python のバージョンが決定されます。詳細については、「AWS Glue でジョブを追加する」を参照してください。	2019 年 7 月 24 日
開発エンドポイントの追加設定オプションのサポート	メモリを大量に使用するワークロードがある開発エンドポイントの設定オプションに関する情報を追加しました。エグゼキューターあたりのメモリを増強した新しい 2 つの設定からも選択できます。詳細については、「AWS Glue コンソールでの開発エンドポイントの使用」を参照してください。	2019 年 7 月 24 日
ワークフローを使用した ETL (抽出、変換、ロード) アクティビティの実行のサポート	ワークフローと呼ばれる新しい構成を使用して、複雑なマルチジョブの抽出、変換、ロード (ETL) アクティビティを設計し、これを AWS Glue で単一のエンティティとして実行および追跡する機能に関する情報を追加しました。詳細については、「Performing Complex ETL Activities Using Workflows in AWS Glue」を参照してください。	2019 年 6 月 20 日

Python シェルジョブでの Python 3.6 のサポート	Python シェルジョブでの Python 3.6 のサポートに関する情報を追加しました。Python 3.6 または Python 2.7 をジョブプロパティとして指定できます。詳細については、「Adding Python Shell Jobs in AWS Glue」を参照してください。	2019 年 6 月 5 日
Virtual Private Cloud (VPC) エンドポイントのサポート	VPC のインターフェイスエンドポイントを介した AWS Glue への直接接続に関する情報を追加しました。VPC インターフェイスエンドポイントを使用すると、AWS ネットワーク内で VPC と AWS Glue 間の通信が完全かつ安全に実施されます。詳細については、「Using AWS Glue with VPC Endpoints」を参照してください。	2019 年 6 月 4 日
AWS Glue ジョブのリアルタイムの連続ログ記録のサポート。	CloudWatch でのリアルタイムの Apache Spark ジョブログの有効化と表示に関する情報を追加しました。これには、ドライバーログ、各エグゼキューターログ、および Spark ジョブの進行状況バーが含まれます。詳細については、「AWS Glue ジョブの連続ログ記録」を参照してください。	2019 年 5 月 28 日

クローラーのソースとして既存のデータカタログテーブルをサポート	既存のデータカタログテーブルのリストをクローラーのソースとして指定することに関する情報を追加しました。新しいデータが利用可能になると、クローラーは、テーブルスキーマの変更を検出し、テーブル定義を更新して、新しいパーティションを登録できます。詳細については、「クローラーのプロパティ」を参照してください。	2019 年 5 月 10 日
メモリ大量使用ジョブ向け追加設定オプションのサポート	メモリを集中的に使用する作業での Apache Spark ジョブの設定オプションに関する情報を追加しました。エグゼキューターあたりのメモリを増強した新しい 2 つの設定からも選択できます。詳細については、「AWS Glue でジョブを追加する」を参照してください。	2019 年 4 月 5 日
CSV カスタム分類子のサポート	カスタム CSV 分類子を使用して、さまざまな種類の CSV データのスキーマを推測することに関する情報を追加しました。詳細については、「カスタム分類子の書き込み」を参照してください。	2019 年 3 月 26 日

AWS リソースタグのサポート	AWS Glue リソースへのアクセスを管理およびコントロールするのに役立つ AWS リソースタグの使用に関する情報を追加しました。AWS リソースタグを AWS Glue のジョブ、トリガー、エンドポイント、およびクローラーに割り当てることができます。詳細については、「AWS Tags in AWS Glue」を参照してください。	2019 年 3 月 20 日
Spark SQL ジョブ用データカタログのサポート	AWS Glue Data Catalog を外部の Apache Hive メタストアとして使用するための AWS Glue ジョブと開発エンドポイントの設定に関する情報を追加しました。これにより、ジョブおよび開発エンドポイントは AWS Glue Data Catalog に格納されているテーブルに対して Apache Spark SQL クエリを直接実行できます。詳細については、「AWS Glue Data Catalog Support for Spark SQL Jobs」を参照してください。	2019 年 3 月 14 日
Python シェルジョブのサポート	Python シェルジョブと新しいフィールド [最小キャパシティー] についての情報を追加しました。詳細については、「Adding Python Shell Jobs in AWS Glue」を参照してください。	2019 年 1 月 18 日

データベースおよびテーブル変更通知のサポート	データベース、テーブル、およびパーティション API 呼び出しへの変更対して生成されるイベントに関する情報を追加しました。CloudWatch Events で、これらのイベントに応答するようにアクションを設定できます。詳細については、「Automating AWS Glue with CloudWatch Events」を参照してください。	2019 年 1 月 16 日
接続パスワード暗号化のサポート	接続オブジェクトで使用されるパスワードの暗号化についての情報を追加しました。詳細については、「接続パスワードの暗号化と復号」を参照してください。	2018 年 12 月 11 日
リソースレベルのアクセス許可とリソースに基づくポリシーのサポート	AWS Glue でリソースレベルのアクセス許可とリソースに基づくポリシーを使用することに関する情報を追加しました。詳細については、「AWS Glue でのセキュリティ」のトピックを参照してください。	2018 年 10 月 15 日
SageMaker AI ノートブックのサポート	AWS Glue 開発エンドポイントで SageMaker AI ノートブックを使用することに関する情報を追加しました。詳細については、「ノートブックの管理」を参照してください。	2018 年 10 月 5 日

暗号化のサポート	AWS Glue で暗号化を使用することに関する情報を追加しました。詳細については、「Encryption at Rest」、「Encryption in Transit」、および「Setting Up Encryption in AWS Glue」を参照してください。	2018 年 8 月 24 日
Apache Spark ジョブメトリクスのサポート	ETL のデバッグとプロファイリングを強化するため、Apache Spark メトリクスの使用に関する情報を追加しました。読み取りおよび書き込みされたバイト数、ドライバーとエグゼキューターのメモリ使用量と CPU 負荷、エグゼキューター間のデータシャッフルなどのランタイムメトリクスを AWS Glue コンソールから簡単に追跡できます。詳細については、「Monitoring AWS Glue Using CloudWatch Metrics」、「Job Monitoring and Debugging」、および「Working with Jobs on the AWS Glue Console」を参照してください。	2018 年 7 月 13 日

データソースとしての DynamoDB のサポート	DynamoDB をクロールし、ETL ジョブのデータソースとして使用する方法に関する情報を追加しました。詳細については、「クローラーを使用してテーブルを分類する」と「接続パラメータ」を参照してください。	2018 年 7 月 10 日
ノートブックサーバー作成手順の更新	開発エンドポイントに関連付けられた Amazon EC2 インスタンスでノートブックサーバーを作成する方法に関する情報を更新しました。詳細については、「開発エンドポイントに関連付けられているノートブックサーバーを作成する」を参照してください。	2018 年 7 月 9 日
更新を RSS で今すぐ入手可能	RSS フィードにサブスクライブすると、AWS Glue デベロッパーガイドの更新に関する通知を受け取れるようになりました。	2018 年 6 月 25 日
ジョブ遅延通知のサポート	ジョブ実行時の遅延しきい値の設定に関する情報を追加しました。詳細については、「AWS Glue でジョブを追加する」を参照してください。	2018 年 5 月 25 日
新しい列を追加するようにクローラーを設定する	クローラー、MergeNewColumns の新しい設定オプションについての情報を追加しました。詳細については、「クローラーの設定」を参照してください。	2018 年 5 月 7 日

ジョブタイムアウトのサポート	ジョブ実行時のタイムアウトしきい値の設定に関する情報を追加しました。詳細については、「AWS Glue でジョブを追加する」を参照してください。	2018 年 4 月 10 日
Scala ETL スクリプトと追加の実行条件に基づくジョブのトリガーのサポート	Scala を ETL プログラミング言語として使用することについての追加の情報。さらに、トリガー API は、(すべての条件に加えて) いずれかの条件が満たされたときの発生をサポートするようになりました。また、ジョブは、(「succeeded」ジョブ実行に加えて)「failed」または「stopped」のジョブ実行に基づいてトリガーすることができます。	2018 年 1 月 12 日

以前の更新

次の表に、2018 年 1 月以前の AWS Glue 開発者ガイドの各リリースにおける重要な変更点を示します。

変更	説明	日付
XML データソースと新しいクローラー設定オプションをサポート	XML データソースとパーティション変更の新しいクローラーオプションについての情報を追加しました。	2017 年 11 月 16 日
新しい変換、追加の Amazon RDS データベースエン	マップとフィルターの変換、Amazon RDS Microsoft SQL Server と Amazon RDS Oracle のサポート、および開発エン	2017 年 9 月 29 日

変更	説明	日付
ジンのサポート、および開発エンドポイントの機能強化	ドポイントの新機能に関する情報を追加しました。	
AWS Glue 初回リリース	これは AWS Glue デベロッパーガイドの最初のリリースです。	2017 年 8 月 14 日

AWS 用語集

AWS の最新の用語については、「AWS の用語集リファレンス」の「AWS 用語集」を参照してください。