SQL 開発者ガイド

Amazon Kinesis Data Analytics for SQL Applications 開発者ガイド

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

Amazon Kinesis Data Analytics for SQL Applications 開発者ガイド: SQL 開発者ガイド

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

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

Table of Contents

... x
Amazon Kinesis Data Analytics for SQL アプリケーションのサポート終了 ... 1
サポート終了計画 ... 1
Amazon Managed Service for Apache Flink Studio への移行 ... 2
よくある質問 ... 2
Managed Service for Apache Flink Studio への移行 ... 4
Amazon Kinesis Data Analytics for SQL Applications とは ... 52
Amazon Kinesis Data Analytics が適している用途 ... 52
Amazon Kinesis Data Analytics を初めてお使いになる方向けの情報 ... 53
仕組み ... 54
Input ... 57
ストリーミングソースの設定 ... 58
リファレンスソースの設定 ... 61
JSONPath の操作 ... 64
SQL 入力列へのストリーミングソース要素のマッピング ... 69
ストリーミングデータのスキーマ検出機能の使用 ... 75
静的データに対するスキーマ検出機能の使用 ... 77
Lambda 関数を使用したデータの事前処理 ... 82
スループットの増加に合わせた入力ストリームの並列処理 ... 93
アプリケーションコード ... 98
Output ... 100
を使用した出力の作成 AWS CLI ... 102
出力としての Lambda 関数の使用 ... 103
アプリケーション出力配信モデル ... 112
エラー処理 ... 112
アプリケーション内エラーストリームを使用してエラーをレポートする ... 113
自動スケーリングアプリケーション ... 114
Tagging ... 114
アプリケーション作成時のタグの追加 ... 115
既存のアプリケーションに対するタグの追加または更新 ... 116
アプリケーションのタグの一覧表示 ... 116
アプリケーションからのタグの削除 ... 116
開始方法 ... 118
にサインアップする AWS アカウント ... 118

管理アクセスを持つユーザーを作成する 119
ステップ 1: アカウントを設定する 120
にサインアップする AWS 120
IAM ユーザーの作成 121
次のステップ 122
にサインアップする AWS アカウント 118
管理アクセスを持つユーザーを作成する 119
ステップ 2: をセットアップする AWS CLI 124
次のステップ 125
ステップ 3: スターター分析アプリケーションを作成する 125
ステップ 3.1: アプリケーションの作成 128
ステップ 3.2: 入力の設定 130
ステップ 3.3: リアルタイム分析を追加する (アプリケーションコードの追加) 133
ステップ 3.4: (オプション) アプリケーションコードを更新する 137
ステップ 4 (オプション) コンソールを使用したスキーマと SQL コードの編集 140
スキーマエディタの使用 140
SQL エディタの使用 149
ストリーミング SQL の概念 154
アプリケーション内ストリームとポンプ 154
タイムスタンプと ROWTIME 列 156
ストリーミング分析でのさまざまな時間を理解する 157
連続クエリ 159
ウィンドウクエリ 160
Stagger Windows 161
タンブリングウィンドウ 166
スライディングウィンドウ 168
ストリーム結合 174
例 1: 注文が出されてから 1 分以内に取引があった注文をレポートする 174
Kinesis Data Analytics for SQL の例 176
データ転送 176
Lambda でストリームを処理する 176
文字列値の変換 177
DateTime 値の変換 197
複数のデータ型の変換 202
ウィンドウと集約 210
Stagger Window 210

ROWTIME を使用したタンブリングウィンドウ 214
イベントのタイムスタンプを使用したタンブリングウィンドウ 218
頻出値 (TOP_K_ITEMS_TUMBLING) 222
部分的な結果の集約 226
Joins 229
例: リファレンスデータソースを追加する 229
機械学習 233
異常の検出 234
例: 異常の検出と説明の取得 242
例: ホットスポットの検出 247
アラートとエラー 261
シンプルなアラート 262
調整されたアラート 263
アプリケーション内エラーストリーム 265
ソリューションアクセラレーター 267
AWS アカウント アクティビティに関するリアルタイムのインサイト 267
Kinesis Data Analytics によるリアルタイムの AWS IoT デバイスモニタリング 267
Kinesis Data Analytics を使ったリアルタイムウェブ分析 267
Amazon コネクテッドカーソリューション 267
セキュリティ 268
データ保護 269
データ暗号化 269
Identity and Access Management 270
信頼ポリシー 270
アクセス許可ポリシー 271
サービス間の混乱した代理の防止 274
認証とアクセスコントロール 276
アクセスコントロール 276
アイデンティティを使用した認証 276
アクセス管理の概要 280
アイデンティティベースのポリシー (IAM ポリシー) を使用する 285
API の権限リファレンス 293
モニタリング 294
コンプライアンス検証 294
耐障害性 295
災害対策 296

インフラストラクチャセキュリティ 296
セキュリティのベストプラクティス 296
IAM ロールを使用して他の Amazon サービスにアクセスする 296
依存リソースでのサーバー側の暗号化の実装 297
CloudTrail を使用して API コールをモニタリングする 297
モニタリング 298
モニタリングツール 299
自動化ツール 299
手動ツール 300
Amazon CloudWatch によるモニターリング 300
メトリクスとディメンション 301
のメトリクスおよびディメンションの表示 303
アラーム 304
ログ 305
の使用 AWS CloudTrail 312
CloudTrail での情報 312
ログファイルエントリの概要 313
制限 316
サービス終了日 316
制限 316
ベストプラクティス 320
アプリケーションを管理する 320
アプリケーションのスケーリング 321
アプリケーションのモニタリング 322
入力スキーマの定義 323
出力への接続 324
アプリケーションコードの作成 324
アプリケーションのテスト 325
テストアプリケーションのセットアップ 325
スキーマ変更のテスト 326
コード変更のテスト 326
トラブルシューティング 328
停止したアプリケーション 328
SQL コードを実行できません 329
スキーマを検出または発見できない 329
リファレンスデータが古い 330

アプリケーションが送信先に書き込まれない 330
モニタリングする重要なアプリケーションの状態パラメータ 331
アプリケーションを実行するときの無効なコードエラー 331
アプリケーションによってエラーがエラーストリームに書き込まれている 332
スループット不足または高い MillisBehindLatest 332
SQL リファレンス 334
API リファレンス 335
アクション 335
AddApplicationCloudWatchLoggingOption 337
AddApplicationInput 340
AddApplicationInputProcessingConfiguration 344
AddApplicationOutput 348
AddApplicationReferenceDataSource 352
CreateApplication 356
DeleteApplication 364
DeleteApplicationCloudWatchLoggingOption 367
DeleteApplicationInputProcessingConfiguration 370
DeleteApplicationOutput 373
DeleteApplicationReferenceDataSource 376
DescribeApplication 379
DiscoverInputSchema 384
ListApplications 390
ListTagsForResource 393
StartApplication 396
StopApplication 399
TagResource 402
UntagResource 405
UpdateApplication 408
データ型 413
ApplicationDetail 416
ApplicationSummary 420
ApplicationUpdate 422
CloudWatchLoggingOption 424
CloudWatchLoggingOptionDescription 426
CloudWatchLoggingOptionUpdate 428
CSVMappingParameters 430

DestinationSchema 431
Input 432
InputConfiguration 435
InputDescription 436
InputLambdaProcessor 439
InputLambdaProcessorDescription 441
InputLambdaProcessorUpdate 442
InputParallelism 444
InputParallelismUpdate 445
InputProcessingConfiguration 446
InputProcessingConfigurationDescription 447
InputProcessingConfigurationUpdate 448
InputSchemaUpdate 449
InputStartingPositionConfiguration 451
InputUpdate 452
JSONMappingParameters 454
KinesisFirehoseInput 455
KinesisFirehoseInputDescription 457
KinesisFirehoseInputUpdate 458
KinesisFirehoseOutput 460
KinesisFirehoseOutputDescription 462
KinesisFirehoseOutputUpdate 463
KinesisStreamsInput 464
KinesisStreamsInputDescription 466
KinesisStreamsInputUpdate 467
KinesisStreamsOutput 468
KinesisStreamsOutputDescription 470
KinesisStreamsOutputUpdate 471
LambdaOutput 472
LambdaOutputDescription 474
LambdaOutputUpdate 475
MappingParameters 477
Output 478
OutputDescription 480
OutputUpdate 482
RecordColumn 484

RecordFormat 486
ReferenceDataSource 487
ReferenceDataSourceDescription 489
ReferenceDataSourceUpdate 491
S3Configuration 493
S3ReferenceDataSource 495
S3ReferenceDataSourceDescription 497
S3ReferenceDataSourceUpdate 499
SourceSchema 501
Tag 503
ドキュメント履歴 504
AWS 用語集 509

慎重な検討の結果、Amazon Kinesis Data Analytics for SQL アプリケーションのサポートは終了することになりました。サポート終了は次の 2 段階で行われます。

1. 2025 年 10 月 15 日以降、新しい Kinesis Data Analytics for SQL アプリケーションを作成することはできなくなります。

2. 2026 年 1 月 27 日以降、アプリケーションは削除されます。Amazon Kinesis Data Analytics for SQL アプリケーションを起動することも操作することもできなくなります。これ以降、Amazon Kinesis Data Analytics for SQL のサポートは終了します。詳細については、「Amazon Kinesis Data Analytics for SQL アプリケーションのサポート終了」を参照してください。

翻訳は機械翻訳により提供されています。提供された翻訳内容と英語版の間で齟齬、不一致または矛盾がある場合、英語版が優先します。

Amazon Kinesis Data Analytics for SQL アプリケーションのサポート終了

サポート終了計画

慎重な検討の結果、Amazon Kinesis Data Analytics for SQL アプリケーションのサポートは終了することになりました。お客様が計画的に Amazon Kinesis Data Analytics for SQL アプリケーションから移行できるように、完全なサポート終了までに 15 か月間の猶予を設け、その間に段階的に終了していく予定です。重要な日付となるのは、2025 年 10 月 15 日と 2026 年 1 月 27 日の 2 つです。

1. 2025 年 10 月 15 日には、アプリケーションが停止され、`READY` 状態に設定されます。この時点ではアプリケーションを再び起動でき、サービスの制限に従って、引き続き通常どおりにアプリケーションを使用できます。
2. 2025 年 10 月 15 日以降、新しい Amazon Kinesis Data Analytics for SQL アプリケーションを作成することはできなくなります。既存のアプリケーションは、サービスの制限に従って通常どおりに実行できます。
3. 2026 年 1 月 27 日以降、アプリケーションは削除されます。Amazon Kinesis Data Analytics for SQL アプリケーションを起動することも操作することもできなくなります。これ以降、Amazon Kinesis Data Analytics for SQL アプリケーションのサポートは終了します。

2025 年 10 月 15 日より前に、アプリケーションを Amazon Managed Service for Apache Flink または Amazon Managed Service for Apache Flink Studio に移行することをお勧めします。移行に役立つリソースについては、「Managed Service for Apache Flink Studio への移行例」を参照してください。Amazon Managed Service for Apache Flink または Amazon Managed Service for Apache Flink Studio の詳細については、「Amazon Managed Service for Apache Flink デベロッパーガイド」を参照してください。

> **Note**
>
> すべての Amazon Kinesis Data Analytics for SQL アプリケーションの一覧を表示するには、`ListApplications` API を呼び出します。詳細については、「ListApplications」を参照してください。

Amazon Managed Service for Apache Flink Studio への移行

アプリケーションの移行方法の詳細と、コードやアーキテクチャの例については、「Managed Service for Apache Flink Studio への移行例」を参照してください。

よくある質問

Amazon Kinesis Data Analytics for SQL アプリケーションのサポートを終了するのはなぜですか?

リアルタイムのデータストリーム処理ワークロードには、Amazon Managed Service for Apache Flink サービスの方がお客様に選ばれていることがわかりました。Amazon Managed Service for Apache Flink は、データストリームを処理するためのオープンソースエンジンである Apache Flink を使用したサーバーレスのリアルタイムストリーム処理サービスで、低レイテンシー、高スケーラビリティ、高可用性を兼ね備えています。Amazon Managed Service for Apache Flink は、ネイティブスケーリング、1 回限りの処理セマンティクス、多言語サポート (SQL を含む)、40 を超えるソースコネクタと宛先コネクタ、永続的なアプリケーション状態などの機能を提供します。これらの機能は、エンドツーエンドのストリーミングパイプラインを構築し、データの正確性と適時性を確保するために役立ちます。

利用者が現在できることは何ですか?

既存の Amazon Kinesis Data Analytics for SQL アプリケーションを Amazon Managed Service for Apache Flink Studio または Amazon Managed Service for Apache Flink にアップグレードすることをお勧めします。Amazon Managed Service for Apache Flink Studio では、インタラクティブノートブックを使用して、SQL、Python、または Scala でクエリを作成します。Amazon Kinesis Data Analytics for SQL の長時間実行アプリケーションには、Amazon Managed Service for Apache Flink をお勧めします。このサービスでは、Apache Flink の API、コネクタ、その他のあらゆる機能を使用して、Java、Python、Scala、埋め込み SQL でアプリケーションを作成できます。

Amazon Kinesis Data Analytics for SQL アプリケーションの利用者数はどれくらいですか?

顧客情報の詳細を公開することはできません。

Amazon Kinesis Data Analytics for SQL アプリケーションから Amazon Managed Service for Apache Flink サービスにアップグレードするにはどうすればよいですか?

Amazon Managed Service for Apache Flink または Amazon Managed Service for Apache Flink Studio にアップグレードするには、アプリケーションを作り直す必要があります。この作業を支援するために、一般的な SQL クエリと、それらを Amazon Managed Service for Apache Flink Studio

で書き換える方法をまとめました。「Managed Service for Apache Flink Studio での Kinesis Data Analytics for SQL クエリの複製」を参照してください。また、Amazon Managed Service for Apache Flink で長時間実行アプリケーションを構築している場合や機械学習を使用している場合に適用できる、一般的なパターンアーキテクチャも提供しています。「ソースとしての Kinesis Data Firehose を Kinesis Data Streams に置き換える 」を参照してください。

Amazon Managed Service for Apache Flink の詳細については、「Amazon Managed Service for Apache Flink とは」を参照してください。

移行ガイドについては、「Managed Service for Apache Flink Studio への移行例」を参照してください。

Amazon Managed Service for Apache Flink は、既存の Amazon Kinesis Data Analytics for SQL アプリケーションの機能をサポートしていますか?

Amazon Managed Service for Apache Flink は、コネクタやウィンドウ処理など、Amazon Kinesis Data Analytics for SQL アプリケーションで利用できる概念の多くと、ネイティブスケーリング、1 回限りの処理セマンティクス、多言語サポート (SQL を含む)、40 を超えるソースコネクタと宛先コネクタ、永続的なアプリケーション状態など、Amazon Kinesis Data Analytics for SQL アプリケーションで利用できなかった機能をサポートしています。

Managed Service for Apache Flink Studio への移行例

慎重な検討の結果、Amazon Kinesis Data Analytics for SQL アプリケーションのサポートは終了することになりました。お客様が計画的に Amazon Kinesis Data Analytics for SQL アプリケーションから移行できるように、完全なサポート終了までに 15 か月間の猶予を設け、その間に段階的に終了していく予定です。重要な日付となるのは、2025 年 10 月 15 日と 2026 年 1 月 27 日の 2 つです。

1. 2025 年 10 月 15 日以降、新しい Amazon Kinesis Data Analytics for SQL アプリケーションを作成することはできなくなります。
2. 2026 年 1 月 27 日以降、アプリケーションは削除されます。Amazon Kinesis Data Analytics for SQL アプリケーションを起動することも操作することもできなくなります。これ以降、Amazon Kinesis Data Analytics for SQL アプリケーションのサポートは終了します。詳細については、「Amazon Kinesis Data Analytics for SQL アプリケーションのサポート終了」を参照してください。

Amazon Managed Service for Apache Flink を使用することをお勧めします。このサービスは、使いやすさと高度な分析機能を兼ね備え、ストリーム処理アプリケーションを数分で構築できます。

このセクションでは、Amazon Kinesis Data Analytics for SQL アプリケーションのワークロードを Managed Service for Apache Flink に移行するために役立つコードとアーキテクチャの例を示します。

詳細については、AWS ブログの記事「Migrate from Amazon Kinesis Data Analytics for SQL Applications to Managed Service for Apache Flink Studio」も参照してください。

Managed Service for Apache Flink Studio での Kinesis Data Analytics for SQL クエリの複製

Managed Service for Apache Flink Studio または Managed Service for Apache Flink にワークロードを移行するために、このセクションでは一般的なユースケースで使用できるクエリ変換について説明します。

これらの例を参照する前に、「Managed Service for Apache Flink で Studio ノートブックを使用する」を確認することをお勧めします。

Managed Service for Apache Flink Studio での Kinesis Data Analytics for SQL クエリの再作成

ここでは、一般的な SQL ベースの Kinesis Data Analytics アプリケーションクエリを Managed Service for Apache Flink Studio に変換する方法を示します。

マルチステップアプリケーション

SQL-based Kinesis Data Analytics application

```
CREATE
OR REPLACE STREAM "IN_APP_STREAM_001" (
   ingest_time TIMESTAMP,
   ticker_symbol VARCHAR(4),
   sector VARCHAR(16), price REAL, change REAL);
CREATE
OR REPLACE PUMP "STREAM_PUMP_001" AS
INSERT INTO
   "IN_APP_STREAM_001"
   SELECT
      STREAM APPROXIMATE_ARRIVAL_TIME,
      ticker_symbol,
      sector,
      price,
      change FROM "SOURCE_SQL_STREAM_001";
-- Second in-app stream and pump
CREATE
OR REPLACE STREAM "IN_APP_STREAM_02" (ingest_time TIMESTAMP,
   ticker_symbol VARCHAR(4),
   sector VARCHAR(16),
   price REAL,
   change REAL);
CREATE
OR REPLACE PUMP "STREAM_PUMP_02" AS
INSERT INTO
   "IN_APP_STREAM_02"
   SELECT
      STREAM ingest_time,
      ticker_symbol,
      sector,
      price,
      change FROM "IN_APP_STREAM_001";
-- Destination in-app stream and third pump
```

```
CREATE
OR REPLACE STREAM "DESTINATION_SQL_STREAM" (ingest_time TIMESTAMP,
   ticker_symbol VARCHAR(4),
   sector VARCHAR(16),
   price REAL,
   change REAL);
CREATE
OR REPLACE PUMP "STREAM_PUMP_03" AS
INSERT INTO
   "DESTINATION_SQL_STREAM"
   SELECT
      STREAM ingest_time,
      ticker_symbol,
      sector,
      price,
      change FROM "IN_APP_STREAM_02";
```

Managed Service for Apache Flink Studio

```
Query 1 - % flink.ssql DROP TABLE IF EXISTS SOURCE_SQL_STREAM_001;

CREATE TABLE SOURCE_SQL_STREAM_001 (TICKER_SYMBOL VARCHAR(4),
   SECTOR VARCHAR(16),
   PRICE DOUBLE,
   CHANGE DOUBLE,
   APPROXIMATE_ARRIVAL_TIME TIMESTAMP(3) METADATA

FROM
   'timestamp' VIRTUAL,
   WATERMARK FOR APPROXIMATE_ARRIVAL_TIME AS APPROXIMATE_ARRIVAL_TIME - INTERVAL '1'
 SECOND )
   PARTITIONED BY (TICKER_SYMBOL) WITH (
      'connector' = 'kinesis',
      'stream' = 'kinesis-analytics-demo-stream',
      'aws.region' = 'us-east-1',
      'scan.stream.initpos' = 'LATEST',
      'format' = 'json',
      'json.timestamp-format.standard' = 'ISO-8601');
DROP TABLE IF EXISTS IN_APP_STREAM_001;

CREATE TABLE IN_APP_STREAM_001 (
   INGEST_TIME TIMESTAMP,
   TICKER_SYMBOL VARCHAR(4),
```

```
   SECTOR VARCHAR(16),
   PRICE DOUBLE,
   CHANGE DOUBLE )
PARTITIONED BY (TICKER_SYMBOL) WITH (
      'connector' = 'kinesis',
      'stream' = 'IN_APP_STREAM_001',
      'aws.region' = 'us-east-1',
      'scan.stream.initpos' = 'LATEST',
      'format' = 'json',
      'json.timestamp-format.standard' = 'ISO-8601');

DROP TABLE IF EXISTS IN_APP_STREAM_02;

CREATE TABLE IN_APP_STREAM_02 (
   INGEST_TIME TIMESTAMP,
   TICKER_SYMBOL VARCHAR(4),
   SECTOR VARCHAR(16),
   PRICE DOUBLE,
   CHANGE DOUBLE )
PARTITIONED BY (TICKER_SYMBOL) WITH (
   'connector' = 'kinesis',
   'stream' = 'IN_APP_STREAM_02',
   'aws.region' = 'us-east-1',
   'scan.stream.initpos' = 'LATEST',
   'format' = 'json',
   'json.timestamp-format.standard' = 'ISO-8601');

DROP TABLE IF EXISTS DESTINATION_SQL_STREAM;

CREATE TABLE DESTINATION_SQL_STREAM (
   INGEST_TIME TIMESTAMP, TICKER_SYMBOL VARCHAR(4), SECTOR VARCHAR(16),
   PRICE DOUBLE, CHANGE DOUBLE )
PARTITIONED BY (TICKER_SYMBOL) WITH (
   'connector' = 'kinesis',
   'stream' = 'DESTINATION_SQL_STREAM',
   'aws.region' = 'us-east-1',
   'scan.stream.initpos' = 'LATEST',
   'format' = 'json',
   'json.timestamp-format.standard' = 'ISO-8601');


Query 2 - % flink.ssql(type =
update
)
```

```
   INSERT INTO
      IN_APP_STREAM_001
      SELECT
         APPROXIMATE_ARRIVAL_TIME AS INGEST_TIME,
         TICKER_SYMBOL,
         SECTOR,
         PRICE,
         CHANGE
      FROM
         SOURCE_SQL_STREAM_001;


Query 3 - % flink.ssql(type =
update
)
   INSERT INTO
      IN_APP_STREAM_02
      SELECT
         INGEST_TIME,
         TICKER_SYMBOL,
         SECTOR,
         PRICE,
         CHANGE
      FROM
         IN_APP_STREAM_001;


Query 4 - % flink.ssql(type =
update
)
   INSERT INTO
      DESTINATION_SQL_STREAM
      SELECT
         INGEST_TIME,
         TICKER_SYMBOL,
         SECTOR,
         PRICE,
         CHANGE
      FROM
         IN_APP_STREAM_02;
```

DateTime 値の変換

SQL-based Kinesis Data Analytics application

```
CREATE
OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
   TICKER VARCHAR(4),
   event_time TIMESTAMP,
   five_minutes_before TIMESTAMP,
   event_unix_timestamp BIGINT,
   event_timestamp_as_char VARCHAR(50),
   event_second INTEGER);

CREATE
OR REPLACE PUMP "STREAM_PUMP" AS
INSERT INTO
   "DESTINATION_SQL_STREAM"
   SELECT
      STREAM TICKER,
      EVENT_TIME,
      EVENT_TIME - INTERVAL '5' MINUTE,
      UNIX_TIMESTAMP(EVENT_TIME),
      TIMESTAMP_TO_CHAR('yyyy-MM-dd hh:mm:ss', EVENT_TIME),
      EXTRACT(SECOND
   FROM
      EVENT_TIME)
   FROM
      "SOURCE_SQL_STREAM_001"
```

Managed Service for Apache Flink Studio

```
Query 1 - % flink.ssql(type =
update
) CREATE TABLE DESTINATION_SQL_STREAM (
   TICKER VARCHAR(4),
   EVENT_TIME TIMESTAMP(3),
   FIVE_MINUTES_BEFORE TIMESTAMP(3),
   EVENT_UNIX_TIMESTAMP INT,
   EVENT_TIMESTAMP_AS_CHAR VARCHAR(50),
   EVENT_SECOND INT)

PARTITIONED BY (TICKER) WITH (
   'connector' = 'kinesis', 'stream' = 'kinesis-analytics-demo-stream',
```

```
   'aws.region' = 'us-east-1',
   'scan.stream.initpos' = 'LATEST',
   'format' = 'json',
   'json.timestamp-format.standard' = 'ISO-8601')

Query 2 - % flink.ssql(type =
   update
)
      SELECT
         TICKER,
         EVENT_TIME,
         EVENT_TIME - INTERVAL '5' MINUTE AS FIVE_MINUTES_BEFORE,
         UNIX_TIMESTAMP() AS EVENT_UNIX_TIMESTAMP,
         DATE_FORMAT(EVENT_TIME, 'yyyy-MM-dd hh:mm:ss') AS EVENT_TIMESTAMP_AS_CHAR,
         EXTRACT(SECOND
      FROM
         EVENT_TIME) AS EVENT_SECOND
      FROM
         DESTINATION_SQL_STREAM;
```

シンプルなアラート

SQL-based Kinesis Data Analytics application

```
CREATE
OR REPLACE STREAM "DESTINATION_SQL_STREAM"(
   ticker_symbol VARCHAR(4),
   sector VARCHAR(12),
   change DOUBLE,
   price DOUBLE);

CREATE
OR REPLACE PUMP "STREAM_PUMP" AS INSERT INTO "DESTINATION_SQL_STREAM"
SELECT
   STREAM ticker_symbol,
   sector,
   change,
   price
FROM
   "SOURCE_SQL_STREAM_001"
WHERE
   (
```

```
      ABS(Change / (Price - Change)) * 100
   )
   > 1
```

Managed Service for Apache Flink Studio

```
Query 1 - % flink.ssql(type =
update
) DROP TABLE IF EXISTS DESTINATION_SQL_STREAM;

CREATE TABLE DESTINATION_SQL_STREAM (
   TICKER_SYMBOL VARCHAR(4),
   SECTOR VARCHAR(4),
   CHANGE DOUBLE,
   PRICE DOUBLE )
PARTITIONED BY (TICKER_SYMBOL) WITH (
   'connector' = 'kinesis',
   'stream' = 'kinesis-analytics-demo-stream',
   'aws.region' = 'us-east-1',
   'scan.stream.initpos' = 'LATEST',
   'format' = 'json',
   'json.timestamp-format.standard' = 'ISO-8601');

Query 2 - % flink.ssql(type =
update
)
   SELECT
      TICKER_SYMBOL,
      SECTOR,
      CHANGE,
      PRICE
   FROM
      DESTINATION_SQL_STREAM
   WHERE
      (
         ABS(CHANGE / (PRICE - CHANGE)) * 100
      )
      > 1;
```

調整されたアラート

SQL-based Kinesis Data Analytics application

```
CREATE
OR REPLACE STREAM "CHANGE_STREAM"(
   ticker_symbol VARCHAR(4),
   sector VARCHAR(12),
   change DOUBLE,
   price DOUBLE);

CREATE
OR REPLACE PUMP "change_pump" AS INSERT INTO "CHANGE_STREAM"
SELECT
   STREAM ticker_symbol,
   sector,
   change,
   price
FROM "SOURCE_SQL_STREAM_001"
WHERE
   (
      ABS(Change / (Price - Change)) * 100
   )
   > 1;
-- ** Trigger Count and Limit **
-- Counts "triggers" or those values that evaluated true against the previous where
 clause
-- Then provides its own limit on the number of triggers per hour per ticker symbol
 to what is specified in the WHERE clause

CREATE
OR REPLACE STREAM TRIGGER_COUNT_STREAM (
   ticker_symbol VARCHAR(4),
   change REAL,
   trigger_count INTEGER);

CREATE
OR REPLACE PUMP trigger_count_pump AS
INSERT INTO
   TRIGGER_COUNT_STREAMSELECT STREAM ticker_symbol,
   change,
   trigger_count
FROM
   (
```

```
      SELECT
         STREAM ticker_symbol,
         change,
         COUNT(*) OVER W1 as trigger_countFROM "CHANGE_STREAM" --window to perform
 aggregations over last minute to keep track of triggers
         WINDOW W1 AS
         (
            PARTITION BY ticker_symbol RANGE INTERVAL '1' MINUTE PRECEDING
         )
   )
WHERE
   trigger_count >= 1;
```

Managed Service for Apache Flink Studio

```
Query 1 - % flink.ssql(type =
update
) DROP TABLE IF EXISTS DESTINATION_SQL_STREAM;

CREATE TABLE DESTINATION_SQL_STREAM (
   TICKER_SYMBOL VARCHAR(4),
   SECTOR VARCHAR(4),
   CHANGE DOUBLE, PRICE DOUBLE,
   EVENT_TIME AS PROCTIME())
PARTITIONED BY (TICKER_SYMBOL)
WITH (
   'connector' = 'kinesis',
   'stream' = 'kinesis-analytics-demo-stream',
   'aws.region' = 'us-east-1',
   'scan.stream.initpos' = 'LATEST',
   'format' = 'json',
   'json.timestamp-format.standard' = 'ISO-8601');
DROP TABLE IF EXISTS TRIGGER_COUNT_STREAM;
CREATE TABLE TRIGGER_COUNT_STREAM (
   TICKER_SYMBOL VARCHAR(4),
   CHANGE DOUBLE,
   TRIGGER_COUNT INT)
PARTITIONED BY (TICKER_SYMBOL);

Query 2 - % flink.ssql(type =
update
)
   SELECT
```

```
      TICKER_SYMBOL,
      SECTOR,
      CHANGE,
      PRICE
   FROM
      DESTINATION_SQL_STREAM
   WHERE
      (
         ABS(CHANGE / (PRICE - CHANGE)) * 100
      )
      > 1;

Query 3 - % flink.ssql(type =
update
)
   SELECT *
   FROM(
         SELECT
            TICKER_SYMBOL,
            CHANGE,
            COUNT(*) AS TRIGGER_COUNT
         FROM
            DESTINATION_SQL_STREAM
         GROUP BY
            TUMBLE(EVENT_TIME, INTERVAL '1' MINUTE),
            TICKER_SYMBOL,
            CHANGE
      )
   WHERE
      TRIGGER_COUNT > 1;
```

例: クエリから部分的な結果を集約する

SQL-based Kinesis Data Analytics application

```
CREATE
OR REPLACE STREAM "CALC_COUNT_SQL_STREAM"(
   TICKER VARCHAR(4),
   TRADETIME TIMESTAMP,
   TICKERCOUNT DOUBLE);

CREATE
```

```
OR REPLACE STREAM "DESTINATION_SQL_STREAM"(
   TICKER VARCHAR(4),
   TRADETIME TIMESTAMP,
   TICKERCOUNT DOUBLE);

CREATE PUMP "CALC_COUNT_SQL_PUMP_001" AS
INSERT INTO
   "CALC_COUNT_SQL_STREAM"(
   "TICKER",
   "TRADETIME",
   "TICKERCOUNT")
   SELECT
      STREAM "TICKER_SYMBOL",
      STEP("SOURCE_SQL_STREAM_001",
      "ROWTIME" BY INTERVAL '1' MINUTE) as "TradeTime",
      COUNT(*) AS "TickerCount "
   FROM
      "SOURCE_SQL_STREAM_001"
   GROUP BY
      STEP("SOURCE_SQL_STREAM_001". ROWTIME BY INTERVAL '1' MINUTE),
      STEP("SOURCE_SQL_STREAM_001"." APPROXIMATE_ARRIVAL_TIME" BY INTERVAL '1'
 MINUTE),
      TICKER_SYMBOL;
CREATE PUMP "AGGREGATED_SQL_PUMP" AS
INSERT INTO
   "DESTINATION_SQL_STREAM" (
   "TICKER",
   "TRADETIME",
   "TICKERCOUNT")
   SELECT
      STREAM "TICKER",
      "TRADETIME",
      SUM("TICKERCOUNT") OVER W1 AS "TICKERCOUNT"
   FROM
      "CALC_COUNT_SQL_STREAM" WINDOW W1 AS
      (
         PARTITION BY "TRADETIME" RANGE INTERVAL '10' MINUTE PRECEDING
      )
;
```

Managed Service for Apache Flink Studio

```
Query 1 - % flink.ssql(type =
```

```
update
) DROP TABLE IF EXISTS SOURCE_SQL_STREAM_001;
CREATE TABLE SOURCE_SQL_STREAM_001 (
   TICKER_SYMBOL VARCHAR(4),
   TRADETIME AS PROCTIME(),
   APPROXIMATE_ARRIVAL_TIME TIMESTAMP(3) METADATA
FROM
   'timestamp' VIRTUAL,
   WATERMARK FOR APPROXIMATE_ARRIVAL_TIME AS APPROXIMATE_ARRIVAL_TIME - INTERVAL '1'
 SECOND)
PARTITIONED BY (TICKER_SYMBOL) WITH (
   'connector' = 'kinesis',
   'stream' = 'kinesis-analytics-demo-stream',
   'aws.region' = 'us-east-1',
   'scan.stream.initpos' = 'LATEST',
   'format' = 'json',
   'json.timestamp-format.standard' = 'ISO-8601');
DROP TABLE IF EXISTS CALC_COUNT_SQL_STREAM;
CREATE TABLE CALC_COUNT_SQL_STREAM (
   TICKER VARCHAR(4),
   TRADETIME TIMESTAMP(3),
   WATERMARK FOR TRADETIME AS TRADETIME - INTERVAL '1' SECOND,
   TICKERCOUNT BIGINT NOT NULL ) PARTITIONED BY (TICKER) WITH (
      'connector' = 'kinesis',
      'stream' = 'CALC_COUNT_SQL_STREAM',
      'aws.region' = 'us-east-1',
      'scan.stream.initpos' = 'LATEST',
      'format' = 'csv');
DROP TABLE IF EXISTS DESTINATION_SQL_STREAM;
CREATE TABLE DESTINATION_SQL_STREAM (
   TICKER VARCHAR(4),
   TRADETIME TIMESTAMP(3),
   WATERMARK FOR TRADETIME AS TRADETIME - INTERVAL '1' SECOND,
   TICKERCOUNT BIGINT NOT NULL )
   PARTITIONED BY (TICKER) WITH ('connector' = 'kinesis',
      'stream' = 'DESTINATION_SQL_STREAM',
      'aws.region' = 'us-east-1',
      'scan.stream.initpos' = 'LATEST',
      'format' = 'csv');

Query 2 - % flink.ssql(type =
update
)
   INSERT INTO
```

```
      CALC_COUNT_SQL_STREAM
      SELECT
         TICKER,
         TO_TIMESTAMP(TRADETIME, 'yyyy-MM-dd HH:mm:ss') AS TRADETIME,
         TICKERCOUNT
      FROM
         (
            SELECT
               TICKER_SYMBOL AS TICKER,
               DATE_FORMAT(TRADETIME, 'yyyy-MM-dd HH:mm:00') AS TRADETIME,
               COUNT(*) AS TICKERCOUNT
            FROM
               SOURCE_SQL_STREAM_001
            GROUP BY
               TUMBLE(TRADETIME, INTERVAL '1' MINUTE),
               DATE_FORMAT(TRADETIME, 'yyyy-MM-dd HH:mm:00'),
               DATE_FORMAT(APPROXIMATE_ARRIVAL_TIME, 'yyyy-MM-dd HH:mm:00'),
               TICKER_SYMBOL
         )
;

Query 3 - % flink.ssql(type =
update
)
   SELECT
      *
   FROM
      CALC_COUNT_SQL_STREAM;

Query 4 - % flink.ssql(type =
update
)
   INSERT INTO
      DESTINATION_SQL_STREAM
      SELECT
         TICKER,
         TRADETIME,
         SUM(TICKERCOUNT) OVER W1 AS TICKERCOUNT
      FROM
         CALC_COUNT_SQL_STREAM WINDOW W1 AS
         (
            PARTITION BY TICKER
         ORDER BY
            TRADETIME RANGE INTERVAL '10' MINUTE PRECEDING
```

```
         )
;

Query 5 - % flink.ssql(type =
update
)
   SELECT
      *
   FROM
      DESTINATION_SQL_STREAM;
```

文字列値の変換

SQL-based Kinesis Data Analytics application

```
CREATE
OR REPLACE STREAM for cleaned up referrerCREATE
OR REPLACE STREAM "DESTINATION_SQL_STREAM" ( "ingest_time" TIMESTAMP, "referrer"
 VARCHAR(32));
CREATE
OR REPLACE PUMP "myPUMP" AS INSERT INTO "DESTINATION_SQL_STREAM"
SELECT
   STREAM "APPROXIMATE_ARRIVAL_TIME",
   SUBSTRING("referrer", 12,
   (
      POSITION('.com' IN "referrer") - POSITION('www.' IN "referrer") - 4
   )
)
FROM
   "SOURCE_SQL_STREAM_001";
```

Managed Service for Apache Flink Studio

```
Query 1 - % flink.ssql(type =
update
) CREATE TABLE DESTINATION_SQL_STREAM (
   referrer VARCHAR(32),
   ingest_time AS PROCTIME() ) PARTITIONED BY (referrer)
WITH (
   'connector' = 'kinesis',
   'stream' = 'kinesis-analytics-demo-stream',
   'aws.region' = 'us-east-1',
```

```
   'scan.stream.initpos' = 'LATEST',
   'format' = 'json',
   'json.timestamp-format.standard' = 'ISO-8601')

Query 2 - % flink.ssql(type =
   update
)
      SELECT
         ingest_time,
         substring(referrer, 12, 6) as referrer
      FROM
         DESTINATION_SQL_STREAM;
```

正規表現を使用した部分文字列の置き換え

SQL-based Kinesis Data Analytics application

```
CREATE
OR REPLACE STREAM for cleaned up referrerCREATE
OR REPLACE STREAM "DESTINATION_SQL_STREAM" ( "ingest_time" TIMESTAMP, "referrer"
 VARCHAR(32));
CREATE
OR REPLACE PUMP "myPUMP" AS INSERT INTO "DESTINATION_SQL_STREAM"
SELECT
   STREAM "APPROXIMATE_ARRIVAL_TIME",
   REGEX_REPLACE("REFERRER", 'http://', 'https://', 1, 0)
FROM
   "SOURCE_SQL_STREAM_001";
```

Managed Service for Apache Flink Studio

```
Query 1 - % flink.ssql(type =
update
) CREATE TABLE DESTINATION_SQL_STREAM (
   referrer VARCHAR(32),
   ingest_time AS PROCTIME())
PARTITIONED BY (referrer) WITH (
   'connector' = 'kinesis',
   'stream' = 'kinesis-analytics-demo-stream',
   'aws.region' = 'us-east-1',
   'scan.stream.initpos' = 'LATEST',
   'format' = 'json',
```

```
   'json.timestamp-format.standard' = 'ISO-8601')

Query 2 - % flink.ssql(type =
   update
)
      SELECT
         ingest_time,
         REGEXP_REPLACE(referrer, 'http', 'https') as referrer
      FROM
         DESTINATION_SQL_STREAM;
```

正規表現ログ解析

SQL-based Kinesis Data Analytics application

```
CREATE
OR REPLACE STREAM "DESTINATION_SQL_STREAM"(
   sector VARCHAR(24),
   match1 VARCHAR(24),
   match2 VARCHAR(24));
CREATE
OR REPLACE PUMP "STREAM_PUMP" AS
INSERT INTO
   "DESTINATION_SQL_STREAM"
   SELECT
      STREAM T.SECTOR,
      T.REC.COLUMN1,
      T.REC.COLUMN2
   FROM
      (
         SELECT
            STREAM SECTOR,
            REGEX_LOG_PARSE(SECTOR, '.*([E].).*([R].*)') AS REC
         FROM
            SOURCE_SQL_STREAM_001
      )
      AS T;
```

Managed Service for Apache Flink Studio

```
Query 1 - % flink.ssql(type =
update
```

```
) CREATE TABLE DESTINATION_SQL_STREAM (
   CHANGE DOUBLE, PRICE DOUBLE,
   TICKER_SYMBOL VARCHAR(4),
   SECTOR VARCHAR(16))
PARTITIONED BY (SECTOR) WITH (
   'connector' = 'kinesis',
   'stream' = 'kinesis-analytics-demo-stream',
   'aws.region' = 'us-east-1',
   'scan.stream.initpos' = 'LATEST',
   'format' = 'json',
   'json.timestamp-format.standard' = 'ISO-8601')

Query 2 - % flink.ssql(type =
   update
)
SELECT
   *
FROM
   (
      SELECT
         SECTOR,
         REGEXP_EXTRACT(SECTOR, '.([E].).([R].)', 1) AS MATCH1,
         REGEXP_EXTRACT(SECTOR, '.([E].).([R].)', 2) AS MATCH2
      FROM
         DESTINATION_SQL_STREAM
   )
WHERE
   MATCH1 IS NOT NULL
   AND MATCH2 IS NOT NULL;
```

DateTime 値の変換

SQL-based Kinesis Data Analytics application

```
CREATE
OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
   TICKER VARCHAR(4),
   event_time TIMESTAMP,
   five_minutes_before TIMESTAMP,
   event_unix_timestamp BIGINT,
   event_timestamp_as_char VARCHAR(50),
   event_second INTEGER);
```

```
CREATE
OR REPLACE PUMP "STREAM_PUMP" AS
INSERT INTO
   "DESTINATION_SQL_STREAM"
   SELECT
      STREAM TICKER,
      EVENT_TIME,
      EVENT_TIME - INTERVAL '5' MINUTE,
      UNIX_TIMESTAMP(EVENT_TIME),
      TIMESTAMP_TO_CHAR('yyyy-MM-dd hh:mm:ss', EVENT_TIME),
      EXTRACT(SECOND
   FROM
      EVENT_TIME)
   FROM
      "SOURCE_SQL_STREAM_001"
```

Managed Service for Apache Flink Studio

```
Query 1 - % flink.ssql(type =
update
) CREATE TABLE DESTINATION_SQL_STREAM (
   TICKER VARCHAR(4),
   EVENT_TIME TIMESTAMP(3),
   FIVE_MINUTES_BEFORE TIMESTAMP(3),
   EVENT_UNIX_TIMESTAMP INT,
   EVENT_TIMESTAMP_AS_CHAR VARCHAR(50),
   EVENT_SECOND INT) PARTITIONED BY (TICKER)
WITH (
   'connector' = 'kinesis',
   'stream' = 'kinesis-analytics-demo-stream',
   'aws.region' = 'us-east-1',
   'scan.stream.initpos' = 'LATEST',
   'format' = 'json',
   'json.timestamp-format.standard' = 'ISO-8601')

Query 2 - % flink.ssql(type =
   update
)
      SELECT
         TICKER,
         EVENT_TIME,
         EVENT_TIME - INTERVAL '5' MINUTE AS FIVE_MINUTES_BEFORE,
         UNIX_TIMESTAMP() AS EVENT_UNIX_TIMESTAMP,
```

```
         DATE_FORMAT(EVENT_TIME, 'yyyy-MM-dd hh:mm:ss') AS EVENT_TIMESTAMP_AS_CHAR,
         EXTRACT(SECOND
      FROM
         EVENT_TIME) AS EVENT_SECOND
      FROM
         DESTINATION_SQL_STREAM;
```

ウィンドウと集約

SQL-based Kinesis Data Analytics application

```
CREATE
OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
   event_time TIMESTAMP,
   ticker_symbol VARCHAR(4),
   ticker_count INTEGER);
CREATE
OR REPLACE PUMP "STREAM_PUMP" AS
INSERT INTO
   "DESTINATION_SQL_STREAM"
   SELECT
      STREAM EVENT_TIME,
      TICKER,
      COUNT(TICKER) AS ticker_count
   FROM
      "SOURCE_SQL_STREAM_001" WINDOWED BY STAGGER ( PARTITION BY
         TICKER,
         EVENT_TIME RANGE INTERVAL '1' MINUTE);
```

Managed Service for Apache Flink Studio

```
Query 1 - % flink.ssql(type =
update
) CREATE TABLE DESTINATION_SQL_STREAM (
   EVENT_TIME TIMESTAMP(3),
   WATERMARK FOR EVENT_TIME AS EVENT_TIME - INTERVAL '60' SECOND,
   TICKER VARCHAR(4),
   TICKER_COUNT INT) PARTITIONED BY (TICKER)
WITH (
   'connector' = 'kinesis',
   'stream' = 'kinesis-analytics-demo-stream',
   'aws.region' = 'us-east-1',
```

```
   'scan.stream.initpos' = 'LATEST',
   'format' = 'json'

Query 2 - % flink.ssql(type =
   update
)
      SELECT
         EVENT_TIME,
         TICKER, COUNT(TICKER) AS ticker_count
      FROM
         DESTINATION_SQL_STREAM
      GROUP BY
         TUMBLE(EVENT_TIME,
         INTERVAL '60' second),
         EVENT_TIME, TICKER;
```

ROWTIME を使用したタンブリングウィンドウ

SQL-based Kinesis Data Analytics application

```
CREATE
OR REPLACE STREAM "DESTINATION_SQL_STREAM"(
   TICKER VARCHAR(4),
   MIN_PRICE REAL,
   MAX_PRICE REAL);
CREATE
OR REPLACE PUMP "STREAM_PUMP" AS
INSERT INTO
   "DESTINATION_SQL_STREAM"
   SELECT
      STREAM TICKER,
      MIN(PRICE),
      MAX(PRICE)
   FROM
      "SOURCE_SQL_STREAM_001"
   GROUP BY
      TICKER,
      STEP("SOURCE_SQL_STREAM_001".
            ROWTIME BY INTERVAL '60' SECOND);
```

Managed Service for Apache Flink Studio

```
Query 1 - % flink.ssql(type =
update
) CREATE TABLE DESTINATION_SQL_STREAM (
   ticker VARCHAR(4),
   price DOUBLE,
   event_time VARCHAR(32),
   processing_time AS PROCTIME())
PARTITIONED BY (ticker) WITH (
   'connector' = 'kinesis',
   'stream' = 'kinesis-analytics-demo-stream',
   'aws.region' = 'us-east-1',
   'scan.stream.initpos' = 'LATEST',
   'format' = 'json',
   'json.timestamp-format.standard' = 'ISO-8601')

Query 2 - % flink.ssql(type =
   update
)
      SELECT
         ticker,
         min(price) AS MIN_PRICE,
         max(price) AS MAX_PRICE
      FROM
         DESTINATION_SQL_STREAM
      GROUP BY
         TUMBLE(processing_time, INTERVAL '60' second),
         ticker;
```

最も頻繁に出現する値 (TOP_K_ITEMS_TUMBLING) の取得

SQL-based Kinesis Data Analytics application

```
CREATE
OR REPLACE STREAM "CALC_COUNT_SQL_STREAM"(TICKER VARCHAR(4),
   TRADETIME TIMESTAMP,
   TICKERCOUNT DOUBLE);
CREATE
OR REPLACE STREAM "DESTINATION_SQL_STREAM"(
   TICKER VARCHAR(4),
   TRADETIME TIMESTAMP,
```

```
   TICKERCOUNT DOUBLE);
CREATE PUMP "CALC_COUNT_SQL_PUMP_001" AS INSERT INTO "CALC_COUNT_SQL_STREAM" (
   "TICKER",
   "TRADETIME",
   "TICKERCOUNT")
SELECT
   STREAM"TICKER_SYMBOL",
   STEP("SOURCE_SQL_STREAM_001"."ROWTIME" BY INTERVAL '1' MINUTE) as "TradeTime",
   COUNT(*) AS "TickerCount"
FROM
   "SOURCE_SQL_STREAM_001"
GROUP BY STEP("SOURCE_SQL_STREAM_001".
   ROWTIME BY INTERVAL '1' MINUTE),
   STEP("SOURCE_SQL_STREAM_001".
      "APPROXIMATE_ARRIVAL_TIME" BY INTERVAL '1' MINUTE),
   TICKER_SYMBOL;
CREATE PUMP "AGGREGATED_SQL_PUMP" AS INSERT INTO "DESTINATION_SQL_STREAM" (
   "TICKER",
   "TRADETIME",
   "TICKERCOUNT")
SELECT
   STREAM "TICKER",
   "TRADETIME",
   SUM("TICKERCOUNT") OVER W1 AS "TICKERCOUNT"
FROM
   "CALC_COUNT_SQL_STREAM" WINDOW W1 AS
   (
      PARTITION BY "TRADETIME" RANGE INTERVAL '10' MINUTE PRECEDING
   )
;
```

Managed Service for Apache Flink Studio

```
Query 1 - % flink.ssql(type =
update
) DROP TABLE IF EXISTS DESTINATION_SQL_STREAM;
CREATE TABLE DESTINATION_SQL_STREAM (
   TICKER VARCHAR(4),
   EVENT_TIME TIMESTAMP(3),
   WATERMARK FOR EVENT_TIME AS EVENT_TIME - INTERVAL '1' SECONDS )
PARTITIONED BY (TICKER) WITH (
   'connector' = 'kinesis', 'stream' = 'kinesis-analytics-demo-stream',
   'aws.region' = 'us-east-1',
```

```
   'scan.stream.initpos' = 'LATEST',
   'format' = 'json',
   'json.timestamp-format.standard' = 'ISO-8601');

Query 2 - % flink.ssql(type =
update
)
   SELECT
      *
   FROM
      (
         SELECT
            TICKER,
            COUNT(*) as MOST_FREQUENT_VALUES,
            ROW_NUMBER() OVER (PARTITION BY TICKER
         ORDER BY
            TICKER DESC) AS row_num
         FROM
            DESTINATION_SQL_STREAM
         GROUP BY
            TUMBLE(EVENT_TIME, INTERVAL '1' MINUTE),
            TICKER
      )
   WHERE
      row_num <= 5;
```

おおよそのトップ K 項目

SQL-based Kinesis Data Analytics application

```
CREATE
OR REPLACE STREAM "DESTINATION_SQL_STREAM" (ITEM VARCHAR(1024), ITEM_COUNT DOUBLE);
CREATE
OR REPLACE PUMP "STREAM_PUMP" AS
INSERT INTO
   "DESTINATION_SQL_STREAM"
   SELECT
      STREAM ITEM,
      ITEM_COUNT
   FROM
      TABLE(TOP_K_ITEMS_TUMBLING(CURSOR(
      SELECT
```

```
         STREAM *
      FROM
         "SOURCE_SQL_STREAM_001"), 'column1', -- name of column in single quotes10,
 -- number of top items60 -- tumbling window size in seconds));
```

Managed Service for Apache Flink Studio

```
%flinkssql
DROP TABLE IF EXISTS SOURCE_SQL_STREAM_001
CREATE TABLE SOURCE_SQL_STREAM_001 ( TS TIMESTAMP(3), WATERMARK FOR TS as TS -
 INTERVAL '5' SECOND, ITEM VARCHAR(1024),
PRICE DOUBLE)
   WITH ( 'connector' = 'kinesis', 'stream' = 'SOURCE_SQL_STREAM_001',
'aws.region' = 'us-east-1', 'scan.stream.initpos' = 'LATEST', 'format' = 'json',
'json.timestamp-format.standard' = 'ISO-8601');


%flink.ssql(type=update)
SELECT
   *
FROM
   (
      SELECT
         *,
         ROW_NUMBER() OVER (PARTITION BY AGG_WINDOW
      ORDER BY
         ITEM_COUNT DESC) as rownum
      FROM
         (
            select
               AGG_WINDOW,
               ITEM,
               ITEM_COUNT
            from
               (
                  select
                     TUMBLE_ROWTIME(TS, INTERVAL '60' SECONDS) as AGG_WINDOW,
                     ITEM,
                     count(*) as ITEM_COUNT
                  FROM
                     SOURCE_SQL_STREAM_001
                  GROUP BY
```

```
                      TUMBLE(TS, INTERVAL '60' SECONDS),
                      ITEM
                )
          )
    )
where
   rownum <= 3
```

例: ウェブログの解析 (W3C_LOG_PARSE 関数)

SQL-based Kinesis Data Analytics application

```
CREATE
OR REPLACE STREAM "DESTINATION_SQL_STREAM" ( column1 VARCHAR(16),
   column2 VARCHAR(16),
   column3 VARCHAR(16),
   column4 VARCHAR(16),
   column5 VARCHAR(16),
   column6 VARCHAR(16),
   column7 VARCHAR(16));
CREATE
OR REPLACE PUMP "myPUMP" ASINSERT INTO "DESTINATION_SQL_STREAM"
SELECT
   STREAM l.r.COLUMN1,
   l.r.COLUMN2,
   l.r.COLUMN3,
   l.r.COLUMN4,
   l.r.COLUMN5,
   l.r.COLUMN6,
   l.r.COLUMN7
FROM
   (
      SELECT
         STREAM W3C_LOG_PARSE("log", 'COMMON')
      FROM
         "SOURCE_SQL_STREAM_001"
   )
   AS l(r);
```

Managed Service for Apache Flink Studio

```
%flink.ssql(type=update)
```

```
DROP TABLE IF EXISTS SOURCE_SQL_STREAM_001 CREATE TABLE SOURCE_SQL_STREAM_001 ( log
 VARCHAR(1024))
   WITH ( 'connector' = 'kinesis',
          'stream' = 'SOURCE_SQL_STREAM_001',
          'aws.region' = 'us-east-1',
          'scan.stream.initpos' = 'LATEST',
          'format' = 'json',
          'json.timestamp-format.standard' = 'ISO-8601');

% flink.ssql(type=update)
   select
      SPLIT_INDEX(log, ' ', 0),
      SPLIT_INDEX(log, ' ', 1),
      SPLIT_INDEX(log, ' ', 2),
      SPLIT_INDEX(log, ' ', 3),
      SPLIT_INDEX(log, ' ', 4),
      SPLIT_INDEX(log, ' ', 5),
      SPLIT_INDEX(log, ' ', 6)
   from
      SOURCE_SQL_STREAM_001;
```

例: 複数のフィールドへの文字列の分割 (VARIABLE_COLUMN_LOG_PARSE 関数)

SQL-based Kinesis Data Analytics application

```
CREATE
OR REPLACE STREAM "DESTINATION_SQL_STREAM"( "column_A" VARCHAR(16),
   "column_B" VARCHAR(16),
   "column_C" VARCHAR(16),
   "COL_1" VARCHAR(16),
   "COL_2" VARCHAR(16),
   "COL_3" VARCHAR(16));
CREATE
OR REPLACE PUMP "SECOND_STREAM_PUMP" AS INSERT INTO "DESTINATION_SQL_STREAM"
SELECT
   STREAM t."Col_A",
   t."Col_B",
   t."Col_C",
   t.r."COL_1",
   t.r."COL_2",
   t.r."COL_3"
FROM
```

```
   (
      SELECT
         STREAM "Col_A",
         "Col_B",
         "Col_C",
         VARIABLE_COLUMN_LOG_PARSE ("Col_E_Unstructured",
         'COL_1 TYPE VARCHAR(16),
         COL_2 TYPE VARCHAR(16),
         COL_3 TYPE VARCHAR(16)', ',') AS r
      FROM
         "SOURCE_SQL_STREAM_001"
   )
   as t;
```

Managed Service for Apache Flink Studio

```
%flink.ssql(type=update)
DROP TABLE IF EXISTS SOURCE_SQL_STREAM_001 CREATE TABLE SOURCE_SQL_STREAM_001 ( log
 VARCHAR(1024))
   WITH ( 'connector' = 'kinesis',
          'stream' = 'SOURCE_SQL_STREAM_001',
          'aws.region' = 'us-east-1',
          'scan.stream.initpos' = 'LATEST',
          'format' = 'json',
          'json.timestamp-format.standard' = 'ISO-8601');

% flink.ssql(type=update)
   select
      SPLIT_INDEX(log, ' ', 0),
      SPLIT_INDEX(log, ' ', 1),
      SPLIT_INDEX(log, ' ', 2),
      SPLIT_INDEX(log, ' ', 3),
      SPLIT_INDEX(log, ' ', 4),
      SPLIT_INDEX(log, ' ', 5)
)
from
   SOURCE_SQL_STREAM_001;
```

Joins

SQL-based Kinesis Data Analytics application

```
CREATE
OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
   ticker_symbol VARCHAR(4),
   "Company" varchar(20),
   sector VARCHAR(12),
   change DOUBLE,
   price DOUBLE);
CREATE
OR REPLACE PUMP "STREAM_PUMP" AS
INSERT INTO
   "DESTINATION_SQL_STREAM"
   SELECT
      STREAM ticker_symbol,
      "c"."Company",
      sector,
      change,
      priceFROM "SOURCE_SQL_STREAM_001"
      LEFT JOIN
         "CompanyName" as "c"
         ON "SOURCE_SQL_STREAM_001".ticker_symbol = "c"."Ticker";
```

Managed Service for Apache Flink Studio

```
Query 1 - % flink.ssql(type =
update
) CREATE TABLE DESTINATION_SQL_STREAM (
   TICKER_SYMBOL VARCHAR(4),
   SECTOR VARCHAR(12),
   CHANGE INT,
   PRICE DOUBLE )
PARTITIONED BY (TICKER_SYMBOL) WITH (
   'connector' = 'kinesis',
   'stream' = 'kinesis-analytics-demo-stream',
   'aws.region' = 'us-east-1',
   'scan.stream.initpos' = 'LATEST',
   'format' = 'json',
   'json.timestamp-format.standard' = 'ISO-8601');

Query 2 - CREATE TABLE CompanyName (
```

```
   Ticker VARCHAR(4),
   Company VARCHAR(4)) WITH (
      'connector' = 'filesystem',
      'path' = 's3://kda-demo-sample/TickerReference.csv',
      'format' = 'csv' );

Query 3 - % flink.ssql(type =
update
)
   SELECT
      TICKER_SYMBOL,
      c.Company,
      SECTOR,
      CHANGE,
      PRICE
   FROM
      DESTINATION_SQL_STREAM
      LEFT JOIN
         CompanyName as c
         ON DESTINATION_SQL_STREAM.TICKER_SYMBOL = c.Ticker;
```

エラー

SQL-based Kinesis Data Analytics application

```
SELECT
   STREAM ticker_symbol,
   sector,
   change,
   (
      price / 0
   )
   as ProblemColumnFROM "SOURCE_SQL_STREAM_001"
WHERE
   sector SIMILAR TO '%TECH%';
```

Managed Service for Apache Flink Studio

```
Query 1 - % flink.ssql(type =
update
) DROP TABLE IF EXISTS DESTINATION_SQL_STREAM;
CREATE TABLE DESTINATION_SQL_STREAM (
```

```
   TICKER_SYMBOL VARCHAR(4),
   SECTOR VARCHAR(16),
   CHANGE DOUBLE,
   PRICE DOUBLE )
PARTITIONED BY (TICKER_SYMBOL) WITH (
   'connector' = 'kinesis',
   'stream' = 'kinesis-analytics-demo-stream',
   'aws.region' = 'us-east-1',
   'scan.stream.initpos' = 'LATEST',
   'format' = 'json',
   'json.timestamp-format.standard' = 'ISO-8601');

Query 2 - % flink.pyflink @udf(input_types = [DataTypes.BIGINT()],
   result_type = DataTypes.BIGINT()) def DivideByZero(price): try: price / 0
except
: return - 1 st_env.register_function("DivideByZero",
   DivideByZero)

   Query 3 - % flink.ssql(type =
update
)
   SELECT
      CURRENT_TIMESTAMP AS ERROR_TIME,
      *
   FROM
      (
         SELECT
            TICKER_SYMBOL,
            SECTOR,
            CHANGE,
            DivideByZero(PRICE) as ErrorColumn
         FROM
            DESTINATION_SQL_STREAM
         WHERE
            SECTOR SIMILAR TO '%TECH%'
      )
      AS ERROR_STREAM;
```

Random Cut Forest ワークロードの移行

Random Cut Forest を使用するワークロードを Kinesis Analytics for SQL から Managed Service for Apache Flink に移行することを検討している方のために、このAWS ブログ記事では、Managed

Service for Apache Flink を使用して異常検出用のオンライン RCF アルゴリズムを実行する方法を紹介します。

ソースとしての Kinesis Data Firehose を Kinesis Data Streams に置き換える

詳細なチュートリアルについては、「Converting-KDASQL-KDAStudio/」を参照してください。

次の演習では、Amazon Managed Service for Apache Flink Studio を使用するためにデータフローを変更します。これは、Amazon Kinesis Data Firehose から Amazon Kinesis Data Streams に切り替えることも意味します。

まずは一般的な KDA-SQL アーキテクチャを紹介し、次に Amazon Managed Service for Apache Flink Studio とAmazon Kinesis Data Streams を使用してこれを置き換える方法を示します。または、ここで AWS CloudFormation テンプレートを起動することもできます。

Amazon Kinesis Data Analytics-SQL と Amazon Kinesis Data Firehose

Amazon Kinesis Data Analytics SQL アーキテクチャフローは次のとおりです。

まず、レガシーの Amazon Kinesis Data Analytics-SQL と Amazon Kinesis Data Firehose セットアップについて調べます。このユースケースでは、株式ティッカーや価格を含む取引データが外部ソースから Amazon Kinesis システムにストリーミングされる取引市場を扱います。Amazon Kinesis Data Analytics for SQL は、入力ストリームを使用してタンブリングウィンドウなどのウィンドウクエリを実行し、各株式ティッカーの 1 分間の取引量と `min`、`max`、`average` 取引価格を特定します。

Amazon Kinesis Data Analytics-SQL は Amazon Kinesis Data Firehose API からデータを取り込むように設定されています。処理の後、Amazon Kinesis Data Analytics-SQL は処理されたデータを別の Amazon Kinesis Data Firehose に送信します。これがその出力を Amazon S3 バケットに保存します。

この場合は、 Amazon Kinesis Data Generator を使用します。Amazon Kinesis Data Generator を使用すると、Amazon Kinesis Data Streams または Amazon Kinesis Data Firehose 配信ストリームに

テストデータを送信できます。開始するには、こちらの手順に従ってください。手順に記載されているテンプレートの代わりに、こちらの AWS CloudFormation テンプレートを使用してください。https://awslabs.github.io/amazon-kinesis-data-generator/web/help.html

AWS CloudFormation テンプレートを実行すると、出力セクションに Amazon Kinesis Data Generator URL が表示されます。ここで設定した Cognito ユーザー ID とパスワードを使用してポータルにログインします。リージョンとターゲットストリーム名を選択します。現在の状態については、Amazon Kinesis Data Firehose 配信ストリームを選択してください。新しい状態については、Amazon Kinesis Data Firehose 配信ストリームを選択してください。要件に応じて複数のテンプレートを作成し、ターゲットストリームに送信する前に [テストテンプレート] ボタンを使用すると、テンプレートをテストできます。

Amazon Kinesis Data Generator を使用したサンプルペイロードを以下に示します。Data Generator は、入力された Amazon Kinesis Firehose Streams をターゲットにして、データを継続的にストリーミングします。Amazon Kinesis SDK クライアントは、他のプロデューサーからのデータも送信できます。

```
2023-02-17 09:28:07.763,"AAPL",5032023-02-17 09:28:07.763,
"AMZN",3352023-02-17 09:28:07.763,
"GOOGL",1852023-02-17 09:28:07.763,
"AAPL",11162023-02-17 09:28:07.763,
"GOOGL",1582
```

次の JSON を使用して、取引日時、株式ティッカー、株価をランダムに生成します。

```
date.now(YYYY-MM-DD HH:mm:ss.SSS),
"random.arrayElement(["AAPL","AMZN","MSFT","META","GOOGL"])",
random.number(2000)
```

[データを送信]を選択すると、Generator はモックデータの送信を開始します。

外部システムが Amazon Kinesis Data Firehose にデータをストリーミングします。Amazon Kinesis Data Analytics for SQL アプリケーションを使用すると、Java を使用してストリーミングデータを処理および分析できます。このサービスを使用すると、ストリーミングソースに対する SQL コードを作成して実行し、時系列分析の実行、ダッシュボードへのリアルタイムフィード、メトリクスのリアルタイム作成を行うことができます。Amazon Kinesis Data Analytics for SQL アプリケーションでは、入力ストリームの SQL クエリから送信先ストリームを作成し、その送信先ストリームを別の Amazon Kinesis Data Firehose に送信できます。送信先の Amazon Kinesis Data Firehose は、分析データを最終状態として Amazon S3 に送信できます。

Amazon Kinesis Data Analytics-SQL レガシーコードは SQL 標準の拡張に基づいています。

Amazon Kinesis Data Analytics-SQL では、次のクエリを使用します。まず、クエリ出力の送信先ストリームを作成します。次に、PUMP を使用します。これは Amazon Kinesis Data Analytics Repository Object (SQL 標準の拡張) で、継続的に実行される `INSERT INTO stream SELECT ... FROM` クエリ機能を提供するため、クエリの結果を名前付きストリームに継続的に入力できます。

```
CREATE
OR REPLACE STREAM "DESTINATION_SQL_STREAM" (EVENT_TIME TIMESTAMP,
INGEST_TIME TIMESTAMP,
TICKER VARCHAR(16),
VOLUME BIGINT,
AVG_PRICE DOUBLE,
MIN_PRICE DOUBLE,
MAX_PRICE DOUBLE);

CREATE
OR REPLACE PUMP "STREAM_PUMP" AS
INSERT INTO
   "DESTINATION_SQL_STREAM"
   SELECT
      STREAM STEP("SOURCE_SQL_STREAM_001"."tradeTimestamp" BY INTERVAL '60' SECOND) AS
 EVENT_TIME,
      STEP("SOURCE_SQL_STREAM_001".ROWTIME BY INTERVAL '60' SECOND) AS
 "STREAM_INGEST_TIME",
      "ticker",
       COUNT(*) AS VOLUME,
      AVG("tradePrice") AS AVG_PRICE,
      MIN("tradePrice") AS MIN_PRICE,
      MAX("tradePrice") AS MAX_PRICEFROM "SOURCE_SQL_STREAM_001"
   GROUP BY
      "ticker",
      STEP("SOURCE_SQL_STREAM_001".ROWTIME BY INTERVAL '60' SECOND),
      STEP("SOURCE_SQL_STREAM_001"."tradeTimestamp" BY INTERVAL '60' SECOND);
```

上記の SQL では 2 つのタイムウィンドウを使用します。`tradeTimestamp` は受信ストリームのペイロードから取得され、`ROWTIME.tradeTimestamp` は `Event Time` または `client-side time` とも呼ばれます。イベントが発生した時間であるため、分析でこの時間を使用するのが望ましい場合がよくあります。しかし、携帯電話やウェブクライアントなど多くのイベントソースは信頼性の高い

時計を持たないため、時間が不正確になる場合があります。さらに、接続性の問題で、レコードがイベントの発生と同じ順序でストリームに現れない場合があります。

アプリケーション内ストリームには、ROWTIME という特別な行も含まれています。Amazon Kinesis Data Analytics によって最初のアプリケーション内ストリームに行が挿入されると、タイムスタンプが保存されます。ROWTIME は、Amazon Kinesis Data Analytics がストリーミングソースからレコードを読み取った後、最初のアプリケーション内ストリームにレコードを挿入した時点のタイムスタンプを反映します。この ROWTIME 値はその後、アプリケーション全体で維持されます。

SQL は、60 秒間隔でティッカーのカウント (volume) と min、max、average 価格を特定します。

時間ベースのウィンドウクエリでこれらの時間を使用するには、それぞれ利点と欠点があります。これらの時間を 1 つ以上選択し、またそれに伴う欠点に対処する戦略をお客様のユースケースシナリオに基づいて選択します。

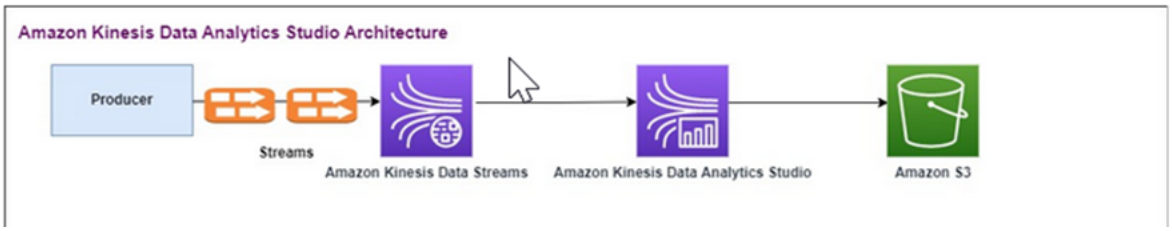
2 ウィンドウ戦略では、2 つの時間ベースの値 (両方の ROWTIME、イベント時間などのもう 1 つの時間) を使用します。

- 次の例に示すように、クエリで結果を発行する頻度を制御する ROWTIME を最初のウィンドウとして使用します。論理時間としては使用されません。
- 分析に関連付ける論理時間であるその他の時間のうち 1 つを使用します。この時間は、いつイベントが発生したかを示します。次の例では、分析の目的はレコードをグループ化し、ティッカーでカウントを返すことです。

Amazon Managed Service for Apache Flink Studio

更新されたアーキテクチャでは、Amazon Kinesis Data Firehose を Amazon Kinesis Data Streams に置き換えます。Amazon Kinesis Data Analytics for SQL アプリケーションは Amazon Managed Service for Apache Flink Studio に置き換えられました。Apache Flink コードは Apache Zeppelin ノートブック内でインタラクティブに実行されます。Amazon Managed Service for Apache Flink Studio は、収集した取引データを保存用の Amazon S3 バケットに送信します。その手順を以下に示します。

Amazon Managed Service for Apache Flink Studio のアーキテクチャフローは次のとおりです。



Kinesis データストリームを作成する

コンソールを使用してデータストリームを作成するには

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/kinesis で Kinesis コンソールを開きます。
2. ナビゲーションバーで、リージョンセレクターを展開し、リージョンを選択します。
3. [データストリームの作成] を選択します。
4. [Kinesis ストリームの作成] ページで、データストリームの名前を入力し、デフォルトの [オンデマンド] 容量モードを選択します。

   [オンデマンド] モードの場合、[Kinesis ストリームの作成] を選択して、データストリームを作成することができます。

   ストリームの作成中、[Kinesis ストリーム] ページのストリームのステータスは、Creating になります。ストリームを使用する準備が完了すると、ステータスは Active に変わります。
5. ストリームの名前を選択します。[ストリームの詳細] ページには、ストリーム設定の概要とモニタリング情報が表示されます。
6. Amazon Kinesis Data Generator で、ストリーム/配信ストリームを新しい Amazon Kinesis Data Streams TRADE_SOURCE_STREAM に変更します。

   JSON とペイロードは Amazon Kinesis Data Analytics-SQL に使用したものと同じになります。Amazon Kinesis Data Generator を使用してサンプルの取引ペイロードデータを作成し、この演習では TRADE_SOURCE_STREAM データストリームをターゲットにします。

   ```
   {{date.now(YYYY-MM-DD HH:mm:ss.SSS)}},
   "{{random.arrayElement(["AAPL","AMZN","MSFT","META","GOOGL"])}}",
   {{random.number(2000)}}
   ```
7. AWS Management Console 「Managed Service for Apache Flink」に移動し、「アプリケーションの作成」を選択します。

8. 左側のナビゲーションペインで、[Studio ノートブック]、[ノートブックインスタンスの作成] の順に選択します。
9. Studio ノートブック名を入力します。
10. [AWS Glue データベース] で、ソースと宛先のメタデータを定義する既存の AWS Glue データベースを指定します。 AWS Glue データベースがない場合は、作成 を選択し、次の操作を行います。

    a. AWS Glue コンソールで、左側のメニューからデータカタログのデータベースを選択します。
    b. [データベースの作成] を選択します。
    c. [データベースの作成] ページで、データベースの名前を入力します。[場所 — オプション] セクションで、[Amazon S3 を参照する] を選択した上で、Amazon S3 バケットを選択します。Amazon S3 バケットをまだセットアップしていない場合は、このステップをスキップし、後に再開することができます。
    d. (オプション)。データベースの説明を入力します。
    e. [データベースの作成] を選択します。
11. [ノートブックの作成)] を選択します。
12. ノートブックを作成したら、[実行] を選択します。
13. ノートブックが正常に起動したら、[Apache Zeppelin で開く] を選択して Zeppelin ノートブックを起動します。
14. Zeppelin ノートブックのページで [新規ノートを作成] を選択し、MarketDataFeed と命名します。

Flink SQL コードについては以下で説明しますが、まず Zeppelin ノートブックの画面は次のようになります。ノートブック内の各ウィンドウは個別のコードブロックで、一度に 1 つずつ実行できます。

Amazon Managed Service for Apache Flink Studio Code

Amazon Managed Service for Apache Flink Studio は、Zeppelin ノートブックを使用してコードを実行します。この例では、Apache Flink 1.13 に基づく ssql コードへのマッピングが行われています。以下では、Zeppelin ノートブックのコードを 1 ブロックずつ示します。

Zeppelin ノートブックでコードを実行する前に、Flink 設定コマンドを実行する必要があります。コード (ssql、Python、または Scala) を実行した後に設定を変更する必要がある場合は、ノートブックを停止して再起動する必要があります。この例では、チェックポイントを設定する必要がありま

す。Amazon S3 のファイルにデータをストリーミングできるようにするには、チェックポイントが必要です。これにより、Amazon S3 へのデータストリームをファイルにフラッシュできます。以下のステートメントは、間隔を 5000 ミリ秒に設定します。

```
%flink.conf
execution.checkpointing.interval 5000
```

`%flink.conf` は、このブロックが設定ステートメントであることを示します。チェックポイントを含む Flink 設定の詳細については、「Apache Flink Checkpointing」を参照してください。

ソース Amazon Kinesis Data Streams の入力テーブルは、以下の Flink ssql コードを使用して作成されます。`TRADE_TIME` フィールドには、データジェネレーターが作成した日付/時刻が格納されることに注意してください。

```
%flink.ssql

DROP TABLE IF EXISTS TRADE_SOURCE_STREAM;
CREATE TABLE TRADE_SOURCE_STREAM (--`arrival_time` TIMESTAMP(3) METADATA FROM
 'timestamp' VIRTUAL,
TRADE_TIME TIMESTAMP(3),
WATERMARK FOR TRADE_TIME as TRADE_TIME - INTERVAL '5' SECOND,TICKER STRING,PRICE
 DOUBLE,
STATUS STRING)WITH ('connector' = 'kinesis','stream' = 'TRADE_SOURCE_STREAM',
'aws.region' = 'us-east-1','scan.stream.initpos' = 'LATEST','format' = 'csv');
```

入力ストリームは次のステートメントで確認できます。

```
%flink.ssql(type=update)-- testing the source stream

select * from TRADE_SOURCE_STREAM;
```

集計データを Amazon S3 に送信する前に、Amazon Managed Service for Apache Flink Studio でタンブリングウィンドウの選択クエリを使用してデータを直接表示できます。これにより、取引データが 1 分のタイムウィンドウに集約されます。%flink.ssql ステートメントには (type=update) という指定が必要であることに注意してください。

```
%flink.ssql(type=update)

select TUMBLE_ROWTIME(TRADE_TIME,
INTERVAL '1' MINUTE) as TRADE_WINDOW,
TICKER, COUNT(*) as VOLUME,
```

```
AVG(PRICE) as AVG_PRICE,
MIN(PRICE) as MIN_PRICE,
MAX(PRICE) as MAX_PRICE FROM TRADE_SOURCE_STREAMGROUP BY TUMBLE(TRADE_TIME, INTERVAL
 '1' MINUTE), TICKER;
```

そうすると、Amazon S3 でターゲット用のテーブルを作成できます。ウォーターマークを使用する必要があります。ウォーターマークは、これ以上遅延イベントが発生しないと確信できる時点を示す進捗指標です。ウォーターマークが表示されるのは、到着が遅れた場合を考慮に入れるためです。この ‘5’ Second 間隔により、5 秒遅れて Amazon Kinesis Data Stream に取引を入力することが可能になり、このウィンドウ内にタイムスタンプが存在する場合は取引が含まれるようになります。詳細については、「Generating Watermarks」を参照してください。

```
%flink.ssql(type=update)

DROP TABLE IF EXISTS TRADE_DESTINATION_S3;
CREATE TABLE TRADE_DESTINATION_S3 (
TRADE_WINDOW_START TIMESTAMP(3),
WATERMARK FOR TRADE_WINDOW_START as TRADE_WINDOW_START - INTERVAL '5' SECOND,
TICKER STRING,
VOLUME BIGINT,
AVG_PRICE DOUBLE,
MIN_PRICE DOUBLE,
MAX_PRICE DOUBLE)
WITH ('connector' = 'filesystem','path' = 's3://trade-destination/','format' = 'csv');
```

このステートメントはデータを TRADE_DESTINATION_S3 に挿入します。TUMPLE_ROWTIME はタンブリングウィンドウの上限を含むタイムスタンプです。

```
%flink.ssql(type=update)

insert into TRADE_DESTINATION_S3
select TUMBLE_ROWTIME(TRADE_TIME,
INTERVAL '1' MINUTE),
TICKER, COUNT(*) as VOLUME,
AVG(PRICE) as AVG_PRICE,
MIN(PRICE) as MIN_PRICE,
MAX(PRICE) as MAX_PRICE FROM TRADE_SOURCE_STREAM
GROUP BY TUMBLE(TRADE_TIME, INTERVAL '1' MINUTE), TICKER;
```

ステートメントを 10 ~ 20 分間実行して、Amazon S3 にデータを蓄積します。その後、ステートメントを中止します。

これにより Amazon S3 内のファイルが閉じて表示可能になります。

内容は以下のようになっています。

part-2d9eca09-1d57-450f-b583-11b33b089518-0-3

```
"2023-03-01 17:23:59.999",AMZN,16,1014.75,224.0,1855.0
"2023-03-01 17:23:59.999",AAPL,20,935.45,123.0,1972.0
"2023-03-01 17:23:59.999",MSFT,20,847.55,15.0,1963.0
"2023-03-01 17:23:59.999",META,23,968.5217391304348,125.0,1995.0
"2023-03-01 17:23:59.999",GOOGL,21,949.0,43.0,1996.0
"2023-03-01 17:26:59.999",GOOGL,19,957.578947368421,180.0,1968.0
"2023-03-01 17:26:59.999",AAPL,25,969.72,24.0,1939.0
"2023-03-01 17:26:59.999",AMZN,24,1021.875,101.0,1995.0
"2023-03-01 17:26:59.999",META,14,715.1428571428571,16.0,1504.0
"2023-03-01 17:26:59.999",MSFT,18,1050.4444444444443,123.0,1977.0
"2023-03-01 17:30:59.999",AAPL,20,860.3,94.0,1776.0
"2023-03-01 17:30:59.999",GOOGL,19,1078.4736842105262,9.0,1732.0
"2023-03-01 17:30:59.999",MSFT,20,1180.35,181.0,1981.0
"2023-03-01 17:30:59.999",AMZN,20,974.3,50.0,1791.0
"2023-03-01 17:30:59.999",META,21,902.2857142857143,6.0,1896.0
"2023-03-01 17:33:59.999",AAPL,21,1029.8095238095239,121.0,1657.0
"2023-03-01 17:33:59.999",GOOGL,21,1008.6190476190476,62.0,1979.0
"2023-03-01 17:33:59.999",META,21,1119.142857142857,126.0,1916.0
"2023-03-01 17:33:59.999",AMZN,27,1073.2592592592594,179.0,1849.0
"2023-03-01 17:33:59.999",MSFT,10,1353.4,584.0,1996.0
"2023-03-01 17:36:59.999",GOOGL,18,1262.5,15.0,1997.0
"2023-03-01 17:36:59.999",MSFT,19,780.8947368421053,30.0,1558.0
"2023-03-01 17:36:59.999",AAPL,15,1263.1333333333334,185.0,1928.0
"2023-03-01 17:36:59.999",AMZN,20,882.85,153.0,1764.0
"2023-03-01 17:36:59.999",META,28,858.5,14.0,1917.0
"2023-03-01 17:40:59.999",AMZN,22,976.9545454545455,98.0,1878.0
"2023-03-01 17:40:59.999",GOOGL,25,830.32,39.0,1991.0
"2023-03-01 17:40:59.999",MSFT,18,1325.0,28.0,1966.0
"2023-03-01 17:40:59.999",META,19,855.578947368421,96.0,1725.0
"2023-03-01 17:40:59.999",AAPL,16,1134.25,1.0,1939.0
```

この AWS CloudFormation テンプレートを使用してインフラストラクチャを作成できます。

AWS CloudFormation は、 AWS アカウントに次のリソースを作成します。

- Amazon Kinesis Data Streams
- Amazon Managed Service for Apache Flink Studio
- AWS Glue データベース
- Amazon S3 バケット
- Amazon Managed Service for Apache Flink Studio で適切なリソースにアクセスするための IAM ロールとポリシー

ノートブックをインポートし、Amazon S3 バケット名を、 によって作成された新しい Amazon S3 バケットに変更します AWS CloudFormation。

```
%flink.ssql(type=update)

DROP TABLE IF EXISTS  TRADE_DESTINATION_S3;
CREATE TABLE TRADE_DESTINATION_S3 (
TRADE_WINDOW_START TIMESTAMP(3),
WATERMARK FOR TRADE_WINDOW_START as TRADE_WINDOW_START - INTERVAL '5' SECOND,
TICKER STRING,
VOLUME BIGINT,
AVG_PRICE DOUBLE,
MIN_PRICE DOUBLE,
MAX_PRICE DOUBLE)
WITH ('connector' = 'filesystem','path' = 's3://kda-studio-test-stack-markettradinganalyticsc[redacted]/','format' = 'csv');
```

詳細を見る

Managed Service for Apache Flink Studio の使用方法の詳細については、次の追加リリソースを参照してください。

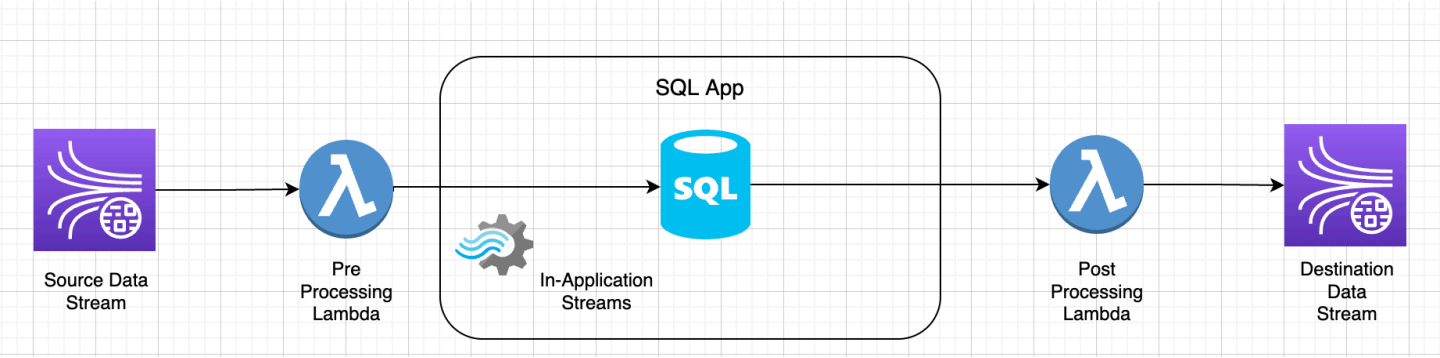
- Managed Service for Apache Flink Studio Notebooks 開発者ガイド
- Apache Flink 1.13 のドキュメント
- Managed Service for Apache Flink Studio ワークショップ
- Apache Flink Windowing
- Amazon Kinesis Data Analytics 開発者ガイド — Kinesis Data Analytics Stream から S3 バケットへの書き込み

ユーザー定義関数 (UDF) の活用

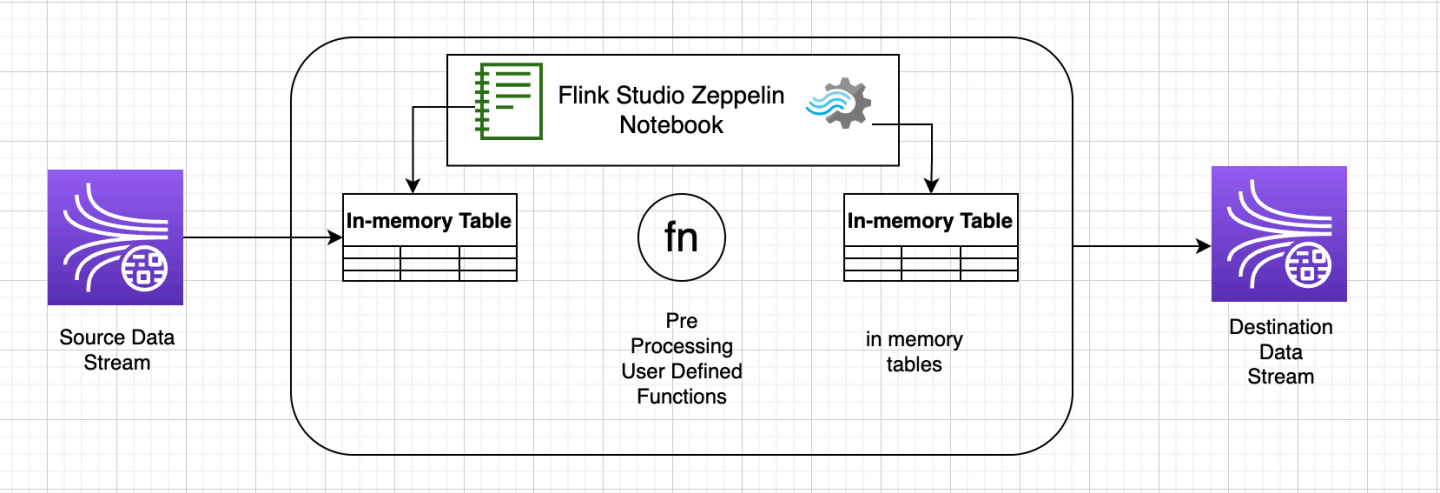
このパターンの目的は、Kinesis Data Analytics-Studio Zeppelin ノートブックの UDF を活用して Kinesis ストリームのデータを処理する方法を説明することです。Managed Service for Apache Flink Studio は、Apache Flink を使用して高度な分析機能を提供します。これには、1 回限りの処理セマンティクス、イベント時間のウィンドウ、ユーザー定義関数とカスタム統合を使用した拡張性、命令型言語サポート、永続的なアプリケーション状態、水平スケーリング、複数のデータソースのサポート、拡張可能な統合などが含まれます。これらの機能は、データストリーム処理の正確性、完全性、一貫性、信頼性を確保するために不可欠で、Amazon Kinesis Data Analytics for SQL では利用できません。

このサンプルアプリケーションでは、KDA-Studio Zeppelin ノートブックの UDF を活用して Kinesis ストリームのデータを処理する方法を紹介します。Kinesis Data Analytics 用 Studio ノートブックでは、データストリームをリアルタイムでインタラクティブにクエリし、標準 SQL、Python、Scala を使用してストリーム処理アプリケーションを簡単に構築して実行できます。を数回クリックするだけで AWS Management Console、サーバーレスノートブックを起動してデータストリームをクエリし、数秒で結果を取得できます。詳細については、「Studio ノートブックを Kinesis Data Analytics for Apache Flink で使用する」を参照してください。

KDA-SQL アプリケーションのデータの前処理/後処理に使用される Lambda 関数。



KDA-Studio Zeppelin ノートブックを使用してデータを前処理または後処理するためのユーザー定義関数



ユーザー定義関数 (UDF)

一般的なビジネスロジックをオペレータに再利用するには、ユーザー定義関数を参照してデータストリームを変換すると便利です。これは Managed Service for Apache Flink Studio ノートブック内で行うことも、外部から参照されるアプリケーション jar ファイルとして行うこともできます。ユーザー定義関数を利用すると、ストリーミングデータに対して実行する変換やデータエンリッチメントを簡略化できます。

ノートブックでは、個人の電話番号を匿名化する機能を備えた単純な Java アプリケーション jar を参照することになります。Python や Scala の UDF を記述してノートブック内で使用することもできます。アプリケーション jar を Pyflink ノートブックにインポートする機能を強調するため、Java アプリケーション jar を選択しています。

環境設定

このガイドに従ってストリーミングデータを操作するには、 AWS CloudFormation スクリプトを使用して以下のリソースを起動します。

- Kinesis Data Streams をソースとする場合
- Glue データベース
- IAM ロール
- Managed Service for Apache Flink アプリケーション
- Managed Service for Apache Flink Studio アプリケーションを開始する Lambda 関数
- 上記の Lambda 関数を実行する Lambda ロール
- Lambda 関数を呼び出すカスタムリソース

AWS CloudFormation テンプレートはこちらからダウンロードしてください。

AWS CloudFormation スタックを作成する

1. に移動 AWS Management Console し、サービスのリストで CloudFormation を選択します。
2. [クラウドの形成] ページでは、[スタックの作成]、[新しいリソースの使用 (スタンダード)] の順に選択します。
3. [スタックの作成] ページで、[テンプレートファイルをアップロード] を選択してから、以前にダウンロードした `kda-flink-udf.yml` を選択します。ファイルを選択してから、[次へ] を選択します。
4. テンプレートには `kinesis-UDF` のような覚えやすい名前を付け、別の名前を付けたい場合は input-stream などの入力パラメータを更新します。[Next (次へ)] を選択します。
5. [スタックオプションの設定] ページで、必要に応じて [タグ] を追加し、[次へ] を選択します。
6. [レビュー] ページで IAM リソースの作成を許可するボックスにチェックを入れ、[提出] を選択します。

起動するリージョンによっては、 AWS CloudFormation スタックの起動に 10 ~ 15 分かかる場合があります。スタック全体の `CREATE_COMPLETE` ステータスが表示されたら、次に進むことができます。

Managed Service for Apache Flink Studio ノートブックで作業する

Kinesis Data Analytics 用 Studio ノートブックでは、データストリームをリアルタイムでインタラクティブにクエリし、標準 SQL、Python、Scala を使用してストリーム処理アプリケーションを簡単に構築して実行できます。を数回クリックするだけで AWS Management Console、サーバーレスノートブックを起動してデータストリームをクエリし、数秒で結果を取得できます。

ノートブックはウェブベースの開発環境です。ノートブックでは、Apache Flink が提供する高度なデータストリーム処理機能と組み合わせて、シンプルでインタラクティブな開発環境を実現できます。Studio ノートブックは、Apache Zeppelin をベースとしたノートブックを使用し、ストリーム処理エンジンとして Apache Flink を使用しています。Studio ノートブックはこれらのテクノロジーをシームレスに組み合わせて、あらゆるスキルを持つ開発者がデータストリームの高度な分析にアクセスできるようにします。

Apache Zeppelin は、Studio ノートブックに次のような分析ツール一式を提供します。

- データの視覚化
- ファイルにデータをエクスポートする
- 分析を容易にする出力形式の制御

ノートブックの使用

1. に移動 AWS Management Console し、サービスのリストで Amazon Kinesis を選択します。
2. 左側のナビゲーションページで [Analytics アプリケーション] を選択してから[Studio ノートブック] を選択します。
3. KinesisDataAnalyticsStudio ノートブックが実行されていることを確認します。
4. ノートブックを選択し、[Apache Zeppelin で開く] を選択します。
5. Kinesis Stream へのデータの読み取りと読み込みに使用するデータプロデューサー Zeppelin ノートブックファイルをダウンロードします。
6. `Data Producer` Zeppelin ノートブックをインポートします。ノートブックで、入力 `STREAM_NAME` と `REGION` のコードを変更してください。入力ストリーム名はAWS CloudFormation スタック出力にあります。
7. [この段落を実行] ボタンを選択して Data Producer ノートブックを実行し、入力の Kinesis Data Stream にサンプルデータを挿入します。

8. サンプルデータが読み込まれている間に、MaskPhoneNumber-Interactive ノートブックをダウンロードします。このノートブックは、入力データを読み取り、入力ストリームから電話番号を匿名化し、匿名化されたデータを出力ストリームに保存します。
9. `MaskPhoneNumber-interactive` Zeppelin ノートブックをインポートします。
10. ノートブック内の各段落を実行します。

    a. 第 1 段落では、電話番号を匿名化するユーザー定義関数をインポートします。

    ```
    %flink(parallelism=1)
    import com.mycompany.app.MaskPhoneNumber
    stenv.registerFunction("MaskPhoneNumber", new MaskPhoneNumber())
    ```

    b. 次の段落では、入力ストリームデータを読み取るためのメモリ内テーブルを作成します。ストリーム名と AWS リージョンが正しいことを確認します。

    ```
    %flink.ssql(type=update)

    DROP TABLE IF EXISTS customer_reviews;

    CREATE TABLE customer_reviews (
    customer_id VARCHAR,
    product VARCHAR,
    review VARCHAR,
    phone VARCHAR
    )
    WITH (
    'connector' = 'kinesis',
    'stream' = 'KinesisUDFSampleInputStream',
    'aws.region' = 'us-east-1',
    'scan.stream.initpos' = 'LATEST',
    'format' = 'json');
    ```

    c. データがメモリ内テーブルに読み込まれているか確認してください。

    ```
    %flink.ssql(type=update)
    select * from customer_reviews
    ```

    d. ユーザー定義関数を呼び出して、電話番号を匿名化します。

    ```
    %flink.ssql(type=update)
    ```

```
select customer_id, product, review, MaskPhoneNumber('mask_phone', phone) as
 phoneNumber from customer_reviews
```

e. 電話番号がマスクされたので、番号をマスクしたビューを作成します。

```
%flink.ssql(type=update)

DROP VIEW IF EXISTS sentiments_view;

CREATE VIEW
    sentiments_view
AS
  select customer_id, product, review, MaskPhoneNumber('mask_phone', phone) as
 phoneNumber from customer_reviews
```

f. データを検証します。

```
%flink.ssql(type=update)
select * from sentiments_view
```

g. 出力 Kinesis Stream 用のメモリ内テーブルを作成します。ストリーム名と AWS リージョンが正しいことを確認します。

```
%flink.ssql(type=update)

DROP TABLE IF EXISTS customer_reviews_stream_table;

CREATE TABLE customer_reviews_stream_table (
customer_id VARCHAR,
product VARCHAR,
review VARCHAR,
phoneNumber varchar
)
WITH (
'connector' = 'kinesis',
'stream' = 'KinesisUDFSampleOutputStream',
'aws.region' = 'us-east-1',
'scan.stream.initpos' = 'TRIM_HORIZON',
'format' = 'json');
```

h. 更新したレコードをターゲット Kinesis Stream に挿入します。

```
%flink.ssql(type=update)
```

```
INSERT INTO customer_reviews_stream_table
SELECT customer_id, product, review, phoneNumber
FROM sentiments_view
```

i. ターゲット Kinesis Stream のデータを表示して検証します。

```
%flink.ssql(type=update)
select * from customer_reviews_stream_table
```

ノートブックをアプリケーションとしてプロモートする

ノートブックのコードをインタラクティブにテストしたので、コードを耐久性の高いストリーミングアプリケーションとしてデプロイします。まず、アプリケーション設定を変更して Amazon S3 内のコードの場所を指定する必要があります。

1. でノートブックを選択し AWS Management Console、アプリケーション設定としてデプロイ - オプションで編集を選択します。
2. [Amazon S3 のコードの送信先] で、AWS CloudFormation スクリプトによって作成された Amazon S3 バケットを選択します。プロセスには数分かかることがあります。
3. ノートをそのままプロモートすることはできません。実行すると、`Select` ステートメントがサポートされていないためエラーになります。この問題を回避するには、MaskPhoneNumber ストリーミング Zeppelin ノートブックをダウンロードしてください。
4. `MaskPhoneNumber-streaming` Zeppelin ノートブックをインポートします。
5. メモを開き、[KinesisDataAnalyticsStudio のアクション] を選択します。
6. [MaskPhoneNumber-Streaming のビルド] を選択し、S3 にエクスポートします。アプリケーション名を変更し、特殊文字を含めないようにしてください。
7. [ビルドしてエクスポート] を選択します。ストリーミングアプリケーションの設定には数分かかります。
8. ビルドが完了したら、[AWS コンソールを使用してデプロイ] を選択します。
9. 次のページで設定を確認し、正しい IAM ロールを選択していることを確認します。次に、[ストリーミングアプリケーションの作成] を選択します。
10. 数分後、ストリーミングアプリケーションが正常に作成されたことを示すメッセージが表示されます。

永続状態と制限のあるアプリケーションのデプロイに関する詳細については、「永続的な状態のアプリケーションとしてデプロイする」を参照してください。

クリーンアップ

オプションで、AWS CloudFormation スタックをアンインストールできるようになりました。これにより、以前に設定したサービスがすべて削除されます。

Amazon Kinesis Data Analytics for SQL Applications とは

Amazon Kinesis Data Analytics for SQL アプリケーションでは、Java を使用してストリーミングデータを処理および分析できます。このサービスでは、時系列分析の実行、リアルタイムでのダッシュボードへのフィード、リアルタイムでのメトリクス作成を行う強力な SQL コードを、ストリーミングソースに対してすぐに作成、実行できます。

Kinesis Data Analytics を開始するには、ストリーミングデータを連続して読み取り、処理する Kinesis Data Analytics アプリケーションを作成します。このサービスでは、Amazon Kinesis Data Streams および Amazon Data Firehose ストリーミングソースからのデータの取り込みがサポートされています。その後、インタラクティブなエディタを使用して SQL コードを作成し、ライブストリーミングデータでテストします。Kinesis Data Analytics で結果を送信する宛先を設定することもできます。

Kinesis Data Analytics は、Amazon Data Firehose (Amazon S3、Amazon Redshift、Amazon OpenSearch Service、Splunk) AWS Lambda、および Amazon Kinesis Data Streams を送信先としてサポートします。

Amazon Kinesis Data Analytics が適している用途

Amazon Kinesis Data Analytics では、ほぼリアルタイムでデータを連続して読み取り、処理し、保存する SQL コードをすばやく作成できます。標準 SQL クエリをストリーミングデータに対して使用し、データを変換してそこから洞察を提供することができるアプリケーションを構築できます。以下は、Kinesis Data Analytics を使用するシナリオの一部です。

- 時系列分析を生成する – 時間ウィンドウに対してメトリクスを算出し、Kinesis データ配信ストリームを介して値を Amazon S3 または Amazon Redshift にストリーミングできます。
- リアルタイムダッシュボードをフィードする – 集約された処理済みのストリーミングデータの結果を下流に送信してリアルタイムダッシュボードをフィードできます。
- リアルタイムメトリクスを作成する – カスタムメトリクスを作成してトリガーし、リアルタイムのモニタリング、通知、アラームに使用できます。

Kinesis Data Analytics でサポートされている SQL 言語要素の詳細については、「Amazon Kinesis Data Analytics SQL Reference」を参照してください。

Amazon Kinesis Data Analytics を初めてお使いになる方向けの情報

Amazon Kinesis Data Analytics を初めて使用する場合は、次のセクションを順に読むことをお勧めします。

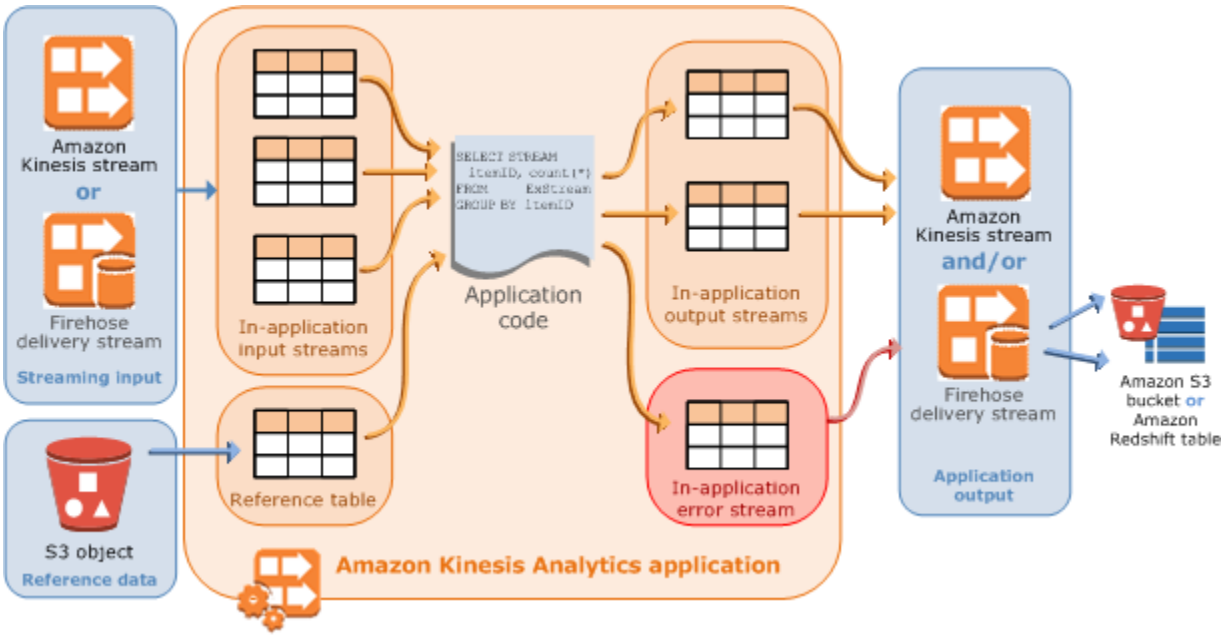
1. このガイドの「仕組み」セクションをお読みください。このセクションでは、エンドツーエンドエクスペリエンスを作成するために使用する Kinesis Data Analytics のさまざまなコンポーネントについて説明しています。詳細については、「Amazon Kinesis Data Analytics for SQL Applications: 仕組み」を参照してください。
2. 「使用開始」実習を実行します。詳細については、「Amazon Kinesis Data Analytics for SQL Applications の開始方法」を参照してください。
3. ストリーミング SQL の概念について学習します。詳細については、「ストリーミング SQL の概念」を参照してください。
4. その他の例を実行します。詳細については、「Kinesis Data Analytics for SQL の例」を参照してください。

Amazon Kinesis Data Analytics for SQL Applications: 仕組み

> **Note**
>
> 2023 年 9 月 12 日以降、Kinesis Data Analytics for SQL をまだ使用していない場合、Kinesis Data Firehose をソースとして使用して新しいアプリケーションを作成することはできません。詳細については、「制限」を参照してください。

アプリケーションは、アカウントで作成できる Amazon Kinesis Data Analytics のプライマリリソースです。 AWS Management Console または Kinesis Data Analytics API を使用して、アプリケーションを作成および管理できます。Kinesis Data Analytics は、アプリケーションを管理するための API オペレーションを提供しています。API オペレーションのリストについては、「アクション」を参照してください。

Kinesis Data Analytics アプリケーションは、ストリーミングデータをリアルタイムで継続的に読み取り処理します。SQL を使用して、入力ストリーミングデータを処理し、出力を生成するアプリケーションを作成します。こうすることで、Kinesis Data Analytics が出力を設定された宛先に書き込みます。次の図は、一般的なアプリケーションのアーキテクチャーです。



各アプリケーションには、名前、説明、バージョン ID、ステータスがあります。Amazon Kinesis Data Analytics は、最初にアプリケーションを作成するときに、バージョン ID を割り当てます。このバージョン ID は、アプリケーション設定の更新時に更新されます。例えば、入力設定の追加、リ

ファレンスデータソースの追加または削除、または出力設定の追加または削除、またはアプリケーションコードの更新などの際に、Kinesis Data Analytics によって現在のアプリケーションバージョン ID が更新されます。また、Kinesis Data Analytics はアプリケーションの作成時および最終更新時のタイムスタンプも保持します。

これらの基本プロパティに加えて、各アプリケーションは、以下で構成されます。

- 入力 – アプリケーションのストリーミングソース。ストリーミングソースとして、Kinesis データストリームと Firehose データ配信ストリームのいずれかを選択できます。入力設定で、アプリケーション内入力ストリームにストリーミングソースをマッピングします。アプリケーション内ストリームは、`SELECT` および `INSERT SQL` オペレーションを実行できる、継続的に更新されるテーブルのようなものです。アプリケーションコードで、中間クエリ結果を保存するための追加のアプリケーション内ストリームを作成することもできます。

  オプションで、スループットを向上させるために単一のソースストリーミングを複数のアプリケーション内入力ストリームに分割できます。詳細については、制限およびアプリケーション入力の設定を参照してください。

  Amazon Kinesis Data Analytics は、各アプリケーション内ストリームに タイムスタンプと ROWTIME 列 というタイムスタンプ列を提供します。この列は時間ベースウィンドウのクエリで使用できます。詳細については、「ウィンドウクエリ」を参照してください。

  オプションでリファレンスデータソースを設定してアプリケーション内の入力データストリームを強化できます。これはアプリケーション内リファレンステーブルになります。リファレンスデータは S3 バケット内のオブジェクトとして保存する必要があります。アプリケーションが起動すると、Amazon Kinesis Data Analytics は Amazon S3 オブジェクトを読み取り、アプリケーション内テーブルを作成します。詳細については、「アプリケーション入力の設定」を参照してください。

- アプリケーションコード – 入力を処理し出力を生成する一連の SQL ステートメントです。アプリケーション内ストリームおよびリファレンステーブルに対して SQL ステートメントを書くことができます。また、JOIN クエリを作成してこれらのソース両方からのデータを結合できます。

Kinesis Data Analytics でサポートされている SQL 言語要素の詳細については、「Amazon Kinesis Data Analytics SQL Reference」を参照してください。

最もシンプルな形式のアプリケーションコードは、ストリーミング入力から選択して結果をストリーミング出力に挿入する単一の SQL ステートメントになります。また、1 つのフィードを出力して次の SQL ステートメントに入力する一連の SQL ステートメントになることもあります。さらに、入力ストリームを複数のストリームに分割するためのアプリケーションコードを書くことができます。その後、これらのストリームを処理するために追加のクエリを適用できます。詳細については、「アプリケーションコード」を参照してください。

- 出力 – アプリケーションコードでは、クエリ結果はアプリケーション内ストリームに入力されます。アプリケーションコードでは、中間結果を保存する 1 つ以上のアプリケーション内ストリームを作成することもできます。その後、オプションでアプリケーション出力を設定してアプリケーション内ストリームのデータを永続化し、アプリケーション出力 (アプリケーション内出力ストリームともいいます) を外部宛先に保持します。外部宛先には、Firehose 配信ストリームまたは Kinesis データストリームを使用できます。これらの宛先について、次の点に注意してください。
  - Firehose 配信ストリームは、結果を Amazon S3、Amazon Redshift、または Amazon OpenSearch Service (OpenSearch Service) に書き込むように設定できます。
  - Amazon S3 や Amazon Redshift ではなく、カスタム宛先に出力するようにアプリケーションを作成することもできます。そのためには、出力環境設定で出力先として Kinesis データストリームを指定します。次に、ストリームをポーリングし、Lambda 関数を呼び出す AWS Lambda ように を設定します。Lambda 関数コードはストリームデータを入力として受け取ります。Lambda 関数コードで、入力データをカスタム宛先に書き込むことができます。詳細については、Amazon Kinesis AWS Lambda で を使用する」を参照してください。

  詳細については、「アプリケーション出力の設定」を参照してください。

以下の点にも注意してください。

- Amazon Kinesis Data Analytics には、ストリーミングソースからレコードを読み取り、アプリケーション出力を外部宛先に書き込むためのアクセス権限が必要です。IAM ロールを使用してこれらのアクセス権限を付与します。

- Kinesis Data Analytics は自動的に各アプリケーションにアプリケーション内エラーストリームを提供します。特定のレコードの処理中にアプリケーションで問題 (たとえばタイプの不一致や到着の遅延など) が発生した場合、そのレコードはエラーストリームに書き込まれます。あとで評価するために、アプリケーション出力を設定して、Kinesis Data Analytics がエラーストリームデータを外部宛先で永続化するようにできます。詳細については、「エラー処理」を参照してください。

- Amazon Kinesis Data Analytics は、アプリケーション出力レコードを設定された宛先に確実に書き込みます。アプリケーションが中断された場合でも、配信モデルでは「少なくとも 1 回」処理を使用します。詳細については、「アプリケーション出力を外部宛先で永続化する配信モデル」を参照してください。

トピック

- アプリケーション入力の設定
- アプリケーションコード
- アプリケーション出力の設定
- エラー処理
- アプリケーションを自動的にスケーリングしてスループットを向上させる
- タグ付けの使用

アプリケーション入力の設定

Amazon Kinesis Data Analytics アプリケーションでは、1 つのストリーミングソースから入力を受け取ることができます。また、オプションで 1 つのリファレンスデータソースを使用できます。詳細については、「Amazon Kinesis Data Analytics for SQL Applications: 仕組み」を参照してください。このトピックのセクションでは、アプリケーション入力ソースについて説明します。

トピック

- ストリーミングソースの設定
- リファレンスソースの設定
- JSONPath の操作
- SQL 入力列へのストリーミングソース要素のマッピング

- ストリーミングデータのスキーマ検出機能の使用
- 静的データに対するスキーマ検出機能の使用
- Lambda 関数を使用したデータの事前処理
- スループットの増加に合わせた入力ストリームの並列処理

ストリーミングソースの設定

アプリケーションを作成するときに、ストリーミングソースを指定します。アプリケーションを作成した後に入力を変更することもできます。Amazon Kinesis Data Analytics では、アプリケーションに対して以下のストリーミングソースがサポートされています。

- Kinesis データストリーム
- Firehose 配信ストリーム

> **Note**
>
> 2023 年 9 月 12 日以降、Kinesis Data Analytics for SQL をまだ使用していない場合、Kinesis Data Firehose をソースとして使用して新しいアプリケーションを作成することはできません。`KinesisFirehoseInput` と一緒に Kinesis Data Analytics for SQL アプリケーションを使用している既存のお客様は、Kinesis Data Analytics を使用して既存のアカウント内で、引き続き `KinesisFirehoseInput` でアプリケーションを追加できます。既存のお客様で、`KinesisFirehoseInput` と一緒に Kinesis Data Analytics for SQL アプリケーションを使用して新規アカウントを作成される場合は、サービス制限拡大フォームを通してケースを作成できます。詳細については、AWS サポート センターを参照してください。新しいアプリケーションを本番環境に移行する前に、必ずテストすることをお勧めします。

> **Note**
>
> Kinesis データストリームが暗号化されている場合、Kinesis Data Analytics は暗号化されたストリームのデータにシームレスにアクセスし、それ以上の設定は必要ありません。Kinesis Data Analytics では、Kinesis Data Streams から読み取った暗号化されていないデータは保存されません。詳細については、「Kinesis Data Streams 用のサーバー側の暗号化とは」を参照してください。

Kinesis Data Analytics は、ストリーミングソースに新しいデータがあるか継続的にポーリングし、入力設定に応じてアプリケーション内ストリームに取り込みます。

> **Note**
>
> アプリケーションの入力として Kinesis ストリームを追加しても、ストリーム内のデータには影響を与えません。同じ Kinesis ストリームに Firehose 配信ストリームなどの別のリソースもアクセスした場合、Firehose 配信ストリームと Kinesis Data Analytics アプリケーションの両方が同じデータを受け取ります。ただし、スループットとスロットリングは影響を受ける可能性があります。

アプリケーションコードは、アプリケーション内ストリームをクエリできます。入力設定の一部として以下を指定します。

- ストリーミングソース – ストリームの Amazon リソースネーム (ARN) と、Kinesis Data Analytics がユーザーに代わってストリームにアクセスするために引き受けることができる IAM ロールを指定します。
- アプリケーション内ストリーム名のプレフィックス – アプリケーションを起動すると、Kinesis Data Analytics によって指定されたアプリケーション内ストリームが作成されます。この名前を使用して、アプリケーションコードでアプリケーション内ストリームにアクセスします。

  オプションで、ストリーミングリソースを複数のアプリケーション内ストリームにマッピングできます。詳細については、「制限」を参照してください。この場合、Amazon Kinesis Data Analytics は、*prefix*_001、*prefix*_002、*prefix*_003 などの名前で、アプリケーション内ストリームを指定された数だけ作成します。デフォルトでは、Kinesis Data Analytics はストリーミングソースを *prefix*_001 という名前の 1 つのアプリケーション内ストリームにマッピングします。

  アプリケーション内ストリームに挿入できる行は、速度の制限があります。そのため、Kinesis Data Analytics では複数のアプリケーション内ストリームをサポートして、より高速にアプリケーションにレコードを届けます。アプリケーションがストリーミングソースのデータをアップし続けることができない場合は、並列処理ユニットを追加してパフォーマンスを向上させることができます。
- マッピングスキーマ – ストリーミングソースでのレコード形式 (JSON、CSV) を指定します。また、ストリームの各レコードが、作成されるアプリケーション内ストリーム内の列にマッピングされる方法も指定します。ここで、列名とデータ型を指定します。

> **Note**
>
> Kinesis Data Analytics は、入力アプリケーション内ストリームの作成時に、識別子 (ストリーム名および列名) に引用符を追加します。このストリームと列をクエリする場合は、完全一致 (大文字と小文字が正確に一致) を使用して引用符内を指定する必要があります。識別子の詳細については、「Amazon Managed Service for Apache Flink SQL リファレンス」の「Identifiers」を参照してください。

Amazon Kinesis Data Analytics コンソールでアプリケーションを作成し、入力を設定できます。その後、コンソールは必要な API コールを行います。アプリケーション入力の設定は、新しいアプリケーション API の作成時、または既存のアプリケーションに入力設定を追加するときに行うことができます。詳細については、「CreateApplication」および「AddApplicationInput」を参照してください。以下は `Createapplication` API リクエストボディの入力設定部分です。

```
"Inputs": [
       {
           "InputSchema": {
               "RecordColumns": [
                   {
                       "Mapping": "string",
                       "Name": "string",
                       "SqlType": "string"
                   }
               ],
               "RecordEncoding": "string",
               "RecordFormat": {
                   "MappingParameters": {
                       "CSVMappingParameters": {
                           "RecordColumnDelimiter": "string",
                           "RecordRowDelimiter": "string"
                       },
                       "JSONMappingParameters": {
                           "RecordRowPath": "string"
                       }
                   },
                   "RecordFormatType": "string"
               }
           },
           "KinesisFirehoseInput": {
               "ResourceARN": "string",
```

```
                "RoleARN": "string"
            },
            "KinesisStreamsInput": {
                "ResourceARN": "string",
                "RoleARN": "string"
            },
            "Name": "string"
        }
    ]
```

リファレンスソースの設定

オプションで既存のアプリケーションにリファレンスデータソースを追加して、ストリーミングソースから入力されるデータを強化できます。リファレンスデータは Amazon S3 バケット内のオブジェクトとして保存する必要があります。アプリケーションが起動すると、Amazon Kinesis Data Analytics は Amazon S3 オブジェクトを読み取り、アプリケーション内リファレンステーブルを作成します。その後、アプリケーションコードでこれをアプリケーション内ストリームと結合できます。

サポートされている形式 (CSV、JSON) で、Amazon S3 オブジェクトにリファレンスデータを保存します。たとえば、アプリケーションで株注文を分析すると仮定します。ストリーミングソースには次に示すレコード形式があるとします。

```
Ticker, SalePrice, OrderId

AMZN     $700        1003
XYZ      $250        1004
...
```

この場合、会社名などの詳細を株価ティッカーに提供するリファレンスデータソースを保持することを検討する場合があります。

```
Ticker, Company
AMZN, Amazon
XYZ, SomeCompany
...
```

アプリケーションのリファレンスデータソースは、API またはコンソールで追加できます。Amazon Kinesis Data Analytics では、以下の API アクションで、リファレンスデータソースを管理します。

- AddApplicationReferenceDataSource

- UpdateApplication

コンソールを使用してリファレンスデータを追加する方法の詳細については、「例: Kinesis Data Analytics アプリケーションにリファレンスデータを追加する」を参照してください。

次の点に注意してください:

- アプリケーションが実行されている場合、Kinesis Data Analytics はアプリケーション内リファレンステーブルを作成し、ただちにリファレンスデータをロードします。
- アプリケーションが実行されていない (例えば、準備完了状態など) 場合、Kinesis Data Analytics は更新された入力設定を保存するだけです。アプリケーションの実行が始まると、Kinesis Data Analytics はリファレンスデータをテーブルとしてアプリケーションにロードします。

Kinesis Data Analytics がアプリケーション内リファレンステーブルを作成した後で、データを更新するとします。Amazon S3 オブジェクトを更新したり、別の Amazon S3 オブジェクトを使用したいという場合があるかもしれません。この場合は、UpdateApplication を明示的に呼び出すか、コンソールで [アクション]、[リファレンスデータテーブルを同期] の順に選択します。Kinesis Data Analytics では、アプリケーション内リファレンステーブルは自動的に更新されません。

リファレンスデータソースとして作成できる Amazon S3 オブジェクトには、サイズの制限があります。詳細については、「制限」を参照してください。オブジェクトのサイズが制限を超えた場合には、Kinesis Data Analytics でデータをロードできません。アプリケーションの状態は実行中と表示されますが、データが読み込まれません。

リファレンスデータソースを追加する場合は、次の情報を指定します。

- S3 バケットおよびオブジェクトキー名 – バケット名およびオブジェクトキーに加えて、ユーザーの代わりにオブジェクトを読み取るために Kinesis Data Analytics が引き受けることができる IAM ロールも指定します。
- アプリケーション内リファレンステーブル名 – Kinesis Data Analytics がこのアプリケーション内テーブルを作成し、Amazon S3 オブジェクトを読み取ってそこに入力します。これは、アプリケーションコードで指定するテーブルの名前です。
- マッピングスキーマ – レコード形式 (JSON, CSV)、Amazon S3 オブジェクトに保存されたデータのエンコードを記述します。また、各データ要素がどのようにアプリケーション内リファレンステーブルにマッピングされるかも記述します。

以下に、`AddApplicationReferenceDataSource` API リクエストの本文を示します。

```
{
    "applicationName": "string",
    "CurrentapplicationVersionId": number,
    "ReferenceDataSource": {
        "ReferenceSchema": {
            "RecordColumns": [
                {
                    "IsDropped": boolean,
                    "Mapping": "string",
                    "Name": "string",
                    "SqlType": "string"
                }
            ],
            "RecordEncoding": "string",
            "RecordFormat": {
                "MappingParameters": {
                    "CSVMappingParameters": {
                        "RecordColumnDelimiter": "string",
                        "RecordRowDelimiter": "string"
                    },
                    "JSONMappingParameters": {
                        "RecordRowPath": "string"
                    }
                },
                "RecordFormatType": "string"
            }
        },
        "S3ReferenceDataSource": {
            "BucketARN": "string",
            "FileKey": "string",
            "ReferenceRoleARN": "string"
        },
        "TableName": "string"
    }
}
```

JSONPath の操作

> **Note**
>
> 2023 年 9 月 12 日以降、Kinesis Data Analytics for SQL をまだ使用していない場合、Kinesis Data Firehose をソースとして使用して新しいアプリケーションを作成することはできません。詳細については、「制限」を参照してください。

JSONPath は、JSON オブジェクトの要素をクエリする標準化された方法です。JSONPath はパス式を使用して、JSON ドキュメントの要素、入れ子要素、配列に移動します。JSON の詳細については、「JSON の入門」を参照してください。

Amazon Kinesis Data Analytics はアプリケーションのソーススキーマで JSONPath 式を使用して、JSON 形式のデータがあるストリーミングソースのデータ要素を識別します。

アプリケーションの入力ストリームにストリーミングデータをマッピングする方法の詳細については、「the section called “SQL 入力列へのストリーミングソース要素のマッピング”」を参照してください。

JSONPath を使用した JSON 要素へのアクセス

JSONPath 式を使用して JSON 形式のさまざまなデータにアクセスする方法について説明します。このセクションの例では、ソースストリームに次の JSON レコードが含まれていると想定します。

```
{
  "customerName":"John Doe",
  "address":
  {
    "streetAddress":
    [
      "number":"123",
      "street":"AnyStreet"
    ],
    "city":"Anytown"
  }
  "orders":
  [
    { "orderId":"23284", "itemName":"Widget", "itemPrice":"33.99" },
    { "orderId":"63122", "itemName":"Gadget", "itemPrice":"22.50" },
    { "orderId":"77284", "itemName":"Sprocket", "itemPrice":"12.00" }
```

```
  ]
}
```

JSON 要素へのアクセス

JSONPath を使用して JSON データの要素をクエリするには、次の構文を使用します。ここで、$ はデータ階層のルート、elementName はクエリを実行する要素ノードの名前を表します。

```
$.elementName
```

次の式では、前述の JSON の例の customerName 要素をクエリします。

```
$.customerName
```

前述の式では、前述の JSON のレコードから次のように返ります。

```
John Doe
```

> **Note**
>
> Path 式では、大文字と小文字が区別されます。式 ($.customername) では、前述の JSON の例から null が返ります。

> **Note**
>
> パス式で指定した場所に要素が表示されない場合、式は null を返します。次の式の例では、前述の JSON の例から null が返ります。これは一致する要素がないためです。
>
> ```
> $.customerId
> ```

ネストされた JSON 要素へのアクセス

ネストされた JSON 要素をクエリするには、次の構文を使用します。

```
$.parentElement.element
```

次の式では、前述の JSON の例の `city` 要素をクエリします。

```
$.address.city
```

前述の式では、前述の JSON のレコードから次のように返ります。

```
Anytown
```

以下の構文を使用して、深いレベルのサブ要素をクエリすることができます。

```
$.parentElement.element.subElement
```

次の式では、前述の JSON の例の `street` 要素をクエリします。

```
$.address.streetAddress.street
```

前述の式では、前述の JSON のレコードから次のように返ります。

```
AnyStreet
```

配列へのアクセス

JSON 配列内のデータにアクセスするには、以下の方法があります。

- 配列内のすべての要素を単一の行として取得します。
- 配列内の各要素を別々の行として取得します。

配列内のすべての要素を単一の行として取得する

配列のコンテンツ全体を単一の行としてクエリを実行するには、次の構文を使用します。

```
$.arrayObject[0:]
```

次の式では、このセクションで使用した前述の JSON の例の `orders` 要素をクエリします。配列の内容は、単一行の 1 つの列で返ります。

```
$.orders[0:]
```

前述の式は、このセクションで使用した JSON レコード例から次を返します。

```
[{"orderId":"23284","itemName":"Widget","itemPrice":"33.99"},
{"orderId":"61322","itemName":"Gadget","itemPrice":"22.50"},
{"orderId":"77284","itemName":"Sprocket","itemPrice":"12.00"}]
```

配列内のすべての要素を別々の行として取得する

配列内の各要素を別々の行としてクエリを実行するには、次の構文を使用します。

```
$.arrayObject[0:].element
```

次の式では、前述の JSON の例の `orderId` 要素をクエリし、配列内の各要素は別々の行で返ります。

```
$.orders[0:].orderId
```

前述の例では、前述の JSON レコードから以下のように返ります。その際、各データ項目は別々の行で返ります。

23284
63122
77284

> **Note**
>
> 非配列要素をクエリする式が各配列要素をクエリするスキーマに含まれている場合、配列内の各要素に対して非配列要素が繰り返されます。たとえば、前述の JSON の例のスキーマに、次の式が含まれていると仮定します。
>
> - $.customerName
> - $.orders[0:].orderId
>
> この場合、サンプルの入力ストリーム要素から返されるデータ行は次のようになります。`orderId` 要素それぞれに対して、`name` 要素が繰り返されます。

John Doe	23284
John Doe	63122
John Doe	77284

> **Note**
>
> Amazon Kinesis Data Analytics の配列式には、次の制限が適用されます。
>
> - 配列式でサポートされる参照解除レベルは 1 つのみです。次の式形式はサポートされていません。
>
> ```
> $.arrayObject[0:].element[0:].subElement
> ```
>
> - スキーマにフラット化できる配列は 1 つのみです。複数の配列を参照する場合、配列のすべての要素を含む 1 つの行で返されます。ただし、個々の行として返る各要素には、1 つの配列のみ持つことができます。
>
> 次の形式の要素を含むスキーマは有効です。この形式では、1 つの列として 2 番目の配列の内容が返り、最初の配列の各要素が繰り返されます。
>
> ```
> $.arrayObjectOne[0:].element
> $.arrayObjectTwo[0:]
> ```
>
> 次の形式の要素を含むスキーマは無効です。
>
> ```
> $.arrayObjectOne[0:].element
> $.arrayObjectTwo[0:].element
> ```

その他の考慮事項

JSONPath を操作するには、他にも次のような考慮事項があります。

- アプリケーションスキーマの JSONPath 式に各要素からアクセスされる配列がない場合、処理される JSON レコードごとに単一の行がアプリケーションの入力ストリームに作成されます。
- 配列がフラット化された場合 (つまり、要素が個別の行として返される場合)、null 値となる欠落した要素はすべてアプリケーション内のストリームで作成されます。
- 配列は常に少なくとも 1 つの行にフラット化されます。返される値がない場合 (つまり、配列が空またはクエリされている要素が存在しない)、すべての値が null で 1 つの行が返ります。

  次の式では、前述の JSON の例から null 値を含むレコードが返ります。これは、指定のパスに一致する要素がないためです。

  ```
  $.orders[0:].itemId
  ```

  前述の例では、前述の JSON のサンプルレコードから次のように返ります。

null
null
null

関連トピック

- JSON のご紹介

SQL 入力列へのストリーミングソース要素のマッピング

> **Note**
> 2023 年 9 月 12 日以降、Kinesis Data Analytics for SQL をまだ使用していない場合、Kinesis Data Firehose をソースとして使用して新しいアプリケーションを作成することはできません。詳細については、「制限」を参照してください。

Amazon Kinesis Data Analytics で、標準 SQL を使用して JSON 形式または CSV 形式でストリーミングデータを処理および分析します。

- ストリーミングする CSV データを処理して分析するには、入力ストリームの列に列名とデータ型を割り当てます。入力ストリームから列定義ごとに 1 つの列が順番にアプリケーションでインポートされます。

  アプリケーションの入力ストリームにすべての列を含める必要はありませんが、ソースストリームから列をスキップすることはできません。たとえば、5 つの要素を含む入力ストリームから最初の 3 つの列をインポートすることはできますが、列 1、2、4 のみインポートすることはできません。
- ストリーミング JSON データを処理して分析するには、JSONPath 式を使用して、ストリーミングソースの JSON 要素を入力ストリームの SQL 列にマッピングします。Amazon Kinesis Data Analytics で JSONPath を使用する方法の詳細については、「JSONPath の操作」を参照してください。SQL テーブルの列は、JSON 型からマッピングされたデータ型です。サポートされているデータ型のリストについては、「データ型」を参照してください。JSON データを SQL データに変換する方法については、「JSON データ型から SQL データ型へのマッピング」を参照してください。

入力ストリームの設定方法の詳細については、「アプリケーション入力の設定」を参照してください。

SQL 列への JSON データのマッピング

AWS Management Console または Kinesis Data Analytics API を使用して、JSON 要素を入力列にマッピングできます。

- コンソールを使用して要素を列にマッピングするには、「スキーマエディタの使用」を参照してください。
- Kinesis Data Analytics API を使用して要素を列にマッピングするには、次のセクションを参照してください。

JSON 要素をアプリケーション内の入力ストリームの列にマッピングするには、スキーマで、以下の情報が各列に必要です。

- ソース式: 列のデータの場所を識別する JSONPath 式。
- 列名: SQL クエリでデータの参照に使用する名前。
- データ型: 列の SQL データ型。

API を使用する場合

ストリーミングソースから入力列に要素をマッピングするには、Kinesis Data Analytics API (CreateApplication) のアクションを使用できます。アプリケーション内ストリームを作成するには、SQL で使用するスキーマ化されたバージョンにデータを変換するスキーマを指定します。この CreateApplication アクションでは、単一のストリーミングソースから入力を受信するようにアプリケーションを設定します。JSON 要素または CSV 列を SQL 列にマッピングするには、「RecordColumnオブジェクトSourceSchema」を `RecordColumns` 配列内に作成します。RecordColumn オブジェクトには以下のスキーマがあります。

```
{
    "Mapping": "String",
    "Name": "String",
    "SqlType": "String"
}
```

RecordColumn オブジェクトのフィールドには、次の値があります。

- `Mapping`: JSONPath 式は、入力ストリームレコードのデータの場所を識別します。ソースストリームの入力スキーマで、この値は CSV 形式で表示されません。
- `Name`: アプリケーション内 SQL データストリームの列名。
- `SqlType`: アプリケーション内 SQL データストリームのデータのデータ型。

JSON 入力スキーマの例

次の例では、JSON スキーマの `InputSchema` の値の形式について説明します。

```
"InputSchema": {
    "RecordColumns": [
        {
            "SqlType": "VARCHAR(4)",
            "Name": "TICKER_SYMBOL",
            "Mapping": "$.TICKER_SYMBOL"
        },
        {
            "SqlType": "VARCHAR(16)",
            "Name": "SECTOR",
```

```
            "Mapping": "$.SECTOR"
        },
        {
            "SqlType": "TINYINT",
            "Name": "CHANGE",
            "Mapping": "$.CHANGE"
        },
        {
            "SqlType": "DECIMAL(5,2)",
            "Name": "PRICE",
            "Mapping": "$.PRICE"
        }
    ],
    "RecordFormat": {
        "MappingParameters": {
            "JSONMappingParameters": {
                "RecordRowPath": "$"
            }
        },
        "RecordFormatType": "JSON"
    },
    "RecordEncoding": "UTF-8"
}
```

CSV 入力スキーマの例

次の例では、カンマ区切り (CSV) 形式の `InputSchema` の値の形式について説明します。

```
"InputSchema": {
    "RecordColumns": [
        {
            "SqlType": "VARCHAR(16)",
            "Name": "LastName"
        },
        {
            "SqlType": "VARCHAR(16)",
            "Name": "FirstName"
        },
        {
            "SqlType": "INTEGER",
             "Name": "CustomerId"
        }
    ],
```

```
    "RecordFormat": {
        "MappingParameters": {
            "CSVMappingParameters": {
                "RecordColumnDelimiter": ",",
                "RecordRowDelimiter": "\n"
            }
        },
        "RecordFormatType": "CSV"
    },
    "RecordEncoding": "UTF-8"
}
```

JSON データ型から SQL データ型へのマッピング

JSON データ型は、アプリケーションの入力スキーマに基づき、対応する SQL データ型に変換されます。サポートされている SQL データ型の詳細については、「データ型」を参照してください。Amazon Kinesis Data Analytics では、次のルールに基づき、JSON データ型から SQL データ型に変換されています。

Null リテラル

JSON 入力ストリームの null リテラル (例: `"City":null`) は、送信先のデータ型に関係なく、SQL null に変換されます。

ブールリテラル

JSON 入力ストリームのブールリテラル (例: `"Contacted":true`) は、次のように SQL データに変換されます。

- 数値型 (DECIMAL、INT など): `true` は 1、`false` は 0 に変換されます。
- バイナリ (BINARY または VARBINARY):
  - `true`: 結果には、設定されている最下位ビットとクリアされた残りのビットが表示されます。
  - `false`: 結果にはクリアされたビットがすべて表示されます。

  VARBINARY へ変換すると、1 バイトのデータ値になります。
- BOOLEAN: 対応する SQL BOOLEAN 値に変換されます。
- 文字型 (CHAR または VARCHAR): 対応する文字列値 (`true` または `false`) に変換されます。値は、フィールドの長さに合わせて切り捨てられます。

- 日時型 (DATE、TIME、TIMESTAMP): 変換は失敗し、強制エラーがエラーストリームに書き込まれます。

数値

JSON 入力ストリームの数値リテラル (例: `"CustomerId":67321`) は、次のように SQL データに変換されます。

- 数値 (DECIMAL、INT など): 直接変換されます。変換後の値が、対象のデータ型 (つまり、`123.4` を INT に変換) のサイズまたは精度を超過した場合、変換は失敗し、強制エラーがエラーストリームに書き込まれます。
- バイナリ (BINARY または VARBINARY): 変換は失敗し、強制エラーがエラーストリームに書き込まれます。
- BOOLEAN:
  - `0`: `false` に変換します。
  - 他のすべての数値: `true` に変換します。
- 文字 (CHAR または VARCHAR): 数値の文字列表現に変換します。
- 日時型 (DATE、TIME、TIMESTAMP): 変換は失敗し、強制エラーがエラーストリームに書き込まれます。

String

JSON 入力ストリームの文字列値 (例: `"CustomerName":"John Doe"`) は、SQL データを次のように変換します。

- 数値 (DECIMAL、INT など): Amazon Kinesis Data Analytics は値をターゲットデータ型に変換しようとします。値を変換できない場合、変換が失敗し、強制エラーがエラーストリームに書き込まれます。
- バイナリ (BINARY または VARBINARY): ソース文字列が有効なバイナリリテラル (つまり、`X'3F67A23A'`、偶数 f) の場合、値は対象のデータ型に変換されます。それ以外の場合、変換は失敗し、強制エラーがエラーストリームに書き込まれます。
- BOOLEAN: ソース文字列が `"true"` の場合、`true` に変換されます。この比較では、大文字小文字を区別しません。それ以外の場合は、`false` に変換されます。
- 文字型 (CHAR または VARCHAR): 入力の文字列値に変換します。対象のデータ型よりも値が長い場合は切り捨てられるため、エラーストリームに書き込まれるエラーはありません。

- 日時型 (DATE、TIME または TIMESTAMP): ソース文字列がターゲット値に変換できる形式の場合、値は変換されます。それ以外の場合、変換は失敗し、強制エラーがエラーストリームに書き込まれます。

  有効な日時型形式は以下のとおりです。

  - "1992-02-14"
  - "1992-02-14 18:35:44.0"

配列またはオブジェクト

JSON 入力ストリームの配列またはオブジェクトは、次のように SQL データに変換されます。

- 文字型 (CHAR または VARCHAR): 配列またはオブジェクトのソーステキストに変換します。「配列へのアクセス」を参照してください。
- それ以外のデータ型: 変換は失敗し、強制エラーがエラーストリームに書き込まれます。

JSON 配列の例については、「JSONPath の操作」を参照してください。

関連トピック

- アプリケーション入力の設定
- データ型
- スキーマエディタの使用
- CreateApplication
- RecordColumn
- SourceSchema

ストリーミングデータのスキーマ検出機能の使用

> **Note**
>
> 2023 年 9 月 12 日以降、Kinesis Data Analytics for SQL をまだ使用していない場合、Kinesis Data Firehose をソースとして使用して新しいアプリケーションを作成することはできません。詳細については、「制限」を参照してください。

ストリーミング入力のレコードをアプリケーション内ストリームにマッピングする方法を記述する入力スキーマの指定は、扱いが難しく、エラーが発生しやすいものです。「DiscoverInputSchema」API (検出 API と呼ばれます) を使用してスキーマを推測できます。この API は、ストリーミングソースのレコードのランダムなサンプルを使用してスキーマ (つまり、列名、データ型、データ要素の入力データ内での位置) を推測します。

> **Note**
>
> 検出 API を使用して Amazon S3 に格納されたファイルからスキーマを生成する方法については、「静的データに対するスキーマ検出機能の使用」を参照してください。

指定されたストリーミングソースに対するスキーマを生成するには、コンソールで 検出 API を使用します。コンソールを使用して、スキーマの更新 (例: 列の追加や削除、列名またはデータ型の変更など) を行うことができます。ただし、無効なスキーマを作成しないように、変更は注意深く実行します。

アプリケーション内ストリームのスキーマをファイナライズした後、文字列値と日時値の操作に使用できる関数があります。結果のアプリケーション内ストリームの行で作業をする場合は、このような関数をアプリケーションコードで使用します。詳細については、「例: DateTime 値の変換」を参照してください。

スキーマ検出時に命名する列

スキーマ検出時、 Amazon Kinesis Data Analytics は、ストリーミング入力ソースのオリジナルの列名をできるだけ保持しようとします。ただし、次の場合を除きます。

- ソースストリームの列名は、予約された SQL キーワード (例: `TIMESTAMP`、`USER`、`VALUES`、`YEAR`) です。
- ソースストリーム列名に無効な文字が含まれています。文字、数字、下線文字 (_) のみサポートされています。
- ソースストリーム列名の先頭が数字になっています。
- ソースストリーム列名の文字数が 100 文字を超えています。

列名を変更した場合、変更後のスキーマ列は `COL_` で始まります。名前全体が無効な文字の場合など、元の列名を保持することができない場合があります。このような場合、列の名前は `COL_#` に変更されます。「#」には、列の順序内の場所を示す数値が入ります。

検出が完了すると、コンソールを使用してスキーマを更新し、列の追加または削除、列名、データ型、データサイズの変更を行うことができます。

検出が推奨される列名の例

ソースストリームの列名	検出推奨列名
USER	COL_USER
USER@DOMAIN	COL_USERDOMAIN
@@	COL_0

スキーマ検出の問題

Kinesis Data Analytics が任意のストリーミングソースのスキーマを推測しない場合、どうなるでしょうか。

Kinesis Data Analytics は、CSV や JSON のような、UTF-8 でエンコードされた列形式に対してスキーマを推測します。Kinesis Data Analytics は、(アプリケーションログやカスタムの列および行区切りを持つレコードなどの未加工テキストを含む) UTF-8 でエンコードされたレコードをサポートしています。Kinesis Data Analytics がスキーマを推測しない場合は、コンソールのスキーマエディタ (または API) を使用して、手動でスキーマを定義できます。

データがパターンに合わない場合は (スキーマエディタを使用して指定できます)、スキーマを VARCHAR(N) 型の単一列として定義できます。N はレコードに含まれる最大文字数です。ここから、文字列および日付時刻操作を使用して、データをアプリケーション内ストリームに入力された後に構築できます。例については「例: DateTime 値の変換」を参照してください。

静的データに対するスキーマ検出機能の使用

> **Note**
>
> 2023 年 9 月 12 日以降、Kinesis Data Analytics for SQL をまだ使用していない場合、Kinesis Data Firehose をソースとして使用して新しいアプリケーションを作成することはできません。詳細については、「制限」を参照してください。

スキーマ検出機能は、Amazon S3 バケットに格納されている静的ファイルのストリームやデータからスキーマを生成できます。参照する場合や、ストリーミングデータを利用できない場合に、Kinesis Data Analytics アプリケーションのスキーマを生成するとします。ストリーミングデータまたはリファレンスデータで期待される形式のデータのサンプルを含む静的ファイルで、スキーマ検出機能を使用できます。Kinesis Data Analytics は、Amazon S3 バケットに格納されている JSON ファイルまたは CSV ファイルのサンプルデータに対してスキーマ検出を実行できます。データファイルでスキーマ検出を使用するには、コンソールか、DiscoverInputSchema パラメータを指定した `S3Configuration` API を使用します。

コンソールを使用したスキーマ検出を実行する

コンソールを使用して静的ファイルで検出を実行するには、以下の操作を行います。

1. リファレンスデータオブジェクトを S3 バケットに追加します。
2. Kinesis Data Analytics コンソールで、アプリケーションのメインページの [リファレンスデータを接続] を選択します。
3. リファレンスデータを含む Amazon S3 オブジェクトにアクセスするために、バケット、パス、IAM ロールデータを指定します。
4. [スキーマの検出] を選択します。

コンソールでリファレンスデータを追加し、スキーマを検出する方法の詳細については、「例: Kinesis Data Analytics アプリケーションにリファレンスデータを追加する」を参照してください。

API を使用したスキーマ検出を実行する

API を使用して静的ファイルで検出を実行するには、API に以下の情報を含む `S3Configuration` 構造を指定します。

- `BucketARN`: ファイルを含む Amazon S3 バケットの Amazon リソースネーム (ARN)。Amazon S3 バケット ARN の形式については、「Amazon リソースネーム (ARN) と Amazon サービスの名前空間: Amazon Simple Storage Service (Amazon S3)」を参照してください。
- `RoleARN`: `AmazonS3ReadOnlyAccess` ポリシーを持つ IAM ロールの ARN。ロールにポリシーを追加する方法については、「ロールの修正」を参照してください。
- `FileKey`: オブジェクトのファイル名。

`DiscoverInputSchema` API を使用して Amazon S3 オブジェクトからスキーマを生成するには

1. AWS CLI が設定されていることを確認します。詳細については、「はじめに」セクションの「ステップ 2: AWS Command Line Interface（AWS CLI) を設定する」を参照してください。
2. 次の内容で、`data.csv` という名前のファイルを作成します。

```
year,month,state,producer_type,energy_source,units,consumption
2001,1,AK,TotalElectricPowerIndustry,Coal,ShortTons,47615
2001,1,AK,ElectricGeneratorsElectricUtilities,Coal,ShortTons,16535
2001,1,AK,CombinedHeatandPowerElectricPower,Coal,ShortTons,22890
2001,1,AL,TotalElectricPowerIndustry,Coal,ShortTons,3020601
2001,1,AL,ElectricGeneratorsElectricUtilities,Coal,ShortTons,2987681
```

3. Amazon S3 コンソール (https://console.aws.amazon.com/s3/) にサインインします。
4. Amazon S3 バケットを作成し、作成した `data.csv` ファイルをアップロードします。作成されたバケットの ARN に注意してください。Amazon S3 バケットの作成およびファイルのアップロードの詳細については、「Amazon Simple Storage Service の開始方法」を参照してください。
5. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。`AmazonS3ReadOnlyAccess` ポリシーを使用してロールを作成します。新しいロールの ARN に注意してください。ロールの作成の詳細については、「Amazon Service にアクセス許可を委任するロールの作成」を参照してください。ロールにポリシーを追加する方法については、「ロールの修正」を参照してください。
6. で次の`DiscoverInputSchema`コマンドを実行し AWS CLI、Amazon S3 バケットと IAM ロールの ARNs を置き換えます。

```
$aws kinesisanalytics discover-input-schema --s3-configuration '{ "RoleARN":
 "arn:aws:iam::123456789012:role/service-role/your-IAM-role", "BucketARN":
 "arn:aws:s3:::your-bucket-name", "FileKey": "data.csv" }'
```

7. 応答は次の例のようになります。

```
{
    "InputSchema": {
        "RecordEncoding": "UTF-8",
        "RecordColumns": [
            {
                "SqlType": "INTEGER",
                "Name": "COL_year"
```

```
            },
            {
                "SqlType": "INTEGER",
                "Name": "COL_month"
            },
            {
                "SqlType": "VARCHAR(4)",
                "Name": "state"
            },
            {
                "SqlType": "VARCHAR(64)",
                "Name": "producer_type"
            },
            {
                "SqlType": "VARCHAR(4)",
                "Name": "energy_source"
            },
            {
                "SqlType": "VARCHAR(16)",
                "Name": "units"
            },
            {
                "SqlType": "INTEGER",
                "Name": "consumption"
            }
        ],
        "RecordFormat": {
            "RecordFormatType": "CSV",
            "MappingParameters": {
                "CSVMappingParameters": {
                    "RecordRowDelimiter": "\r\n",
                    "RecordColumnDelimiter": ","
                }
            }
        }
    },
    "RawInputRecords": [
        "year,month,state,producer_type,energy_source,units,consumption
\r\n2001,1,AK,TotalElectricPowerIndustry,Coal,ShortTons,47615\r
\n2001,1,AK,ElectricGeneratorsElectricUtilities,Coal,ShortTons,16535\r
\n2001,1,AK,CombinedHeatandPowerElectricPower,Coal,ShortTons,22890\r
\n2001,1,AL,TotalElectricPowerIndustry,Coal,ShortTons,3020601\r
\n2001,1,AL,ElectricGeneratorsElectricUtilities,Coal,ShortTons,2987681"
    ],
```

```
    "ParsedInputRecords": [
        [
            null,
            null,
            "state",
            "producer_type",
            "energy_source",
            "units",
            null
        ],
        [
            "2001",
            "1",
            "AK",
            "TotalElectricPowerIndustry",
            "Coal",
            "ShortTons",
            "47615"
        ],
        [
            "2001",
            "1",
            "AK",
            "ElectricGeneratorsElectricUtilities",
            "Coal",
            "ShortTons",
            "16535"
        ],
        [
            "2001",
            "1",
            "AK",
            "CombinedHeatandPowerElectricPower",
            "Coal",
            "ShortTons",
            "22890"
        ],
        [
            "2001",
            "1",
            "AL",
            "TotalElectricPowerIndustry",
            "Coal",
            "ShortTons",
```

```
            "3020601"
        ],
        [
            "2001",
            "1",
            "AL",
            "ElectricGeneratorsElectricUtilities",
            "Coal",
            "ShortTons",
            "2987681"
        ]
    ]
}
```

Lambda 関数を使用したデータの事前処理

> **Note**
>
> 2023 年 9 月 12 日以降、Kinesis Data Analytics for SQL をまだ使用していない場合、Kinesis Data Firehose をソースとして使用して新しいアプリケーションを作成することはできません。詳細については、「制限」を参照してください。

ストリーム内のデータに形式変換、変換、エンリッチメント、フィルタリングが必要な場合は、AWS Lambda 関数を使用してデータを前処理できます。アプリケーションの SQL コードが実行される前、またはアプリケーションがデータストリームからスキーマを作成する前に、これを行うことができます。

Lambda 関数によるレコードの事前処理は、次のシナリオで役立ちます。

- 他のフォーマット (KPL や GZIP など) から Kinesis Data Analytics が分析できる形式にレコードを変換します。Kinesis Data Analytics は、現在 JSON データ形式または CSV データ形式をサポートしています。
- 集計検出や異常検出などの操作でよりアクセスしやすい形式にデータを拡張します。たとえば、複数のデータ値が文字列にまとめて格納されている場合は、データを別々の列に拡張できます。
- 外挿やエラー修正などの他の Amazon サービスによるデータの強化。
- レコードのフィールドに複雑な文字列変換を適用します。
- データをクリーンアップするためのデータフィルタリング。

レコードを事前処理するための Lambda 関数の使用

Kinesis Data Analytics アプリケーションを作成するときは、[ソースに接続] ページで Lambda 事前処理を有効にします。

Lambda 関数を使用して Kinesis Data Analytics アプリケーションでレコードを事前処理するには

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/kinesisanalytics で Managed Service for Apache Flink コンソールを開きます。
2. アプリケーションの [ソースに接続] ページの [レコード事前処理] セクションで [有効化 AWS Lambda] を選択します。
3. 既に作成した Lambda 関数を使用するには、[Lambda 関数] ドロップダウンリストで関数を選択します。
4. Lambda 事前処理テンプレートの 1 つから新規の Lambda 関数を作成する場合は、ドロップダウンリストからテンプレートを選択します。次に、[View <template name> in Lambda (Lambda で <テンプレート名> を表示)] を選択して関数を編集します。
5. 新しい Lambda 関数を作成するには、[新規作成] を選択します。Lambda 関数の作成については、AWS Lambda 開発者ガイドの「HelloWorld Lambda 関数を作成してコンソールを探る」を参照してください。
6. 使用する Lambda 関数のバージョンを選択します。最新のバージョンを使用するには、[$LATEST] を選択します。

レコードの事前処理に Lambda 関数を選択または作成すると、アプリケーションの SQL コードがレコードからスキーマを実行したり、アプリケーションがレコードからスキーマを生成したりする前に、レコードが事前処理されます。

Lambda 事前処理アクセス権限

Lambda 事前処理を使用するには、アプリケーションの IAM ロールに次のアクセス許可ポリシーが必要です。

```
    {
      "Sid": "UseLambdaFunction",
      "Effect": "Allow",
      "Action": [
          "lambda:InvokeFunction",
          "lambda:GetFunctionConfiguration"
      ],
```

```
        "Resource": "<FunctionARN>"
    }
```

Lambda 事前処理メトリクス

Amazon CloudWatch を使用して、Lambda 呼び出しの数、処理されたバイト数、成功と失敗の数などをモニタリングすることができます。Lambda の事前処理で出力される CloudWatch メトリクスについては、「Amazon Kinesis Analytics のメトリクス」を参照してください。

Kinesis プロデューサーライブラリ AWS Lambda での の使用

Kinesis Producer Library (KPL) は、小さなユーザーフォーマットレコードを最大 1 MB のレコードに集約して、Amazon Kinesis Data Streams スループットを有効に利用できます。Kinesis Client Library (KCL) for Java は、これらのレコードの集約解除をサポートしています。ただし、ストリームのコンシューマー AWS Lambda として を使用する場合は、特別なモジュールを使用してレコードの集約を解除する必要があります。

必要なプロジェクトコードと手順については、GitHub で AWS Lambda用の Kinesis プロデューサーライブラリの集約解除モジュールについて参照してください。このプロジェクトのコンポーネントを使用して、Java、Node.js、Python AWS Lambda の 内で KPL シリアル化されたデータを処理できます。これらのコンポーネントは、複数言語 KCL アプリケーションの一部として使用することもできます。

データ事前処理イベント入力データモデル / レコードレスポンスモデル

レコードを事前処理するには、Lambda 関数が、必要なイベント入力データおよびレコードレスポンスモデルに準拠している必要があります。

イベント入力データモデル

Kinesis Data Analytics は、Kinesis データストリームまたは Firehose 配信ストリームから継続的にデータを読み取ります。取得したレコードの各バッチが Lambda 関数にどのように渡されたか、サービスが管理しています。関数はレコードのリストを入力として受け取ります。関数内では、リストを繰り返し処理し、ビジネスロジックを適用して、事前処理要件 (データ形式の変換や強化など) を実行します。

事前処理関数への入力モデルは、データが Kinesis データストリームから受信されたか、Firehose 配信ストリームから受信されたかによってわずかに異なります。

ソースが Firehose 配信ストリームの場合、イベント入力データモデルは次のようになります。

Kinesis Data Firehose のリクエストデータモデル

フィールド	説明
`invocationId`	Lambda 呼び出し ID (ランダム GUID)。
`applicationArn`	Kinesis Data Analytics アプリケーションの Amazon リソースネーム (ARN)
`streamArn`	配信ストリーム ARN
レコード	

フィールド	説明
`recordId`	レコード ID (ランダム GUID)
`kinesisFirehoseRecordMetadata`	

フィールド	説明
`approximateArrivalTimestamp`	配信ストリームレコードの概算到着時間

`data`	Base64 でエンコードされたソースレコードのペイロード

次の例は、Firehose 配信ストリームからの入力を示しています。

```
{
   "invocationId":"00540a87-5050-496a-84e4-e7d92bbaf5e2",
   "applicationArn":"arn:aws:kinesisanalytics:us-east-1:12345678911:application/lambda-test",
   "streamArn":"arn:aws:firehose:us-east-1:AAAAAAAAAAAA:deliverystream/lambda-test",
   "records":[
      {
         "recordId":"49572672223665514422805246926656954630972486059535892482",
```

```
            "data":"aGVsbG8gd29ybGQ=",
            "kinesisFirehoseRecordMetadata":{
               "approximateArrivalTimestamp":1520280173
            }
         }
      ]
}
```

ソースが Kinesis データストリームの場合、イベント入力データモデルは次のとおりです。

Kinesis ストリームのリクエストデータモデル

フィールド	説明
`invocationId`	Lambda 呼び出し ID (ランダム GUID)。
`applicationArn`	Kinesis Data Analytics アプリケーション ARN
`streamArn`	配信ストリーム ARN
レコード	

フィールド	説明
`recordId`	Kinesis レコードのシーケンス番号に基づいたレコード ID
`kinesisStreamRecordMetadata`	

フィールド	説明
`sequenceNumber`	Kinesis ストリームレコードからのシーケンス番号
`partitionKey`	Kinesis ストリームレコードからのパーティションキー

フィールド	説明
フィールド	説明
	フィールド: 説明 shardId: Kinesis ストリームレコードからの ShardId approximateArrivalTimestamp: 配信ストリームレコードの概算到着時間
データ	Base64 でエンコードされたソースレコードのペイロード

次の例は、Kinesis データストリームからの入力を示しています。

```
{
  "invocationId": "00540a87-5050-496a-84e4-e7d92bbaf5e2",
  "applicationArn": "arn:aws:kinesisanalytics:us-east-1:12345678911:application/lambda-test",
  "streamArn": "arn:aws:kinesis:us-east-1:AAAAAAAAAAAA:stream/lambda-test",
  "records": [
    {
      "recordId": "49572672223665514422805246926656954630972486059535892482",
      "data": "aGVsbG8gd29ybGQ=",
      "kinesisStreamRecordMetadata":{
            "shardId" :"shardId-000000000003",
            "partitionKey":"7400791606",

 "sequenceNumber":"49572672223665514422805246926656954630972486059535892482",
            "approximateArrivalTimestamp":1520280173
         }
    }
```

```
  ]
}
```

レコードレスポンスモデル

Lambda 関数に送信された Lambda 事前処理関数 (レコード ID 付き) から返されたすべてのレコードは返される必要があります。レコードには次のパラメータが含まれている必要があります。含まれていない場合、Kinesis Data Analytics がレコードを拒否し、データ事前処理を失敗とみなします。レコードのデータペイロード部分は、事前処理要件を達成するために変換できます。

レスポンスデータモデル

レコード

フィールド	説明
`recordId`	レコード ID は呼び出し時に Kinesis Data Analytics から Lambda に渡されます。変換されたレコードには、同じレコード ID が含まれる必要があります。元のレコードの ID と変換されたレコードの ID との不一致は、データ事前処理の失敗として扱われます。
`result`	レコードのデータ変換のステータス。指定できる値は以下のとおりです。 • `Ok`: レコードが正常に変換されました。Kinesis Data Analytics は SQL 処理のレコードを取り込みます。 • `Dropped`: レコードは処理ロジックによって意図的に削除されました。Kinesis Data Analytics は SQL 処理のレコードを削除します。データペイロードフィールドは、`Dropped` レコードではオプションです。 • `ProcessingFailed` ：レコードを変換できませんでした。Kinesis Data Analytics は、Lambda 関数の処理が失敗したとみなし、エラーストリームにエラーを書き込みます。エラーストリームの詳細については、「エラー処理」を参照してください。データペイロードフィールドは、`ProcessingFailed` レコードではオプションです。

フィールド	説明
`data`	base64 エンコード後の変換されたデータペイロード。アプリケーションの取り込みデータ形式が JSON である場合、各データペイロードには複数の JSON ドキュメントを含めることができます。または、アプリケーションの取り込みデータ形式が CSV である場合、それぞれに複数の CSV 行を含めることができます (各行には行の区切り文字が入ります)。Kinesis Data Analytics サービスは、同じデータペイロード内の複数の JSON ドキュメントまたは CSV 行のいずれかを使用して、データを正常に解析して処理します。

次の例は、Lambda 関数からの出力を示しています。

```
{
  "records": [
    {
      "recordId": "49572672223665514422805246926656954630972486059535892482",
      "result": "Ok",
      "data": "SEVMTE8gV09STEQ="
    }
  ]
}
```

一般的なデータ事前処理の失敗

事前処理が失敗する一般的な理由は次のとおりです。

- Lambda 関数に送信されるバッチのレコード (レコード ID 付き) の一部が Kinesis Data Analytics サービスに返されていません。
- レスポンスにレコード ID、ステータス、データペイロードフィールドのいずれかが欠落しています。データペイロードフィールドは、`Dropped` または `ProcessingFailed` レコードの場合はオプションです。
- Lambda 関数のタイムアウトが、データを事前処理するのに十分ではありません。
- Lambda 関数のレスポンスが、 AWS Lambda サービスによって定められたレスポンスの上限を超えています。

データの事前処理が失敗した場合、Kinesis Data Analytics は、成功するまで同じレコードセットで Lambda 呼び出しを再試行し続けます。次の CloudWatch メトリクスを監視して、失敗から洞察を得ることができます。

- Kinesis Data Analytics アプリケーション (`MillisBehindLatest`): アプリケーションの読み取りがストリーミングソースからどれだけ離れているかを示します。
- Kinesis Data Analytics アプリケーション (`InputPreprocessing`) の CloudWatch メトリクス: 統計の中でも、特に成功と失敗の数を示します。詳細については、「Amazon Kinesis Analytics Metrics」を参照してください。
- AWS Lambda 関数 CloudWatch メトリクスとログ。

事前処理用の Lambda 関数の作成

Amazon Kinesis Data Analytics アプリケーションは、アプリケーションに取り込まれる際に、Lambda 関数をレコードの事前処理に使用できます。Kinesis Data Analytics では、データの事前処理用の開始点として使用するため、コンソールで以下のテンプレートが用意されています。

トピック

- Node.js での事前処理 Lambda 関数の作成
- Python での事前処理 Lambda 関数の作成
- Java での事前処理 Lambda 関数の作成
- .NET での事前処理 Lambda 関数の作成

Node.js での事前処理 Lambda 関数の作成

事前処理の Lambda 関数を Node.js で作成するための次のテンプレートが Kinesis Data Analytics コンソールで利用できます。

Lambda の設計図	言語とバージョン	説明
一般的な Kinesis Data Analytics の入力処理	Node.js 6.10	JSON レコードまたは CSV レコードを入力として受け取り、処理ステータスで返す Kinesis Data Analytics レコードのプリプロセッサ。このプロセッサをカスタム変換ロジックの開始点として使用します。

Lambda の設計図	言語とバージョン	説明
圧縮入力処理	Node.js 6.10	圧縮された (圧縮 GZIP または Deflate) JSON レコードまたは CSV レコードを入力として受け取り、圧縮解除されたレコードを処理ステータスで返す Kinesis Data Analytics のレコードプロセッサ。

Python での事前処理 Lambda 関数の作成

事前処理の Lambda 関数を Python で作成するための次のテンプレートがコンソールで利用できます。

Lambda の設計図	言語とバージョン	説明
一般的な Kinesis Analytics の入力処理	Python 2.7	JSON レコードまたは CSV レコードを入力として受け取り、処理ステータスで返す Kinesis Data Analytics レコードのプリプロセッサ。このプロセッサをカスタム変換ロジックの開始点として使用します。
KPL 入力処理	Python 2.7	Kinesis Producer Library (KPL) の JSON レコードまたは CSV レコードの集計を入力として受け取り、処理ステータスと一緒に集約解除されたレコードを返す Kinesis Data Analytics のレコードプロセッサ。

Java での事前処理 Lambda 関数の作成

Java でレコードの事前処理用の Lambda 関数を作成するには、Java イベントクラスを使用します。

次のコードは、Java を使用してレコードを事前処理する Lambda 関数のサンプルを示しています。

```
public class LambdaFunctionHandler implements
        RequestHandler<KinesisAnalyticsStreamsInputPreprocessingEvent,
 KinesisAnalyticsInputPreprocessingResponse> {

    @Override
```

```
    public KinesisAnalyticsInputPreprocessingResponse handleRequest(
            KinesisAnalyticsStreamsInputPreprocessingEvent event, Context context) {
        context.getLogger().log("InvocatonId is : " + event.invocationId);
        context.getLogger().log("StreamArn is : " + event.streamArn);
        context.getLogger().log("ApplicationArn is : " + event.applicationArn);

        List<KinesisAnalyticsInputPreprocessingResponse.Record> records = new
 ArrayList<KinesisAnalyticsInputPreprocessingResponse.Record>();
        KinesisAnalyticsInputPreprocessingResponse response = new
 KinesisAnalyticsInputPreprocessingResponse(records);

        event.records.stream().forEach(record -> {
            context.getLogger().log("recordId is : " + record.recordId);
            context.getLogger().log("record aat is :" +
 record.kinesisStreamRecordMetadata.approximateArrivalTimestamp);
             // Add your record.data pre-processing logic here.

            // response.records.add(new Record(record.recordId,
 KinesisAnalyticsInputPreprocessingResult.Ok, <preprocessedrecordData>));
        });
        return response;
    }

}
```

.NET での事前処理 Lambda 関数の作成

.NET でレコードの事前処理用の Lambda 関数を作成するには、.NET イベントクラスを使用します。

次のコードは、C# を使用してレコードを前処理する Lambda 関数のサンプルを示しています。

```
public class Function
    {
        public KinesisAnalyticsInputPreprocessingResponse
 FunctionHandler(KinesisAnalyticsStreamsInputPreprocessingEvent evnt, ILambdaContext
 context)
        {
            context.Logger.LogLine($"InvocationId: {evnt.InvocationId}");
            context.Logger.LogLine($"StreamArn: {evnt.StreamArn}");
            context.Logger.LogLine($"ApplicationArn: {evnt.ApplicationArn}");

            var response = new KinesisAnalyticsInputPreprocessingResponse
            {
```

```
                Records = new List<KinesisAnalyticsInputPreprocessingResponse.Record>()
            };

            foreach (var record in evnt.Records)
            {
                context.Logger.LogLine($"\tRecordId: {record.RecordId}");
                context.Logger.LogLine($"\tShardId: {record.RecordMetadata.ShardId}");
                context.Logger.LogLine($"\tPartitionKey:
{record.RecordMetadata.PartitionKey}");
                context.Logger.LogLine($"\tRecord ApproximateArrivalTime:
{record.RecordMetadata.ApproximateArrivalTimestamp}");
                context.Logger.LogLine($"\tData: {record.DecodeData()}");

                // Add your record preprocessig logic here.

                var preprocessedRecord = new
KinesisAnalyticsInputPreprocessingResponse.Record
                {
                    RecordId = record.RecordId,
                    Result = KinesisAnalyticsInputPreprocessingResponse.OK
                };
                preprocessedRecord.EncodeData(record.DecodeData().ToUpperInvariant());
                response.Records.Add(preprocessedRecord);
            }
            return response;
        }
    }
```

事前処理および宛先の Lambda 関数を .NET で作成する場合の詳細については、「Amazon.Lambda.KinesisAnalyticsEvents」を参照してください。

スループットの増加に合わせた入力ストリームの並列処理

> **Note**
>
> 2023 年 9 月 12 日以降、Kinesis Data Analytics for SQL をまだ使用していない場合、Kinesis Data Firehose をソースとして使用して新しいアプリケーションを作成することはできません。詳細については、「制限」を参照してください。

Amazon Kinesis Data Analytics アプリケーションでは、アプリケーション内入力ストリームのスループットを超えるアプリケーションをスケーリングするために、複数のアプリケーション内入力

ストリームをサポートできます。アプリケーション内入力ストリームの詳細については、「Amazon Kinesis Data Analytics for SQL Applications: 仕組み」を参照してください。

ほとんどの場合、Amazon Kinesis Data Analytics では、アプリケーションにフィードされる Kinesis ストリームまたは Firehose ソースストリームの容量を処理できるように、アプリケーションがスケールされます。ただし、ソースストリームのスループットが、単一のアプリケーション内入力ストリームのスループットを超える場合は、アプリケーションで使用されるアプリケーション内入力ストリームの数を明示的に増やすことができます。そのためには、`InputParallelism` パラメータを使用します。

`InputParallelism` パラメータが 1 以上の場合、Amazon Kinesis Data Analytics は、アプリケーション内ストリーム間のソースストリームのパーティションを均等に分割します。たとえば、ソースストリームに 50 シャードあり、`InputParallelism` を 2 に設定した場合、アプリケーション内入力ストリームはそれぞれ、25 のソースストリームのシャードから入力を受け取ります。

アプリケーション内ストリームの数を増やす場合は、アプリケーションから、各ストリームのデータに明示的にアクセスする必要があります。コードで複数のアプリケーション内ストリームにアクセスする方法については、「Amazon Kinesis Data Analytics アプリケーションでの別のアプリケーション内ストリームへのアクセス」を参照してください。

Kinesis Data Streams と Firehose ストリームのシャードは、どちらも同様にアプリケーション内ストリーム間で分割されますが、アプリケーションに認識される形式は異なります。

- Kinesis データストリームのレコードには、`shard_id` フィールドが含まれており、レコードのソースシャードを識別できます。
- Firehose 配信ストリームのレコードには、レコードのソースシャードまたはパーティションを識別するフィールドは含まれていません。これは、Firehose がこの情報をアプリケーションから抽象化するためです。

アプリケーション内入力ストリームの数の増加を評価する

ほとんどの場合、入力ストリームの複雑性やデータサイズに応じて、1 つのアプリケーション内入力ストリームで、1 つのソースストリームのスループットを処理することができます。アプリケーション内入力ストリームの数を増やす必要の有無を判断するには、Amazon CloudWatch の `InputBytes` および `MillisBehindLatest` メトリクスをモニタリングします。

`InputBytes` メトリクスが 100 MB/秒より大きい場合 (または、このレートより大きくなることが予想される場合)、`MillisBehindLatest` が増えたり、アプリケーションの問題の影響が大きくなっ

たりする可能性があります。この問題に対応するため、アプリケーションに対して次の言語を選択することをお勧めします。

- アプリケーションのスケーリングニーズが 100 MB/秒を超える場合は、複数のストリームと Kinesis Data Analytics for SQL アプリケーションを使用します。
- 1 つのストリームとアプリケーションを使用する場合は、Kinesis Data Analytics for Java Applications を使用します。

`MillisBehindLatest` メトリクスに次のいずれかの特性がある場合は、アプリケーションの `InputParallelism` 設定を増やす必要があります。

- `MillisBehindLatest` メトリクスが増加しつつあります。これは、アプリケーションにおいて、ストリーム内の最新データが遅延していることを意味します。
- `MillisBehindLatest` メトリクスは一貫して 1,000 (1 秒あたり) を超えています。

以下が真の場合は、アプリケーションの `InputParallelism` 設定を増やす必要はありません。

- `MillisBehindLatest` メトリクスが減少しつつあります。これは、アプリケーションにおいて、ストリーム内の最新データの遅れを取り戻していることを意味します。
- `MillisBehindLatest` メトリクスは一貫して 1,000 (1 秒あたり) を下回っています。

CloudWatch の使用方法の詳細については、「CloudWatch ユーザーガイド」を参照してください。

複数のアプリケーション内入力ストリームの実装

「CreateApplication」を使用してアプリケーションを作成する際、アプリケーション内入力ストリームの数を設定できます。この数は、「UpdateApplication」を使用してアプリケーションを作成した後に設定します。

> **Note**
>
> `InputParallelism` 設定は、Amazon Kinesis Data Analytics API または AWS CLIを使ってのみ設定できます。を使用してこの設定を設定することはできません AWS Management Console。のセットアップについては AWS CLI、「」を参照してくださいステップ 2: AWS Command Line Interface（AWS CLI) を設定する。

新しいアプリケーションの入力ストリームカウントの設定

次の例では、API アクション (CreateApplication) を使用して、新しいアプリケーションの入力ストリームカウントを 2 に設定する方法について解説します。

CreateApplication の詳細については、「CreateApplication」を参照してください。

```
{
   "ApplicationCode": "<The SQL code the new application will run on the input
 stream>",
   "ApplicationDescription": "<A friendly description for the new application>",
   "ApplicationName": "<The name for the new application>",
   "Inputs": [
    {
      "InputId": "ID for the new input stream",
      "InputParallelism": {
        "Count": 2
    }],
   "Outputs": [ ... ],
 }]
}
```

既存アプリケーションの入力ストリームカウントの設定

次の例では、API アクション (UpdateApplication) を使用して、既存アプリケーションの入力ストリームカウントを 2 に設定する方法について解説します。

Update_Application の詳細については、「UpdateApplication」を参照してください。

```
{
   "InputUpdates": [
      {
         "InputId": "yourInputId",
         "InputParallelismUpdate": {
            "CountUpdate": 2
         }
      }
   ],
}
```

Amazon Kinesis Data Analytics アプリケーションでの別のアプリケーション内ストリームへのアクセス

複数のアプリケーション内入力ストリームをアプリケーションで使用するには、別のストリームから明示的に選択する必要があります。次のコード例では、入門チュートリアルで作成した複数の入力ストリームをアプリケーションでクエリを行う方法について説明します。

次の例では、in_application_stream001 という 1 つのアプリケーション内ストリームに結合される前に、まず COUNT を使用して各ソースストリームが集約されます。事前にソースストリームを集約すると、結合されたアプリケーション内ストリームで、負荷をかけ過ぎることなく複数のストリームからのトラフィックを処理しやすくなります。

> **Note**
>
> この例を実行して、両方のアプリケーション内入力ストリームから結果を得るには、ソースストリームのシャード数とアプリケーションの InputParallelism パラメータを両方とも更新します。

```
CREATE OR REPLACE STREAM in_application_stream_001 (
    ticker VARCHAR(64),
    ticker_count INTEGER
);

CREATE OR REPLACE PUMP pump001 AS
INSERT INTO in_application_stream_001
SELECT STREAM ticker_symbol, COUNT(ticker_symbol)
FROM source_sql_stream_001
GROUP BY STEP(source_sql_stream_001.rowtime BY INTERVAL '60' SECOND),
    ticker_symbol;

CREATE OR REPLACE PUMP pump002 AS
INSERT INTO in_application_stream_001
SELECT STREAM ticker_symbol, COUNT(ticker_symbol)
FROM source_sql_stream_002
GROUP BY STEP(source_sql_stream_002.rowtime BY INTERVAL '60' SECOND),
    ticker_symbol;
```

前述のコード例では、以下のような出力を in_application_stream001 に生成します。

Filter by column name

ROWTIME	TICKER	TICKER_COUNT
2017-05-17 22:05:00.0	QAZ	15
2017-05-17 22:06:00.0	SAC	16
2017-05-17 22:06:00.0	PLM	10
2017-05-17 22:06:00.0	AMZN	15

追加の考慮事項

複数の入力ストリームを使用する場合は、以下の点に注意してください。

- アプリケーション内入力ストリームの最大数は 64 です。
- アプリケーション内入力ストリームは、アプリケーションの入力ストリームのシャード間で均等に分散されます。
- アプリケーション内ストリームの追加により向上するパフォーマンスは、直線的にスケールしません。つまり、アプリケーション内ストリームの数を 2 倍にしても、スループットは 2 倍になりません。一般的な行サイズを使用すると、アプリケーション内ストリームはそれぞれ、1 秒あたり約 5,000 ~ 15,000 行のスループットを達成します。アプリケーション内ストリームカウントを 10 に増やすことによって、1 秒あたり 20,000 ~ 30,000 行のスループットを達成できます。スループット速度は、入力ストリームのフィールドのカウント、データ型、サイズによって異なります。
- 一部の集計関数 (AVG) では、別のシャードに分割されている入力ストリームに適用されると、予期しない結果が生成される場合があります。集計ストリームに結合する前に、個々のシャードで集計オペレーションを実行する必要があるため、レコードが多く含まれているストリームに関係なく加重される場合があります。
- 入力ストリームの数を増やした後にアプリケーションのパフォーマンスが低下し続ける (高い `MillisBehindLatest` メトリクスにより反映される) 場合は、Kinesis 処理ユニット (KPU) の上限に達している可能性があります。詳細については、「アプリケーションを自動的にスケーリングしてスループットを向上させる」を参照してください。

アプリケーションコード

アプリケーションコードは、入力を処理し出力を生成する一連の SQL ステートメントです。この SQL ステートメントはアプリケーション内ストリームおよびリファレンステーブルで動作します。

詳細については、「Amazon Kinesis Data Analytics for SQL Applications: 仕組み」を参照してください。

Kinesis Data Analytics でサポートされている SQL 言語要素の詳細については、「Amazon Kinesis Data Analytics SQL Reference」を参照してください。

リレーショナルデータベースでは、レコードを追加する INSERT ステートメントと、データをクエリする SELECT ステートメントを使用して、テーブルで作業を行います。Amazon Kinesis Data Analytics では、ストリームを操作します。これらのストリームをクエリする SQL ステートメントを作成できます。1 つのアプリケーション内ストリームをクエリした結果は、常に別のアプリケーション内ストリームに送信されます。複雑な分析を実行する場合は、分析の中間結果を保持する複数のアプリケーション内ストリームを作成する場合があります。最終的には、アプリケーション出力を設定して、(1 つまたは複数のアプリケーション内ストリームからの) 最終分析を外部宛先で永続化します。要約すると、アプリケーションコードを作成する一般的なパターンは以下のとおりです。

- SELECT ステートメントは、常に INSERT ステートメントのコンテキストで使用されます。つまり、行を選択すると、結果を別のアプリケーション内ストリームに挿入します。
- INSERT ステートメントは、常にポンプのコンテキストで使用されます。つまり、ポンプを使用してアプリケーション内ストリームに書き込みます。

次のアプリケーションコードの例では、あるアプリケーション内 (SOURCE_SQL_STREAM_001) ストリームからレコードを読み取り、別のアプリケーション内ストリーム (DESTINATION_SQL_STREAM) に書き込みます。次のように、ポンプを使用してレコードをアプリケーション内ストリームに挿入できます。

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (ticker_symbol VARCHAR(4),
                                                   change DOUBLE,
                                                   price DOUBLE);
-- Create a pump and insert into output stream.
CREATE OR REPLACE PUMP "STREAM_PUMP" AS

  INSERT INTO "DESTINATION_SQL_STREAM"
    SELECT STREAM ticker_symbol, change,price
    FROM   "SOURCE_SQL_STREAM_001";
```

> **Note**
>
> ストリーム名および列名に指定する識別子は標準 SQL の命名規則に従います。たとえば、識別子を引用符で囲むと、識別子で大文字と小文字が区別されるようになります。囲まない場合は、識別子はデフォルトで大文字になります。識別子の詳細については、「Amazon Managed Service for Apache Flink SQL リファレンス」の「Identifiers」を参照してください。

アプリケーションコードは、多数の SQL ステートメントで構成できます。以下に例を示します。

- 1 つの SQL ステートメントの結果を次の SQL ステートメントにフィードするシーケンシャルな SQL クエリを作成できます。
- また、相互に独立して実行される SQL クエリを作成することもできます。たとえば、同じアプリケーション内ストリームをクエリするが、異なるアプリケーション内ストリームに出力を送信する、2 つの SQL ステートメントを作成できます。その後、新しく作成されたアプリケーション内ストリームを個別にクエリできます。

中間結果を保存するアプリケーション内ストリームを作成できます。ポンプを使用してアプリケーション内ストリームにデータを挿入します。詳細については、「アプリケーション内ストリームとポンプ」を参照してください。

アプリケーション内リファレンステーブルを追加する場合は、アプリケーション内ストリームとリファレンステーブルのデータを結合する SQL を作成できます。詳細については、「例: Kinesis Data Analytics アプリケーションにリファレンスデータを追加する」を参照してください。

アプリケーションの出力設定に従って、Amazon Kinesis Data Analytics はアプリケーションの出力設定に従って特定のアプリケーション内ストリームからのデータを外部宛先に書き込みます。アプリケーションコードが、出力設定で指定されたアプリケーション内ストリームに書き込むことを確認してください。

詳細については、以下の各トピックを参照してください。

- ストリーミング SQL の概念
- Amazon Kinesis Data Analytics SQL Reference

アプリケーション出力の設定

アプリケーションコードで、SQL ステートメントの出力を 1 つ以上のアプリケーション内ストリームに書き込みます。必要に応じて、出力設定をアプリケーションに追加できます。 は、アプリケーション内ストリームに書き込まれたすべてを Amazon Kinesis データストリーム、Firehose 配信ストリーム、 AWS Lambda 関数などの外部宛先に保持します。

アプリケーション出力の永続化に使用できる外部宛先の数には制限があります。詳細については、「制限」を参照してください。

> **Note**
>
> エラーを精査するためにアプリケーション内エラーストリームのデータを永続化する外部宛先は、1 つにすることをお勧めします。

これらの出力設定で、以下を指定します。

- アプリケーション内ストリーム名 – 外部宛先で永続化するストリームです。

  Kinesis Data Analytics は、出力設定で指定されたアプリケーション内ストリームを検索します。(ストリーム名は大文字と小文字が区別され、正確に一致する必要があることに注意してください)。アプリケーションコードがこのアプリケーション内ストリームを作成することを確認してください。
- 外部宛先 – データを Kinesis データストリーム、Firehose データ配信ストリーム、または Lambda 関数に永続化できます。ストリームまたは関数の Amazon リソースネーム (ARN) を指定します。また、Amazon Kinesis Analytics がユーザーに代わってストリームまたは Lambda 関数に書き込むために引き受けることができる IAM ロールも指定します。外部宛先に書き込むときに Kinesis Data Analytics が使用するレコード形式 (JSON, CSV) も記述します。

Kinesis Data Analytics でストリーミングまたは Lambda 宛先に書き込むことができない場合、サービスは無限に試行を続けます。これはバックプレッシャーを生み出し、アプリケーションに遅延が生じます。この問題が解決しない場合、アプリケーションは最終的に新しいデータの処理を停止します。Kinesis Data Analytics Metrics をモニタリングし、障害のアラームを設定できます。メトリクスとアラームの詳細については、Amazon CloudWatch メトリクスを使用するとAmazon CloudWatch アラームを作成するを参照してください。

AWS Management Consoleを使用してアプリケーション出力を設定できます。コンソールは API コールを実行して設定を保存します。

を使用した出力の作成 AWS CLI

このセクションでは、`CreateApplication` または `AddApplicationOutput` オペレーションのリクエストボディの `Outputs` セクションを作成する方法について説明します。

Kinesis ストリーム出力を作成する

次の JSON フラグメントは、Amazon Kinesis データストリームの宛先を作成する `CreateApplication` リクエストボディの `Outputs` セクションを示しています。

```
"Outputs": [
   {
       "DestinationSchema": {
           "RecordFormatType": "string"
       },
       "KinesisStreamsOutput": {
           "ResourceARN": "string",
           "RoleARN": "string"
       },
       "Name": "string"
   }

]
```

Firehose 配信ストリーム出力を作成する

次の JSON フラグメントは、Amazon Data Firehose 配信ストリームの宛先を作成する `CreateApplication` リクエスト本文の `Outputs` セクションを示しています。

```
"Outputs": [
   {
       "DestinationSchema": {
           "RecordFormatType": "string"
       },
       "KinesisFirehoseOutput": {
           "ResourceARN": "string",
           "RoleARN": "string"
       },
       "Name": "string"
   }
]
```

Lambda 関数出力を作成する

次の JSON フラグメントは、 AWS Lambda 関数の送信先を作成するためのCreateApplicationリクエスト本文の Outputsセクションを示しています。

```
"Outputs": [
   {
       "DestinationSchema": {
           "RecordFormatType": "string"
       },
       "LambdaOutput": {
           "ResourceARN": "string",
           "RoleARN": "string"
       },
       "Name": "string"
   }
]
```

出力としての Lambda 関数の使用

を送信先 AWS Lambda として使用すると、最終送信先に送信する前に SQL 結果の後処理をより簡単に実行できます。一般的な後処理タスクには次のものがあります。

- 複数の行を 1 つのレコードに集約する
- 現在の結果と過去の結果を組み合わせて、遅れて届くデータに対処する
- 情報のタイプに基づいて異なる送信先に配信する
- レコード形式の変換 (Protobuf への変換など)
- 文字列操作または変換
- 分析処理後のデータの強化
- 地理空間ユースケースのカスタム処理
- データ暗号化

Lambda 関数は、次のようなさまざまな AWS サービスやその他の送信先に分析情報を配信できます。

- Amazon Simple Storage Service (Amazon S3)
- カスタム API

- Amazon DynamoDB
- Amazon Aurora
- Amazon Redshift
- Amazon Simple Notification Service (Amazon SNS)
- Amazon Simple Queue Service (Amazon SQS)
- Amazon CloudWatch

Lambda アプリケーションの作成の詳細については、「AWS Lambdaのご利用開始にあたって」を参照してください。

トピック

- 出力許可としての Lambda
- 出力メトリクスとしての Lambda
- 出力イベント入力データモデルおよびレコードレスポンスモデルとしての Lambda
- Lambda 出力呼び出しの頻度
- 出力として使用するための Lambda 関数の追加
- 出力エラーとしてよく見られる Lambda
- アプリケーションの送信先の Lambda 関数の作成

出力許可としての Lambda

出力として Lambda を使用するには、アプリケーションの Lambda 出力 IAM ロールに次のアクセス許可ポリシーが必要です。

```
{
   "Sid": "UseLambdaFunction",
   "Effect": "Allow",
   "Action": [
       "lambda:InvokeFunction",
       "lambda:GetFunctionConfiguration"
   ],
   "Resource": "FunctionARN"
}
```

出力メトリクスとしての Lambda

Amazon CloudWatch を使用して、送信バイト数、成功および失敗などをモニタリングします。出力として Lambda を使用する Kinesis Data Analytics によって出力される CloudWatch メトリクスについては、「Amazon Kinesis Analytics Metrics」を参照してください。

出力イベント入力データモデルおよびレコードレスポンスモデルとしての Lambda

Kinesis Data Analytics 出力レコードを送信する場合、Lambda 関数は、必要なイベント入力データおよびレコードレスポンスモデルに準拠している必要があります。

イベント入力データモデル

Kinesis Data Analytics は、次のリクエストモデルの出力関数として、アプリケーションから Lambda に出力レコードを継続的に送信します。関数内では、リストを繰り返し処理し、ビジネスロジックを適用して、出力要件 (最終的な送信先に送信する前のデータ変換など) を実行します。

フィールド	説明
`invocationId`	Lambda 呼び出し ID (ランダム GUID)。
`applicationArn`	Kinesis Data Analytics アプリケーションの Amazon リソースネーム (ARN)。
レコード	

フィールド	説明
`recordId`	レコード ID (ランダム GUID)
`lambdaDeliveryRecordMetadata`	

フィールド	説明
`retryHir`	配信再試行回数

データ	Base64 でエンコードされた出力レコードのペイロード

> **Note**
>
> `retryHint` は配信失敗ごとに増加する値です。この値は永続的に保持されず、アプリケーションが中断された場合にリセットされます。

レコードレスポンスモデル

出力関数として `Ok` に (レコード ID と共に) 送信される各レコードは、 または `DeliveryFailed` のどちらかで確認される必要があり、次のパラメータを含める必要があります。それ以外の場合、Kinesis Data Analytics はそれらを配信失敗として扱います。

レコード

フィールド	説明
`recordId`	レコード ID は呼び出し時に Kinesis Data Analytics から Lambda に渡されます。元のレコードの ID と確認されたレコードの ID との不一致は、配信失敗として扱われます。
`result`	レコード配信のステータス。以下の値を指定できます。 • `Ok`: レコードは正常に変換され、最終的な送信先に送信されました。Kinesis Data Analytics は SQL 処理のレコードを取り込みます。 • `DeliveryFailed` ：レコードは Lambda によって出力関数として最終的な送信先に正常に配信されませんでした。Kinesis Data Analytics は失敗したレコードの出力関数としての Lambda への送信を継続的に再試行します。

Lambda 出力呼び出しの頻度

Kinesis Data Analytics アプリケーションは、出力レコードをバッファして AWS Lambda 宛先関数を頻繁に呼び出します。

- タンブリングウィンドウとしてデータ分析アプリケーション内の送信先アプリケーション内ストリームにレコードが出力された場合、送信 AWS Lambda 先関数はタンブリングウィンドウトリ

ガーごとに呼び出されます。例えば、タンブリングウィンドウを 60 秒に設定してレコードを宛先のアプリケーション内ストリームに出力すると、Lambda 関数は、60 秒ごとに 1 回呼び出されます。

- アプリケーション内で連続するクエリまたはスライディングウィンドウとしてレコードがアプリケーション内ストリームに出力される場合、Lambda 宛先関数は約 1 秒に 1 回呼び出されます。

> **Note**
>
> Lambda 関数あたりの呼び出しリクエストのペイロードサイズの制限が適用されます。これらの制限を超えると、出力レコードが分割され、複数の Lambda 関数呼び出しに分けて送信されます。

出力として使用するための Lambda 関数の追加

次の手順では、Kinesis Data Analytics アプリケーションの出力として Lambda 関数を追加する方法を示しています。

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/kinesisanalytics で Managed Service for Apache Flink コンソールを開きます。
2. リストからアプリケーションを選択し、[Application details] を選択します。
3. [宛先] セクションで、[Connect new destination] を選択します。
4. [宛先] 項目に、[AWS Lambda 関数] を選択します。
5. [AWS Lambdaにレコードを配信] セクションで、既存の Lambda 関数とバージョンを選択するか、[新規作成] を選択します。
6. 新しい Lambda 関数を作成する場合は、次の操作を行います。
   a. 提供されているいずれかのテンプレートのいずれかを選択します。詳細については、アプリケーションの送信先の Lambda 関数の作成。
   b. [関数の作成] ページが新しいブラウザタブで開きます。[Name (名前)] ボックスで、関数にわかりやすい名前を付けます (例: **`myLambdaFunction`**)。
   c. アプリケーションの後処理機能のテンプレートを更新します。Lambda 関数作成の詳細については、AWS Lambda 開発者ガイドの入門ガイドを参照してください。
   d. Kinesis Data Analytics コンソールの [Lambda 関数] リストで、先ほど作成した Lambda 関数を選択します。Lambda 関数のバージョンは [$最新] を選択します。

7. [In-application stream] セクションで、[Choose an existing in-application stream] を選択します。[In-application stream name] に、アプリケーションの出力ストリームを選択します。選択した出力ストリームからの結果は、Lambda 出力関数に送信されます。
8. 残りのフォームはデフォルト値のままにして、[Save and continue] を選択します。

アプリケーションはアプリケーション内ストリームから Lambda 関数にレコードを送信するようになりました。Amazon CloudWatch コンソールのデフォルトテンプレートの結果を確認できます。`AWS/KinesisAnalytics/LambdaDelivery.OkRecords` メトリクスをモニタリングして、Lambda 関数に配信されるレコードの数を確認します。

出力エラーとしてよく見られる Lambda

以下は、Lambda 関数への配信が失敗する可能性のある一般的な理由です。

- Lambda 関数に送信されるバッチのレコード (レコード ID 付き) の一部が Kinesis Data Analytics サービスに返されていません。
- レスポンスにレコード ID、またはステータスフィールドのいずれかが欠落しています。
- Lambda 関数のタイムアウトが Lambda 関数内のビジネスロジックを達成するのに十分ではありません。
- Lambda 関数内のビジネスロジックは、すべてのエラーをキャッチしないため、処理されない例外のためにタイムアウトとバックプレッシャーが生じます。これらのメッセージは、「ポイズンピル」と呼ばれることが少なくありません。

データ配信が失敗した場合、Kinesis Data Analytics は、成功するまで同じレコードセットで Lambda 呼び出しを再試行し続けます。次の CloudWatch メトリクスを監視して、失敗から情報を得ることができます。

- Kinesis Data Analytics アプリケーションの Output CloudWatch メトリクスとしてのLambda : 統計の中でも、特に成功と失敗の数を示します。詳細については、「Amazon Kinesis Analytics Metrics」を参照してください。
- AWS Lambda 関数 CloudWatch メトリクスとログ。

アプリケーションの送信先の Lambda 関数の作成

Kinesis Data Analytics アプリケーションは、 AWS Lambda 関数を出力として使用できます。Kinesis Data Analytics はアプリケーションの送信先として使用する Lambda 関数を作成するテ

ンプレートを提供します。これらのテンプレートは、アプリケーションからの後処理出力の開始点として使用します。

トピック

- Node.js での Lambda 関数の送信先の作成
- Python での Lambda 関数の送信先の作成
- Java での Lambda 関数の送信先の作成
- .NET での Lambda 関数の送信先の作成

Node.js での Lambda 関数の送信先の作成

Lambda 関数の宛先を Node.js で作成するための次のテンプレートがコンソールで利用できます。

出力ブループリントとしての Lambda	言語とバージョン	説明
`kinesis-analytics-output`	Node.js 12.x	Kinesis Data Analytics アプリケーションからカスタム送信先に出力レコードを配信します。

Python での Lambda 関数の送信先の作成

Lambda 関数の宛先を Python で作成するための次のテンプレートがコンソールで利用できます。

出力ブループリントとしての Lambda	言語とバージョン	説明
`kinesis-analytics-output-sns`	Python 2.7	Kinesis Data Analytics アプリケーションから Amazon SNS に出力レコードを配信します。
`kinesis-analytics-output-ddb`	Python 2.7	Kinesis Data Analytics アプリケーションから Amazon

出力ブループリントとしての Lambda	言語とバージョン	説明
		DynamoDB に出力レコードを配信します。

Java での Lambda 関数の送信先の作成

Java で Lambda 関数の送信先を作成するには、Java イベントクラスを使用します。

次のコードは、Java を使用したサンプルの送信先 Lambda 関数を示しています。

```
public class LambdaFunctionHandler
        implements RequestHandler<KinesisAnalyticsOutputDeliveryEvent,
 KinesisAnalyticsOutputDeliveryResponse> {

    @Override
    public KinesisAnalyticsOutputDeliveryResponse
 handleRequest(KinesisAnalyticsOutputDeliveryEvent event,
            Context context) {
        context.getLogger().log("InvocatonId is : " + event.invocationId);
        context.getLogger().log("ApplicationArn is : " + event.applicationArn);

        List<KinesisAnalyticsOutputDeliveryResponse.Record> records = new
 ArrayList<KinesisAnalyticsOutputDeliveryResponse.Record>();
        KinesisAnalyticsOutputDeliveryResponse response = new
 KinesisAnalyticsOutputDeliveryResponse(records);

        event.records.stream().forEach(record -> {
            context.getLogger().log("recordId is : " + record.recordId);
            context.getLogger().log("record retryHint is :" +
 record.lambdaDeliveryRecordMetadata.retryHint);
            // Add logic here to transform and send the record to final destination of
 your choice.
            response.records.add(new Record(record.recordId,
 KinesisAnalyticsOutputDeliveryResponse.Result.Ok));
        });
        return response;
    }

}
```

.NET での Lambda 関数の送信先の作成

.NET で Lambda 関数の送信先を作成するには、.NET イベントクラスを使用します。

次のコードは、C# を使用したサンプルの送信先 Lambda 関数を示しています。

```
public class Function
    {
        public KinesisAnalyticsOutputDeliveryResponse
 FunctionHandler(KinesisAnalyticsOutputDeliveryEvent evnt, ILambdaContext context)
        {
            context.Logger.LogLine($"InvocationId: {evnt.InvocationId}");
            context.Logger.LogLine($"ApplicationArn: {evnt.ApplicationArn}");

            var response = new KinesisAnalyticsOutputDeliveryResponse
            {
                Records = new List<KinesisAnalyticsOutputDeliveryResponse.Record>()
            };

            foreach (var record in evnt.Records)
            {
                context.Logger.LogLine($"\tRecordId: {record.RecordId}");
                context.Logger.LogLine($"\tRetryHint:
 {record.RecordMetadata.RetryHint}");
                context.Logger.LogLine($"\tData: {record.DecodeData()}");

                // Add logic here to send to the record to final destination of your
 choice.

                var deliveredRecord = new KinesisAnalyticsOutputDeliveryResponse.Record
                {
                    RecordId = record.RecordId,
                    Result = KinesisAnalyticsOutputDeliveryResponse.OK
                };
                response.Records.Add(deliveredRecord);
            }
            return response;
        }
    }
```

事前処理および宛先の Lambda 関数を .NET で作成する場合の詳細については、「Amazon.Lambda.KinesisAnalyticsEvents」を参照してください。

アプリケーション出力を外部宛先で永続化する配信モデル

Amazon Kinesis Data Analytics は、設定された宛先へのアプリケーション出力に「1 回以上」の配信モデルを使用します。アプリケーションの実行時に、Kinesis Data Analytics は内部チェックポイントを取ります。このチェックポイントは、出力レコードが宛先に配信されデータ損失がない場合のポイントを時間で示すものです。サービスでは必要に応じてチェックポイントを使用し、アプリケーション出力が少なくとも 1 回、設定された宛先に配信されたことを確認します。

通常の状況では、アプリケーションは受信データを継続的に処理します。Kinesis Data Analytics は、Kinesis データストリームや Firehose 配信ストリームなど、設定された宛先に出力を書き込みます。ただし、次の例に示すように、アプリケーションはときどき中断される可能性があります。

- アプリケーションを停止して、後で再起動する場合。
- 設定された宛先に Kinesis Data Analytics がアプリケーション出力を書き込むために必要な IAM ロールを削除した場合。IAM ロールがない場合、Kinesis Data Analytics にはユーザーの代わりに外部宛先に書き込むアクセス権限がありません。
- ネットワークの停止またはその他の内部サービスの障害により、一時的にアプリケーションが実行を停止した場合。

アプリケーションが再起動すると、Kinesis Data Analytics は、障害の発生時またはその前の時点からの出力の処理および書き込みを続けます。これにより、設定された宛先にアプリケーション出力が確実に配信されます。

同じアプリケーション内ストリームから複数の宛先を設定したとします。アプリケーションが障害から復旧したら、Kinesis Data Analytics は、設定された宛先への永続的出力を、最も低速な宛先に配信された最後のレコードから再開します。これにより、同じ出力レコードが別の宛先に複数回配信される場合があります。この場合は、外部で、宛先での重複を処理する必要があります。

エラー処理

Amazon Kinesis Data Analytics は、API や SQL のエラーを直接ユーザーに返します。API オペレーションの詳細については、「アクション」を参照してください。SQL エラーの処理の詳細については、「Amazon Kinesis Data Analytics SQL Reference」を参照してください。

Amazon Kinesis Data Analytics は、`error_stream` というアプリケーション内エラーストリームを使用して、ランタイムエラーをレポートします。

アプリケーション内エラーストリームを使用してエラーをレポートする

Amazon Kinesis Data Analytics は、`error_stream` というアプリケーション内エラーストリームを使用して、ランタイムエラーをレポートします。発生する可能性のあるエラーの例を以下に示します。

- ストリーミングソースから読み取られたレコードが入力スキーマに適合していない。
- アプリケーションコードがゼロでの除算を指定している。
- 行が入れ替わっている (たとえば、ユーザーによって `ROWTIME` 値が変更されたレコードがストリームに現れると、レコードの順序が乱れます)。
- ソースストリームのデータをスキーマで指定されたデータ型に変換することはできません (強制エラー)。変換できるデータ型の詳細については、「JSON データ型から SQL データ型へのマッピング」を参照してください。

これらのエラーは、SQL コードでプログラム的に処理するか、外部宛先へのエラーストリームにデータを保持することをお勧めします。これには、アプリケーションに出力設定を追加する必要があります (「アプリケーション出力の設定」を参照)。アプリケーション内エラーストリームの動作の例については、「例: アプリケーション内エラーストリームの確認」を参照してください。

> **Note**
>
> エラーストリームはシステムアカウントを使用して作成されるため、Kinesis Data Analytics アプリケーションはプログラムでエラーストリームにアクセスすることや、エラーストリームを変更することはできません。エラー出力を使用して、アプリケーションで発生する可能性のあるエラーを確認する必要があります。次に、アプリケーションの SQL コードを記述して、予期されるエラー条件を処理します。

エラーストリームのスキーマ

エラーストリームには、次のスキーマがあります。

フィールド	データ型	Notes (メモ)
ERROR_TIME	TIMESTAMP	エラーが発生した時刻。
ERROR_LEVEL	VARCHAR(10)	

ERROR_NAME	VARCHAR(32)	
MESSAGE	VARCHAR(4096)	
DATA_ROWTIME	TIMESTAMP	受信レコードの ROWTIME。
DATA_ROW	VARCHAR(49152)	元の行の 16 進エンコードデータ。標準ライブラリを使用してこの値を 16 進数でデコードするか、この Hex to String Converter などのウェブリソースを使用できます。
PUMP_NAME	VARCHAR(128)	`CREATE PUMP` で定義されている送信ポンプ。

アプリケーションを自動的にスケーリングしてスループットを向上させる

Amazon Kinesis Data Analytics は、ほとんどのシナリオでソースストリームのデータスループットとクエリの複雑さに対応するようにアプリケーションを柔軟に拡張します。Kinesis Data Analytics は、Kinesis 処理ユニット (KPU) の形式で容量をプロビジョニングします。単一の KPU には、メモリ (4 GB) とそれに対応するコンピューティングとネットワークがあります。

アプリケーションの KPU のデフォルト制限は 64 です。この制限の拡大をリクエストする方法については、「Amazon サービスの制限」にある「制限の拡大をリクエストするには」を参照してください。

タグ付けの使用

このセクションでは、Kinesis Data Analytics アプリケーションに key-value メタデータタグを追加する方法について説明します。これらのタグは以下の目的に使用できます。

- 個々の Kinesis Data Analytics アプリケーションに対する請求を決定する。詳細については、「Billing and Cost Management ユーザーガイド」の「AWS コスト配分タグの使用」を参照してください。

- タグに基づいてアプリケーションリソースへのアクセスをコントロールする。詳細については、ユーザーガイドのタグを使用したアクセス制御を参照してください。
- ユーザー定義の目的で。ユーザータグに基づいてアプリケーションの機能を定義できます。

タグ付けに関する以下の情報に注意してください。

- アプリケーションタグの最大数にはシステムタグが含まれます。ユーザー定義のアプリケーションタグの最大数は 50 です。
- アクションに含まれているタグリストで Key 値が重複している場合、サービスは `InvalidArgumentException` をスローします。

このトピックには、次のセクションが含まれています。

- アプリケーション作成時のタグの追加
- 既存のアプリケーションに対するタグの追加または更新
- アプリケーションのタグの一覧表示
- アプリケーションからのタグの削除

アプリケーション作成時のタグの追加

タグの追加は、CreateApplication アクションの `tags` パラメータを使ってアプリケーションを作成する際に行います。

以下のリクエスト例では、`CreateApplication` リクエストの `Tags` ノードを示しています。

```
"Tags": [
    {
        "Key": "Key1",
        "Value": "Value1"
    },
    {
        "Key": "Key2",
        "Value": "Value2"
    }
]
```

既存のアプリケーションに対するタグの追加または更新

TagResource アクションを使用して、アプリケーションにタグを追加します。UpdateApplication アクションを使用して、アプリケーションにタグを追加することはできません。

既存のタグを更新するには、既存のタグのものと同じキーを含むタグを追加します。

TagResource アクションの以下のリクエスト例では、新しいタグを追加するか、既存のタグを更新します。

```
{
   "ResourceARN": "string",
   "Tags": [
      {
         "Key": "NewTagKey",
         "Value": "NewTagValue"
      },
      {
         "Key": "ExistingKeyOfTagToUpdate",
         "Value": "NewValueForExistingTag"
      }
   ]
}
```

アプリケーションのタグの一覧表示

既存のタグを一覧表示するには、ListTagsForResource アクションを使用します。

ListTagsForResource アクションの以下のリクエスト例では、アプリケーションのタグを一覧表示します。

```
{
   "ResourceARN": "arn:aws:kinesisanalytics:us-west-2:012345678901:application/
MyApplication"
}
```

アプリケーションからのタグの削除

アプリケーションからタグを削除するには、UntagResource アクションを使用します。

UntagResource アクションの以下のリクエスト例では、アプリケーションからタグを削除します。

```
{
   "ResourceARN": "arn:aws:kinesisanalytics:us-west-2:012345678901:application/
MyApplication",
   "TagKeys": [ "KeyOfFirstTagToRemove", "KeyOfSecondTagToRemove" ]
}
```

Amazon Kinesis Data Analytics for SQL Applications の開始方法

以下のトピックでは、Amazon Kinesis Data Analytics for SQL アプリケーションの利用開始について説明します。初めて Kinesis Data Analytics for SQL アプリケーションを使用する場合は、基本操作セクションのステップを実行する前に、「Amazon Kinesis Data Analytics for SQL Applications: 仕組み」の概念と用語を確認することをお勧めします。

トピック

- にサインアップする AWS アカウント
- 管理アクセスを持つユーザーを作成する
- ステップ 1: アカウントを設定して管理ユーザーを作成する
- にサインアップする AWS アカウント
- 管理アクセスを持つユーザーを作成する
- ステップ 2: AWS Command Line Interface（AWS CLI) を設定する
- ステップ 3: スターター Amazon Kinesis Data Analytics アプリケーションを作成する
- ステップ 4 (オプション) コンソールを使用したスキーマと SQL コードの編集

にサインアップする AWS アカウント

がない場合は AWS アカウント、次の手順を実行して作成します。

にサインアップするには AWS アカウント

1. https://portal.aws.amazon.com/billing/signup を開きます。
2. オンラインの手順に従います。

   サインアップ手順の一部では、電話またはテキストメッセージを受信し、電話のキーパッドに検証コードを入力します。

   にサインアップすると AWS アカウント、 AWS アカウントのルートユーザー が作成されます。ルートユーザーには、アカウントのすべての AWS のサービス とリソースへのアクセス権があります。セキュリティベストプラクティスとして、ユーザーに管理アクセス権を割り当て、ルートユーザーアクセスが必要なタスクの実行にはルートユーザーのみを使用するようにしてください。

AWS サインアッププロセスが完了すると、 から確認メールが送信されます。https://aws.amazon.com/ の [マイアカウント] をクリックして、いつでもアカウントの現在のアクティビティを表示し、アカウントを管理することができます。

管理アクセスを持つユーザーを作成する

にサインアップしたら AWS アカウント、日常的なタスクにルートユーザーを使用しないように AWS アカウントのルートユーザー、 を保護し AWS IAM Identity Center、 を有効にして管理ユーザーを作成します。

を保護する AWS アカウントのルートユーザー

1. ルートユーザーを選択し、 AWS アカウント E メールアドレスを入力して、アカウント所有者AWS Management Consoleとして にサインインします。次のページでパスワードを入力します。

   ルートユーザーを使用してサインインする方法については、AWS サインイン ユーザーガイドのルートユーザーとしてサインインするを参照してください。

2. ルートユーザーの多要素認証 (MFA) を有効にします。

   手順については、IAM ユーザーガイドの AWS アカウント 「ルートユーザー (コンソール) の仮想 MFA デバイスを有効にする」を参照してください。

管理アクセスを持つユーザーを作成する

1. IAM アイデンティティセンターを有効にします。

   手順については、「AWS IAM Identity Center ユーザーガイド」の「AWS IAM Identity Centerの有効化」を参照してください。

2. IAM アイデンティティセンターで、ユーザーに管理アクセスを付与します。

   を ID ソース IAM アイデンティティセンターディレクトリ として使用する方法のチュートリアルについては、AWS IAM Identity Center 「 ユーザーガイド」の「デフォルトを使用してユーザーアクセスを設定する IAM アイデンティティセンターディレクトリ」を参照してください。

管理アクセス権を持つユーザーとしてサインインする

- IAM アイデンティティセンターのユーザーとしてサインインするには、IAM アイデンティティセンターのユーザーの作成時に E メールアドレスに送信されたサインイン URL を使用します。

  IAM Identity Center ユーザーを使用してサインインする方法については、AWS サインイン「 ユーザーガイド」の AWS 「 アクセスポータルにサインインする」を参照してください。

追加のユーザーにアクセス権を割り当てる

1. IAM アイデンティティセンターで、最小特権のアクセス許可を適用するというベストプラクティスに従ったアクセス許可セットを作成します。

   手順については、「AWS IAM Identity Center ユーザーガイド」の「権限設定を作成する」を参照してください。
2. グループにユーザーを割り当て、そのグループにシングルサインオンアクセス権を割り当てます。

   手順については、「AWS IAM Identity Center ユーザーガイド」の「グループの結合」を参照してください。

ステップ 1: アカウントを設定して管理ユーザーを作成する

Amazon Kinesis Data Analytics を初めて使用する場合は、事前に以下のタスクを実行してください。

1. にサインアップする AWS
2. IAM ユーザーの作成

にサインアップする AWS

Amazon Web Services にサインアップすると、 AWS アカウント は Amazon Kinesis Data Analytics を含む AWSのすべてのサービスに自動的にサインアップされます。料金は、使用するサービスの料金のみが請求されます。

Kinesis Data Analytics は、使用したリソース分のみお支払いいただくだけで利用可能です。新規の AWS お客様は、無料で Kinesis Data Analytics の使用を開始できます。詳細については、「AWS 無料利用枠」を参照してください。

が既にある場合は AWS アカウント、次のタスクに進みます。がない場合は AWS アカウント、次の手順のステップを実行して作成します。

を作成するには AWS アカウント

1. https://portal.aws.amazon.com/billing/signup を開きます。
2. オンラインの手順に従います。

   サインアップ手順の一部では、電話またはテキストメッセージを受信し、電話のキーパッドに検証コードを入力します。

   にサインアップすると AWS アカウント、 AWS アカウントのルートユーザー が作成されます。ルートユーザーには、アカウントのすべての AWS のサービス とリソースへのアクセス権があります。セキュリティベストプラクティスとして、ユーザーに管理アクセス権を割り当て、ルートユーザーアクセスが必要なタスクの実行にはルートユーザーのみを使用するようにしてください。

次のタスクで必要になるため、 AWS アカウント ID を書き留めます。

IAM ユーザーの作成

Amazon Kinesis Data Analytics などの のサービスでは AWS、アクセス時に認証情報を指定する必要があります。これにより、サービスが所有するリソースにアクセスするためのアクセス許可があるかどうかをサービスが判断できます。コンソールを使用するにはパスワードが必要です。 AWS CLI または API AWS アカウント にアクセスするための のアクセスキーを作成できます。ただし、 の認証情報 AWS を使用して にアクセスすることはお勧めしません AWS アカウント。代わりに AWS Identity and Access Management (IAM) を使用することをお勧めします。IAM ユーザーを作成し、管理者アクセス許可を持つ IAM グループにユーザーを追加したら、作成した IAM ユーザーに管理者アクセス許可を付与します。その後、特別な URL とその IAM ユーザーの認証情報 AWS を使用して にアクセスできます。

にサインアップしたが AWS、自分で IAM ユーザーを作成していない場合は、IAM コンソールを使用して作成できます。

このガイドの「使用開始」実習では、管理者権限を持つユーザー (`adminuser`) が存在すること想定しています。手順に従ってアカウントに `adminuser` を作成します。

管理者ユーザーを作成し、コンソールにサインインするには

1. AWS アカウントに `adminuser` という管理者ユーザーを作成します。手順については、「IAM ユーザーガイド」の「最初の IAM ユーザーと管理者グループの作成」を参照してください。
2. ユーザーは、特別な URL AWS Management Console を使用して にサインインできます。詳細については、「IAM ユーザーガイド」の「ユーザーがアカウントにサインインする方法」を参照してください。

IAM の詳細については、以下を参照してください。

- AWS Identity and Access Management (IAM)
- 開始方法
- IAM ユーザーガイド

次のステップ

ステップ 2: AWS Command Line Interface（AWS CLI) を設定する

にサインアップする AWS アカウント

がない場合は AWS アカウント、次の手順を実行して作成します。

にサインアップするには AWS アカウント

1. https://portal.aws.amazon.com/billing/signup を開きます。
2. オンラインの手順に従います。

   サインアップ手順の一部では、電話またはテキストメッセージを受信し、電話のキーパッドに検証コードを入力します。

   にサインアップすると AWS アカウント、 AWS アカウントのルートユーザー が作成されます。ルートユーザーには、アカウントのすべての AWS のサービス とリソースへのアクセス権があります。セキュリティベストプラクティスとして、ユーザーに管理アクセス権を割り当て、ルートユーザーアクセスが必要なタスクの実行にはルートユーザーのみを使用するようにしてください。

AWS サインアッププロセスが完了すると、 から確認メールが送信されます。https://aws.amazon.com/ の [マイアカウント] をクリックして、いつでもアカウントの現在のアクティビティを表示し、アカウントを管理することができます。

管理アクセスを持つユーザーを作成する

にサインアップしたら AWS アカウント、日常的なタスクにルートユーザーを使用しないように AWS アカウントのルートユーザー、 を保護し AWS IAM Identity Center、 を有効にして管理ユーザーを作成します。

を保護する AWS アカウントのルートユーザー

1. ルートユーザーを選択し、 AWS アカウント E メールアドレスを入力して、アカウント所有者AWS Management Consoleとして にサインインします。次のページでパスワードを入力します。

   ルートユーザーを使用してサインインする方法については、AWS サインイン ユーザーガイドのルートユーザーとしてサインインするを参照してください。

2. ルートユーザーの多要素認証 (MFA) を有効にします。

   手順については、IAM ユーザーガイドの AWS アカウント 「ルートユーザー (コンソール) の仮想 MFA デバイスを有効にする」を参照してください。

管理アクセスを持つユーザーを作成する

1. IAM アイデンティティセンターを有効にします。

   手順については、「AWS IAM Identity Center ユーザーガイド」の「AWS IAM Identity Centerの有効化」を参照してください。

2. IAM アイデンティティセンターで、ユーザーに管理アクセスを付与します。

   を ID ソース IAM アイデンティティセンターディレクトリ として使用する方法のチュートリアルについては、「 AWS IAM Identity Center ユーザーガイド」の「デフォルトを使用してユーザーアクセスを設定する IAM アイデンティティセンターディレクトリ」を参照してください。

管理アクセス権を持つユーザーとしてサインインする

- IAM アイデンティティセンターのユーザーとしてサインインするには、IAM アイデンティティセンターのユーザーの作成時に E メールアドレスに送信されたサインイン URL を使用します。

  IAM Identity Center ユーザーを使用してサインインする方法については、AWS サインイン「 ユーザーガイド」の AWS 「 アクセスポータルにサインインする」を参照してください。

追加のユーザーにアクセス権を割り当てる

1. IAM アイデンティティセンターで、最小特権のアクセス許可を適用するというベストプラクティスに従ったアクセス許可セットを作成します。

   手順については、「AWS IAM Identity Center ユーザーガイド」の「権限設定を作成する」を参照してください。
2. グループにユーザーを割り当て、そのグループにシングルサインオンアクセス権を割り当てます。

   手順については、「AWS IAM Identity Center ユーザーガイド」の「グループの結合」を参照してください。

ステップ 2: AWS Command Line Interface （ AWS CLI) を設定する

() をダウンロードして設定する手順に従います AWS Command Line Interface AWS CLI。

> **Important**
>
> 「開始方法」の演習のステップを実行する AWS CLI ために は必要ありません。ただし、このガイドの一部の実習では AWS CLIを使用します。このステップをスキップして に移動し ステップ 3: スターター Amazon Kinesis Data Analytics アプリケーションを作成する、 AWS CLI 後で必要なときに をセットアップできます。

をセットアップするには AWS CLI

1. AWS CLIをダウンロードして設定します。手順については、AWS Command Line Interface ユーザーガイドの次のトピックを参照してください。

   - のセットアップ AWS Command Line Interface

- の設定 AWS Command Line Interface

2. AWS CLI 設定ファイルに管理者ユーザーの名前付きプロファイルを追加します。 AWS CLI コマンドを実行するときは、このプロファイルを使用します。名前付きプロファイルの詳細については、AWS Command Line Interface ユーザーガイドの「名前付きプロファイル」を参照してください。

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

利用可能な のリストについては AWS リージョン、『』の「リージョンとエンドポイント」を参照してくださいAmazon Web Services 全般のリファレンス。

3. コマンドプロンプトで以下のヘルプコマンドを入力して、セットアップを確認します。

```
aws help
```

次のステップ

ステップ 3: スターター Amazon Kinesis Data Analytics アプリケーションを作成する

ステップ 3: スターター Amazon Kinesis Data Analytics アプリケーションを作成する

このセクションのステップに従い、コンソールを使用して最初の Kinesis Data Analytics アプリケーションを作成します。

> **Note**
>
> 「はじめに」の実習を始める前に、「Amazon Kinesis Data Analytics for SQL Applications: 仕組み」を確認することをお勧めします。

この「はじめに」の演習では、コンソールを使用して、アプリケーションコードでデモストリームまたはテンプレートのいずれかを操作することができます。

- デモストリームを使用する場合、コンソールはアカウントに Kinesis データストリーム (`kinesis-analytics-demo-stream`) を作成します。

  Kinesis Data Analytics アプリケーションには、ストリーミングソースが必要です。このソースの場合、このガイドの複数の SQL の例では、デモストリーム (`kinesis-analytics-demo-stream`) を使用しています。また、コンソールは、次のようにサンプルデータ (模擬株取引レコード) を連続してこのストリームに追加するスクリプトを実行します。

Raw | Lambda output | Formatted

Filter by column name or column type

TICKER_SYMBOL VARCHAR(4)	SECTOR VARCHAR(16)	CHANGE REAL	PRICE REAL
JYB	HEALTHCARE	-2.05	43.17
DFT	RETAIL	0.17	95.96000000000001
JYB	HEALTHCARE	1.8900000000000001	45.22
WFC	FINANCIAL	0.05	47.51
SED	HEALTHCARE	0.11	2.31
QAZ	FINANCIAL	-1.01	194.02
QXZ	FINANCIAL	-4.36	219.21
TGT	RETAIL	1.51	69.9
AAPL	TECHNOLOGY	-0.27	101.37
DFT	RETAIL	-0.7000000000000001	95.79

  この実習では、アプリケーションのストリーミングソースとして `kinesis-analytics-demo-stream` を使用できます。

> **Note**
>
> デモストリームはアカウントに残ります。これを使用して、このガイドの他の例をテストできます。ただし、コンソールを離れると、コンソールが使用するスクリプトによってデータの入力が停止されます。必要な場合は、コンソールによってストリームへの入力を再度開始するオプションが提供されます。

- アプリケーションコードの例を含むテンプレートを使用すると、コンソールのテンプレートコードを使用して、デモストリームでシンプルな分析を実行することができます。

これらの機能を使用して、次のように最初のアプリケーションをすばやく設定できます。

1. アプリケーションの作成 – 名前を指定するだけです。コンソールによってアプリケーションが作成され、サービスによってアプリケーションの状態が READY に設定されます。

2. 入力の設定 – まず、ストリーミングソースとしてデモストリームを追加します。これを使用するには、コンソールでデモストリームを作成する必要があります。その後、コンソールは、デモストリームのレコードをランダムにサンプリングして、作成されるアプリケーション内入力ストリームのスキーマを推測します。コンソールは、アプリケーション内ストリームを `SOURCE_SQL_STREAM_001` と命名します。

   コンソールは、検出 API を使用してスキーマを推測します。必要に応じて、推測スキーマを編集できます。詳細については、「DiscoverInputSchema」を参照してください。Kinesis Data Analytics では、このスキーマを使用してアプリケーション内ストリームを作成します。

   アプリケーションを開始すると、Kinesis Data Analytics はユーザーに代わってデモストリームを継続的に読み取り、アプリケーション内入力ストリームの `SOURCE_SQL_STREAM_001` に行を挿入します。

3. アプリケーションコードの指定 – 次のコードを提供するテンプレート ([Continuous filter] といいます) を使用します。

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM"
  (symbol VARCHAR(4), sector VARCHAR(12), CHANGE DOUBLE, price DOUBLE);

-- Create pump to insert into output.
CREATE OR REPLACE PUMP "STREAM_PUMP" AS
   INSERT INTO "DESTINATION_SQL_STREAM"
      SELECT STREAM ticker_symbol, sector, CHANGE, price
      FROM "SOURCE_SQL_STREAM_001"
      WHERE sector SIMILAR TO '%TECH%';
```

   このアプリケーションコードでは、アプリケーション内ストリーム (`SOURCE_SQL_STREAM_001`) をクエリします。また、ポンプを使用して、表示された行を別のアプリケーション内ストリーム

(DESTINATION_SQL_STREAM) に挿入します。コーディングパターンの詳細については、「アプリケーションコード」を参照してください。

Kinesis Data Analytics でサポートされている SQL 言語要素の詳細については、「Amazon Kinesis Data Analytics SQL Reference」を参照してください。

4. 出力の設定 – この実習では、出力は設定しません。つまり、アプリケーションが作成するアプリケーション内ストリームのデータを外部宛先で永続化しません。代わりに、コンソールでクエリの結果を確認します。このガイドのその他の例では、出力を設定する方法について説明します。例については、「例: 簡単なアラートの作成」を参照してください。

> **Important**
>
> この実習では米国東部 (バージニア北部) リージョン (us-east-1) を使用してアプリケーションをセットアップします。サポートされている のいずれかを使用できます AWS リージョン。

次のステップ

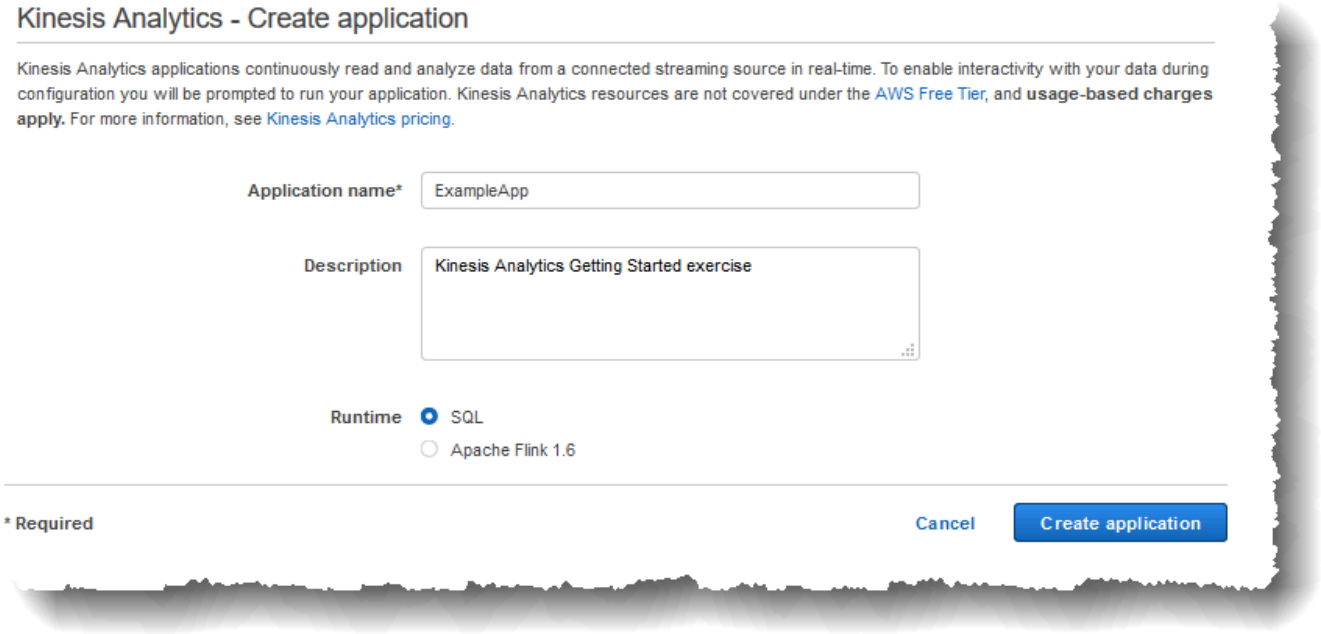
ステップ 3.1: アプリケーションの作成

ステップ 3.1: アプリケーションの作成

このセクションでは、Amazon Kinesis Data Analytics アプリケーションを作成します。次のステップで、アプリケーションを設定します。

データ分析アプリケーションを作成するには

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/kinesisanalytics で Managed Service for Apache Flink コンソールを開きます。
2. [アプリケーションの作成] を選択します。
3. [アプリケーションの作成] ページで、アプリケーション名を入力し、説明を入力して、アプリケーションの [ランタイム] 設定に [SQL] を選択します。そして、[アプリケーションの作成] を選択します。



これにより、ステータスが READY の Kinesis Data Analytics アプリケーションが作成されます。コンソールに、入力と出力を設定できるアプリケーションハブが表示されます。

> **Note**
>
> アプリケーションを作成するために「CreateApplication」オペレーションで必要なのはアプリケーション名のみです。入力と出力の設定は、コンソールでアプリケーションを作成した後に追加できます。

次のステップでは、アプリケーションの入力を設定します。入力設定では、アプリケーションのストリーミングデータソースを追加し、ストリーミングソースのデータをサンプリングして、アプリケーション内入力ストリームのスキーマを検出します。

次のステップ

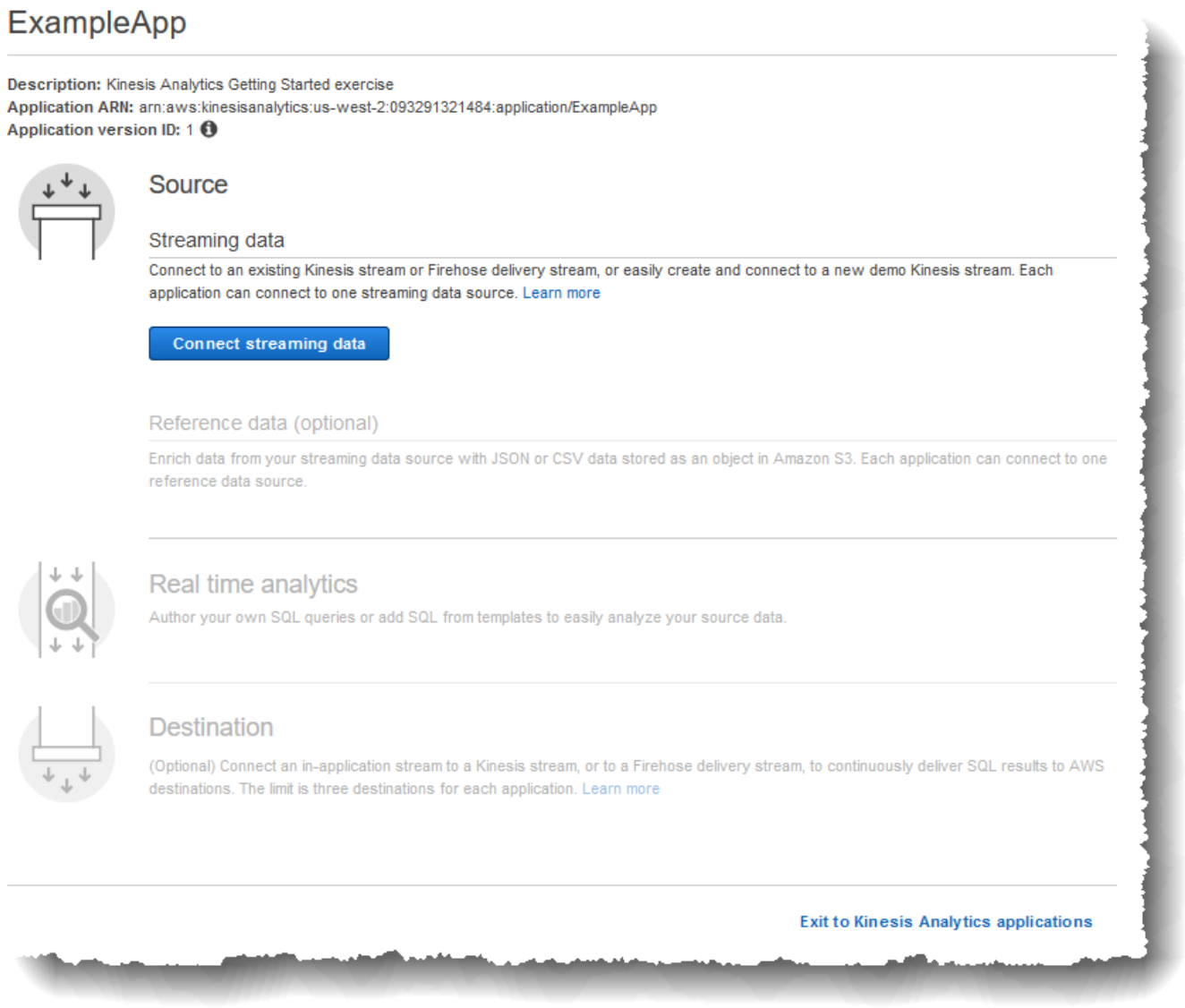
ステップ 3.2: 入力の設定

ステップ 3.2: 入力の設定

アプリケーションにはストリーミングソースが必要です。開始しやすくするために、コンソールでデモストリーム (`kinesis-analytics-demo-stream` といいます) を作成できます。また、コンソールでストリームにレコードを入力するスクリプトが実行されます。

ストリーミングソースをアプリケーションに追加するには

1. コンソールのアプリケーションハブページで、[ストリーミングデータの接続] を選択します。



2. 表示されたページで、以下を確認します。

- [Source] セクション。アプリケーションのストリーミングソースを指定します。既存のストリームソースを選択するか、1 つ作成できます。この実習では、新しいストリームとしてデモストリームを作成します。

  デフォルトでは、コンソールは作成されたアプリケーション内入力ストリームを `INPUT_SQL_STREAM_001` と命名します。この実習では、表示された名前のままにします。

  - [ストリームリファレンス名] – このオプションでは、作成されているアプリケーション内入力ストリームの名前 (`SOURCE_SQL_STREAM_001`) が表示されます。名前は変更できますが、この実習ではこの名前のままにします。

    入力設定で、デモストリームを作成したアプリケーション内入力ストリームにマッピングします。アプリケーションを開始すると、Amazon Kinesis Data Analytics はデモストリームを継続的に読み取り、アプリケーション内入力ストリームに行を挿入します。アプリケーションコードでこのアプリケーション内入力ストリームをクエリします。

  - を使用したレコード前処理 AWS Lambda: このオプションは、アプリケーションコードが実行される前に入力ストリームのレコードを変更する AWS Lambda 式を指定する場所です。この演習では、[Disabled] オプションを選択したままにしておきます。Lambda 事前処理の詳細については、「Lambda 関数を使用したデータの事前処理」を参照してください。

このページのすべての情報を指定すると、コンソールは更新リクエスト (「UpdateApplication」を参照) を送信して入力設定をアプリケーションに追加します。

3. [Source] ページで、[Configure a new stream] を選択します。
4. [Create demo stream] を選択します。コンソールで、以下を実行してアプリケーション入力を設定します。

   - コンソールで Kinesis データストリーム (`kinesis-analytics-demo-stream`) を作成します。
   - コンソールで、株のティッカーデータのサンプルがストリームに入力されます。

- DiscoverInputSchema 入力アクションを使用して、ストリームのサンプルレコードを読み取って、コンソールでスキーマを推測します。推定されるスキーマは、作成したアプリケーション内入力ストリーム用のスキーマです。詳細については、「アプリケーション入力の設定」を参照してください。
- コンソールに、推測スキーマと、スキーマを推測するためにストリーミングソースから読み取ったサンプルデータが表示されます。

コンソールに、ストリーミングソースのサンプルレコードが表示されます。

Raw | Lambda output | Formatted

Filter by column name or column type

TICKER_SYMBOL VARCHAR(4)	SECTOR VARCHAR(16)	CHANGE REAL	PRICE REAL
JYB	HEALTHCARE	-2.05	43.17
DFT	RETAIL	0.17	95.96000000000001
JYB	HEALTHCARE	1.8900000000000001	45.22
WFC	FINANCIAL	0.05	47.51
SED	HEALTHCARE	0.11	2.31
QAZ	FINANCIAL	-1.01	194.02
QXZ	FINANCIAL	-4.36	219.21
TGT	RETAIL	1.51	69.9
AAPL	TECHNOLOGY	-0.27	101.37
DFT	RETAIL	-0.7000000000000001	95.79

[Stream sample] コンソールページに次のように表示されます。

- [Raw stream sample] タブには、スキーマを推測するための、API アクション (DiscoverInputSchema) でサンプリングされた未加工のストリームレコードが表示されます。
- [Formatted stream sample] タブには、[Raw stream sample] タブのデータの表形式バージョンが表示されます。
- [Edit schema] を選択すると、推定スキーマを編集することができます。この実習では、推測スキーマを変更しないでください。スキーマの詳細については、「スキーマエディタの使用」を参照してください。

[Rediscover schema] を選択している場合は、再度「DiscoverInputSchema」を実行してスキーマを推測するようにコンソールにリクエストできます。

5. [Save and continue] を選択します。

   これで、入力設定が追加されたアプリケーションができました。次のステップでは、SQL コードを追加して、アプリケーション内入力ストリームのデータに分析を実行します。

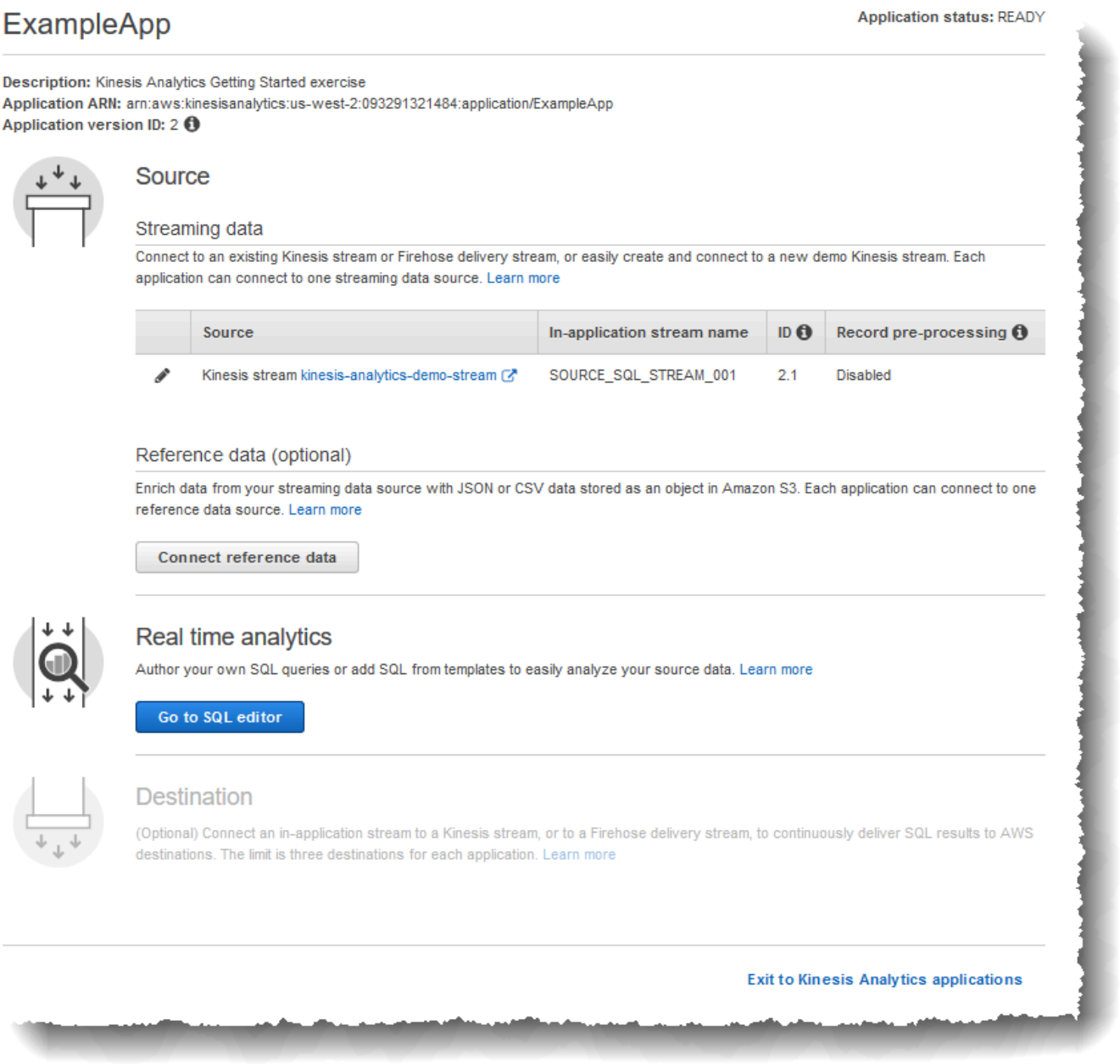
次のステップ

ステップ 3.3: リアルタイム分析を追加する (アプリケーションコードの追加)

ステップ 3.3: リアルタイム分析を追加する (アプリケーションコードの追加)

アプリケーション内ストリームに対して独自の SQL クエリを作成することもできますが、以下のステップではサンプルコードを提供するテンプレートの 1 つを使用します。

1. アプリケーションハブページで、[Go to SQL editor] を選択します。



2. [Would you like to start running "ExampleApp"? (「ExampleApp」を実行しますか?)] ダイアログボックスで、[Yes, start application (はい、アプリケーションを開始します)] を選択します。

   コンソールでアプリケーションの開始リクエストが送信され (「StartApplication」を参照)、SQL エディタページが表示されます。

3. コンソールに SQL エディタページが開きます。ボタン ([Add SQL from templates]、[Save and run SQL]) およびさまざまなタブを含むページを確認します。
4. SQL エディタで、[Add SQL from templates] を選択します。

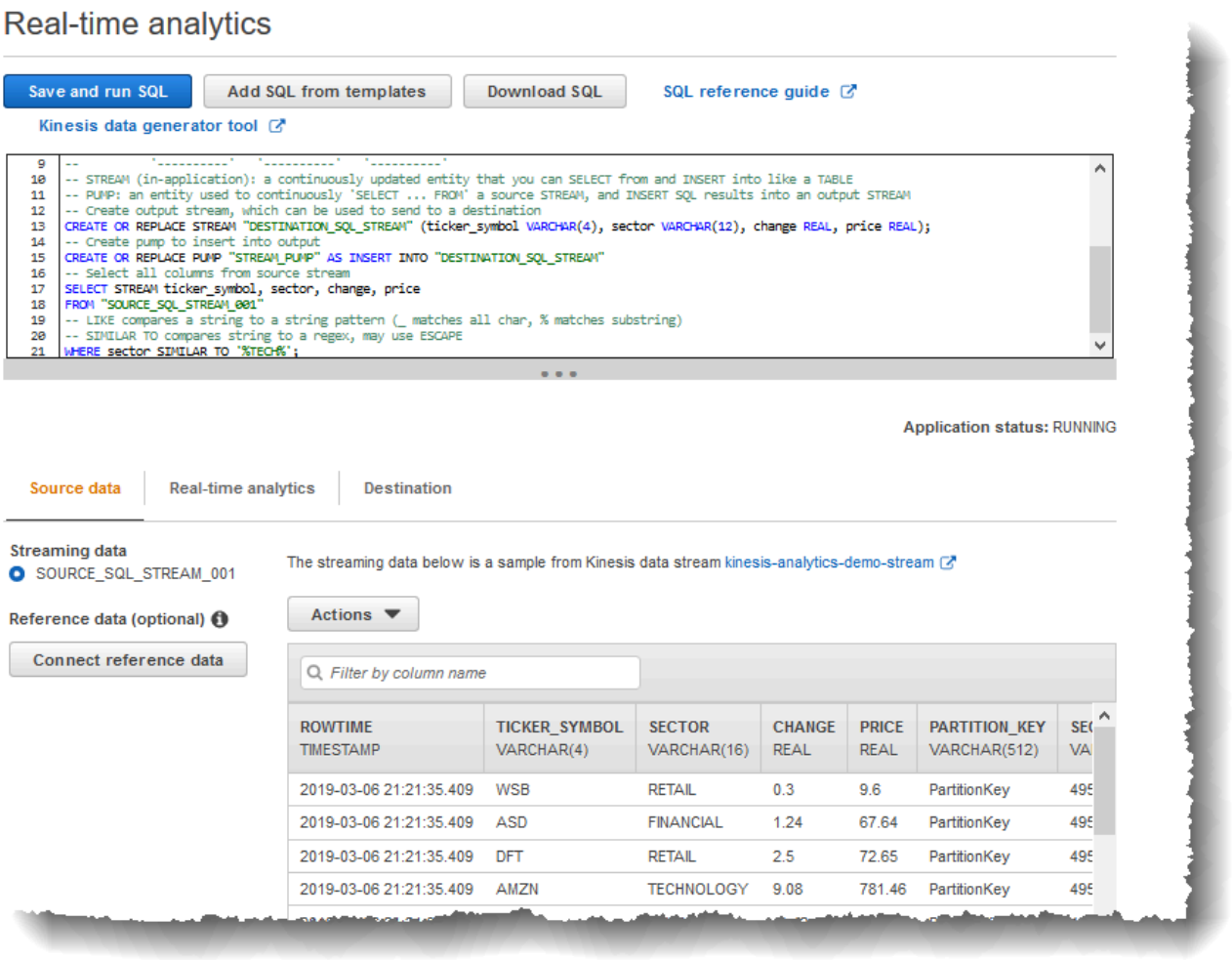
5. 使用可能なテンプレートの一覧から、[Continuous filter] を選択します。次のように、サンプルコードは 1 つのアプリケーション内ストリームからデータを読み取り (WHERE 句は行をフィルタします)、別のアプリケーション内ストリームに挿入します。

   - アプリケーション内ストリーム DESTINATION_SQL_STREAM を作成します。
   - ポンプ (STREAM_PUMP) を作成し、これを使用して SOURCE_SQL_STREAM_001 から行から選択して、DESTINATION_SQL_STREAM に挿入します。

6. [Add this SQL to editor] を選択します。
7. 次のように、アプリケーションコードをテストします。

   アプリケーションが既に開始されていることを忘れないでください (ステータスは RUNNING です)。従って、Amazon Kinesis Data Analytics は既にストリーミングソースから継続的に読み取り、アプリケーション内ストリーム SOURCE_SQL_STREAM_001 に行を追加しています。

   a. SQL エディタで、[Save and run SQL] を選択します。コンソールはまず更新リクエストを送信してアプリケーションコードを保存します。その後、コードは継続実行されます。
   b. 結果は [Real-time analytics] に表示されます。



SQL エディタには次のタブがあります。

- [Source data] タブに、ストリーミングソースにマッピングされたアプリケーション内入力ストリームが表示されます。アプリケーション内ストリームを選択すると、データが入力されます。入力設定で指定されていないアプリケーション内入力ストリームの追加列を書き留めてください。これらには以下のタイプスタンプ列が含まれます。

  - ROWTIME – アプリケーション内ストリームの各行には、`ROWTIME` という特殊な列があります。この列は、Amazon Kinesis Data Analytics が最初のアプリケーション内ストリーム (ストリーミングソースにマッピングされたアプリケーション内入力ストリーム) に行を挿入したときのタイムスタンプです。

- Approximate_Arrival_Time – 各 Kinesis レコードには、`Approximate_Arrival_Time` という値が含まれています。この値は、ストリーミングソースが正常にレコードを受信して保存したときに設定されるおおよその到達タイムスタンプです。Kinesis Data Analytics がストリーミングソースからレコードを読み取る際に、この列をアプリケーション内入力ストリームにフェッチします。

これらのタイムスタンプ値は時間ベースのウィンドウクエリで役に立ちます。詳細については、「ウィンドウクエリ」を参照してください。

- [Real-time analytics] タブには、アプリケーションコードによって作成された他のすべてのアプリケーション内ストリームが表示されます。また、エラーストリームも含まれます。Kinesis Data Analytics では、処理できない行はエラーストリームに送信されます。詳細については、「エラー処理」を参照してください。

`DESTINATION_SQL_STREAM` を選択して、アプリケーションコードが挿入した行を表示します。アプリケーションコードで作成されなかった追加列を書き留めてください。この列には、`ROWTIME` タイムスタンプ列が含まれます。Kinesis Data Analytics は、単純にこれらの値をソース (`SOURCE_SQL_STREAM_001`) からコピーします。

- [宛先] タブには、Kinesis Data Analytics がクエリ結果を書き込む外部宛先が表示されます。まだアプリケーション出力用の外部宛先は設定されていません。

次のステップ

ステップ 3.4: (オプション) アプリケーションコードを更新する

ステップ 3.4: (オプション) アプリケーションコードを更新する

このステップでは、アプリケーションコードを更新する方法について学びます。

アプリケーションコードを更新するには

1. 次のように別のアプリケーション内ストリームを作成します。

- DESTINATION_SQL_STREAM_2 という別のアプリケーション内ストリームを作成します。
- ポンプを作成し、それを使用して DESTINATION_SQL_STREAM から行を選択して、新しく作成したストリームに行を挿入します。

SQL エディタで、既存アプリケーションコードに次のコードを追加します。

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM_2"
           (ticker_symbol VARCHAR(4),
            change        DOUBLE,
            price         DOUBLE);

CREATE OR REPLACE PUMP "STREAM_PUMP_2" AS
   INSERT INTO "DESTINATION_SQL_STREAM_2"
      SELECT STREAM ticker_symbol, change, price
      FROM   "DESTINATION_SQL_STREAM";
```

コードを保存して実行します。追加のアプリケーション内ストリームが [Real-time analytics] タブに表示されます。

2. 2 つのアプリケーション内ストリームを作成します。SOURCE_SQL_STREAM_001 の行を株価ティッカーに基づいてフィルタし、それらを別々のストリームに挿入します。

   次の SQL ステートメントをアプリケーションコードに追加します。

```
CREATE OR REPLACE STREAM "AMZN_STREAM"
           (ticker_symbol VARCHAR(4),
            change        DOUBLE,
            price         DOUBLE);

CREATE OR REPLACE PUMP "AMZN_PUMP" AS
   INSERT INTO "AMZN_STREAM"
      SELECT STREAM ticker_symbol, change, price
      FROM   "SOURCE_SQL_STREAM_001"
      WHERE  ticker_symbol SIMILAR TO '%AMZN%';

CREATE OR REPLACE STREAM "TGT_STREAM"
           (ticker_symbol VARCHAR(4),
            change        DOUBLE,
            price         DOUBLE);

CREATE OR REPLACE PUMP "TGT_PUMP" AS
```

```
INSERT INTO "TGT_STREAM"
   SELECT STREAM ticker_symbol, change, price
   FROM   "SOURCE_SQL_STREAM_001"
   WHERE  ticker_symbol SIMILAR TO '%TGT%';
```

コードを保存して実行します。[Real-time analytics] タブの追加のアプリケーション内ストリームに注意してください。

これで、機能する最初の Amazon Kinesis Data Analytics アプリケーションができました。この実習では、次の操作を行います。

- 最初の Kinesis Data Analytics アプリケーションを作成しました。

- デモストリームをストリーミングソースとして識別し、作成されたアプリケーション内ストリーム (SOURCE_SQL_STREAM_001) にマッピングするアプリケーション入力を設定しました。Kinesis Data Analytics は連続してデモストリームを読み込み、レコードをアプリケーション内ストリームに挿入します。

- アプリケーションコードで SOURCE_SQL_STREAM_001 をクエリし、出力を DESTINATION_SQL_STREAM という別のアプリケーション内ストリームに書き込みました。

これで、オプションでアプリケーション出力を設定し、アプリケーション出力を外部宛先に書き込むことができます。つまり、DESTINATION_SQL_STREAM のレコードを外部宛先に書き込むようにアプリケーション出力を設定できます。この演習では、このステップはオプションです。この送信先を設定する方法については、次のステップに進みます。

次のステップ

ステップ 4 (オプション) コンソールを使用したスキーマと SQL コードの編集.

ステップ 4 (オプション) コンソールを使用したスキーマと SQL コードの編集

推測したスキーマを編集する方法と、Amazon Kinesis Data Analytics の SQL コードを編集する方法は以下のとおりです。そのためには、Kinesis Data Analytics コンソールの一部であるスキーマエディタと SQL エディタを操作します。

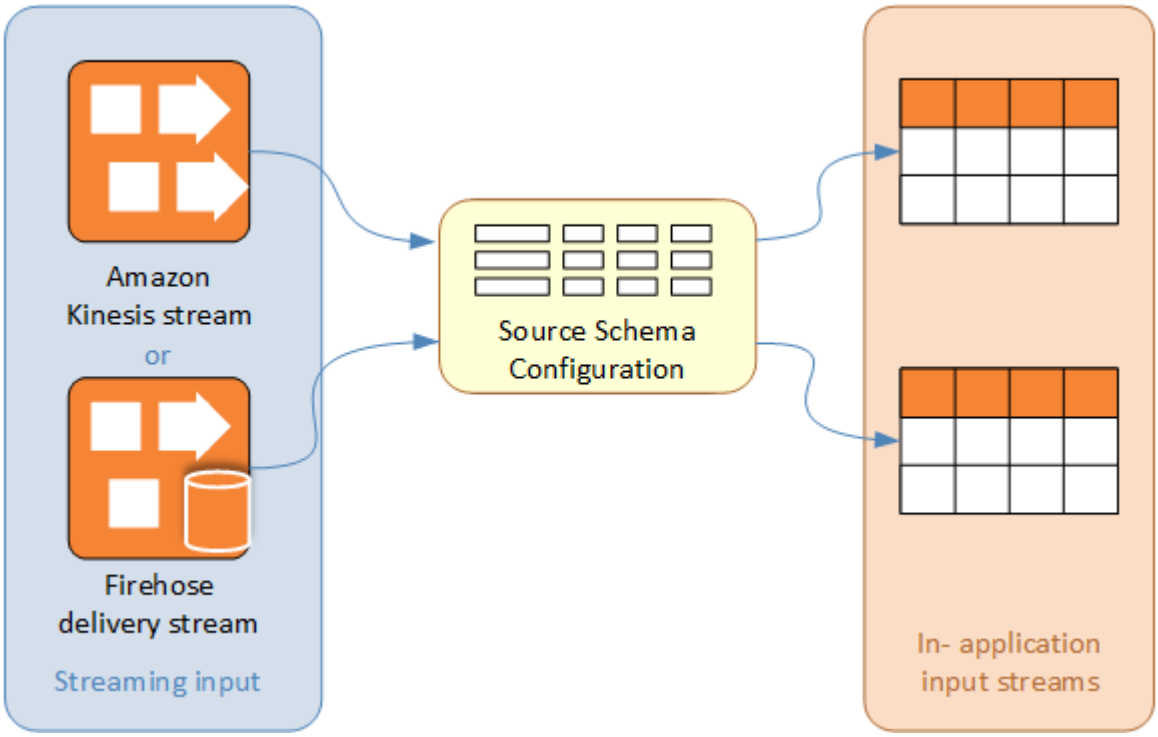
> **Note**
>
> コンソールのデータにアクセスまたはサンプリングするには、ログインユーザーのロールに `kinesisanalytics:GetApplicationState` アクセス許可が必要です。Kinesis Data Analytics アプリケーションのアクセス許可の詳細については、「アクセス管理の概要」を参照してください。

トピック

- スキーマエディタの使用
- SQL エディタの使用

スキーマエディタの使用

Amazon Kinesis Data Analytics アプリケーションの入力ストリームのスキーマでは、ストリームのデータをアプリケーションの SQL クエリで利用できるようにする方法が定義されています。



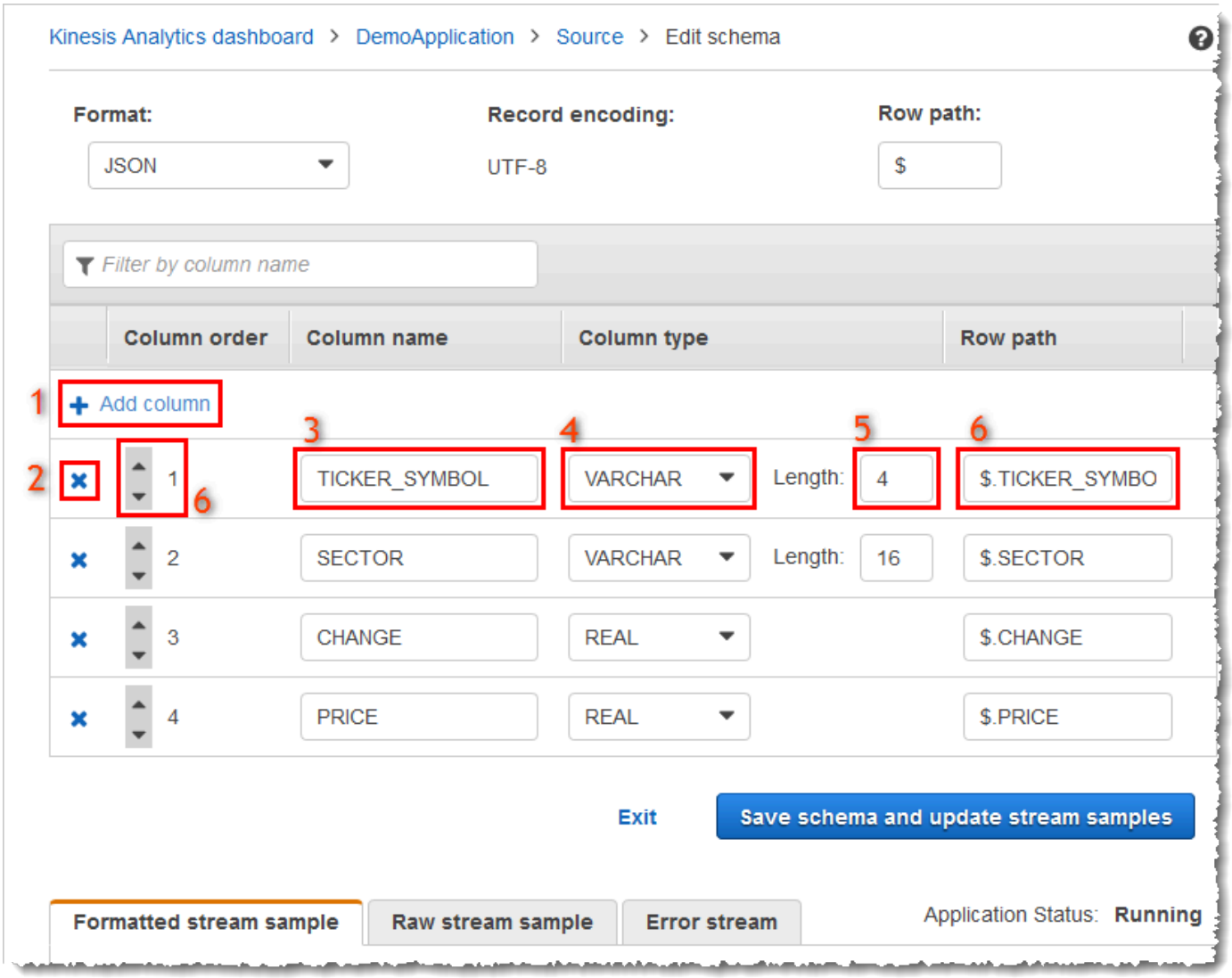
このスキーマには、アプリケーション内入力ストリームのデータ列に変換されるストリーミング入力の部分を判断する選択条件が含まれます。この入力には、次のいずれかの値を指定できます。

- JSON 入力ストリームの JSONPath 式。JSONPath は、JSON データに対してクエリを実行するためのツールです。
- カンマ区切り値 (CSV) 形式の入力ストリームの列番号。
- アプリケーション内データストリームのデータを提供する列名と SQL データ型。このデータ型にも、文字データまたはバイナリデータ長が含まれています。

コンソールは、「DiscoverInputSchema」を使用してスキーマを生成しようとします。スキーマの検出に失敗した場合や、不適切または未完了のスキーマが返る場合、スキーマエディタを使用して、手動でスキーマを編集する必要があります。

スキーマエディタのメイン画面

次のスクリーンショットは、スキーマエディタのメイン画面です。



次の編集をスキーマに適用することができます。

- 列の追加 (1): データ項目が自動的に検出されない場合は、データ列を追加する必要がある場合があります。
- 列の削除 (2): アプリケーションで不要な場合は、ソースストリームからデータを除外することができます。除外によって、ソースストリームのデータに影響を及ぼすことはありません。除外されたデータは、アプリケーションで利用できません。
- 列名 (3) の変更 列名は空白にすることはできません。必ず 2 文字以上にし、予約された SQL キーワードを含めることはできません。SQL の通常の識別子の命名基準を満たす必要があります。名前は必ず文字で開始し、文字、アンダースコア文字、数字のみ含めることができます。

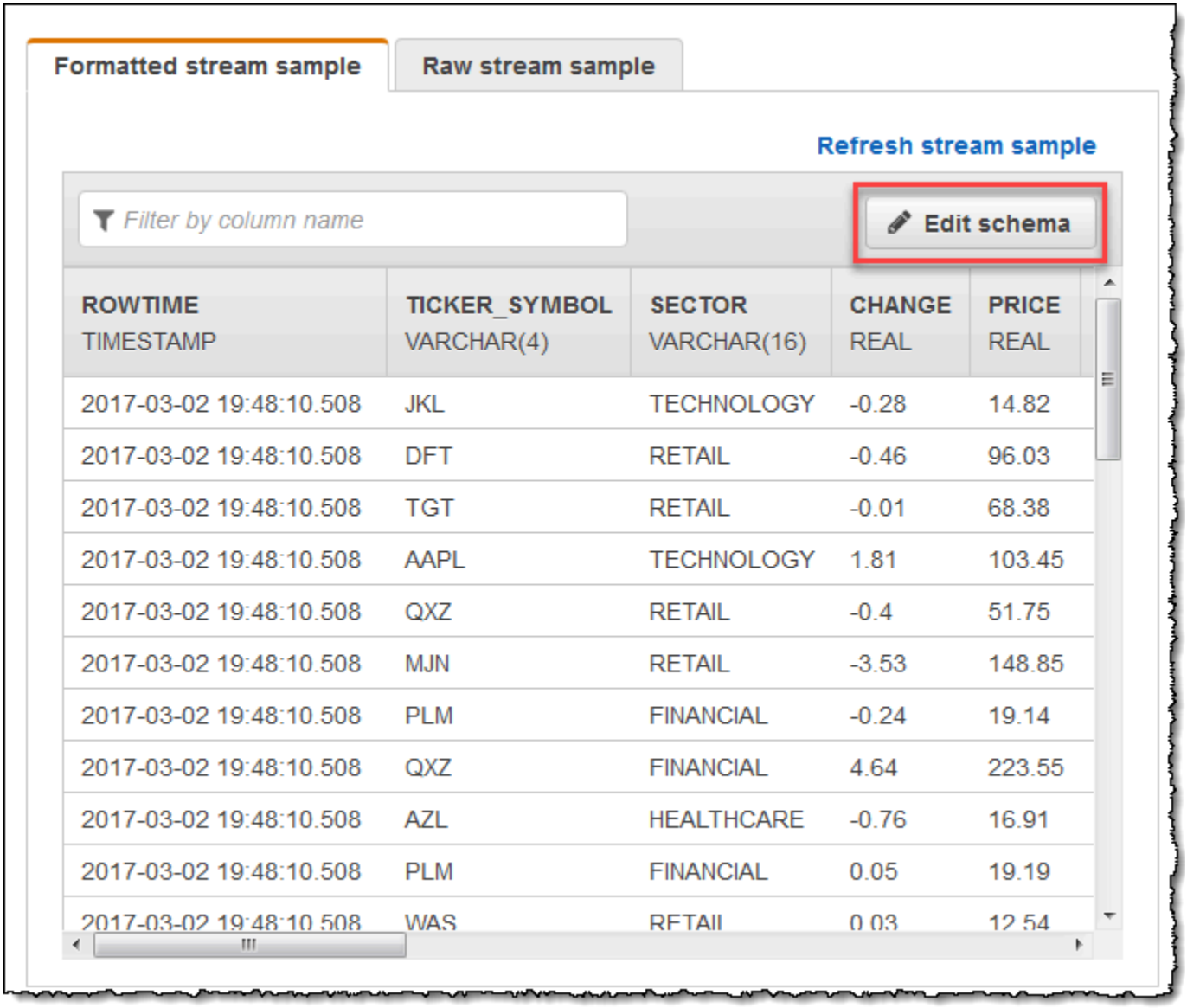
- 列のデータ型 (4) または長さ (5) の変更: 列に対して、互換性のあるデータ型を指定することができます。互換性のないデータ型を指定した場合、その列には NULL が入力されるか、アプリケーション内ストリームは追加されません。後者の場合は、エラーがエラーストリームに書き込まれます。指定する列の長さが小さすぎると、受信データは切り捨てられます。
- 列の選択条件の変更 (6): 列のデータのソースを判断するために使用される JSONPath 式または CSV 列の順序を編集できます。JSON スキーマの選択条件を変更するには、行のパス式に新しい値を入力します。CSV スキーマでは、選択条件として列の順序が使用されます。CSV スキーマの選択条件を変更するには、列の順序を変更します。

ストリーミングソースのスキーマの編集

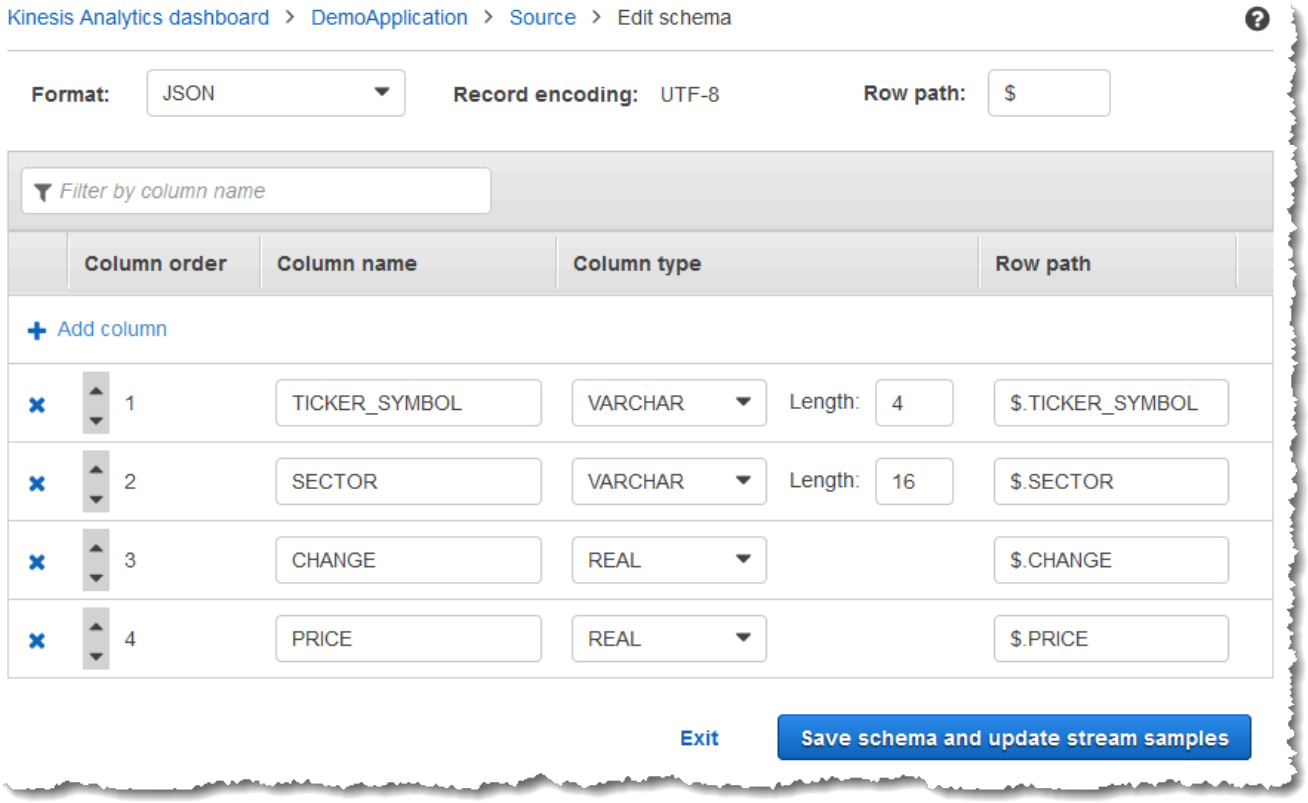
ストリーミングソースのスキーマを編集する必要がある場合は、以下の手順を実行します。

ストリーミングソースのスキーマを編集するには

1. [Source] ページで、[Edit schema] を選択します。



2. [Edit schema] ページで、ソーススキーマを編集します。



3. [Format] で、[JSON] または [CSV] を選択します。JSON 形式または CSV 形式の場合、ISO 8859-1 エンコードがサポートされています。

JSON 形式または CSV 形式の詳細については、次のセクションの手順を参照してください。

JSON スキーマの編集

JSON スキーマは次のステップを使用して編集できます。

JSON スキーマを編集するには

1. スキーマエディタで、[Add column] を編集して列を追加します。

   新しい列が最初の列の位置に表示されます。列の順序を変更するには、列名の横にある上向き矢印と下向き矢印を選択します。

   新しい列に、以下の情報を入力します。

   - [Column name] に名前を入力します。

列名は空白にすることはできません。必ず 2 文字以上にし、予約された SQL キーワードを含めることはできません。SQL の通常の識別子の命名基準を満たす必要があります。名前は必ず文字で開始し、文字、アンダースコア文字、数字のみ含めることができます。

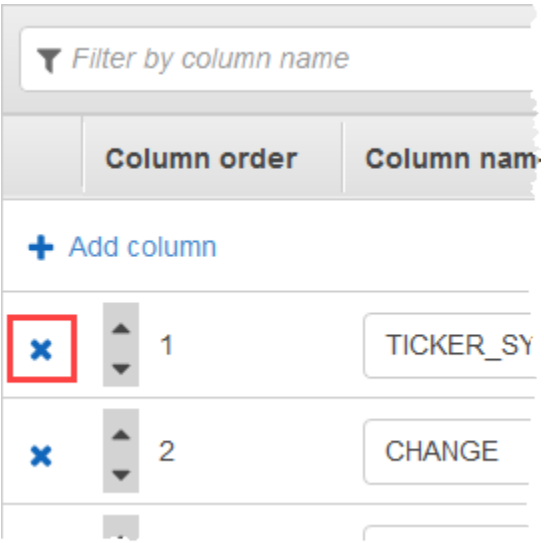
- [Column type] に、SQL データ型を入力します。

  列タイプは、サポートされている任意の SQL データ型です。新しいデータ型が CHAR、VARBINARY、VARCHAR のいずれかの場合は、[Length] にデータ長を指定します。詳細については、「データ型」を参照してください。

- [Row path] に、列パスを指定します。行のパスは、JSON 要素にマッピングする有効な JSONPath 式です。

> **Note**
>
> 基本の [Row path] 値は、最上位の親へのパスで、インポートされるデータを含みます。デフォルトでは、この値は [$] です。詳細については、「`JSONMappingParameters`」の `RecordRowPath` を参照してください。

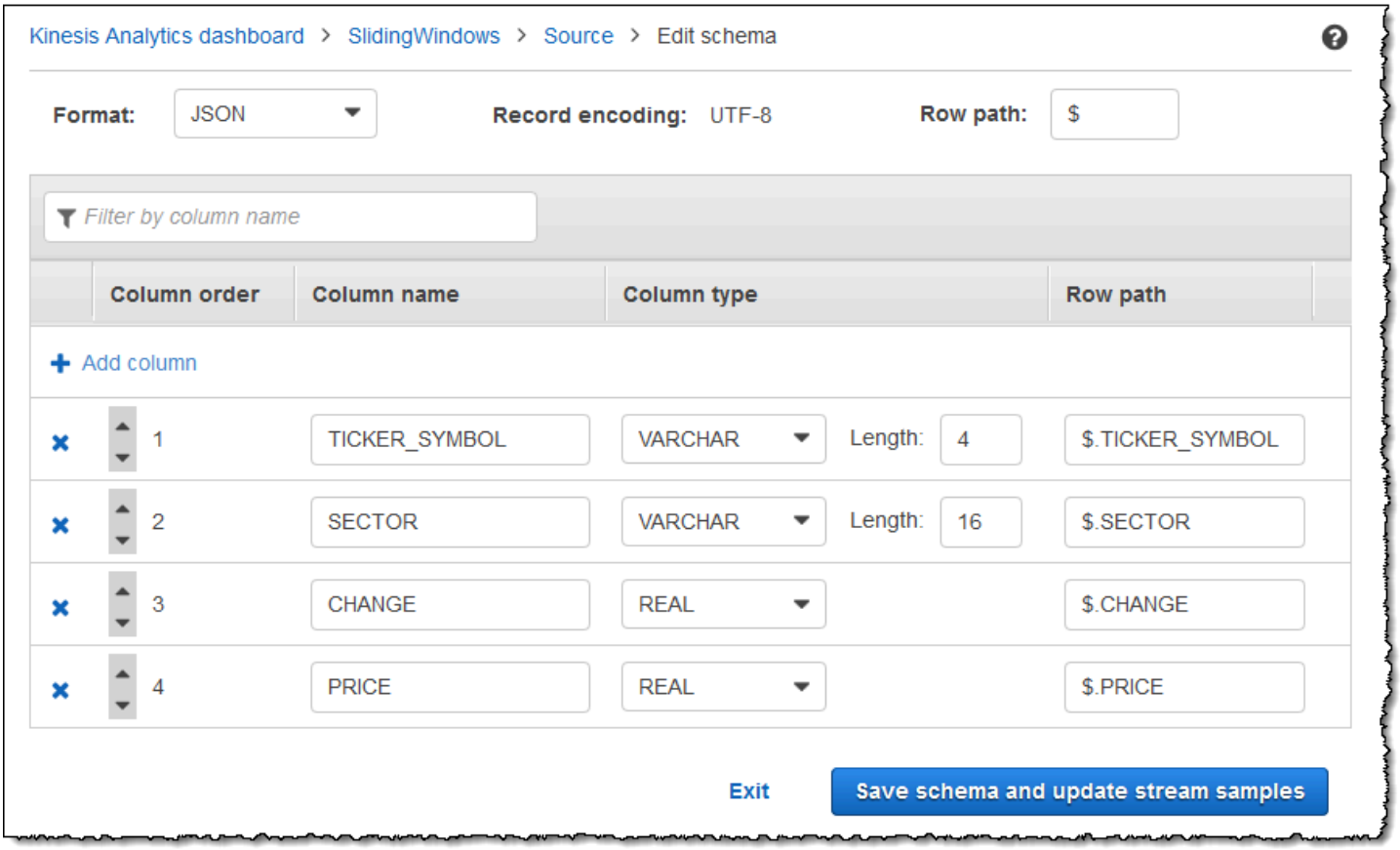
2. 列を削除するには、列番号の横にある [x] アイコンを選択します。



3. 列の名前を変更するには、[列名] に新しい名前を入力します。新しい列名は空白にすることはできません。必ず 2 文字以上にし、予約された SQL キーワードを含めることはできません。SQL の通常の識別子の命名基準を満たす必要があります。名前は必ず文字で開始し、文字、アンダースコア文字、数字のみ含めることができます。

4. 列のデータ型を変更するには、[Column type] で新しいデータ型を選択します。新しいデータ型が CHAR、VARBINARY、VARCHAR のいずれかの場合は、[Length (長さ)] にデータ長を指定します。詳細については、「データ型」を参照してください。
5. [Save schema and update stream] を選択して変更を保存します。

変更後のスキーマが以下のようにエディタに表示されます。



スキーマに行が多数ある場合は、[Filter by column name] を使用して行をフィルタリングすることができます。例えば、P から始まる列 (例: Price 列) の名前を編集するには、[列名によるフィルタリング] ボックスに P と入力します。

CSV スキーマの編集

CSV スキーマは次のステップを使用して編集できます。

CSV スキーマを編集するには

1. スキーマエディタで、[Row delimiter] の受信データストリームで使用されている区切り記号を選択します。つまり、ストリーム内のデータレコード間の区切り記号 (例: 改行文字) です。

2. [Column delimiter] で、受信データストリームで使用されている区切り記号を選択します。つまり、ストリーム内のデータフィールド間の区切り記号 (例: カンマ) です。
3. 列を追加するには、[Add column] を選択します。

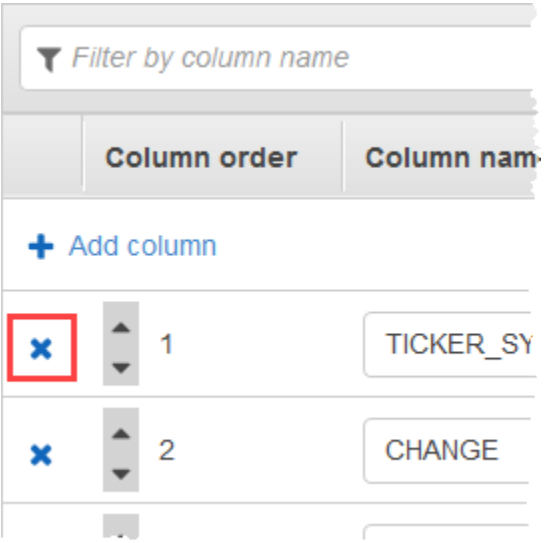
   新しい列が最初の列の位置に表示されます。列の順序を変更するには、列名の横にある上向き矢印と下向き矢印を選択します。

   新しい列に、以下の情報を入力します。

   - [Column name] に名前を入力します。

     列名は空白にすることはできません。必ず 2 文字以上にし、予約された SQL キーワードを含めることはできません。SQL の通常の識別子の命名基準を満たす必要があります。名前は必ず文字で開始し、文字、アンダースコア文字、数字のみ含めることができます。
   - [Column type] に、SQL データ型を入力します。

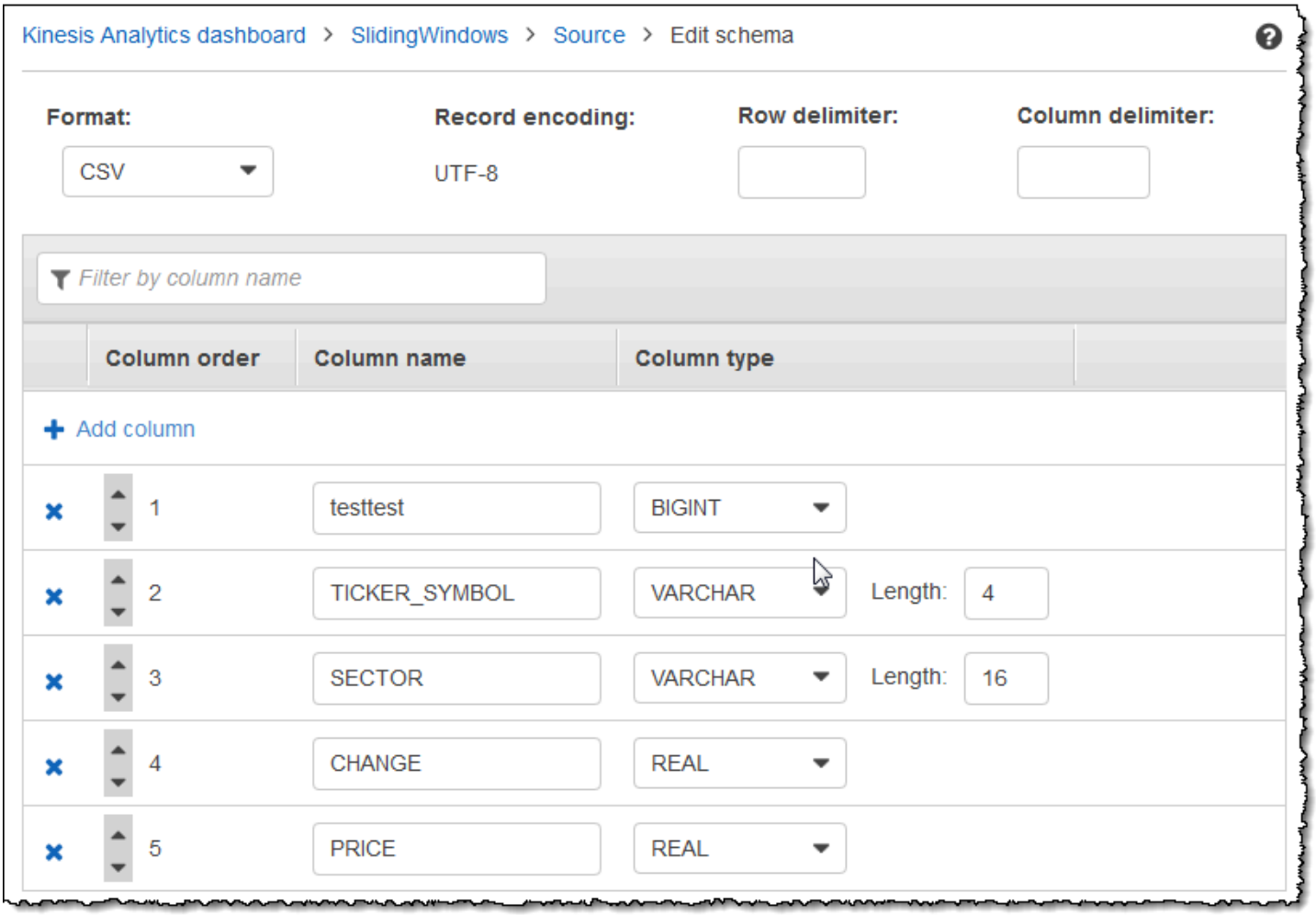
     列タイプは、サポートされている任意の SQL データ型です。新しいデータ型が CHAR、VARBINARY、VARCHAR のいずれかの場合は、[Length] にデータ長を指定します。詳細については、「データ型」を参照してください。
4. 列を削除するには、列番号の横にある [x] アイコンを選択します。



5. 列の名前を変更するには、[Column name] に新しい名前を入力します。新しい列名は空白にすることはできません。必ず 2 文字以上にし、予約された SQL キーワードを含めることはできません。SQL の通常の識別子の命名基準を満たす必要があります。名前は必ず文字で開始し、文字、アンダースコア文字、数字のみ含めることができます。

6. 列のデータ型を変更するには、[Column type] で新しいデータ型を選択します。新しいデータ型が CHAR、VARBINARY、VARCHAR のいずれかの場合は、[Length] にデータ長を指定します。詳細については、「データ型」を参照してください。
7. [Save schema and update stream] を選択して変更を保存します。

変更後のスキーマが以下のように エディタに表示されます。

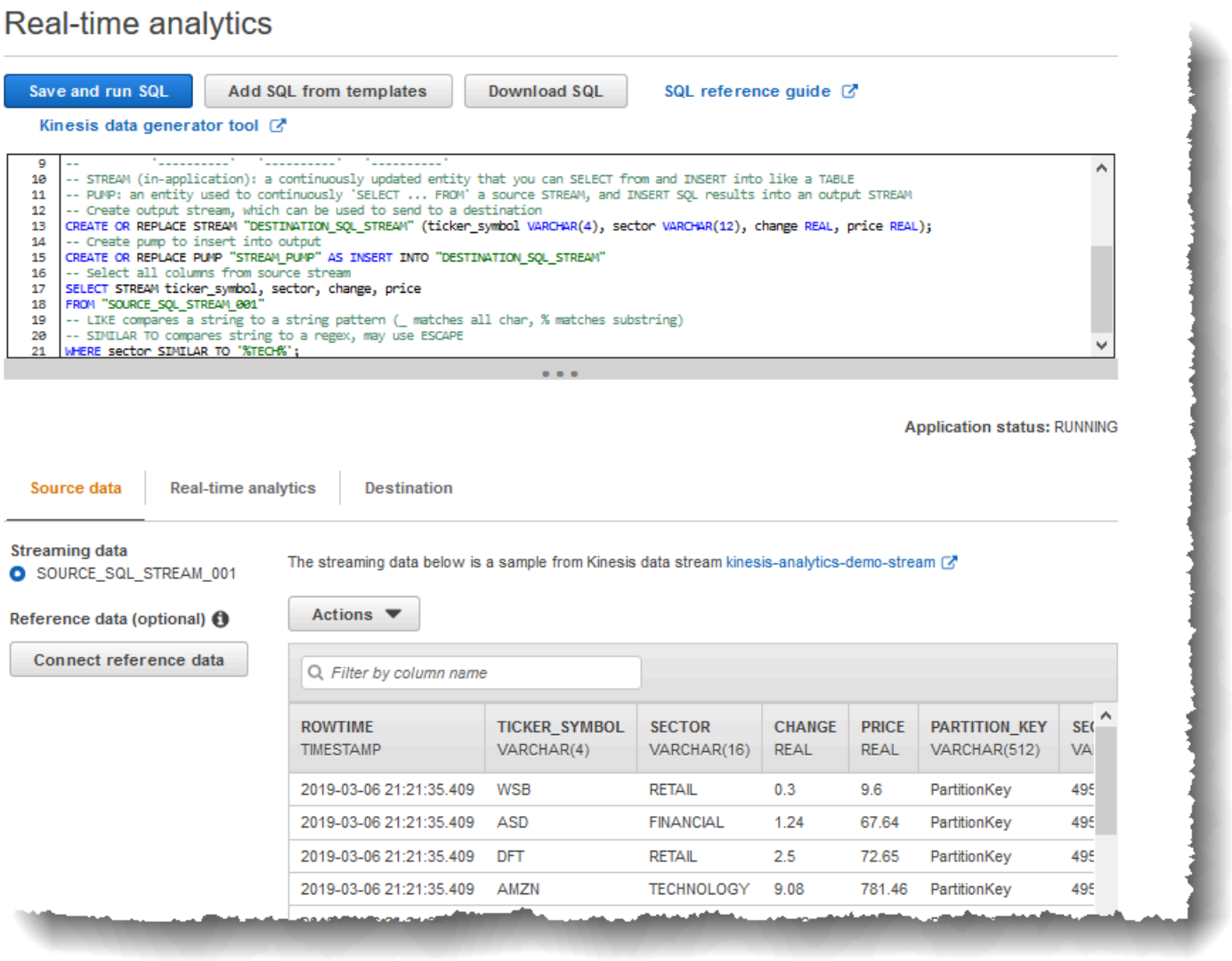


スキーマに行が多数ある場合は、[Filter by column name] を使用して行をフィルタリングすることができます。例えば、P から始まる列 (例: `Price` 列) の名前を編集するには、[列名によるフィルタリング] ボックスに P と入力します。

SQL エディタの使用

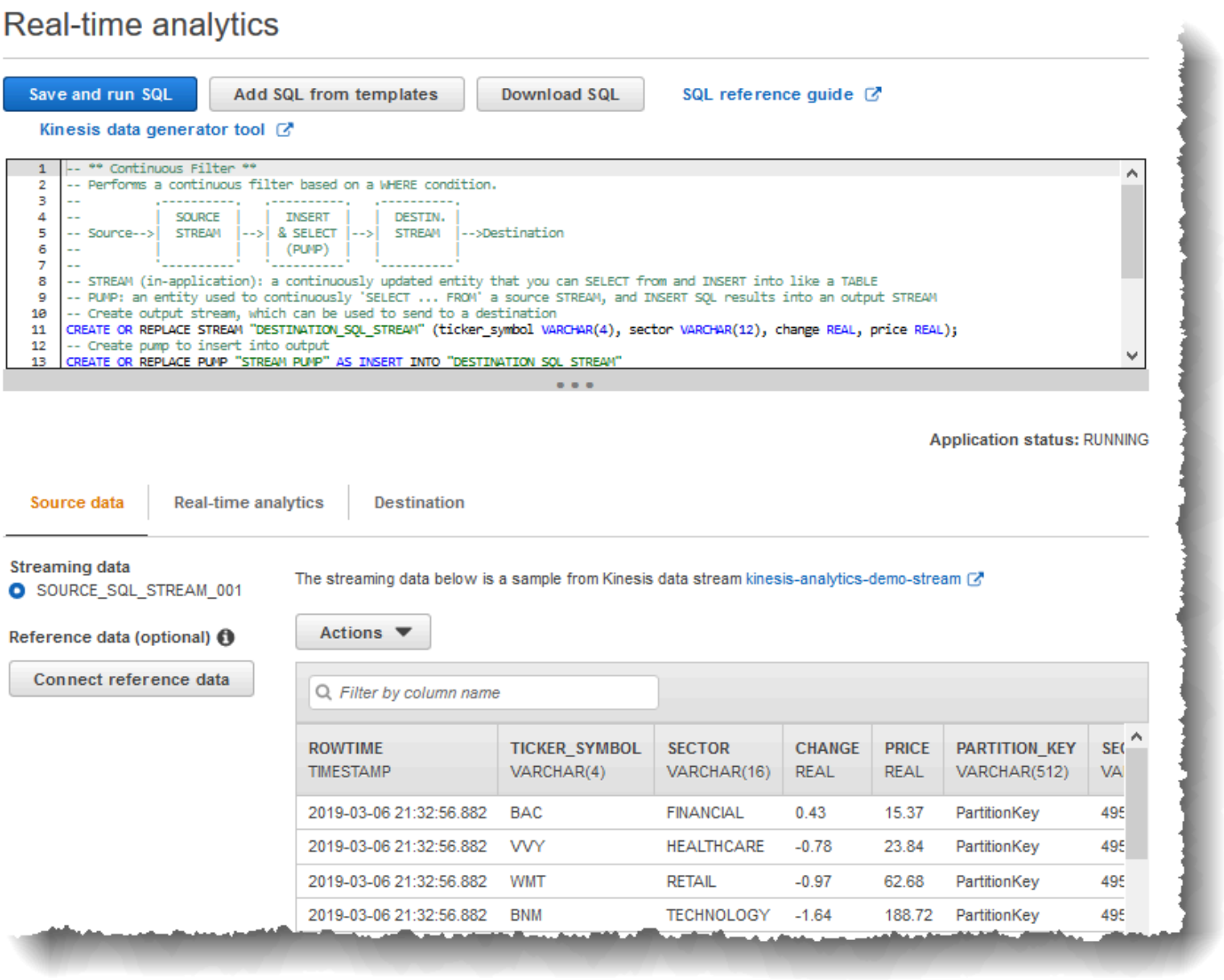
以下は、SQL エディタのセクションと各セクションの内容に関する情報です。SQL エディタで、独自のコードを自分で作成するか、[Add SQL from templates] を選択できます。SQL テンプレートには、一般的な Amazon Kinesis Data Analytics アプリケーションの記述に役立つサンプルの SQL コー

ドが記載されています。このガイドのサンプルアプリケーションでは、これらのテンプレートの一部を使用します。詳細については、「Kinesis Data Analytics for SQL の例」を参照してください。



[Source Data] タブ

[Source data] タブは、ストリーミングソースを識別します。また、このソースでマッピングされ、アプリケーション入力設定として提供されるアプリケーション内入力ストリームを識別します。



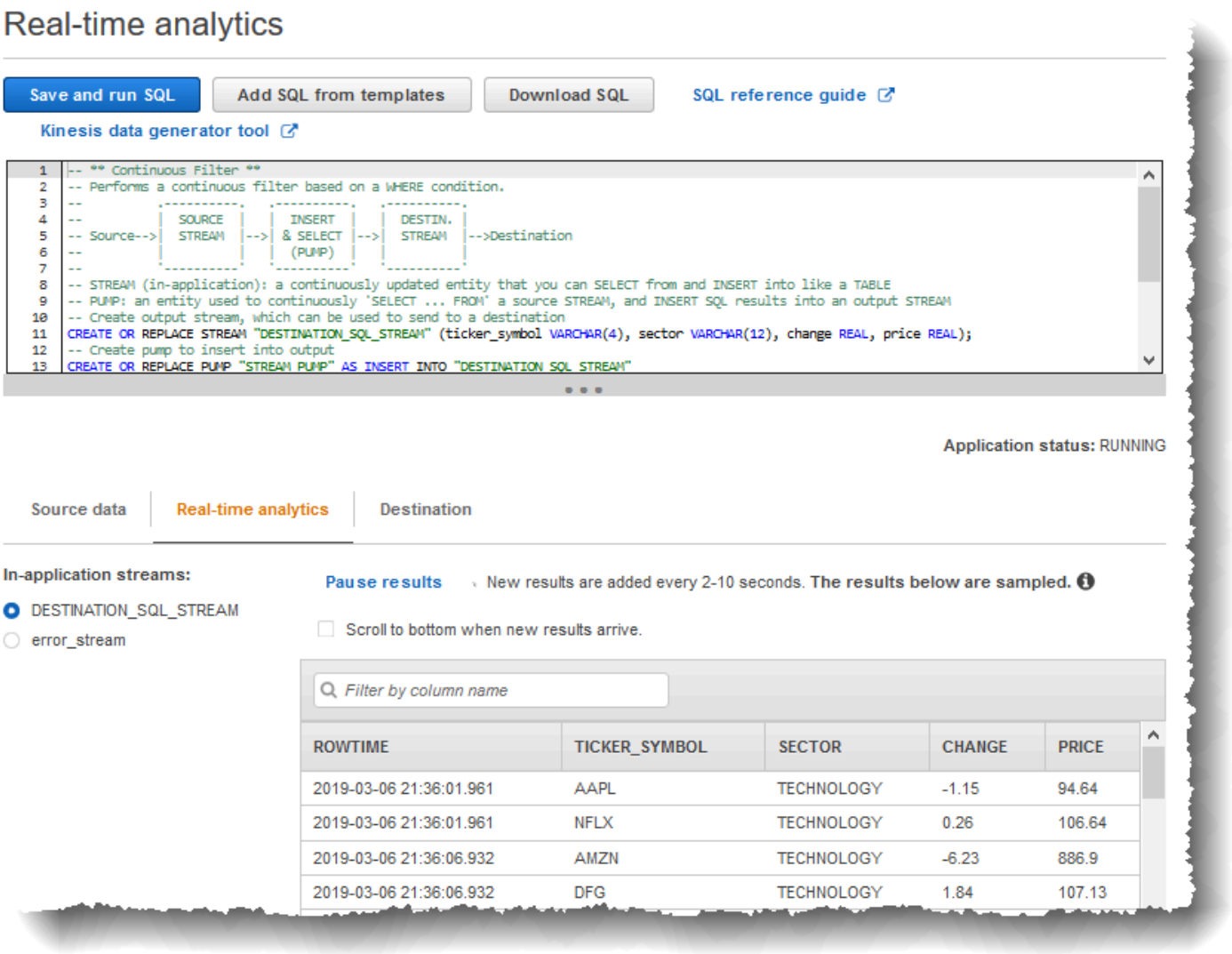
Amazon Kinesis Data Analytics は以下のタイムスタンプを提供しているため、入力設定で明示的にマッピングを指定する必要はありません。

- ROWTIME – アプリケーション内ストリームの各行には、`ROWTIME` という特殊な列があります。この列は、Kinesis Data Analytics が最初のアプリケーション内ストリームに行を挿入したときのタイムスタンプです。
- Approximate_Arrival_Time – ストリーミングソースのレコードには `Approximate_Arrival_Timestamp` 列が含まれています。これは、ストリーミングソースが正常に関連レコードを受信して保存したときに設定されるおおよその到達タイムスタンプです。Kinesis Data Analytics はこの列を `Approximate_Arrival_Time` としてアプリケーション内入力ストリームにフェッチします。Amazon Kinesis Data Analytics は、ストリーミングソースにマッピングされたアプリケーション内入力ストリームでのみこの列を提供します。

これらのタイムスタンプ値は時間ベースのウィンドウクエリで役に立ちます。詳細については、「ウィンドウクエリ」を参照してください。

[Real-Time Analytics] タブ

[Real-time analytics] タブに、アプリケーションコードによって作成されるすべてのアプリケーション内ストリームが表示されます。このストリームのグループには、Amazon Kinesis Data Analytics がすべてのアプリケーションで提供するエラーストリーム (`error_stream`) が含まれます。



[送信先] タブ

[送信先] タブでは、アプリケーション出力を設定して、外部宛先へのアプリケーション内ストリームを維持することができます。任意のアプリケーション内ストリームのデータを外部宛先で永続化する

ように出力を設定できます。詳細については、「アプリケーション出力の設定」を参照してください。

ストリーミング SQL の概念

Amazon Kinesis Data Analytics は、ANSI 2008 SQL 標準を拡張機能とともに実装しています。これらの拡張機能により、ストリーミングデータを処理できます。以下のトピックで、キーとなるストリーミング SQL の概念を取り上げます。

トピック

- アプリケーション内ストリームとポンプ
- タイムスタンプと ROWTIME 列
- 連続クエリ
- ウィンドウクエリ
- ストリーミングデータオペレーション: ストリーム結合

アプリケーション内ストリームとポンプ

アプリケーション入力を設定する際に、ストリーミングソースを作成済みのアプリケーション内ストリームにマッピングします。データは絶えずストリーミングソースからアプリケーション内ストリームに流れます。アプリケーション内ストリームは、SQL ステートメントを使用してクエリできるテーブルのように機能しますが、データが絶えず流れているためにストリームと呼ばれます。

> **Note**
>
> アプリケーション内ストリームを Amazon Kinesis データストリームや Firehose 配信ストリームと混同しないでください。アプリケーション内ストリームは、Amazon Kinesis Data Analytics アプリケーションのコンテキスト内のみに存在します。Kinesis のデータストリームと Firehose の配信ストリームは、アプリケーションからは独立した存在です。アプリケーションの入力設定におけるストリーミング送信元として、または出力設定における送信先として、これらを設定できます。

また、必要に応じて、中間クエリ結果を保存するためにアプリケーション内ストリームをさらに作成することもできます。アプリケーション内ストリームを作成する手順は、2 ステップです。まず、アプリケーション内ストリームを作成し、そこにデータをポンプします。たとえば、アプリケーションの入力設定で `INPUTSTREAM` という名前のアプリケーション内ストリームを作成するとします。次

の例では、別のストリーム (TEMPSTREAM) を作成して、INPUTSTREAM からそこにデータをポンプしています。

1. 次の図のように、3 つの列を持つアプリケーション内ストリーム (TEMPSTREAM) を作成します。

```
CREATE OR REPLACE STREAM "TEMPSTREAM" (
   "column1" BIGINT NOT NULL,
   "column2" INTEGER,
   "column3" VARCHAR(64));
```

   列名は引用符で指定され、大文字小文字を区別します。詳細については、Amazon Kinesis Data Analytics SQL Reference の「Identifiers」を参照してください。

2. ポンプを使用してストリームにデータを挿入します。ポンプとは、1 つのアプリケーション内ストリームから別のアプリケーション内ストリームにデータを挿入する、連続して実行される挿入クエリです。次のステートメントは、ポンプ (SAMPLEPUMP) を作成し、別のストリーム (INPUTSTREAM) からレコードを選択して、TEMPSTREAM にデータを挿入します。

```
CREATE OR REPLACE PUMP "SAMPLEPUMP" AS
INSERT INTO "TEMPSTREAM" ("column1",
                          "column2",
                          "column3")
SELECT STREAM inputcolumn1,
              inputcolumn2,
              inputcolumn3
FROM "INPUTSTREAM";
```

複数のライターから 1 つのアプリケーション内ストリームに挿入できます。また、ストリームから複数のリーダーを選択できます。アプリケーション内ストリームを、配信/購読メッセージングパラダイムの実装と考えることができます。このパラダイムでは、そこに含まれる作成時刻と受信時刻を含むデータ列はストリーミング SQL ステートメントのカスケードで処理、変換、転送でき、従来の RDBMS に保存する必要はありません。

アプリケーション内ストリームが作成された後は、通常の SQL クエリを実行できます。

> **Note**
> ストリームのクエリを実行するとき、ほとんどの SQL ステートメントは、行ベースまたは時間ベースのウィンドウを使用してバインドされます。詳細については、「ウィンドウクエリ」を参照してください。

また、ストリームを結合することもできます。ストリーム結合の例については、「ストリーミングデータオペレーション: ストリーム結合」を参照してください。

タイムスタンプと ROWTIME 列

アプリケーション内ストリームには、ROWTIME という特別な行が含まれています。Amazon Kinesis Data Analytics によって最初のアプリケーション内ストリームに行が挿入されると、タイムスタンプが保存されます。ROWTIME は、Amazon Kinesis Data Analytics がストリーミングソースからレコードを読み取った後、最初のアプリケーション内ストリームにレコードを挿入した時点のタイムスタンプを反映します。この ROWTIME 値はその後、アプリケーション全体で維持されます。

> **Note**
> 1 つのアプリケーション内ストリームから別のアプリケーション内ストリームにレコードをポンプする際に、ROWTIME 列を明示的にコピーする必要はありません。この列は Amazon Kinesis Data Analytics でコピーされます。

Amazon Kinesis Data Analytics は、ROWTIME の値が一定間隔で増加することを保証します。このタイムスタンプは、時間ベースウィンドウのクエリで使用されます。詳細については、「ウィンドウクエリ」を参照してください。

ROWTIME 列には、アプリケーション内ストリームの他の列と同様に、SELECT ステートメント内でアクセスできます。以下に例を示します。

```
SELECT STREAM ROWTIME,
              some_col_1,
              some_col_2
FROM  SOURCE_SQL_STREAM_001
```

ストリーミング分析でのさまざまな時間を理解する

ROWTIME の他に、リアルタイムストリーミングアプリケーションには別のタイプの時間があります。次のようなものがあります。

- イベント時間 – イベントが発生したときのタイムスタンプ。クライアント側の時間と呼ばれることもあります。イベントが発生した時間であるため、分析でこの時間を使用するのが望ましい場合がよくあります。しかし、携帯電話やウェブクライアントなど多くのイベントソースは信頼性の高い時計を持たないため、時間が不正確になる場合があります。さらに、接続性の問題で、レコードがイベントの発生と同じ順序でストリームに現れない場合があります。

- 取り込み時間 — レコードがストリーミングソースに追加されたときのタイムスタンプ。Amazon Kinesis Data Streams は、このタイムスタンプを提供する APPROXIMATE_ARRIVAL_TIME というフィールドをすべてのレコードに含んでいます。サーバー側の時間と呼ばれることもあります。取り込み時間は、多くの場合、イベント時間にかなり近い近似値です。ストリームへのレコード取り込みに何らかの遅延が発生した場合は不正確になることがありますが、通常は稀なケースです。また、取り込み時間の順序が入れ替わることはめったにありません。ただし、ストリーミングデータの分散特性のために発生する可能性はあります。そのため、取り込み時間はイベント時間をもっとも正確に順序正しく反映しています。

- 処理時間 — Amazon Kinesis Data Analytics が最初のアプリケーション内ストリームに行を挿入したときのタイムスタンプ。Amazon Kinesis Data Analytics は、このタイムスタンプを各アプリケーション内ストリームに存在する ROWTIME 列に提供します。処理時間は常に一定間隔で増加しています。ただし、アプリケーションが遅れている場合は正確ではありません。(アプリケーションが遅れた場合、処理時間がイベント時間を正確に反映しなくなります)。この ROWTIME は経過時間に関しては正確ですが、実際にイベントが発生した時間ではない場合があります。

時間ベースのウィンドウクエリでこれらの時間を使用するには、それぞれ利点と欠点があります。これらの時間を 1 つ以上選択し、またそれに伴う欠点に対処する戦略をお客様のユースケースシナリオに基づいて選択することをお勧めします。

> **Note**
>
> 行ベースのウィンドウを使用する場合は、時刻は問題ではないため、このセクションは無視してかまいません。

ROWTIME と他の時間 (取り込み時間またはイベント時間) の 2 つの時間ベースを両方使用した 2 ウィンドウ戦略をお勧めします。

- 次の例に示すように、クエリで結果を発行する頻度を制御する ROWTIME を最初のウィンドウとして使用します。論理時間としては使用されません。
- 分析に関連付ける論理時間であるその他の時間のうち 1 つを使用します。この時間は、いつイベントが発生したかを示します。次の例では、分析の目的はレコードをグループ化し、ティッカーでカウントを返すことです。

この戦略の利点は、イベントが発生したときを示す時間を使用できることです。アプリケーションが遅れたときやイベントの到達順序が入れ替わったときに適切に処理できます。アプリケーション内ストリームにレコードを持ってくるときにアプリケーションが遅れた場合でも、2 番目のウィンドウの論理時間でグループ化されます。クエリは ROWTIME を使用して処理順序を保証します。遅延したレコード (取り込みタイムスタンプの値が ROWTIME 値よりも早い) も正常に処理されます。

「使用開始」実習で使用されているデモストリームに対して次のクエリを検討します。クエリは GROUP BY 句を使用し、1 分ごとのタンブリングウィンドウでティッカーカウントを発行します。

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM"
    ("ingest_time"    timestamp,
    "APPROXIMATE_ARRIVAL_TIME" timestamp,
    "ticker_symbol"  VARCHAR(12),
    "symbol_count"        integer);


CREATE OR REPLACE PUMP "STREAM_PUMP" AS
    INSERT INTO "DESTINATION_SQL_STREAM"
    SELECT STREAM STEP("SOURCE_SQL_STREAM_001".ROWTIME BY INTERVAL '60' SECOND) AS
 "ingest_time",
        STEP("SOURCE_SQL_STREAM_001".APPROXIMATE_ARRIVAL_TIME BY INTERVAL '60' SECOND)
 AS "APPROXIMATE_ARRIVAL_TIME",
        "TICKER_SYMBOL",
        COUNT(*) AS "symbol_count"
    FROM "SOURCE_SQL_STREAM_001"
    GROUP BY "TICKER_SYMBOL",
        STEP("SOURCE_SQL_STREAM_001".ROWTIME BY INTERVAL '60' SECOND),
        STEP("SOURCE_SQL_STREAM_001".APPROXIMATE_ARRIVAL_TIME BY INTERVAL '60' SECOND);
```

GROUP BY で、まず 1 分ごとのウィンドウの ROWTIME に基づいて、次に APPROXIMATE_ARRIVAL_TIME に基づいてレコードをグループ化します。

結果のタイムスタンプ値は、最も近い 60 秒間隔で切り捨てられます。クエリによって発行された最初のグループ結果が、最初の 1 分間のレコードを示しています。発行された 2 つめの結果グループは、ROWTIME に基づいた次の分単位のレコードを示しています。最後のレコードは、アプリケーションで、アプリケーション内ストリームにレコードを持ってくるのが後れたことを示します (取り込みタイムスタンプに対して、ROWTIME 値が遅れていることを示します)。

```
ROWTIME                   INGEST_TIME     TICKER_SYMBOL  SYMBOL_COUNT

--First one minute window.
2016-07-19 17:05:00.0    2016-07-19 17:05:00.0    ABC      10
2016-07-19 17:05:00.0    2016-07-19 17:05:00.0    DEF      15
2016-07-19 17:05:00.0    2016-07-19 17:05:00.0    XYZ      6
--Second one minute window.
2016-07-19 17:06:00.0    2016-07-19 17:06:00.0    ABC      11
2016-07-19 17:06:00.0    2016-07-19 17:06:00.0    DEF      11
2016-07-19 17:06:00.0    2016-07-19 17:05:00.0    XYZ      1  ***

***late-arriving record, instead of appearing in the result of the
first 1-minute windows (based on ingest_time, it is in the result
of the second 1-minute window.
```

ダウンストリームデータベースに結果をプッシュすることで、最終的な 1 分あたりの正確なカウントを得るために結果を 1 つにできます。例えば、Amazon Redshift テーブルに書き込む Firehose 配信ストリームに結果を永続化するように、アプリケーション出力を設定できます。結果が Amazon Redshift テーブルに書き込まれた後は、テーブルにクエリして Ticker_Symbol によってカウントグループの総数をコンピューティングできます。XYZ の場合、レコードが遅延したとしても総数は正確 (6+1) です。

連続クエリ

ストリーム上のクエリは、ストリーミングデータに対して連続して実行されます。この連続実行によって、アプリケーションが連続してストリーミングにクエリしアラートを生成する機能などのシナリオが可能になります。

「使用開始」の実習では、SOURCE_SQL_STREAM_001 という名前のアプリケーション内ストリームを使用します。これはデモストリーム (Kinesis データストリーム) から連続して株価を受信します。スキーマは次のとおりです。

```
(TICKER_SYMBOL VARCHAR(4),
```

```
 SECTOR varchar(16),
 CHANGE REAL,
 PRICE REAL)
```

15 パーセントを超える株価の変動に関心があるとします。アプリケーションコードで次のクエリを使用できます。このクエリは連続して実行され、15 パーセントを超える株価の変動が検出された場合にレコードを発行します。

```
SELECT STREAM TICKER_SYMBOL, PRICE
      FROM   "SOURCE_SQL_STREAM_001"
      WHERE  (ABS((CHANGE / (PRICE-CHANGE)) * 100)) > 15
```

次の手順を使用して Amazon Kinesis Data Analytics アプリケーションをセットアップし、このクエリをテストします。

クエリをテストするには

1. 「使用開始」実習に従ってアプリケーションを作成します。
2. アプリケーションコード内の SELECT ステートメントを前述の SELECT クエリに置き換えます。アプリケーションコードは次のようになります。

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (ticker_symbol VARCHAR(4),
                                                   price DOUBLE);
-- CREATE OR REPLACE PUMP to insert into output
CREATE OR REPLACE PUMP "STREAM_PUMP" AS
  INSERT INTO "DESTINATION_SQL_STREAM"
      SELECT STREAM TICKER_SYMBOL,
                    PRICE
      FROM   "SOURCE_SQL_STREAM_001"
      WHERE  (ABS((CHANGE / (PRICE-CHANGE)) * 100)) > 15;
```

ウィンドウクエリ

アプリケーションコードの SQL クエリはアプリケーション内ストリームに対して連続で実行されます。アプリケーション内ストリームとは、アプリケーション内を常時流れる未バインドのデータのことです。したがって、常時更新されているこの入力から結果セットを得るために、時間と行の条件で定義されるウィンドウを使用してクエリをバインドする場合が多くあります。これらはウィンドウ SQL とも呼ばれます。

時間ベースのウィンドウクエリの場合は、ウィンドウのサイズを時間で (たとえば、1 分のウィンドウ) 指定します。これには、一定間隔で増加するアプリケーション内ストリームにタイムスタンプ列が必要です。(新しい行のタイムスタンプが前の行と同じまたは前の行より大きい)。Amazon Kinesis Data Analytics は、各アプリケーション内ストリームに `ROWTIME` というタイムスタンプ列を提供します。時間ベースのクエリを指定するとき、この列を使用できます。アプリケーションで、他のタイムスタンプオプションを選択する場合もあります。詳細については、「タイムスタンプと ROWTIME 列」を参照してください。

行ベースのウィンドウクエリの場合は、列数の条件でウィンドウサイズを指定します。

アプリケーションの必要に応じて、タンブリングウィンドウ、スライディングウィンドウ、またはずらしウィンドウ方式でレコードを処理するクエリを指定できます。Kinesis Data Analytics では、次のウィンドウタイプがサポートされています。

- Stagger Windows: データが届くと開く、キー付けされた時間ベースのウィンドウを使用してデータを集計するクエリ。キーによって、複数の重なり合うウィンドウが可能になります。タンブリングウィンドウと比較すると、Stagger Windows は遅延データまたは順序通りでないデータを削減するため、これは、時間ベースのウィンドウを使用してデータを集約する方法として推奨されます。
- タンブリングウィンドウ: 定期的に開閉する、個別の時間ベースのウィンドウを使用してデータを集計するクエリ。
- スライディングウィンドウ: 固定時間または rowcount 間隔を使用して、データを継続的に集計するクエリ。

Stagger Windows

ずらしウィンドウは、一貫性のない時間に届くデータのグループを分析するのに適したウィンドウ処理メソッドです。これは、関連する一連のセールスやログレコードなど、時系列分析のユースケースに適しています。

たとえば、VPC フローログには約 10 分のキャプチャウィンドウがあります。しかし、クライアントにデータを集約する場合は最大 15 分のキャプチャウィンドウを持つことができます。ずらしウィンドウは、これらのログを分析のために集計するのに理想的です。

ずらしウィンドウでは、タンブリングウィンドウが使用されたときなど、同じ時間制限付きウィンドウに収まらない関連レコードの問題を解決します。

Tumbling Windows の部分的な結果

遅延データまたは順序通りでないデータの集約に タンブリングウィンドウ を使用する場合、一定の制限があります。

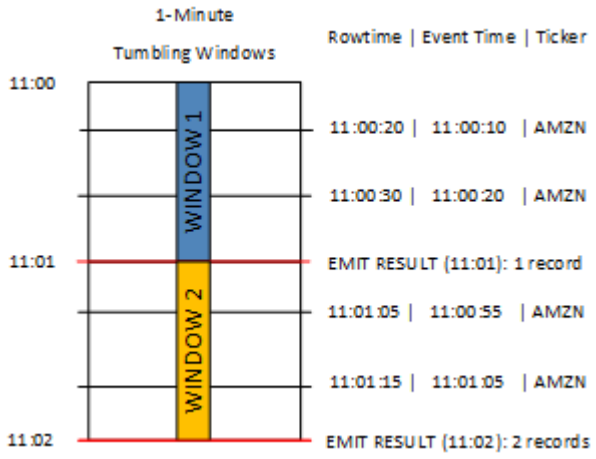
時間関連のデータのグループを分析するためにタンブリングウィンドウを使用する場合、個々のレコードは別々のウィンドウに分類される可能性があります。したがって、各ウィンドウの部分的な結果を後で組み合わせて、各レコードグループの完全な結果を得る必要があります。

次のタンブリングウィンドウクエリでは、レコードは行時間、イベント時間、およびティッカーシンボルによってウィンドウにグループ化されます。

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
    TICKER_SYMBOL VARCHAR(4),
    EVENT_TIME timestamp,
    TICKER_COUNT     DOUBLE);

CREATE OR REPLACE PUMP "STREAM_PUMP" AS
  INSERT INTO "DESTINATION_SQL_STREAM"
    SELECT STREAM
        TICKER_SYMBOL,
        FLOOR(EVENT_TIME TO MINUTE),
        COUNT(TICKER_SYMBOL) AS TICKER_COUNT
    FROM "SOURCE_SQL_STREAM_001"
    GROUP BY ticker_symbol, FLOOR(EVENT_TIME TO MINUTE),
 STEP("SOURCE_SQL_STREAM_001".ROWTIME BY INTERVAL '1' MINUTE);
```

次の図でアプリケーションは、1 分の粒度で取引が発生したとき (イベント時間) に基づき、受信した取引の数をカウントしています。アプリケーションは行時間とイベント時間に基づき、タンブリングウィンドウを使用してデータをグループ化できます。アプリケーションは、すべてが 1 分間隔で届く 4 つのレコードを受け取ります。次に、行時間、イベント時間、およびティッカーシンボルでレコードをグループ化します。レコードの一部は最初のタンブリングウィンドウが終了してから届くため、すべてのレコードが同じ 1 分のタンブリングウィンドウに収まるわけではありません。



前述の図には、以下のイベントが含まれます。

ROWTIME	EVENT_TIME	TICKER_SYMBOL
11:00:20	11:00:10	AMZN
11:00:30	11:00:20	AMZN
11:01:05	11:00:55	AMZN
11:01:15	11:01:05	AMZN

タンブリングウィンドウアプリケーションからの結果セットは、以下のようになります。

ROWTIME	EVENT_TIME	TICKER_SYMBOL	COUNT
11:01:00	11:00:00	AMZN	2
11:02:00	11:00:00	AMZN	1
11:02:00	11:01:00	AMZN	1

前述の結果では、3 つの結果が返されます。

- 最初の 2 つのレコードを集計する、`ROWTIME` が 11:01:00 のレコード。

- 3 つ目のレコードのみを集計する、11:02:00 のレコード。このレコードは、2 番目のウィンドウ内に ROWTIME がありますが、EVENT_TIME は 1 番目のウィンドウ内にあります。
- 4 つ目のレコードのみを集計する、11:02:00 のレコード。

完全な結果セットを分析するには、レコードが永続的なストアに集約されている必要があります。これにより、アプリケーションに複雑性と処理要件が加わります。

Stagger Windows での完全な結果

時間関連のデータレコードの分析の精度を向上させるため、Kinesis Data Analytics ではずらしウィンドウという新しいウィンドウのタイプを提供しています。このウィンドウタイプでは、パーティションキーに一致する最初のイベントが届いたときにウィンドウが開きます。固定の時間間隔でウィンドウが開くことはありません。ウィンドウは、ウィンドウを開いたときから測定される、指定された経過時間に基づいて閉じます。

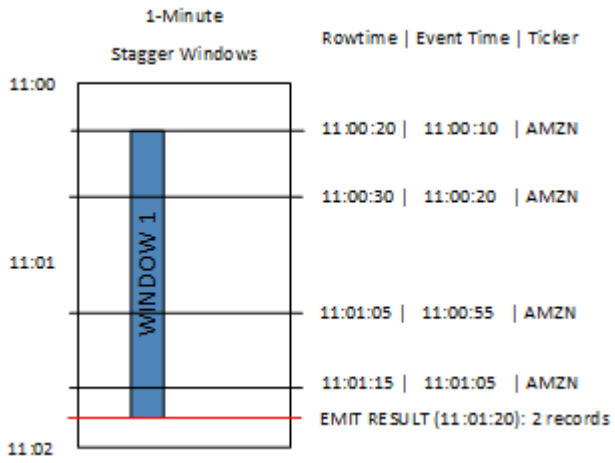
ずらしウィンドウは、ウィンドウ句の各キーグループのための、別個の時間制限付きウィンドウです。アプリケーションは、すべての結果に対して単一のウィンドウを使用するのではなく、独自の時間ウィンドウ内にウィンドウ句のそれぞれの結果を集計します。

次のずらしウィンドウクエリでは、レコードはイベント時間、およびティッカーシンボルによってウィンドウにグループ化されます。

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
    ticker_symbol    VARCHAR(4),
    event_time       TIMESTAMP,
    ticker_count     DOUBLE);

CREATE OR REPLACE PUMP "STREAM_PUMP" AS
  INSERT INTO "DESTINATION_SQL_STREAM"
    SELECT STREAM
        TICKER_SYMBOL,
        FLOOR(EVENT_TIME TO MINUTE),
        COUNT(TICKER_SYMBOL) AS ticker_count
    FROM "SOURCE_SQL_STREAM_001"
    WINDOWED BY STAGGER (
            PARTITION BY FLOOR(EVENT_TIME TO MINUTE), TICKER_SYMBOL RANGE INTERVAL '1'
 MINUTE);
```

次の図では、イベントはイベント時間、およびティッカーシンボルによってずらしウィンドウに集計されます。



前述の図には、タンブリングウィンドウアプリケーションが分析したものと同じイベントである、以下のイベントが含まれています。

ROWTIME	EVENT_TIME	TICKER_SYMBOL
11:00:20	11:00:10	AMZN
11:00:30	11:00:20	AMZN
11:01:05	11:00:55	AMZN
11:01:15	11:01:05	AMZN

ずらしウィンドウアプリケーションからの結果セットは、以下のようになります。

ROWTIME	EVENT_TIME	TICKER_SYMBOL	カウント
11:01:20	11:00:00	AMZN	3
11:02:15	11:01:00	AMZN	1

返されたレコードは、最初の 3 つの入力レコードを集計します。レコードは、1 分間のずらしウィンドウでグループ化されます。ずらしウィンドウは、アプリケーションが最初の AMZN レコード (ROWTIME が 11:00:20 のもの) を受信したときに開始されます。1 分間のずらしウィンドウが終了す

ると (11:01:20)、ずらしウィンドウ内に収められる結果 (ROWTIME および EVENT_TIME に基づく) が、出力ストリームに書き込まれます。ずらしウィンドウを使用すると、1 分間ウィンドウ内にある ROWTIME および EVENT_TIME を持つすべてのレコードが 1 つの結果として出力されます。

最後のレコード (1 分間集計から外れた EVENT_TIME がある) は別々に集計されます。これは、レコードを結果セットに分割するために使用されるパーティションキーの 1 つが EVENT_TIME であり、最初のウィンドウの EVENT_TIME のパーティションキーが 11:00 であるためです。

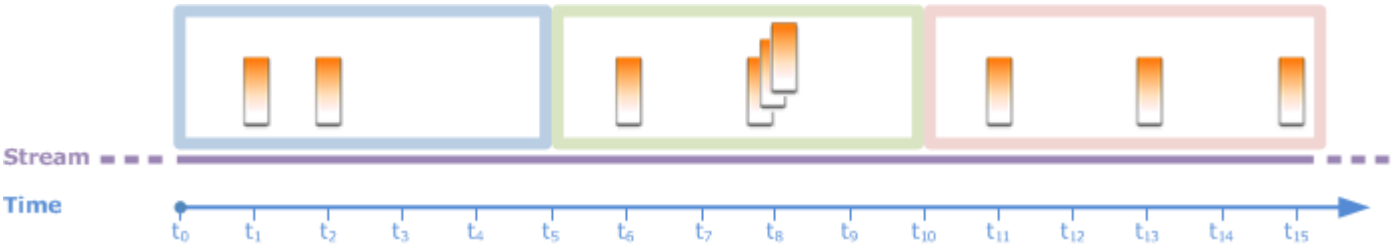
ずらしウィンドウの構文は、WINDOWED BY という特別な句で定義されています。この句は、ストリーミング集計の GROUP BY 句の代わりに使用されます。この句は、オプションの WHERE 句の直後、および HAVING 句の前に表示されます。

ずらしウィンドウは、WINDOWED BY 句で定義され、パーティションキーとウィンドウ長の 2 つのパラメータを取ります。パーティションキーは、受信データストリームを分割し、ウィンドウが開いたときに定義します。ずらしウィンドウは、固有のパーティションキーを持つ最初のイベントがストリームに表示されたとき開きます。ずらしウィンドウは、ウィンドウ長により定義された一定期間の後で閉じます。次のコード例にその構文を示します。

```
...
FROM <stream-name>
WHERE <... optional statements...>
WINDOWED BY STAGGER(
 PARTITION BY <partition key(s)>
 RANGE INTERVAL <window length, interval>
);
```

タンブリングウィンドウ (GROUP BY を使用した集計)

ウィンドウクエリが各ウィンドウを重複しない方式で処理する場合、ウィンドウはタンブリングウィンドウと呼ばれます。この場合、アプリケーション内ストリームの各レコードは特定のウィンドウに属します。これは 1 回 (そのレコードが属するウィンドウをクエリが処理するとき) のみ処理されます。



たとえば、GROUP BY 句を使用した集計クエリは、タンブリングウィンドウの行を処理します。「使用開始」実習のデモストリームは、アプリケーションのアプリケーション内ストリーム SOURCE_SQL_STREAM_001 にマッピングされた株価データを受信します。このストリームには、次のスキーマがあります。

```
(TICKER_SYMBOL VARCHAR(4),
 SECTOR varchar(16),
 CHANGE REAL,
 PRICE REAL)
```

アプリケーションコードで、1 分のウィンドウに対して各ティッカーでの合計 (最低、最高) 価格を検索するとします。以下のクエリを使用できます。

```
SELECT STREAM ROWTIME,
              Ticker_Symbol,
              MIN(Price) AS Price,
              MAX(Price) AS Price
FROM     "SOURCE_SQL_STREAM_001"
GROUP BY Ticker_Symbol,
         STEP("SOURCE_SQL_STREAM_001".ROWTIME BY INTERVAL '60' SECOND);
```

上記は、時間ベースのウィンドウクエリの例です。クエリは、ROWTIME 値でレコードをグループ化します。分単位でレポートするために、STEP 関数は ROWTIME 値を直近の分に四捨五入します。

> **Note**
>
> また、FLOOR 関数を使用してレコードをウィンドウにグループ化することもできます。ただし、FLOOR は時間値を時間単位 (時間、分、秒など) に丸めることのみできます。STEP は、値を任意の間隔 (たとえば 30 秒など) に丸めることができるため、レコードをタンブリングウィンドウにグループ化する場合に使用することをお勧めします。

このクエリは、重複しない (タンブリング) ウィンドウの例です。GROUP BY 句によって、レコードが 1 分のウィンドウにグループ化されます。各レコードは特定のウィンドウに属します (重複しない)。クエリでは、1 分ごとに 1 つの出力レコードが発行され、特定の分にレコードされた最低/最高ティッカー価格が提供されます。このタイプのクエリは、入力データストリームから定期的にレポートを生成する場合に便利です。この例では、1 分ごとにレポートが生成されます。

クエリをテストするには

1. 「使用開始」実習に従ってアプリケーションをセットアップします。
2. アプリケーションコード内の SELECT ステートメントを前述の SELECT クエリに置き換えます。アプリケーションコードは次のようになります。

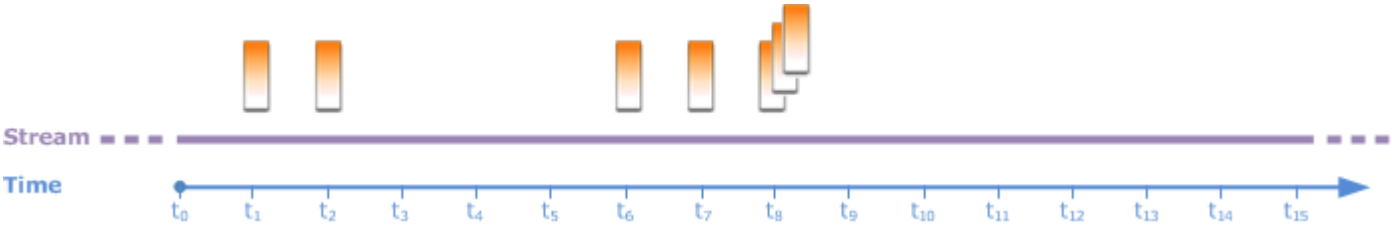
```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
                                  ticker_symbol VARCHAR(4),
                                  Min_Price     DOUBLE,
                                  Max_Price     DOUBLE);
-- CREATE OR REPLACE PUMP to insert into output
CREATE OR REPLACE PUMP "STREAM_PUMP" AS
  INSERT INTO "DESTINATION_SQL_STREAM"
    SELECT STREAM Ticker_Symbol,
                  MIN(Price) AS Min_Price,
                  MAX(Price) AS Max_Price
    FROM    "SOURCE_SQL_STREAM_001"
    GROUP BY Ticker_Symbol,
             STEP("SOURCE_SQL_STREAM_001".ROWTIME BY INTERVAL '60' SECOND);
```

スライディングウィンドウ

GROUP BY を使用してレコードをグループ化する代わりに、時間ベースまたは行ベースのウィンドウを定義できます。そのためには、WINDOW 句を明示的に追加します。

この場合、ウィンドウが時間と共にスライドしながら、新しいレコードがストリームに現れると Amazon Kinesis Data Analytics が出力を発行します。Kinesis Data Analytics は、ウィンドウの行を処理して出力を発行します。このタイプの処理ではウィンドウが重複するだけでなく、レコードが複数のウィンドウの一部となり、ウィンドウごとに処理される場合があります。次の例では、スライディングウィンドウについて説明します。

ストリームのレコードをカウントする簡単なクエリを考えます。この例では、5 秒のウィンドウを前提としています。次のストリームの例では、新しいレコードが t_1、t_2、t_6、t_7 の時間に受信され、t_8 秒では同時に 3 つのレコードを受信しています。



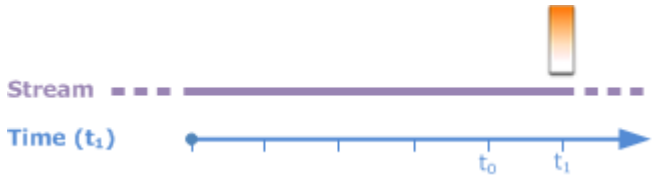
以下に留意してください。

- この例では、5 秒のウィンドウを前提としています。5 秒ウィンドウは時間とともに継続的にスライドします。
- 行がウィンドウに入力されるごとに、出力行がスライディングウィンドウによって発行されます。アプリケーションを起動してすぐは、まだ 5 秒のウィンドウが経過していなくても、ストリームで受信された新しいレコードのそれぞれに対してクエリが出力を発行します。たとえば、1 秒目と 2 秒目にレコードが現れると、クエリは出力を発行します。その後、クエリは 5 秒ウィンドウでレコードを処理します。
- ウィンドウは時間とともにスライドします。古いレコードがウィンドウから押し出されても、その 5 秒ウィンドウに含まれるストリームに新しいレコードがない限り、クエリは出力を発行しません。

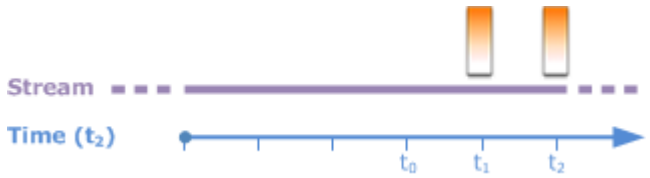
クエリが t_0 に実行を開始するとします。そして次のようになります。

1. t_0 にクエリが開始されます。この時点ではレコードがないため、クエリは出力 (カウント値) を発行しません。

2. 時間 t_1 に、新しいレコードがストリームに現れ、クエリはカウント値 1 を発行します。

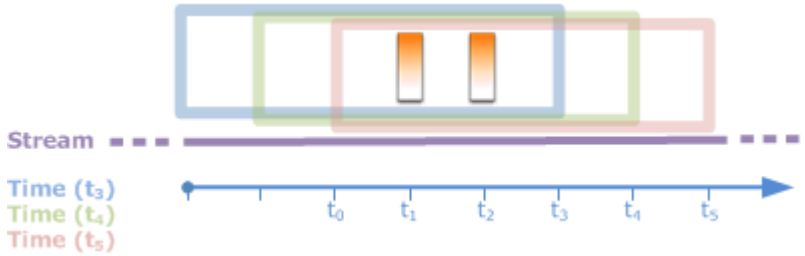


3. 時間 t_2 に、別のレコードが現れ、クエリはカウント値 2 を発行します。

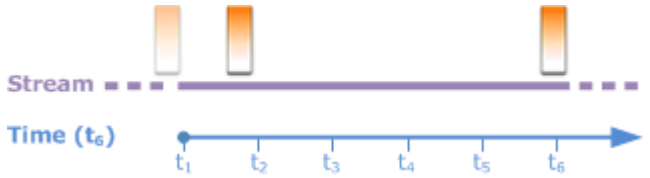


4. 5 秒ウィンドウは時間とともにスライドします。
   - t_3 では、スライディングウィンドウは t_3 から t_0 です。
   - t4 (スライディングウィンドウは t_4 から t_0)
   - t_5 では、スライディングウィンドウは t_5 から t_0 です。

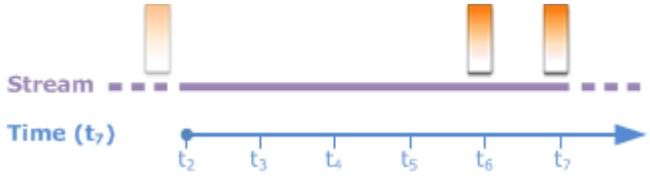
   この間、5 秒ウィンドウのレコードはまったく同じです。新規レコードはありません。そのため、クエリは出力を発行しません。



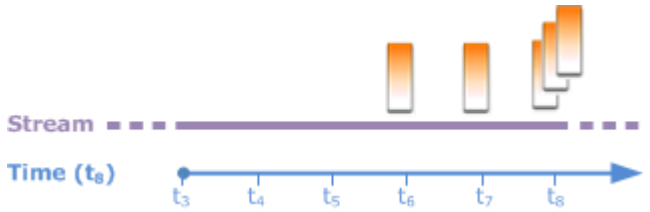
5. t_6 時、5 秒ウィンドウは (t_6 から t_1) です。クエリは、t_6 で新しいレコードを検出するため、出力 2 を発行します。t_1 のレコードはウィンドウ内になくなったため、カウントされません。



6. t_7 時、5 秒ウィンドウは (t_7 から t_2) です。クエリは、t_7 で新しいレコードを検出するため、出力 2 を発行します。t_2 のレコードは 5 秒ウィンドウ内になくなったため、カウントされません。



7. t_8 時、5 秒ウィンドウは (t_8 から t_3) です。クエリが 3 つの新しいレコードを検出したため、レコードカウント 5 を発行します。



要約すると、このウィンドウは固定サイズであり、時間とともにスライドします。クエリは新しいレコードが現れたときに出力を発行します。

> **Note**
>
> スライディングウィンドウの使用は1 時間以内にすることをお勧めします。これよりも長いウィンドウを使用する場合、通常のシステムメンテナンス後のアプリケーションの再起動に時間がかかります。これは、ソースデータを再度ストリームから読み取る必要があるためです。

以下は、WINDOW 句を使用してウィンドウを定義し集計を実行するクエリの例です。クエリが GROUP BY を指定しないため、このクエリではスライディングウィンドウの方法を使用してストリームのレコードを処理します。

例 1: 1 分のスライディングウィンドウを使用してストリームを処理する

アプリケーション内ストリームに入力する「はじめに」実習のデモストリーム、SOURCE_SQL_STREAM_001 を考えてみます。スキーマは次のとおりです。

```
(TICKER_SYMBOL VARCHAR(4),
 SECTOR varchar(16),
 CHANGE REAL,
 PRICE REAL)
```

1 分のスライディングウィンドウを使用して、アプリケーションで集計をコンピューティングすると仮定します。つまり、ストリームに現れる新しいレコードそれぞれについて、前の 1 分ウィンドウのレコードの集計を適用することで、アプリケーションに出力を発行させます。

以下の時間ベースのウィンドウクエリを使用できます。クエリは、WINDOW 句を使用して 1 分間隔の範囲を定義します。WINDOW 句の PARTITION BY はスライディングウィンドウ内のティッカー値でレコードをグループ化します。

```
SELECT STREAM ticker_symbol,
              MIN(Price) OVER W1 AS Min_Price,
              MAX(Price) OVER W1 AS Max_Price,
              AVG(Price) OVER W1 AS Avg_Price
FROM   "SOURCE_SQL_STREAM_001"
WINDOW W1 AS (
   PARTITION BY ticker_symbol
```

```
   RANGE INTERVAL '1' MINUTE PRECEDING);
```

クエリをテストするには

1. 「使用開始」実習に従ってアプリケーションをセットアップします。
2. アプリケーションコード内の SELECT ステートメントを前述の SELECT クエリに置き換えます。アプリケーションコードは次のようになります。

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
                        ticker_symbol VARCHAR(10),
                        Min_Price     double,
                        Max_Price     double,
                        Avg_Price     double);
CREATE OR REPLACE PUMP "STREAM_PUMP" AS
   INSERT INTO "DESTINATION_SQL_STREAM"
     SELECT STREAM ticker_symbol,
                   MIN(Price) OVER W1 AS Min_Price,
                   MAX(Price) OVER W1 AS Max_Price,
                   AVG(Price) OVER W1 AS Avg_Price
     FROM   "SOURCE_SQL_STREAM_001"
     WINDOW W1 AS (
        PARTITION BY ticker_symbol
        RANGE INTERVAL '1' MINUTE PRECEDING);
```

例 2: スライディングウインドウに集計を適用するクエリ

デモストリームに対する次のクエリは、10 秒ウィンドウの各ティッカーの価格の変動パーセントの平均を返します。

```
SELECT STREAM Ticker_Symbol,
              AVG(Change / (Price - Change)) over W1 as Avg_Percent_Change
FROM "SOURCE_SQL_STREAM_001"
WINDOW W1 AS (
        PARTITION BY ticker_symbol
        RANGE INTERVAL '10' SECOND PRECEDING);
```

クエリをテストするには

1. 「使用開始」実習に従ってアプリケーションをセットアップします。

2. アプリケーションコード内の SELECT ステートメントを前述の SELECT クエリに置き換えます。アプリケーションコードは次のようになります。

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
                            ticker_symbol VARCHAR(10),
                            Avg_Percent_Change double);
CREATE OR REPLACE PUMP "STREAM_PUMP" AS
   INSERT INTO "DESTINATION_SQL_STREAM"
      SELECT STREAM Ticker_Symbol,
                    AVG(Change / (Price - Change)) over W1 as Avg_Percent_Change
      FROM "SOURCE_SQL_STREAM_001"
      WINDOW W1 AS (
              PARTITION BY ticker_symbol
              RANGE INTERVAL '10' SECOND PRECEDING);
```

例 3: 同じストリームの複数のスライディングウィンドウからのデータのクエリ

同じストリームに対して定義された別々のスライディングウィンドウを使用して各列値を計算し出力を発行するクエリを作成できます。

次の例では、クエリは出力ティッカー、価格、a2、a10 を発行します。また、2 行の移動平均に 10 行の移動平均を交えたティッカーシンボルについて出力を発行します。列 a2 および a10 の値は、2 行および 10 行のスライディングウィンドウから取得されます。

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
                           ticker_symbol      VARCHAR(12),
                           price              double,
                           average_last2rows  double,
                           average_last10rows double);

CREATE OR REPLACE PUMP "myPump" AS INSERT INTO "DESTINATION_SQL_STREAM"
SELECT STREAM ticker_symbol,
              price,
              avg(price) over last2rows,
              avg(price) over last10rows
FROM SOURCE_SQL_STREAM_001
WINDOW
    last2rows AS (PARTITION BY ticker_symbol ROWS 2 PRECEDING),
    last10rows AS (PARTITION BY ticker_symbol ROWS 10 PRECEDING);
```

デモストリームに対してこのクエリをテストするには、「例 1」で説明されているテスト手順に従います。

ストリーミングデータオペレーション: ストリーム結合

アプリケーションに複数のアプリケーション内ストリームを指定できます。これらストリームに届くデータを関連付ける JOIN クエリを記述できます。たとえば、以下のアプリケーション内ストリームがあるとします。

- OrderStream – 発注された株注文を受け取ります。

```
(orderId SqlType, ticker SqlType, amount SqlType, ROWTIME TimeStamp)
```

- TradeStream – それらの注文に対する株取引結果を受け取ります。

```
(tradeId SqlType, orderId SqlType, ticker SqlType, amount SqlType, ticker SqlType,
 amount SqlType, ROWTIME TimeStamp)
```

以下は、これらのストリームのデータを関連付ける JOIN クエリの例です。

例 1: 注文が出されてから 1 分以内に取引があった注文をレポートする

この例では、クエリは OrderStream と TradeStream の両方を結合します。ただし、注文から 1 分で発生した取引のみが必要であるため、クエリで TradeStream に対して 1 分ウィンドウを定義します。ウィンドウクエリについては、「スライディングウィンドウ」を参照してください。

```
SELECT STREAM
     ROWTIME,
     o.orderId, o.ticker, o.amount AS orderAmount,
     t.amount AS tradeAmount
FROM OrderStream AS o
JOIN TradeStream OVER (RANGE INTERVAL '1' MINUTE PRECEDING) AS t
ON   o.orderId = t.orderId;
```

WINDOW 句を使用してウィンドウを明示的に定義し、前述のクエリを次のように記述できます。

```
SELECT STREAM
    ROWTIME,
    o.orderId, o.ticker, o.amount AS orderAmount,
```

```
    t.amount AS tradeAmount
FROM OrderStream AS o
JOIN TradeStream OVER t
ON o.orderId = t.orderId
WINDOW t AS
    (RANGE INTERVAL '1' MINUTE PRECEDING)
```

このクエリをアプリケーションコードに含めると、アプリケーションコードは連続実行されます。OrderStream の各到着レコードについて、注文の発注に続いて 1 分ウィンドウ内で取引があれば、アプリケーションで出力が発行されます。

前述のクエリでの結合は内部結合であり、クエリは TradeStream に一致するレコードがある OrderStream のレコードを発行します (逆も同様です)。外部結合を使用すると、別の興味深いシナリオを作成できます。株注文が発注されてから 1 分以内に取引がない株注文と、同じウィンドウでレポートされた別の注文に対する取引を指定するとします。これは、外部結合の例です。

```
SELECT STREAM
    ROWTIME,
    o.orderId, o.ticker, o.amount AS orderAmount,
    t.ticker, t.tradeId, t.amount AS tradeAmount,
FROM OrderStream AS o
LEFT OUTER JOIN TradeStream OVER (RANGE INTERVAL '1' MINUTE PRECEDING) AS t
ON    o.orderId = t.orderId;
```

Kinesis Data Analytics for SQL の例

このセクションでは、Amazon Kinesis Data Analytics でのアプリケーションの作成と操作の例を示します。これには、Kinesis Data Analytics アプリケーションを作成し、結果をテストするために役立つコード例と詳しい手順が含まれます。

例に進む前に、「Amazon Kinesis Data Analytics for SQL Applications: 仕組み」および「Amazon Kinesis Data Analytics for SQL Applications の開始方法」に目を通しておくことをお勧めします。

トピック

- 例: データの変換
- 例: ウィンドウと集約
- 例: 結合
- 例: 機械学習
- 例: アラートとエラー
- 例: ソリューションアクセラレーター

例: データの変換

Amazon Kinesis Data Analytics で分析を実行する前に、アプリケーションコードで入力レコードの事前処理が必要になる場合があります。これは、さまざまな理由により生じる可能性があります。たとえば、レコードがサポートされているレコード形式に準拠しておらず、アプリケーション内入力ストリームに正規化されていない列が発生する場合などです。

このセクションでは、使用できる文字列関数を使用してデータを正規化する方法、文字列の列から必要な情報を抽出する方法などの例を示します。また、参考になる日付時刻関数も紹介します。

Lambda でストリームを処理する

を使用したストリームの事前処理については AWS Lambda、「」を参照してくださいLambda 関数を使用したデータの事前処理。

トピック

- 例: 文字列値の変換
- 例: DateTime 値の変換
- 例: 複数のデータ型の変換

例: 文字列値の変換

Amazon Kinesis Data Analytics では、ストリーミングソースのレコードで JSON や CSV などの形式をサポートしています。詳細については、「RecordFormat」を参照してください。これらのレコードは入力設定によってアプリケーション内ストリームの行にマッピングされます。詳細については、「アプリケーション入力の設定」を参照してください。入力設定は、ストリーミングソースのレコードのフィールドが、アプリケーション内ストリームの列にどのようにマッピングされるかを指定します。

マッピングは、ストリーミングソースのレコードがサポートされている形式に従っている場合に機能し、正規化されたデータを持つアプリケーション内ストリームが作成されます。それでは、ストリーミングソースのデータがサポートされている規格に準拠しない場合はどうなるでしょうか? たとえば、ストリーミングソースにクリックストリームデータ、IoT センサー、アプリケーションログなどのデータが含まれる場合は、どうなるでしょうか?

次の例を検討してください。

- ストリーミングソースにアプリケーションログが含まれている – アプリケーションログは、標準の Apache ログ形式に従っており、JSON 形式を使用してストリームに書き込まれます。

  ```
  {
     "Log":"192.168.254.30 - John [24/May/2004:22:01:02 -0700] "GET /icons/
  apache_pb.gif HTTP/1.1" 304 0"
  }
  ```

  標準の Apache ログ形式については、Apache ウェブサイトの Log Files を参照してください。

- ストリーミングソースに半構造化データが含まれている – 以下に 2 つのレコードの例を示します。`Col_E_Unstructured` フィールド値は、一連のカンマ区切り値です。5 つの列があり、最初の 4 つに文字列型の値があり、最後の列にはカンマ区切り値が含まれている。

  ```
  { "Col_A" : "string",
    "Col_B" : "string",
    "Col_C" : "string",
    "Col_D" : "string",
    "Col_E_Unstructured" : "value,value,value,value"}

  { "Col_A" : "string",
    "Col_B" : "string",
  ```

```
  "Col_C" : "string",
  "Col_D" : "string",
  "Col_E_Unstructured" : "value,value,value,value"}
```

- ストリーミングソースのレコードに URL が含まれており、分析には URL のドメイン名の一部が必要である。

```
{ "referrer" : "http://www.amazon.com"}
{ "referrer" : "http://www.stackoverflow.com" }
```

このような場合、正規化データを含むアプリケーション内ストリームを作成するには、以下の 2 ステップの処理で大抵の場合は機能します。

1. 非構造化フィールドを、作成されるアプリケーション内入力ストリームの `VARCHAR(N)` タイプの列にマッピングするように、アプリケーション入力を設定します。
2. アプリケーションコードで、文字列関数を使用してこの 1 つの列を複数の列に分割し、それらの行を別のアプリケーション内ストリームに保存します。アプリケーションコードによって作成されたこのアプリケーション内ストリームは、正規化データを持ちます。その後、このアプリケーション内ストリームで分析を実行できます。

Amazon Kinesis Data Analytics は、以下の文字列の列で機能する文字列オペレーション、標準 SQL 関数、および SQL 標準の拡張を提供します。

- 文字列演算子 – `LIKE` や `SIMILAR` のような演算子は、文字列の比較に便利です。詳細については、「Amazon Managed Service for Apache Flink SQL リファレンス」の「String Operators」を参照してください。
- SQL 関数 – 個々の文字列を操作する場合は以下の関数が便利です。詳細については、「Amazon Managed Service for Apache Flink SQL リファレンス」の「String and Search Functions」を参照してください。
  - `CHAR_LENGTH` – 文字列の長さを提供します。
  - `INITCAP` – 各単語をスペースで区切り、各単語の最初の文字を大文字に、他のすべての文字を小文字に変換して、入力文字列を返します。
  - `LOWER/UPPER` – 文字列を小文字または大文字に変換します。
  - `OVERLAY` – 最初の文字列引数 (元の文字列) の一部を 2 番目の文字列引数 (置換文字列) で置き換えます。
  - `POSITION` – 文字列で別の文字列内を検索します。

  - REGEX_REPLACE – 部分文字列を別の部分文字列に置き換えます。
  - SUBSTRING – 特定の部分から始まるソース文字列の部分を抽出します。
  - TRIM - ソース文字列の最初または最後から、指定された文字のインスタンスを削除します。
- SQL 拡張 – ログや URI などの非構造化文字列での作業に便利です。詳細については、「Amazon Managed Service for Apache Flink SQL リファレンス」の「Log Parsing Functions」を参照してください。
  - FAST_REGEX_LOG_PARSER – 正規表現パーサーと同様に機能しますが、いくつかのショートカットを受け取ってより速く結果を生成できます。たとえば、高速正規表現解析は、検出された最初の一致で停止します (レイジーセマンティクスと呼ばれます)。
  - FIXED_COLUMN_LOG_PARSE – 固定幅のフィールドを解析し、特定の SQL 型に自動的に変換します。
  - REGEX_LOG_PARSE – デフォルトの Java 正規表現パターンに基づいて文字列を解析します。
  - SYS_LOG_PARSE – UNIX/Linux システムログによく見られるエントリを処理します。
  - VARIABLE_COLUMN_LOG_PARSE – 入力文字列を、区切り文字または区切り文字列で区切られたフィールドに分割します。
  - W3C_LOG_PARSE – Apache ログをすばやくフォーマットするために使用できます。

これらの関数を使用した例については、以下のトピックを参照してください。

トピック

- 例: 文字列の一部の抽出 (SUBSTRING 関数)
- 例: 正規表現 (REGEX_REPLACE 関数) を使用した部分文字列の置き換え
- 例: 正規表現 (REGEX_LOG_PARSE 関数) に基づくログ文字列の解析
- 例: ウェブログの解析 (W3C_LOG_PARSE 関数)
- 例: 複数のフィールドへの文字列の分割 (VARIABLE_COLUMN_LOG_PARSE 関数)

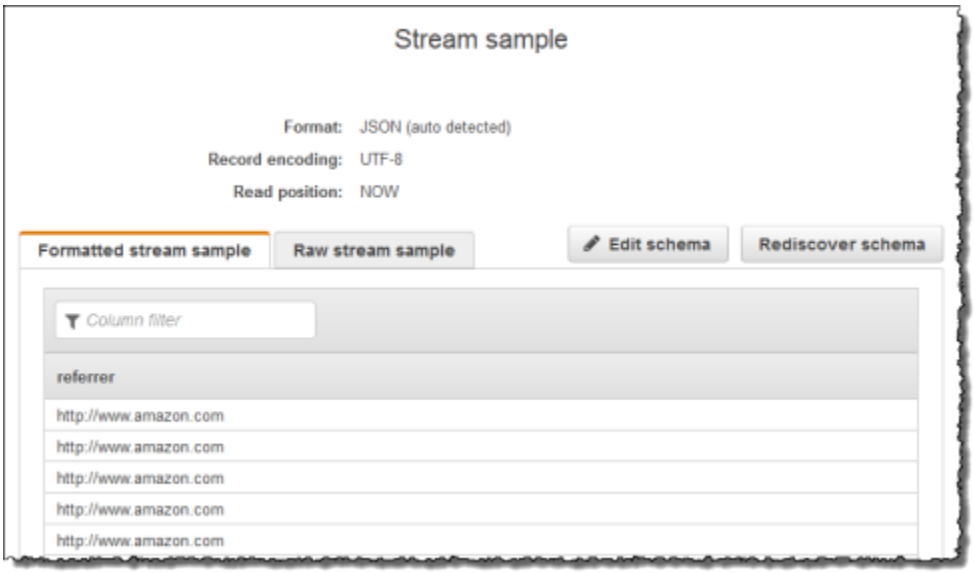
例: 文字列の一部の抽出 (SUBSTRING 関数)

この例では、SUBSTRING 関数を使用して Amazon Kinesis Data Analytics で文字列を変換します。SUBSTRING 関数は、特定の部分から始まるソース文字列の一部を抽出します。詳細については、「Amazon Managed Service for Apache Flink SQL リファレンス」の「SUBSTRING」を参照してください。

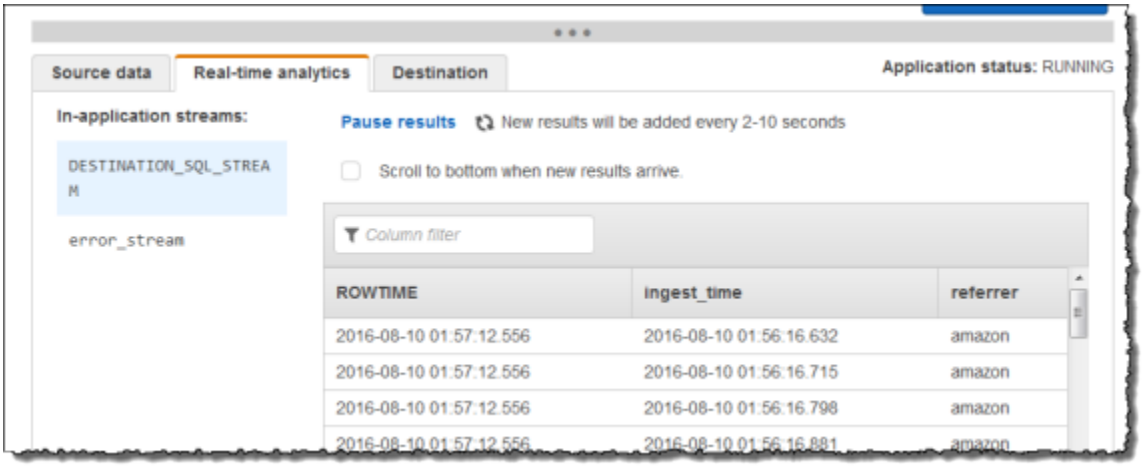
この例では、次のレコードを Amazon Kinesis データストリームに書き込みます。

```
{ "REFERRER" : "http://www.amazon.com" }
{ "REFERRER" : "http://www.amazon.com"}
{ "REFERRER" : "http://www.amazon.com"}
...
```

次に、Kinesis データストリームをストリーミングソースとして使用して、コンソールで Kinesis Data Analytics アプリケーションを作成します。検出プロセスでストリーミングソースのサンプルレコードが読み込まれ、次のように、アプリケーション内スキーマの列が 1 つ (REFERRER) であると推察します。



次に、SUBSTRING 関数を持つアプリケーションコードを使用して、URL 文字列を解析して会社名を取得します。その後、次に示すように生成されたデータを別のアプリケーション内ストリームに挿入します。



トピック

- ステップ 1: Kinesis データストリームを作成する
- ステップ 2: Kinesis Data Analytics アプリケーションを作成する

ステップ 1: Kinesis データストリームを作成する

次のように、Amazon Kinesis データストリームを作成して、ログレコードを追加します。

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/kinesis で Kinesis コンソールを開きます。
2. ナビゲーションペインで、[データストリーム] を選択します。
3. [Kinesis ストリームの作成] を選択し、1 つのシャードがあるストリームを作成します。詳細については、「Amazon Kinesis Data Streams デベロッパーガイド」の「Create a Stream」を参照してください。
4. サンプルログレコードを入力するには、以下の Python コードを実行します。このシンプルなコードは、同じログレコードを連続してストリームに書き込みます。

```
import json
import boto3

STREAM_NAME = "ExampleInputStream"


def get_data():
    return {"REFERRER": "http://www.amazon.com"}


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


if __name__ == "__main__":
```

```
    generate(STREAM_NAME, boto3.client("kinesis"))
```

ステップ 2: Kinesis Data Analytics アプリケーションを作成する

続いて、次のように Kinesis Data Analytics アプリケーションを作成します。

1. https://console.aws.amazon.com/kinesisanalytics にある Managed Service for Apache Flink コンソールを開きます。
2. [アプリケーションの作成] を選択し、アプリケーション名を入力して、[アプリケーションの作成] を選択します。
3. アプリケーション詳細ページで、[ストリーミングデータの接続] を選択します。
4. [ソースに接続] ページで、以下の操作を実行します。
   a. 前のセクションで作成したストリームを選択します。
   b. IAM ロールを作成するオプションを選択します。
   c. [スキーマの検出] を選択します。作成されたアプリケーション内ストリーム用の推測スキーマと、推測に使用されたサンプルレコードがコンソールに表示されるまで待ちます。推測スキーマの列は 1 つのみです。
   d. [Save and continue] を選択します。
5. アプリケーション詳細ページで、[SQL エディタに移動] を選択します。アプリケーションを起動するには、表示されたダイアログボックスで [はい、アプリケーションを起動します] を選択します。
6. SQL エディタで、次のように、アプリケーションコードを作成してその結果を確認します。
   a. 次のアプリケーションコードをコピーしてエディタに貼り付けます。

```
-- CREATE OR REPLACE STREAM for cleaned up referrer
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
    "ingest_time" TIMESTAMP,
    "referrer" VARCHAR(32));

CREATE OR REPLACE PUMP "myPUMP" AS
   INSERT INTO "DESTINATION_SQL_STREAM"
      SELECT STREAM
         "APPROXIMATE_ARRIVAL_TIME",
```

```
        SUBSTRING("referrer", 12, (POSITION('.com' IN "referrer") -
POSITION('www.' IN "referrer") - 4))
     FROM "SOURCE_SQL_STREAM_001";
```

b. [Save and run SQL] を選択します。[リアルタイム分析] タブに、アプリケーションで作成されたすべてのアプリケーション内ストリームが表示され、データを検証できます。

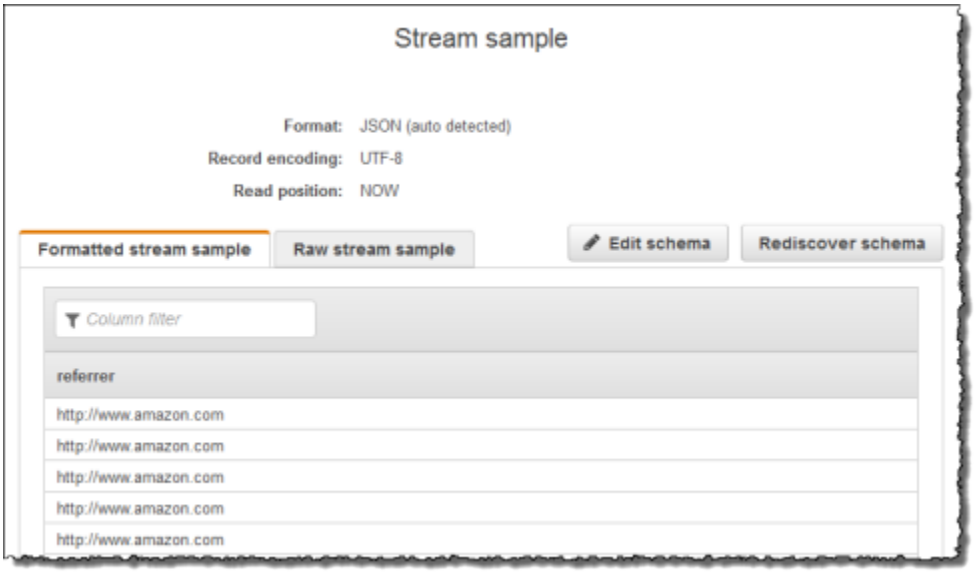
例: 正規表現 (REGEX_REPLACE 関数) を使用した部分文字列の置き換え

この例では、REGEX_REPLACE 関数を使用して Amazon Kinesis Data Analytics の文字列を変換します。REGEX_REPLACE は、部分文字列を別の部分文字列に置き換えます。詳細については、「Amazon Managed Service for Apache Flink SQL リファレンス」の「REGEX_REPLACE」を参照してください。

この例では、次のレコードを Amazon Kinesis データストリームに書き込みます。

```
{ "REFERRER" : "http://www.amazon.com" }
{ "REFERRER" : "http://www.amazon.com"}
{ "REFERRER" : "http://www.amazon.com"}
...
```

次に、Kinesis データストリームをストリーミングソースとして使用して、コンソールで Kinesis Data Analytics アプリケーションを作成します。検出プロセスでストリーミングソースのサンプルレコードが読み込まれ、次のように、アプリケーション内スキーマの列が 1 つ (REFERRER) であると推察します。



次に、REGEX_REPLACE 関数でアプリケーションコードを使用して URL を変換し、http:// の代わりに https:// を使用します。次に示すように、生成されたデータを別のアプリケーション内ストリームに挿入します。

Filter by column name

ROWTIME	ingest_time	referrer
2018-04-20 19:19:01.134	2018-04-20 19:19:00.137	https://www.amazon.com
2018-04-20 19:19:01.134	2018-04-20 19:19:00.227	https://www.amazon.com
2018-04-20 19:19:01.134	2018-04-20 19:19:00.317	https://www.amazon.com
2018-04-20 19:19:01.134	2018-04-20 19:19:00.407	https://www.amazon.com

トピック

- ステップ 1: Kinesis データストリームを作成する
- ステップ 2: Kinesis Data Analytics アプリケーションを作成する

ステップ 1: Kinesis データストリームを作成する

次のように、Amazon Kinesis データストリームを作成して、ログレコードを追加します。

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/kinesis で Kinesis コンソールを開きます。
2. ナビゲーションペインで、[データストリーム] を選択します。
3. [Kinesis ストリームの作成] を選択し、1 つのシャードがあるストリームを作成します。詳細については、「Amazon Kinesis Data Streams デベロッパーガイド」の「Create a Stream」を参照してください。
4. サンプルログレコードを入力するには、以下の Python コードを実行します。このシンプルなコードは、同じログレコードを連続してストリームに書き込みます。

```
import json
import boto3

STREAM_NAME = "ExampleInputStream"
```

```
def get_data():
    return {"REFERRER": "http://www.amazon.com"}


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


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

ステップ 2: Kinesis Data Analytics アプリケーションを作成する

続いて、次のように Kinesis Data Analytics アプリケーションを作成します。

1. https://console.aws.amazon.com/kinesisanalytics にある Managed Service for Apache Flink コンソールを開きます。
2. [アプリケーションの作成] を選択し、アプリケーション名を入力して、[アプリケーションの作成] を選択します。
3. アプリケーション詳細ページで、[ストリーミングデータの接続] を選択します。
4. [ソースに接続] ページで、以下の操作を実行します。
   a. 前のセクションで作成したストリームを選択します。
   b. IAM ロールを作成するオプションを選択します。
   c. [スキーマの検出] を選択します。作成されたアプリケーション内ストリーム用の推測スキーマと、推測に使用されたサンプルレコードがコンソールに表示されるまで待ちます。推測スキーマの列は 1 つのみです。
   d. [Save and continue] を選択します。

5. アプリケーション詳細ページで、[SQL エディタに移動] を選択します。アプリケーションを起動するには、表示されたダイアログボックスで [はい、アプリケーションを起動します] を選択します。
6. SQL エディタで、次のように、アプリケーションコードを作成してその結果を確認します。
   a. 次のアプリケーションコードをコピーしてエディタに貼り付けます。

```
-- CREATE OR REPLACE STREAM for cleaned up referrer
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
    "ingest_time" TIMESTAMP,
    "referrer" VARCHAR(32));

CREATE OR REPLACE PUMP "myPUMP" AS
   INSERT INTO "DESTINATION_SQL_STREAM"
      SELECT STREAM
         "APPROXIMATE_ARRIVAL_TIME",
         REGEX_REPLACE("REFERRER", 'http://', 'https://', 1, 0)
      FROM "SOURCE_SQL_STREAM_001";
```

   b. [Save and run SQL] を選択します。[リアルタイム分析] タブに、アプリケーションで作成されたすべてのアプリケーション内ストリームが表示され、データを検証できます。

例: 正規表現 (REGEX_LOG_PARSE 関数) に基づくログ文字列の解析

この例では、REGEX_LOG_PARSE 関数を使用して Amazon Kinesis Data Analytics の文字列を変換します。REGEX_LOG_PARSE は、デフォルトの Java 正規表現パターンに基づいて文字列を解析します。詳細については、「Amazon Managed Service for Apache Flink SQL リファレンス」の「REGEX_LOG_PARSE」を参照してください。

この例では、次のレコードを Amazon Kinesis ストリームに書き込みます。

```
{"LOGENTRY": "203.0.113.24 - - [25/Mar/2018:15:25:37 -0700] \"GET /index.php HTTP/1.1\"
 200 125 \"-\" \"Mozilla/5.0 [en] Gecko/20100101 Firefox/52.0\""}
{"LOGENTRY": "203.0.113.24 - - [25/Mar/2018:15:25:37 -0700] \"GET /index.php HTTP/1.1\"
 200 125 \"-\" \"Mozilla/5.0 [en] Gecko/20100101 Firefox/52.0\""}
{"LOGENTRY": "203.0.113.24 - - [25/Mar/2018:15:25:37 -0700] \"GET /index.php HTTP/1.1\"
 200 125 \"-\" \"Mozilla/5.0 [en] Gecko/20100101 Firefox/52.0\""}
...
```

次に、Kinesis データストリームをストリーミングソースとして使用して、コンソールで Kinesis Data Analytics アプリケーションを作成します。検出プロセスでストリーミングソースのサンプルレコードが読み込まれ、次のように、アプリケーション内スキーマの列が 1 つ (LOGENTRY) であると推察します。

ROWTIME TIMESTAMP	LOGENTRY VARCHAR(256)
2018-05-09 18:12:18.552	203.0.113.24 - - [25/Mar/2018:15:25:37 -0700] "GET /index.php HTTP/1.1"
2018-05-09 18:12:18.552	203.0.113.24 - - [25/Mar/2018:15:25:37 -0700] "GET /index.php HTTP/1.1"
2018-05-09 18:12:18.552	203.0.113.24 - - [25/Mar/2018:15:25:37 -0700] "GET /index.php HTTP/1.1"
2018-05-09 18:12:18.552	203.0.113.24 - - [25/Mar/2018:15:25:37 -0700] "GET /index.php HTTP/1.1"

次に、REGEX_LOG_PARSE 関数を持つアプリケーションコードを使用し、ログ文字列を解析してデータ要素を取得します。次のスクリーンショットに示すように生成されたデータを別のアプリケーション内ストリームに挿入します。

Filter by column name

ROWTIME	LOGENTRY	MATCH1	MATCH2
2018-05-09 18:16:11.616	203.0.113.24 - - [25/Mar	203.0.113.24 - - [25/Mar	125 "-" "Mozilla/5.0 [
2018-05-09 18:16:11.616	203.0.113.24 - - [25/Mar	203.0.113.24 - - [25/Mar	125 "-" "Mozilla/5.0 [
2018-05-09 18:16:11.616	203.0.113.24 - - [25/Mar	203.0.113.24 - - [25/Mar	125 "-" "Mozilla/5.0 [
2018-05-09 18:16:11.616	203.0.113.24 - - [25/Mar	203.0.113.24 - - [25/Mar	125 "-" "Mozilla/5.0 [

トピック

- ステップ 1: Kinesis データストリームを作成する
- ステップ 2: Kinesis Data Analytics アプリケーションを作成する

ステップ 1: Kinesis データストリームを作成する

次のように、Amazon Kinesis データストリームを作成して、ログレコードを追加します。

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/kinesis で Kinesis コンソールを開きます。
2. ナビゲーションペインで、[データストリーム] を選択します。
3. [Kinesis ストリームの作成] を選択し、1 つのシャードがあるストリームを作成します。詳細については、「Amazon Kinesis Data Streams デベロッパーガイド」の「Create a Stream」を参照してください。
4. サンプルログレコードを入力するには、以下の Python コードを実行します。このシンプルなコードは、同じログレコードを連続してストリームに書き込みます。

```
import json
import boto3

STREAM_NAME = "ExampleInputStream"


def get_data():
    return {
        "LOGENTRY": "203.0.113.24 - - [25/Mar/2018:15:25:37 -0700] "
        '"GET /index.php HTTP/1.1" 200 125 "-" '
        '"Mozilla/5.0 [en] Gecko/20100101 Firefox/52.0"'
    }


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


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

ステップ 2: Kinesis Data Analytics アプリケーションを作成する

続いて、次のように Kinesis Data Analytics アプリケーションを作成します。

1. https://console.aws.amazon.com/kinesisanalytics にある Managed Service for Apache Flink コンソールを開きます。
2. [Create application] を選択し、アプリケーション名を指定します。
3. アプリケーション詳細ページで、[ストリーミングデータの接続] を選択します。
4. [ソースに接続] ページで、以下の操作を実行します。

   a. 前のセクションで作成したストリームを選択します。

   b. IAM ロールを作成するオプションを選択します。

   c. [スキーマの検出] を選択します。作成されたアプリケーション内ストリーム用の推測スキーマと、推測に使用されたサンプルレコードがコンソールに表示されるまで待ちます。推測スキーマの列は 1 つのみです。

   d. [Save and continue] を選択します。

5. アプリケーション詳細ページで、[SQL エディタに移動] を選択します。アプリケーションを起動するには、表示されたダイアログボックスで [はい、アプリケーションを起動します] を選択します。
6. SQL エディタで、次のように、アプリケーションコードを作成してその結果を確認します。

   a. 次のアプリケーションコードをコピーしてエディタに貼り付けます。

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (logentry VARCHAR(24), match1
 VARCHAR(24), match2 VARCHAR(24));

CREATE OR REPLACE PUMP "STREAM_PUMP" AS INSERT INTO "DESTINATION_SQL_STREAM"
    SELECT STREAM T.LOGENTRY, T.REC.COLUMN1, T.REC.COLUMN2
    FROM
         (SELECT STREAM LOGENTRY,
             REGEX_LOG_PARSE(LOGENTRY, '(\w.+) (\d.+) (\w.+) (\w.+)') AS REC
             FROM SOURCE_SQL_STREAM_001) AS T;
```

   b. [Save and run SQL] を選択します。[リアルタイム分析] タブに、アプリケーションで作成されたすべてのアプリケーション内ストリームが表示され、データを検証できます。

例: ウェブログの解析 (W3C_LOG_PARSE 関数)

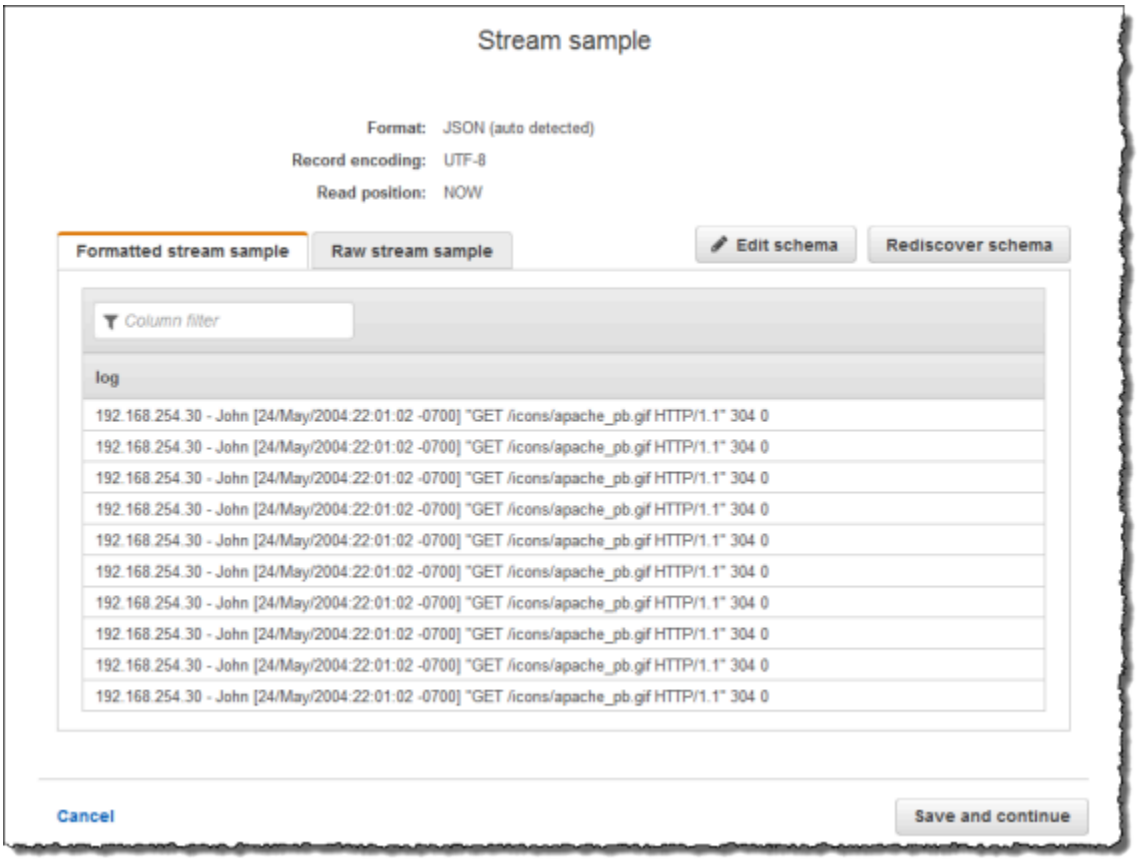
この例では、`W3C_LOG_PARSE` 関数を使用して Amazon Kinesis Data Analytics で文字列を変換します。`W3C_LOG_PARSE` を使用して、Apache ログをすばやくフォーマットできます。詳細について

は、「Amazon Managed Service for Apache Flink SQL リファレンス」の「W3C_LOG_PARSE」を参照してください。

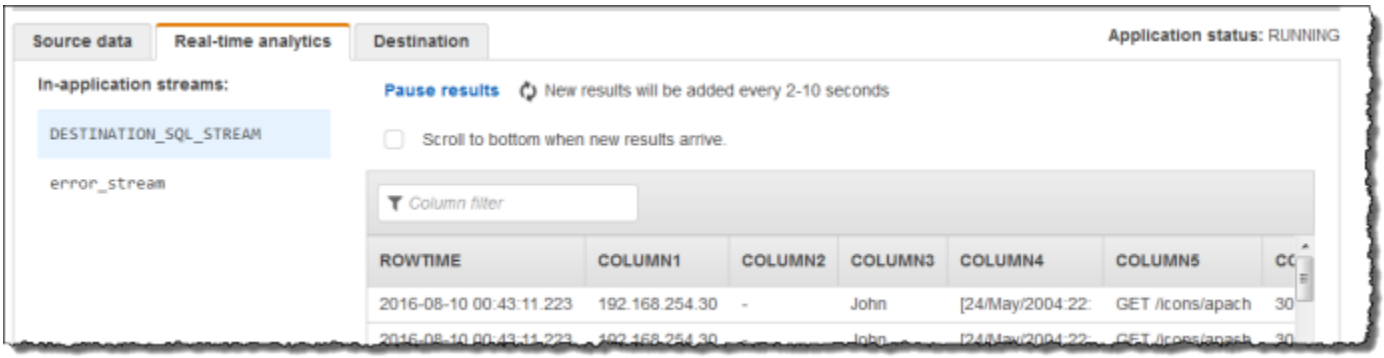
この例では、ログレコードを Amazon Kinesis データストリームに書き込みます。以下にサンプルのログを示します。

```
{"Log":"192.168.254.30 - John [24/May/2004:22:01:02 -0700] "GET /icons/apache_pba.gif
 HTTP/1.1" 304 0"}
{"Log":"192.168.254.30 - John [24/May/2004:22:01:03 -0700] "GET /icons/apache_pbb.gif
 HTTP/1.1" 304 0"}
{"Log":"192.168.254.30 - John [24/May/2004:22:01:04 -0700] "GET /icons/apache_pbc.gif
 HTTP/1.1" 304 0"}
...
```

次に、Kinesis データストリームをストリーミングソースとして使用して、コンソールで Kinesis Data Analytics アプリケーションを作成します。検出プロセスでストリーミングソースのサンプルレコードが読み込まれ、次のように、アプリケーション内スキーマの列が 1 つ (ログ) であると推察します。



次に、アプリケーションコードで W3C_LOG_PARSE 関数を使用してログを解析し、次のように別々の列にさまざまなログフィールドを持つ別のアプリケーション内ストリームを作成します。



トピック

- ステップ 1: Kinesis データストリームを作成する
- ステップ 2: Kinesis Data Analytics アプリケーションを作成する

ステップ 1: Kinesis データストリームを作成する

次のように、Amazon Kinesis データストリームを作成して、ログレコードを追加します。

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/kinesis で Kinesis コンソールを開きます。
2. ナビゲーションペインで、[データストリーム] を選択します。
3. [Kinesis ストリームの作成] を選択し、1 つのシャードがあるストリームを作成します。詳細については、「Amazon Kinesis Data Streams デベロッパーガイド」の「Create a Stream」を参照してください。
4. サンプルログレコードを入力するには、以下の Python コードを実行します。このシンプルなコードは、同じログレコードを連続してストリームに書き込みます。

```
import json
import boto3

STREAM_NAME = "ExampleInputStream"


def get_data():
    return {
        "log": "192.168.254.30 - John [24/May/2004:22:01:02 -0700] "
```

```
        '"GET /icons/apache_pb.gif HTTP/1.1" 304 0'
    }


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


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

ステップ 2: Kinesis Data Analytics アプリケーションを作成する

次のように Kinesis Data Analytics アプリケーションを作成します。

1. https://console.aws.amazon.com/kinesisanalytics にある Managed Service for Apache Flink コンソールを開きます。
2. [アプリケーションの作成] を選択し、アプリケーション名を入力して、[アプリケーションの作成] を選択します。
3. アプリケーション詳細ページで、[ストリーミングデータの接続] を選択します。
4. [ソースに接続] ページで、以下の操作を実行します。

   a. 前のセクションで作成したストリームを選択します。

   b. IAM ロールを作成するオプションを選択します。

   c. [スキーマの検出] を選択します。作成されたアプリケーション内ストリーム用の推測スキーマと、推測に使用されたサンプルレコードがコンソールに表示されるまで待ちます。推測スキーマの列は 1 つのみです。

   d. [Save and continue] を選択します。

5. アプリケーション詳細ページで、[SQL エディタに移動] を選択します。アプリケーションを起動するには、表示されたダイアログボックスで [はい、アプリケーションを起動します] を選択します。

6. SQL エディタで、次のように、アプリケーションコードを作成してその結果を確認します。

   a. 次のアプリケーションコードをコピーしてエディタに貼り付けます。

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
column1 VARCHAR(16),
column2 VARCHAR(16),
column3 VARCHAR(16),
column4 VARCHAR(16),
column5 VARCHAR(16),
column6 VARCHAR(16),
column7 VARCHAR(16));

CREATE OR REPLACE PUMP "myPUMP" AS
INSERT INTO "DESTINATION_SQL_STREAM"
        SELECT STREAM
            l.r.COLUMN1,
            l.r.COLUMN2,
            l.r.COLUMN3,
            l.r.COLUMN4,
            l.r.COLUMN5,
            l.r.COLUMN6,
            l.r.COLUMN7
        FROM (SELECT STREAM W3C_LOG_PARSE("log", 'COMMON')
              FROM "SOURCE_SQL_STREAM_001") AS l(r);
```

   b. [Save and run SQL] を選択します。[リアルタイム分析] タブに、アプリケーションで作成されたすべてのアプリケーション内ストリームが表示され、データを検証できます。

例: 複数のフィールドへの文字列の分割 (VARIABLE_COLUMN_LOG_PARSE 関数)

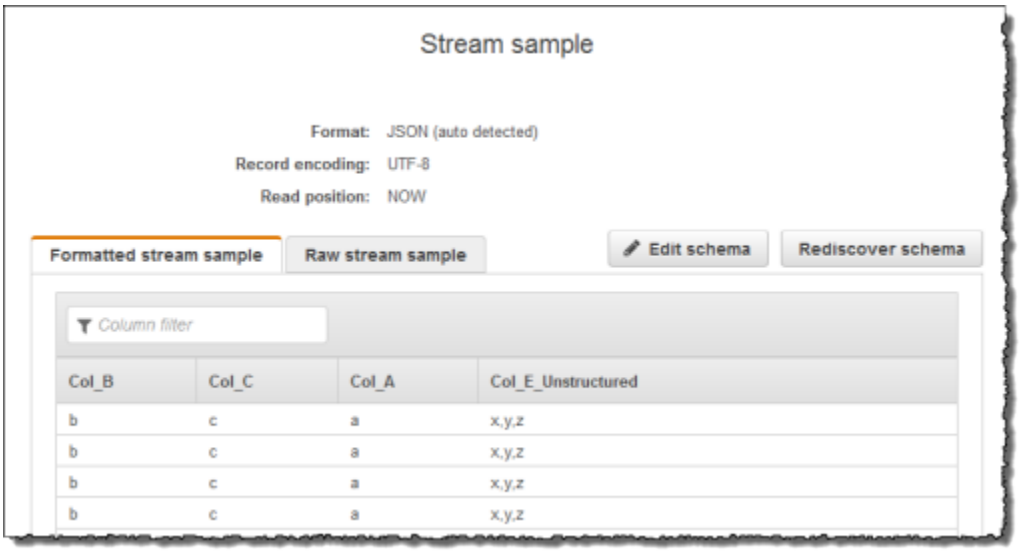
この例では、VARIABLE_COLUMN_LOG_PARSE 関数を使用して Kinesis Data Analytics の文字列を操作します。VARIABLE_COLUMN_LOG_PARSE は、入力文字列を、区切り文字または区切り文字列で区切られたフィールドに分割します。詳細については、「Amazon Managed Service for Apache Flink SQL リファレンス」の「VARIABLE_COLUMN_LOG_PARSE」を参照してください。

この例では、Amazon Kinesis データストリームに半構造化レコードを作成します。レコード例は次のとおりです。

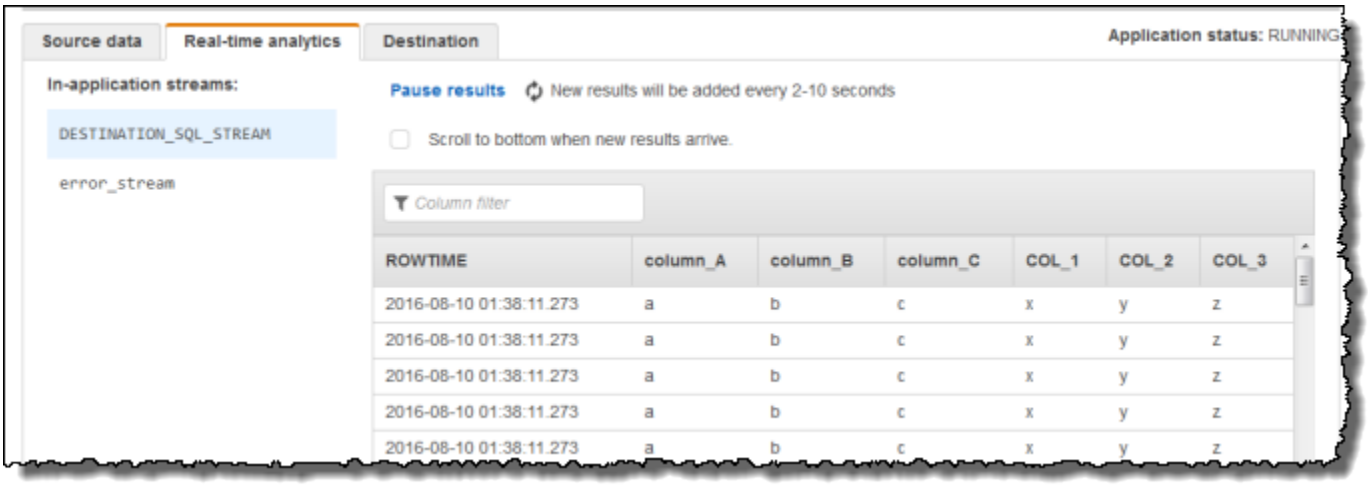
```
{ "Col_A" : "string",
  "Col_B" : "string",
  "Col_C" : "string",
```

```
  "Col_D_Unstructured" : "value,value,value,value"}
{ "Col_A" : "string",
  "Col_B" : "string",
  "Col_C" : "string",
  "Col_D_Unstructured" : "value,value,value,value"}
```

次に、Kinesis ストリームをストリーミングソースとして使用して、コンソールで Kinesis Data Analytics アプリケーションを作成します。検出プロセスでストリーミングソースのサンプルレコードが読み込まれ、次のように、アプリケーション内スキーマの列が 4 つであると推察します。



次に、VARIABLE_COLUMN_LOG_PARSE 関数を持つアプリケーションコードを使用してカンマ区切り値を解析し、次のように正規化行をアプリケーション内ストリームに挿入します。



トピック

- ステップ 1: Kinesis データストリームを作成する
- ステップ 2: Kinesis Data Analytics アプリケーションを作成する

ステップ 1: Kinesis データストリームを作成する

次のように、Amazon Kinesis データストリームを作成して、ログレコードを追加します。

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/kinesis で Kinesis コンソールを開きます。
2. ナビゲーションペインで、[データストリーム] を選択します。
3. [Kinesis ストリームの作成] を選択し、1 つのシャードがあるストリームを作成します。詳細については、「Amazon Kinesis Data Streams デベロッパーガイド」の「Create a Stream」を参照してください。
4. サンプルログレコードを入力するには、以下の Python コードを実行します。このシンプルなコードは、同じログレコードを連続してストリームに書き込みます。

```
import json
import boto3

STREAM_NAME = "ExampleInputStream"


def get_data():
    return {"Col_A": "a", "Col_B": "b", "Col_C": "c", "Col_E_Unstructured":
 "x,y,z"}


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


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

ステップ 2: Kinesis Data Analytics アプリケーションを作成する

次のように Kinesis Data Analytics アプリケーションを作成します。

1. https://console.aws.amazon.com/kinesisanalytics にある Managed Service for Apache Flink コンソールを開きます。
2. [アプリケーションの作成] を選択し、アプリケーション名を入力して、[アプリケーションの作成] を選択します。
3. アプリケーション詳細ページで、[ストリーミングデータの接続] を選択します。
4. [ソースに接続] ページで、以下の操作を実行します。
   a. 前のセクションで作成したストリームを選択します。
   b. IAM ロールを作成するオプションを選択します。
   c. [スキーマの検出] を選択します。作成されたアプリケーション内ストリーム用の推測スキーマと、推測に使用されたサンプルレコードがコンソールに表示されるまで待ちます。推測スキーマの列は 1 つのみであることに注意してください。
   d. [Save and continue] を選択します。
5. アプリケーション詳細ページで、[SQL エディタに移動] を選択します。アプリケーションを起動するには、表示されたダイアログボックスで [はい、アプリケーションを起動します] を選択します。
6. SQL エディタで、アプリケーションコードを作成してその結果を確認します。
   a. 次のアプリケーションコードをコピーしてエディタに貼り付けます。

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM"(
            "column_A" VARCHAR(16),
            "column_B" VARCHAR(16),
            "column_C" VARCHAR(16),
            "COL_1" VARCHAR(16),
            "COL_2" VARCHAR(16),
            "COL_3" VARCHAR(16));

CREATE OR REPLACE PUMP "SECOND_STREAM_PUMP" AS
INSERT INTO "DESTINATION_SQL_STREAM"
   SELECT STREAM  t."Col_A", t."Col_B", t."Col_C",
```

```
                  t.r."COL_1", t.r."COL_2", t.r."COL_3"
   FROM (SELECT STREAM
           "Col_A", "Col_B", "Col_C",
           VARIABLE_COLUMN_LOG_PARSE ("Col_E_Unstructured",
                                      'COL_1 TYPE VARCHAR(16), COL_2 TYPE
 VARCHAR(16), COL_3 TYPE VARCHAR(16)',
                                      ',') AS r
         FROM "SOURCE_SQL_STREAM_001") as t;
```

b. [Save and run SQL] を選択します。[リアルタイム分析] タブに、アプリケーションで作成されたすべてのアプリケーション内ストリームが表示され、データを検証できます。

例: DateTime 値の変換

Amazon Kinesis Data Analytics は、タイムスタンプへの列の変換をサポートします。例えば、`GROUP BY` 列に加えて、独自のタイムスタンプを `ROWTIME` 句の一部として、別の時間ベースウィンドウとして使用するとします。Kinesis Data Analytics では、日付と時刻のフィールドで機能するオペレーションと SQL 関数がサポートされています。

- 日付と時刻の演算子 - 日付、時間、間隔の各データ型に算術オペレーションを実行できます。詳細については、「Amazon Managed Service for Apache Flink SQL リファレンス」の「Date, Timestamp, and Interval Operators」を参照してください。

- SQL 関数 – 以下が含まれます。詳細については、「Amazon Managed Service for Apache Flink SQL リファレンス」の「Date and Time Functions」を参照してください。
  - `EXTRACT()` – 日付、時刻、タイムスタンプ、または間隔式から 1 つのフィールドを抽出します。
  - `CURRENT_TIME` – クエリが実行された時刻 (UTC) を返します。
  - `CURRENT_DATE` – クエリが実行された日付 (UTC) を返します。
  - `CURRENT_TIMESTAMP` – クエリが実行されたタイムスタンプ (UTC) を返します。
  - `LOCALTIME` – Kinesis Data Analytics が実行されている環境で定義されたとおりにクエリが実行された現在時刻 (UTC) を返します。
  - `LOCALTIMESTAMP` – Kinesis Data Analytics が実行されている環境で定義された現在時刻のタイムスタンプ (UTC) を返します。

- SQL 拡張 – 以下が含まれます。詳細については、「Amazon Managed Service for Apache Flink SQL リファレンス」の「Date and Time Functions」と「Datetime Conversion Functions」を参照してください。
  - CURRENT_ROW_TIMESTAMP – ストリームの各行の新しいタイムスタンプを返します。
  - TSDIFF – 2 つのタイムスタンプの差をミリ秒で返します。
  - CHAR_TO_DATE – 文字列を日付に変換します。
  - CHAR_TO_TIME – 文字列を時間に変換します。
  - CHAR_TO_TIMESTAMP – 文字列をタイムスタンプに変換します。
  - DATE_TO_CHAR – 日付を文字列に変換します。
  - TIME_TO_CHAR – 時間を文字列に変換します。
  - TIMESTAMP_TO_CHAR – タイムスタンプを文字列に変換します。

前述の SQL 関数のほとんどは列を変換する形式を使用します。形式には柔軟性があります。たとえば、yyyy-MM-dd hh:mm:ss という形式を指定して入力文字列 2009-09-16 03:15:24 をタイムスタンプに変換できます。詳細については、「Amazon Managed Service for Apache Flink SQL リファレンス」の「Char To Timestamp(Sys)」を参照してください。

例: 日付の変換

この例では、次のレコードを Amazon Kinesis データストリームに書き込みます。

```
{"EVENT_TIME": "2018-05-09T12:50:41.337510", "TICKER": "AAPL"}
{"EVENT_TIME": "2018-05-09T12:50:41.427227", "TICKER": "MSFT"}
{"EVENT_TIME": "2018-05-09T12:50:41.520549", "TICKER": "INTC"}
{"EVENT_TIME": "2018-05-09T12:50:41.610145", "TICKER": "MSFT"}
{"EVENT_TIME": "2018-05-09T12:50:41.704395", "TICKER": "AAPL"}
...
```

次に、Kinesis ストリームをストリーミングソースとして使用して、コンソールで Kinesis Data Analytics アプリケーションを作成します。検出プロセスでストリーミングソースのサンプルレコードが読み込まれ、次のように、アプリケーション内スキーマの列が 2 つ (EVENT_TIME および TICKER) であると推察します。

Filter by column name

ROWTIME	EVENT_TIME TIMESTAMP	TICKER VARCHAR(4)	PARTITION_KEY	SEQUENCE
2018-05-09 21:48:06.198	2018-05-09 14:48:05.169	INTC	partitionkey	495838547
2018-05-09 21:48:06.198	2018-05-09 14:48:05.259	TBV	partitionkey	495838547
2018-05-09 21:48:06.198	2018-05-09 14:48:05.348	INTC	partitionkey	495838547
2018-05-09 21:48:06.198	2018-05-09 14:48:05.436	MSFT	partitionkey	495838547

次に、SQL 関数でアプリケーションコードを使用して、さまざまな方法で `EVENT_TIME` タイムスタンプフィールドを変換します。その後、次のスクリーンショットに示すように生成されたデータを別のアプリケーション内ストリームに挿入します。

Filter by column name

ROWTIME	TICKER	EVENT_TIME	FIVE_MINUTES_BEFORE	EVE
2018-05-09 21:51:07.244	AAPL	2018-05-09 14:51:06.237	2018-05-09 14:46:06.237	1525
2018-05-09 21:51:07.244	INTC	2018-05-09 14:51:06.326	2018-05-09 14:46:06.326	1525
2018-05-09 21:51:07.244	AAPL	2018-05-09 14:51:06.414	2018-05-09 14:46:06.414	1525
2018-05-09 21:51:07.244	TBV	2018-05-09 14:51:06.503	2018-05-09 14:46:06.503	1525

ステップ 1: Kinesis データストリームを作成する

Amazon Kinesis データストリームを作成し、次のようにイベント時間およびティッカーレコードを入力します。

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/kinesis で Kinesis コンソールを開きます。
2. ナビゲーションペインで、[データストリーム] を選択します。
3. [Kinesis ストリームの作成] を選択し、1 つのシャードがあるストリームを作成します。

4. 以下の Python コードを実行して、サンプルデータをストリームに入力します。このシンプルなコードは、ランダムなティッカーシンボルと現在時刻のタイムスタンプを含むレコードを継続的にストリームに書き込みます。

```
import datetime
import json
import random
import boto3

STREAM_NAME = "ExampleInputStream"


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


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


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

ステップ 2: Amazon Kinesis Data Analytics アプリケーションを作成する

次のようにアプリケーションを作成します。

1. https://console.aws.amazon.com/kinesisanalytics にある Managed Service for Apache Flink コンソールを開きます。

2. [アプリケーションの作成] を選択し、アプリケーション名を入力して、[アプリケーションの作成] を選択します。
3. アプリケーション詳細ページで、[ストリーミングデータの接続] を選択してソースに接続します。
4. [ソースに接続] ページで、以下の操作を実行します。
   a. 前のセクションで作成したストリームを選択します。
   b. IAM ロールの作成を選択します。
   c. [スキーマの検出] を選択します。作成されたアプリケーション内ストリーム用の推測スキーマと、推測に使用されたサンプルレコードがコンソールに表示されるまで待ちます。推測されたスキーマには 2 つの列があります。
   d. [スキーマの編集] を選択します。[EVENT_TIME] 列の [列のタイプ] を `TIMESTAMP` に変更します。
   e. [Save schema and update stream samples] を選択します。コンソールでスキーマが保存されたら、[終了] を選択します。
   f. [Save and continue] を選択します。
5. アプリケーション詳細ページで、[SQL エディタに移動] を選択します。アプリケーションを起動するには、表示されたダイアログボックスで [はい、アプリケーションを起動します] を選択します。
6. SQL エディタで、次のように、アプリケーションコードを作成してその結果を確認します。
   a. 次のアプリケーションコードをコピーしてエディタに貼り付けます。

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
    TICKER VARCHAR(4),
    event_time TIMESTAMP,
    five_minutes_before TIMESTAMP,
    event_unix_timestamp BIGINT,
    event_timestamp_as_char VARCHAR(50),
    event_second INTEGER);

CREATE OR REPLACE PUMP "STREAM_PUMP" AS INSERT INTO "DESTINATION_SQL_STREAM"

SELECT STREAM
    TICKER,
    EVENT_TIME,
```

```
    EVENT_TIME - INTERVAL '5' MINUTE,
    UNIX_TIMESTAMP(EVENT_TIME),
    TIMESTAMP_TO_CHAR('yyyy-MM-dd hh:mm:ss', EVENT_TIME),
    EXTRACT(SECOND FROM EVENT_TIME)
FROM "SOURCE_SQL_STREAM_001"
```

b. [Save and run SQL] を選択します。[リアルタイム分析] タブに、アプリケーションで作成されたすべてのアプリケーション内ストリームが表示され、データを検証できます。

例: 複数のデータ型の変換

変換、抽出、およびロード (ETL) アプリケーションで一般的な要件は、ストリーミングソースの複数のレコードタイプを処理することです。このようなストリーミングソースを処理する Kinesis Data Analytics アプリケーションを作成できます。手順は次のとおりです。

1. 最初に、他のすべての Kinesis Data Analytics アプリケーションと同様に、ストリーミングソースをアプリケーション内入力ストリームにマッピングします。
2. 次に、アプリケーションコードで、アプリケーション内入力ストリームから特定のタイプの列を取得する SQL ステートメントを作成します。次に、別のアプリケーション内ストリームにそれらのストリームを挿入します (アプリケーションコードで追加のアプリケーション内ストリームを作成できます)。

この実習では、2 つのタイプ (`Order` と `Trade`) のレコードを受け取るストリーミングソースがあります。これらは、株注文と、それに対応する取引です。各注文に対して、ゼロ以上の取引があり得ます。各種タイプのレコード例を以下に示します。

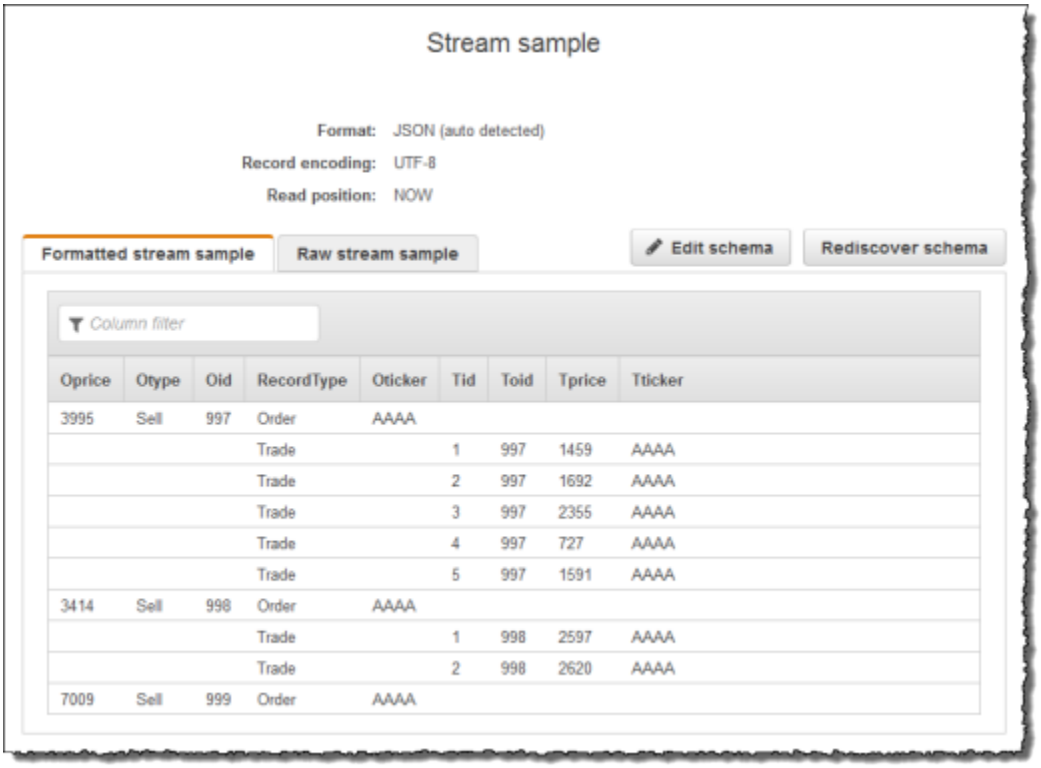
Order record

```
{"RecordType": "Order", "Oprice": 9047, "Otype": "Sell", "Oid": 3811, "Oticker":
 "AAAA"}
```

Trade record

```
{"RecordType": "Trade", "Tid": 1, "Toid": 3812, "Tprice": 2089, "Tticker": "BBBB"}
```

を使用してアプリケーションを作成すると AWS Management Console、コンソールには作成されたアプリケーション内入力ストリームの次の推測スキーマが表示されます。デフォルトでは、このアプリケーション内ストリームはコンソールで `SOURCE_SQL_STREAM_001` と命名されます。



設定を保存すると、Amazon Kinesis Data Analytics はストリーミングソースから継続してデータを読み取り、アプリケーション内ストリームに行を挿入します。これで、アプリケーション内ストリームのデータの分析を実行できます。

この例のアプリケーションコードで、まず 2 つの追加アプリケーション内ストリーム (`Order_Stream` および `Trade_Stream`) を作成します。次に、`SOURCE_SQL_STREAM_001` ストリームから、レコードタイプに基づいて行をフィルタし、ポンプを使用して新しく作成したストリームに挿入します。コーディングパターンについては、「アプリケーションコード」を参照してください。

1. 注文と取引をフィルタリングして別々のアプリケーション内ストリームにします。

   a. `SOURCE_SQL_STREAM_001` の注文レコードをフィルタリングし、注文を `Order_Stream` に保存します。

```
--Create Order_Stream.
CREATE OR REPLACE STREAM "Order_Stream"
           (
            order_id     integer,
            order_type   varchar(10),
            ticker       varchar(4),
            order_price  DOUBLE,
```

```
            record_type  varchar(10)
            );

CREATE OR REPLACE PUMP "Order_Pump" AS
   INSERT INTO "Order_Stream"
      SELECT STREAM oid, otype,oticker, oprice, recordtype
      FROM   "SOURCE_SQL_STREAM_001"
      WHERE  recordtype = 'Order';
```

b. SOURCE_SQL_STREAM_001 の取引レコードをフィルタリングし、注文を Trade_Stream に保存します。

```
--Create Trade_Stream.
CREATE OR REPLACE STREAM "Trade_Stream"
           (trade_id     integer,
            order_id     integer,
            trade_price  DOUBLE,
            ticker       varchar(4),
            record_type  varchar(10)
            );

CREATE OR REPLACE PUMP "Trade_Pump" AS
   INSERT INTO "Trade_Stream"
      SELECT STREAM tid, toid, tprice, tticker, recordtype
      FROM   "SOURCE_SQL_STREAM_001"
      WHERE  recordtype = 'Trade';
```

2. これで、これらのストリームに追加で分析を実行できます。この例では、1 分のタンブリングウィンドウでティッカーごとの取引数をカウントし、結果をさらに別のストリーム DESTINATION_SQL_STREAM に保存します。

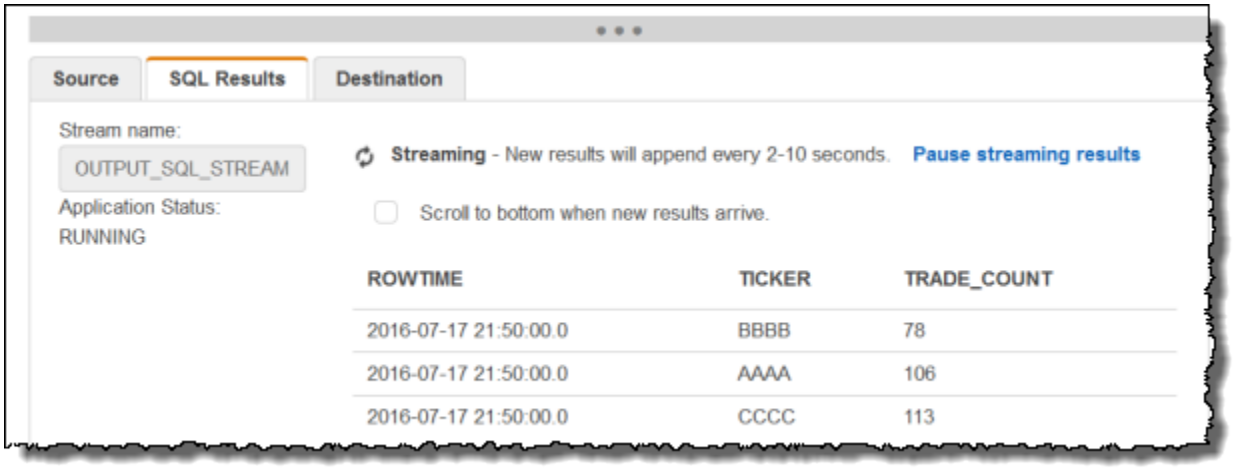
```
--do some analytics on the Trade_Stream and Order_Stream.
-- To see results in console you must write to OPUT_SQL_STREAM.

CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
            ticker  varchar(4),
            trade_count   integer
            );

CREATE OR REPLACE PUMP "Output_Pump" AS
   INSERT INTO "DESTINATION_SQL_STREAM"
      SELECT STREAM ticker, count(*) as trade_count
      FROM   "Trade_Stream"
```

```
    GROUP BY ticker,
             FLOOR("Trade_Stream".ROWTIME TO MINUTE);
```

結果は次のようになります。



トピック

- ステップ 1: データを準備する
- ステップ 2: アプリケーションの作成

次のステップ

ステップ 1: データを準備する

ステップ 1: データを準備する

このセクションでは、Kinesis データストリームを作成し、注文レコードと取引レコードをストリームに入力します。これは、次のステップで作成するアプリケーションのストリーミングソースです。

トピック

- ステップ 1.1: ストリーミングソースを作成する
- ステップ 1.2: ストリーミングソースに入力する

ステップ 1.1: ストリーミングソースを作成する

コンソールまたは AWS CLIを使用して Kinesis データストリームを作成できます。例では、`OrdersAndTradesStream` をストリーム名とします。

- コンソールの使用 – にサインイン AWS Management Console し、https://console.aws.amazon.com/kinesis で Kinesis コンソールを開きます。[Data Streams] を選択し、1 つのシャードがあるストリームを作成します。詳細については、「Amazon Kinesis Data Streams デベロッパーガイド」の「Create a Stream」を参照してください。
- の使用 AWS CLI – 次の Kinesis `create-stream` AWS CLI コマンドを使用してストリームを作成します。

```
$ aws kinesis create-stream \
--stream-name OrdersAndTradesStream \
--shard-count 1 \
--region us-east-1 \
--profile adminuser
```

ステップ 1.2: ストリーミングソースに入力する

以下の Python スクリプトを実行して `OrdersAndTradesStream` にサンプルレコードを入力します。別の名前でストリームを作成した場合は、Python コードを適切に更新してください。

1. Python および `pip` をインストールします。

   Python のインストールについては、Python ウェブサイトをご覧ください。

   pip を使用して依存関係をインストールできます。pip のインストールについては、pip ウェブサイトの「Installation」を参照してください。

2. 以下の Python コードを実行します。コードの `put-record` コマンドは、ストリームに JSON レコードを書き込みます。

```
import json
import random
import boto3

STREAM_NAME = "OrdersAndTradesStream"
PARTITION_KEY = "partition_key"


def get_order(order_id, ticker):
    return {
        "RecordType": "Order",
        "Oid": order_id,
```

```
        "Oticker": ticker,
        "Oprice": random.randint(500, 10000),
        "Otype": "Sell",
    }


def get_trade(order_id, trade_id, ticker):
    return {
        "RecordType": "Trade",
        "Tid": trade_id,
        "Toid": order_id,
        "Tticker": ticker,
        "Tprice": random.randint(0, 3000),
    }


def generate(stream_name, kinesis_client):
    order_id = 1
    while True:
        ticker = random.choice(["AAAA", "BBBB", "CCCC"])
        order = get_order(order_id, ticker)
        print(order)
        kinesis_client.put_record(
            StreamName=stream_name, Data=json.dumps(order),
 PartitionKey=PARTITION_KEY
        )
        for trade_id in range(1, random.randint(0, 6)):
            trade = get_trade(order_id, trade_id, ticker)
            print(trade)
            kinesis_client.put_record(
                StreamName=stream_name,
                Data=json.dumps(trade),
                PartitionKey=PARTITION_KEY,
            )
        order_id += 1


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

次のステップ

ステップ 2: アプリケーションの作成

ステップ 2: アプリケーションの作成

このセクションでは、Kinesis Data Analytics アプリケーションを作成します。その後、前のセクションで作成したストリーミングソースをアプリケーション内入力ストリームにマッピングする入力設定を追加して、アプリケーションを更新します。

1. https://console.aws.amazon.com/kinesisanalytics にある Managed Service for Apache Flink コンソールを開きます。
2. [Create application] を選択します。この例では、アプリケーション名 **ProcessMultipleRecordTypes** を使用します。
3. アプリケーション詳細ページで、[ストリーミングデータの接続] を選択してソースに接続します。
4. [ソースに接続] ページで、以下の操作を実行します。
   a. 「ステップ 1: データを準備する」で作成したストリームを選択します。
   b. IAM ロールの作成を選択します。
   c. 作成されたアプリケーション内ストリーム用の推測スキーマと、推測に使用されたサンプルレコードがコンソールに表示されるまで待ちます。
   d. [Save and continue] を選択します。
5. アプリケーションハブで、[Go to SQL editor] を選択します。アプリケーションを起動するには、表示されたダイアログボックスで [はい、アプリケーションを起動します] を選択します。
6. SQL エディタで、アプリケーションコードを作成してその結果を確認します。
   a. 次のアプリケーションコードをコピーしてエディタに貼り付けます。

```
--Create Order_Stream.
CREATE OR REPLACE STREAM "Order_Stream"
           (
            "order_id"     integer,
            "order_type"   varchar(10),
            "ticker"       varchar(4),
            "order_price"  DOUBLE,
            "record_type"  varchar(10)
            );

CREATE OR REPLACE PUMP "Order_Pump" AS
```

```
   INSERT INTO "Order_Stream"
      SELECT STREAM "Oid", "Otype","Oticker", "Oprice", "RecordType"
      FROM   "SOURCE_SQL_STREAM_001"
      WHERE  "RecordType" = 'Order';
--*******************************************
--Create Trade_Stream.
CREATE OR REPLACE STREAM "Trade_Stream"
           ("trade_id"     integer,
            "order_id"     integer,
            "trade_price"  DOUBLE,
            "ticker"       varchar(4),
            "record_type"  varchar(10)
            );

CREATE OR REPLACE PUMP "Trade_Pump" AS
   INSERT INTO "Trade_Stream"
      SELECT STREAM "Tid", "Toid", "Tprice", "Tticker", "RecordType"
      FROM   "SOURCE_SQL_STREAM_001"
      WHERE  "RecordType" = 'Trade';
--*************************************************************
--do some analytics on the Trade_Stream and Order_Stream.
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
            "ticker"  varchar(4),
            "trade_count"   integer
            );

CREATE OR REPLACE PUMP "Output_Pump" AS
   INSERT INTO "DESTINATION_SQL_STREAM"
      SELECT STREAM "ticker", count(*) as trade_count
      FROM   "Trade_Stream"
      GROUP BY "ticker",
                FLOOR("Trade_Stream".ROWTIME TO MINUTE);
```

b. [Save and run SQL] を選択します。[リアルタイム分析] タブを選択して、アプリケーションで作成されたすべてのアプリケーション内ストリームを表示し、データを検証します。

次のステップ

結果を永続化するために、別の Kinesis ストリームや Firehose データ配信ストリームなどの外部宛先に書き込むようにアプリケーション出力を設定できます。

例: ウィンドウと集約

このセクションでは、ウィンドウクエリと集約クエリを使用する Amazon Kinesis Data Analytics アプリケーションの例を示します。(詳細については、ウィンドウクエリ を参照してください)。それぞれの例では、Kinesis Data Analytics アプリケーションをセットアップするための詳しい手順とサンプルコードを示します。

トピック

- 例: Stagger Window
- 例: ROWTIME を使用したタンブリングウィンドウ
- 例: イベントのタイムスタンプを使用したタンブリングウィンドウ
- 例: 頻出値 (TOP_K_ITEMS_TUMBLING) の取得
- 例: クエリから部分的な結果を集約する

例: Stagger Window

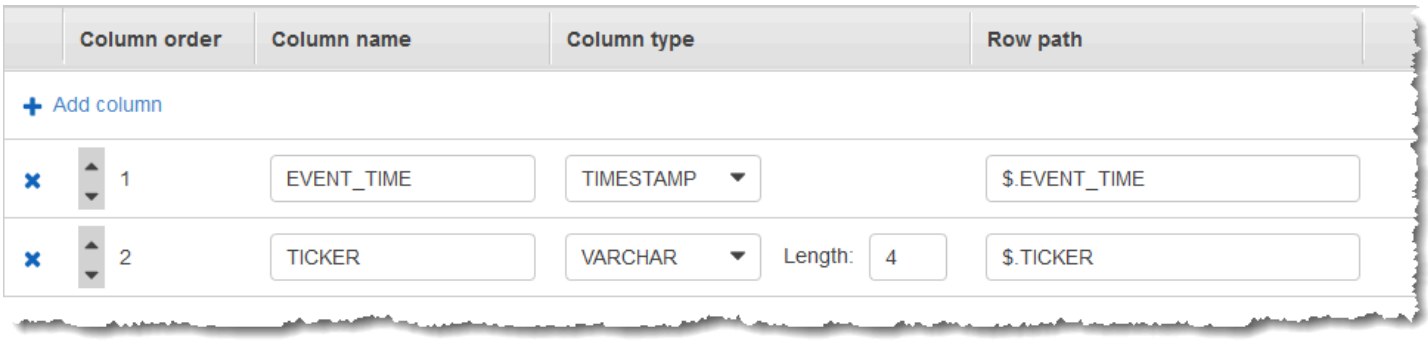
ウィンドウクエリがそれぞれ固有のパーティションキーごとに別々のウィンドウを処理するとき、一致するキーを持つデータが届くと、このウィンドウはずらしウィンドウと呼ばれます。詳細については、「Stagger Windows」を参照してください。この Amazon Kinesis Data Analytics 例では、EVENT_TIME 列と TICKER 列を使用してずらしウィンドウを作成しています。ソースストリームには、同じ EVENT_TIME 値と TICKER 値を持つ、6 つのレコードのグループが含まれています。これらは 1 分間以内に届きますが、必ずしも同じ分値で届くわけではありません (例: `18:41:xx`)。

この例では、次のレコードを Kinesis データストリームに次のタイミングで書き込みます。スクリプトは時間をストリームに書き込みませんが、レコードがアプリケーションによって取り込まれた時間が `ROWTIME` フィールドに書き込まれます。

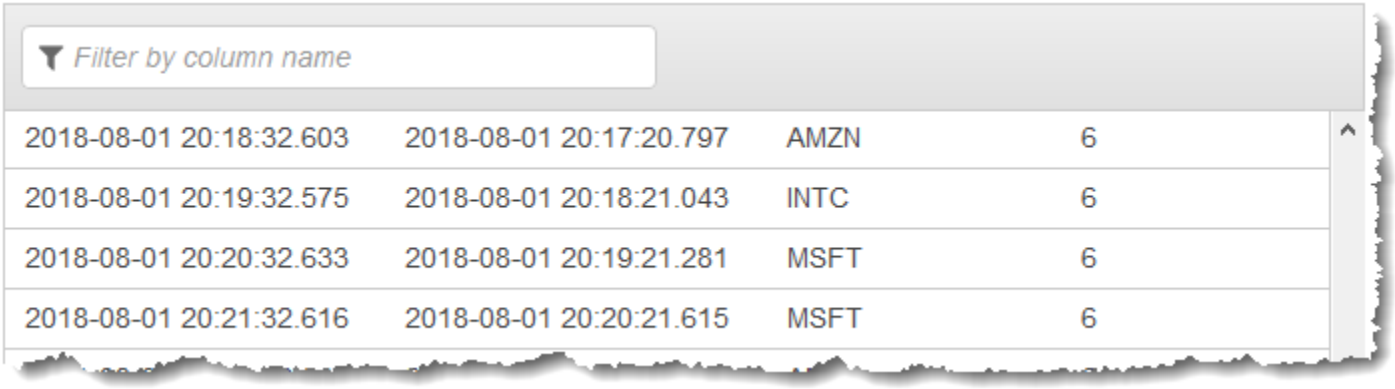
```
{"EVENT_TIME": "2018-08-01T20:17:20.797945", "TICKER": "AMZN"}   20:17:30
{"EVENT_TIME": "2018-08-01T20:17:20.797945", "TICKER": "AMZN"}   20:17:40
{"EVENT_TIME": "2018-08-01T20:17:20.797945", "TICKER": "AMZN"}   20:17:50
{"EVENT_TIME": "2018-08-01T20:17:20.797945", "TICKER": "AMZN"}   20:18:00
{"EVENT_TIME": "2018-08-01T20:17:20.797945", "TICKER": "AMZN"}   20:18:10
{"EVENT_TIME": "2018-08-01T20:17:20.797945", "TICKER": "AMZN"}   20:18:21
{"EVENT_TIME": "2018-08-01T20:18:21.043084", "TICKER": "INTC"}   20:18:31
{"EVENT_TIME": "2018-08-01T20:18:21.043084", "TICKER": "INTC"}   20:18:41
{"EVENT_TIME": "2018-08-01T20:18:21.043084", "TICKER": "INTC"}   20:18:51
{"EVENT_TIME": "2018-08-01T20:18:21.043084", "TICKER": "INTC"}   20:19:01
```

```
{"EVENT_TIME": "2018-08-01T20:18:21.043084", "TICKER": "INTC"}   20:19:11
{"EVENT_TIME": "2018-08-01T20:18:21.043084", "TICKER": "INTC"}   20:19:21
...
```

次に、Kinesis Data Streams をストリーミングソースとして AWS Management Console、 で Kinesis Data Analytics アプリケーションを作成します。検出プロセスでストリーミングソースのサンプルレコードが読み込まれます。次のように、アプリケーション内スキーマに 2 つの列 (`EVENT_TIME` および `TICKER`) があると推察します。



データのウィンドウ集約を作成するには、アプリケーションコードで `COUNT` 関数を使用します。続いて、次のスクリーンショットに示すように、生成されたデータを別のアプリケーション内ストリームに挿入します。



次の手順では、EVENT_TIME および TICKER に基づき、ずらしウィンドウの入力ストリームに値を集約する Kinesis Data Analytics アプリケーションを作成します。

トピック

- ステップ 1: Kinesis データストリームを作成する
- ステップ 2: Kinesis Data Analytics アプリケーションを作成する

ステップ 1: Kinesis データストリームを作成する

次のように、Amazon Kinesis データストリームを作成して、レコードを追加します。

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/kinesis で Kinesis コンソールを開きます。
2. ナビゲーションペインで、[データストリーム] を選択します。
3. [Kinesis ストリームの作成] を選択後、1 つのシャードがあるストリームを作成します。詳細については、「Amazon Kinesis Data Streams デベロッパーガイド」の「Create a Stream」を参照してください。
4. 本稼働環境の Kinesis データストリームにレコードを書き込むには、Kinesis Producer Library または Kinesis Data Streams API を使用することをお勧めします。わかりやすいように、この例では、以下の Python スクリプトを使用してレコードを生成します。サンプルのティッカーレコードを入力するには、このコードを実行します。このシンプルなコードは、同じランダムな EVENT_TIME とティッカーシンボルを持つ 6 つのレコードのグループを、1 分間に渡ってストリームに継続的に書き込みます。後のステップでアプリケーションスキーマを生成できるように、スクリプトを実行したままにしておきます。

```
import datetime
import json
import random
import time
import boto3

STREAM_NAME = "ExampleInputStream"


def get_data():
    event_time = datetime.datetime.utcnow() - datetime.timedelta(seconds=10)
    return {
        "EVENT_TIME": event_time.isoformat(),
        "TICKER": random.choice(["AAPL", "AMZN", "MSFT", "INTC", "TBV"]),
    }


def generate(stream_name, kinesis_client):
    while True:
        data = get_data()
        # Send six records, ten seconds apart, with the same event time and ticker
```

```
        for _ in range(6):
            print(data)
            kinesis_client.put_record(
                StreamName=stream_name,
                Data=json.dumps(data),
                PartitionKey="partitionkey",
            )
            time.sleep(10)


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

ステップ 2: Kinesis Data Analytics アプリケーションを作成する

次のように Kinesis Data Analytics アプリケーションを作成します。

1. https://console.aws.amazon.com/kinesisanalytics にある Managed Service for Apache Flink コンソールを開きます。
2. [アプリケーションの作成] を選択し、アプリケーション名を入力して、[アプリケーションの作成] を選択します。
3. アプリケーション詳細ページで、[ストリーミングデータの接続] を選択してソースに接続します。
4. [ソースに接続] ページで、以下の操作を実行します。

    a. 前のセクションで作成したストリームを選択します。

    b. [スキーマの検出] を選択します。作成されたアプリケーション内ストリーム用の推測スキーマと、推測に使用されたサンプルレコードがコンソールに表示されるまで待ちます。推測されたスキーマには 2 つの列があります。

    c. [スキーマの編集] を選択します。[EVENT_TIME] 列の [列のタイプ] を `TIMESTAMP` に変更します。

    d. [Save schema and update stream samples] を選択します。コンソールでスキーマが保存されたら、[終了] を選択します。

    e. [Save and continue] を選択します。

5. アプリケーション詳細ページで、[SQL エディタに移動] を選択します。アプリケーションを起動するには、表示されたダイアログボックスで [はい、アプリケーションを起動します] を選択します。
6. SQL エディタで、次のように、アプリケーションコードを作成してその結果を確認します。

   a. 次のアプリケーションコードをコピーしてエディタに貼り付けます。

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
    event_time TIMESTAMP,
    ticker_symbol    VARCHAR(4),
    ticker_count     INTEGER);

CREATE OR REPLACE PUMP "STREAM_PUMP" AS
  INSERT INTO "DESTINATION_SQL_STREAM"
    SELECT STREAM
        EVENT_TIME,
        TICKER,
        COUNT(TICKER) AS ticker_count
    FROM "SOURCE_SQL_STREAM_001"
    WINDOWED BY STAGGER (
            PARTITION BY TICKER, EVENT_TIME RANGE INTERVAL '1' MINUTE);
```

   b. [Save and run SQL] を選択します。

      [リアルタイム分析] タブに、アプリケーションで作成されたすべてのアプリケーション内ストリームが表示され、データを検証できます。

例: ROWTIME を使用したタンブリングウィンドウ

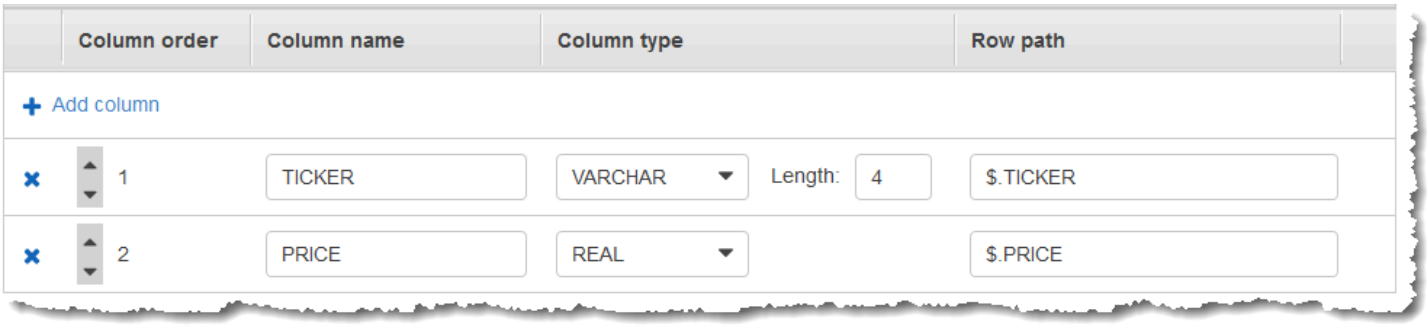
ウィンドウクエリが各ウィンドウを重複しない方式で処理する場合、ウィンドウはタンブリングウィンドウと呼ばれます。詳細については、「タンブリングウィンドウ (GROUP BY を使用した集計)」を参照してください。この Amazon Kinesis Data Analytics の例では、ROWTIME 列を使用してタンブリングウィンドウを作成します。ROWTIME 列は、アプリケーションによってレコードが読み取られた時間を表します。

この例では、次のレコードを Kinesis データストリームに書き込みます。

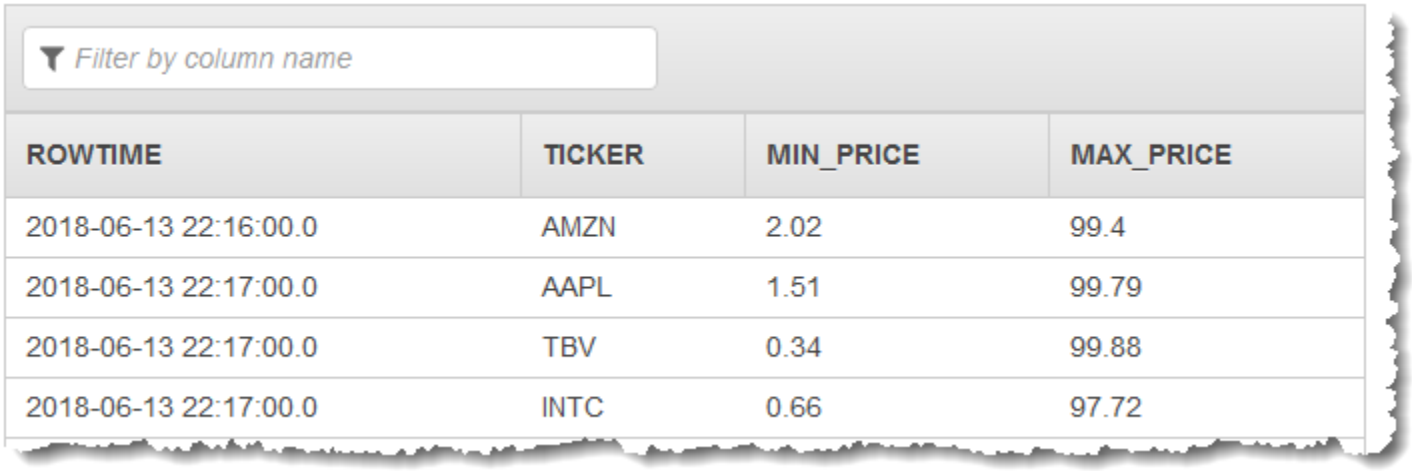
```
{"TICKER": "TBV", "PRICE": 33.11}
{"TICKER": "INTC", "PRICE": 62.04}
{"TICKER": "MSFT", "PRICE": 40.97}
{"TICKER": "AMZN", "PRICE": 27.9}
```

```
...
```

次に、Kinesis Data Streams をストリーミングソースとして AWS Management Console、 で Kinesis Data Analytics アプリケーションを作成します。検出プロセスでストリーミングソースのサンプルレコードが読み込まれます。次のように、アプリケーション内スキーマに 2 つの列 (TICKER および PRICE) があると推察します。



データのウィンドウ集約を作成するには、アプリケーションコードで MIN 関数および MAX 関数を使用します。続いて、次のスクリーンショットに示すように、生成されたデータを別のアプリケーション内ストリームに挿入します。



ROWTIME	TICKER	MIN_PRICE	MAX_PRICE
2018-06-13 22:16:00.0	AMZN	2.02	99.4
2018-06-13 22:17:00.0	AAPL	1.51	99.79
2018-06-13 22:17:00.0	TBV	0.34	99.88
2018-06-13 22:17:00.0	INTC	0.66	97.72

次の手順では、ROWTIME に基づき、タンブリングウィンドウの入力ストリームに値を集約する Kinesis Data Analytics アプリケーションを作成します。

トピック

- ステップ 1: Kinesis データストリームを作成する
- ステップ 2: Kinesis Data Analytics アプリケーションを作成する

ステップ 1: Kinesis データストリームを作成する

次のように、Amazon Kinesis データストリームを作成して、レコードを追加します。

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/kinesis で Kinesis コンソールを開きます。
2. ナビゲーションペインで、[データストリーム] を選択します。
3. [Kinesis ストリームの作成] を選択後、1 つのシャードがあるストリームを作成します。詳細については、「Amazon Kinesis Data Streams デベロッパーガイド」の「Create a Stream」を参照してください。
4. 本稼働環境の Kinesis データストリームにレコードを書き込むには、Kinesis Client Library または Kinesis Data Streams API を使用することをお勧めします。わかりやすいように、この例では、以下の Python スクリプトを使用してレコードを生成します。サンプルのティッカーレコードを入力するには、このコードを実行します。このシンプルなコードによって、ランダムなティッカーレコードが連続してストリームに書き込まれます。後のステップでアプリケーションスキーマを生成できるように、スクリプトを実行したままにしておきます。

```
import datetime
import json
import random
import boto3

STREAM_NAME = "ExampleInputStream"


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


def generate(stream_name, kinesis_client):
    while True:
        data = get_data()
        print(data)
        kinesis_client.put_record(
```

```
            StreamName=stream_name, Data=json.dumps(data),
 PartitionKey="partitionkey"
        )


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

ステップ 2: Kinesis Data Analytics アプリケーションを作成する

次のように Kinesis Data Analytics アプリケーションを作成します。

1. https://console.aws.amazon.com/kinesisanalytics にある Managed Service for Apache Flink コンソールを開きます。
2. [アプリケーションの作成] を選択し、アプリケーション名を入力して、[アプリケーションの作成] を選択します。
3. アプリケーション詳細ページで、[ストリーミングデータの接続] を選択してソースに接続します。
4. [ソースに接続] ページで、以下の操作を実行します。

   a. 前のセクションで作成したストリームを選択します。

   b. [スキーマの検出] を選択します。作成されたアプリケーション内ストリーム用の推測スキーマと、推測に使用されたサンプルレコードがコンソールに表示されるまで待ちます。推測されたスキーマには 2 つの列があります。

   c. [Save schema and update stream samples] を選択します。コンソールでスキーマが保存されたら、[終了] を選択します。

   d. [Save and continue] を選択します。
5. アプリケーション詳細ページで、[SQL エディタに移動] を選択します。アプリケーションを起動するには、表示されたダイアログボックスで [はい、アプリケーションを起動します] を選択します。
6. SQL エディタで、次のように、アプリケーションコードを作成してその結果を確認します。

   a. 次のアプリケーションコードをコピーしてエディタに貼り付けます。

   ```
   CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (TICKER VARCHAR(4), MIN_PRICE
    REAL, MAX_PRICE REAL);
   ```

```
CREATE OR REPLACE PUMP "STREAM_PUMP" AS INSERT INTO "DESTINATION_SQL_STREAM"
    SELECT STREAM TICKER, MIN(PRICE), MAX(PRICE)
        FROM "SOURCE_SQL_STREAM_001"
        GROUP BY TICKER,
            STEP("SOURCE_SQL_STREAM_001".ROWTIME BY INTERVAL '60' SECOND);
```

b. [Save and run SQL] を選択します。

   [リアルタイム分析] タブに、アプリケーションで作成されたすべてのアプリケーション内ストリームが表示され、データを検証できます。

例: イベントのタイムスタンプを使用したタンブリングウィンドウ

ウィンドウクエリが各ウィンドウを重複しない方式で処理する場合、ウィンドウはタンブリングウィンドウと呼ばれます。詳細については、「タンブリングウィンドウ (GROUP BY を使用した集計)」を参照してください。この Amazon Kinesis Data Analytics の例は、イベントのタイムスタンプを使用するタンブリングウィンドウを示します。このタイムスタンプは、ストリーミングデータに含まれているユーザーが作成したタイムスタンプです。アプリケーションがレコードを受信したときに Kinesis Data Analytics が作成する ROWTIME タイムスタンプだけを使用するのではなく、こうした方法を用います。アプリケーションの受信時ではなく、イベント発生時に基づいて集約を作成する場合は、ストリーミングデータでイベントのタイムスタンプを使用します。この例では、ROWTIME 値によって、1 分ごとに集計がトリガーされ、レコードは、ROWTIME とそこに含まれるイベント時間の両方で集計されます。

この例では、次のレコードを Amazon Kinesis ストリームに書き込みます。EVENT_TIME 値は、イベント発生時からレコードが Kinesis Data Analytics に取り込まれるまで、遅延が生じる可能性のあるプロセスや送信の遅延をシミュレートするために、過去 5 秒間に設定されます。

```
{"EVENT_TIME": "2018-06-13T14:11:05.766191", "TICKER": "TBV", "PRICE": 43.65}
{"EVENT_TIME": "2018-06-13T14:11:05.848967", "TICKER": "AMZN", "PRICE": 35.61}
{"EVENT_TIME": "2018-06-13T14:11:05.931871", "TICKER": "MSFT", "PRICE": 73.48}
{"EVENT_TIME": "2018-06-13T14:11:06.014845", "TICKER": "AMZN", "PRICE": 18.64}
...
```

次に、Kinesis Data Streams をストリーミングソースとして AWS Management Console、 で Kinesis Data Analytics アプリケーションを作成します。検出プロセスでストリーミングソースのサンプルレコードが読み込まれます。次のように、アプリケーション内スキーマに 3 つの列 (EVENT_TIME、TICKER、PRICE) あると推察します。

	Column order	Column name	Column type		Row path
+ Add column					
×	1	EVENT_TIME	TIMESTAMP		$.EVENT_TIME
×	2	TICKER	VARCHAR	Length: 4	$.TICKER
×	3	PRICE	DECIMAL		$.PRICE

データのウィンドウ集約を作成するには、アプリケーションコードで `MIN` 関数および `MAX` 関数を使用します。続いて、次のスクリーンショットに示すように、生成されたデータを別のアプリケーション内ストリームに挿入します。

Filter by column name

ROWTIME	EVENT_TIME	TICKER	MIN_PRICE	MAX_PRICE
2018-06-18 21:49:00.0	2018-06-18 21:48:00.0	MSFT	8.67	97.91
2018-06-18 21:50:00.0	2018-06-18 21:48:00.0	INTC	3.67	84.7
2018-06-18 21:50:00.0	2018-06-18 21:48:00.0	AAPL	2.39	91.35
2018-06-18 21:50:00.0	2018-06-18 21:48:00.0	AMZN	7.52	93.71

次の手順では、イベント時間に基づき、タンブリングウィンドウの入力ストリームに値を集約する Kinesis Data Analytics アプリケーションを作成します。

トピック

- ステップ 1: Kinesis データストリームを作成する
- ステップ 2: Kinesis Data Analytics アプリケーションを作成する

ステップ 1: Kinesis データストリームを作成する

次のように、Amazon Kinesis データストリームを作成して、レコードを追加します。

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/kinesis で Kinesis コンソールを開きます。

2. ナビゲーションペインで、[データストリーム] を選択します。
3. [Kinesis ストリームの作成] を選択後、1 つのシャードがあるストリームを作成します。詳細については、「Amazon Kinesis Data Streams デベロッパーガイド」の「Create a Stream」を参照してください。
4. 本稼働環境の Kinesis データストリームにレコードを書き込むには、Kinesis Client Library または Kinesis Data Streams API を使用することをお勧めします。わかりやすいように、この例では、以下の Python スクリプトを使用してレコードを生成します。サンプルのティッカーレコードを入力するには、このコードを実行します。このシンプルなコードによって、ランダムなティッカーレコードが連続してストリームに書き込まれます。後のステップでアプリケーションスキーマを生成できるように、スクリプトを実行したままにしておきます。

```
import datetime
import json
import random
import boto3

STREAM_NAME = "ExampleInputStream"


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


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


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

ステップ 2: Kinesis Data Analytics アプリケーションを作成する

次のように Kinesis Data Analytics アプリケーションを作成します。

1. https://console.aws.amazon.com/kinesisanalytics にある Managed Service for Apache Flink コンソールを開きます。
2. [アプリケーションの作成] を選択し、アプリケーション名を入力して、[アプリケーションの作成] を選択します。
3. アプリケーション詳細ページで、[ストリーミングデータの接続] を選択してソースに接続します。
4. [ソースに接続] ページで、以下の操作を実行します。

   a. 前のセクションで作成したストリームを選択します。
   b. [スキーマの検出] を選択します。作成されたアプリケーション内ストリーム用の推測スキーマと、推測に使用されたサンプルレコードがコンソールに表示されるまで待ちます。推測されたスキーマには 3 つの列があります。
   c. [スキーマの編集] を選択します。[EVENT_TIME] 列の [列のタイプ] を `TIMESTAMP` に変更します。
   d. [Save schema and update stream samples] を選択します。コンソールでスキーマが保存されたら、[終了] を選択します。
   e. [Save and continue] を選択します。
5. アプリケーション詳細ページで、[SQL エディタに移動] を選択します。アプリケーションを起動するには、表示されたダイアログボックスで [はい、アプリケーションを起動します] を選択します。
6. SQL エディタで、次のように、アプリケーションコードを作成してその結果を確認します。

   a. 次のアプリケーションコードをコピーしてエディタに貼り付けます。

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (EVENT_TIME timestamp, TICKER
 VARCHAR(4), min_price REAL, max_price REAL);

CREATE OR REPLACE PUMP "STREAM_PUMP" AS
  INSERT INTO "DESTINATION_SQL_STREAM"
    SELECT STREAM STEP("SOURCE_SQL_STREAM_001".EVENT_TIME BY INTERVAL '60'
 SECOND),
        TICKER,
         MIN(PRICE) AS MIN_PRICE,
```

```
        MAX(PRICE) AS MAX_PRICE
FROM    "SOURCE_SQL_STREAM_001"
GROUP BY TICKER,
         STEP("SOURCE_SQL_STREAM_001".ROWTIME BY INTERVAL '60' SECOND),
         STEP("SOURCE_SQL_STREAM_001".EVENT_TIME BY INTERVAL '60' SECOND);
```

b. [Save and run SQL] を選択します。

   [リアルタイム分析] タブに、アプリケーションで作成されたすべてのアプリケーション内ストリームが表示され、データを検証できます。

例: 頻出値 (TOP_K_ITEMS_TUMBLING) の取得

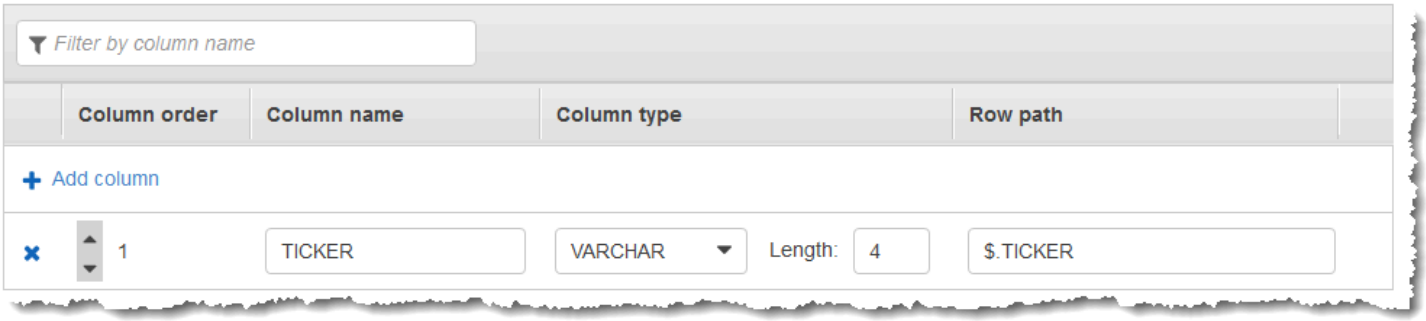
この Amazon Kinesis Data Analytics の例では、TOP_K_ITEMS_TUMBLING 関数を使用して、タンブリングウィンドウで頻出値を取得する方法について説明します。詳細については、「Amazon Managed Service for Apache Flink SQL リファレンス」の「TOP_K_ITEMS_TUMBLING function」を参照してください。

TOP_K_ITEMS_TUMBLING 関数は、数万または数十万のキーを集計しながら、リソース使用量を削減する場合に便利です。この関数では、GROUP BY 句および ORDER BY 句を使用した場合と同じ集計結果が生成されます。

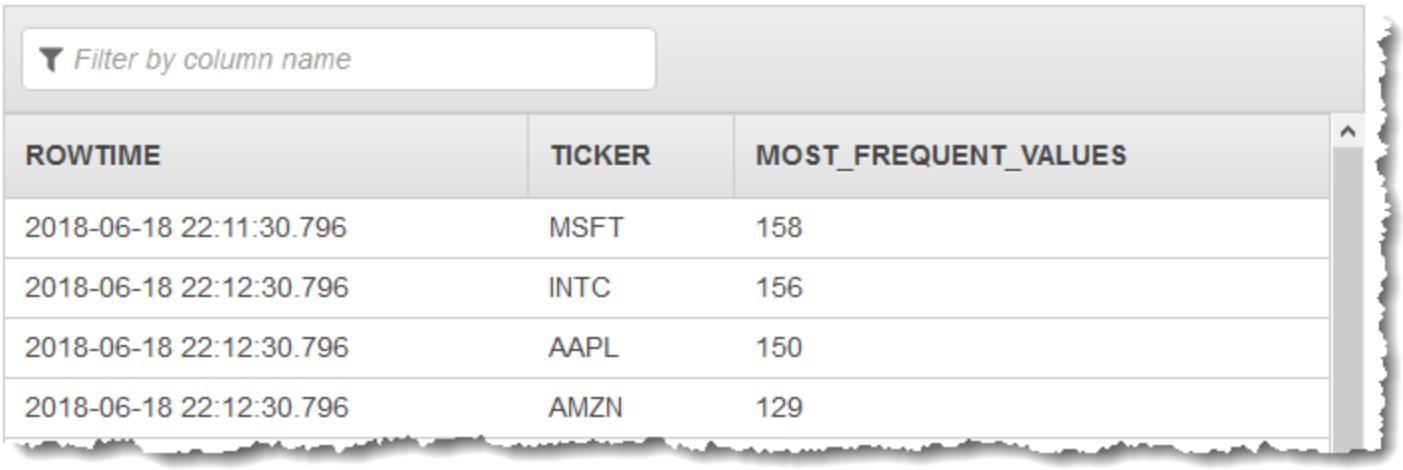
この例では、次のレコードを Amazon Kinesis データストリームに書き込みます。

```
{"TICKER": "TBV"}
{"TICKER": "INTC"}
{"TICKER": "MSFT"}
{"TICKER": "AMZN"}
...
```

次に、Kinesis Data Streams をストリーミングソースとして AWS Management Console、 で Kinesis Data Analytics アプリケーションを作成します。検出プロセスでストリーミングソースのサンプルレコードが読み込まれ、次のように、アプリケーション内スキーマに 1 つの列 (TICKER) があると推察します。



データのウィンドウ集約を作成するには、アプリケーションコードで `TOP_K_VALUES_TUMBLING` 関数を使用します。続いて、次のスクリーンショットに示すように、生成されたデータを別のアプリケーション内ストリームに挿入します。



次の手順では、入力ストリームで頻出値を取得する Kinesis Data Analytics アプリケーションを作成します。

トピック

- ステップ 1: Kinesis データストリームを作成する
- ステップ 2: Kinesis Data Analytics アプリケーションを作成する

ステップ 1: Kinesis データストリームを作成する

次のように、Amazon Kinesis データストリームを作成して、レコードを追加します。

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/kinesis で Kinesis コンソールを開きます。
2. ナビゲーションペインで、[データストリーム] を選択します。

3. [Kinesis ストリームの作成] を選択後、1 つのシャードがあるストリームを作成します。詳細については、「Amazon Kinesis Data Streams デベロッパーガイド」の「Create a Stream」を参照してください。

4. 本稼働環境の Kinesis データストリームにレコードを書き込むには、Kinesis Client Library または Kinesis Data Streams API を使用することをお勧めします。わかりやすいように、この例では、以下の Python スクリプトを使用してレコードを生成します。サンプルのティッカーレコードを入力するには、このコードを実行します。このシンプルなコードによって、ランダムなティッカーレコードが連続してストリームに書き込まれます。後のステップでアプリケーションスキーマを生成できるように、スクリプトは実行したままにしておきます。

```
import datetime
import json
import random
import boto3

STREAM_NAME = "ExampleInputStream"


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


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


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

ステップ 2: Kinesis Data Analytics アプリケーションを作成する

次のように Kinesis Data Analytics アプリケーションを作成します。

1. https://console.aws.amazon.com/kinesisanalytics にある Managed Service for Apache Flink コンソールを開きます。
2. [アプリケーションの作成] を選択し、アプリケーション名を入力して、[アプリケーションの作成] を選択します。
3. アプリケーション詳細ページで、[ストリーミングデータの接続] を選択してソースに接続します。
4. [ソースに接続] ページで、以下の操作を実行します。

   a. 前のセクションで作成したストリームを選択します。

   b. [スキーマの検出] を選択します。作成されたアプリケーション内ストリーム用の推測スキーマと、推測に使用されたサンプルレコードがコンソールに表示されるまで待ちます。推測スキーマの列は 1 つです。

   c. [Save schema and update stream samples] を選択します。コンソールでスキーマが保存されたら、[終了] を選択します。

   d. [Save and continue] を選択します。
5. アプリケーション詳細ページで、[SQL エディタに移動] を選択します。アプリケーションを起動するには、表示されたダイアログボックスで [はい、アプリケーションを起動します] を選択します。
6. SQL エディタで、次のように、アプリケーションコードを作成してその結果を確認します。

   a. 次のアプリケーションコードをコピーしてエディタに貼り付けます。

```
CREATE OR REPLACE STREAM DESTINATION_SQL_STREAM (
  "TICKER" VARCHAR(4),
  "MOST_FREQUENT_VALUES" BIGINT
);

CREATE OR REPLACE PUMP "STREAM_PUMP" AS
    INSERT INTO "DESTINATION_SQL_STREAM"
    SELECT STREAM *
        FROM TABLE (TOP_K_ITEMS_TUMBLING(
            CURSOR(SELECT STREAM * FROM "SOURCE_SQL_STREAM_001"),
            'TICKER',         -- name of column in single quotes
```

```
            5,                      -- number of the most frequently occurring
values
            60                      -- tumbling window size in seconds
            )
        );
```

b. [Save and run SQL] を選択します。

   [リアルタイム分析] タブに、アプリケーションで作成されたすべてのアプリケーション内ストリームが表示され、データを検証できます。

例: クエリから部分的な結果を集約する

Amazon Kinesis データストリームに、取り込み時刻と完全には一致しないイベント時間のレコードが含まれている場合、タンブリングウィンドウの結果の選択には、到達したレコードが含まれますが、必ずしもウィンドウ内に表示されるとは限りません。この場合、タンブリングウィンドウには、必要な結果の一部のみ含まれます。この問題の修正に使用できる方法がいくつかあります。

- タンブリングウィンドウのみを使用します。この方法では、upserts を使用してデータベースまたはデータウェアハウスの後処理を行う部分的な結果を集約します。このアプローチは、アプリケーションを効率的に処理します。これは、集約演算子 (`sum`、`min`、`max`、など) の遅延データを無限に処理します。このアプローチの欠点として、データベースレイヤーで追加のアプリケーションロジックを開発して管理する必要があります。
- タンブリングウィンドウやスライディングウィンドウを使用します。これにより、部分的な結果が早期に生成されますが、スライディングウィンドウ期間において、完全な結果も得られます。このアプローチでは、データベースレイヤーに追加のアプリケーションロジックを追加する必要がないように、upsert ではなく、overwrite を使用して遅延データを処理します。このアプローチの欠点として、多くの Kinesis 処理ユニット (KPU) を使用することと、2 つの結果が生成される点があります。一部のユースケースでは動作しないことがあります。

タンブリングウィンドウおよびライディングウィンドウの詳細については、「ウィンドウクエリ」を参照してください。

以下の手順では、タンブリングウィンドウ集約によって 2 つの部分的な結果 (アプリケーション内ストリーム `CALC_COUNT_SQL_STREAM` に送信される) が生成されます。最後の結果を生成するには、この結果を結合する必要があります。アプリケーションは 2 つめの集約 (アプリケーション内ストリーム `DESTINATION_SQL_STREAM` に送信される) が生成されます。この集約は、2 つの部分的な結果が結合されています。

イベント時間を使用して部分的な結果を集計するアプリケーションを作成するには

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/kinesis で Kinesis コンソールを開きます。
2. ナビゲーションペインで、[データ分析] を選択します。Amazon Kinesis Data Analytics for SQL Applications の開始方法 チュートリアルに従って、Kinesis Data Analytics アプリケーションを作成します。
3. SQL エディタで、アプリケーションコードを以下に置き換えます。

```
CREATE OR REPLACE STREAM "CALC_COUNT_SQL_STREAM"
    (TICKER      VARCHAR(4),
    TRADETIME   TIMESTAMP,
    TICKERCOUNT       DOUBLE);

CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM"
    (TICKER      VARCHAR(4),
    TRADETIME   TIMESTAMP,
    TICKERCOUNT       DOUBLE);

CREATE PUMP "CALC_COUNT_SQL_PUMP_001" AS
    INSERT INTO "CALC_COUNT_SQL_STREAM" ("TICKER","TRADETIME", "TICKERCOUNT")
    SELECT STREAM
        "TICKER_SYMBOL",
        STEP("SOURCE_SQL_STREAM_001"."ROWTIME" BY INTERVAL '1' MINUTE) as
 "TradeTime",
        COUNT(*) AS "TickerCount"
    FROM "SOURCE_SQL_STREAM_001"
    GROUP BY
        STEP("SOURCE_SQL_STREAM_001".ROWTIME BY INTERVAL '1' MINUTE),
        STEP("SOURCE_SQL_STREAM_001"."APPROXIMATE_ARRIVAL_TIME" BY INTERVAL '1'
 MINUTE),
        TICKER_SYMBOL;

CREATE PUMP "AGGREGATED_SQL_PUMP" AS
    INSERT INTO "DESTINATION_SQL_STREAM" ("TICKER","TRADETIME", "TICKERCOUNT")
    SELECT STREAM
        "TICKER",
        "TRADETIME",
        SUM("TICKERCOUNT") OVER W1 AS "TICKERCOUNT"
    FROM "CALC_COUNT_SQL_STREAM"
    WINDOW W1 AS (PARTITION BY "TRADETIME" RANGE INTERVAL '10' MINUTE PRECEDING);
```

アプリケーション内の SELECT ステートメントは、SOURCE_SQL_STREAM_001 の行を、1 パーセントを超える株価の変動でフィルタリングして、ポンプを使用してそれらの行を別のアプリケーション内ストリーム CHANGE_STREAM に挿入します。

4. [Save and run SQL] を選択します。

最初のポンプでは、次のようなストリームが CALC_COUNT_SQL_STREAM に出力されます。結果セットは未完成であることに注意してください。

Filter by column name

ROWTIME	TICKER	TRADETIME	TICKERCOUNT
2018-01-30 22:57:00.0	BAC	2018-01-30 22:56:00.0	2.0
2018-01-30 22:57:00.0	ALY	2018-01-30 22:56:00.0	2.0
2018-01-30 22:57:00.0	DFG	2018-01-30 22:56:00.0	5.0
2018-01-30 22:57:00.0	CVB	2018-01-30 22:56:00.0	6.0

続いて、2 番目のポンプで、完全な結果セットが含まれるストリームが DESTINATION_SQL_STREAM に出力されます。

Filter by column name

ROWTIME	TICKER	TRADETIME	TICKERCOUNT
2018-01-30 23:17:00.0	PPL	2018-01-30 23:16:00.0	4.0
2018-01-30 23:18:00.0	TGT	2018-01-30 23:17:00.0	8.0
2018-01-30 23:18:00.0	DFT	2018-01-30 23:17:00.0	6.0
2018-01-30 23:18:00.0	KFU	2018-01-30 23:17:00.0	5.0

例: 結合

このセクションでは、結合クエリを使用する Kinesis Data Analytics アプリケーションの例を示します。それぞれの例では、データ分析アプリケーションをセットアップし、Kinesis Data Analytics アプリケーションをテストするための詳しい手順を示します。

トピック

- 例: Kinesis Data Analytics アプリケーションにリファレンスデータを追加する

例: Kinesis Data Analytics アプリケーションにリファレンスデータを追加する

この実習では、既存の Kinesis Data Analytics アプリケーションにリファレンスデータを追加します。リファレンスデータについては、次のトピックを参照してください。

- Amazon Kinesis Data Analytics for SQL Applications: 仕組み
- アプリケーション入力の設定

この実習では、Kinesis Data Analytics 開始方法の実習で作成したアプリケーションにリファレンスデータを追加します。リファレンスデータは、各ティッカーシンボルの会社名を提供します。次に例を示します。

```
Ticker, Company
AMZN,Amazon
ASD, SomeCompanyA
MMB, SomeCompanyB
WAS,  SomeCompanyC
```

まず、使用開始の演習のステップを完了してスターターアプリケーションを作成します。次に、以下のステップに従ってリファレンスデータを設定し、アプリケーションに追加します。

1. データを準備する
   - 上記の参照データをオブジェクトとして Amazon Simple Storage Service (Amazon S3) に保存します。
   - ユーザーに代わって Kinesis Data Analytics が Amazon S3 オブジェクトの読み取りを引き受けられるよう IAM ロールを作成します。

2. アプリケーションにリファレンスデータソースを追加します。

   Kinesis Data Analytics は Amazon S3 オブジェクトを読み取り、アプリケーションコードでクエリできるアプリケーション内リファレンステーブルを作成します。

3. コードをテストします。

   アプリケーションコード内に、アプリケーション内ストリームをアプリケーション内リファレンステーブルに結合して各ティッカーシンボルの会社名を取得する結合クエリを作成できます。

トピック

- ステップ 1: 準備
- ステップ 2: リファレンスデータソースをアプリケーション設定に追加する
- ステップ 3: テスト: アプリケーション内リファレンステーブルをクエリする

ステップ 1: 準備

このセクションでは、サンプルのリファレンスデータを Amazon S3 バケットにオブジェクトとして保存します。ユーザーに代わって Kinesis Data Analytics がオブジェクトの読み取りを引き受けられるよう IAM ロールを作成することもできます。

Amazon S3 オブジェクトとしてのリファレンスデータの保存

このステップでは、サンプルのリファレンスデータを Amazon S3 オブジェクトとして保存します。

1. テキストエディタを開き、以下のデータを追加して、ファイルを `TickerReference.csv` として保存します。

   ```
   Ticker, Company
   AMZN,Amazon
   ASD, SomeCompanyA
   MMB, SomeCompanyB
   WAS,  SomeCompanyC
   ```

2. `TickerReference.csv` ファイルを S3 バケットにアップロードします。手順については、Amazon Simple Storage Service ユーザーガイドの「Amazon S3 へのオブジェクトのアップロード」を参照してください。

IAM ロールを作成します。

次に、Kinesis Data Analytics が Amazon S3 オブジェクトの読み取りを引き受けられるよう IAM ロールを作成します。

1. AWS Identity and Access Management (IAM) で、 という名前の IAM ロールを作成します**KinesisAnalytics-ReadS3Object**。ロールを作成するには、IAM ユーザーガイドにある「Amazon Service (AWS Management Console) 用のロールを作成する」の手順に従ってください。

   IAM コンソールで、以下を指定します。

   - [ロールタイプの選択] で、[AWS Lambda] を選択します。ロールを作成したら、Kinesis Data Analytics (ではない AWS Lambda) がロールを引き受けることを許可するように信頼ポリシーを変更します。
   - [Attach Policy] ページでポリシーをアタッチしないでください。
2. IAM ロールポリシーを更新します。

   a. IAM コンソールで、作成したロールを選択します。

   b. [信頼関係] タブで、信頼ポリシーを更新してロールを引き受ける権限を Kinesis Data Analytics に付与します。以下に信頼ポリシーを示します。

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

   c. [Permissions] タブで、[AmazonS3ReadOnlyAccess] という Amazon 管理ポリシーをアタッチします。これにより、Amazon S3 オブジェクトを読み取るアクセス権限をロールに付与します。このポリシーを以下に示します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:Get*",
        "s3:List*"
      ],
      "Resource": "*"
    }
  ]
}
```

ステップ 2: リファレンスデータソースをアプリケーション設定に追加する

このステップでは、リファレンスデータソースをアプリケーション設定に追加します。最初に、次の情報が必要です。

- S3 バケット名およびオブジェクトキー名
- IAM ロールの Amazon リソースネーム (ARN)

1. アプリケーションのメインページで [リファレンスデータの接続] を選択します。
2. [リファレンスデータソースの接続] ページで、リファレンスデータオブジェクトを含む Amazon S3 バケットを選択し、オブジェクトのキー名を入力します。
3. [アプリケーション内リファレンステーブル名] に **`CompanyName`** と入力します。
4. [選択したリソースへのアクセス] セクションで、[Kinesis Analytics が引き受ける IAM ロール] を選択後、前のセクションで作成した IAM ロール [KinesisAnalytics-ReadS3Object] を選択します。
5. [スキーマの検出] を選択します。コンソールによって、リファレンスデータの 2 つの列が検出されます。
6. [保存して閉じる] を選択します。

ステップ 3: テスト: アプリケーション内リファレンステーブルをクエリする

アプリケーション内リファレンステーブル CompanyName をクエリできるようになりました。ティッカー価格データをリファレンステーブルに結合することにより、リファレンス情報を使用してアプリケーションを強化できます。結果に会社名が表示されます。

1. アプリケーションコードを以下に置き換えます。クエリはアプリケーション内入力ストリームをアプリケーション内リファレンステーブルと結合します。アプリケーションコードは、結果を別のアプリケーション内ストリーム、DESTINATION_SQL_STREAM に書き込みます。

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (ticker_symbol VARCHAR(4),
 "Company" varchar(20), sector VARCHAR(12), change DOUBLE, price DOUBLE);

CREATE OR REPLACE PUMP "STREAM_PUMP" AS INSERT INTO "DESTINATION_SQL_STREAM"
   SELECT STREAM ticker_symbol, "c"."Company", sector, change, price
   FROM "SOURCE_SQL_STREAM_001" LEFT JOIN "CompanyName" as "c"
   ON "SOURCE_SQL_STREAM_001".ticker_symbol = "c"."Ticker";
```

2. アプリケーション出力が [SQLResults] タブに表示されていることを確認します。一部の行に会社名が表示されることを確認します (サンプルのリファレンスデータには、すべての会社名はありません)。

例: 機械学習

このセクションでは、機械学習クエリを使用する Amazon Kinesis Data Analytics アプリケーションの例を示します。機械学習クエリでは、データの複雑な分析を実行し、ストリームのデータの履歴に基づいて異常なパターンを見つけます。この例では、Kinesis Data Analytics アプリケーションをセットアップしてテストするための詳しい手順を示します。

トピック

- 例: ストリームでデータの異常を検出する (RANDOM_CUT_FOREST 関数)
- 例: データ異常の検出と説明の取得 (RANDOM_CUT_FOREST_WITH_EXPLANATION 関数)
- 例 : ストリーム上のホットスポットの検出 (HOTSPOTS 関数)

例: ストリームでデータの異常を検出する (RANDOM_CUT_FOREST 関数)

Amazon Kinesis Data Analytics では、数値列の値に基づいて異常スコアを各レコードに割り当てる関数 (RANDOM_CUT_FOREST) を提供しています。詳細については、「Amazon Managed Service for Apache Flink SQL リファレンス」の「RANDOM_CUT_FOREST Function」を参照してください。

この実習では、アプリケーションのストリーミングソースのレコードに異常スコアを割り当てるアプリケーションコードを作成します。アプリケーションをセットアップするには、以下を実行します。

1. ストリーミングソースのセットアップ – Kinesis データストリームをセットアップして、次のようにサンプル heartRate データを書き込みます。

   ```
   {"heartRate": 60, "rateType":"NORMAL"}
   ...
   {"heartRate": 180, "rateType":"HIGH"}
   ```

   この手順では、ストリームに入力するための Python スクリプトを提供しています。heartRate 値はランダムに生成されます。レコードの 99 パーセントは heartRate 値が 60 から 100 の間で、heartRate 値の 1 パーセントのみが 150 から 200 の間です。したがって、heartRate 値が 150 から 200 の間のレコードは異常です。

2. 入力の設定 – コンソールを使用して、Kinesis Data Analytics アプリケーションを作成し、ストリーミングソースをアプリケーション内ストリーム (SOURCE_SQL_STREAM_001) にマッピングすることでアプリケーション入力を設定します。アプリケーションが起動すると、Kinesis Data Analytics は継続的にストリーミングソースを読み取り、アプリケーション内ストリームにレコードを挿入します。

3. アプリケーションコードの指定 – この例では、次のアプリケーションコードを使用します。

   ```
   --Creates a temporary stream.
   CREATE OR REPLACE STREAM "TEMP_STREAM" (
            "heartRate"        INTEGER,
            "rateType"         varchar(20),
            "ANOMALY_SCORE"    DOUBLE);

   --Creates another stream for application output.
   CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
            "heartRate"        INTEGER,
            "rateType"         varchar(20),
            "ANOMALY_SCORE"    DOUBLE);
   ```

```
-- Compute an anomaly score for each record in the input stream
-- using Random Cut Forest
CREATE OR REPLACE PUMP "STREAM_PUMP" AS
   INSERT INTO "TEMP_STREAM"
      SELECT STREAM "heartRate", "rateType", ANOMALY_SCORE
      FROM TABLE(RANDOM_CUT_FOREST(
              CURSOR(SELECT STREAM * FROM "SOURCE_SQL_STREAM_001")));

-- Sort records by descending anomaly score, insert into output stream
CREATE OR REPLACE PUMP "OUTPUT_PUMP" AS
   INSERT INTO "DESTINATION_SQL_STREAM"
      SELECT STREAM * FROM "TEMP_STREAM"
      ORDER BY FLOOR("TEMP_STREAM".ROWTIME TO SECOND), ANOMALY_SCORE DESC;
```

コードは SOURCE_SQL_STREAM_001 の行を読み取り、異常スコアを割り当て、結果の行を別のアプリケーション内ストリーム (TEMP_STREAM) に書き込みます。次に、アプリケーションコードはTEMP_STREAM のレコードをソートして、結果を別のアプリケーション内ストリーム (DESTINATION_SQL_STREAM) に保存します。ポンプを使用して、アプリケーション内ストリームに行を挿入します。詳細については、「アプリケーション内ストリームとポンプ」を参照してください。

4. 出力の設定 – DESTINATION_SQL_STREAM のデータを永続化して、別の Kinesis データストリームの外部宛先に書き込むようにアプリケーション出力を設定できます。各レコードに割り当てられた異常スコアを確認して、どのスコアが異常を示しているか (また、アラートが必要か) を調べるのは、アプリケーションの範囲外です。 AWS Lambda 関数を使用して、これらの異常スコアを処理し、アラートを設定できます。

この実習では、米国東部 (バージニア北部) (us-east-1) を使用して、これらのストリームとアプリケーションを作成します。他のリージョンも使用する場合は、それに応じてコードを更新する必要があります。

トピック

- ステップ 1: 準備
- ステップ 2: アプリケーションの作成
- ステップ 3: アプリケーション出力を設定する
- ステップ 4: 出力の確認

次のステップ

ステップ 1: 準備

ステップ 1: 準備

この実習用の Amazon Kinesis Data Analytics アプリケーションを作成する前に、2 つの Kinesis データストリームを作成する必要があります。ストリームの 1 つはアプリケーションのストリーミングソースとして設定し、もう 1 つのストリームは Kinesis Data Analytics がアプリケーション出力を永続化する宛先として設定します。

トピック

- ステップ 1.1: 入力ストリームと出力データストリームを作成する
- ステップ 1.2: 入力ストリームにサンプルレコードを書き込みます

ステップ 1.1: 入力ストリームと出力データストリームを作成する

このセクションでは、2 つの Kinesis ストリーム (`ExampleInputStream` および `ExampleOutputStream`) を作成します。 AWS Management Console または AWS CLIを使用してこれらのストリームを作成できます。

- コンソールを使用するには
  1. にサインイン AWS Management Console し、https://console.aws.amazon.com/kinesis で Kinesis コンソールを開きます。
  2. [データストリームの作成] を選択します。`ExampleInputStream` という名前の 1 つのシャードがあるストリームを作成します。詳細については、「Amazon Kinesis Data Streams デベロッパーガイド」の「Create a Stream」を参照してください。
  3. 前のステップを繰り返し、`ExampleOutputStream` という名前の 1 つのシャードを持つストリームを作成します。
- を使用するには AWS CLI
  1. 次の Kinesis `create-stream` AWS CLI コマンドを使用して、最初のストリーム () を作成しますExampleInputStream。

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

2. 同じコマンドを実行し、ストリーム名を `ExampleOutputStream` に変更します。このコマンドは、アプリケーションが出力の書き込みに使用する 2 つ目のストリームを作成します。

ステップ 1.2: 入力ストリームにサンプルレコードを書き込みます

このステップでは、Python コードを実行してサンプルレコードを連続生成し、それらのレコードを `ExampleInputStream` ストリームに書き込みます。

```
{"heartRate": 60, "rateType":"NORMAL"}
...
{"heartRate": 180, "rateType":"HIGH"}
```

1. Python および `pip` をインストールします。

   Python のインストールについては、Python ウェブサイトをご覧ください。

   pip を使用して依存関係をインストールできます。pip のインストールについては、pip ウェブサイトの「Installation」を参照してください。

2. 以下の Python コードを実行します。コードの `put-record` コマンドは、ストリームに JSON レコードを書き込みます。

```
from enum import Enum
import json
import random
import boto3

STREAM_NAME = "ExampleInputStream"


class RateType(Enum):
    normal = "NORMAL"
    high = "HIGH"


def get_heart_rate(rate_type):
    if rate_type == RateType.normal:
        rate = random.randint(60, 100)
    elif rate_type == RateType.high:
        rate = random.randint(150, 200)
    else:
```

```
        raise TypeError
    return {"heartRate": rate, "rateType": rate_type.value}


def generate(stream_name, kinesis_client, output=True):
    while True:
        rnd = random.random()
        rate_type = RateType.high if rnd < 0.01 else RateType.normal
        heart_rate = get_heart_rate(rate_type)
        if output:
            print(heart_rate)
        kinesis_client.put_record(
            StreamName=stream_name,
            Data=json.dumps(heart_rate),
            PartitionKey="partitionkey",
        )


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

次のステップ

ステップ 2: アプリケーションの作成

ステップ 2: アプリケーションの作成

このセクションでは、次のように Amazon Kinesis Data Analytics アプリケーションを作成します。

- the section called “ステップ 1: 準備” で作成した Kinesis データストリームをストリーミングソースとして使用するように、アプリケーション入力を設定します。
- コンソールで [異常検出] テンプレートを使用します。

アプリケーションを作成するには

1. Kinesis Data Analytics の開始方法の実習のステップ 1、2、および 3 (「ステップ 3.1: アプリケーションの作成」を参照) に従います。
    - ソース設定で、以下を実行します。
        - 前のセクションで作成したストリーミングソースを指定します。

- コンソールがスキーマを推測した後、スキーマを編集し、heartRate 列の型を INTEGER に設定します。

  心拍値のほとんどは正常で、検出プロセスはこの列に TINYINT 型を割り当てることがほとんどです。ただし、低い割合で値が高い心拍数を示します。これらの高い値が TINYINT 型に合わない場合、Kinesis Data Analytics はそれらの列をエラーストリームに送ります。データ型を INTEGER に更新して、生成された心拍数データのすべてに対応できるようにします。

- コンソールで [異常検出] テンプレートを使用します。次にテンプレートコードを更新して適切な列名を指定します。

2. 列名を指定してアプリケーションコードを更新します。その結果アプリケーションコードは次のようになります (このコードを SQL エディタに貼り付けます)。

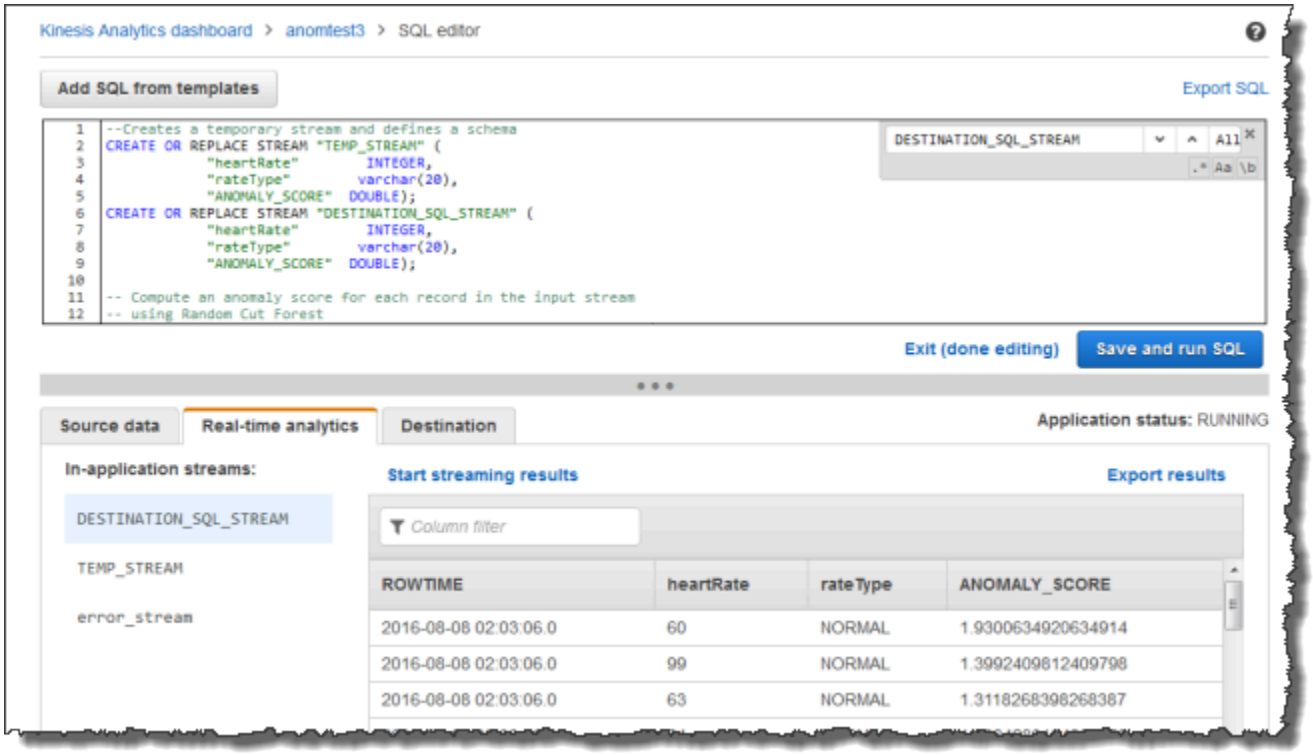
```
--Creates a temporary stream.
CREATE OR REPLACE STREAM "TEMP_STREAM" (
         "heartRate"        INTEGER,
         "rateType"         varchar(20),
         "ANOMALY_SCORE"    DOUBLE);

--Creates another stream for application output.
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
         "heartRate"        INTEGER,
         "rateType"         varchar(20),
         "ANOMALY_SCORE"    DOUBLE);

-- Compute an anomaly score for each record in the input stream
-- using Random Cut Forest
CREATE OR REPLACE PUMP "STREAM_PUMP" AS
   INSERT INTO "TEMP_STREAM"
      SELECT STREAM "heartRate", "rateType", ANOMALY_SCORE
      FROM TABLE(RANDOM_CUT_FOREST(
              CURSOR(SELECT STREAM * FROM "SOURCE_SQL_STREAM_001")));

-- Sort records by descending anomaly score, insert into output stream
CREATE OR REPLACE PUMP "OUTPUT_PUMP" AS
   INSERT INTO "DESTINATION_SQL_STREAM"
      SELECT STREAM * FROM "TEMP_STREAM"
      ORDER BY FLOOR("TEMP_STREAM".ROWTIME TO SECOND), ANOMALY_SCORE DESC;
```

3. SQL コードを実行して Kinesis Data Analytics コンソールで結果を確認します。



次のステップ

[ステップ 3: アプリケーション出力を設定する]

ステップ 3: アプリケーション出力を設定する

「the section called “ステップ 2: アプリケーションの作成”」を完了すると、ストリーミングソースから心拍数データを読み取って、異常データをそれぞれに割り当てるアプリケーションコードが作成されます。

これで、アプリケーションの結果をアプリケーション内ストリームから外部宛先である別のデータストリーム (`OutputStreamTestingAnomalyScores`) に送信できます。異常スコアを分析し、どの心拍数が異常であるか判断できます。次に、このアプリケーションを拡張してアラートを生成できます。

以下のステップに従って、アプリケーション出力を設定します。

1. Amazon Kinesis Data Analytics コンソールを開きます。SQL エディタのアプリケーションダッシュボードで、[Destination] または [Add a destination] を選択します。

2. [送信先への接続] ページで、前のセクションで作成した `OutputStreamTestingAnomalyScores` ストリームを選択します。

   こうしてできた外部宛先に、アプリケーションがアプリケーション内ストリーム `DESTINATION_SQL_STREAM` に書き込むレコードを Amazon Kinesis Data Analytics が永続化できます。

3. オプションで、ストリームをモニタリングしてアラートを送信する AWS Lambda ように `OutputStreamTestingAnomalyScores` を設定できます。手順については、Lambda 関数を使用したデータの事前処理 を参照してください。アラートを設定しない場合は、`OutputStreamTestingAnomalyScores` が外部宛先に書き込むレコードを確認できます。ステップ 4: 出力の確認 で説明する Kinesis データストリームがこれにあたります。

次のステップ

ステップ 4: 出力の確認

ステップ 4: 出力の確認

「the section called “ステップ 3: アプリケーション出力を設定する”」でアプリケーション出力を設定した後、次の AWS CLI コマンドを使用して、アプリケーションによって書き込まれた宛先ストリームのレコードを読み取ります。

1. `get-shard-iterator` コマンドを実行して出力ストリームのデータへのポインタを取得します。

```
aws kinesis get-shard-iterator \
--shard-id shardId-000000000000 \
--shard-iterator-type TRIM_HORIZON \
--stream-name OutputStreamTestingAnomalyScores \
--region us-east-1 \
--profile adminuser
```

   次のレスポンス例に示すように、シャードイテレーター値を含むレスポンスを受け取ります。

```
  {
      "ShardIterator":
      "shard-iterator-value"   }
```

   シャードイテレーター値をコピーします。

2. AWS CLI `get-records` コマンドを実行します。

```
aws kinesis get-records \
--shard-iterator shared-iterator-value \
--region us-east-1 \
--profile adminuser
```

コマンドはレコードのページと、別のシャードイテレーターを返します。これは後続の `get-records` コマンドで次のレコードのセットを取得するために使用できます。

例: データ異常の検出と説明の取得 (RANDOM_CUT_FOREST_WITH_EXPLANATION 関数)

Amazon Kinesis Data Analytics は、数値列の値に基づいて異常スコアを各レコードに割り当てる `RANDOM_CUT_FOREST_WITH_EXPLANATION` 関数を提供しています。この関数は、異常の説明も提供します。詳細については、「Amazon Managed Service for Apache Flink SQL リファレンス」の「RANDOM_CUT_FOREST_WITH_EXPLANATION」を参照してください。

この実習では、アプリケーションのストリーミングソースのレコードに異常スコアを取得するアプリケーションコードを作成します。それぞれの異常についての説明も取得できます。

トピック

- ステップ 1: データを準備する
- ステップ 2: 分析アプリケーションを作成する
- ステップ 3: 結果の検証

最初のステップ

ステップ 1: データを準備する

ステップ 1: データを準備する

この例の Amazon Kinesis Data Analytics アプリケーションを作成する前に、アプリケーションのストリーミングソースとして使用する Kinesis データストリームを作成します。また、シミュレーションされた血圧データをストリームに書き込むために Python コードを実行します。

トピック

- ステップ 1.1: Kinesis データストリームを作成する

- ステップ 1.2: 入力ストリームにサンプルレコードを書き込みます

ステップ 1.1: Kinesis データストリームを作成する

このセクションでは、`ExampleInputStream` という名前のデータストリームを作成します。このデータストリームは、 AWS Management Console または を使用して作成できます AWS CLI。

- コンソールを使用するには
  1. にサインイン AWS Management Console し、https://console.aws.amazon.com/kinesis で Kinesis コンソールを開きます。
  2. ナビゲーションペインで、[データストリーム] を選択します。次に、[Kinesis ストリームの作成] を選択します。
  3. 名前に **`ExampleInputStream`** を入力します。シャード数に **`1`** と入力します。
- または、 を使用してデータストリーム AWS CLI を作成するには、次のコマンドを実行します。

```
$ aws kinesis create-stream --stream-name ExampleInputStream --shard-count 1
```

ステップ 1.2: 入力ストリームにサンプルレコードを書き込みます

このステップでは、Python コードを実行してサンプルレコードを連続生成し、作成したデータストリームに書き込みます。

1. Python および pip をインストールします。

   Python のインストールの詳細については、Python を参照してください。

   pip を使用して依存関係をインストールできます。pip のインストールについての詳細は、pip ドキュメントのインストールを参照してください。

2. 以下の Python コードを実行します。この例で使用するリージョンに変更することができます。コードの `put-record` コマンドは、ストリームに JSON レコードを書き込みます。

```
from enum import Enum
import json
import random
import boto3
```

```
STREAM_NAME = "ExampleInputStream"


class PressureType(Enum):
    low = "LOW"
    normal = "NORMAL"
    high = "HIGH"


def get_blood_pressure(pressure_type):
    pressure = {"BloodPressureLevel": pressure_type.value}
    if pressure_type == PressureType.low:
        pressure["Systolic"] = random.randint(50, 80)
        pressure["Diastolic"] = random.randint(30, 50)
    elif pressure_type == PressureType.normal:
        pressure["Systolic"] = random.randint(90, 120)
        pressure["Diastolic"] = random.randint(60, 80)
    elif pressure_type == PressureType.high:
        pressure["Systolic"] = random.randint(130, 200)
        pressure["Diastolic"] = random.randint(90, 150)
    else:
        raise TypeError
    return pressure


def generate(stream_name, kinesis_client):
    while True:
        rnd = random.random()
        pressure_type = (
            PressureType.low
            if rnd < 0.005
            else PressureType.high
            if rnd > 0.995
            else PressureType.normal
        )
        blood_pressure = get_blood_pressure(pressure_type)
        print(blood_pressure)
        kinesis_client.put_record(
            StreamName=stream_name,
            Data=json.dumps(blood_pressure),
            PartitionKey="partitionkey",
        )
```

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

次のステップ

ステップ 2: 分析アプリケーションを作成する

ステップ 2: 分析アプリケーションを作成する

このセクションでは、Amazon Kinesis Data Analytics アプリケーションを作成し、「the section called “ステップ 1: データを準備する”」でストリーミングソースとして作成した Kinesis データストリームを使用するように設定します。RANDOM_CUT_FOREST_WITH_EXPLANATION 関数を使用するアプリケーションコードを実行します。

アプリケーションを作成するには

1. Kinesis コンソール (https://console.aws.amazon.com/kinesis) を開きます。
2. ナビゲーションペインで、[データ分析]、[アプリケーションの作成] の順に選択します。
3. アプリケーション名と説明 (オプション) を指定して、[Create application] を選択します。
4. [ストリーミングデータの接続] を選択し、リストから [ExampleInputStream] を選択します。
5. [スキーマの検出] を選択して、INTEGER 列として Systolic および Diastolic が表示されていることを確認します。別のタイプがある場合は、[Edit schema] を選択し、INTEGER タイプを両方に割り当てます。
6. [Real time analytics] の下で、[Go to SQL editor] を選択します。プロンプトが表示されたら、アプリケーションの実行を選択します。
7. 次のコードを SQL エディタに貼り付けて、[Save and run SQL] を選択します。

```
--Creates a temporary stream.
CREATE OR REPLACE STREAM "TEMP_STREAM" (
         "Systolic"                  INTEGER,
         "Diastolic"                 INTEGER,
         "BloodPressureLevel"        varchar(20),
         "ANOMALY_SCORE"             DOUBLE,
         "ANOMALY_EXPLANATION"       varchar(512));

--Creates another stream for application output.
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
         "Systolic"                  INTEGER,
         "Diastolic"                 INTEGER,
```

```
         "BloodPressureLevel"        varchar(20),
         "ANOMALY_SCORE"             DOUBLE,
         "ANOMALY_EXPLANATION"       varchar(512));

-- Compute an anomaly score with explanation for each record in the input stream
-- using RANDOM_CUT_FOREST_WITH_EXPLANATION
CREATE OR REPLACE PUMP "STREAM_PUMP" AS
   INSERT INTO "TEMP_STREAM"
      SELECT STREAM "Systolic", "Diastolic", "BloodPressureLevel", ANOMALY_SCORE,
 ANOMALY_EXPLANATION
      FROM TABLE(RANDOM_CUT_FOREST_WITH_EXPLANATION(
              CURSOR(SELECT STREAM * FROM "SOURCE_SQL_STREAM_001"), 100, 256,
 100000, 1, true));

-- Sort records by descending anomaly score, insert into output stream
CREATE OR REPLACE PUMP "OUTPUT_PUMP" AS
   INSERT INTO "DESTINATION_SQL_STREAM"
      SELECT STREAM * FROM "TEMP_STREAM"
      ORDER BY FLOOR("TEMP_STREAM".ROWTIME TO SECOND), ANOMALY_SCORE DESC;
```

次のステップ

ステップ 3: 結果の検証

ステップ 3: 結果の検証

この例で SQL コードを実行すると、まず、異常スコアがゼロの行が表示されます。これは最初の学習フェーズに発生します。その後、次のような結果が表示されます。

```
ROWTIME SYSTOLIC DIASTOLIC BLOODPRESSURELEVEL ANOMALY_SCORE ANOMALY_EXPLANATION
27:49.0 101      66        NORMAL             0.711460417   {"Systolic":
{"DIRECTION":"LOW","STRENGTH":"0.0922","ATTRIBUTION_SCORE":"0.3792"},"Diastolic":
{"DIRECTION":"HIGH","STRENGTH":"0.0210","ATTRIBUTION_SCORE":"0.3323"}}
27:50.0 144      123       HIGH               3.855851061   {"Systolic":
{"DIRECTION":"HIGH","STRENGTH":"0.8567","ATTRIBUTION_SCORE":"1.7447"},"Diastolic":
{"DIRECTION":"HIGH","STRENGTH":"7.0982","ATTRIBUTION_SCORE":"2.1111"}}
27:50.0 113      69        NORMAL             0.740069409   {"Systolic":
{"DIRECTION":"LOW","STRENGTH":"0.0549","ATTRIBUTION_SCORE":"0.3750"},"Diastolic":
{"DIRECTION":"LOW","STRENGTH":"0.0394","ATTRIBUTION_SCORE":"0.3650"}}
27:50.0 105      64        NORMAL             0.739644157   {"Systolic":
{"DIRECTION":"HIGH","STRENGTH":"0.0245","ATTRIBUTION_SCORE":"0.3667"},"Diastolic":
{"DIRECTION":"LOW","STRENGTH":"0.0524","ATTRIBUTION_SCORE":"0.3729"}}
```

```
27:50.0 100      65        NORMAL             0.736993425   {"Systolic":
{"DIRECTION":"HIGH","STRENGTH":"0.0203","ATTRIBUTION_SCORE":"0.3516"},"Diastolic":
{"DIRECTION":"LOW","STRENGTH":"0.0454","ATTRIBUTION_SCORE":"0.3854"}}
27:50.0 108      69        NORMAL             0.733767202   {"Systolic":
{"DIRECTION":"LOW","STRENGTH":"0.0974","ATTRIBUTION_SCORE":"0.3961"},"Diastolic":
{"DIRECTION":"LOW","STRENGTH":"0.0189","ATTRIBUTION_SCORE":"0.3377"}}
```

- RANDOM_CUT_FOREST_WITH_EXPLANATION 関数のアルゴリズムは、Systolic と Diastolic 列が数値であることを確認して、入力として使用します。
- BloodPressureLevel 列にはテキストデータがあるため、アルゴリズムによって考慮されません。この例で、この列は、正常、高、および低血圧レベルをすばやく見つけるのに役立つ視覚資料にすぎません。
- ANOMALY_SCORE 列で、スコアが高いレコードほど異常です。このサンプルの結果セットの 2 番目のレコードは、最も異常なもので、異常スコアは 3.855851061 です。
- アルゴリズムによって考慮された各数値列が異常スコアに寄与する範囲を理解するには、ANOMALY_SCORE 列の、ATTRIBUTION_SCORE という名前の JSON フィールドを参照してください。この一連のサンプル結果の 2 行目の場合、Systolic および Diastolic 列は 1.7447:2.1111 の割合で異常に寄与しています。つまり、異常スコアの説明の 45% は収縮期の値に起因し、残りは拡張期の値に起因しています。
- このサンプルの 2 番目の行で表されるどの点の方向が異常であるかを判断するには、DIRECTION という名前の JSON フィールドを参照してください。この場合、拡張期および収縮期の両方の値は HIGH とマークされます。これらの方向が正しいという確信を判断するには、STRENGTH という名前の JSON フィールドを参照してください。この例では、アルゴリズムは、拡張期の値が高いことをより確かなものとみています。実際に、通常の拡張期の値は 60 ~ 80 で、123 は予想よりもはるかに高い値です。

例 : ストリーム上のホットスポットの検出 (HOTSPOTS 関数)

Amazon Kinesis Data Analytics は、データの相対的に高密度なリージョンを検索してその情報を返す HOTSPOTS 関数を提供しています。詳細については、「Amazon Managed Service for Apache Flink SQL リファレンス」の「HOTSPOTS」を参照してください。

この実習では、アプリケーションのストリーミングソースのホットスポットを見つけるアプリケーションコードを作成します。アプリケーションをセットアップするには、以下のステップを実行します。

1. ストリーミングソースのセットアップ – Kinesis ストリームをセットアップして、次のようにサンプル座標データを書き込みます。

```
{"x": 7.921782426109737, "y": 8.746265312709893, "is_hot": "N"}
{"x": 0.722248626528026, "y": 4.648868803193405, "is_hot": "Y"}
```

この例では、ストリームに入力するための Python スクリプトを提供しています。x および y 値はランダムに生成され、一部のレコードは特定の場所の周りにクラスター化されます。

スクリプトがホットスポットの一部として意図的に値を生成した場合、`is_hot` フィールドはインジケータとして提供されます。これは、ホットスポット検出関数が正常に動作しているかどうかを評価するのに役立ちます。

2. アプリケーションの作成 – AWS Management Consoleを使用して Kinesis Data Analytics アプリケーションを作成します。ストリーミングソースをアプリケーション内ストリーム (`SOURCE_SQL_STREAM_001`) にマッピングして、アプリケーション入力を設定します。アプリケーションが起動すると、Kinesis Data Analytics は継続的にストリーミングソースを読み取り、アプリケーション内ストリームにレコードを挿入します。

この演習では、アプリケーションに次のコードを使用します。

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
    "x" DOUBLE,
    "y" DOUBLE,
    "is_hot" VARCHAR(4),
    HOTSPOTS_RESULT VARCHAR(10000)
);
CREATE OR REPLACE PUMP "STREAM_PUMP" AS
    INSERT INTO "DESTINATION_SQL_STREAM"
    SELECT "x", "y", "is_hot", "HOTSPOTS_RESULT"
    FROM TABLE (
        HOTSPOTS(
            CURSOR(SELECT STREAM "x", "y", "is_hot" FROM "SOURCE_SQL_STREAM_001"),
            1000,
            0.2,
            17)
    );
```

コードは `SOURCE_SQL_STREAM_001` の行を読み取り、重要なホットスポットを分析し、結果のデータを別のアプリケーション内ストリーム (`DESTINATION_SQL_STREAM`) に書き込みま

す。ポンプを使用して、アプリケーション内ストリームに行を挿入します。詳細については、「アプリケーション内ストリームとポンプ」を参照してください。

3. 出力の設定 – アプリケーションから別の Kinesis データストリームである外部送信先にデータを送信するように、アプリケーション出力を設定します。ホットスポットのスコアを確認し、ホットスポットが発生したことを示すスコア (およびアラートが必要なスコア) を判断します。 AWS Lambda 関数を使用して、ホットスポット情報をさらに処理し、アラートを設定できます。
4. 出力を確認する – この例には、出力ストリームからデータを読み込んでグラフィカルに表示する JavaScript アプリケーションが含まれているので、アプリケーションが生成するホットスポットをリアルタイムで表示できます。

この実習では、米国西部 (オレゴン) (`us-west-2`) を使用して、これらのストリームとアプリケーションを作成します。他のリージョンも使用する場合は、それに応じてコードを更新してください。

トピック

- ステップ 1: 入力ストリームと出力ストリームを作成する
- ステップ 2: Kinesis Data Analytics アプリケーションを作成する
- ステップ 3: アプリケーション出力を設定する
- ステップ 4: アプリケーション出力を検証する

ステップ 1: 入力ストリームと出力ストリームを作成する

ホットスポット例用の Amazon Kinesis Data Analytics アプリケーションを作成する前に、2 つの Kinesis データストリームを作成する必要があります。ストリームの 1 つはアプリケーションのストリーミングソースとして設定し、もう 1 つのストリームは Kinesis Data Analytics がアプリケーション出力を永続化する宛先として設定します。

トピック

- ステップ 1.1: Kinesis データストリームを作成する
- ステップ 1.2: 入力ストリームにサンプルレコードを書き込みます

ステップ 1.1: Kinesis データストリームを作成する

このセクションでは、2 つの Kinesis データストリーム (`ExampleInputStream` および `ExampleOutputStream`) を作成します。

コンソールまたは AWS CLIを使用してこれらのデータストリームを作成します。

- コンソールを使用してデータストリームを作成するには

  1. にサインイン AWS Management Console し、https://console.aws.amazon.com/kinesis で Kinesis コンソールを開きます。
  2. ナビゲーションペインで、[データストリーム] を選択します。
  3. [Kinesis ストリームの作成] を選択し、`ExampleInputStream` という名前の 1 つのシャードを持つストリームを作成します。
  4. 前のステップを繰り返し、`ExampleOutputStream` という名前の 1 つのシャードを持つストリームを作成します。

- AWS CLIを使用してデータストリームを作成するには

  - 次の Kinesis `create-stream` AWS CLI コマンドを使用して、ストリーム (`ExampleInputStream` および `ExampleOutputStream`) を作成します。アプリケーションが出力の書き込みに使用する 2 つめのストリームを作成するには、ストリーム名を `ExampleOutputStream` に変更して同じコマンドを実行します。

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

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

ステップ 1.2: 入力ストリームにサンプルレコードを書き込みます

このステップでは、Python コードを実行してサンプルレコードを連続生成し、`ExampleInputStream` ストリームに書き込みます。

```
{"x": 7.921782426109737, "y": 8.746265312709893, "is_hot": "N"}
{"x": 0.722248626580026, "y": 4.648868803193405, "is_hot": "Y"}
```

1. Python および pip をインストールします。

   Python のインストールについては、Python ウェブサイトをご覧ください。

   pip を使用して依存関係をインストールできます。pip のインストールについては、pip ウェブサイトの「Installation」を参照してください。

2. 以下の Python コードを実行します。このコードは以下の処理を実行します。

   - 潜在的なホットスポットを (X、Y) 平面のどこかに生成します。
   - ホットスポットごとに 1000 ポイントのセットを生成します。これらのポイントのうち、20 パーセントがホットスポットの周囲にクラスター化されています。残りはスペース全体でランダムに生成されます。
   - `put-record` コマンドは、ストリームに JSON レコードを書き込みます。

> **Important**
>
> このファイルには AWS 認証情報が含まれているため、このファイルをウェブサーバーにアップロードしないでください。

```
import json
from pprint import pprint
import random
import time
import boto3

STREAM_NAME = "ExampleInputStream"


def get_hotspot(field, spot_size):
    hotspot = {
        "left": field["left"] + random.random() * (field["width"] - spot_size),
        "width": spot_size,
        "top": field["top"] + random.random() * (field["height"] - spot_size),
        "height": spot_size,
    }
    return hotspot
```

```
def get_record(field, hotspot, hotspot_weight):
    rectangle = hotspot if random.random() < hotspot_weight else field
    point = {
        "x": rectangle["left"] + random.random() * rectangle["width"],
        "y": rectangle["top"] + random.random() * rectangle["height"],
        "is_hot": "Y" if rectangle is hotspot else "N",
    }
    return {"Data": json.dumps(point), "PartitionKey": "partition_key"}


def generate(
    stream_name, field, hotspot_size, hotspot_weight, batch_size, kinesis_client
):
    """
    Generates points used as input to a hotspot detection algorithm.
    With probability hotspot_weight (20%), a point is drawn from the hotspot;
    otherwise, it is drawn from the base field. The location of the hotspot
    changes for every 1000 points generated.
    """
    points_generated = 0
    hotspot = None
    while True:
        if points_generated % 1000 == 0:
            hotspot = get_hotspot(field, hotspot_size)
        records = [
            get_record(field, hotspot, hotspot_weight) for _ in range(batch_size)
        ]
        points_generated += len(records)
        pprint(records)
        kinesis_client.put_records(StreamName=stream_name, Records=records)

        time.sleep(0.1)


if __name__ == "__main__":
    generate(
        stream_name=STREAM_NAME,
        field={"left": 0, "width": 10, "top": 0, "height": 10},
        hotspot_size=1,
        hotspot_weight=0.2,
        batch_size=10,
        kinesis_client=boto3.client("kinesis"),
```

```
    )
```

次のステップ

ステップ 2: Kinesis Data Analytics アプリケーションを作成する

ステップ 2: Kinesis Data Analytics アプリケーションを作成する

ホットスポット例のこのセクションでは、Kinesis Data Analytics アプリケーションを次のように作成します。

- ステップ 1 で作成した Kinesis データストリームをストリーミングソースとして使用するように、アプリケーション入力を設定します。
- 提供されているアプリケーションコードを AWS Management Consoleで使用します。

アプリケーションを作成するには

1. 「使用開始」実習のステップ 1、2、および 3 (「ステップ 3.1: アプリケーションの作成」を参照) に従って Kinesis Data Analytics アプリケーションを作成します。

   ソース設定で、以下を実行します。

   - 作成したストリーミングソースを、the section called “ステップ 1: ストリームを作成する” で指定します。
   - コンソールがスキーマを推測した後、スキーマを編集します。x 列および y 列のタイプが DOUBLE に設定され、IS_HOT 列のタイプが VARCHAR に設定されていることを確認します。
2. 次のアプリケーションコードを使用します (このコードを SQL エディタに貼り付けることができます)。

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
    "x" DOUBLE,
    "y" DOUBLE,
    "is_hot" VARCHAR(4),
    HOTSPOTS_RESULT VARCHAR(10000)
);
CREATE OR REPLACE PUMP "STREAM_PUMP" AS
    INSERT INTO "DESTINATION_SQL_STREAM"
    SELECT "x", "y", "is_hot", "HOTSPOTS_RESULT"
```

```
    FROM TABLE (
        HOTSPOTS(
            CURSOR(SELECT STREAM "x", "y", "is_hot" FROM "SOURCE_SQL_STREAM_001"),
            1000,
            0.2,
            17)
    );
```

3. SQL コードを実行して結果を確認します。

Filter by column name

ROWTIME	x	y	is_hot	HOTSPOTS_RESULT
2018-03-19 20:19:20.298	3.2902233757560313	1.1460673734716675	N	{"hotspots":[{"density":159.34972933221212,"minValues":[0.4791038226753084,6.8274746613580275],"maxValues":[1.142036836117179,7.0925303040
2018-03-19 20:19:21.313	9.758694911135013	9.66632832516424	N	{"hotspots":[{"density":180.8921951354484,"minValues":[0.6623354726891375,6.8274746613580275],"maxValues":[1.142036836117179,7.0925303040
2018-03-19 20:19:21.313	8.986657300548824	3.558000293320571	N	{"hotspots":[{"density":180.8921951354484,"minValues":[0.6623354726891375,6.8274746613580275],"maxValues":[1.142036836117179,7.0925303040
2018-03-19 20:19:21.313	5.193048038272014	4.94448855569874	Y	{"hotspots":[{"density":180.8921951354484,"minValues":[0.6623354726891375,6.8274746613580275],"maxValues":[1.142036836117179,7.0925303040

次のステップ

ステップ 3: アプリケーション出力を設定する

ステップ 3: アプリケーション出力を設定する

ホットスポット例のこの時点で、Amazon Kinesis Data Analytics アプリケーションコードでストリーミングソースから重要なホットスポットを検出し、それぞれにヒートスコアを割り当てることができます。

これで、アプリケーションの結果をアプリケーション内ストリームから外部宛先である別の Kinesis データストリーム (`ExampleOutputStream`) に送信できます。その後、ホットスポットのスコアを分析して、ホットスポットのヒートに適したしきい値を決定することができます。このアプリケーションを拡張してアラートを生成できます。

アプリケーション出力を設定するには

1. Kinesis Data Analytics コンソール (https://console.aws.amazon.com/kinesisanalytics) を開きます。
2. SQL エディタのアプリケーションダッシュボードで、[Destination] または [Add a destination] を選択します。
3. [送信先の追加] ページで、[ストリームから選択] を選択します。前のセクションで作成した `ExampleOutputStream` ストリームを選択します。

こうしてできた外部宛先に、アプリケーションがアプリケーション内ストリーム `DESTINATION_SQL_STREAM` に書き込むレコードを Amazon Kinesis Data Analytics が永続化できます。

4. オプションで、ストリームをモニタリングしてアラートを送信する AWS Lambda ように `ExampleOutputStream` を設定できます。詳細については、「出力としての Lambda 関数の使用」を参照してください。ステップ 4: アプリケーション出力を検証する で説明するように、`ExampleOutputStream` が外部宛先である Kinesis ストリームに書き込んだレコードを確認することもできます。

次のステップ

ステップ 4: アプリケーション出力を検証する

ステップ 4: アプリケーション出力を検証する

ホットスポット例のこのセクションで、ホットスポット情報を Scalable Vector Graphics (SVG) コントロールに表示するウェブアプリケーションを設定します。

1. 次の内容で、`index.html` という名前のファイルを作成します。

```
<!doctype html>
<html lang=en>
<head>
    <meta charset=utf-8>
    <title>hotspots viewer</title>

    <style>
    #visualization {
      display: block;
      margin: auto;
    }

    .point {
      opacity: 0.2;
    }

    .hot {
      fill: red;
    }
```

```
    .cold {
      fill: blue;
    }

    .hotspot {
      stroke: black;
      stroke-opacity: 0.8;
      stroke-width: 1;
      fill: none;
    }
    </style>
    <script src="https://sdk.amazonaws.com/js/aws-sdk-2.202.0.min.js"></script>
    <script src="https://d3js.org/d3.v4.min.js"></script>
</head>
<body>
<svg id="visualization" width="600" height="600"></svg>
<script src="hotspots_viewer.js"></script>
</body>
</html>
```

2. 次の内容の hotspots_viewer.js というファイルを、同じディレクトリに作成します。認証情報、および出力ストリーム名を指定された変数に設定します。

```
// Visualize example output from the Kinesis Analytics hotspot detection algorithm.
// This script assumes that the output stream has a single shard.

// Modify this section to reflect your AWS configuration
var awsRegion = "",        // The  where your Kinesis Analytics application is
 configured.
    accessKeyId = "",      // Your Access Key ID
    secretAccessKey = "",  // Your Secret Access Key
    outputStream = "";     // The name of the Kinesis Stream where the output from
 the HOTSPOTS function is being written

// The variables in this section should reflect way input data was generated and
 the parameters that the HOTSPOTS
// function was called with.
var windowSize = 1000, // The window size used for hotspot detection
    minimumDensity = 40,  // A filter applied to returned hotspots before
 visualization
    xRange = [0, 10],  // The range of values to display on the x-axis
    yRange = [0, 10];  // The range of values to display on the y-axis
```

```
////////////////////////////////////////////////////////////////////////////////////////////////
// D3 setup
////////////////////////////////////////////////////////////////////////////////////////////////

var svg = d3.select("svg"),
    margin = {"top": 20, "right": 20, "bottom": 20, "left": 20},
    graphWidth = +svg.attr("width") - margin.left - margin.right,
    graphHeight = +svg.attr("height") - margin.top - margin.bottom;

// Return the linear function that maps the segment [a, b] to the segment [c, d].
function linearScale(a, b, c, d) {
    var m = (d - c) / (b - a);
    return function(x) {
        return c + m * (x - a);
    };
}

// helper functions to extract the x-value from a stream record and scale it for
 output
var xValue = function(r) { return r.x; },
    xScale = linearScale(xRange[0], xRange[1], 0, graphWidth),
    xMap = function(r) { return xScale(xValue(r)); };

// helper functions to extract the y-value from a stream record and scale it for
 output
var yValue = function(r) { return r.y; },
    yScale = linearScale(yRange[0], yRange[1], 0, graphHeight),
    yMap = function(r) { return yScale(yValue(r)); };

// a helper function that assigns a CSS class to a point based on whether it was
 generated as part of a hotspot
var classMap = function(r) { return r.is_hot == "Y" ? "point hot" : "point
 cold"; };

var g = svg.append("g")
    .attr("transform", "translate(" + margin.left + "," + margin.top + ")");

function update(records, hotspots) {

    var points = g.selectAll("circle")
        .data(records, function(r) { return r.dataIndex; });

    points.enter().append("circle")
```

```
        .attr("class", classMap)
        .attr("r", 3)
        .attr("cx", xMap)
        .attr("cy", yMap);

    points.exit().remove();

    if (hotspots) {
        var boxes = g.selectAll("rect").data(hotspots);

        boxes.enter().append("rect")
            .merge(boxes)
            .attr("class", "hotspot")
            .attr("x", function(h) { return xScale(h.minValues[0]); })
            .attr("y", function(h) { return yScale(h.minValues[1]); })
            .attr("width", function(h) { return xScale(h.maxValues[0]) -
 xScale(h.minValues[0]); })
            .attr("height", function(h) { return yScale(h.maxValues[1]) -
 yScale(h.minValues[1]); });

        boxes.exit().remove();
    }
}

////////////////////////////////////////////////////////////////////////////////////////////
// Use the AWS SDK to pull output records from Kinesis and update the visualization
////////////////////////////////////////////////////////////////////////////////////////////

var kinesis = new AWS.Kinesis({
    "region": awsRegion,
    "accessKeyId": accessKeyId,
    "secretAccessKey": secretAccessKey
});

var textDecoder = new TextDecoder("utf-8");

// Decode an output record into an object and assign it an index value
function decodeRecord(record, recordIndex) {
    var record = JSON.parse(textDecoder.decode(record.Data));
    var hotspots_result = JSON.parse(record.HOTSPOTS_RESULT);
    record.hotspots = hotspots_result.hotspots
        .filter(function(hotspot) { return hotspot.density >= minimumDensity});
    record.index = recordIndex
    return record;
```

```
}

// Fetch a new records from the shard iterator, append them to records, and update
 the visualization
function getRecordsAndUpdateVisualization(shardIterator, records, lastRecordIndex)
 {
    kinesis.getRecords({
        "ShardIterator": shardIterator
    }, function(err, data) {
        if (err) {
            console.log(err, err.stack);
            return;
        }

        var newRecords = data.Records.map(function(raw) { return decodeRecord(raw,
 ++lastRecordIndex); });
        newRecords.forEach(function(record) { records.push(record); });

        var hotspots = null;
        if (newRecords.length > 0) {
            hotspots = newRecords[newRecords.length - 1].hotspots;
        }

        while (records.length > windowSize) {
            records.shift();
        }

        update(records, hotspots);

        getRecordsAndUpdateVisualization(data.NextShardIterator, records,
 lastRecordIndex);
    });
}

// Get a shard iterator for the output stream and begin updating the visualization.
 Note that this script will only
// read records from the first shard in the stream.
function init() {
    kinesis.describeStream({
        "StreamName": outputStream
    }, function(err, data) {
        if (err) {
            console.log(err, err.stack);
            return;
```

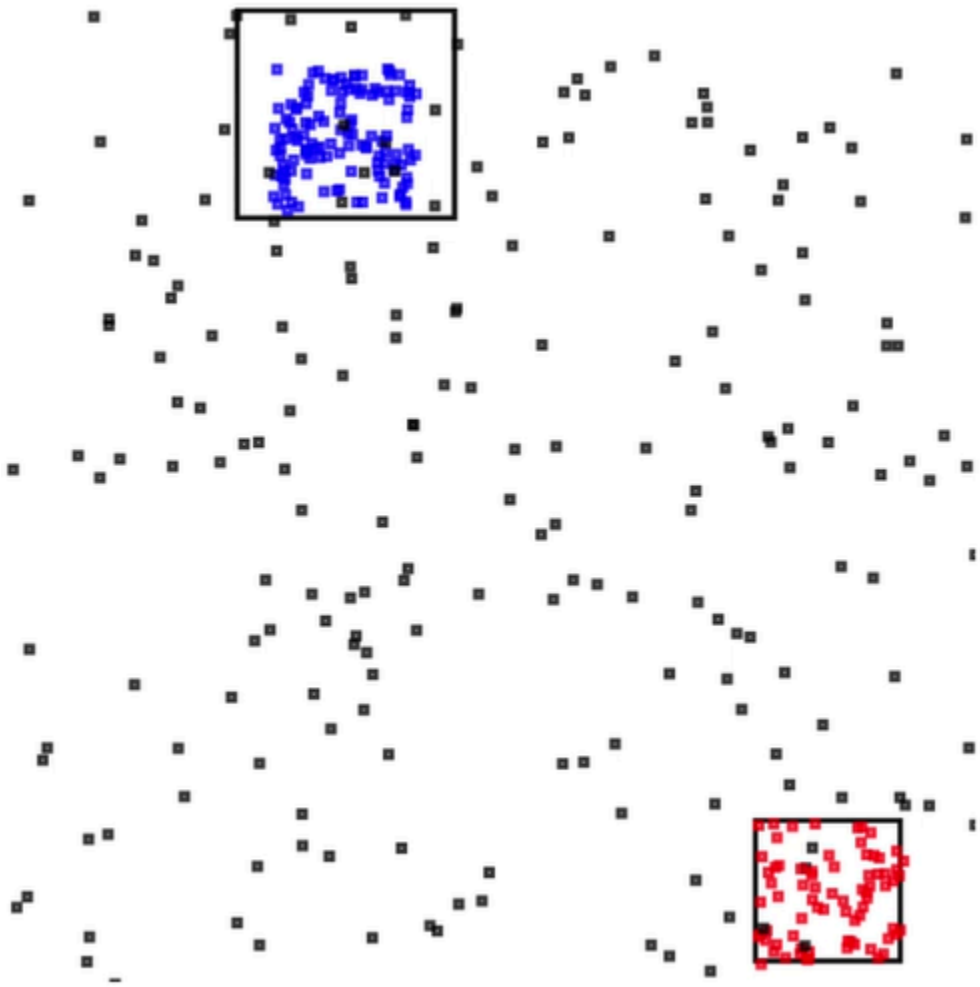
```
        }

        var shardId = data.StreamDescription.Shards[0].ShardId;

        kinesis.getShardIterator({
            "StreamName": outputStream,
            "ShardId": shardId,
            "ShardIteratorType": "LATEST"
        }, function(err, data) {
            if (err) {
                console.log(err, err.stack);
                return;
            }
            getRecordsAndUpdateVisualization(data.ShardIterator, [], 0);
        })
    });
}

// Start the visualization
init();
```

3. 最初のセクションの Python コードを実行し、ウェブブラウザで `index.html` を開きます。ホットスポット情報がページに次のように表示されます。



例: アラートとエラー

このセクションでは、アラートとエラーを使用する Kinesis Data Analytics アプリケーションの例を示します。それぞれの例では、Kinesis Data Analytics アプリケーションをセットアップし、テストするための詳しい手順とコードを示します。

トピック

- 例: 簡単なアラートの作成
- 例: 調整されたアラートの作成
- 例: アプリケーション内エラーストリームの確認

例: 簡単なアラートの作成

この Kinesis Data Analytics アプリケーションでは、デモストリームを基に作成されたアプリケーション内ストリームに対して、クエリが継続的に実行されます。詳細については、「連続クエリ」を参照してください。

株価の変動が 1% を超えることを示す行があれば、それらの行はアプリケーション内ストリームに挿入されます。実習では、結果を永続化して外部宛先に書き込むようにアプリケーション出力を設定できます。その後、結果を調査できます。例えば、 AWS Lambda 関数を使用してレコードを処理し、アラートを送信できます。

シンプルなアラートアプリケーションを作成するには

1. Kinesis Data Analytics の開始方法の実習に従って分析アプリケーションを作成します。
2. Kinesis Data Analytics の SQL エディタで、アプリケーションコードを以下に置き換えます。

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM"
           (ticker_symbol VARCHAR(4),
            sector        VARCHAR(12),
            change        DOUBLE,
            price         DOUBLE);

CREATE OR REPLACE PUMP "STREAM_PUMP" AS
   INSERT INTO "DESTINATION_SQL_STREAM"
      SELECT STREAM ticker_symbol, sector, change, price
      FROM   "SOURCE_SQL_STREAM_001"
      WHERE  (ABS(Change / (Price - Change)) * 100) > 1;
```

   アプリケーションコード内の SELECT ステートメントは、SOURCE_SQL_STREAM_001 で、株価の変動が 1% を超える行をフィルタリングします。次に、ポンプを使用して、それらの行を別のアプリケーション内ストリーム (DESTINATION_SQL_STREAM) に挿入します。ポンプを使用したアプリケーション内ストリームへの行の挿入を記述するコーディングパターンの詳細については、「アプリケーションコード」を参照してください。
3. [Save and run SQL] を選択します。
4. 宛先を追加します。これを行うには、SQL エディタで [送信先] タブを選択するか、アプリケーション詳細ページで [送信先の追加] を選択します。

   a. SQL エディタで、[送信先] タブを選択し、[送信先への接続] を選択します。

[送信先への接続] ページで、[新規作成] を選択します。

b. [Go to Kinesis Streams] を選択します。

c. Amazon Kinesis Data Streams コンソールで、1 つのシャードを持つ新規の Kinesis ストリーム (`gs-destination` など) を作成します。ストリームのステータスが [ACTIVE] になるまで待ちます。

d. Kinesis Data Analytics コンソールに戻ります。[送信先への接続] ページで、作成したストリームを選択します。

   ストリームが表示されない場合は、ページを更新してください。

e. [Save and continue] を選択します。

外部宛先として Kinesis データストリームができ、Kinesis Data Analytics はそこに `DESTINATION_SQL_STREAM` アプリケーション内ストリームのアプリケーション出力を永続化できます。

5. 作成した Kinesis ストリームをモニタリングし、Lambda 関数を呼び出す AWS Lambda ように を設定します。

   手順については、Lambda 関数を使用したデータの事前処理 を参照してください。

例: 調整されたアラートの作成

この Kinesis Data Analytics アプリケーションでは、デモストリームを基に作成されたアプリケーション内ストリームに継続的にクエリが実行されます。詳細については、「連続クエリ」を参照してください。株価の変動が 1% を超えることを示す行があれば、それらの行はアプリケーション内ストリームに挿入されます。アプリケーションはアラートを調整し、株価が変動したときにただちにアラートが送信されるようにします。ただし、1 分あたり、株式シンボルあたり複数のアラートがアプリケーション内ストリームに送信されることはありません。

調整されたアラートアプリケーションを作成するには

1. Kinesis Data Analytics の開始方法の実習に従って Kinesis Data Analytics アプリケーションを作成します。
2. Kinesis Data Analytics の SQL エディタで、アプリケーションコードを以下に置き換えます。

```
CREATE OR REPLACE STREAM "CHANGE_STREAM"
           (ticker_symbol VARCHAR(4),
```

```
            sector        VARCHAR(12),
            change        DOUBLE,
            price         DOUBLE);

CREATE OR REPLACE PUMP "change_pump" AS
   INSERT INTO "CHANGE_STREAM"
      SELECT STREAM ticker_symbol, sector, change, price
      FROM   "SOURCE_SQL_STREAM_001"
      WHERE  (ABS(Change / (Price - Change)) * 100) > 1;

-- ** Trigger Count and Limit **
-- Counts "triggers" or those values that evaluated true against the previous where
 clause
-- Then provides its own limit on the number of triggers per hour per ticker symbol
 to what
-- is specified in the WHERE clause

CREATE OR REPLACE STREAM TRIGGER_COUNT_STREAM (
   ticker_symbol VARCHAR(4),
   change REAL,
   trigger_count INTEGER);

CREATE OR REPLACE PUMP trigger_count_pump AS INSERT INTO TRIGGER_COUNT_STREAM
SELECT STREAM ticker_symbol, change, trigger_count
FROM (
    SELECT STREAM ticker_symbol, change, COUNT(*) OVER W1 as trigger_count
    FROM "CHANGE_STREAM"
    --window to perform aggregations over last minute to keep track of triggers
    WINDOW W1 AS (PARTITION BY ticker_symbol RANGE INTERVAL '1' MINUTE PRECEDING)
)
WHERE trigger_count >= 1;
```

アプリケーション内の SELECT ステートメントは、SOURCE_SQL_STREAM_001 の行を、1 パーセントを超える株価の変動でフィルタリングして、ポンプを使用してそれらの行を別のアプリケーション内ストリーム CHANGE_STREAM に挿入します。

アプリケーションは、調整されたアラートに TRIGGER_COUNT_STREAM と呼ばれる 2 番目のストリームを作成します。2 番目のクエリはレコードが入力されるたびにホップするウィンドウからレコードを選択します。これにより、株価ティッカーごとに 1 分あたり 1 つのレコードのみがストリームに書き込まれます。

3. [Save and run SQL] を選択します。

例では、TRIGGER_COUNT_STREAM に次のようなストリームが出力されます。

Filter by column name

ROWTIME	TICKER_SYMBOL	CHANGE	TRIGGER_COUNT
2018-01-08 22:59:15.742	ASD	-1.77	1
2018-01-08 22:59:15.742	ASD	-1.77	1
2018-01-08 22:59:20.752	DFT	-3.16	1
2018-01-08 22:59:35.775	IOP	-1.88	1

例: アプリケーション内エラーストリームの確認

Amazon Kinesis Data Analytics は作成した各アプリケーションのアプリケーション内エラーストリームを提供します。アプリケーションが処理できない行は、このエラーストリームに送信されます。エラーストリームのデータを外部宛先で永続化して、調査できるようにすることを検討してください。

コンソールで次の演習を実行します。これらの例では、検出プロセスで推測されたスキーマを編集して入力設定にエラーを発生させ、次にエラーストリームに送信される行を検証します。

トピック

- 解析エラーの確認
- ゼロ除算エラーの導入

解析エラーの確認

この実習では、解析エラーを確認します。

1. Kinesis Data Analytics の開始方法の実習に従って Kinesis Data Analytics アプリケーションを作成します。
2. アプリケーション詳細ページで、[ストリーミングデータの接続] を選択します。
3. 「使用開始」実習を行った場合は、アカウントにデモストリーム (`kinesis-analytics-demo-stream`) があります。[ソースへの接続] ページで、このデモストリームを選択します。
4. Kinesis Data Analytics はデモストリームからサンプルを取得して、作成するアプリケーション内入力ストリームのスキーマを推測します。コンソールの [Formatted stream sample] タブに推測スキーマとサンプルデータが表示されます。

5. 次に、スキーマを編集して列タイプを変更し、解析エラーを発生させます。[Edit schema] を選択します。
6. `TICKER_SYMBOL` の列タイプを `VARCHAR(4)` から `INTEGER` に変更します。

   作成されたアプリケーション内スキーマの列のタイプが無効なため、Kinesis Data Analytics は、アプリケーション内ストリームにデータを持ち込むことはできません。代わりに、行がエラーストリームに送信されます。
7. [Save schema] を選択します。
8. [Refresh schema samples] を選択します。

   [Formatted stream] 例には行がないことに注意してください。ただし、[Error stream] タブには、エラーメッセージ付きのデータが表示されます。[Error stream] タブに、アプリケーション内エラーストリームに送信されたデータが表示されます。

   列のデータ型を変更したので、Kinesis Data Analytics はアプリケーション内入力ストリームにデータを持ち込むことができませんでした。代わりに、エラーストリームにデータが送信されました。

ゼロ除算エラーの導入

この練習では、ランタイムエラー (ゼロ除算) を導入するようにアプリケーションコードを更新します。結果が書き込まれるはずの `DESTINATION_SQL_STREAM` アプリケーション内ストリームにではなく、このアプリケーション内エラーストリームに Amazon Kinesis Data Analytics が結果の行を送信することに注意してください。

1. Kinesis Data Analytics の開始方法の実習に従って Kinesis Data Analytics アプリケーションを作成します。

   [Real-time analytics] タブで次のとおり結果を確認します。

   Sour
2. アプリケーションコードで `SELECT` ステートメントを更新してゼロ除算を発生させます。その例を次に示します。

```
SELECT STREAM ticker_symbol, sector, change, (price / 0) as ProblemColumn
FROM "SOURCE_SQL_STREAM_001"
WHERE sector SIMILAR TO '%TECH%';
```

3. アプリケーションを実行します。

   ゼロ除算ランタイムエラーが発生するため、Kinesis Data Analytics は結果を `DESTINATION_SQL_STREAM` に書き込む代わりに、アプリケーション内エラーストリームに行を送信します。[リアルタイム分析] タブで、エラーストリームを選択すると、アプリケーション内ストリームの行が表示されます。

例: ソリューションアクセラレーター

AWS ソリューションサイトには、完全なストリーミングデータソリューションをすばやく作成するために使用できる AWS CloudFormation テンプレートがあります。

次のテンプレートを使用できます。

AWS アカウント アクティビティに関するリアルタイムのインサイト

このソリューションは、 AWS アカウント (複数可) のリソースアクセスと使用状況メトリクスをリアルタイムで記録および視覚化します。詳細については、AWS アカウント 「アクティビティに関するリアルタイムインサイト」を参照してください。

Kinesis Data Analytics によるリアルタイムの AWS IoT デバイスモニタリング

このソリューションでは、IoT デバイスの接続とアクティビティデータをリアルタイムで収集、処理、分析、視覚化します。詳細については、「Kinesis Data Analytics によるリアルタイム AWS IoT デバイスモニタリング」を参照してください。

Kinesis Data Analytics を使ったリアルタイムウェブ分析

このソリューションでは、ウェブサイトのクリックストリームデータをリアルタイムで収集、処理、分析、視覚化します。詳細については、「Real-time web analytics with Kinesis Data Analytics」を参照してください。

Amazon コネクテッドカーソリューション

このソリューションでは、車両からの IoT データをリアルタイムで収集、処理、分析、視覚化します。詳細については、「Amazon Connected Vehicle Solution」を参照してください。

Amazon Kinesis Data Analytics でのセキュリティ

でのクラウドセキュリティが最優先事項 AWS です。お客様は AWS 、セキュリティを最も重視する組織の要件を満たすように構築されたデータセンターとネットワークアーキテクチャを活用できます。

セキュリティは、 AWS とお客様の間の責任共有です。責任共有モデルでは、これをクラウドのセキュリティおよびクラウド内のセキュリティとして説明しています。

- クラウドのセキュリティ – AWS クラウドで AWS サービスを実行するインフラストラクチャを保護する AWS 責任があります。 AWS また、 では、安全に使用できるサービスも提供しています。セキュリティの有効性は、AWS コンプライアンスプログラムの一環として、サードパーティーの審査機関によって定期的にテストおよび検証されています。Kinesis Data Analytics に適用されるコンプライアンスプログラムについては、「コンプライアンスプログラムによる対象範囲内の AWS のサービス」を参照してください。
- クラウド内のセキュリティ – お客様の責任は、使用する AWS サービスによって決まります。また、お客様は、お客様のデータの機密性、組織の要件、および適用可能な法律および規制などの他の要因についても責任を担います。

このドキュメントは、Kinesis Data Analytics の使用時に責任共有モデルがどのように適用されるかを理解するために役立ちます。以下のトピックでは、セキュリティおよびコンプライアンスの目的を達成するように Kinesis Data Analytics を設定する方法を説明します。また、Kinesis Data Analytics リソースのモニタリングや保護に役立つ Amazon のその他のサービスを使用する方法についても説明します。

トピック

- Amazon Kinesis Data Analytics for SQL Applications でのデータ保護
- Kinesis Data Analytics の Identity and Access Management
- に対する認証とアクセスコントロール
- Amazon Kinesis Data Analytics のモニタリング
- Amazon Kinesis Data Analytics for SQL Applications のコンプライアンス検証
- Amazon Kinesis Data Analytics の耐障害性
- Kinesis Data Analytics for SQL アプリケーションのインフラストラクチャセキュリティ
- Kinesis Data Analytics のセキュリティのベストプラクティス

Amazon Kinesis Data Analytics for SQL Applications でのデータ保護

が提供するツールを使用してデータを保護できます AWS。Kinesis Data Analytics は、Kinesis Data Streams、Firehose、Amazon S3 など、データの暗号化をサポートするサービスと連携できます。

Kinesis Data Analytics でのデータ暗号化

保管時の暗号化

Kinesis Data Analytics による保管中のデータの暗号化については、以下の点に注意してください。

- 受信する Kinesis Data Streamsのデータは、StartStreamEncryption を使用して暗号化できます。詳細については、「Kinesis Data Streams用のサーバー側の暗号化とは」を参照してください。
- 出力データは、保管時に Firehose を使用して暗号化し、暗号化された Amazon S3 バケットに格納できます。 バケットが使用する暗号化キーを指定できます。詳細については、「KMS マネージドキーによるサーバー側の暗号化 (SSE-KMS) を使用したデータの保護」を参照してください。
- アプリケーションのコードは保管時に暗号化されます。
- アプリケーションの参照データは保管時に暗号化されます。

転送中の暗号化

Kinesis Data Analytics は、転送中のすべてのデータを暗号化します。転送中の暗号化は、すべての Kinesis Data Analytics アプリケーションで有効になり、無効にすることはできません。

Kinesis Data Analytics は以下のシナリオで転送中のデータを暗号化します。

- Kinesis Data Streams から Kinesis Data Analytics に転送中のデータ。
- Kinesis Data Analytics 内の内部コンポーネント間で転送中のデータ。
- Kinesis Data Analytics と Firehose 間で転送中のデータ。

キーの管理

Kinesis Data Analytics のデータ暗号化では、サービスで管理されたキーが使用されます。カスタマー管理のキーはサポートされていません。

Kinesis Data Analytics の Identity and Access Management

Amazon Kinesis Data Analytics には、アプリケーション入力設定で指定されたストリーミングソースからレコードを読み取るためのアクセス権限が必要です。Amazon Kinesis Data Analytics には、アプリケーション出力設定で指定されたストリームにアプリケーション出力を書き込むためのアクセス権限も必要です。

こうしたアクセス権限は、Amazon Kinesis Data Analytics が引き受けることのできる IAM ロールを作成することで付与できます。このロールに付与するアクセス権限によって、サービスがそのロールを引き受けたときに Amazon Kinesis Data Analytics が実行する内容が決まります。

> **Note**
>
> このセクションの情報は、IAM ロールを独自に作成する場合に役立ちます。Amazon Kinesis Data Analytics コンソールでアプリケーションを作成する場合、コンソールはその時点で IAM ロールを作成します。コンソールは、作成する IAM ロールに以下の命名規則を使用します。
>
> ```
> kinesis-analytics-ApplicationName
> ```
>
> ロールが作成されたら、ロールおよびアタッチされたポリシーを IAM コンソールで確認できます。

各 IAM ロールには、2 つのポリシーがアタッチされます。信頼ポリシーでは、だれがこのロールを引き受けることができるかを指定します。アクセス権限ポリシー (1 つまたは複数の場合があります) では、このロールに付与するアクセス権限を指定します。次のセクションで、IAM ロールの作成時に使用できるこうしたポリシーについて説明します。

信頼ポリシー

ストリーミングソースやリファレンスソースにアクセスするロールを引き受ける権限を Amazon Kinesis Data Analytics に付与するには、以下の信頼ポリシーを IAM ロールにアタッチします。

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

```
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "kinesisanalytics.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

アクセス許可ポリシー

アプリケーションのストリーミングソースからの読み込みを Amazon Kinesis Data Analytics に許可する IAM ロールを作成する場合は、関連する読み取りアクションのアクセス権限を付与する必要があります。ソース (Kinesis ストリーム、Firehose 配信ストリーム、Amazon S3 バケット内の参照ソースなど) に応じて、次のアクセス許可ポリシーをアタッチします。

Kinesis ストリームを読み取るためのアクセス権限ポリシー

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ReadInputKinesis",
            "Effect": "Allow",
            "Action": [
                "kinesis:DescribeStream",
                "kinesis:GetShardIterator",
                "kinesis:GetRecords",
                "kinesis:ListShards"
            ],
            "Resource": [
                "arn:aws:kinesis:aws-region:aws-account-id:stream/inputStreamName"
            ]
        }
    ]
}
```

Firehose 配信ストリームを読み取るためのアクセス許可ポリシー

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ReadInputFirehose",
            "Effect": "Allow",
            "Action": [
                "firehose:DescribeDeliveryStream",
                "firehose:Get*"
            ],
            "Resource": [
                "arn:aws:firehose:aws-region:aws-account-
id:deliverystream/inputFirehoseName"
            ]
        }
    ]
}
```

> **Note**
>
> `firehose:Get*` アクセス権限とは、Kinesis Data Analytics がストリームへのアクセスに使用する内部アクセサーを指します。Firehose 配信ストリーム用のパブリックアクセサーはありません。

アプリケーション出力設定で Amazon Kinesis Data Analytics が外部宛先に出力を書き込むよう指定している場合は、次のアクセス権限を IAM ロールに付与する必要があります。

Kinesis ストリームに書き込むためのアクセス権限ポリシー

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "WriteOutputKinesis",
            "Effect": "Allow",
            "Action": [
                "kinesis:DescribeStream",
                "kinesis:PutRecord",
                "kinesis:PutRecords"
            ],
            "Resource": [
                "arn:aws:kinesis:aws-region:aws-account-id:stream/output-stream-name"
```

```
            ]
        }
    ]
}
```

Firehose 配信ストリームに書き込むためのアクセス権限ポリシー

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "WriteOutputFirehose",
            "Effect": "Allow",
            "Action": [
                "firehose:DescribeDeliveryStream",
                "firehose:PutRecord",
                "firehose:PutRecordBatch"
            ],
            "Resource": [
                "arn:aws:firehose:aws-region:aws-account-id:deliverystream/output-
firehose-name"
            ]
        }
    ]
}
```

Amazon S3 バケットからリファレンスデータソースを読み取るためのアクセス権限ポリシー

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:Get*",
        "s3:List*"
      ],
      "Resource": "*"
    }
```

```
  ]
}
```

サービス間の混乱した代理の防止

では AWS、あるサービス (呼び出し元のサービス) が別のサービス (呼び出し元のサービス) を呼び出すと、サービス間のなりすましが発生する可能性があります。呼び出し側のサービスは、適切なアクセス許可を持たないはずの場合でも、別の顧客のリソースを操作するように操作される可能性があり、その結果、混乱した代理問題が発生します。

混乱した代理を防ぐために、 は、アカウント内のリソースへのアクセス権が付与されたサービスプリンシパルを使用して、すべてのサービスのデータを保護するのに役立つツール AWS を提供します。このセクションでは、Kinesis Data Analytics に固有のサービス間での混乱した代理防止に焦点を当てていますが、このトピックの詳細については、IAM ユーザーガイドの「混乱する代理問題」セクションを参照してください。

Kinesis Data Analytics for SQL のコンテキストでは、ロール信頼ポリシーに aws:SourceArn および aws:SourceAccount のグローバル条件コンテキストキーを使用して、期待されるリソースによって生成されたリクエストのみに、ロールのアクセスを制限することをお勧めします。

クロスサービスアクセスにリソースを 1 つだけ関連付けたい場合は、`aws:SourceArn` を使用します。そのアカウント内のリソースをクロスサービスの使用に関連付けることを許可する場合は、`aws:SourceAccount` を使用します。

`aws:SourceArn` の値は、Kinesis Data Analytics が使用するリソースの ARN でなければなりません。この値は `arn:aws:kinesisanalytics:region:account:resource` 形式で指定されます。

混乱した代理問題から保護するために推奨されるアプローチは、リソースの完全な ARN を指定しながら、`aws:SourceArn` グローバル条件コンテキストキーを使用することです。

リソースの完全な ARN が不明な場合や、複数のリソースを指定する場合には、`aws:SourceArn`キー で、ARN の未知部分を示すためにワイルドカード文字 (*) を使用します。例: `arn:aws:kinesisanalytics::111122223333:*`。

CreateApplication、AddApplicationInput、DeleteApplication など、Kinesis Data Analytics for SQL API のほとんどのアクションは特定のアプリケーションのコンテキストで実行されますが、DiscoverInputSchema アクションはどのアプリケーションのコンテキストでも実行されません。つまり、このアクションで使用されるロールでは、`SourceArn` 条件キーにリソースを完全に指定してはなりません。ワイルドカード ARN を使用する例を以下に示します。

```
{
   ...
   "ArnLike":{
      "aws:SourceArn":"arn:aws:kinesisanalytics:us-east-1:123456789012:*"
   }
   ...
}
```

Kinesis Data Analytics for SQL によって生成されるデフォルトのロールは、このワイルドカードを使用します。これにより、コンソールでの入力スキーマの検出がシームレスに機能します。ただし、完全に混乱した代理の緩和策を実装するには、スキーマを発見した後で信頼ポリシーを編集し、完全な ARN を使用するようお勧めします。

Kinesis Data Analytics に提供するロールのポリシーだけでなく、ユーザー向けに生成されるロールの信頼ポリシーは、aws:SourceArn と aws:SourceAccount の条件キーを使用できます。

混乱した代理問題から保護するために、次の手順を実行します。

「混乱した代理」問題からの保護

1. AWS マネジメントコンソールにサインインし、https://console.aws.amazon.com/iam/ で IAM コンソールを開きます。
2. ロールを選択して、変更したいロールを選択します。
3. [信頼ポリシーを編集] を選択します。
4. 信頼ポリシーの編集ページで、デフォルトの JSON ポリシーを、`aws:SourceArn`および`aws:SourceAccount`グローバル条件コンテキストキーのいずれかまたは両方を使用するポリシーに置き換えます。以下のポリシー例を参照してください。
5. [ポリシーの更新] を選択してください。

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

```
                "aws:SourceAccount":"Account ID"
            },
            "ArnEquals":{
                "aws:SourceArn":"arn:aws:kinesisanalytics:us-
east-1:123456789012:application/my-app"
            }
        }
    }
  ]
}
```

に対する認証とアクセスコントロール

へのアクセスには、認証情報が必要です。これらの認証情報には、アプリケーションや Amazon Elastic Compute Cloud (Amazon EC2) インスタンスなどの AWS リソースにアクセスするためのアクセス許可が必要です。次のセクションでは、AWS Identity and Access Management (IAM) および を使用して、リソースへのアクセスを保護する方法について詳しく説明します。

アクセスコントロール

有効な認証情報があればリクエストを認証できますが、アクセス許可が付与されている場合を除き、リソースの作成やアクセスはできません。たとえば、 アプリケーションの作成にはアクセス権限が必要です。

次のセクションでは、 の許可を管理する方法について説明します。最初に概要のセクションを読むことをお勧めします。

- リソースへのアクセス許可の管理の概要
- でアイデンティティベースのポリシー (IAM ポリシー) を使用する
- API アクセス許可: アクション、アクセス許可、リソースの参照

アイデンティティを使用した認証

認証とは、ID 認証情報 AWS を使用して にサインインする方法です。として、IAM ユーザーとして AWS アカウントのルートユーザー、または IAM ロールを引き受けることによって、認証 (にサインイン AWS) される必要があります。

ID ソースを介して提供された認証情報を使用して、フェデレーティッド ID AWS として にサインインできます。 AWS IAM Identity Center （ IAM Identity Center) ユーザー、会社のシングルサインオン

認証、Google または Facebook 認証情報は、フェデレーション ID の例です。フェデレーティッド ID としてサインインする場合、IAM ロールを使用して、前もって管理者により ID フェデレーションが設定されています。フェデレーション AWS を使用して にアクセスすると、間接的にロールを引き受けることになります。

ユーザーのタイプに応じて、 AWS Management Console または AWS アクセスポータルにサインインできます。へのサインインの詳細については AWS、「 AWS サインイン ユーザーガイド」の「 へのサインイン AWS アカウント方法」を参照してください。

AWS プログラムで にアクセスする場合、 はソフトウェア開発キット (SDK) とコマンドラインインターフェイス (CLI) AWS を提供し、認証情報を使用してリクエストを暗号化して署名します。AWS ツールを使用しない場合は、自分でリクエストに署名する必要があります。リクエストに自分で署名する推奨方法の使用については、「IAM ユーザーガイド」の「API リクエストに対するAWS Signature Version 4」を参照してください。

使用する認証方法を問わず、追加セキュリティ情報の提供をリクエストされる場合もあります。たとえば、 では、アカウントのセキュリティを高めるために多要素認証 (MFA) を使用する AWS ことをお勧めします。詳細については、「AWS IAM Identity Center ユーザーガイド」の「多要素認証」および「IAM ユーザーガイド」の「IAM のAWS 多要素認証」を参照してください。

AWS アカウント ルートユーザー

を作成するときは AWS アカウント、アカウント内のすべての およびリソースへの AWS のサービス 完全なアクセス権を持つ 1 つのサインインアイデンティティから始めます。この ID は AWS アカウント ルートユーザーと呼ばれ、アカウントの作成に使用した E メールアドレスとパスワードでサインインすることでアクセスできます。日常的なタスクには、ルートユーザーを使用しないことを強くお勧めします。ルートユーザーの認証情報は保護し、ルートユーザーでしか実行できないタスクを実行するときに使用します。ルートユーザーとしてサインインする必要があるタスクの完全なリストについては、「IAM ユーザーガイド」の「ルートユーザー認証情報が必要なタスク」を参照してください。

フェデレーティッドアイデンティティ

ベストプラクティスとして、管理者アクセスを必要とするユーザーを含む人間のユーザーに、ID プロバイダーとのフェデレーションを使用して一時的な認証情報 AWS のサービス を使用して にアクセスすることを要求します。

フェデレーティッド ID は、エンタープライズユーザーディレクトリ、ウェブ ID プロバイダー、AWS Directory Service、アイデンティティセンターディレクトリ、または ID ソースを介して提供さ

れた認証情報 AWS のサービス を使用して にアクセスするユーザーです。フェデレーティッド ID がアクセスすると AWS アカウント、ロールを引き受け、ロールは一時的な認証情報を提供します。

アクセスを一元管理する場合は、 AWS IAM Identity Centerを使用することをお勧めします。IAM Identity Center でユーザーとグループを作成するか、独自の ID ソースのユーザーとグループのセットに接続して同期し、すべての AWS アカウント とアプリケーションで使用できます。IAM Identity Center の詳細については、「AWS IAM Identity Center ユーザーガイド」の「What is IAM Identity Center?」(IAM Identity Center とは) を参照してください。

IAM ユーザーとグループ

IAM ユーザーは、単一のユーザーまたはアプリケーションに対して特定のアクセス許可 AWS アカウント を持つ 内の ID です。可能であれば、パスワードやアクセスキーなどの長期的な認証情報を保有する IAM ユーザーを作成する代わりに、一時的な認証情報を使用することをお勧めします。ただし、IAM ユーザーでの長期的な認証情報が必要な特定のユースケースがある場合は、アクセスキーをローテーションすることをお勧めします。詳細については、「IAM ユーザーガイド」の「長期的な認証情報を必要とするユースケースのためにアクセスキーを定期的にローテーションする」を参照してください。

IAM グループは、IAM ユーザーの集団を指定するアイデンティティです。グループとしてサインインすることはできません。グループを使用して、複数のユーザーに対して一度に権限を指定できます。多数のユーザーグループがある場合、グループを使用することで権限の管理が容易になります。例えば、IAMAdmins という名前のグループを設定して、そのグループに IAM リソースを管理する許可を与えることができます。

ユーザーは、ロールとは異なります。ユーザーは 1 人の人または 1 つのアプリケーションに一意に関連付けられますが、ロールはそれを必要とする任意の人が引き受けるようになっています。ユーザーには永続的な長期の認証情報がありますが、ロールでは一時認証情報が提供されます。詳細については、「IAM ユーザーガイド」の「IAM ユーザーに関するユースケース」を参照してください。

IAM ロール

IAM ロールは、特定のアクセス許可 AWS アカウント を持つ 内の ID です。これは IAM ユーザーに似ていますが、特定のユーザーには関連付けられていません。で IAM ロールを一時的に引き受けるには AWS Management Console、ユーザーから IAM ロール (コンソール) に切り替えることができます。ロールを引き受けるには、 または AWS API オペレーションを AWS CLI 呼び出すか、カスタム URL を使用します。ロールを使用する方法の詳細については、「IAM ユーザーガイド」の「ロールを引き受けるための各種方法」を参照してください。

IAM ロールと一時的な認証情報は、次の状況で役立ちます:

- フェデレーションユーザーアクセス – フェデレーティッド ID に許可を割り当てるには、ロールを作成してそのロールの許可を定義します。フェデレーティッド ID が認証されると、その ID はロールに関連付けられ、ロールで定義されている許可が付与されます。フェデレーションのロールについては、「IAM ユーザーガイド」の「サードパーティー ID プロバイダー (フェデレーション) 用のロールを作成する」を参照してください。IAM Identity Center を使用する場合は、許可セットを設定します。アイデンティティが認証後にアクセスできるものを制御するため、IAM Identity Center は、権限セットを IAM のロールに関連付けます。アクセス許可セットの詳細については、「AWS IAM Identity Center User Guide」の「Permission sets」を参照してください。
- 一時的な IAM ユーザー権限 - IAM ユーザーまたはロールは、特定のタスクに対して複数の異なる権限を一時的に IAM ロールで引き受けることができます。
- クロスアカウントアクセス - IAM ロールを使用して、自分のアカウントのリソースにアクセスすることを、別のアカウントの人物 (信頼済みプリンシパル) に許可できます。クロスアカウントアクセス権を付与する主な方法は、ロールを使用することです。ただし、一部の では AWS のサービス、 (プロキシとしてロールを使用する代わりに) リソースに直接ポリシーをアタッチできます。クロスアカウントアクセスにおけるロールとリソースベースのポリシーの違いについては、「IAM ユーザーガイド」の「IAM でのクロスアカウントのリソースへのアクセス」を参照してください。
- クロスサービスアクセス — 一部の は他の の機能 AWS のサービス を使用します AWS のサービス。例えば、あるサービスで呼び出しを行うと、通常そのサービスによって Amazon EC2 でアプリケーションが実行されたり、Amazon S3 にオブジェクトが保存されたりします。サービスでは、呼び出し元プリンシパルの許可、サービスロール、またはサービスリンクロールを使用してこれを行う場合があります。
  - 転送アクセスセッション (FAS) – IAM ユーザーまたはロールを使用してアクションを実行すると AWS、プリンシパルと見なされます。一部のサービスを使用する際に、アクションを実行することで、別のサービスの別のアクションがトリガーされることがあります。FAS は、 を呼び出すプリンシパルのアクセス許可を AWS のサービス、ダウンストリームサービス AWS のサービス へのリクエストをリクエストする と組み合わせて使用します。FAS リクエストは、サービスが他の AWS のサービス またはリソースとのやり取りを完了する必要があるリクエストを受け取った場合にのみ行われます。この場合、両方のアクションを実行するためのアクセス許可が必要です。FAS リクエストを行う際のポリシーの詳細については、「転送アクセスセッション」を参照してください。
  - サービスロール - サービスがユーザーに代わってアクションを実行するために引き受ける IAM ロールです。IAM 管理者は、IAM 内からサービスロールを作成、変更、削除することができます。詳細については、「IAM ユーザーガイド」の「AWS のサービスに許可を委任するロールを作成する」を参照してください。

- サービスにリンクされたロール – サービスにリンクされたロールは、 にリンクされたサービスロールの一種です AWS のサービス。サービスは、ユーザーに代わってアクションを実行するロールを引き受けることができます。サービスにリンクされたロールは に表示され AWS アカウント 、サービスによって所有されます。IAM 管理者は、サービスリンクロールのアクセス許可を表示できますが、編集することはできません。

- Amazon EC2 で実行されているアプリケーション – IAM ロールを使用して、EC2 インスタンスで実行され、 AWS CLI または AWS API リクエストを行うアプリケーションの一時的な認証情報を管理できます。これは、EC2 インスタンス内でのアクセスキーの保存に推奨されます。EC2 インスタンスに AWS ロールを割り当て、そのすべてのアプリケーションで使用できるようにするには、インスタンスにアタッチされたインスタンスプロファイルを作成します。インスタンスプロファイルにはロールが含まれ、EC2 インスタンスで実行されるプログラムは一時的な認証情報を取得できます。詳細については、「IAM ユーザーガイド」の「Amazon EC2 インスタンスで実行されるアプリケーションに IAM ロールを使用して許可を付与する」を参照してください。

リソースへのアクセス許可の管理の概要

> **Warning**
>
> 新しいプロジェクトでは、Kinesis Data Analytics for SQL アプリケーションよりも新しい Managed Service for Apache Flink Studio を使用することをお勧めします。Managed Service for Apache Flink Studio は、使いやすさと高度な分析機能を兼ね備えているため、高度なストリーム処理アプリケーションを数分で構築できます。

アクセス権限を付与するにはユーザー、グループ、またはロールにアクセス許可を追加します。

- 以下のユーザーとグループ AWS IAM Identity Center :

  アクセス許可セットを作成します。「AWS IAM Identity Center ユーザーガイド」の「権限設定を作成する」の手順に従ってください。

- IAM 内で、ID プロバイダーによって管理されているユーザー:

  ID フェデレーションのロールを作成します。詳細については「IAM ユーザーガイド」の「サードパーティー ID プロバイダー (フェデレーション) 用のロールを作成する」を参照してください。

- IAM ユーザー:

- ユーザーが担当できるロールを作成します。手順については「IAM ユーザーガイド」の「IAM ユーザーのロールの作成」を参照してください。
- (お奨めできない方法) ポリシーをユーザーに直接アタッチするか、ユーザーをユーザーグループに追加します。詳細については「IAM ユーザーガイド」の「ユーザー (コンソール) へのアクセス権限の追加」を参照してください。

> **Note**
>
> アカウント管理者 (または管理者ユーザー) は、管理者権限を持つユーザーです。詳細については、「IAM ユーザーガイド」の「IAM のベストプラクティス」を参照してください。

トピック

- リソースおよびオペレーション
- リソース所有権について
- リソースへのアクセスの管理
- ポリシー要素 (アクション、効果、プリンシパル) の指定
- ポリシーでの条件の指定

リソースおよびオペレーション

では、プライマリリソースはアプリケーションです。ポリシーで Amazon リソースネーム (ARN) を使用して、ポリシーを適用するリソースを識別します。

これらのリソースには、次の表に示すとおり、一意の Amazon リソースネーム (ARN) が関連付けられています。

リソースタイプ	ARN 形式
アプリケーション	`arn:aws:kinesisanalytics:` *`region`*`:`*`account-id`*` :application/ `*`application-name`*

では、リソースを操作する一連のオペレーションが用意されています。使用可能なオペレーションのリストについては、「アクション」を参照してください。

リソース所有権について

は、リソースを作成したユーザーに関係なく、アカウントで作成されたリソース AWS アカウント を所有します。具体的には、リソース所有者は、リソース作成リクエスト AWS アカウント を認証するプリンシパルエンティティ (ルートアカウント、ユーザー、または IAM ロール) の です。次の例は、この仕組みを示しています。

- のルートアカウントの認証情報を使用してアプリケーション AWS アカウント を作成する場合、AWS アカウント はリソースの所有者です。(では、リソースはアプリケーションです。)
- でユーザーを作成し AWS アカウント 、そのユーザーにアプリケーションを作成するアクセス許可を付与すると、そのユーザーはアプリケーションを作成できます。ただし、ユーザーが属 AWS アカウントする はアプリケーションリソースを所有します。ユーザーではなく、ロールに権限を付与するよう強くお勧めします。
- アプリケーションを作成するアクセス許可 AWS アカウント を持つ で IAM ロールを作成する場合、ロールを引き受けることのできるすべてのユーザーがアプリケーションを作成できます。ユーザーが属 AWS アカウントする は、アプリケーションリソースを所有します。

リソースへのアクセスの管理

アクセス権限ポリシー では、誰が何にアクセスできるかを記述します。次のセクションで、アクセス許可ポリシーを作成するために使用可能なオプションについて説明します。

> **Note**
>
> このセクションでは、 のコンテキストでの IAM の使用について説明します。IAM サービスに関する詳しい説明はしません。完全な IAM ドキュメンテーションについては、[IAM ユーザーガイド] の [IAM とは] を参照してください。IAM ポリシー構文の詳細と説明については、IAM ユーザーガイドの「IAM JSON ポリシーのリファレンス」を参照してください。

IAM アイデンティティにアタッチされているポリシーは、アイデンティティベースのポリシー (IAM ポリシー) と呼ばれます。リソースにアタッチされたポリシーを、リソースベースのポリシーと呼びます。アイデンティティベースのポリシー (IAM ポリシー) のみをサポートします。

トピック

- アイデンティティベースのポリシー (IAM ポリシー)
- リソースベースのポリシー

アイデンティティベースのポリシー (IAM ポリシー)

ポリシーを IAM アイデンティティにアタッチできます。例えば、次のオペレーションを実行できます。

- アカウントのユーザーまたはグループにアクセス権限ポリシーをアタッチする – アプリケーションなどのリソースを作成するアクセス権限を付与するには、ユーザーまたはユーザーが所属するグループにアクセス許可のポリシーをアタッチできます。
- アクセス権限ポリシーをロールにアタッチする (クロスアカウントの許可を付与) - ID ベースのアクセス権限ポリシーを IAM ロールにアタッチして、クロスアカウントの権限を付与することができます。たとえば、アカウント A の管理者は、次のように別の AWS アカウント（アカウント B など) または Amazon サービスにクロスアカウントアクセス許可を付与するロールを作成できます。
  1. アカウント A の管理者は、IAM ロールを作成して、アカウント A のリソースに許可を付与するロールに許可ポリシーをアタッチします。
  2. アカウント A の管理者は、アカウント B をそのロールを引き受けるプリンシパルとして識別するロールに、信頼ポリシーをアタッチします。
  3. アカウント B の管理者は、アカウント B のユーザーにロールを引き受ける権限を委任できるようになります。これにより、アカウント B のユーザーはアカウント A のリソースの作成とアクセスができます。ロールを引き受ける権限を Amazon のサービスに付与すると、信頼ポリシー内のプリンシパルも Amazon サービスのプリンシパルとなることができます。

  IAM を使用した許可の委任の詳細については、「IAM ユーザーガイド」の「アクセス管理」を参照してください。

以下に、アプリケーションの作成に必要な `kinesisanalytics:CreateApplication` アクションのアクセス権限を付与するポリシーの例を示します。

> **Note**
>
> これは簡単なポリシー例です。ポリシーをユーザーにアタッチすると、ユーザーは AWS CLI または AWS SDK を使用してアプリケーションを作成できます。しかし、入出力を設定するにはより多くのアクセス権限が必要です。また、このユーザーがコンソールを使用する場合もより多くのアクセス権限が必要です。後のセクションで、詳細な情報を説明します。

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Stmt1473028104000",
            "Effect": "Allow",
            "Action": [
                "kinesisanalytics:CreateApplication"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

でアイデンティティベースのポリシーを使用する場合の詳細については、「 でアイデンティティベースのポリシー (IAM ポリシー) を使用する 」を参照してください。ユーザー、グループ、ロール、許可の詳細については、「IAM ユーザーガイド」の「アイデンティティ (ユーザー、グループ、ロール)」を参照してください。

リソースベースのポリシー

Amazon S3 などの他のサービスでは、リソースベースの許可ポリシーもサポートされています。例えば、ポリシーを S3 バケットにアタッチして、そのバケットに対する許可を管理できます。 はリソースベースのポリシーをサポートしていません。

ポリシー要素 (アクション、効果、プリンシパル) の指定

サービスは、リソースごとに一連の API オペレーションを定義します。こうした API オペレーションへの許可を付与するために、 はポリシーに定義できる一連のアクションを定義します。一部の API オペレーションは、API オペレーションを実行するために複数のアクションに対するアクセス許可を要求できます。リソースおよび API オペレーションに関する詳細については、「 リソースおよびオペレーション」および「アクション」を参照してください。

最も基本的なポリシーの要素を次に示します。

- リソース - Amazon リソースネーム (ARN) を使用して、ポリシーを適用するリソースを識別します。詳細については、「 リソースおよびオペレーション」を参照してください。
- アクション – アクションキーワードを使用して、許可または拒否するリソース操作を特定します。たとえば、`create` を使用して、アプリケーションの作成をユーザーに許可することができます。

- 効果 – ユーザーが特定のアクションをリクエストする際の効果 (許可または拒否) を指定します。リソースへのアクセスを明示的に許可していない場合、アクセスは暗黙的に拒否されます。また、明示的にリソースへのアクセスを拒否すると、別のポリシーによってアクセスが許可されている場合でも、ユーザーはそのリソースにアクセスできなくなります。
- プリンシパル – ID ベースのポリシー (IAM ポリシー) で、ポリシーがアタッチされているユーザーが黙示的なプリンシパルとなります。リソースベースのポリシーでは、アクセス許可を受け取りたいユーザー、アカウント、サービス、またはその他のエンティティを指定します (リソースベースのポリシーにのみ適用)。 では、リソースベースのポリシーはサポートされていません。

IAM ポリシーの構文と記述の詳細については、IAM ユーザーガイドの「IAM JSON ポリシーのリファレンス」を参照してください。

適用する API オペレーションやリソースがすべて表示されているのリストについては、「 API アクセス許可: アクション、アクセス許可、リソースの参照」を参照してください。

ポリシーでの条件の指定

アクセス許可を付与するとき、アクセスポリシー言語を使用して、ポリシーが有効になる条件を指定できます。例えば、特定の日付の後にのみ適用されるポリシーが必要になる場合があります。ポリシー言語での条件の指定の詳細については、「IAM ユーザーガイド」の「条件」を参照してください。

条件を表すには、あらかじめ定義された条件キーを使用します。 に固有の条件キーはありません。ただし、必要に応じて使用できる AWS広範な条件キーがあります。 AWS全体のキーの完全なリストについては、IAM ユーザーガイドの「条件に使用可能なキー」を参照してください。

でアイデンティティベースのポリシー (IAM ポリシー) を使用する

以下のアイデンティティベースのポリシーの例では、アカウント管理者が IAM アイデンティティ (ユーザー、グループ、およびロール) にアクセス権限ポリシーをアタッチし、リソースに対するオペレーションを実行するアクセス権限を付与する方法を示します。

> **Important**
>
> 初めに、リソースへのアクセスを管理するための基本概念と、使用可能なオプションについて説明する概要トピックをお読みになることをお勧めします。詳細については、「 リソースへのアクセス許可の管理の概要」を参照してください。

トピック

- コンソールを使用するために必要なアクセス権限
- の Amazon 管理 (事前定義) ポリシー
- お客様が管理するポリシーの例

以下に示しているのは、アクセス権限ポリシーの例です。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Stmt1473028104000",
            "Effect": "Allow",
            "Action": [
                "kinesisanalytics:CreateApplication"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

このポリシーには以下の 1 つのステートメントがあります。

- 最初のステートメントでは、アプリケーションの Amazon リソースネーム (ARN) を使用して、リソースに対し 1 つのアクション (`kinesisanalytics:CreateApplication`) を行うアクセス権限を付与します。この場合の ARN はワイルドカード文字 (*) を指定して、どのリソースにもアクセス権限を付与することを示します。

すべての API オペレーションとそれらが適用されるリソースの表については、「API アクセス許可: アクション、アクセス許可、リソースの参照」を参照してください。

コンソールを使用するために必要なアクセス権限

ユーザーが コンソールで作業するには、必要なアクセス権限を付与する必要があります。たとえば、ユーザーにアプリケーションを作成するアクセス権限を付与する場合、ユーザーのアカウントでストリーミングソースを表示できるアクセス権限を付与し、ユーザーがコンソールで入出力を設定できるようにする必要があります。

次の構成を推奨します。

- Amazon 管理ポリシーを使用してユーザーにアクセス権限を付与します。使用できるポリシーについては、「 の Amazon 管理 (事前定義) ポリシー 」を参照してください。
- カスタムポリシーを作成します。このケースでは、このセクションで提供されている例を確認することをお勧めします。詳細については、「お客様が管理するポリシーの例」を参照してください。

の Amazon 管理 (事前定義) ポリシー

AWS は、 によって作成および管理されるスタンドアロン IAM ポリシーを提供することで、多くの一般的なユースケースに対処します AWS。これらの Amazon 管理ポリシーは、一般的なユースケースに必要な許可を付与することで、どの許可が必要なのかを調査する必要がなくなります。詳細については、「IAM ユーザーガイド」の「Amazon 管理ポリシー」を参照してください。

アカウントのユーザーにアタッチできる次の Amazon 管理ポリシーは、 に固有のものです。

- **`AmazonKinesisAnalyticsReadOnly`** – ユーザーがアプリケーションをリストし、入出力設定を確認できるアクションのアクセス権限を付与します。また、ユーザーが Kinesis ストリームと Firehose 配信ストリームのリストを表示するためのアクセス権限も付与します。アプリケーションが実行されているときは、ユーザーはコンソールでソースデータとリアルタイム分析の結果を表示できます。

- **`AmazonKinesisAnalyticsFullAccess`** – すべてのアクションのアクセス権限およびユーザーがアプリケーションを作成、管理できる他のすべてのアクセス権限を付与します。ただし、以下の点に注意してください。

  - これらのアクセス権限は、ユーザーがコンソールで新しい IAM ロールを作成するには不十分です (これらのアクセス権限では、ユーザーが既存ロールを選択することはできます)。ユーザーがコンソールで IAM ロールを作成できるようにする場合、`IAMFullAccess` Amazon 管理ポリシーを追加します。

  - アプリケーションを設定する際に IAM ロールを指定する場合は、`iam:PassRole` アクションのアクセス権限が必要です。この Amazon 管理ポリシーは、プレフィックス `service-role/`

`kinesis-analytics` から始まる IAM ロールについてのみ、ユーザーに `iam:PassRole` アクションのアクセス権限を付与します。

ユーザーが、このプレフィックスがついていないロールを使用して `iam:PassRole` アプリケーションを設定する場合、その特定のロールで明示的にユーザーにアクションのアクセス権限を付与する必要があります。

独自のカスタム IAM ポリシーを作成して、 アクションとリソースのための権限を許可することもできます。こうしたカスタムポリシーは、該当するアクセス許可が必要なユーザーまたはグループにアタッチできます。

お客様が管理するポリシーの例

このセクションの例では、ユーザーにアタッチできるサンプルポリシーのグループが用意されています。ポリシーの作成が初めての場合は、お客様のアカウントにユーザーを作成することをお勧めします。次に、このセクションのステップで説明している順番でポリシーをそのユーザーにアタッチします。その後、コンソールを使用して、ユーザーにポリシーをアタッチしながら各ポリシーの効果を確認できます。

最初は、ユーザーにアクセス権限が付与されていないため、コンソールを使用してできることは何もありません。ユーザーにポリシーをアタッチすることで、ユーザーがコンソールで多様なアクションを実行できることを確認できます。

2 つのブラウザウィンドウを使用することをお勧めします。1 つのウィンドウでユーザーを作成し、アクセス権限を付与します。もう 1 つは、ユーザーの認証情報 AWS Management Console を使用して にサインインし、アクセス許可を付与するときにアクセス許可を検証します。

アプリケーションの実行ロールとして使用する IAM ロールの作成例については、「IAM ユーザーガイド」の「IAM ロールの作成」を参照してください。

ステップ例

- ステップ 1: IAM ユーザーを作成する
- ステップ 2:に固有ではないアクションのユーザーアクセス権限を許可する
- ステップ 3: ユーザーにアプリケーション一覧の表示と詳細の表示を許可する
- ステップ 4: ユーザーに特定のアプリケーションの起動を許可する
- ステップ 5: ユーザーにアプリケーションの作成を許可する
- ステップ 6: アプリケーションが Lambda 事前処理を使用できるようにする

ステップ 1: IAM ユーザーを作成する

まず、ユーザーを作成し、管理者許可を持つ IAM グループにユーザーを追加したら、作成したユーザーに管理者許可を付与する必要があります。その後、特別な URL とそのユーザーの認証情報 AWS を使用して にアクセスできます。

手順については、「IAM ユーザーガイド」の「最初の IAM ユーザーと管理者グループの作成」を参照してください。

ステップ 2:に固有ではないアクションのユーザーアクセス権限を許可する

まず、 に固有ではないすべてのアクションのアクセス権限をユーザーに付与します。これはユーザーがアプリケーションを使用するときに必要となります。これには、ストリームを使用するためのアクセス権限 (Amazon Kinesis Data Streams アクション、Amazon Data Firehose アクション) や、CloudWatch アクションのアクセス権限が含まれます。次のポリシーをユーザーにアタッチします。

`iam:PassRole` アクセス権限を付与する IAM ロール名を入力するか、すべての IAM ロールを示すワイルドカード文字 (*) を指定して、ポリシーを更新する必要があります。これは安全なプラクティスではありませんが、このテスト中に作成された特定の IAM ロールがない場合があります。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kinesis:CreateStream",
                "kinesis:DeleteStream",
                "kinesis:DescribeStream",
                "kinesis:ListStreams",
                "kinesis:PutRecord",
                "kinesis:PutRecords"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "firehose:DescribeDeliveryStream",
                "firehose:ListDeliveryStreams"
            ],
```

```
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:GetMetricStatistics",
                "cloudwatch:ListMetrics"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "logs:GetLogEvents",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:ListPolicyVersions",
                "iam:ListRoles"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "arn:aws:iam::*:role/service-role/role-name"
        }
    ]
}
```

ステップ 3: ユーザーにアプリケーション一覧の表示と詳細の表示を許可する

次のポリシーでは、ユーザーに以下のアクセス権限を付与します。

- ユーザーがアプリケーションの一覧を表示できる `kinesisanalytics:ListApplications` アクションのアクセス権限。これはサービスレベルの API コールで、`Resource` 値として「`*`」を指定することに注意してください。
- 任意のアプリケーションの情報を取得できる `kinesisanalytics:DescribeApplication` アクションのアクセス権限。

このポリシーをユーザーに追加します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kinesisanalytics:ListApplications"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "kinesisanalytics:DescribeApplication"
            ],
            "Resource": "arn:aws:kinesisanalytics:aws-region:aws-account-
id:application/*"
        }
    ]
}
```

ユーザー認証情報を使用してコンソールにサインインし、これらのアクセス権限を検証します。

ステップ 4: ユーザーに特定のアプリケーションの起動を許可する

ユーザーに既存の アプリケーションのどれかを実行できるようにする場合、次のポリシーをユーザーにアタッチします。このポリシーでは、`kinesisanalytics:StartApplication` アクションを実行するためのアクセス権限を付与します。アカウント ID、 AWS リージョン、アプリケーション名を指定してポリシーを更新する必要があります。

```
{
    "Version": "2012-10-17",
    "Statement": [
                {
            "Effect": "Allow",
            "Action": [
                "kinesisanalytics:StartApplication"
            ],
            "Resource": "arn:aws:kinesisanalytics:aws-region:aws-account-
id:application/application-name"
        }
    ]
```

```
}
```

ステップ 5: ユーザーにアプリケーションの作成を許可する

ユーザーがアプリケーションを作成できるようにする場合、次のポリシーをユーザーにアタッチできます。ポリシーを更新し、 AWS リージョン、アカウント ID、およびユーザーが作成する特定のアプリケーション名、またはユーザーが任意のアプリケーション名を指定 (したがって複数のアプリケーションを作成) できるように「*」を指定する必要があります。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Stmt1473028104000",
            "Effect": "Allow",
            "Action": [
                "kinesisanalytics:CreateApplication"
            ],
            "Resource": [
                "*"
            ]
        },

        {
            "Effect": "Allow",
            "Action": [
                "kinesisanalytics:StartApplication",
                "kinesisanalytics:UpdateApplication",
                "kinesisanalytics:AddApplicationInput",
                "kinesisanalytics:AddApplicationOutput"
            ],
            "Resource": "arn:aws:kinesisanalytics:aws-region:aws-account-
id:application/application-name"
        }
    ]
}
```

ステップ 6: アプリケーションが Lambda 事前処理を使用できるようにする

アプリケーションで Lambda の事前処理を使用できるようにする場合、次のポリシーをロールアにアタッチします。

```
    {
      "Sid": "UseLambdaFunction",
      "Effect": "Allow",
      "Action": [
          "lambda:InvokeFunction",
          "lambda:GetFunctionConfiguration"
      ],
      "Resource": "<FunctionARN>"
  }
```

API アクセス許可: アクション、アクセス許可、リソースの参照

アクセスコントロール をセットアップし、IAM アイデンティティにアタッチできるアクセス権限ポリシー (アイデンティティベースのポリシー) を作成するときは、以下のリストをリファレンスとして使用できます。には、各 API オペレーション、アクションを実行するためのアクセス許可を付与できる対応するアクション、およびアクセス許可を付与できる AWS リソースが含まれます。ポリシーの `Action` フィールドでアクションを指定し、ポリシーの `Resource` フィールドでリソースの値を指定します。

ポリシーで AWS全体の条件キーを使用して、条件を表現できます。 AWS全体のキーの完全なリストについては、IAM ユーザーガイドの「使用可能なキー」を参照してください。

> **Note**
>
> アクションを指定するには、API オペレーション名 (`kinesisanalytics:AddApplicationInput` など) の前に `kinesisanalytics` プレフィックスを使用します。

API とアクションで必要なアクセス権限

API オペレーション:

必要なアクセス許可 (API アクション):

リソース:

API とアクションで必要なアクセス権限

Amazon RDS API とアクションで必要なアクセス許可

API オペレーション:AddApplicationInput

アクション:`kinesisanalytics:AddApplicationInput`

リソース:

`arn:aws:kinesisanalytics: region:accountId:application/application-name`

GetApplicationState

コンソールは、`GetApplicationState` と呼ばれる内部メソッドを使用して、アプリケーションデータをサンプリングするか、アプリケーションデータにアクセスします。サービスアプリケーションには、 AWS Management Consoleを介してアプリケーションデータをサンプリングするか、アプリケーションデータにアクセスするために、内部 `kinesisanalytics:GetApplicationState` API に対するアクセス許可が必要です。

Amazon Kinesis Data Analytics のモニタリング

Kinesis Data Analytics は、アプリケーションのモニタリング機能を備えています。詳細については、「モニタリング」を参照してください。

Amazon Kinesis Data Analytics for SQL Applications のコンプライアンス検証

サードパーティーの監査者は、複数のコンプライアンスプログラムの一環として Amazon Kinesis Data Analytics のセキュリティと AWS コンプライアンスを評価します。このプログラムには、SOC、PCI、HIPAA などを含みます。

特定のコンプライアンスプログラムの対象となる AWS サービスのリストについては、「コンプライアンスプログラムによる Amazon Services in Scope」を参照してください。一般的な情報については、「AWS コンプライアンスプログラム」を参照してください。

を使用して、サードパーティーの監査レポートをダウンロードできます AWS Artifact。詳細については、「 でレポートをダウンロードする AWS Artifact」を参照してください。

Kinesis Data Analytics を使用する際のお客様のコンプライアンス責任は、お客様のデータの機密性や貴社のコンプライアンス目的、適用可能な法律および規制によって決定されます。Kinesis Data Analytics の使用が、HIPAA または PCI などの規格との適合を条件とする場合、 AWS では、次の支援リソースを提供しています。

- セキュリティとコンプライアンスのクイックスタートガイド – これらのデプロイガイドでは、アーキテクチャ上の考慮事項について説明し、セキュリティとコンプライアンスに重点を置いたベースライン環境をデプロイする手順について説明します AWS。
- 「Architecting for HIPAA Security and Compliance」ホワイトペーパー – このホワイトペーパーでは、企業が AWS を使用して HIPAA 準拠のアプリケーションを作成する方法について説明します。
- AWS コンプライアンスリソース – このワークブックとガイドのコレクションは、お客様の業界や地域に適用される場合があります。
- AWS Config – この AWS サービスは、リソース設定が内部プラクティス、業界ガイドライン、および規制にどの程度準拠しているかを評価します。
- AWS Security Hub – この AWS サービスは、 内のセキュリティ状態を包括的に把握 AWS し、セキュリティ業界標準とベストプラクティスへの準拠を確認するのに役立ちます。

Amazon Kinesis Data Analytics の耐障害性

AWS グローバルインフラストラクチャは、 AWS リージョンとアベイラビリティーゾーンを中心に構築されています。 AWS リージョンは、低レイテンシー、高スループット、高度に冗長なネットワークで接続された、物理的に分離された複数のアベイラビリティーゾーンを提供します。アベイラビリティーゾーンでは、アベイラビリティーゾーン間で中断せずに、自動的にフェイルオーバーするアプリケーションとデータベースを設計および運用することができます。アベイラビリティーゾーンは、従来の単一または複数のデータセンターインフラストラクチャよりも可用性、耐障害性、およびスケーラビリティが優れています。

AWS リージョンとアベイラビリティーゾーンの詳細については、AWS 「 グローバルインフラストラクチャ」を参照してください。

グローバル AWS インフラストラクチャに加えて、Kinesis Data Analytics には、データの耐障害性とバックアップのニーズをサポートするのに役立つ機能がいくつか用意されています。

災害対策

Kinesis Data Analytics はサーバーレスモードで実行され、ホストのパフォーマンス低下、アベイラビリティーゾーンの可用性、および自動移行に伴うインフラストラクチャ関連のその他の問題に対応します。この場合、Kinesis Data Analytics により、すべてのアプリケーションがデータ損失なしで処理されます。詳細については、「アプリケーション出力を外部宛先で永続化する配信モデル」を参照してください。

Kinesis Data Analytics for SQL アプリケーションのインフラストラクチャセキュリティ

マネージドサービスである Amazon Kinesis Data Analytics は、ホワイトペーパー「Amazon Web Services: セキュリティプロセスの概要」に記載されている AWS グローバルネットワークセキュリティ手順で保護されています。

AWS 公開された API コールを使用して、ネットワーク経由で Kinesis Data Analytics にアクセスします。クライアントは、Transport Layer Security (TLS) 1.2 以降をサポートする必要があります。クライアントは、Ephemeral Diffie-Hellman (DHE) や Elliptic Curve Ephemeral Diffie-Hellman (ECDHE) などの Perfect Forward Secrecy (PFS) を使用する暗号スイートもサポートする必要があります。これらのモードは、Java 7 以降など、最近のほとんどのシステムでサポートされています。

また、リクエストにはアクセスキー ID と、IAM プリンシパルに関連付けられているシークレットアクセスキーを使用して署名する必要があります。またはAWS Security Token Service (AWS STS) を使用して、一時的なセキュリティ認証情報を生成し、リクエストに署名することもできます。

Kinesis Data Analytics のセキュリティのベストプラクティス

Amazon Kinesis Data Analytics には、独自のセキュリティポリシーを策定および実装する際に考慮すべきさまざまなセキュリティ機能が用意されています。以下のベストプラクティスは一般的なガイドラインであり、完全なセキュリティソリューションを説明するものではありません。これらのベストプラクティスはお客様の環境に適切ではないか、十分ではない場合があるため、これらは指示ではなく、有用な考慮事項と見なしてください。

IAM ロールを使用して他の Amazon サービスにアクセスする

他のサービスのリソース (Kinesis データストリーム、Firehose 配信ストリーム、Amazon S3 バケットなど) にアクセスするには、Kinesis Data Analytics アプリケーションに有効な認証情報が必要で

す。 AWS 認証情報をアプリケーションや Amazon S3 バケットに直接保存しないでください。これらは自動的にローテーションされない長期的な認証情報であり、漏洩するとビジネスに大きな影響が及ぶ場合があります。

CloudWatch Logs ロググループの作成代わりに、 IAM ロールを使用して、他のリソースにアクセスするためのアプリケーションの一時的な認証情報を管理してください。ロールを使用する場合、長期的な認証情報 (ユーザー名やパスワード、アクセスキーなど) を使用して他のリソースにアクセスする必要はありません。

詳細については、IAM ユーザーガイド にある下記のトピックを参照してください。

- IAM ロール
- ロールの一般的なシナリオ: ユーザー、アプリケーション、およびサービス

依存リソースでのサーバー側の暗号化の実装

保管中のデータと転送中のデータは Kinesis Data Analytics で暗号化されます。この暗号化を無効にすることはできません。Kinesis データストリーム、Firehose 配信ストリーム、Amazon S3 バケットなどの依存リソースには、サーバー側の暗号化を実装する必要があります。依存リソースでのサーバー側の暗号化の実装の詳細については、「データ保護」を参照してください。

CloudTrail を使用して API コールをモニタリングする

Kinesis Data Analytics は AWS CloudTrail、Kinesis Data Analytics のユーザー、ロール、または Amazon サービスによって実行されたアクションを記録するサービスである と統合されています。

CloudTrail によって収集された情報を使用して、Kinesis Data Analytics に対して実行されたリクエスト、リクエスト実行元の IP アドレス、リクエストの実行者、リクエストの実行時、およびその他の詳細を判断することができます。

詳細については、「the section called “の使用 AWS CloudTrail”」を参照してください。

for SQL Applications のモニタリング

モニタリングは、 と アプリケーションの信頼性、可用性、パフォーマンスを維持する上で重要な部分です。マルチポイント障害が発生した場合は、その障害をより簡単にデバッグできるように、AWS ソリューションのすべての部分からモニタリングデータを収集する必要があります。ただし、のモニタリングを開始する前に、以下の質問に対する回答を反映したモニタリング計画を作成する必要があります。

- モニタリングの目的は何ですか?
- どのリソースをモニタリングしますか?
- どのくらいの頻度でこれらのリソースをモニタリングしますか?
- どのモニタリングツールを利用しますか?
- 誰がモニタリングタスクを実行しますか?
- 問題が発生したときに誰が通知を受け取りますか?

次のステップでは、さまざまなタイミングと負荷条件でパフォーマンスを測定することにより、お客様の環境で通常のパフォーマンスのベースラインを確定します。 をモニタリングする際、過去のモニタリングデータを保存することができます。保存すれば、パフォーマンスデータをこの過去のデータと比較して、通常のパフォーマンスパターンとパフォーマンス異常を識別することで、問題の対処方法を考案しやすくなります。

でアプリケーションをモニタリングします。アプリケーションはデータストリーム (入力および出力) を処理します。どちらにも識別子が含まれており、CloudWatch ログで検索の絞り込みに使用できます。データストリームの処理方法については、「Amazon Kinesis Data Analytics for SQL Applications: 仕組み」を参照してください。

最も重要なメトリクスは `millisBehindLatest` です。これはアプリケーションがどのくらい遅れてストリーミングソースから読み取るかを示します。一般的なケースでは、ミリ秒の遅延またはほぼゼロであるべきです。短期間のスパイクが発生することはよくあります。これは `millisBehindLatest` での増加として表示されます。

アプリケーションでストリーミングソースからの読み取りが 1 時間以上遅延する場合にトリガーされる CloudWatch アラームをセットアップすることをお勧めします。処理されたデータをライブアプリケーションに発行するなど、ほぼリアルタイムの処理を必要とする一部のユースケースでは、5 分などより低い値でアラームを設定する場合もあります。

トピック

- モニタリングツール
- Amazon CloudWatch によるモニターリング
- での AWS CloudTrail API コールのログ記録

モニタリングツール

AWS には、 のモニタリングに使用できるさまざまなツールが用意されています。これらのツールの一部はモニタリングを行うように設定できますが、一部のツールは手動による介入が必要です。モニタリングタスクをできるだけ自動化することをお勧めします。

自動モニタリングツール

以下の自動化されたモニタリングツールを使用して、 を監視し、問題が発生したときにレポートできます。

- Amazon CloudWatch アラーム - 指定した期間にわたって単一のメトリクスをモニタリングし、複数の期間にわたる特定のしきい値に対するメトリクスの値に基づいて 1 つ以上のアクションを実行します。アクションは、Amazon Simple Notification Service (Amazon SNS) のトピックまたは Amazon EC2 Auto Scaling のポリシーに送信される通知です。CloudWatch アラームは、特定の状態にあるという理由だけでアクションを呼び出すことはありません。状態が変更され、指定された期間維持されている必要があります。詳細については、「Amazon CloudWatch によるモニターリング」を参照してください。
- Amazon CloudWatch Logs – AWS CloudTrail またはその他のソースのログファイルのモニタリング、保存、アクセスを行います。詳細については、「Amazon CloudWatch ユーザーガイド」の「ログファイルのモニタリング」を参照してください。
- Amazon CloudWatch Events - イベントに一致したものを 1 つ以上のターゲットの関数またはストリームに渡して、変更、状態の情報の収集、是正措置を行います。詳細については、「Amazon CloudWatch ユーザーガイド」の「Amazon CloudWatch Events とは」を参照してください。
- AWS CloudTrail ログモニタリング – アカウント間でログファイルを共有し、CloudWatch Logs に送信CloudWatch CloudTrail ログファイルをリアルタイムでモニタリングし、Java でログ処理アプリケーションを書き込み、CloudTrail による配信後にログファイルが変更されていないことを確認します。詳細については、「AWS CloudTrail ユーザーガイド」の「CloudTrail ログファイルの使用」を参照してください。

手動モニタリングツール

のモニタリングでもう 1 つ重要なパートは、CloudWatch のアラームの対象外の項目を手動でモニタリングすることです。、CloudWatch Trusted Advisor、およびその他の AWS Management Console ダッシュボードには、 AWS 環境の状態がat-a-glanceビューが表示されます。

- CloudWatch のホームページでは以下について確認できます。
  - 現在のアラームとステータス
  - アラームとリソースのグラフ
  - サービスのヘルスステータス

  また、CloudWatch を使用して以下のことを行えます。
  - 重視するサービスをモニタリングするためのカスタマイズしたダッシュボードを作成します
  - メトリクスデータをグラフ化して、問題のトラブルシューティングを行い、傾向を確認する
  - すべてのメトリクスを検索して、参照する
  - 問題があることを通知するアラームを作成/編集する
- AWS Trusted Advisor は、 をモニタリングして、パフォーマンス、信頼性、セキュリティ、コスト効率を向上させるのに役立ちます。すべてのユーザーは、4 つの Trusted Advisor チェックを利用できます。ビジネスまたはエンタープライズサポートプランのユーザーは、50 以上のチェックを利用できます。詳細については、「AWS Trusted Advisor」を参照してください。

Amazon CloudWatch によるモニターリング

Amazon CloudWatch を使用して、アプリケーションをモニタリングできます。CloudWatch は、 から raw データを収集して、読み取り可能なほぼリアルタイムのメトリクスに変換します。これらの統計は 2 週間保持されます。履歴情報にアクセスして、ウェブアプリケーションやサービスの動作をより的確に把握できます。デフォルトでは、 メトリクスデータは CloudWatch に自動的に送信されます。詳細については、Amazon CloudWatch ユーザーガイドの「Amazon CloudWatch とは」を参照してください。

トピック

- メトリクスとディメンション
- のメトリクスおよびディメンションの表示
- をモニタリングする CloudWatch アラームの作成
- Amazon CloudWatch Logs の使用

メトリクスとディメンション

AWS/KinesisAnalytics 名前空間には、次のメトリクスが含まれます。

メトリクス	説明
Bytes	読み取りバイト数 (入力ストリームあたり) または書き込みバイト数 (出力ストリームあたり)。 レベル: 入力ストリームあたりおよび出力ストリームあたり
KPUs	ストリーム処理アプリケーションを実行するために使用される Kinesis 処理ユニットの数。毎時間使用される KPU の平均数によって、アプリケーションへの課金が決まります。 レベル: アプリケーションレベル
MillisBehindLatest	アプリケーションが現在時刻からどのくらい遅れてストリーミングソースから読み取るかを示します。 レベル: アプリケーションレベル
Records	読み取りレコード数 (入力ストリームあたり) または書き込みレコード数 (出力ストリームあたり)。 レベル: 入力ストリームあたりおよび出力ストリームあたり
Success	アプリケーションに設定された宛先への配信が成功するたびに 1、失敗した配信試行ごとに0。このメトリクスの平均値は、成功した配信の回数を示します。 レベル: 宛先ごと。
InputProcessing.Duration	によって実行される各 AWS Lambda 関数呼び出しにかかる時間。

メトリクス	説明
	レベル: 入力ストリームあたり
InputProcessing.OkRecords	Ok ステータスとマークされた Lambda 関数によって返されたレコードの数。 レベル: 入力ストリームあたり
InputProcessing.OkBytes	Ok ステータスとマークされた Lambda 関数によって返されたレコードのバイトの合計数。 レベル: 入力ストリームあたり
InputProcessing.DroppedRecords	Dropped ステータスとマークされた Lambda 関数によって返されたレコードの数。 レベル: 入力ストリームあたり
InputProcessing.ProcessingFailedRecords	ProcessingFailed ステータスとマークされた Lambda 関数によって返されたレコードの数。 レベル: 入力ストリームあたり
InputProcessing.Success	による正常な Lambda の呼び出しの数。 レベル: 入力ストリームあたり
LambdaDelivery.OkRecords	Ok ステータスとマークされた Lambda 関数によって返されたレコードの数。 レベル: Lambda 宛先ごと
LambdaDelivery.DeliveryFailedRecords	DeliveryFailed ステータスとマークされた Lambda 関数によって返されたレコードの数。 レベル: Lambda 宛先ごと

メトリクス	説明
`LambdaDelivery.Duration`	によって実行される各 Lambda 関数呼び出しにかかる時間。 レベル: Lambda 宛先ごと

は以下のディメンションのメトリクスを提供します。

ディメンション	説明
`Flow`	入力ストリームあたり: 入力 出力ストリームあたり: 出力
`Id`	入力ストリームあたり: 入力 ID 出力ストリームあたり: 出力 ID

のメトリクスおよびディメンションの表示

アプリケーションでデータストリームを処理するときに、 は次のメトリクスとディメンションを、CloudWatch に送信します。 のメトリクスを表示するには、以下の手順を使用できます

コンソールで、メトリクスはまずサービスの名前空間ごとにグループ化され、次に各名前空間内のディメンションの組み合わせごとにグループ化されます。

CloudWatch コンソールを使用してメトリクスを表示するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで メトリクスを選択します。
3. の [CloudWatch Metrics by Category] ペインで、メトリクスカテゴリを選択します。
4. 上部のペインで、スクロールするとメトリクスの詳細なリストが表示されます。

を使用してメトリクスを表示するには AWS CLI

- コマンドプロンプトで、次のコマンドを使用します。

```
aws cloudwatch list-metrics --namespace "AWS/KinesisAnalytics" --region region
```

メトリクスは次のレベルで収集されます。

- アプリケーション
- 入力ストリーム
- 出力ストリーム

をモニタリングする CloudWatch アラームの作成

アラームの状態が変わったときに Amazon SNS メッセージを送信する Amazon CloudWatch のアラームを作成することができます。1 つのアラームで、指定した期間中、1 つのメトリクスを監視します。このアラームは、複数の期間にわたる一定のしきい値とメトリクスの値の関係性に基づき、1 つ以上のアクションを実行します。アクションは、Amazon SNS のトピックまたは自動スケーリングのポリシーに送信される通知です。

アラームは、持続した状態の変化に対してのみアクションを呼び出します。CloudWatch アラームでアクションを呼び出すには、状態を変更し、指定期間にわたって維持する必要があります。

アラームは、以下で説明するように AWS Management Console、、CloudWatch AWS CLI、または CloudWatch API を使用して設定できます。

CloudWatch コンソールを使用してアラームを設定するには

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/cloudwatch/ で CloudWatch コンソールを開きます。
2. アラームの作成(アラームの作成) を選択します。[アラームの作成ウィザード] が開始されます。
3. [Kinesis Analytics Metrics] を選択します。その後、 メトリクスをスクロールして、アラームを設定するメトリクスを見つけます。

   メトリクスのみ表示するには、ファイルシステムのシステム ID を検索します。アラームを作成するメトリクスを選択し、[次へ] をクリックします。
4. [名前]、[説明]、[次の時] のそれぞれにメトリクスの値を入力します。
5. アラーム状態に達したときに CloudWatch からメールを送信する場合は、[このアラームが次の時:]フィールドで[状態はアラーム]を選択します。[通知の送信先:] フィールドで、既存の SNS ト

ピックを選択します。[トピックの作成] を選択すると、新しいメールサブスクリプションリストの名前とメールアドレスを設定できます。このリストは保存され、今後のアラーム用のフィールドに表示されます。

> **Note**
>
> [トピックの作成] を使用して新しい Amazon SNS トピックを作成する場合、メールアドレスを検証しなければ、そのアドレスで通知を受け取ることができません。メールは、アラームがアラーム状態になったときにのみ送信されます。アラーム状態になった際に、メールアドレスの検証がまだ完了していない場合は、そのアドレスで通知を受け取ることはできません。

6. [アラームのプレビュー] セクションで、作成するアラームをプレビューします。
7. [アラームの作成] を選択して、アラームを作成します。

CloudWatch CLI を使用してアラームを設定するには

- `mon-put-metric-alarm` を呼び出します。詳細については、「Amazon CloudWatch CLI リファレンス」を参照してください。

CloudWatch API を使用してアラームを設定するには

- `PutMetricAlarm` を呼び出します。詳細については、「Amazon CloudWatch API リファレンス」を参照してください。

Amazon CloudWatch Logs の使用

アプリケーションが誤設定されていると、アプリケーションは起動時に実行状態に移行することがあります。または、データを更新できても、アプリケーション内の入力ストリームのデータは処理されません。アプリケーション設定の問題をモニタリングするには、CloudWatch ログオプションをアプリケーションに追加します。

では、以下の条件で構成エラーを生成できます。

- 入力に使用されている Kinesis データストリームは存在しません。
- 入力に使用されている Amazon Data Firehose 配信ストリームが存在しない。
- リファレンスデータソースとして使用されている Amazon S3 バケットは存在しません。

- S3 バケットのリファレンスデータソースに指定されたファイルは存在しません。
- 関連するアクセス許可を管理する AWS Identity and Access Management (IAM) ロールに正しいリソースが定義されていません。
- 関連アクセス権限を管理する IAM ロールで、適切な権限が定義されていません。
- には、関連アクセス権限を管理する IAM ロールを引き受けるアクセス権限がありません。

Amazon CloudWatch の詳細については、Amazon CloudWatch ユーザーガイドを参照してください。

PutLogEvents ポリシーアクションの追加

では、誤設定エラーを CloudWatch に書き込むための権限が必要です。以下に説明しているように、で引き受ける IAM ロールにこれらのアクセス権限を追加することができます。 で IAM ロールを使用する方法については、「Kinesis Data Analytics の Identity and Access Management」を参照してください。

信頼ポリシー

IAM ロールを引き受けるためのアクセス権限を に付与するには、以下の信頼ポリシーをそのロールにアタッチします。

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

アクセス許可ポリシー

リソースから CloudWatch にログイベントを書き込むアクセス権限をアプリケーションに付与するには、以下の IAM 権限ポリシーを使用します。

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Stmt0123456789000",
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:us-east-1:123456789012:log-group:my-log-group:log-
stream:my-log-stream*"
            ]
        }
    ]
}
```

モニタリング中の設定エラーの追加

CloudWatch ログオプションを新しいアプリケーションまたは既存アプリケーションに追加するか、既存アプリケーションのログオプションを変更するには、以下の API アクションを使用します。

> **Note**
>
> 現在は、API アクションを使用して、アプリケーションへの CloudWatch ログオプションの追加のみ行うことができます。コンソールを使用して CloudWatch ログオプションを追加することはできません。

アプリケーション作成時の CloudWatch ログオプションの追加

以下のコード例では、`CreateApplication` アクションを使用して、アプリケーション作成時に CloudWatch ログオプションを使用する方法について説明します。`Create_Application` の詳細については、「CreateApplication」をご参照ください。

```
{
   "ApplicationCode": "<The SQL code the new application will run on the input
 stream>",
   "ApplicationDescription": "<A friendly description for the new application>",
   "ApplicationName": "<The name for the new application>",
   "Inputs": [ ... ],
   "Outputs": [ ... ],
```

```
   "CloudWatchLoggingOptions": [{
      "LogStreamARN": "<Amazon Resource Name (ARN) of the CloudWatch log stream to add
 to the new application>",
      "RoleARN": "<ARN of the role to use to access the log>"
 }]
}
```

既存のアプリケーションへの CloudWatch ログオプションの追加

以下のコード例では、AddApplicationCloudWatchLoggingOption アクションを使用して、既存アプリケーションに CloudWatch ログオプションを追加する方法について説明します。AddApplicationCloudWatchLoggingOption の詳細については、「AddApplicationCloudWatchLoggingOption」を参照してください。

```
{
   "ApplicationName": "<Name of the application to add the log option to>",
   "CloudWatchLoggingOption": {
      "LogStreamARN": "<ARN of the log stream to add to the application>",
      "RoleARN": "<ARN of the role to use to access the log>"
   },
   "CurrentApplicationVersionId": <Version of the application to add the log to>
}
```

既存の CloudWatch ログオプションの更新

以下のコード例では、UpdateApplication アクションを使用して、既存の CloudWatch ログオプションを変更する方法について説明します。UpdateApplication の詳細については、「UpdateApplication」を参照してください。

```
{
   "ApplicationName": "<Name of the application to update the log option for>",
   "ApplicationUpdate": {
      "CloudWatchLoggingOptionUpdates": [
         {
            "CloudWatchLoggingOptionId": "<ID of the logging option to modify>",
            "LogStreamARNUpdate": "<ARN of the new log stream to use>",
            "RoleARNUpdate": "<ARN of the new role to use to access the log stream>"
         }
      ],
   },
   "CurrentApplicationVersionId": <ID of the application version to modify>
```

```
}
```

アプリケーションからの CloudWatch ログオプションの削除

以下のコード例では、`DeleteApplicationCloudWatchLoggingOption` アクションを使用して、既存の CloudWatch ログオプションを削除する方法について説明します。`DeleteApplicationCloudWatchLoggingOption` の詳細については、「DeleteApplicationCloudWatchLoggingOption」を参照してください。

```
{
   "ApplicationName": "<Name of application to delete log option from>",
   "CloudWatchLoggingOptionId": "<ID of the application log option to delete>",
   "CurrentApplicationVersionId": <Version of the application to delete the log option
 from>
}
```

設定エラー

以下のセクションには、誤設定したアプリケーションの Amazon CloudWatch Logs で表示される可能性のあるエラーの詳細が含まれています。

エラーメッセージ形式

アプリケーションの誤設定によって生成されるエラーメッセージは、次の形式で表示されます。

```
{
"applicationARN": "string",
"applicationVersionId": integer,
"messageType": "ERROR",
"message": "string",
"inputId": "string",
"referenceId": "string",
"errorCode": "string"
"messageSchemaVersion": "integer",
}
```

エラーメッセージのフィールドには、次の情報が含まれています。

- applicationARN: 生成アプリケーションの Amazon リソースネーム (ARN) (例: arn:aws:kinesisanalytics:us-east-1:112233445566:application/sampleApp)。
- applicationVersionId: エラー発生時のアプリケーションのバージョン。詳細については、「ApplicationDetail」を参照してください。
- messageType: メッセージの種類。現在、この種類は ERROR のみです。
- message: エラーの詳細など。

```
There is a problem related to the configuration of your input. Please check that the
 resource exists, the role has the correct permissions to access the resource and
 that Kinesis Analytics can assume the role provided.
```

- inputId: アプリケーション入力に関連付けられた ID。この値は、入力がエラーの原因である場合にのみ表示されます。この値は、referenceId が存在する場合は存在しません。詳細については、「DescribeApplication」を参照してください。
- referenceId: アプリケーションのリファレンスデータソースに関連付けられた ID。この値は、ソースがエラーの原因である場合にのみ表示されます。この値は、inputId が存在する場合は存在しません。詳細については、「DescribeApplication」を参照してください。
- errorCode: エラーの識別子。この ID は、InputError または ReferenceDataError になります。
- messageSchemaVersion: 現在のメッセージスキーマバージョンを指定する値。現在は 1 です。エラーメッセージスキーマが更新されているかどうかを確認するには、この値を確認します。

エラー

の CloudWatch ログに表示される可能性のあるエラーには、次のようなものがあります。

リソースが存在しません

ARN が存在しない Kinesis 入力ストリームに指定されていても、その ARN が構文的に正しい場合は、以下のようなエラーが生成されます。

```
{
"applicationARN": "arn:aws:kinesisanalytics:us-east-1:112233445566:application/
sampleApp",
"applicationVersionId": "5",
 "messageType": "ERROR",
 "message": "There is a problem related to the configuration of your input. Please
 check that the resource exists, the role has the correct permissions to access the
 resource and that Kinesis Analytics can assume the role provided.",
```

```
 "inputId":"1.1",
 "errorCode": "InputError",
 "messageSchemaVersion": "1"
}
```

不適切な Amazon S3 ファイルキーがリファレンスデータで使用されている場合は、次のような エラーが生成されます。

```
{
 "applicationARN": "arn:aws:kinesisanalytics:us-east-1:112233445566:application/
sampleApp",
 "applicationVersionId": "5",
 "messageType": "ERROR",
 "message": "There is a problem related to the configuration of your reference data.
 Please check that the bucket and the file exist, the role has the correct permissions
 to access these resources and that Kinesis Analytics can assume the role provided.",
 "referenceId":"1.1",
 "errorCode": "ReferenceDataError",
 "messageSchemaVersion": "1"
}
```

ロールが存在しません

存在しない IAM 入力ロールで ARN が指定されているが、その ARN が正しくない場合、以下のような エラーが生成されます。

```
{
 "applicationARN": "arn:aws:kinesisanalytics:us-east-1:112233445566:application/
sampleApp",
 "applicationVersionId": "5",
 "messageType": "ERROR",
 "message": "There is a problem related to the configuration of your input. Please
 check that the resource exists, the role has the correct permissions to access the
 resource and that Kinesis Analytics can assume the role provided.",
 "inputId":null,
 "errorCode": "InputError",
 "messageSchemaVersion": "1"
}
```

リソースにアクセスするアクセスする権限がロールにありません

入力リソースにアクセスするためのアクセス権限がない入力ロール (例: Kinesis ソースストリーム) が使用されている場合は、次のようなエラーが表示されます。

```
{
 "applicationARN": "arn:aws:kinesisanalytics:us-east-1:112233445566:application/
sampleApp",
 "applicationVersionId": "5",
 "messageType": "ERROR",
 "message": "There is a problem related to the configuration of your input. Please
 check that the resource exists, the role has the correct permissions to access the
 resource and that Kinesis Analytics can assume the role provided.",
 "inputId":null,
 "errorCode": "InputError",
 "messageSchemaVersion": "1"
}
```

での AWS CloudTrail API コールのログ記録

は、 のユーザー AWS CloudTrail、ロール、または サービスによって実行されたアクションを記録する AWS サービスである と統合されています。CloudTrail は、 のすべての API コールをイベントとしてキャプチャします。キャプチャされたコールには、 コンソールのコールと、 API オペレーションへのコードのコールが含まれます。証跡を作成する場合は、 のイベントなど、Amazon S3 バケットへの CloudTrail イベントの継続的な配信を有効にすることができます 証跡を設定しない場合でも、「CloudTrail」 コンソールの 「イベント履歴」 で最新のイベントを表示できます。CloudTrail が収集した情報を使用して、 に対して行われた要求、要求が行われた IP アドレス、要求を行った人、要求が行われた日時、および追加の詳細を判別できます。

CloudTrail の詳細については、「AWS CloudTrail ユーザーガイド」を参照してください。

CloudTrail での情報

CloudTrail は、 AWS アカウントの作成時にアカウントで有効になります。でアクティビティが発生すると、そのアクティビティはイベント履歴の他の AWS サービスイベントとともに CloudTrail イベントに記録されます。 AWS アカウントで最近のイベントを表示、検索、ダウンロードできます。詳細については、「CloudTrailイベント履歴でのイベントの表示」を参照してください。

のイベントなど、 AWS アカウントのイベントの継続的な記録については、証跡を作成します。証跡により、CloudTrail はログファイルを Amazon S3 バケットに配信できます。デフォルトでは、コ

ンソールで証跡を作成するときに、証跡がすべての に適用されます。証跡は、 AWS パーティション内のすべてのリージョンからのイベントをログに記録し、指定した Amazon S3 バケットにログファイルを配信します。さらに、CloudTrail ログで収集されたイベントデータをより詳細に分析し、それに基づいて行動するように、他の AWS サービスを設定できます。詳細については、次を参照してください:

- 証跡の作成のための概要
- CloudTrail がサポートするサービスと統合
- CloudTrail 用 Amazon SNS 通知の構成
- 複数のリージョンから CloudTrail ログファイルを受け取る、および複数のアカウントから CloudTrail ログファイルを受け取る

すべてのアクションは CloudTrail によってログに記録され、「 API リファレンス」に記録されます。例えば、`CreateApplication` および `UpdateApplication` の各アクションを呼び出すと、CloudTrail ログファイルにエントリが生成されます。

各イベントまたはログエントリには、リクエストの生成者に関する情報が含まれます。アイデンティティ情報は、以下を判別するのに役立ちます。

- リクエストが AWS アカウントのルートユーザー または ユーザー認証情報を使用して行われたかどうか。
- リクエストがロールまたはフェデレーションユーザーのテンポラリなセキュリティ認証情報を使用して行われたかどうか。
- リクエストが別の AWS サービスによって行われたかどうか。

詳細については、CloudTrail userIdentity 要素を参照してください。

ログファイルエントリの概要

追跡は、指定したAmazon S3バケットにイベントをログファイルとして配信するように設定できます。CloudTrail のログファイルは、単一か複数のログエントリを含みます。イベントは任意ソースからの単一リクエストを表し、リクエストされたアクション、アクションの日時、リクエストパラメータなどの情報を含みます。CloudTrail ログファイルは、パブリック API 呼び出しの順序付けられたスタックトレースではないため、特定の順序では表示されません。

次の例は、AddApplicationCloudWatchLoggingOption アクションと DescribeApplication アクションを示す CloudTrail ログエントリを表しています。

```
{
    "Records": [
        {
            "eventVersion": "1.05",
            "userIdentity": {
                "type": "IAMUser",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::012345678910:user/Alice",
                "accountId": "012345678910",
                "accessKeyId": "EXAMPLE_KEY_ID",
                "userName": "Alice"
            },
            "eventTime": "2019-03-14T01:03:00Z",
            "eventSource": "kinesisanalytics.amazonaws.com",
            "eventName": "AddApplicationCloudWatchLoggingOption",
            "awsRegion": "us-east-1",
            "sourceIPAddress": "127.0.0.1",
            "userAgent": "aws-sdk-java/unknown-version Linux/x.xx",
            "requestParameters": {
                "currentApplicationVersionId": 1,
                "cloudWatchLoggingOption": {
                    "roleARN": "arn:aws:iam::012345678910:role/cloudtrail_test",
                    "logStreamARN": "arn:aws:logs:us-east-1:012345678910:log-
group:cloudtrail-test:log-stream:sql-cloudwatch"
                },
                "applicationName": "cloudtrail-test"
            },
            "responseElements": null,
            "requestID": "e897cd34-45f4-11e9-8912-e52573a36cd9",
            "eventID": "57fe50e9-c764-47c3-a0aa-d0c271fa1cbb",
            "eventType": "AwsApiCall",
            "apiVersion": "2015-08-14",
            "recipientAccountId": "303967445486"
        },
        {
            "eventVersion": "1.05",
            "userIdentity": {
                "type": "IAMUser",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::012345678910:user/Alice",
                "accountId": "012345678910",
                "accessKeyId": "EXAMPLE_KEY_ID",
                "userName": "Alice"
```

```
            },
            "eventTime": "2019-03-14T05:37:20Z",
            "eventSource": "kinesisanalytics.amazonaws.com",
            "eventName": "DescribeApplication",
            "awsRegion": "us-east-1",
            "sourceIPAddress": "127.0.0.1",
            "userAgent": "aws-sdk-java/unknown-version Linux/x.xx",
            "requestParameters": {
                "applicationName": "cloudtrail-test"
            },
            "responseElements": null,
            "requestID": "3b74eb29-461b-11e9-a645-fb677e53d147",
            "eventID": "750d0def-17b6-4c20-ba45-06d9d45e87ee",
            "eventType": "AwsApiCall",
            "apiVersion": "2015-08-14",
            "recipientAccountId": "012345678910"
        }
    ]
}
```

制限

サービス終了日

慎重な検討の結果、Amazon Kinesis Data Analytics for SQL アプリケーションのサポートは終了することになりました。お客様が計画的に Amazon Kinesis Data Analytics for SQL アプリケーションから移行できるように、完全なサポート終了までに 15 か月間の猶予を設け、その間に段階的に終了していく予定です。重要な日付となるのは、2025 年 10 月 15 日と 2026 年 1 月 27 日の 2 つです。

1. 2025 年 10 月 15 日には、アプリケーションが停止され、READY 状態に設定されます。この時点ではアプリケーションを再び起動でき、サービスの制限に従って、引き続き通常どおりにアプリケーションを使用できます。
2. 2025 年 10 月 15 日以降、新しい Amazon Kinesis Data Analytics for SQL アプリケーションを作成することはできなくなります。既存のアプリケーションは、サービスの制限に従って通常どおりに実行できます。
3. 2026 年 1 月 27 日以降、アプリケーションは削除されます。Amazon Kinesis Data Analytics for SQL アプリケーションを起動することも操作することもできなくなります。これ以降、Amazon Kinesis Data Analytics for SQL アプリケーションのサポートは終了します。

2025 年 10 月 15 日より前に、アプリケーションを Amazon Managed Service for Apache Flink または Amazon Managed Service for Apache Flink Studio に移行することをお勧めします。移行に役立つリソースについては、「Managed Service for Apache Flink Studio への移行例」を参照してください。Amazon Managed Service for Apache Flink または Amazon Managed Service for Apache Flink Studio の詳細については、「Amazon Managed Service for Apache Flink デベロッパーガイド」を参照してください。

制限

Amazon Kinesis Data Analytics for SQL アプリケーションを使用する場合、次の制限に注意してください。

- Kinesis Data Analytics for SQL は、米国東部 (オハイオ) 、米国東部 (バージニア北部) 、米国西部 (オレゴン) 、カナダ (中部) 、欧州 (パリ) 、欧州 (アイルランド) 、欧州 (フランクフルト) 、欧州 (ロンドン) 、アジアパシフィック (香港) 、アジアパシフィック (ムンバイ) 、アジアパシフィック (シドニー) 、アジアパシフィック (シンガポール) 、アジアパシフィック (ソウル) 、アジアパシフィック (東京) 、南米 (サンパウロ) 、 AWS GovCloud (米国東部) 、 AWS

GovCloud (米国西部) の各 AWS リージョンで利用できます。追加の AWS リージョンで Kinesis Data Analytics for SQL を起動する予定はありません。

- 2023 年 6 月 28 日以降、Kinesis Data Analytics for SQL をまだ使用していない場合、 AWS マネジメントコンソールを使用して新しい Kinesis Data Analytics for SQL アプリケーションを作成することはできません。Kinesis Data Analytics for SQL のサポート終了日については、「サービス終了日」を参照してください。2023 年 6 月 28 日より前に Kinesis Data Analytics for SQL アプリケーションを作成した場合、Kinesis Data Analytics for SQL を既に使用している AWS リージョンで、現在アプリケーションを作成および実行する方法に変更はありません。ただし、Kinesis Data Analytics for SQL を使用しないリージョンでは、 AWS コンソールを使用して新しいアプリケーションを作成できなくなります。
- 2023 年 9 月 12 日以降、Kinesis Data Analytics for SQL をまだ使用していない場合、Kinesis Data Firehose をソースとして使用する新しいアプリケーションを作成することはできません。Kinesis Data Analytics for SQL のサポート終了日については、「サービス終了日」を参照してください。`KinesisFirehoseInput` と一緒に Kinesis Data Analytics for SQL アプリケーションを使用している既存のお客様は、Kinesis Data Analytics を使用して既存のアカウント内で、引き続き `KinesisFirehoseInput` でアプリケーションを追加できます。既存のお客様で、`KinesisFirehoseInput` で Kinesis Data Analytics for SQL アプリケーションを使用して新規アカウントを作成したい場合は、サポートケースを開くことができます。詳細については、AWS サポート センターを参照してください。
- アプリケーション内ストリームの行のサイズは 512 KB に制限されています。Kinesis Data Analytics は、メタデータを保存するために最大 1 KB を使用します。このメタデータは、行の制限に対してカウントされます。ストリーミングソースのレコードサイズが 50 KB より大きい場合は、入力設定で行の区切り記号を入力して適切なスキーマを指定することで、レコードをアプリケーションストリーム内で複数の行に分割できます。
- アプリケーション内の SQL コードは 100 KB に制限されています。
- ウィンドウクエリで推奨される最長ウィンドウは 1 時間です。アプリケーション内ストリームは揮発性ストレージに保存され、アプリケーションが予期せず中断すると、アプリケーションによって揮発性ストレージ内のソースデータからストリームが再構築されます。
- 1 つのアプリケーション内ストリームに対して推奨される最大スループットは、アプリケーションのクエリの複雑さに応じて、2 ~ 20 MB/秒です。
- アカウントでは、 AWS リージョンごとに最大 50 個の Kinesis Data Analytics アプリケーションを作成できます。サービス制限拡大フォームを通して、追加のアプリケーションをリクエストするケースを作成できます。詳細については、AWS サポート センターを参照してください。
- 単一の Kinesis Data Analytics for SQL アプリケーションで処理できる最大ストリーミングスループットは、約 100 MB/秒です。これは、アプリケーション内ストリームの数を最大値の 64 に増や

し、KPU の制限を 8 より大きくしたことを前提にしています (詳細については、以下の制限を参照してください)。アプリケーションで 100 MB/秒以上の入力を処理する必要がある場合は、次のいずれかの操作を行います。

- 複数の Kinesis Data Analytics for SQL を使用して入力を処理する
- 引き続き 1 つのストリームとアプリケーションを使用する場合は、Managed Service for Apache Flink for Java Applications を使用します。

> **Note**
>
> アプリケーションの予測入力スループットが 100 MB/秒を超える場合は、複数の SQL アプリケーションを使用する計画を事前に立てたり、Amazon Managed Service for Apache Flink for Java Applications に移行したりできるように、アプリケーションの `InputProcessing.OkBytes` メトリクスを定期的に確認することをお勧めします。また、アプリケーションが入力スループットの制限に近づいたときに通知できるように、`InputProcessing.OkBytes` に関する CloudWatch アラームを作成することをお勧めします。これは、アプリケーションのクエリを更新してスループットの向上とトレードオフすることができ、分析のバックプレッシャーや遅延を回避できるので便利です。詳細については、「トラブルシューティング」を参照してください。アラームは、アップストリームのスループットを低下させるメカニズムがある場合にも役立ちます。

- Kinesis 処理ユニット (KPU) の数は 8 に制限されています。この制限の拡大をリクエストする方法については、「Amazon サービスの制限」にある「制限の拡大をリクエストするには」を参照してください。

  Kinesis Data Analytics は、使用した分のみお支払いいただくだけで利用可能です。ストリーム処理アプリケーションの実行に使用される KPU の平均数に基づいた時間料金で課金されます。1 つの KPU で 1 つの vCPU および 4 GB のメモリーが提供されます。
- 各アプリケーションは 1 つのストリーミングソースと最大 1 つのリファレンスデータソースを持つことができます。
- Kinesis Data Analytics アプリケーションには、最大 3 つの宛先を設定できます。これらの宛先の 1 つを使用して、アプリケーション内のエラーストリームデータを保持することをお勧めします。
- リファレンスデータを保存する Amazon S3 オブジェクトのサイズは最大 1 GB です。
- アプリケーション内テーブルにリファレンスデータをアップロードした後に S3 バケットに保存されているリファレンスデータを変更する場合は、 UpdateApplicationオペレーション (API または を使用 AWS CLI) を使用してアプリケーション内テーブルのデータを更新する必要があります。現

在、 AWS Management Console はアプリケーションのリファレンスデータの更新をサポートしていません。

- 現在のところ、Kinesis Data Analytics では Amazon Kinesis Producer Library (KPL) で生成されたデータはサポートされていません。
- アプリケーションあたり最大 50 個のタグを割り当てることができます。

ベストプラクティス

このセクションでは、Amazon Kinesis Data Analytics アプリケーションを使用する場合のベストプラクティスを説明します。

トピック

- アプリケーションを管理する
- アプリケーションのスケーリング
- アプリケーションのモニタリング
- 入力スキーマの定義
- 出力への接続
- アプリケーションコードの作成
- アプリケーションのテスト

アプリケーションを管理する

Amazon Kinesis Data Analytics アプリケーションを管理するときは、以下のベストプラクティスに従います。

- Amazon CloudWatch アラームのセットアップ – Kinesis Data Analytics が提供する CloudWatch メトリクスを使用して、以下をモニタリングできます。
  - 入力バイトおよび入力レコード (アプリケーションに入力されるバイト数およびレコード数)
  - 出力バイトおよび出力レコード
  - `MillisBehindLatest` (アプリケーションがストリーミングソースからの読み取りにおいてどの程度遅延しているか)

  本稼働アプリケーションでは、以下のメトリクスで最低 2 つの CloudWatch アラームをセットアップすることをお勧めします。

  - `MillisBehindLatest` – ほとんどの場合、アプリケーションが 1 分間の平均で最新のデータから 1 時間遅延した場合にトリガーされるようにこのアラームを設定することをお勧めします。より低いレベルのエンドツーエンド処理が必要なアプリケーションでは、許容度を下げた設定に調整できます。このアラームを使用して、アプリケーションが最新データを読み取っていることを確認できます。

- `ReadProvisionedThroughputException` の例外の発生を回避するには、同じ Kinesis データストリームから読み取る本稼働アプリケーションの数を 2 つのアプリケーションに制限します。

> **Note**
>
> この場合、アプリケーションは、ストリーミングソースを読み取ることができる任意のアプリケーションを意味します。Firehose 配信ストリームから読み取ることができるのは、Kinesis Data Analytics アプリケーションだけです。ただし、Kinesis Data Analytics アプリケーションや など、多くのアプリケーションは Kinesis データストリームから読み取ることができます AWS Lambda。アプリケーション制限の推奨は、ストリーミングソースを読み取ることができるすべてのアプリケーションに対するものです。

Amazon Kinesis Data Analytics は、アプリケーションごとに 1 秒に約 1 度、ストリーミングソースを読み取ります。ただし、遅延したアプリケーションは、追いつくためにより高速でデータを読み取る場合があります。アプリケーションが追い付くために十分なスループットを使用できるように、同じデータソースを読み取るアプリケーションの数を制限します。

- 同じ Firehose 配信ストリームから読み取る本稼働アプリケーションの数を、1 つのアプリケーションに制限します。

  Firehose 配信ストリームは、Amazon S3 や Amazon Redshift などの宛先に書き込むことができます。これは、Kinesis Data Analytics アプリケーションのストリーミングソースとすることもできます。したがって、1 つの Firehose 配信ストリームに複数の Kinesis Data Analytics アプリケーションを設定しないようにすることをお勧めします。これにより、配信ストリームが別の宛先にも配信できるようになります。

アプリケーションのスケーリング

アプリケーション内入力ストリームの数をデフォルト (1) からプロアクティブに増やすことで、将来のスケーリングニーズに備えてアプリケーションをセットアップします。アプリケーションのスループットに基づいて、次の言語を選択することをお勧めします。

- アプリケーションのスケーリングニーズが 100 MB/秒を超える場合は、複数のストリームと Kinesis Data Analytics for SQL アプリケーションを使用します。
- 引き続き 1 つのストリームとアプリケーションを使用する場合は、Managed Service for Apache Flink Applications を使用します。

> **Note**
> アプリケーションの予測入力スループットが 100 MB/秒を超える場合は、複数の SQL アプリケーションを使用する計画を事前に立てたり、Apache Flink for Java アプリケーション用の managed-flink/latest/java/ に移行したりできるように、アプリケーションの `InputProcessing.OkBytes` メトリックを定期的に確認することをお勧めします。

アプリケーションのモニタリング

アプリケーションが入力スループットの制限に近づいたときに通知されるように、`InputProcessing.OkBytes` に CloudWatch アラームを作成することをお勧めします。これは、アプリケーションのクエリを更新してスループットの向上とトレードオフすることができ、分析のバックプレッシャーや遅延を回避できるので便利です。詳細については、「トラブルシューティング」を参照してください。これは、アップストリームのスループットを低下させるメカニズムがある場合にも役立ちます。

- 1 つのアプリケーション内ストリームに対して推奨される最大スループットは、アプリケーションのクエリの複雑さに応じて、2 ~ 20 MB/秒です。
- 単一の Kinesis Data Analytics for SQL アプリケーションで処理できる最大ストリーミングスループットは、約 100 MB/秒です。これは、アプリケーション内ストリームの数を最大値の 64 に増やし、KPU の制限を 8 より大きくしたことを前提にしています。詳細については、「制限」を参照してください。

> **Note**
> アプリケーションの予測入力スループットが 100 MB/秒を超える場合は、複数の SQL アプリケーションを使用する計画を事前に立てたり、Apache Flink for Java アプリケーション用の managed-flink/latest/java/ に移行したりできるように、アプリケーションの `InputProcessing.OkBytes` メトリックを定期的に確認することをお勧めします。

入力スキーマの定義

コンソールでアプリケーション入力を設定するばあいは、まずストリーミングソースを指定します。その後、コンソールは配信 API (「DiscoverInputSchema」を参照) を使用してストリーミングソースのレコードをサンプリングし、スキーマを推測します。このスキーマは、特に、結果のアプリケーション内ストリームの列の名前およびデータ型を定義します。コンソールにスキーマが表示されます。この推測スキーマで以下を行うことをお勧めします。

- 推測スキーマを十分にテストします。検出プロセスでは、ストリーミングソースのレコードのサンプルのみを使用してスキーマを推測します。ストリーミングソースに多くのレコードタイプがある場合、検出 API で 1 つまたは複数のレコードタイプのサンプリングが行われていない可能性があります。この状況により、ストリーミングソースのデータを正確に反映しないスキーマが発生する可能性があります。

  アプリケーションが起動した時に、これらの除外されたレコードタイプによって解析エラーが発生する場合があります。Amazon Kinesis Data Analytics は、これらのレコードをアプリケーション内エラーストリームに送信します。このような解析エラーを減らすために、推測スキーマをコンソールでインタラクティブにテストし、欠落したレコードがないかアプリケーション内ストリームをモニタリングすることをお勧めします。

- API では、入力設定で列に対する `NOT NULL` 制約の指定はサポートされていません。アプリケーション内ストリームの列に `NOT NULL` 制約をつける場合は、アプリケーションコードを使用してそのようなアプリケーション内ストリームを作成します。その後、1 つのアプリケーション内ストリームから別の 1 つにデータをコピーすると、制約が反映されます。

  値が必要なときに `NULL` 値を含む行を挿入しようとすると、エラーが発生します。Kinesis Data Analytics は、これらのエラーをアプリケーション内エラーストリームに送信します。

- 検出処理で推測されるデータ型を緩和します。検出プロセスでは、ランダムにサンプリングしたストリーミングソースのレコードに基づいて列とデータ型が推奨されます。これらを注意深く確認して、入力したレコードのあらゆるケースをカバーできるようにデータ型の緩和を検討することをお勧めします。これにより、アプリケーションの実行中に全体で解析エラーを減らすことができます。たとえば、推測スキーマに列タイプとして `SMALLINT` がある場合、これを `INTEGER` に変更することを検討します。

- アプリケーションコードで SQL 関数を使用して、非構造化データまたは列を処理します。入力に、ログデータなど、非構造化データまたは列がある場合があります。例については「例: DateTime 値の変換」を参照してください。このタイプのデータを処理する方法の 1 つとして、タイプが `VARCHAR(N)` である列 1 つのみを持つようにスキーマを定義する方法があります。ここでの N はストリームで発生する可能性のある最も大きい行です。その後、入力されるレコードをアプリケーションコードが読み取り、`String` および `Date Time` 関数を使用して、未加工データを解析してスキーマ化します。

- 2 レベル以上の入れ子構造になっているストリーミングソースデータが完全に処理されていることを確認します。ソースデータが JSON である場合、入れ子構造になっていることがあります。検出 API は入れ子構造を 1 レベルにフラット化したスキーマを推測します。2 レベルの入れ子構造の場合も、検出 API でこれらのフラット化を試みます。2 レベルを超える入れ子構造の場合、フラット化のサポートには制限があります。入れ子構造を完全に処理するには、ニーズに合わせて推測スキーマを手動で編集する必要があります。以下の方法のいずれかを使用して行ってください。

  - JSON 列パスを使用して、アプリケーションに必要なキーと値のペアのみを、選択的に抽出します。JSON 列パスは、アプリケーションに持ってくる特定のキーと値のペアに対するポインタを提供します。これは入れ子構造の任意のレベルに対して実行できます。
  - JSON 列パスを使用して複雑な JSON オブジェクトを選択的に抽出し、アプリケーションコードで文字列操作関数を使用して必要な特定のデータを抽出します。

出力への接続

各アプリケーションに最低 2 つの出力を設定することをお勧めします。

- 最初の宛先は、SQL クエリの結果を挿入するために使用します。
- 2 つ目の宛先は、エラーストリーム全体を挿入し、Firehose 配信ストリームを経由して S3 バケットに送信するために使用します。

アプリケーションコードの作成

次の構成を推奨します。

- 以下の理由で、SQL ステートメントでは 1 時間を超える時間ベースのウィンドウを指定しないでください。
  - 場合によっては、アプリケーションを再起動する必要があります。これは、アプリケーションを更新したためか、Kinesis Data Analytics の内部的な理由によるものです。再起動すると、ウィンドウに含まれるすべてのデータはストリーミングデータソースからもう一度読み取る必要があります。これにより、Kinesis Data Analytics でこのウィンドウの出力を発行できるようになるまでに時間がかかります。
  - Kinesis Data Analytics は、関連データを含むアプリケーションの状態に関連するすべてを、期間中保持する必要があります。これには、Kinesis Data Analytics 処理ユニットが大量に消費されます。
- 開発中は、結果がより速く表示されるように、SQL ステートメントでウィンドウのサイズを小さくしておいてください。アプリケーションを本稼働環境にデプロイするときに、ウィンドウを適切なサイズに設定できます。
- 1 つの複雑な SQL ステートメントよりも、複数のステートメントに分割して、それぞれのステップで結果を中間アプリケーション内ストリームに保存することを検討してください。迅速なデバッグに役立ちます。
- タンブリングウィンドウを使用する場合は、2 つのウィンドウを使用し、1 つを処理時間、もう 1 つを論理時間 (取り込み時間またはイベント時間) にすることをお勧めします。詳細については、「タイムスタンプと ROWTIME 列」を参照してください。

アプリケーションのテスト

Kinesis Data Analytics アプリケーションのスキーマまたはアプリケーションコードを変更する場合は、テストアプリケーションを使用して、本稼働環境にデプロイする前に変更を確認することをお勧めします。

テストアプリケーションのセットアップ

テストアプリケーションのセットアップは、コンソールを通じて、または AWS CloudFormation テンプレートを使用して行うことができます。 AWS CloudFormation テンプレートを使用すると、テストアプリケーションとライブアプリケーションに加えたコード変更が一貫していることを確認できます。

テストアプリケーションをセットアップする場合は、ライブデータにアプリケーションを接続するか、テスト対象のストリームにモックデータを入力できます。ストリームにモックデータを入力するために、次の 2 つの方法をお勧めします。

- Kinesis Data Generator (KDG) を使用します。KDG では、データテンプレートを使用して、Kinesis ストリームにランダムなデータを送信します。KDG は簡単に使用できますが、データホットスポットや異常を検出するアプリケーションなど、データ項目間の複雑な関係をテストすることには適していません。
- より複雑なデータを Kinesis データストリームに送信するには、カスタム Python アプリケーションを使用します。Python アプリケーションはホットスポットや異常など、データ項目間の複雑な関係を生成できます。データホットスポットにクラスター化されたデータを送信する Python アプリケーションの例については、「例 : ストリーム上のホットスポットの検出 (HOTSPOTS 関数)」を参照してください。

テストアプリケーションの実行時には、コンソールでアプリケーション内ストリームを確認するのではなく、宛先 (Amazon Redshift データベースへの Firehose 配信ストリームなど) を使用して結果を確認します。コンソールに表示されるデータは、ストリームのサンプリングであり、すべてのレコードは含まれません。

スキーマ変更のテスト

アプリケーションの入力ストリームのスキーマを変更する場合は、テストアプリケーションを使用して、以下の条件が満たされていることを確認します。

- ストリームからのデータが正しいデータ型に強制変換される。たとえば、日時データが文字列としてアプリケーションに取り込まれないことを確認します。
- データが解析され、目的のデータ型に強制変換される。解析または強制変換エラーが発生した場合は、コンソールで表示できます。または、エラーストリームに送信先を割り当てて、送信先ストアでエラーを表示できます。
- 文字データのデータフィールドが十分な長さで、アプリケーションが文字データを切り捨てない。送信先ストアのデータレコードで、アプリケーションデータが切り捨てられていないことを確認できます。

コード変更のテスト

SQL コードの変更のテストでは、アプリケーションのドメインに関する知識がいくらか必要です。テストする必要がある出力と、正しい出力について判断できる必要があります。アプリケーションの

SQL コードを変更するときに確認する、問題の起きそうな分野については、「Amazon Kinesis Data Analytics for SQL アプリケーションのトラブルシューティング」を参照してください。

Amazon Kinesis Data Analytics for SQL アプリケーションのトラブルシューティング

以下は、Amazon Kinesis Data Analytics for SQL アプリケーションで発生する可能性のある問題を解決するために役立ちます。

トピック

- 停止したアプリケーション
- SQL コードを実行できません
- スキーマを検出または発見できない
- リファレンスデータが古い
- アプリケーションが送信先に書き込まれない
- モニタリングする重要なアプリケーションの状態パラメータ
- アプリケーションを実行するときの無効なコードエラー
- アプリケーションによってエラーがエラーストリームに書き込まれている
- スループット不足または高い MillisBehindLatest

停止したアプリケーション

- Amazon Kinesis Data Analytics for SQL アプリケーションとは何ですか?

  停止されたアプリケーションとは、最低 3 か月間レコードを処理していないことが確認されたアプリケーションのことです。つまり、顧客が使用していない Kinesis Data Analytics for SQL のリソースに対して料金を支払っているということです。

- はアイドル状態のアプリケーションの停止をいつ AWS 開始しますか？

  AWS は、2023 年 11 月 14 日にアイドル状態のアプリケーションの停止を開始し、2023 年 11 月 21 日までに完了します。そのリージョンのタイムゾーンの営業時間に、アイドル状態のアプリケーションを停止します。

- 停止した Kinesis Data Analytics for SQL アプリケーションを再起動できますか?

  はい。アプリケーションを再起動する必要がある場合は、通常どおり再起動できます。サポートチケットを提出する必要はありません。

- がアイドル状態のアプリケーション AWS を停止すると、クエリ結果も削除されますか？

いいえ。まず、アプリケーションはアイドル状態なので、クエリを処理しません。次に、クエリ結果は Kinesis Data Analytics for SQL には保存されません。Kinesis Data Analytics for SQL アプリケーションに、計算結果が送信されるシンクの宛先 (Amazon S3 または別のデータストリームなど) を設定します。そのため、お客様がデータの完全な所有権を保持し、ストレージサービスの条件に基づいてデータを引き続き取得できます。

- アプリケーションを停止したくない場合はどうすれば良いですか?

  2023 年 11 月 10 日までにアプリケーションを停止しないように要請するメールをサービスチーム (kda-sql-questions@amazon.com) に送信してください。メールには、アカウント ID とアプリケーション ARN を記載する必要があります。

SQL コードを実行できません

特定の SQL ステートメントを正しく動作させる方法を判断する必要がある場合は、Kinesis Data Analytics を使用するときにさまざまなリソースを使用できます。

- SQL ステートメントの詳細については、「Kinesis Data Analytics for SQL の例」を参照してください。このセクションでは、使用可能な SQL の例が多数紹介されています。
- Amazon Kinesis Data Analytics SQL Reference では、ストリーミング SQL ステートメントの記述に関する詳細なガイドを提供しています。
- それでも問題が解決しない場合は、「Kinesis Data Analytics フォーラム」に質問することをお勧めします。

スキーマを検出または発見できない

場合によっては、Kinesis Data Analytics でスキーマを検出または発見できないことがあります。このような場合の多くは、Kinesis Data Analytics をそのまま使用できます。

区切り文字を使用しない UTF-8 エンコードデータ、カンマ区切り値 (CSV) 以外の形式を使用するデータ、またはスキーマを検出しなかった検索 API があると想定します。このような場合は、スキーマを手動で定義するか、文字列操作関数を使用して、データを構造化できます。

ストリームのスキーマを検出するために、Kinesis Data Analytics はランダムにストリームの最新データをサンプリングします。ストリームに継続的にデータを送信していない場合、Kinesis Data Analytics はサンプルの取得やスキーマの検出を行えないことがあります。詳細については、「ストリーミングデータのスキーマ検出機能の使用」を参照してください。

リファレンスデータが古い

リファレンスデータは、アプリケーションの起動または更新時、あるいはサービスの問題によって発生したアプリケーションの中断時に、Amazon Simple Storage Service (Amazon S3) オブジェクトからアプリケーションにロードされます。

基になる Amazon S3 オブジェクトが更新された場合、リファレンスデータはアプリケーションにロードされません。

アプリケーション内のリファレンスデータが最新ではない場合、次のステップに従ってデータを再ロードできます。

1. Kinesis Data Analytics コンソールで、リストからアプリケーション名を選択し、[アプリケーションの詳細] を選択します。
2. [SQL エディタに移動] を選択し、アプリケーションの [リアルタイム分析] ページを開きます。
3. [ソースデータ] ビューで、リファレンスデータテーブル名を選択します。
4. [アクション]、[リファレンスデータテーブルの同期] の順に選択します。

アプリケーションが送信先に書き込まれない

データが送信先に書き込まれない場合は、以下を確認してください。

- アプリケーションのロールに、送信先へのアクセスに十分なアクセス許可があることを確認します。詳細については、Kinesis ストリームに書き込むためのアクセス権限ポリシー または Firehose 配信ストリームに書き込むためのアクセス権限ポリシー を参照してください。
- アプリケーションの送信先が正しく設定されていることと、アプリケーションの出力ストリームに正しい名前が使用されていることを確認します。
- データが書き込まれているかどうかを確認するには、出力ストリームの Amazon CloudWatch メトリクスを確認します。CloudWatch メトリクスの詳細については、「Amazon CloudWatch によるモニターリング」を参照してください。
- the section called “AddApplicationCloudWatchLoggingOption” を使用して CloudWatch ログストリームを追加します。アプリケーションはログストリームに設定エラーを書き込みます。

ロールと送信先が正しく設定されている場合は、アプリケーションを再起動し、`LAST_STOPPED_POINT` に InputStartingPositionConfiguration を指定します。

モニタリングする重要なアプリケーションの状態パラメータ

アプリケーションが正しく実行されていることを確認するには、特定の重要なパラメータをモニタリングすることをお勧めします。

最も重要なパラメータとして、Amazon CloudWatch メトリクス (`MillisBehindLatest`) をモニタリングする必要があります。このメトリクスは、ストリームの読み込みが現在時刻からどの程度遅延しているかを表します。このメトリクスは、ソースストリームからのレコードの処理が十分に高速かどうかを判断するのに役立ちます。

一般的に、1 時間以上遅れる場合は、CloudWatch アラームのトリガーを設定する必要があります。ただし、これにかかる時間はユースケースによって異なります。時間は必要に応じて調整することができます。

詳細については、「ベストプラクティス」を参照してください。

アプリケーションを実行するときの無効なコードエラー

Amazon Kinesis Data Analytics アプリケーションの SQL コードを保存および実行できない場合、一般的な原因は次のとおりです。

- ストリームが SQL コードで再定義された – ストリームと、ストリームに関連付けられたポンプを作成した後で、コードで同じストリームを再定義することはできません。ストリームの作成の詳細については、Amazon Kinesis Data Analytics SQL Reference の「CREATE STREAM」を参照してください。ポンプの作成の詳細については、「CREATE PUMP」を参照してください。
- GROUP BY 句で複数の ROWTIME 列を使用している – GROUP BY 句には、1 つの ROWTIME 列のみ指定できます。詳細については、Amazon Kinesis Data Analytics SQL Reference の「GROUP BY」および「ROWTIME」を参照してください。
- 1 つ以上のデータ型に無効なキャストがある – この場合は、コードに無効で暗黙的なキャストがあります。たとえば、コードの `bigint` に `timestamp` をキャストする場合があります。
- ストリームに、サービスの予約済みストリーム名と同じ名前がある – ストリームに、サービスの予約済みストリーム (`error_stream`) と同じ名前を付けることはできません。

アプリケーションによってエラーがエラーストリームに書き込まれている

アプリケーションによってエラーがアプリケーション内のエラーストリームに書き込まれている場合は、標準ライブラリを使用して、DATA_ROW フィールドの値をデコードします。エラーストリームの詳細については、「エラー処理」を参照してください。

スループット不足または高い MillisBehindLatest

アプリケーションの MillisBehindLatest メトリクスが常に増加しているか、継続的に 1000 (1 秒) を超える場合は、次のような理由が考えられます。

- アプリケーションの InputBytes CloudWatch メトリクスを確認します。取り込みが 4 MB/秒を超えている場合は、これが MillisBehindLatest が増加する原因となっている可能性があります。アプリケーションのスループットを向上させるため、InputParallelism パラメータの値を増やします。詳細については、「スループットの増加に合わせた入力ストリームの並列処理」を参照してください。
- 送信先への配信の失敗について、アプリケーションの出力配信の Success メトリクスを確認します。出力を正しく設定済みで、出力ストリームに十分な容量があることを確認します。
- アプリケーションで事前処理または出力に AWS Lambda 関数を使用している場合は、アプリケーションの InputProcessing.Duration または LambdaDelivery.Duration CloudWatch メトリクスを確認します。Lambda 関数の呼び出し所要時間が 5 秒より長い場合は、次のことを検討してください。
  - Lambda 関数のメモリ割り当てを増やします。この操作は、 AWS Lambda コンソールの [設定] ページの [基本設定] で行うことができます。詳細については、https://docs.aws.amazon.com/lambda/latest/dg/resource-model.html 開発者ガイドの「AWS Lambda Lambda 関数の設定」を参照してください。
  - アプリケーションの入力ストリームで、シャードの数を増やします。これにより、アプリケーションが呼び出す並行関数の数が増え、スループットが向上する可能性があります。
  - 関数が、外部リソースに対する同時リクエストなど、パフォーマンスに影響を与えるようなブロック呼び出しを実行していないことを確認します。
  - AWS Lambda 関数を調べて、パフォーマンスを向上させることができる他の領域があるかどうかを確認してください。アプリケーション Lambda 関数の CloudWatch Logs を確認します。詳細については、AWS Lambda デベロッパーガイドの「 の Amazon CloudWatch メトリクスへのアクセス」を参照してください。

- アプリケーションが、Kinesis 処理単位 (KPU) のデフォルトの制限に達していないことを確認します。アプリケーションがこの制限に達している場合は、制限の引き上げをリクエストできます。詳細については、「アプリケーションを自動的にスケーリングしてスループットを向上させる」を参照してください。
- KPU の上限を引き上げてもアプリケーションの問題が解決しない場合は、アプリケーションの入力スループットが 100 MB/秒を超えていないことを確認してください。100 MB/秒を超える場合は、Kinesis Data Analytics SQL アプリケーションが読み取るデータソースに送信されるデータ量を減らすなど、アプリケーションを安定させるために全体のスループットを下げるよう、変更を行うことをお勧めします。また、アプリケーションの並列処理を増やす、計算時間を短縮する、列指向データタイプを VARCHAR からサイズの小さいデータ型 (INTEGER、LONG など) に変更する、サンプリングやフィルタリングによって処理されるデータを減らすなど、他のアプローチも推奨していました。

> **Note**
>
> アプリケーションの予測入力スループットが 100 MB/秒を超えることになる場合は、複数の SQL アプリケーションを使用する計画を事前に立てたり、managed-flink/latest/java/ に移行したりできるように、アプリケーションの `InputProcessing.OkBytes` メトリックを定期的に確認することをお勧めします。

Kinesis Data Analytics SQL リファレンス

Kinesis Data Analytics でサポートされている SQL 言語要素の詳細については、「Kinesis Data Analytics SQL Reference」を参照してください。

API リファレンス

> **Note**
>
> このドキュメントの対象は、SQL アプリケーションのみがサポートされる Amazon Kinesis Data Analytics API のバージョン 1 です。バージョン 2 の API では、SQL および Java アプリケーションがサポートされます。バージョン 2 の詳細については、「Amazon Managed Service for Apache Flink API V2 ドキュメント」を参照してください。

を使用して AWS CLI 、Amazon Kinesis Data Analytics API を調べることができます。このガイドでは、 AWS CLIを使用する Amazon Kinesis Data Analytics for SQL Applications の開始方法 の実習について説明します。

トピック

- アクション
- データ型

アクション

以下のアクションがサポートされています:

- AddApplicationCloudWatchLoggingOption
- AddApplicationInput
- AddApplicationInputProcessingConfiguration
- AddApplicationOutput
- AddApplicationReferenceDataSource
- CreateApplication
- DeleteApplication
- DeleteApplicationCloudWatchLoggingOption
- DeleteApplicationInputProcessingConfiguration
- DeleteApplicationOutput
- DeleteApplicationReferenceDataSource

- DescribeApplication
- DiscoverInputSchema
- ListApplications
- ListTagsForResource
- StartApplication
- StopApplication
- TagResource
- UntagResource
- UpdateApplication

AddApplicationCloudWatchLoggingOption

> **Note**
>
> このドキュメントの対象は、SQL アプリケーションのみがサポートされる Amazon Kinesis Data Analytics API のバージョン 1 です。バージョン 2 の API では、SQL および Java アプリケーションがサポートされます。バージョン 2 の詳細については、Amazon Kinesis Data Analytics API V2 のドキュメントを参照してください。

アプリケーションの構成エラーをモニタリングするために CloudWatch ログストリームを追加します。Amazon Kinesis Analytics アプリケーションで CloudWatch ログストリームを使用する方法の詳細については、「Amazon CloudWatch Logs の使用」を参照してください。

リクエストの構文

```
{
   "ApplicationName": "string",
   "CloudWatchLoggingOption": {
      "LogStreamARN": "string",
      "RoleARN": "string"
   },
   "CurrentApplicationVersionId": number
}
```

リクエストパラメーター

リクエストは以下のデータを JSON 形式で受け入れます。

ApplicationName

Kinesis Analytics アプリケーション名。

タイプ: 文字列

長さの制限: 最小長は 1 です。最大長は 128 です。

パターン: `[a-zA-Z0-9_.-]+`

必須: はい

CloudWatchLoggingOption

CloudWatch ログストリームの Amazon リソースネーム (ARN) と IAM ロール ARN を提供します。注意: CloudWatch にアプリケーションメッセージを書き込むには、使用する IAM ロールで `PutLogEvents` ポリシーアクションが有効になっている必要があります。

型: CloudWatchLoggingOption オブジェクト

必須: はい

CurrentApplicationVersionId

Kinesis Analytics アプリケーションのバージョン ID。

型: 長整数

有効範囲: 最小値は 1 です。最大値は 999999999 です。

必須: はい

レスポンス要素

アクションが成功した場合、サービスは空の HTTP 本文を持つ HTTP 200 レスポンスを返します。

エラー

ConcurrentModificationException

アプリケーションへの同時変更の結果としてスローされる例外。例えば、2 人の個人が同じアプリケーションを同時に編集しようとしています。

HTTP ステータスコード: 400

InvalidArgumentException

指定された入力パラメータ値が無効です。

HTTP ステータスコード: 400

ResourceInUseException

このオペレーションではアプリケーションを使用できません。

HTTP ステータスコード: 400

ResourceNotFoundException

指定されたアプリケーションが見つかりません。

HTTP ステータスコード: 400

UnsupportedOperationException

指定されたパラメータがサポートされていないか、指定されたリソースがこのオペレーションに対して有効でないため、リクエストは拒否されました。

HTTP ステータスコード: 400

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

AddApplicationInput

> **Note**
>
> このドキュメントの対象は、SQL アプリケーションのみがサポートされる Amazon Kinesis Data Analytics API のバージョン 1 です。バージョン 2 の API では、SQL および Java アプリケーションがサポートされます。バージョン 2 の詳細については、Amazon Kinesis Data Analytics API V2 のドキュメントを参照してください。

Amazon Kinesis アプリケーションにストリーミングソースを追加します。概念については、「アプリケーション入力の設定」を参照してください。

ストリーミングソースは、アプリケーションの作成時に追加することも、アプリケーションの作成後にこのオペレーションを使用して追加することもできます。詳細については、API リファレンスの「CreateApplication」を参照してください。

このオペレーションを使用したストリーミングソースの追加などのすべての構成の更新により、アプリケーションの新しいバージョンになります。DescribeApplication オペレーションを使用して、現在のアプリケーションバージョンを見つけることができます。

このオペレーションには `kinesisanalytics:AddApplicationInput` アクションを実行するアクセス許可が必要です。

リクエストの構文

```
{
   "ApplicationName": "string",
   "CurrentApplicationVersionId": number,
   "Input": {
      "InputParallelism": {
         "Count": number
      },
      "InputProcessingConfiguration": {
         "InputLambdaProcessor": {
            "ResourceARN": "string",
            "RoleARN": "string"
         }
      },
      "InputSchema": {
```

```
            "RecordColumns": [
               {
                  "Mapping": "string",
                  "Name": "string",
                  "SqlType": "string"
               }
            ],
            "RecordEncoding": "string",
            "RecordFormat": {
               "MappingParameters": {
                  "CSVMappingParameters": {
                     "RecordColumnDelimiter": "string",
                     "RecordRowDelimiter": "string"
                  },
                  "JSONMappingParameters": {
                     "RecordRowPath": "string"
                  }
               },
               "RecordFormatType": "string"
            }
         },
         "KinesisFirehoseInput": {
            "ResourceARN": "string",
            "RoleARN": "string"
         },
         "KinesisStreamsInput": {
            "ResourceARN": "string",
            "RoleARN": "string"
         },
         "NamePrefix": "string"
      }
   }
```

リクエストパラメーター

リクエストは以下のデータを JSON 形式で受け入れます。

ApplicationName

ストリーミングソースを追加する既存 Amazon Kinesis Analytics アプリケーションの名前です。

タイプ: 文字列

長さの制限: 最小長は 1 です。最大長は 128 です。

パターン: `[a-zA-Z0-9_.-]+`

必須: はい

CurrentApplicationVersionId

Amazon Kinesis Analytics アプリケーションの現在のバージョンです。DescribeApplication オペレーションを使用して、現在のアプリケーションバージョンを見つけることができます。

型: 長整数

有効範囲: 最小値は 1 です。最大値は 999999999 です。

必須: はい

Input

追加する入力。

型: Input オブジェクト

必須: はい

レスポンス要素

アクションが成功した場合、サービスは空の HTTP 本文を持つ HTTP 200 レスポンスを返します。

エラー

CodeValidationException

ユーザー指定のアプリケーションコード (クエリ) が無効です。これは単純な構文エラーである可能性があります。

HTTP ステータスコード: 400

ConcurrentModificationException

アプリケーションへの同時変更の結果としてスローされる例外。例えば、2 人の個人が同じアプリケーションを同時に編集しようとしています。

HTTP ステータスコード: 400

InvalidArgumentException

指定された入力パラメータ値が無効です。

HTTP ステータスコード: 400

ResourceInUseException

このオペレーションではアプリケーションを使用できません。

HTTP ステータスコード: 400

ResourceNotFoundException

指定されたアプリケーションが見つかりません。

HTTP ステータスコード: 400

UnsupportedOperationException

指定されたパラメータがサポートされていないか、指定されたリソースがこのオペレーションに対して有効でないため、リクエストは拒否されました。

HTTP ステータスコード: 400

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

AddApplicationInputProcessingConfiguration

> **Note**
>
> このドキュメントの対象は、SQL アプリケーションのみがサポートされる Amazon Kinesis Data Analytics API のバージョン 1 です。バージョン 2 の API では、SQL および Java アプリケーションがサポートされます。バージョン 2 の詳細については、Amazon Kinesis Data Analytics API V2 のドキュメントを参照してください。

InputProcessingConfiguration をアプリケーションに追加します。入力プロセッサは、アプリケーションの SQL コードが実行される前に、入力ストリーム上のレコードを処理します。現在、利用可能な唯一の入力プロセッサは AWS Lambda です。

リクエストの構文

```
{
   "ApplicationName": "string",
   "CurrentApplicationVersionId": number,
   "InputId": "string",
   "InputProcessingConfiguration": {
      "InputLambdaProcessor": {
         "ResourceARN": "string",
         "RoleARN": "string"
      }
   }
}
```

リクエストパラメーター

リクエストは以下のデータを JSON 形式で受け入れます。

ApplicationName

入力処理設定を追加するアプリケーションの名前。

タイプ: 文字列

長さの制限: 最小長は 1 です。最大長は 128 です。

パターン: `[a-zA-Z0-9_.-]+`

必須: はい

CurrentApplicationVersionId

入力処理設定を追加するアプリケーションのバージョン。DescribeApplication オペレーションを使用して、アプリケーションの現在のバージョンを見つけることができます。指定したバージョンが現在のバージョンでない場合は、`ConcurrentModificationException` が返されます。

型: 長整数

有効範囲: 最小値は 1 です。最大値は 999999999 です。

必須: はい

InputId

入力処理設定を追加する入力設定の ID。DescribeApplication オペレーションを使用して、アプリケーションの入力 ID のリストを取得できます。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 50 です。

パターン: `[a-zA-Z0-9_.-]+`

必須: はい

InputProcessingConfiguration

入力処理設定をアプリケーションに追加します。

型: InputProcessingConfiguration オブジェクト

必須: はい

レスポンス要素

アクションが成功した場合、サービスは空の HTTP 本文を持つ HTTP 200 レスポンスを返します。

エラー

ConcurrentModificationException

アプリケーションへの同時変更の結果としてスローされる例外。例えば、2 人の個人が同じアプリケーションを同時に編集しようとしています。

HTTP ステータスコード: 400

InvalidArgumentException

指定された入力パラメータ値が無効です。

HTTP ステータスコード: 400

ResourceInUseException

このオペレーションではアプリケーションを使用できません。

HTTP ステータスコード: 400

ResourceNotFoundException

指定されたアプリケーションが見つかりません。

HTTP ステータスコード: 400

UnsupportedOperationException

指定されたパラメータがサポートされていないか、指定されたリソースがこのオペレーションに対して有効でないため、リクエストは拒否されました。

HTTP ステータスコード: 400

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python

- AWS SDK for Ruby V3

AddApplicationOutput

> **Note**
>
> このドキュメントの対象は、SQL アプリケーションのみがサポートされる Amazon Kinesis Data Analytics API のバージョン 1 です。バージョン 2 の API では、SQL および Java アプリケーションがサポートされます。バージョン 2 の詳細については、Amazon Kinesis Data Analytics API V2 のドキュメントを参照してください。

外部宛先を Amazon Kinesis Analytics アプリケーションに追加します。

Amazon Kinesis Analytics でアプリケーション内のアプリケーション内ストリームから外部宛先 (Amazon Kinesis ストリーム、Amazon Kinesis Firehose 配信ストリーム、 AWS Lambda 関数など) にデータを配信する場合は、このオペレーションを使用して関連する設定をアプリケーションに追加します。アプリケーションに対して 1 つ以上の出力を設定できます。各出力構成により、アプリケーション内ストリームと外部送信先がマッピングされます。

いずれかの出力構成を使用して、データをアプリケーション内エラーストリームから外部送信先に配信できます。これにより、エラーの分析が可能になります。詳細については、「アプリケーション出力 (宛先) について」を参照してください。

このオペレーションを使用したストリーミングソースの追加などのすべての構成の更新により、アプリケーションの新しいバージョンになります。DescribeApplication オペレーションを使用して、現在のアプリケーションバージョンを見つけることができます。

設定できるアプリケーションの入力数および出力数の制限については、「制限」を参照してください。

このオペレーションには `kinesisanalytics:AddApplicationOutput` アクションを実行するアクセス許可が必要です。

リクエストの構文

```
{
   "ApplicationName": "string",
   "CurrentApplicationVersionId": number,
   "Output": {
      "DestinationSchema": {
```

```
            "RecordFormatType": "string"
        },
        "KinesisFirehoseOutput": {
            "ResourceARN": "string",
            "RoleARN": "string"
        },
        "KinesisStreamsOutput": {
            "ResourceARN": "string",
            "RoleARN": "string"
        },
        "LambdaOutput": {
            "ResourceARN": "string",
            "RoleARN": "string"
        },
        "Name": "string"
    }
}
```

リクエストパラメーター

リクエストは以下のデータを JSON 形式で受け入れます。

ApplicationName

出力構成を追加するアプリケーションの名前。

タイプ: 文字列

長さの制限: 最小長は 1 です。最大長は 128 です。

パターン: `[a-zA-Z0-9_.-]+`

必須: はい

CurrentApplicationVersionId

出力設定を追加するアプリケーションのバージョン。DescribeApplication オペレーションを使用して、アプリケーションの現在のバージョンを見つけることができます。指定したバージョンが現在のバージョンでない場合は、`ConcurrentModificationException` が返されます。

型: 長整数

有効範囲: 最小値は 1 です。最大値は 999999999 です。

必須: はい

Output

それぞれが 1 つの出力構成を記述するオブジェクトの配列です。出力設定では、アプリケーション内ストリームの名前、送信先 (Amazon Kinesis ストリーム、Amazon Kinesis Firehose 配信ストリーム、または AWS Lambda 関数) を指定し、送信先に書き込むときに使用するフォーメーションを記録します。

型: Output オブジェクト

必須: はい

レスポンス要素

アクションが成功した場合、サービスは空の HTTP 本文を持つ HTTP 200 レスポンスを返します。

エラー

ConcurrentModificationException

アプリケーションへの同時変更の結果としてスローされる例外。例えば、2 人の個人が同じアプリケーションを同時に編集しようとしています。

HTTP ステータスコード: 400

InvalidArgumentException

指定された入力パラメータ値が無効です。

HTTP ステータスコード: 400

ResourceInUseException

このオペレーションではアプリケーションを使用できません。

HTTP ステータスコード: 400

ResourceNotFoundException

指定されたアプリケーションが見つかりません。

HTTP ステータスコード: 400

UnsupportedOperationException

指定されたパラメータがサポートされていないか、指定されたリソースがこのオペレーションに対して有効でないため、リクエストは拒否されました。

HTTP ステータスコード: 400

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

AddApplicationReferenceDataSource

> **Note**
>
> このドキュメントの対象は、SQL アプリケーションのみがサポートされる Amazon Kinesis Data Analytics API のバージョン 1 です。バージョン 2 の API では、SQL および Java アプリケーションがサポートされます。バージョン 2 の詳細については、Amazon Kinesis Data Analytics API V2 のドキュメントを参照してください。

既存のアプリケーションにリファレンスデータソースを追加します。

Amazon Kinesis Analytics は、リファレンスデータ (つまり、Amazon S3 オブジェクト) を読み取り、アプリケーション内にアプリケーション内テーブルを作成します。リクエストでは、ソース (S3 バケット名とオブジェクトのキー名)、作成するアプリケーション内テーブルの名前、および Amazon S3 オブジェクトのデータが結果のアプリケーション内テーブルの列にマッピングされる方法を説明する必要なマッピング情報を入力します。

概念については、「アプリケーション入力の設定」を参照してください。アプリケーションに追加できるデータソースの制限については、「制限」を参照してください。

このオペレーションには `kinesisanalytics:AddApplicationOutput` アクションを実行するアクセス許可が必要です。

リクエストの構文

```
{
   "ApplicationName": "string",
   "CurrentApplicationVersionId": number,
   "ReferenceDataSource": {
      "ReferenceSchema": {
         "RecordColumns": [
            {
               "Mapping": "string",
               "Name": "string",
               "SqlType": "string"
            }
         ],
         "RecordEncoding": "string",
         "RecordFormat": {
```

```
            "MappingParameters": {
               "CSVMappingParameters": {
                  "RecordColumnDelimiter": "string",
                  "RecordRowDelimiter": "string"
               },
               "JSONMappingParameters": {
                  "RecordRowPath": "string"
               }
            },
            "RecordFormatType": "string"
         }
      },
      "S3ReferenceDataSource": {
         "BucketARN": "string",
         "FileKey": "string",
         "ReferenceRoleARN": "string"
      },
      "TableName": "string"
   }
}
```

リクエストパラメーター

リクエストは以下のデータを JSON 形式で受け入れます。

ApplicationName

既存のアプリケーションの名前。

タイプ: 文字列

長さの制限: 最小長は 1 です。最大長は 128 です。

パターン: `[a-zA-Z0-9_.-]+`

必須: はい

CurrentApplicationVersionId

参照データソースを追加するアプリケーションのバージョン。DescribeApplication オペレーションを使用して、アプリケーションの現在のバージョンを見つけることができます。指定したバージョンが現在のバージョンでない場合は、`ConcurrentModificationException` が返されます。

型: 長整数

有効範囲: 最小値は 1 です。最大値は 999999999 です。

必須: はい

ReferenceDataSource

リファレンスデータソースは、Amazon S3 バケット内のオブジェクトにすることができます。Amazon Kinesis Analytics は、オブジェクトを読み取り、作成されているアプリケーション内テーブルにデータをコピーします。S3 バケット、オブジェクトのキー名、および作成されているアプリケーション内テーブルを指定します。また、ユーザーに代わって Amazon Kinesis Analytics が S3 バケットからのオブジェクトの読み取りを引き受けることができる必要なアクセス許可を持つ IAM ロールも指定する必要があります。

型: ReferenceDataSource オブジェクト

必須: はい

レスポンス要素

アクションが成功した場合、サービスは空の HTTP 本文を持つ HTTP 200 レスポンスを返します。

エラー

ConcurrentModificationException

アプリケーションへの同時変更の結果としてスローされる例外。例えば、2 人の個人が同じアプリケーションを同時に編集しようとしています。

HTTP ステータスコード: 400

InvalidArgumentException

指定された入力パラメータ値が無効です。

HTTP ステータスコード: 400

ResourceInUseException

このオペレーションではアプリケーションを使用できません。

HTTP ステータスコード: 400

ResourceNotFoundException

指定されたアプリケーションが見つかりません。

HTTP ステータスコード: 400

UnsupportedOperationException

指定されたパラメータがサポートされていないか、指定されたリソースがこのオペレーションに対して有効でないため、リクエストは拒否されました。

HTTP ステータスコード: 400

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

CreateApplication

> **Note**
>
> このドキュメントの対象は、SQL アプリケーションのみがサポートされる Amazon Kinesis Data Analytics API のバージョン 1 です。バージョン 2 の API では、SQL および Java アプリケーションがサポートされます。バージョン 2 の詳細については、Amazon Kinesis Data Analytics API V2 のドキュメントを参照してください。

Amazon Kinesis Analytics アプリケーションを作成します。各アプリケーションには、入力として 1 つのストリーミングソース、入力を処理するアプリケーションコード、Amazon Kinesis Analytics がアプリケーションからの出力データを書き込む宛先を 3 つまで設定できます。概要については、「仕組み」を参照してください。

入力の設定では、ストリーミングソースをアプリケーション内ストリームにマッピングします。アプリケーション内ストリームは常に更新されるテーブルとみなすことができます。マッピングでは、アプリケーション内ストリーム用のスキーマを提供し、アプリケーション内ストリーム内の各データ列をストリーミングソース内のデータ要素にマッピングする必要があります。

アプリケーションコードとは、入力データを読み取って変換し、出力を生成する 1 つ以上の SQL ステートメントです。アプリケーションコードでは、SQL ストリームやポンプなど、複数の SQL アーティファクトを作成できます。

出力設定では、アプリケーションで作成されたアプリケーション内ストリームから最大 3 つの宛先にデータを書き込むようにアプリケーションを設定できます。

ソースストリームからデータを読み取ったり、送信先ストリームにデータを書き込んだりするには、Amazon Kinesis Analytics にアクセス許可が必要です。このアクセス権限は、IAM ロールを作成することによって付与します。このオペレーションには `kinesisanalytics:CreateApplication` アクションを実行するアクセス許可が必要です。

Amazon Kinesis Analytics アプリケーションを作成するための入門演習については、「開始方法」を参照してください。

リクエストの構文

```
{
   "ApplicationCode": "string",
   "ApplicationDescription": "string",
```

```
    "ApplicationName": "string",
    "CloudWatchLoggingOptions": [
       {
          "LogStreamARN": "string",
          "RoleARN": "string"
       }
    ],
    "Inputs": [
       {
          "InputParallelism": {
             "Count": number
          },
          "InputProcessingConfiguration": {
             "InputLambdaProcessor": {
                "ResourceARN": "string",
                "RoleARN": "string"
             }
          },
          "InputSchema": {
             "RecordColumns": [
                {
                   "Mapping": "string",
                   "Name": "string",
                   "SqlType": "string"
                }
             ],
             "RecordEncoding": "string",
             "RecordFormat": {
                "MappingParameters": {
                   "CSVMappingParameters": {
                      "RecordColumnDelimiter": "string",
                      "RecordRowDelimiter": "string"
                   },
                   "JSONMappingParameters": {
                      "RecordRowPath": "string"
                   }
                },
                "RecordFormatType": "string"
             }
          },
          "KinesisFirehoseInput": {
             "ResourceARN": "string",
             "RoleARN": "string"
          },
```

```
            "KinesisStreamsInput": {
               "ResourceARN": "string",
               "RoleARN": "string"
            },
            "NamePrefix": "string"
         }
      ],
      "Outputs": [
         {
            "DestinationSchema": {
               "RecordFormatType": "string"
            },
            "KinesisFirehoseOutput": {
               "ResourceARN": "string",
               "RoleARN": "string"
            },
            "KinesisStreamsOutput": {
               "ResourceARN": "string",
               "RoleARN": "string"
            },
            "LambdaOutput": {
               "ResourceARN": "string",
               "RoleARN": "string"
            },
            "Name": "string"
         }
      ],
      "Tags": [
         {
            "Key": "string",
            "Value": "string"
         }
      ]
   }
```

リクエストパラメーター

リクエストは以下のデータを JSON 形式で受け入れます。

ApplicationCode

入力データを読み取って変換し、出力を生成する 1 つ以上の SQL ステートメント。たとえば、1 つのアプリケーション内ストリームからデータを読み取り、ベンダー別の広告クリック数の移動

平均を生成する SQL ステートメントを記述し、ポンプを使用して、表示された行を別のアプリケーション内ストリームに挿入することができます。一般的なパターンの詳細については、「アプリケーションコード」を参照してください。

このような一連の SQL ステートメントを入力できます。ここでは、1 つのステートメントの出力は次のステートメントの入力として使用できます。アプリケーション内ストリームとポンプを作成して、中間結果を保存します。

アプリケーションコードは、`Outputs` で指定された名前でストリームを作成する必要があります。たとえば、`Outputs` で `ExampleOutputStream1` および `ExampleOutputStream2` という名前の出力ストリームを定義した場合、アプリケーションコードはこれらのストリームを作成する必要があります。

タイプ: 文字列

長さの制約: 最小長は 0 です。最大長は 102400 です。

必須: いいえ

ApplicationDescription

アプリケーションの簡単な説明。

タイプ: 文字列

長さの制約: 最小長は 0 です。最大長は 1,024 です。

必須: いいえ

ApplicationName

Amazon Kinesis Analytics アプリケーションの名前 (例 : `sample-app`)。

タイプ: 文字列

長さの制限: 最小長は 1 です。最大長は 128 です。

パターン: `[a-zA-Z0-9_.-]+`

必須: はい

CloudWatchLoggingOptions

このパラメータでは、CloudWatch ログストリームを設定してアプリケーションの設定エラーをモニタリングします。詳細については、「Amazon CloudWatch Logs の使用」を参照してください。

タイプ: CloudWatchLoggingOption オブジェクトの配列

必須: いいえ

Inputs

このパラメータを使用して、アプリケーション入力を設定します。

単一のストリーミングソースから入力を受信するようにアプリケーションを設定できます。この構成では、このストリーミングソースを作成済みのアプリケーション内ストリームにマッピングします。その後、アプリケーションコードは、テーブルのようにアプリケーション内ストリームのクエリを実行できるようになります (このコードは常に更新されるテーブルと考えることができます)。

ストリーミングソースの場合、その Amazon リソースネーム (ARN) とストリームのデータの形式 (JSON、CSV など) を指定します。ユーザーに代わって Amazon Kinesis Analytics がこのストリームの読み取りを引き受けられるよう IAM ロールを指定する必要もあります。

アプリケーション内ストリームを作成するには、SQL で使用するスキーマ化されたデータに変換するスキーマを指定する必要があります。スキーマでは、アプリケーション内ストリームの列を記録するために、ストリーミングソース内のデータ要素の必要なマッピングを指定します。

タイプ: Input オブジェクトの配列

必須: いいえ

Outputs

アプリケーション内ストリームのいずれかから最大 3 つの宛先にデータを書き込むように、アプリケーション出力を設定できます。

これらの宛先には、Amazon Kinesis ストリーム、Amazon Kinesis Firehose 配信ストリーム、AWS Lambda 送信先、またはこれらの 3 つの任意の組み合わせを使用できます。

設定では、アプリケーション内ストリーム名、送信先ストリームまたは Lambda 関数の Amazon リソースネーム (ARN)、データを書き込むときに使用する形式を指定します。また、Amazon Kinesis Analytics がユーザーに代わって送信先ストリームまたは Lambda 関数に書き込むために引き受けることができる IAM ロールを指定する必要があります。

出力設定では、出力ストリームまたは Lambda 関数 ARN も指定します。ストリームの送信先には、ストリーム内のデータの形式 (JSON、CSV など) を指定します。また、Amazon Kinesis

Analytics がユーザーに代わってストリームまたは Lambda 関数に書き込むために引き受けることができる IAM ロールを指定する必要があります。

タイプ: Output オブジェクトの配列

必須: いいえ

Tags

アプリケーションに割り当てる 1 つ以上のタグのリスト。タグは、アプリケーションを識別するキーと値のペアです。アプリケーションタグの最大数にはシステムタグが含まれることに注意してください。ユーザー定義のアプリケーションタグの最大数は 50 です。詳細については、「タグ付けの使用」を参照してください。

型: Tag オブジェクトの配列

配列メンバー: 最小数は 1 項目です。最大数は 200 項目です。

必須: いいえ

レスポンスの構文

```
{
   "ApplicationSummary": {
      "ApplicationARN": "string",
      "ApplicationName": "string",
      "ApplicationStatus": "string"
   }
}
```

レスポンス要素

アクションが成功すると、サービスは HTTP 200 レスポンスを返します。

サービスから以下のデータが JSON 形式で返されます。

ApplicationSummary

Amazon Kinesis Analytics は、`CreateApplication` リクエストへのレスポンスとしてアプリケーションの Amazon リソースネーム (ARN)、名前、ステータスなど、作成したアプリケーションの概要を記載したレスポンスを返します。

型: ApplicationSummary オブジェクト

エラー

CodeValidationException

ユーザー指定のアプリケーションコード (クエリ) が無効です。これは単純な構文エラーである可能性があります。

HTTP ステータスコード: 400

ConcurrentModificationException

アプリケーションへの同時変更の結果としてスローされる例外。例えば、2 人の個人が同じアプリケーションを同時に編集しようとしています。

HTTP ステータスコード: 400

InvalidArgumentException

指定された入力パラメータ値が無効です。

HTTP ステータスコード: 400

LimitExceededException

許可されているアプリケーションの数を超えました。

HTTP ステータスコード: 400

ResourceInUseException

このオペレーションではアプリケーションを使用できません。

HTTP ステータスコード: 400

TooManyTagsException

多すぎるタグを使用してアプリケーションが作成されたか、またはアプリケーションに追加されたタグが多すぎます。アプリケーションタグの最大数にはシステムタグが含まれることに注意してください。ユーザー定義のアプリケーションタグの最大数は 50 です。

HTTP ステータスコード: 400

UnsupportedOperationException

指定されたパラメータがサポートされていないか、指定されたリソースがこのオペレーションに対して有効でないため、リクエストは拒否されました。

HTTP ステータスコード: 400

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

DeleteApplication

> **Note**
>
> このドキュメントの対象は、SQL アプリケーションのみがサポートされる Amazon Kinesis Data Analytics API のバージョン 1 です。バージョン 2 の API では、SQL および Java アプリケーションがサポートされます。バージョン 2 の詳細については、Amazon Kinesis Data Analytics API V2 のドキュメントを参照してください。

指定されたアプリケーションを削除します。Amazon Kinesis Analytics はアプリケーションの実行を停止し、あらゆるアプリケーションアーティファクト (アプリケーション内ストリーム、参照テーブル、アプリケーションコードなど) を含むアプリケーションを削除します。

このオペレーションには `kinesisanalytics:DeleteApplication` アクションを実行するアクセス許可が必要です。

リクエストの構文

```
{
   "ApplicationName": "string",
   "CreateTimestamp": number
}
```

リクエストパラメーター

リクエストは以下のデータを JSON 形式で受け入れます。

ApplicationName

削除する Amazon Kinesis Analytics アプリケーションの名前。

タイプ: 文字列

長さの制限: 最小長は 1 です。最大長は 128 です。

パターン: `[a-zA-Z0-9_.-]+`

必須: はい

CreateTimestamp

`DescribeApplication` オペレーションを使用して、この値を取得できます。

タイプ: タイムスタンプ

必須: はい

レスポンス要素

アクションが成功した場合、サービスは空の HTTP 本文を持つ HTTP 200 レスポンスを返します。

エラー

ConcurrentModificationException

アプリケーションへの同時変更の結果としてスローされる例外。例えば、2 人の個人が同じアプリケーションを同時に編集しようとしています。

HTTP ステータスコード: 400

ResourceInUseException

このオペレーションではアプリケーションを使用できません。

HTTP ステータスコード: 400

ResourceNotFoundException

指定されたアプリケーションが見つかりません。

HTTP ステータスコード: 400

UnsupportedOperationException

指定されたパラメータがサポートされていないか、指定されたリソースがこのオペレーションに対して有効でないため、リクエストは拒否されました。

HTTP ステータスコード: 400

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

DeleteApplicationCloudWatchLoggingOption

> **Note**
>
> このドキュメントの対象は、SQL アプリケーションのみがサポートされる Amazon Kinesis Data Analytics API のバージョン 1 です。バージョン 2 の API では、SQL および Java アプリケーションがサポートされます。バージョン 2 の詳細については、Amazon Kinesis Data Analytics API V2 のドキュメントを参照してください。

アプリケーションから CloudWatch ログストリームを削除します。Amazon Kinesis Analytics アプリケーションで CloudWatch ログストリームを使用する方法の詳細については、「Amazon CloudWatch Logs の使用」を参照してください。

リクエストの構文

```
{
   "ApplicationName": "string",
   "CloudWatchLoggingOptionId": "string",
   "CurrentApplicationVersionId": number
}
```

リクエストパラメーター

リクエストは以下のデータを JSON 形式で受け入れます。

ApplicationName

Kinesis Analytics アプリケーション名。

タイプ: 文字列

長さの制限: 最小長は 1 です。最大長は 128 です。

パターン: `[a-zA-Z0-9_.-]+`

必須: はい

CloudWatchLoggingOptionId

削除する CloudWatch ログ記録オプションの `CloudWatchLoggingOptionId`。`CloudWatchLoggingOptionId` は DescribeApplication オペレーションを呼び出すことで取得できます。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 50 です。

パターン: `[a-zA-Z0-9_.-]+`

必須: はい

CurrentApplicationVersionId

Kinesis Analytics アプリケーションのバージョン ID。

型: 長整数

有効範囲: 最小値は 1 です。最大値は 999999999 です。

必須: はい

レスポンス要素

アクションが成功した場合、サービスは空の HTTP 本文を持つ HTTP 200 レスポンスを返します。

エラー

ConcurrentModificationException

アプリケーションへの同時変更の結果としてスローされる例外。例えば、2 人の個人が同じアプリケーションを同時に編集しようとしています。

HTTP ステータスコード: 400

InvalidArgumentException

指定された入力パラメータ値が無効です。

HTTP ステータスコード: 400

ResourceInUseException

このオペレーションではアプリケーションを使用できません。

HTTP ステータスコード: 400

ResourceNotFoundException

指定されたアプリケーションが見つかりません。

HTTP ステータスコード: 400

UnsupportedOperationException

指定されたパラメータがサポートされていないか、指定されたリソースがこのオペレーションに対して有効でないため、リクエストは拒否されました。

HTTP ステータスコード: 400

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

DeleteApplicationInputProcessingConfiguration

> **Note**
>
> このドキュメントの対象は、SQL アプリケーションのみがサポートされる Amazon Kinesis Data Analytics API のバージョン 1 です。バージョン 2 の API では、SQL および Java アプリケーションがサポートされます。バージョン 2 の詳細については、Amazon Kinesis Data Analytics API V2 のドキュメントを参照してください。

入力から InputProcessingConfiguration を削除します。

リクエストの構文

```
{
   "ApplicationName": "string",
   "CurrentApplicationVersionId": number,
   "InputId": "string"
}
```

リクエストパラメーター

リクエストは以下のデータを JSON 形式で受け入れます。

ApplicationName

Kinesis Analytics アプリケーション名。

タイプ: 文字列

長さの制限: 最小長は 1 です。最大長は 128 です。

パターン: `[a-zA-Z0-9_.-]+`

必須: はい

CurrentApplicationVersionId

Kinesis Analytics アプリケーションのバージョン ID。

型: 長整数

有効範囲: 最小値は 1 です。最大値は 999999999 です。

必須: はい

InputId

入力処理設定から削除する入力設定の ID。DescribeApplication オペレーションを使用して、アプリケーションの入力 ID のリストを取得できます。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 50 です。

パターン: `[a-zA-Z0-9_.-]+`

必須: はい

レスポンス要素

アクションが成功した場合、サービスは空の HTTP 本文を持つ HTTP 200 レスポンスを返します。

エラー

ConcurrentModificationException

アプリケーションへの同時変更の結果としてスローされる例外。例えば、2 人の個人が同じアプリケーションを同時に編集しようとしています。

HTTP ステータスコード: 400

InvalidArgumentException

指定された入力パラメータ値が無効です。

HTTP ステータスコード: 400

ResourceInUseException

このオペレーションではアプリケーションを使用できません。

HTTP ステータスコード: 400

ResourceNotFoundException

指定されたアプリケーションが見つかりません。

HTTP ステータスコード: 400

UnsupportedOperationException

指定されたパラメータがサポートされていないか、指定されたリソースがこのオペレーションに対して有効でないため、リクエストは拒否されました。

HTTP ステータスコード: 400

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

DeleteApplicationOutput

> **Note**
>
> このドキュメントの対象は、SQL アプリケーションのみがサポートされる Amazon Kinesis Data Analytics API のバージョン 1 です。バージョン 2 の API では、SQL および Java アプリケーションがサポートされます。バージョン 2 の詳細については、Amazon Kinesis Data Analytics API V2 のドキュメントを参照してください。

出力先の設定をアプリケーション設定から削除します。Amazon Kinesis Analytics は、対応するアプリケーション内ストリームから外部出力先にデータを書き込まなくなります。

このオペレーションには `kinesisanalytics:DeleteApplicationOutput` アクションを実行するアクセス許可が必要です。

リクエストの構文

```
{
   "ApplicationName": "string",
   "CurrentApplicationVersionId": number,
   "OutputId": "string"
}
```

リクエストパラメーター

リクエストは以下のデータを JSON 形式で受け入れます。

ApplicationName

Amazon Kinesis Analytics アプリケーション名。

タイプ: 文字列

長さの制限: 最小長は 1 です。最大長は 128 です。

パターン: `[a-zA-Z0-9_.-]+`

必須: はい

CurrentApplicationVersionId

Amazon Kinesis Analytics アプリケーションバージョン。DescribeApplication オペレーションを使用して、アプリケーションの現在のバージョンを見つけることができます。指定したバージョンが現在のバージョンでない場合は、`ConcurrentModificationException` が返されます。

型: 長整数

有効範囲: 最小値は 1 です。最大値は 999999999 です。

必須: はい

OutputId

削除する設定の ID。アプリケーションの作成時、または後から AddApplicationOutput オペレーションを使用してアプリケーションに追加される各出力設定には、一意の ID があります。アプリケーション設定から削除する出力構成を一意に識別するために、ID を指定する必要があります。DescribeApplication オペレーションを使用して特定の `OutputId` を取得できます。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 50 です。

パターン: `[a-zA-Z0-9_.-]+`

必須: はい

レスポンス要素

アクションが成功した場合、サービスは空の HTTP 本文を持つ HTTP 200 レスポンスを返します。

エラー

ConcurrentModificationException

アプリケーションへの同時変更の結果としてスローされる例外。例えば、2 人の個人が同じアプリケーションを同時に編集しようとしています。

HTTP ステータスコード: 400

InvalidArgumentException

指定された入力パラメータ値が無効です。

HTTP ステータスコード: 400

ResourceInUseException

このオペレーションではアプリケーションを使用できません。

HTTP ステータスコード: 400

ResourceNotFoundException

指定されたアプリケーションが見つかりません。

HTTP ステータスコード: 400

UnsupportedOperationException

指定されたパラメータがサポートされていないか、指定されたリソースがこのオペレーションに対して有効でないため、リクエストは拒否されました。

HTTP ステータスコード: 400

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

DeleteApplicationReferenceDataSource

> **Note**
>
> このドキュメントの対象は、SQL アプリケーションのみがサポートされる Amazon Kinesis Data Analytics API のバージョン 1 です。バージョン 2 の API では、SQL および Java アプリケーションがサポートされます。バージョン 2 の詳細については、Amazon Kinesis Data Analytics API V2 のドキュメントを参照してください。

指定したアプリケーションの設定からリファレンスデータソース設定を削除します。

アプリケーションが実行中の場合、Amazon Kinesis Analytics は AddApplicationReferenceDataSource オペレーションを使用して作成したアプリケーション内テーブルをすぐに削除します。

このオペレーションには `kinesisanalytics.DeleteApplicationReferenceDataSource` アクションを実行するアクセス許可が必要です。

リクエストの構文

```
{
   "ApplicationName": "string",
   "CurrentApplicationVersionId": number,
   "ReferenceId": "string"
}
```

リクエストパラメーター

リクエストは以下のデータを JSON 形式で受け入れます。

ApplicationName

既存のアプリケーションの名前。

タイプ: 文字列

長さの制限: 最小長は 1 です。最大長は 128 です。

パターン: `[a-zA-Z0-9_.-]+`

必須: はい

CurrentApplicationVersionId

アプリケーションのバージョン。DescribeApplication オペレーションを使用して、アプリケーションの現在のバージョンを見つけることができます。指定したバージョンが現在のバージョンでない場合は、`ConcurrentModificationException` が返されます。

型: 長整数

有効範囲: 最小値は 1 です。最大値は 999999999 です。

必須: はい

ReferenceId

リファレンスデータソースの ID。AddApplicationReferenceDataSource オペレーションを使用してリファレンスデータソースをアプリケーションに追加するときに、Amazon Kinesis Analytics が ID を割り当てます。リファレンス ID を取得するには、DescribeApplication オペレーションを使用します。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 50 です。

パターン: `[a-zA-Z0-9_.-]+`

必須: はい

レスポンス要素

アクションが成功した場合、サービスは空の HTTP 本文を持つ HTTP 200 レスポンスを返します。

エラー

ConcurrentModificationException

アプリケーションへの同時変更の結果としてスローされる例外。例えば、2 人の個人が同じアプリケーションを同時に編集しようとしています。

HTTP ステータスコード: 400

InvalidArgumentException

指定された入力パラメータ値が無効です。

HTTP ステータスコード: 400

ResourceInUseException

このオペレーションではアプリケーションを使用できません。

HTTP ステータスコード: 400

ResourceNotFoundException

指定されたアプリケーションが見つかりません。

HTTP ステータスコード: 400

UnsupportedOperationException

指定されたパラメータがサポートされていないか、指定されたリソースがこのオペレーションに対して有効でないため、リクエストは拒否されました。

HTTP ステータスコード: 400

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

DescribeApplication

> **Note**
>
> このドキュメントの対象は、SQL アプリケーションのみがサポートされる Amazon Kinesis Data Analytics API のバージョン 1 です。バージョン 2 の API では、SQL および Java アプリケーションがサポートされます。バージョン 2 の詳細については、Amazon Kinesis Data Analytics API V2 のドキュメントを参照してください。

特定の Amazon Kinesis Analytics アプリケーションに関する情報を返します。

アカウント内のすべてのアプリケーションのリストを取得するには、 ListApplications オペレーションを使用します。

このオペレーションには `kinesisanalytics:DescribeApplication` アクションを実行するアクセス許可が必要です。`DescribeApplication` を使用して現在のアプリケーションバージョン Id を取得できます。この情報は、`Update` などの他のオペレーションを呼び出すときに必要です。

リクエストの構文

```
{
   "ApplicationName": "string"
}
```

リクエストパラメーター

リクエストは以下のデータを JSON 形式で受け入れます。

ApplicationName

アプリケーションの名前。

タイプ: 文字列

長さの制限: 最小長は 1 です。最大長は 128 です。

パターン: `[a-zA-Z0-9_.-]+`

必須: はい

レスポンスの構文

```
{
   "ApplicationDetail": {
      "ApplicationARN": "string",
      "ApplicationCode": "string",
      "ApplicationDescription": "string",
      "ApplicationName": "string",
      "ApplicationStatus": "string",
      "ApplicationVersionId": number,
      "CloudWatchLoggingOptionDescriptions": [
         {
            "CloudWatchLoggingOptionId": "string",
            "LogStreamARN": "string",
            "RoleARN": "string"
         }
      ],
      "CreateTimestamp": number,
      "InputDescriptions": [
         {
            "InAppStreamNames": [ "string" ],
            "InputId": "string",
            "InputParallelism": {
               "Count": number
            },
            "InputProcessingConfigurationDescription": {
               "InputLambdaProcessorDescription": {
                  "ResourceARN": "string",
                  "RoleARN": "string"
               }
            },
            "InputSchema": {
               "RecordColumns": [
                  {
                     "Mapping": "string",
                     "Name": "string",
                     "SqlType": "string"
                  }
               ],
               "RecordEncoding": "string",
               "RecordFormat": {
                  "MappingParameters": {
                     "CSVMappingParameters": {
```

```
                        "RecordColumnDelimiter": "string",
                        "RecordRowDelimiter": "string"
                    },
                    "JSONMappingParameters": {
                        "RecordRowPath": "string"
                    }
                },
                "RecordFormatType": "string"
            }
        },
        "InputStartingPositionConfiguration": {
            "InputStartingPosition": "string"
        },
        "KinesisFirehoseInputDescription": {
            "ResourceARN": "string",
            "RoleARN": "string"
        },
        "KinesisStreamsInputDescription": {
            "ResourceARN": "string",
            "RoleARN": "string"
        },
        "NamePrefix": "string"
    }
],
"LastUpdateTimestamp": number,
"OutputDescriptions": [
    {
        "DestinationSchema": {
            "RecordFormatType": "string"
        },
        "KinesisFirehoseOutputDescription": {
            "ResourceARN": "string",
            "RoleARN": "string"
        },
        "KinesisStreamsOutputDescription": {
            "ResourceARN": "string",
            "RoleARN": "string"
        },
        "LambdaOutputDescription": {
            "ResourceARN": "string",
            "RoleARN": "string"
        },
        "Name": "string",
        "OutputId": "string"
```

```
            }
        ],
        "ReferenceDataSourceDescriptions": [
            {
                "ReferenceId": "string",
                "ReferenceSchema": {
                    "RecordColumns": [
                        {
                            "Mapping": "string",
                            "Name": "string",
                            "SqlType": "string"
                        }
                    ],
                    "RecordEncoding": "string",
                    "RecordFormat": {
                        "MappingParameters": {
                            "CSVMappingParameters": {
                                "RecordColumnDelimiter": "string",
                                "RecordRowDelimiter": "string"
                            },
                            "JSONMappingParameters": {
                                "RecordRowPath": "string"
                            }
                        },
                        "RecordFormatType": "string"
                    }
                },
                "S3ReferenceDataSourceDescription": {
                    "BucketARN": "string",
                    "FileKey": "string",
                    "ReferenceRoleARN": "string"
                },
                "TableName": "string"
            }
        ]
    }
}
```

レスポンス要素

アクションが成功すると、サービスは HTTP 200 レスポンスを返します。

サービスから以下のデータが JSON 形式で返されます。

ApplicationDetail

アプリケーションの Amazon リソースネーム (ARN)、ステータス、最新バージョン、入出力の設定に関する詳細など、アプリケーションの説明を提供します。

型: ApplicationDetail オブジェクト

エラー

ResourceNotFoundException

指定されたアプリケーションが見つかりません。

HTTP ステータスコード: 400

UnsupportedOperationException

指定されたパラメータがサポートされていないか、指定されたリソースがこのオペレーションに対して有効でないため、リクエストは拒否されました。

HTTP ステータスコード: 400

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

DiscoverInputSchema

> **Note**
>
> このドキュメントの対象は、SQL アプリケーションのみがサポートされる Amazon Kinesis Data Analytics API のバージョン 1 です。バージョン 2 の API では、SQL および Java アプリケーションがサポートされます。バージョン 2 の詳細については、Amazon Kinesis Data Analytics API V2 のドキュメントを参照してください。

指定したストリーミングソース (Amazon Kinesis ストリームまたは Amazon Kinesis Firehose 配信ストリーム) または S3 オブジェクトのサンプルレコードを評価することによって、スキーマを推測します。レスポンスでは、推測されたスキーマと、オペレーションでスキーマを推測するために使用されたサンプルレコードも返されます。

推測されたスキーマは、アプリケーションのストリーミングソースを設定するときに使用できます。概念については、「アプリケーション入力の設定」を参照してください。Amazon Kinesis Analytics コンソールを使用してアプリケーションを作成する場合、コンソールはこのオペレーションを使用してスキーマを推測し、コンソールのユーザーインターフェイスに表示します。

このオペレーションには `kinesisanalytics:DiscoverInputSchema` アクションを実行するアクセス許可が必要です。

リクエストの構文

```
{
   "InputProcessingConfiguration": {
      "InputLambdaProcessor": {
         "ResourceARN": "string",
         "RoleARN": "string"
      }
   },
   "InputStartingPositionConfiguration": {
      "InputStartingPosition": "string"
   },
   "ResourceARN": "string",
   "RoleARN": "string",
   "S3Configuration": {
      "BucketARN": "string",
      "FileKey": "string",
```

```
        "RoleARN": "string"
    }
}
```

リクエストパラメーター

リクエストは以下のデータを JSON 形式で受け入れます。

InputProcessingConfiguration

レコードのスキーマを検出する前に、InputProcessingConfiguration を使用してレコードを前処理します。

型: InputProcessingConfiguration オブジェクト

必須: いいえ

InputStartingPositionConfiguration

Amazon Kinesis Analytics で、指定したストリーミングソースから検出の目的でレコードの読み取りを開始するポイント。

型: InputStartingPositionConfiguration オブジェクト

必須: いいえ

ResourceARN

ストリーミングソースの Amazon リソースネーム (ARN)。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 2,048 です。

パターン: `arn:.*`

必須: いいえ

RoleARN

Amazon Kinesis Analytics がユーザーに代わってストリームにアクセスするために引き受けることができる IAM ロールの ARN。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 2,048 です。

パターン: `arn:.*`

必須: いいえ

S3Configuration

Amazon S3 オブジェクトのデータからスキーマを検出するには、このパラメータを指定します。

型: S3Configuration オブジェクト

必須: いいえ

レスポンスの構文

```
{
   "InputSchema": {
      "RecordColumns": [
         {
            "Mapping": "string",
            "Name": "string",
            "SqlType": "string"
         }
      ],
      "RecordEncoding": "string",
      "RecordFormat": {
         "MappingParameters": {
            "CSVMappingParameters": {
               "RecordColumnDelimiter": "string",
               "RecordRowDelimiter": "string"
            },
            "JSONMappingParameters": {
               "RecordRowPath": "string"
            }
         },
         "RecordFormatType": "string"
      }
   },
   "ParsedInputRecords": [
      [ "string" ]
   ],
   "ProcessedInputRecords": [ "string" ],
```

```
    "RawInputRecords": [ "string" ]
}
```

レスポンス要素

アクションが成功すると、サービスは HTTP 200 レスポンスを返します。

サービスから以下のデータが JSON 形式で返されます。

InputSchema

ストリーミングソースから推測されたスキーマ。ストリーミングソース内のデータ形式、アプリケーション内ストリームで作成可能な対応カラムに対して各データ要素をマッピングする方法を定義します。

型: SourceSchema オブジェクト

ParsedInputRecords

要素の配列。各要素はストリームレコード内の行に対応します (ストリームレコードは複数の行を持つことができます)。

型: 文字列の配列の配列

ProcessedInputRecords

`InputProcessingConfiguration` パラメータで指定されたプロセッサによって変更されたストリームデータ。

型: 文字列の配列

RawInputRecords

スキーマを推測するためにサンプリングされた生のストリームデータ。

型: 文字列の配列

エラー

InvalidArgumentException

指定された入力パラメータ値が無効です。

HTTP ステータスコード: 400

ResourceProvisionedThroughputExceededException

Amazon Kinesis Streams ProvisionedThroughputExceededException が原因で、検出においてストリーミングソースからレコードを取得できませんでした。詳細については、Amazon Kinesis Streams API リファレンスの GetRecords を参照してください。

HTTP ステータスコード: 400

ServiceUnavailableException

サービスは利用できません。戻ってオペレーションを再試行してください。

HTTP ステータスコード: 500

UnableToDetectSchemaException

日付形式が有効ではありません。Amazon Kinesis Analytics は、指定されたストリーミングソースのスキーマを検出できません。

HTTP ステータスコード: 400

UnsupportedOperationException

指定されたパラメータがサポートされていないか、指定されたリソースがこのオペレーションに対して有効でないため、リクエストは拒否されました。

HTTP ステータスコード: 400

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3

- AWS SDK for Python
- AWS SDK for Ruby V3

ListApplications

> **Note**
>
> このドキュメントの対象は、SQL アプリケーションのみがサポートされる Amazon Kinesis Data Analytics API のバージョン 1 です。バージョン 2 の API では、SQL および Java アプリケーションがサポートされます。バージョン 2 の詳細については、Amazon Kinesis Data Analytics API V2 のドキュメントを参照してください。

アカウントの Amazon Kinesis Analytics アプリケーションのリストを返します。レスポンスには、アプリケーションごとにアプリケーション名、Amazon リソースネーム (ARN)、ステータスが含まれます。レスポンスで `HasMoreApplications` 値が true として返された場合は、リクエスト本文に `ExclusiveStartApplicationName` を追加し、この値を前のレスポンスからの最後のアプリケーション名に設定すると、別のリクエストを送信できます。

特定のアプリケーションの詳細情報が必要な場合は、DescribeApplication を使用します。

このオペレーションには `kinesisanalytics:ListApplications` アクションを実行するアクセス許可が必要です。

リクエストの構文

```
{
   "ExclusiveStartApplicationName": "string",
   "Limit": number
}
```

リクエストパラメーター

リクエストは以下のデータを JSON 形式で受け入れます。

ExclusiveStartApplicationName

リストを開始するアプリケーションの名前。ページ割りを使用してリストを取得する場合、最初のリクエストでこのパラメータを指定する必要はありません。ただし、後続のリクエストでは、前のレスポンスの最後のアプリケーション名を追加して、アプリケーションの次のページを取得します。

タイプ: 文字列

長さの制限: 最小長は 1 です。最大長は 128 です。

パターン: `[a-zA-Z0-9_.-]+`

必須: いいえ

Limit

リストするアプリケーションの最大数。

タイプ: 整数

有効範囲: 最小値 は 1 です。最大値は 50 です。

必須: いいえ

レスポンスの構文

```
{
   "ApplicationSummaries": [
      {
         "ApplicationARN": "string",
         "ApplicationName": "string",
         "ApplicationStatus": "string"
      }
   ],
   "HasMoreApplications": boolean
}
```

レスポンス要素

アクションが成功すると、サービスは HTTP 200 レスポンスを返します。

サービスから以下のデータが JSON 形式で返されます。

ApplicationSummaries

`ApplicationSummary` オブジェクトのリスト

型: ApplicationSummary オブジェクトの配列

HasMoreApplications

取得するアプリケーションがさらに存在する場合は true を返します。

型: ブール値

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

ListTagsForResource

アプリケーションに割り当てられたキーと値のタグのリストを取得します。詳細については、「タグ付けの使用」を参照してください。

リクエストの構文

```
{
   "ResourceARN": "string"
}
```

リクエストパラメーター

リクエストは以下のデータを JSON 形式で受け入れます。

ResourceARN

タグを取得するアプリケーションの ARN。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 2,048 です。

パターン: `arn:.*`

必須: はい

レスポンスの構文

```
{
   "Tags": [
      {
         "Key": "string",
         "Value": "string"
      }
   ]
}
```

レスポンス要素

アクションが成功すると、サービスは HTTP 200 レスポンスを返します。

サービスから以下のデータが JSON 形式で返されます。

Tags

アプリケーションに割り当てられたキーと値のタグ。

型: Tag オブジェクトの配列

配列メンバー: 最小数は 1 項目です。最大数は 200 項目です。

エラー

ConcurrentModificationException

アプリケーションへの同時変更の結果としてスローされる例外。例えば、2 人の個人が同じアプリケーションを同時に編集しようとしています。

HTTP ステータスコード: 400

InvalidArgumentException

指定された入力パラメータ値が無効です。

HTTP ステータスコード: 400

ResourceNotFoundException

指定されたアプリケーションが見つかりません。

HTTP ステータスコード: 400

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2

- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

StartApplication

> **Note**
>
> このドキュメントの対象は、SQL アプリケーションのみがサポートされる Amazon Kinesis Data Analytics API のバージョン 1 です。バージョン 2 の API では、SQL および Java アプリケーションがサポートされます。バージョン 2 の詳細については、Amazon Kinesis Data Analytics API V2 のドキュメントを参照してください。

指定された Amazon Kinesis Analytics アプリケーションを起動します。アプリケーションを作成した後、アプリケーションを起動するには、このオペレーションを排他的に呼び出す必要があります。

アプリケーションの起動後、入力データの使用を開始し、処理し、設定された宛先に出力を書き込みます。

アプリケーションを起動するには、アプリケーションのステータスが `READY` でなければなりません。アプリケーションのステータスは、コンソールで取得するか、DescribeApplication オペレーションを使用して取得できます。

アプリケーションを起動した後、StopApplication オペレーションを呼び出して、アプリケーションによる入力の処理を停止できます。

このオペレーションには `kinesisanalytics:StartApplication` アクションを実行するアクセス許可が必要です。

リクエストの構文

```
{
   "ApplicationName": "string",
   "InputConfigurations": [
      {
         "Id": "string",
         "InputStartingPositionConfiguration": {
            "InputStartingPosition": "string"
         }
      }
   ]
}
```

リクエストパラメーター

リクエストは以下のデータを JSON 形式で受け入れます。

ApplicationName

アプリケーションの名前。

タイプ: 文字列

長さの制限: 最小長は 1 です。最大長は 128 です。

パターン: `[a-zA-Z0-9_.-]+`

必須: はい

InputConfigurations

アプリケーションが使用を開始する特定の入力を ID で識別します。Amazon Kinesis Analytics は、入力に関連付けられたストリーミングソースの読み取りを開始します。Amazon Kinesis Analytics が読み取りを開始するストリーミングソースの場所を指定することもできます。

型: InputConfiguration オブジェクトの配列

必須: はい

レスポンス要素

アクションが成功した場合、サービスは空の HTTP 本文を持つ HTTP 200 レスポンスを返します。

エラー

InvalidApplicationConfigurationException

ユーザー指定のアプリケーション設定が無効です。

HTTP ステータスコード: 400

InvalidArgumentException

指定された入力パラメータ値が無効です。

HTTP ステータスコード: 400

ResourceInUseException

このオペレーションではアプリケーションを使用できません。

HTTP ステータスコード: 400

ResourceNotFoundException

指定されたアプリケーションが見つかりません。

HTTP ステータスコード: 400

UnsupportedOperationException

指定されたパラメータがサポートされていないか、指定されたリソースがこのオペレーションに対して有効でないため、リクエストは拒否されました。

HTTP ステータスコード: 400

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

StopApplication

> **Note**
>
> このドキュメントの対象は、SQL アプリケーションのみがサポートされる Amazon Kinesis Data Analytics API のバージョン 1 です。バージョン 2 の API では、SQL および Java アプリケーションがサポートされます。バージョン 2 の詳細については、Amazon Kinesis Data Analytics API V2 のドキュメントを参照してください。

アプリケーションによる入力データの処理を停止します。実行中の状態にある場合にのみ、アプリケーションを停止できます。DescribeApplication オペレーションを使用して、アプリケーション状態を見つけることができます。アプリケーションが停止すると、Amazon Kinesis Analytics は入力からのデータの読み取りを停止し、アプリケーションはデータの処理を停止するため、宛先へ書きこまれる出力はありません。

このオペレーションには `kinesisanalytics:StopApplication` アクションを実行するアクセス許可が必要です。

リクエストの構文

```
{
   "ApplicationName": "string"
}
```

リクエストパラメーター

リクエストは以下のデータを JSON 形式で受け入れます。

ApplicationName

停止する実行中のアプリケーションの名前。

タイプ: 文字列

長さの制限: 最小長は 1 です。最大長は 128 です。

パターン: `[a-zA-Z0-9_.-]+`

必須: はい

レスポンス要素

アクションが成功した場合、サービスは空の HTTP 本文を持つ HTTP 200 レスポンスを返します。

エラー

ResourceInUseException

このオペレーションではアプリケーションを使用できません。

HTTP ステータスコード: 400

ResourceNotFoundException

指定されたアプリケーションが見つかりません。

HTTP ステータスコード: 400

UnsupportedOperationException

指定されたパラメータがサポートされていないか、指定されたリソースがこのオペレーションに対して有効でないため、リクエストは拒否されました。

HTTP ステータスコード: 400

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python

- AWS SDK for Ruby V3

TagResource

Kinesis Analytics アプリケーションに 1 つ以上のキーと値のタグを追加します。アプリケーションタグの最大数にはシステムタグが含まれることに注意してください。ユーザー定義のアプリケーションタグの最大数は 50 です。詳細については、「タグ付けの使用」を参照してください。

リクエストの構文

```
{
   "ResourceARN": "string",
   "Tags": [
      {
         "Key": "string",
         "Value": "string"
      }
   ]
}
```

リクエストパラメーター

リクエストは以下のデータを JSON 形式で受け入れます。

ResourceARN

タグを割り当てるアプリケーションの ARN。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 2,048 です。

パターン: `arn:.*`

必須: はい

Tags

アプリケーションに割り当てるキー値のタグ。

型: Tag オブジェクトの配列

配列メンバー: 最小数は 1 項目です。最大数は 200 項目です。

必須: はい

レスポンス要素

アクションが成功した場合、サービスは空の HTTP 本文を持つ HTTP 200 レスポンスを返します。

エラー

ConcurrentModificationException

アプリケーションへの同時変更の結果としてスローされる例外。例えば、2 人の個人が同じアプリケーションを同時に編集しようとしています。

HTTP ステータスコード: 400

InvalidArgumentException

指定された入力パラメータ値が無効です。

HTTP ステータスコード: 400

ResourceInUseException

このオペレーションではアプリケーションを使用できません。

HTTP ステータスコード: 400

ResourceNotFoundException

指定されたアプリケーションが見つかりません。

HTTP ステータスコード: 400

TooManyTagsException

多すぎるタグを使用してアプリケーションが作成されたか、またはアプリケーションに追加されたタグが多すぎます。アプリケーションタグの最大数にはシステムタグが含まれることに注意してください。ユーザー定義のアプリケーションタグの最大数は 50 です。

HTTP ステータスコード: 400

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス

- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

UntagResource

Kinesis Analytics アプリケーションから 1 つ以上のタグを削除します。詳細については、「タグ付けの使用」を参照してください。

リクエストの構文

```
{
   "ResourceARN": "string",
   "TagKeys": [ "string" ]
}
```

リクエストパラメーター

リクエストは以下のデータを JSON 形式で受け入れます。

ResourceARN

タグを削除する Kinesis Analytics アプリケーションの ARN。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 2,048 です。

パターン: `arn:.*`

必須: はい

TagKeys

指定されたアプリケーションから削除するタグのキーのリスト。

型: 文字列の配列

配列メンバー: 最小数は 1 項目です。最大数は 200 項目です。

長さの制限：最小長 1、最大長は 128 です。

必須: はい

レスポンス要素

アクションが成功した場合、サービスは空の HTTP 本文を持つ HTTP 200 レスポンスを返します。

エラー

ConcurrentModificationException

アプリケーションへの同時変更の結果としてスローされる例外。例えば、2 人の個人が同じアプリケーションを同時に編集しようとしています。

HTTP ステータスコード: 400

InvalidArgumentException

指定された入力パラメータ値が無効です。

HTTP ステータスコード: 400

ResourceInUseException

このオペレーションではアプリケーションを使用できません。

HTTP ステータスコード: 400

ResourceNotFoundException

指定されたアプリケーションが見つかりません。

HTTP ステータスコード: 400

TooManyTagsException

多すぎるタグを使用してアプリケーションが作成されたか、またはアプリケーションに追加されたタグが多すぎます。アプリケーションタグの最大数にはシステムタグが含まれることに注意してください。ユーザー定義のアプリケーションタグの最大数は 50 です。

HTTP ステータスコード: 400

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++

- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

UpdateApplication

> **Note**
>
> このドキュメントの対象は、SQL アプリケーションのみがサポートされる Amazon Kinesis Data Analytics API のバージョン 1 です。バージョン 2 の API では、SQL および Java アプリケーションがサポートされます。バージョン 2 の詳細については、Amazon Kinesis Data Analytics API V2 のドキュメントを参照してください。

既存の Amazon Kinesis Analytics アプリケーションを更新します。この API を使用して、アプリケーションコード、入力設定、および出力設定を更新できます。

Amazon Kinesis Analytics は、アプリケーションが更新されるたびに、`CurrentApplicationVersionId` を更新します。

このオペレーションには `kinesisanalytics:UpdateApplication` アクションに対するアクセス許可が必要です。

リクエストの構文

```
{
   "ApplicationName": "string",
   "ApplicationUpdate": {
      "ApplicationCodeUpdate": "string",
      "CloudWatchLoggingOptionUpdates": [
         {
            "CloudWatchLoggingOptionId": "string",
            "LogStreamARNUpdate": "string",
            "RoleARNUpdate": "string"
         }
      ],
      "InputUpdates": [
         {
            "InputId": "string",
            "InputParallelismUpdate": {
               "CountUpdate": number
            },
            "InputProcessingConfigurationUpdate": {
               "InputLambdaProcessorUpdate": {
                  "ResourceARNUpdate": "string",
                  "RoleARNUpdate": "string"
```

```
            }
        },
        "InputSchemaUpdate": {
            "RecordColumnUpdates": [
                {
                    "Mapping": "string",
                    "Name": "string",
                    "SqlType": "string"
                }
            ],
            "RecordEncodingUpdate": "string",
            "RecordFormatUpdate": {
                "MappingParameters": {
                    "CSVMappingParameters": {
                        "RecordColumnDelimiter": "string",
                        "RecordRowDelimiter": "string"
                    },
                    "JSONMappingParameters": {
                        "RecordRowPath": "string"
                    }
                },
                "RecordFormatType": "string"
            }
        },
        "KinesisFirehoseInputUpdate": {
            "ResourceARNUpdate": "string",
            "RoleARNUpdate": "string"
        },
        "KinesisStreamsInputUpdate": {
            "ResourceARNUpdate": "string",
            "RoleARNUpdate": "string"
        },
        "NamePrefixUpdate": "string"
    }
],
"OutputUpdates": [
    {
        "DestinationSchemaUpdate": {
            "RecordFormatType": "string"
        },
        "KinesisFirehoseOutputUpdate": {
            "ResourceARNUpdate": "string",
            "RoleARNUpdate": "string"
        },
```

```
            "KinesisStreamsOutputUpdate": {
               "ResourceARNUpdate": "string",
               "RoleARNUpdate": "string"
            },
            "LambdaOutputUpdate": {
               "ResourceARNUpdate": "string",
               "RoleARNUpdate": "string"
            },
            "NameUpdate": "string",
            "OutputId": "string"
         }
      ],
      "ReferenceDataSourceUpdates": [
         {
            "ReferenceId": "string",
            "ReferenceSchemaUpdate": {
               "RecordColumns": [
                  {
                     "Mapping": "string",
                     "Name": "string",
                     "SqlType": "string"
                  }
               ],
               "RecordEncoding": "string",
               "RecordFormat": {
                  "MappingParameters": {
                     "CSVMappingParameters": {
                        "RecordColumnDelimiter": "string",
                        "RecordRowDelimiter": "string"
                     },
                     "JSONMappingParameters": {
                        "RecordRowPath": "string"
                     }
                  },
                  "RecordFormatType": "string"
               }
            },
            "S3ReferenceDataSourceUpdate": {
               "BucketARNUpdate": "string",
               "FileKeyUpdate": "string",
               "ReferenceRoleARNUpdate": "string"
            },
            "TableNameUpdate": "string"
         }
```

```
        ]
    },
    "CurrentApplicationVersionId": number
}
```

リクエストパラメーター

リクエストは以下のデータを JSON 形式で受け入れます。

ApplicationName

更新する Amazon Kinesis Analytics アプリケーションの名前。

タイプ: 文字列

長さの制限: 最小長は 1 です。最大長は 128 です。

パターン: `[a-zA-Z0-9_.-]+`

必須: はい

ApplicationUpdate

アプリケーションの更新を記述します。

型: ApplicationUpdate オブジェクト

必須: はい

CurrentApplicationVersionId

現在のアプリケーションバージョン ID。DescribeApplication オペレーションを使用してこの値を取得できます。

型: 長整数

有効範囲: 最小値は 1 です。最大値は 999999999 です。

必須: はい

レスポンス要素

アクションが成功した場合、サービスは空の HTTP 本文を持つ HTTP 200 レスポンスを返します。

エラー

CodeValidationException

ユーザー指定のアプリケーションコード (クエリ) が無効です。これは単純な構文エラーである可能性があります。

HTTP ステータスコード: 400

ConcurrentModificationException

アプリケーションへの同時変更の結果としてスローされる例外。例えば、2 人の個人が同じアプリケーションを同時に編集しようとしています。

HTTP ステータスコード: 400

InvalidArgumentException

指定された入力パラメータ値が無効です。

HTTP ステータスコード: 400

ResourceInUseException

このオペレーションではアプリケーションを使用できません。

HTTP ステータスコード: 400

ResourceNotFoundException

指定されたアプリケーションが見つかりません。

HTTP ステータスコード: 400

UnsupportedOperationException

指定されたパラメータがサポートされていないか、指定されたリソースがこのオペレーションに対して有効でないため、リクエストは拒否されました。

HTTP ステータスコード: 400

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go v2
- AWS SDK for Java V2
- AWS SDK for JavaScript V3
- AWS SDK for Kotlin
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

データ型

以下のデータ型 (タイプ) がサポートされています。

- ApplicationDetail
- ApplicationSummary
- ApplicationUpdate
- CloudWatchLoggingOption
- CloudWatchLoggingOptionDescription
- CloudWatchLoggingOptionUpdate
- CSVMappingParameters
- DestinationSchema
- Input
- InputConfiguration
- InputDescription
- InputLambdaProcessor
- InputLambdaProcessorDescription
- InputLambdaProcessorUpdate
- InputParallelism
- InputParallelismUpdate

- InputProcessingConfiguration
- InputProcessingConfigurationDescription
- InputProcessingConfigurationUpdate
- InputSchemaUpdate
- InputStartingPositionConfiguration
- InputUpdate
- JSONMappingParameters
- KinesisFirehoseInput
- KinesisFirehoseInputDescription
- KinesisFirehoseInputUpdate
- KinesisFirehoseOutput
- KinesisFirehoseOutputDescription
- KinesisFirehoseOutputUpdate
- KinesisStreamsInput
- KinesisStreamsInputDescription
- KinesisStreamsInputUpdate
- KinesisStreamsOutput
- KinesisStreamsOutputDescription
- KinesisStreamsOutputUpdate
- LambdaOutput
- LambdaOutputDescription
- LambdaOutputUpdate
- MappingParameters
- Output
- OutputDescription
- OutputUpdate
- RecordColumn
- RecordFormat
- ReferenceDataSource
- ReferenceDataSourceDescription

- ReferenceDataSourceUpdate
- S3Configuration
- S3ReferenceDataSource
- S3ReferenceDataSourceDescription
- S3ReferenceDataSourceUpdate
- SourceSchema
- Tag

ApplicationDetail

> **Note**
>
> このドキュメントの対象は、SQL アプリケーションのみがサポートされる Amazon Kinesis Data Analytics API のバージョン 1 です。バージョン 2 の API では、SQL および Java アプリケーションがサポートされます。バージョン 2 の詳細については、Amazon Kinesis Data Analytics API V2 のドキュメントを参照してください。

アプリケーションの Amazon リソースネーム (ARN)、ステータス、最新バージョン、入出力の設定など、アプリケーションの説明を提供します。

内容

ApplicationARN

アプリケーションの ARN。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 2,048 です。

Pattern: `arn:.*`

必須: はい

ApplicationName

アプリケーションの名前。

タイプ: 文字列

長さの制限: 最小長は 1 です。最大長は 128 です。

Pattern: `[a-zA-Z0-9_.-]+`

必須: はい

ApplicationStatus

アプリケーションのステータス。

タイプ: 文字列

有効な値: `DELETING | STARTING | STOPPING | READY | RUNNING | UPDATING | AUTOSCALING`

必須: はい

ApplicationVersionId

現在のアプリケーションバージョンを指定します。

型: 長整数

有効範囲: 最小値は 1 です。最大値は 999999999 です。

必須: はい

ApplicationCode

アプリケーション内の任意のアプリケーション内ストリームでデータ分析を実行するために指定したアプリケーションコードを返します。

タイプ: 文字列

長さの制約: 最小長は 0 です。最大長は 102400 です。

必須: いいえ

ApplicationDescription

アプリケーションの説明。

タイプ: 文字列

長さの制約: 最小長は 0 です。最大長は 1,024 です。

必須: いいえ

CloudWatchLoggingOptionDescriptions

アプリケーションメッセージを受信するように設定された CloudWatch ログストリームを記述します。Amazon Kinesis Analytics アプリケーションで CloudWatch ログストリームを使用する方法の詳細については、「Amazon CloudWatch Logs の使用」を参照してください。

型: CloudWatchLoggingOptionDescription オブジェクトの配列

必須: いいえ

CreateTimestamp

アプリケーションバージョンが作成されたときのタイムスタンプ。

型: タイムスタンプ

必須: いいえ

InputDescriptions

アプリケーション入力の設定を記述します。詳細については、「アプリケーション入力の設定」を参照してください。

型: InputDescription オブジェクトの配列

必須: いいえ

LastUpdateTimestamp

アプリケーションが最後に更新されたときのタイムスタンプ。

型: タイムスタンプ

必須: いいえ

OutputDescriptions

アプリケーション出力の設定を記述します。詳細については、「アプリケーション出力の設定」を参照してください。

型: OutputDescription オブジェクトの配列

必須: いいえ

ReferenceDataSourceDescriptions

アプリケーション用に設定されたリファレンスデータソースを記述します。詳細については、「アプリケーション入力の設定」を参照してください。

型: ReferenceDataSourceDescription オブジェクトの配列

必須：いいえ

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

ApplicationSummary

> **Note**
>
> このドキュメントの対象は、SQL アプリケーションのみがサポートされる Amazon Kinesis Data Analytics API のバージョン 1 です。バージョン 2 の API では、SQL および Java アプリケーションがサポートされます。バージョン 2 の詳細については、Amazon Kinesis Data Analytics API V2 のドキュメントを参照してください。

アプリケーションの Amazon リソースネーム (ARN)、名前、ステータスなど、アプリケーションの概要情報を指定します。

内容

ApplicationARN

アプリケーションの ARN。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 2,048 です。

Pattern: `arn:.*`

必須: はい

ApplicationName

アプリケーションの名前。

タイプ: 文字列

長さの制限: 最小長は 1 です。最大長は 128 です。

Pattern: `[a-zA-Z0-9_.-]+`

必須: はい

ApplicationStatus

アプリケーションのステータス。

タイプ: 文字列

有効な値: `DELETING | STARTING | STOPPING | READY | RUNNING | UPDATING | AUTOSCALING`

必須: はい

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

ApplicationUpdate

既存の Amazon Kinesis Analytics アプリケーションに適用する更新を記述します。

内容

ApplicationCodeUpdate

アプリケーションコードの更新を記述します。

タイプ: 文字列

長さの制約: 最小長は 0 です。最大長は 102400 です。

必須: いいえ

CloudWatchLoggingOptionUpdates

アプリケーションの CloudWatch ログ記録オプションの更新を記述します。

型: CloudWatchLoggingOptionUpdate オブジェクトの配列

必須: いいえ

InputUpdates

アプリケーション入力の設定の更新を記述します。

型: InputUpdate オブジェクトの配列

必須: いいえ

OutputUpdates

アプリケーション出力の設定の更新を記述します。

型: OutputUpdate オブジェクトの配列

必須: いいえ

ReferenceDataSourceUpdates

アプリケーションのリファレンスデータソースの更新を記述します。

型: ReferenceDataSourceUpdate オブジェクトの配列

必須：いいえ

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

CloudWatchLoggingOption

ログストリームの Amazon リソースネーム (ARN) やロール ARN など、CloudWatch ログ記録オプションの記述を提供します。

内容

LogStreamARN

アプリケーションのメッセージを受信するための CloudWatch ログの ARN。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 2,048 です。

Pattern: `arn:.*`

必須: はい

RoleARN

アプリケーションメッセージの送信に使用するロールの IAM ARN。注意: CloudWatch にアプリケーションメッセージを書き込むには、使用する IAM ロールで `PutLogEvents` ポリシーアクションが有効になっている必要があります。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 2,048 です。

Pattern: `arn:.*`

必須: はい

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

CloudWatchLoggingOptionDescription

CloudWatch ログ記録オプションの記述。

内容

LogStreamARN

アプリケーションのメッセージを受信するための CloudWatch ログの ARN。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 2,048 です。

Pattern: `arn:.*`

必須: はい

RoleARN

アプリケーションメッセージの送信に使用するロールの IAM ARN。注意: CloudWatch にアプリケーションメッセージを書き込むには、使用する IAM ロールで `PutLogEvents` ポリシーアクションが有効になっている必要があります。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 2,048 です。

Pattern: `arn:.*`

必須: はい

CloudWatchLoggingOptionId

CloudWatch ログ記録オプションの記述の ID。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 50 です。

パターン: `[a-zA-Z0-9_.-]+`

必須: いいえ

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

CloudWatchLoggingOptionUpdate

CloudWatch ログ記録オプションの更新を記述します。

内容

CloudWatchLoggingOptionId

更新する CloudWatch ログ記録オプションの ID

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 50 です。

Pattern: `[a-zA-Z0-9_.-]+`

必須: はい

LogStreamARNUpdate

アプリケーションのメッセージを受信するための CloudWatch ログの ARN。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 2,048 です。

パターン: `arn:.*`

必須: いいえ

RoleARNUpdate

アプリケーションメッセージの送信に使用するロールの IAM ARN。注意: CloudWatch にアプリケーションメッセージを書き込むには、使用する IAM ロールで `PutLogEvents` ポリシーアクションが有効になっている必要があります。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 2,048 です。

パターン: `arn:.*`

必須: いいえ

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

CSVMappingParameters

レコード形式で CSV などの区切り記号が使用されている場合に、追加のマッピング情報を提供します。たとえば、次のサンプルレコードでは CSV 形式を使用しています。レコードでは、行の区切り記号として「\n」、列の区切り文字としてカンマ (「、」) が使用されています。

```
"name1", "address1"

"name2", "address2"
```

内容

RecordColumnDelimiter

列の区切り記号。たとえば、CSV 形式では、カンマ (「,」) は典型的な列の区切り文字です。

タイプ: 文字列

長さの制約: 最小長は 1 です。

必須: はい

RecordRowDelimiter

行の区切り記号。たとえば、CSV 形式では、「\n」は典型的な行の区切り記号です。

タイプ: 文字列

長さの制約: 最小長は 1 です。

必須: はい

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

DestinationSchema

レコードが送信先に書き込まれるときのデータ形式を記述します。詳細については、「アプリケーション出力の設定」を参照してください。

内容

RecordFormatType

出力ストリームのレコードの形式を指定します。

タイプ: 文字列

有効な値: `JSON | CSV`

必須: はい

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

Input

アプリケーション入力を設定する際に、ストリーミングソース、作成されたアプリケーション内ストリーム名、そしてその 2 つの間のマッピングを指定します。詳細については、「アプリケーション入力の設定」を参照してください。

内容

InputSchema

ストリーミングソース内のデータ形式、アプリケーション内ストリームで作成されている、対応するカラムに対して各データ要素をマッピングする方法を記述します。

リファレンスデータソースの形式を説明するためにも使用されます。

型: SourceSchema オブジェクト

必須: はい

NamePrefix

アプリケーション内ストリームを作成するときに使用する名前プレフィックス。たとえば、プレフィックス「MyInApplicationStream」を指定したと仮定します。その後、Amazon Kinesis Analytics は、「MyInApplicationStream_001」、「MyInApplicationStream_002」などの名前を持つ 1 つ以上の (指定した `InputParallelism` カウントに従って) アプリケーション内ストリームを作成します。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 32 です。

必須: はい

InputParallelism

作成するアプリケーション内ストリームの数を記述します。

ソースからのデータは、これらのアプリケーション内入力ストリームにルーティングされます。

「アプリケーション入力の設定」を参照してください。

型: InputParallelism オブジェクト

必須: いいえ

InputProcessingConfiguration

入力の InputProcessingConfiguration。入力プロセッサは、アプリケーションの SQL コードが実行される前に、ストリームからレコードを受け取るとそのレコードを変換します。現在のところ、使用可能な入力処理構成は InputLambdaProcessor だけです。

型: InputProcessingConfiguration オブジェクト

必須: いいえ

KinesisFirehoseInput

ストリーミングソースが Amazon Kinesis Firehose 配信ストリームの場合は、配信ストリームの ARN と、Amazon Kinesis Analytics がユーザーに代わってストリームにアクセス可能にする IAM ロールを識別します。

注意: `KinesisStreamsInput` または `KinesisFirehoseInput` が必要です。

型: KinesisFirehoseInput オブジェクト

必須: いいえ

KinesisStreamsInput

ストリーミングソースが Amazon Kinesis Firehose ストリームの場合は、ストリームの Amazon リソースネーム (ARN) と、Amazon Kinesis Analytics がユーザーに代わってストリームにアクセス可能にする IAM ロールを識別します。

注意: `KinesisStreamsInput` または `KinesisFirehoseInput` が必要です。

型: KinesisStreamsInput オブジェクト

必須: いいえ

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

InputConfiguration

アプリケーションを起動するときに、この設定を指定します。この設定では、入力ソースと、アプリケーションでレコードの処理を開始する入力ソース内のポイントを識別します。

内容

Id

入力ソース ID。この ID は DescribeApplication オペレーションを呼び出すことで取得できます。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 50 です。

Pattern: `[a-zA-Z0-9_.-]+`

必須: はい

InputStartingPositionConfiguration

ストリーミングソースからのレコードの処理をアプリケーションに開始させるポイント。

型: InputStartingPositionConfiguration オブジェクト

必須: はい

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

InputDescription

アプリケーション入力の設定を記述します。詳細については、「アプリケーション入力の設定」を参照してください。

内容

InAppStreamNames

ストリームソースにマップされているアプリケーション内ストリーム名を返します。

型: 文字列の配列

長さの制限：最小長 1、最大長は 32 です。

必須: いいえ

InputId

アプリケーション入力に関連付けられた入力 ID。これは、Amazon Kinesis Analytics がアプリケーションに追加する各入力設定に割り当てる ID です。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 50 です。

パターン: `[a-zA-Z0-9_.-]+`

必須: いいえ

InputParallelism

設定されている並列処理 (ストリーミングソースにマッピングされているアプリケーション内ストリームの数) を記述します。

型: InputParallelism オブジェクト

必須: いいえ

InputProcessingConfigurationDescription

アプリケーションのコードが実行される前に、この入力のレコードで実行されるプリプロセッサの記述。

型: InputProcessingConfigurationDescription オブジェクト

必須: いいえ

InputSchema

ストリーミングソース内のデータ形式、アプリケーション内ストリームで作成されている、対応するカラムに対して各データ要素をマッピングする方法を記述します。

型: SourceSchema オブジェクト

必須: いいえ

InputStartingPositionConfiguration

アプリケーションが入力ストリームから読み取るように設定されているポイント。

型: InputStartingPositionConfiguration オブジェクト

必須: いいえ

KinesisFirehoseInputDescription

Amazon Kinesis Firehose 配信ストリームがストリーミングソースとして設定されている場合は、配信ストリームの ARN と、Amazon Kinesis Analytics がユーザーに代わってストリームにアクセス可能にする IAM ロールを提供します。

型: KinesisFirehoseInputDescription オブジェクト

必須: いいえ

KinesisStreamsInputDescription

Amazon Kinesis ストリームがストリーミングソースとして設定されている場合は、Amazon Kinesis ストリームの Amazon リソースネーム (ARN) と、Amazon Kinesis Analytics がユーザーに代わってストリームにアクセス可能にする IAM ロールを提供します。

型: KinesisStreamsInputDescription オブジェクト

必須: いいえ

NamePrefix

アプリケーション内の接頭辞。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 32 です。

必須: いいえ

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

InputLambdaProcessor

ストリーム内のレコードを前処理するために使用される AWS Lambda 関数の Amazon リソースネーム (ARN) と、 AWS Lambda 関数へのアクセスに使用される IAM ロールの ARN を含むオブジェクト。

内容

ResourceARN

ストリーム内のレコードを操作する AWS Lambda 関数の ARN。

> **Note**
> 最新のバージョンよりも前のバージョンの Lambda 関数を指定するには、Lambda 関数のバージョンを Lambda 関数 ARN に含めます。Lambda ARNsARNs AWS」を参照してください。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 2,048 です。

Pattern: `arn:.*`

必須: はい

RoleARN

AWS Lambda 関数へのアクセスに使用される IAM ロールの ARN。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 2,048 です。

Pattern: `arn:.*`

必須: はい

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

InputLambdaProcessorDescription

ストリーム内のレコードを前処理するために使用される AWS Lambda 関数の Amazon リソースネーム (ARN) と、 AWS Lambda 式へのアクセスに使用される IAM ロールの ARN を含むオブジェクト。

内容

ResourceARN

ストリーム内のレコードを前処理するために使用されるAWS Lambda 関数の ARN。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 2,048 です。

パターン: `arn:.*`

必須: いいえ

RoleARN

AWS Lambda 関数へのアクセスに使用される IAM ロールの ARN。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 2,048 です。

パターン: `arn:.*`

必須: いいえ

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

InputLambdaProcessorUpdate

ストリーム内のレコードを前処理するために使用される InputLambdaProcessor の更新を表します。

内容

ResourceARNUpdate

トリーム内のレコードを前処理するために使用される新しい AWS Lambda 関数の Amazon リソースネーム (ARN)。

> **Note**
> 最新のバージョンよりも前のバージョンの Lambda 関数を指定するには、Lambda 関数のバージョンを Lambda 関数 ARN に含めます。Lambda ARNsARNs AWS」を参照してください。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 2,048 です。

パターン: `arn:.*`

必須: いいえ

RoleARNUpdate

AWS Lambda 関数へのアクセスに使用される新しい IAM ロールの ARN。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 2,048 です。

パターン: `arn:.*`

必須: いいえ

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

InputParallelism

特定のストリーミングソース用に作成するアプリケーション内ストリームの数を記述します。並列処理については、「アプリケーション入力の設定」を参照してください。

内容

Count

作成するアプリケーション内ストリームの数。詳細については、「制限」を参照してください。

タイプ: 整数

有効範囲: 最小値は 1 です。最大値は 64 です。

必須: いいえ

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

InputParallelismUpdate

並列処理数の更新を提供します。

内容

CountUpdate

特定のストリーミングソース用に作成するアプリケーション内ストリームの数。

タイプ: 整数

有効範囲: 最小値は 1 です。最大値は 64 です。

必須: いいえ

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

InputProcessingConfiguration

アプリケーションコードによって処理される前に、ストリーム内のレコードを前処理するために使用されるプロセッサの説明をします。現在、利用可能な唯一の入力プロセッサは AWS Lambda です。

内容

InputLambdaProcessor

アプリケーションコードによって処理される前にストリーム内のレコードの事前処理に使用される InputLambdaProcessor。

型: InputLambdaProcessor オブジェクト

必須: はい

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

InputProcessingConfigurationDescription

入力プロセッサに関する設定情報を提供します。現在、利用可能な唯一の入力プロセッサは AWS Lambda です。

内容

InputLambdaProcessorDescription

関連付けられている InputLambdaProcessorDescription に関する設定情報を提供します。

型: InputLambdaProcessorDescription オブジェクト

必須: いいえ

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

InputProcessingConfigurationUpdate

InputProcessingConfiguration の更新を記述します。

内容

InputLambdaProcessorUpdate

InputLambdaProcessor の更新情報を提供します。

型: InputLambdaProcessorUpdate オブジェクト

必須: はい

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

InputSchemaUpdate

アプリケーションの入力スキーマの更新を記述します。

内容

RecordColumnUpdates

`RecordColumn` オブジェクトのリスト。各オブジェクトは、ストリーミングソース要素からアプリケーション内ストリームの対応する列へのマッピングを記述します。

型: RecordColumn オブジェクトの配列

配列メンバー: 最小数は 1 項目です。最大数は 1,000 項目です。

必須: いいえ

RecordEncodingUpdate

ストリーミングソースのレコードのエンコードを指定します。たとえば、UTF-8 です。

タイプ: 文字列

パターン: `UTF-8`

必須: いいえ

RecordFormatUpdate

ストリーミングソースのレコードの形式を指定します。

型: RecordFormat オブジェクト

必須: いいえ

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

InputStartingPositionConfiguration

アプリケーションがストリーミングソースから読み取るポイントを記述します。

内容

InputStartingPosition

ストリームの開始位置。

- `NOW` - ストリーム内の最新レコードの直後から読み取りを開始します。顧客が発行したリクエストのタイムスタンプで開始します。
- `TRIM_HORIZON` - ストリーム内の最後のトリミングされていないレコード (ストリームで利用可能な最も古いレコード) から読み取りを開始します。このオプションは、Amazon Kinesis Firehose 配信ストリームでは使用できません。
- `LAST_STOPPED_POINT` - アプリケーションが最後に読み取りを停止した場所から読み取りを再開します。

タイプ: 文字列

有効な値: `NOW | TRIM_HORIZON | LAST_STOPPED_POINT`

必須: いいえ

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

InputUpdate

特定の入力設定 (アプリケーションの `InputId` で識別) の更新を記述します。

内容

InputId

更新するアプリケーション入力の入力 ID。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 50 です。

Pattern: `[a-zA-Z0-9_.-]+`

必須: はい

InputParallelismUpdate

並列処理の更新 (アプリケーション内ストリームで、特定のストリーミングソースに対して Amazon Kinesis Analytics が作成する識別番号) を記述します。

型: InputParallelismUpdate オブジェクト

必須: いいえ

InputProcessingConfigurationUpdate

入力処理設定の更新を記述します。

型: InputProcessingConfigurationUpdate オブジェクト

必須: いいえ

InputSchemaUpdate

ストリーミングソース上のデータ形式を記述します。また、ストリーミングソース上のレコード要素が、作成されるアプリケーション内ストリームの列にどのようにマッピングされるかを記述します。

型: InputSchemaUpdate オブジェクト

必須: いいえ

KinesisFirehoseInputUpdate

Amazon Kinesis Firehose 配信ストリームが更新対象のストリーミングソースである場合、更新されたストリーム ARN と IAM ロール ARN を提供します。

型: KinesisFirehoseInputUpdate オブジェクト

必須: いいえ

KinesisStreamsInputUpdate

Amazon Kinesis ストリームが更新対象のストリーミングソースである場合、更新されたストリームの Amazon リソースネーム (ARN) と IAM ロール ARN を提供します。

型: KinesisStreamsInputUpdate オブジェクト

必須: いいえ

NamePrefixUpdate

Amazon Kinesis Analytics が特定のストリーミングソースに対して作成するアプリケーション内ストリーム名のプレフィックス。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 32 です。

必須: いいえ

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

JSONMappingParameters

JSON がストリーミングソースのレコード形式である場合、追加のマッピング情報を提供します。

内容

RecordRowPath

レコードが含まれている最上位の親へのパス

タイプ: 文字列

長さの制約: 最小長は 1 です。

必須: はい

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

KinesisFirehoseInput

ストリーミングソースとして Amazon Kinesis Firehose 配信ストリームを識別します。配信ストリームの Amazon リソースネーム (ARN) と、Amazon Kinesis Analytics がユーザーに代わってストリームにアクセス可能にする IAM ロール ARN を指定します。

内容

ResourceARN

入力配信ストリームの ARN

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 2,048 です。

Pattern: `arn:.*`

必須: はい

RoleARN

Amazon Kinesis Analytics がユーザーに代わってストリームにアクセスするために引き受けることができる IAM ロールの ARN。ロールにストリームにアクセスするために必要な権限があることを確認する必要があります。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 2,048 です。

Pattern: `arn:.*`

必須: はい

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2

- AWS SDK for Ruby V3

KinesisFirehoseInputDescription

アプリケーション入力の設定でストリーミングソースとして設定された Amazon Kinesis Firehose 配信ストリームを記述します。

内容

ResourceARN

Amazon Kinesis Firehose 配信ストリームの Amazon リソースネーム (ARN)。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 2,048 です。

パターン: `arn:.*`

必須: いいえ

RoleARN

Amazon Kinesis Analytics がストリームにアクセスするために引き受ける IAM ロールの ARN。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 2,048 です。

パターン: `arn:.*`

必須: いいえ

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

KinesisFirehoseInputUpdate

アプリケーション入力の設定を更新するときに、ストリーミングソースとしての Amazon Kinesis Firehose 配信ストリームに関する情報を提供します。

内容

ResourceARNUpdate

読み取り対象の入力 Amazon Kinesis Firehose 配信ストリームの Amazon リソースネーム (ARN)。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 2,048 です。

パターン: `arn:.*`

必須: いいえ

RoleARNUpdate

Amazon Kinesis Analytics がユーザーに代わってストリームにアクセスするために引き受けることができる IAM ロールの ARN。このロールに必要なアクセス許可を付与する必要があります。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 2,048 です。

パターン: `arn:.*`

必須: いいえ

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

KinesisFirehoseOutput

アプリケーション出力を設定するときに、送信先として Amazon Kinesis Firehose 配信ストリームを識別します。ストリームに Amazon リソースネーム (ARN) と、Amazon Kinesis Analytics がユーザーに代わってストリームに書き込むことを可能にする IAM ロールを指定します。

内容

ResourceARN

書き込み対象の送信先 Amazon Kinesis Firehose 配信ストリームの ARN。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 2,048 です。

Pattern: `arn:.*`

必須: はい

RoleARN

Amazon Kinesis Analytics がユーザーに代わって送信先ストリームに書き込むことを想定できる IAM ロールの ARN。このロールに必要なアクセス許可を付与する必要があります。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 2,048 です。

Pattern: `arn:.*`

必須: はい

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

KinesisFirehoseOutputDescription

アプリケーションの出力について、その送信先として設定された Amazon Kinesis Firehose ストリームを記述します。

内容

ResourceARN

Amazon Kinesis Firehose 配信ストリームの Amazon リソースネーム (ARN)。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 2,048 です。

パターン: `arn:.*`

必須: いいえ

RoleARN

Amazon Kinesis Analytics がストリームにアクセスするために引き受けることができる IAM ロールの ARN。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 2,048 です。

パターン: `arn:.*`

必須: いいえ

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

KinesisFirehoseOutputUpdate

UpdateApplication オペレーションを使用して出力設定を更新するときに、送信先として設定された Amazon Kinesis Firehose 配信ストリームに関する情報を提供します。

内容

ResourceARNUpdate

書き込み対象の Amazon Kinesis Firehose 配信ストリームの Amazon リソースネーム (ARN)。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 2,048 です。

パターン: `arn:.*`

必須: いいえ

RoleARNUpdate

Amazon Kinesis Analytics がユーザーに代わってストリームにアクセスするために引き受けることができる IAM ロールの ARN。このロールに必要なアクセス許可を付与する必要があります。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 2,048 です。

パターン: `arn:.*`

必須: いいえ

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

KinesisStreamsInput

ストリーミングソースとして Amazon Kinesis ストリームを識別します。ストリームの Amazon リソースネーム (ARN) と、Amazon Kinesis Analytics がユーザーに代わってストリームにアクセス可能にする IAM ロール ARN を指定します。

内容

ResourceARN

読み取り対象の入力 Amazon Kinesis ストリームの ARN。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 2,048 です。

Pattern: `arn:.*`

必須: はい

RoleARN

Amazon Kinesis Analytics がユーザーに代わってストリームにアクセスするために引き受けることができる IAM ロールの ARN。このロールに必要なアクセス許可を付与する必要があります。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 2,048 です。

Pattern: `arn:.*`

必須: はい

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

KinesisStreamsInputDescription

アプリケーション入力の設定でストリーミングソースとして設定された Amazon Kinesis ストリームを記述します。

内容

ResourceARN

Amazon Kinesis ストリームの Amazon リソースネーム (ARN)。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 2,048 です。

パターン: `arn:.*`

必須: いいえ

RoleARN

Amazon Kinesis Analytics がストリームにアクセスするために引き受けることができる IAM ロールの ARN。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 2,048 です。

パターン: `arn:.*`

必須: いいえ

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

KinesisStreamsInputUpdate

アプリケーション入力の設定を更新するときに、ストリーミングソースとして Amazon Kinesis ストリームに関する情報を提供します。

内容

ResourceARNUpdate

読み取り対象の入力 Amazon Kinesis ストリームの Amazon リソースネーム (ARN)。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 2,048 です。

パターン: `arn:.*`

必須: いいえ

RoleARNUpdate

Amazon Kinesis Analytics がユーザーに代わってストリームにアクセスするために引き受けることができる IAM ロールの ARN。このロールに必要なアクセス許可を付与する必要があります。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 2,048 です。

パターン: `arn:.*`

必須: いいえ

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

KinesisStreamsOutput

アプリケーションの出力を設定するときに、送信先として Amazon Kinesis ストリームを識別します。ストリームに Amazon リソースネーム (ARN) と、Amazon Kinesis Analytics がユーザーに代わってストリームに書き込むために使用できる IAM ロール ARN を指定します。

内容

ResourceARN

書き込み対象の送信先 Amazon Kinesis ストリーム 配信ストリームの ARN。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 2,048 です。

Pattern: `arn:.*`

必須: はい

RoleARN

Amazon Kinesis Analytics がユーザーに代わって送信先ストリームに書き込むことを想定できる IAM ロールの ARN。このロールに必要なアクセス許可を付与する必要があります。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 2,048 です。

Pattern: `arn:.*`

必須: はい

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

KinesisStreamsOutputDescription

アプリケーションの出力について、その送信先として設定された Amazon Kinesis ストリームを記述します。

内容

ResourceARN

Amazon Kinesis ストリームの Amazon リソースネーム (ARN)。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 2,048 です。

パターン: `arn:.*`

必須: いいえ

RoleARN

Amazon Kinesis Analytics がストリームにアクセスするために引き受けることができる IAM ロールの ARN。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 2,048 です。

パターン: `arn:.*`

必須: いいえ

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

KinesisStreamsOutputUpdate

UpdateApplication オペレーションを使用して出力設定を更新するときに、送信先として設定された Amazon Kinesis ストリームに関する情報を提供します。

内容

ResourceARNUpdate

出力を書き込む Amazon Kinesis ストリームの Amazon リソースネーム (ARN)。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 2,048 です。

パターン: `arn:.*`

必須: いいえ

RoleARNUpdate

Amazon Kinesis Analytics がユーザーに代わってストリームにアクセスするために引き受けることができる IAM ロールの ARN。このロールに必要なアクセス許可を付与する必要があります。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 2,048 です。

パターン: `arn:.*`

必須: いいえ

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

LambdaOutput

アプリケーション出力を設定するときに、Lambda AWS 関数を送信先として識別します。関数に Amazon リソースネーム (ARN) と、Amazon Kinesis Analytics がユーザーに代わって関数に書き込むために使用できる IAM ロール ARN を指定します。

内容

ResourceARN

書き込み対象送信先 Lambda 関数の Amazon リソースネーム (ARN)。

> **Note**
> 最新のバージョンよりも前のバージョンの Lambda 関数を指定するには、Lambda 関数のバージョンを Lambda 関数 ARN に含めます。Lambda ARNsARNs AWS」を参照してください。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 2,048 です。

Pattern: `arn:.*`

必須: はい

RoleARN

Amazon Kinesis Analytics がユーザーに代わって送信先関数への書き込みを引き受けることができる IAM ロールの ARN。このロールに必要なアクセス許可を付与する必要があります。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 2,048 です。

Pattern: `arn:.*`

必須: はい

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

LambdaOutputDescription

アプリケーション出力の場合、送信先として設定された AWS Lambda 関数を記述します。

内容

ResourceARN

送信先 Lambda 関数の Amazon リソースネーム (ARN)。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 2,048 です。

パターン: `arn:.*`

必須: いいえ

RoleARN

Amazon Kinesis Analytics が送信先関数への書き込みを引き受けることができる IAM ロールの ARN。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 2,048 です。

パターン: `arn:.*`

必須: いいえ

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

LambdaOutputUpdate

UpdateApplication オペレーションを使用して出力設定を更新する場合、 は送信先として設定された AWS Lambda 関数に関する情報を提供します。

内容

ResourceARNUpdate

送信先 Lambda 関数の Amazon リソースネーム (ARN)。

> **Note**
> 最新のバージョンよりも前のバージョンの Lambda 関数を指定するには、Lambda 関数のバージョンを Lambda 関数 ARN に含めます。Lambda ARNsARNs AWS」を参照してください。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 2,048 です。

パターン: `arn:.*`

必須: いいえ

RoleARNUpdate

Amazon Kinesis Analytics がユーザーに代わって送信先関数への書き込みを引き受けることができる IAM ロールの ARN。このロールに必要なアクセス許可を付与する必要があります。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 2,048 です。

パターン: `arn:.*`

必須: いいえ

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

MappingParameters

アプリケーションを作成または更新する際にアプリケーション入力を設定する場合、レコード形式 (JSON、CSV、または何らかの区切り文字によって区切られたレコードフィールド) に固有の追加マッピング情報をストリーミングソースに提供します。

内容

CSVMappingParameters

レコード形式が区切り文字 (たとえば CSV) を使用する際に追加のマッピング情報を提供します。

型: CSVMappingParameters オブジェクト

必須: いいえ

JSONMappingParameters

JSON がストリーミングソースのレコード形式である場合、追加のマッピング情報を提供します。

型: JSONMappingParameters オブジェクト

必須: いいえ

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

Output

アプリケーション内ストリームを識別するためのアプリケーション出力構成と、アプリケーション内ストリームデータを書き込む送信先を記述します。送信先は、Amazon Kinesis ストリームまたは Amazon Kinesis Firehose 配信ストリームにすることができます。

アプリケーションが書き込める送信先の数の制限およびその他の制限については、制限を参照してください。

内容

DestinationSchema

レコードが送信先に書き込まれるときのデータ形式を記述します。詳細については、「アプリケーション出力の設定」を参照してください。

型: DestinationSchema オブジェクト

必須: はい

Name

アプリケーション内ストリームの名前。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 32 です。

必須: はい

KinesisFirehoseOutput

送信先として Amazon Kinesis Firehose 配信ストリームを識別します。

型: KinesisFirehoseOutput オブジェクト

必須: いいえ

KinesisStreamsOutput

送信先として Amazon Kinesis ストリームを識別します。

型: KinesisStreamsOutput オブジェクト

必須: いいえ

LambdaOutput

AWS Lambda 関数を送信先として識別します。

型: LambdaOutput オブジェクト

必須: いいえ

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

OutputDescription

アプリケーション内ストリーム名と、ストリームデータが書き込まれる送信先を含む、アプリケーション出力設定を記述します。送信先は、Amazon Kinesis ストリームまたは Amazon Kinesis Firehose 配信ストリームにすることができます。

内容

DestinationSchema

送信先へのデータの書き込みに使用されるデータ形式。

型: DestinationSchema オブジェクト

必須: いいえ

KinesisFirehoseOutputDescription

出力が書き込まれる送信先として設定された Amazon Kinesis Firehose 配信ストリームを記述します。

型: KinesisFirehoseOutputDescription オブジェクト

必須: いいえ

KinesisStreamsOutputDescription

出力が書き込まれる送信先として設定された Amazon Kinesis ストリームを記述します。

型: KinesisStreamsOutputDescription オブジェクト

必須: いいえ

LambdaOutputDescription

出力が書き込まれる送信先として設定された AWS Lambda 関数について説明します。

型: LambdaOutputDescription オブジェクト

必須: いいえ

Name

出力として設定されているアプリケーション内ストリームの名前。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 32 です。

必須: いいえ

OutputId

出力設定の一意の識別子。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 50 です。

パターン: `[a-zA-Z0-9_.-]+`

必須: いいえ

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

OutputUpdate

`OutputId` によって識別される出力設定の更新を記述します。

内容

OutputId

更新する特定の出力設定を識別します。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 50 です。

Pattern: `[a-zA-Z0-9_.-]+`

必須: はい

DestinationSchemaUpdate

レコードが送信先に書き込まれるときのデータ形式を記述します。詳細については、「アプリケーション出力の設定」を参照してください。

型: DestinationSchema オブジェクト

必須: いいえ

KinesisFirehoseOutputUpdate

出力の送信先として Amazon Kinesis Firehose 配信ストリームを記述します。

型: KinesisFirehoseOutputUpdate オブジェクト

必須: いいえ

KinesisStreamsOutputUpdate

出力の送信先として Amazon Kinesis ストリームを記述します。

型: KinesisStreamsOutputUpdate オブジェクト

必須: いいえ

LambdaOutputUpdate

出力の送信先として AWS Lambda 関数を記述します。

型: LambdaOutputUpdate オブジェクト

必須: いいえ

NameUpdate

この出力設定に別のアプリケーション内ストリームを指定する場合は、このフィールドを使用して新しいアプリケーション内ストリーム名を指定します。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 32 です。

必須: いいえ

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

RecordColumn

ストリーミングソース内の各データ要素からアプリケーション内ストリーム内の対応する列へのマッピングを説明します。

リファレンスデータソースの形式を説明するためにも使用されます。

内容

Name

アプリケーション内の入力ストリームまたはリファレンステーブルで作成される列の名前。

タイプ: 文字列

必須: はい

SqlType

アプリケーション内の入力ストリームまたはリファレンステーブルで作成される列の型。

タイプ: 文字列

長さの制約: 最小長は 1 です。

必須: はい

Mapping

ストリーミング入力またはリファレンスデータソース内のデータ要素への参照。RecordFormatType が JSON の場合、この要素は必須です。

タイプ: 文字列

必須: いいえ

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2

- AWS SDK for Ruby V3

RecordFormat

ストリームのレコードをスキーマ化するために適用されるレコード形式と関連するマッピング情報を記述します。

内容

RecordFormatType

レコード形式の種類。

タイプ: 文字列

有効な値: `JSON | CSV`

必須: はい

MappingParameters

アプリケーションを作成または更新する際にアプリケーション入力を設定する場合、レコード形式 (JSON、CSV、または何らかの区切り文字によって区切られたレコードフィールド) に固有の追加マッピング情報をストリーミングソースに提供します。

型: MappingParameters オブジェクト

必須: いいえ

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

ReferenceDataSource

ソース情報 (S3 バケット名およびオブジェクトキー名)、作成されているアプリケーション内のテーブル名、Amazon S3 オブジェクトのデータ要素をアプリケーション内のテーブルにマッピングするのに必要なスキーマを提供することによりリファレンスデータソースを説明。

内容

ReferenceSchema

ストリーミングソース内のデータ形式、アプリケーション内ストリームで作成されている対応カラムに対して各データ要素をマッピングする方法を記述します。

型: SourceSchema オブジェクト

必須: はい

TableName

作成するアプリケーション内テーブルの名前。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 32 です。

必須: はい

S3ReferenceDataSource

リファレンスデータを含む S3 バケットおよびオブジェクトを識別します。また、Amazon Kinesis Analytics がユーザーに代わってこのオブジェクトを読み取ることができる IAM のロールも識別します。Amazon Kinesis Analytics アプリケーションは 1 回のみ参照データをロードします。データが変更された場合は、アプリケーションへのデータの再ロードをトリガーする `UpdateApplication` オペレーションを呼び出します。

型: S3ReferenceDataSource オブジェクト

必須: いいえ

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

ReferenceDataSourceDescription

アプリケーション用に設定されているリファレンスデータソースを記述します。

内容

ReferenceId

リファレンスデータソースの ID。これは、AddApplicationReferenceDataSource オペレーションを使用してリファレンスデータソースをアプリケーションに追加するときに、Amazon Kinesis Analytics が割り当てる ID です。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 50 です。

Pattern: `[a-zA-Z0-9_.-]+`

必須: はい

S3ReferenceDataSourceDescription

S3 バケット名、リファレンスデータを含むオブジェクトキー名を提供します。また、Amazon S3 オブジェクトを読み取り、アプリケーション内リファレンステーブルにデータを入力するために Amazon Kinesis Analytics が引き受けることができる IAM ロールの Amazon リソースネーム (ARN) を提供します。

型: S3ReferenceDataSourceDescription オブジェクト

必須: はい

TableName

特定のリファレンスデータソース設定によって作成されたアプリケーション内テーブル名。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 32 です。

必須: はい

ReferenceSchema

ストリーミングソース内のデータ形式、アプリケーション内ストリームで作成されている対応カラムに対して各データ要素をマッピングする方法を記述します。

型: SourceSchema オブジェクト

必須: いいえ

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

ReferenceDataSourceUpdate

アプリケーションのリファレンスデータソース設定を更新すると、このオブジェクトは、更新されたすべての値 (ソースバケット名やオブジェクトキー名など)、作成されているアプリケーション内のテーブル名、Amazon S3 オブジェクトのデータを作成済みのアプリケーション内リファレンステーブルにマッピングする更新済みのマッピング情報を提供します。

内容

ReferenceId

更新されるリファレンスデータソースの ID。DescribeApplication オペレーションを使用してこの値を取得できます。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 50 です。

Pattern: `[a-zA-Z0-9_.-]+`

必須: はい

ReferenceSchemaUpdate

ストリーミングソース内のデータ形式、アプリケーション内ストリームで作成されている対応カラムに対して各データ要素をマッピングする方法を記述します。

型: SourceSchema オブジェクト

必須: いいえ

S3ReferenceDataSourceUpdate

Amazon Kinesis Analytics がユーザーに代わって Amazon S3 オブジェクトを読み取り、アプリケーション内リファレンステーブルにデータを入力するために引き受けることができる S3 バケット名、オブジェクトキー名、および IAM ロールを記述します。

型: S3ReferenceDataSourceUpdate オブジェクト

必須: いいえ

TableNameUpdate

この更新によって作成されたアプリケーション内テーブル。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 32 です。

必須: いいえ

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

S3Configuration

S3 バケットの Amazon リソースネーム (ARN)、バケットへのアクセスに使用される IAM ロールの ARN、データを含む Amazon S3 オブジェクトの名前など、Amazon S3 データソースの記述を提供します。

内容

BucketARN

データが含まれている S3 バケットの ARN。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 2,048 です。

Pattern: `arn:.*`

必須: はい

FileKey

データが含まれているオブジェクトの名前。

タイプ: 文字列

長さの制限: 最小長は 1 です。最大長は 1,024 です。

必須: はい

RoleARN

データへのアクセスに使用するロールの IAM ARN。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 2,048 です。

Pattern: `arn:.*`

必須: はい

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

S3ReferenceDataSource

リファレンスデータを含む S3 バケットおよびオブジェクトを識別します。また、Amazon Kinesis Analytics がユーザーに代わってこのオブジェクトを読み取ることができる IAM のロールも識別します。

Amazon Kinesis Analytics アプリケーションは 1 回のみ参照データをロードします。データが変更された場合は、アプリケーションへのデータの再ロードをトリガーする UpdateApplication オペレーションを呼び出します。

内容

BucketARN

S3 バケットの Amazon リソースネーム (ARN)。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 2,048 です。

Pattern: `arn:.*`

必須: はい

FileKey

リファレンスデータを含むオブジェクトキー名。

タイプ: 文字列

長さの制限: 最小長は 1 です。最大長は 1,024 です。

必須: はい

ReferenceRoleARN

ユーザーに代わって、このサービスでデータを読み出すのに使用する IAM ロールの ARN。このロールは、Amazon Kinesis Analytics サービスプリンシパルがこのロールを引き受けることを可能にするオブジェクトおよび信頼ポリシーに対する `s3:GetObject` アクションに対するアクセス許可を持っている必要があります。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 2,048 です。

Pattern: `arn:.*`

必須: はい

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

S3ReferenceDataSourceDescription

リファレンスデータを格納するバケット名とオブジェクトキー名を提供します。

内容

BucketARN

S3 バケットの Amazon リソースネーム (ARN)。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 2,048 です。

Pattern: `arn:.*`

必須: はい

FileKey

Amazon S3 オブジェクトキー名。

タイプ: 文字列

長さの制限: 最小長は 1 です。最大長は 1,024 です。

必須: はい

ReferenceRoleARN

Amazon Kinesis Analytics がユーザーに代わってAmazon S3 オブジェクトを読み取り、アプリケーション内リファレンステーブルにデータを入力するために引き受けることができる IAM ロールの ARN。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 2,048 です。

Pattern: `arn:.*`

必須: はい

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

S3ReferenceDataSourceUpdate

Amazon Kinesis Analytics がユーザーに代わって Amazon S3 オブジェクトを読み取り、アプリケーション内リファレンステーブルにデータを入力するために引き受けることができる S3 バケット名、オブジェクトキー名、および IAM ロールを記述します。

内容

BucketARNUpdate

S3 バケットの Amazon リソースネーム (ARN)。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 2,048 です。

パターン: `arn:.*`

必須: いいえ

FileKeyUpdate

オブジェクトキー名。

タイプ: 文字列

長さの制限: 最小長は 1 です。最大長は 1,024 です。

必須: いいえ

ReferenceRoleARNUpdate

Amazon Kinesis Analytics が Amazon S3 オブジェクトを読み取り、アプリケーション内にデータを入力するために引き受けることができる IAM ロールの ARN。

タイプ: 文字列

長さの制約: 最小長は 1 です。最大長は 2,048 です。

パターン: `arn:.*`

必須: いいえ

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

SourceSchema

ストリーミングソース内のデータ形式、アプリケーション内ストリームで作成されている対応カラムに対して各データ要素をマッピングする方法を記述します。

内容

RecordColumns

`RecordColumn` オブジェクトのリスト。

型: RecordColumn オブジェクトの配列

配列メンバー: 最小数は 1 項目です。最大数は 1000 項目です。

必須: はい

RecordFormat

ストリーミングソースのレコードの形式を指定します。

型: RecordFormat オブジェクト

必須: はい

RecordEncoding

ストリーミングソースのレコードのエンコードを指定します。たとえば、UTF-8 です。

タイプ: 文字列

パターン: `UTF-8`

必須: いいえ

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

Tag

AWS リソースに定義して割り当てることができるキーと値のペア (値はオプション) 。既に存在するタグを指定すると、タグの値はリクエストで指定した値に置き換えられます。アプリケーションタグの最大数にはシステムタグが含まれることに注意してください。ユーザー定義のアプリケーションタグの最大数は 50 です。詳細については、「タグ付けの使用」を参照してください。

内容

Key

キーバリュータグのキー。

タイプ: 文字列

長さの制限: 最小長は 1 です。最大長は 128 です。

必須: はい

Value

キーバリュータグの値。値はオプションです。

タイプ: 文字列

長さの制約: 最小長は 0 です。最大長は 256 です。

必須: いいえ

以下の資料も参照してください。

言語固有の AWS SDKs のいずれかでこの API を使用する方法の詳細については、以下を参照してください。

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

Amazon Kinesis Data Analytics のドキュメント履歴

次の表に、Amazon Kinesis Data Analytics の前回のリリース以後に行われたドキュメントの重要な変更を示します。

- API バージョン: 2015-08-14
- ドキュメントの最終更新日: 2019 年 5 月 8 日

変更	説明	日付
Kinesis Data Analytics アプリケーションのタグ付け	アプリケーションあたりのコストを決定するためや、アクセスをコントロールするためや、ユーザー定義の目的で、アプリケーションのタグ付けを使用します。詳細については、「タグ付けの使用」を参照してください。	2019 年 5 月 8 日
を使用した Kinesis Data Analytics API コールのログ記録 AWS CloudTrail	Amazon Kinesis Data Analytics は AWS CloudTrail、Kinesis Data Analytics のユーザー、ロール、またはサービスによって実行されたアクションを記録する AWS サービスである と統合されています。詳細については、「の使用 AWS CloudTrail」を参照してください。	2019 年 3 月 22 日
フランクフルトリージョンで利用可能な Kinesis Data Analytics	Kinesis Analytics は、欧州 (フランクフルト) リージョンで利用可能になりました。詳細については、「 とエンドポイント: Kinesis Data Analytics」を参照してください。	2018 年 7 月 18 日

変更	説明	日付
コンソールでリファレンスデータを使用する	アプリケーションのリファレンスデータをコンソールで処理できるようになりました。詳細については、「例: Kinesis Data Analytics アプリケーションにリファレンスデータを追加する」を参照してください。	2018 年 7 月 13 日
ウィンドウクエリの例	ウィンドウと集約のサンプルアプリケーション。詳細については、「例: ウィンドウと集約」を参照してください。	2018 年 7 月 9 日
アプリケーションのテスト	アプリケーションスキーマおよびコードへの変更のテストに関するガイダンスです。詳細については、「アプリケーションのテスト」を参照してください。	2018 年 7 月 3 日
データを事前処理するためのアプリケーションの例	REGEX_LOG_PARSE、REGEX_REPLACE、および DateTime 演算子のその他のサンプルコードです。詳細については、「例: データの変換」を参照してください。	2018 年 5 月 18 日
返される行と SQL コードのサイズの引き上げ	返される行のサイズの上限が 512 KB に引き上げられ、アプリケーション内の SQL コードのサイズの上限が 100 KB に引き上げられました。詳細については、「制限」を参照してください。	2018 年 5 月 2 日

変更	説明	日付
AWS Lambda Java および .NET の 関数の例	レコードの前処理およびアプリケーションの宛先用に Lambda 関数を作成するためのコードサンプルです。詳細については、事前処理用の Lambda 関数の作成およびアプリケーションの送信先の Lambda 関数の作成を参照してください。	2018 年 3 月 22 日
新しい HOTSPOTS 関数	データの相対的に高密度なリージョンを見つけ、その情報を返します。詳細については、「例 : ストリーム上のホットスポットの検出 (HOTSPOTS 関数)」を参照してください。	2018 年 3 月 19 日
送信先としての Lambda 関数	Lambda 関数を送信先として分析結果を送信します。詳細については、「出力としての Lambda 関数の使用」を参照してください。	2017 年 12 月 20 日
新しい RANDOM_CUT_FOREST_WITH_EXPLANATION 関数	どのフィールドがデータストリームの異常スコアの原因となっているかについての説明を取得します。詳細については、「例: データ異常の検出と説明の取得 (RANDOM_CUT_FOREST_WITH_EXPLANATION 関数)」を参照してください。	2017 年 11 月 2 日

変更	説明	日付
静的データに対するスキーマ検出	Amazon S3 バケットに保存された静的データに対してスキーマ検出を実行します。詳細については、「静的データに対するスキーマ検出機能の使用」を参照してください。	2017 年 10 月 6 日
Lambda 事前処理機能	分析 AWS Lambda 前に、 を使用して入力ストリームのレコードを前処理します。詳細については、「Lambda 関数を使用したデータの事前処理」を参照してください。	2017 年 10 月 6 日
自動スケーリングアプリケーション	自動スケーリングを使用して、アプリケーションのデータスループットを自動的に向上させます。詳細については、「アプリケーションを自動的にスケーリングしてスループットを向上させる」を参照してください。	2017 年 9 月 13 日
複数のアプリケーション内入力ストリーム	複数のアプリケーション内ストリームを使用してアプリケーションスループットを向上させます。詳細については、「スループットの増加に合わせた入力ストリームの並列処理」を参照してください。	2017 年 6 月 29 日

変更	説明	日付
Kinesis Data Analytics AWS Management Console で を使用するためのガイド	Kinesis Data Analytics コンソールのスキーマエディタと SQL エディタを使用して、推測スキーマと SQL コードを編集します。詳細については、「ステップ 4 (オプション) コンソールを使用したスキーマと SQL コードの編集」を参照してください。	2017 年 4 月 7 日
パブリックリリース	Amazon Kinesis Data Analytics 開発者ガイドのパブリックリリース。	2016 年 8 月 11 日
プレビューリリース	Amazon Kinesis Data Analytics 開発者ガイドのリリースをプレビュー。	2016 年 1 月 29 日

AWS 用語集

最新の AWS 用語については、「 AWS の用語集 リファレンス」のAWS 「 用語集」を参照してください。