aws

Managed Service for Apache Flink デベロッパーガイド

Managed Service for Apache Flink

Managed Service for Apache Flink: Managed Service for Apache Flink デベロッパーガイド

Table of Contents

..... xvi
Apache Flink のマネージドサービスとは何ですか? 1
Apache Flink にはマネージドサービスを、Apache Flink Studio にはマネージドサービスを選択する 1
Apache Flink のマネージドサービスで使用する Apache Flink API を選択する 3
Flink API を選択する 3
ストリーミングデータアプリケーション入門 5
仕組み 6
Apache Flink アプリケーションのプログラミング 6
DataStream API 6
Table API 7
Managed Service for Apache Flink アプリケーションの作成 8
アプリケーションの作成 8
Managed Service for Apache Flink アプリケーションコードの構築 8
Managed Service for Apache Flink アプリケーションの作成 9
Managed Service for Apache Flink アプリケーションの起動 10
Managed Service for Apache Flink アプリケーションの検証 11
Managed Service for Apache Flink アプリケーションのシステムロールバックの有効化 11
アプリケーションの実行 14
アプリケーションとジョブのステータス 14
バッチワークロード 16
アプリケーションリソース 16
Managed Service for Apache Flink アプリケーションリソース 16
Apache Flink アプリケーションリソース 17
DataStream API 18
DataStream API コネクタ 18
DataStream API 演算子 28
DataStream API タイムスタンプ 30
Table API 30
テーブル API コネクタ 31
テーブル API の時間属性 32
Pythonの使用 33
アプリケーションのプログラミング 33
アプリケーションを作成する 36

モニタリング 38
ランタイムプロパティ 39
コンソールでのランタイムプロパティの使用 39
CLI でのランタイムプロパティの使用 40
Managed Service for Apache Flink アプリケーションのランタイムプロパティへのアクセス 43
Apache Flink コネクタ 44
耐障害性 46
チェックポインティングの構成 47
チェックポイント API の例 48
スナップショット 50
スナップショットの自動作成 52
互換性のない状態データを含むスナップショットからの復元 52
スナップショット API の例 54
インプレースバージョンアップグレード 56
Apache Flink のインプレースバージョンアップグレードを使用したアプリケーションのアップグレード 57
アプリケーションを新しい Apache Flink バージョンにアップグレードする 58
ロールバック 64
一般的なベストプラクティスと推奨事項 65
注意事項と既知の問題 65
スケーリング 67
アプリケーション並列処理と ParallelismPerKPU の設定 67
Kinesis 処理ユニットの割り当て 68
アプリケーションの並列処理の更新 69
Auto Scaling 70
タグ付け 72
アプリケーション作成時のタグの追加 73
既存のアプリケーションにタグを追加するか、タグを u 個追加します。 74
アプリケーションのタグを一覧表示する。 74
アプリケーションからタグを削除する。 75
Apache Flink CloudFormation 用マネージドサービスとの併用 75
開始する前に 75
Lambda 関数の記述 75
Lambda ロールの作成 77
Lambda 関数を呼び出す 78

Lambda 関数を呼び出す ... 78
Apache Flink Dashboard ... 84
アプリケーションの Apache Flink ダッシュボードへのアクセス ... 85
リリースバージョン ... 87
Amazon Managed Service for Apache Flink 1.19 ... 88
サポートされている機能 ... 89
Amazon Managed Service for Apache Flink 1.19.1 の変更点 ... 91
コンポーネント ... 92
既知の問題 ... 92
Amazon Managed Service for Apache Flink 1.18 ... 93
Apache Flink 1.15 における Amazon Managed Service for Apache Flink の変更点 ... 95
コンポーネント ... 95
バグ修正 ... 96
既知の問題 ... 96
Amazon Managed Service for Apache Flink 1.15 ... 96
Apache Flink 1.15 における Amazon Managed Service for Apache Flink の変更点 ... 95
コンポーネント ... 95
以前のバージョン ... 100
以前の Apache Flink バージョンでの Apache Flink Kinesis Streams コネクタの使用 ... 100
Apache Flink 1.8.2 を使用したアプリケーションの構築 ... 102
Apache Flink 1.6.2 を使用したアプリケーションの構築 ... 102
アプリケーションのアップグレード ... 104
Apache Flink 1.6.2 および 1.8.2 で利用可能なコネクタ ... 104
入門:Flink 1.13.2 ... 104
入門:Flink 1.11.1 ... 131
開始方法: Flink 1.8.2 - 非推奨 ... 158
開始方法: Flink 1.6.2 - 非推奨 ... 184
レガシーの例 ... 210
Studio のノートブック ... 384
Studio ノートブックのバージョン ... 385
Studio ノートブックの作成 ... 386
ストリーミングデータのインタラクティブな分析 ... 387
Flink インタプリタ ... 388
Apache Flink テーブルの環境変数 ... 389
永続的な状態のアプリケーションとしてデプロイする ... 389
Scala/Python の基準 ... 391

SQL 基準 ... 391
IAM アクセス許可 ... 392
コネクタおよび依存関係 ... 392
デフォルトコネクター ... 392
依存関係とカスタムコネクタの追加 ... 394
ユーザー定義関数 ... 395
ユーザー定義関数に関する考慮事項 ... 396
チェックポイントの有効化 ... 397
チェックポイント間隔の設定 ... 397
チェックポインティングタイプの設定 ... 398
Studio ランタイムのアップグレード ... 398
ノートブックを新しい Studio ランタイムにアップグレードする ... 398
の使用 AWS Glue ... 403
テーブルプロパティ ... 403
サンプルおよびチュートリアル ... 405
Studio ノートブックチュートリアルの作成 ... 406
耐久性ステートチュートリアルによるアプリケーションとしてのデプロイ ... 426
例 ... 430
トラブルシューティング ... 442
動かなくなったアプリケーションの停止 ... 442
インターネットにアクセスできない VPC に永続的な状態のアプリケーションとしてデプロイする ... 442
D eploy-as-app サイズとビルド時間の短縮 ... 443
ジョブのキャンセル ... 445
Apache Flink インタープリタの再起動 ... 446
付録:カスタム IAM ポリシーの作成 ... 447
AWS Glue ... 447
CloudWatch ログ ... 448
Kinesis Streams ... 449
Amazon MSK クラスター ... 451
開始方法 (DataStream API) ... 452
アプリケーションのコンポーネント ... 159
前提条件 ... 453
ステップ 1: アカウントを設定する ... 453
にサインアップする AWS アカウント ... 106
管理アクセスを持つユーザーを作成する ... 106

プログラマチックアクセス権を付与する 455
次のステップ 457
ステップ 2: を設定する AWS CLI 457
次のステップ 459
ステップ 3: アプリケーションを作成する 459
2 つの Amazon Kinesis データストリームを作成する 459
サンプルレコードを入力ストリームに書き込む 460
Apache Flink ストリーミング Java コードをダウンロードして調べる 461
アプリケーションコードをコンパイルする 462
Apache Flink ストリーミング Java コードをアップロードする 463
Managed Service for Apache Flink アプリケーションを作成して実行する 464
次のステップ 476
ステップ 4: クリーンアップする 476
Managed Service for Apache Flink アプリケーションを削除する 476
Kinesis データストリームを削除する 476
Amazon S3 オブジェクトとバケットを削除する 477
IAM リソースを削除する 477
CloudWatch リソースを削除する 477
次のステップ 477
ステップ 5: リソースを使用して次のステップを完了する 478
開始方法 (テーブル API) 479
アプリケーションのコンポーネント 479
前提条件 480
アプリケーションを作成します。 480
依存リソースを作成する 481
samplerRecords を入力ストリームに書き込む 482
Apache Flink ストリーミング Java コードをダウンロードして調べる 483
アプリケーションコードをコンパイルする 485
Apache Flink ストリーミング Java コードをアップロードする 486
Managed Service for Apache Flink アプリケーションを作成して実行する 487
次のステップ 491
クリーンアップ 491
Managed Service for Apache Flink アプリケーションを削除する 492
Amazon MSK クラスターを削除する 492
VPC の削除 492
Amazon S3 オブジェクトとバケットを削除する 492

IAM リソースを削除する 493
CloudWatch リソースを削除する 493
次のステップ 493
次のステップ 493
開始方法 (Python) 494
Pyflink 入門-Apache 用 Python インタープリター | Amazon Web Services 494
アプリケーションのコンポーネント 494
前提条件 495
アプリケーションを作成します。 495
依存リソースを作成する 496
サンプルレコードを入力ストリームに書き込む 497
Apache Flink ストリーミング Python コードを作成して調べる 499
Python アプリへのサードパーティの依存関係の追加 500
Apache Flink ストリーミング Python コードをアップロードする 502
Managed Service for Apache Flink アプリケーションを作成して実行する 503
次のステップ 508
クリーンアップ 508
Managed Service for Apache Flink アプリケーションを削除する 508
Kinesis データストリームを削除する 509
Amazon S3 オブジェクトとバケットを削除する 509
IAM リソースを削除する 509
CloudWatch リソースを削除する 510
開始方法 (スカラー) 511
依存リソースを作成する 511
サンプルレコードを入力ストリームに書き込む 512
アプリケーションコードをダウンロードして調べる 514
アプリケーション・コードをコンパイルしてアップロードするには 515
アプリケーションの作成と実行 (コンソール) 516
アプリケーションの作成 516
アプリケーションを設定する 517
IAM ポリシーを編集する 519
アプリケーションを実行する 521
アプリケーションの停止 521
アプリケーションの作成と実行 (CLI) 521
許可ポリシーを作成する 521
IAM ポリシーを作成する 523

アプリケーションの作成 525
アプリケーションを起動する 526
アプリケーションの停止 379
CloudWatch ログ記録オプションを追加する 380
環境プロパティを更新する 380
アプリケーションコードの更新 381
クリーンアップ 529
Managed Service for Apache Flink アプリケーションを削除する 529
Kinesis データストリームを削除する 530
Amazon S3 オブジェクトとバケットを削除する 530
IAM リソースを削除する 530
CloudWatch リソースを削除する 530
Apache Beams の使用 532
Managed Service for Apache Flink で Apache Beam を使用する 532
ビーム機能 533
Apache Beam を使用してアプリケーションを作成する 533
依存リソースを作成する 533
サンプルレコードを入力ストリームに書き込む 534
アプリケーションコードをダウンロードして調べる 535
アプリケーションコードをコンパイルする 536
Apache Flink ストリーミング Java コードをアップロードする 537
Managed Service for Apache Flink アプリケーションを作成して実行する 537
クリーンアップ 541
次のステップ 542
トレーニングワークショップ、ラボ、ソリューション実装 544
Managed Service for Apache Flink ワークショップ 544
Managed Service for Apache Flink にデプロイする前に、Apache Flink アプリケーションをローカルで開発する 544
Apache Flink Studio 用 Managed Service によるイベント検出 545
AWS ストリーミングデータソリューション 545
クリックストリームラボ 545
カスタムスケーリング 546
CloudWatch ダッシュボード 546
Amazon MSK 546
で Apache Flink ソリューション用 Managed Service の詳細をご覧ください。 GitHub 547
ユーティリティ 548

スナップショットマネージャ 548
ベンチマーキング 548
例 549
Java の例 549
Python の例 551
..... 551
..... 552
Scala の例 553
セキュリティ 554
データ保護 555
データ暗号化 555
Identity and Access Management 556
対象者 556
アイデンティティを使用した認証 557
ポリシーを使用したアクセスの管理 561
Amazon Managed Service for Apache FlinkはどのようにIAMと協力しますか。 563
アイデンティティベースポリシーの例 571
トラブルシューティング 574
サービス間の混乱した代理の防止 576
モニタリング 578
コンプライアンス検証 578
FedRAMP 579
耐障害性 579
ディザスタリカバリ 579
バージョニング 580
インフラストラクチャセキュリティ 580
セキュリティに関するベストプラクティス 581
最小特権アクセスの実装 581
IAM ロールを使用して他の Amazon のサービスにアクセスする 581
依存リソースにサーバー側の暗号化を実装する 582
CloudTrail を使用して API コールをモニタリングする 582
ロギングとモニタリング 583
ログ記録 584
Logs Insights を使用した CloudWatch ログのクエリ 584
モニタリング 584
ログ記録の設定 586

コンソールを使用した CloudWatch ログ記録の設定 587
CLI を使用した CloudWatch ログ記録の設定 587
アプリケーションモニタリングレベル 592
ベストプラクティスのログ記録 593
ログ記録のトラブルシューティング 593
次のステップ 594
ログの分析 594
サンプルクエリを実行する 594
クエリの例 596
Managed Service for Apache Flink のメトリクスとディメンション 598
アプリケーションメトリクス 599
Kinesis Data Streams コネクタメトリクス 628
Amazon MSK コネクタメトリクス 629
Apache Zeppelin メトリクス 631
CloudWatch メトリクスの表示 632
メトリクス 633
カスタムメトリクス 635
アラーム 638
カスタムメッセージの書き込み 651
Log4J を使用して CloudWatch ログに書き込む 651
SLF4J を使用して CloudWatch ログに書き込む 652
の使用 AWS CloudTrail 653
の Managed Service for Apache Flink 情報 CloudTrail 653
Managed Service for Apache Flink ログファイルエントリについて 654
パフォーマンス 657
パフォーマンスのトラブルシューティング 657
データパス 657
パフォーマンストラブルシューティングソリューション 658
パフォーマンスに関するベストプラクティス 660
スケーリングを適切に管理する 660
外部依存リソースの使用状況を監視します。 662
Apache Flink アプリケーションをローカルで実行します。 663
パフォーマンスのモニタリング 663
メトリクスを用いたパフォーマンスモニタリング CloudWatch 663
CloudWatch ログとアラームを使ったパフォーマンスモニタリング 663
クォータ 665

メンテナンス 667
すべてのオペレータに UUID を設定 669
メンテナンスインスタンスを特定する 669
プロダクション・レディネス 670
「アプリケーションの負荷テスト」 670
最大並列度 670
すべてのオペレータに UUID を設定 671
ベストプラクティス 672
フォールトトレランス：チェックポイントとセーブポイント 672
サポートされていないコネクタのバージョン。 673
パフォーマンスと並列処理 673
オペレータごとの並列処理の設定 674
ログ記録 675
コーディング 675
ルート認証情報の管理。 676
シャード/パーティションが少ないソースからの読み取り 676
Studio ノートブックの更新間隔 677
Studio ノートブックの最適なパフォーマンス 677
ウォーターマーク戦略とアイドルシャードがタイムウィンドウに与える影響 677
[概要] 679
例 679
すべてのオペレータに UUID を設定 688
Maven シェードプラグイン ServiceResourceTransformer に を追加する 689
Apache フリンクステートフル関数 690
Apache Flink アプリケーションテンプレート 690
モジュール設定の場所 691
フリンク設定 692
Apache フリンク設定 692
ステートバックエンド 692
Checkpointing 693
セーブポインティング 695
ヒープサイズ 695
バッファデブローティング 696
変更可能な Flink 設定プロパティ 696
耐障害性 696
チェックポイントとステートバックエンド 696

Checkpointing 696
RocksDB ネイティブメトリクス 696
高度な状態バックエンドオプション 698
フルオプション TaskManager 698
メモリ設定 699
RPC/Akka 699
クライアント 699
高度なクラスターオプション 699
ファイルシステムの設定 700
高度なフォールトトレランスオプション 700
メモリ設定 699
メトリクス 700
REST エンドポイントとクライアント向けの拡張オプション 700
高度な SSL セキュリティオプション 700
高度なスケジュールオプション 700
Flink ウェブ UI のアドバンスオプション 700
設定済み Flink プロパティの表示 701
Amazon VPC の使用 702
Amazon VPC の概念 702
VPC アプリケーション権限 703
Amazon VPC にアクセスするためのアクセス権限ポリシー 703
インターネットとサービスへのアクセス 705
関連情報 706
VPC API 706
アプリケーションの作成 706
AddApplicationVpcConfiguration 707
DeleteApplicationVpcConfiguration 708
アプリケーションを更新します。 708
例: VPC を使用する 709
トラブルシューティング 710
開発のトラブルシューティング 710
FlinkRuntimeException：「許可されていない設定変更 (複数可) が検出されました」 711
システムロールバックのベストプラクティス 712
Hudi 設定のベストプラクティス 713
Apache Flink Flame Graphs 713
EFO コネクタ 1.15.2 の認証情報プロバイダーの問題 714

サポートされていない Kinesis コネクタを使用するアプリケーション 714
コンパイルエラー：「プロジェクトの依存関係を解決できませんでした」 717
無効な選択肢：「kinesisanalyticsv2」 717
UpdateApplication アクションがアプリケーションコードをリロードしていない 717
S3 StreamingFileSink FileNotFoundExceptions 718
FlinkKafkaConsumer セーブポイントによる停止に関する問題 720
Flink 1.15 非同期シンクデッドロック 720
Amazon Kinesis Data Streams のリシャーディング中にソース処理が順序どおりに行われない 730
ランタイムのトラブルシューティング 731
トラブルシューティングツール 731
アプリケーションの問題 731
アプリケーションを再起動中 736
スループットが遅すぎる 739
州の無限の成長 740
I/O バウンドオペレーター 741
Kinesis データストリームからのアップストリームまたはソーススロットリング 742
チェックポイント 743
チェックポイントタイムアウト 749
チェックポイント障害 (ビーム) 751
バックプレッシャー 753
データスキュー機能 754
ステートスキュー機能 755
さまざまな地域のリソースとの統合 755
ドキュメント履歴 756
API サンプルコード 763
AddApplicationCloudWatchLoggingOption 764
AddApplicationInput 764
AddApplicationInputProcessingConfiguration 765
AddApplicationOutput 766
AddApplicationReferenceDataSource 766
AddApplicationVpcConfiguration 767
CreateApplication 767
CreateApplicationSnapshot 769
DeleteApplication 769
DeleteApplicationCloudWatchLoggingOption 769

DeleteApplicationInputProcessingConfiguration 769
DeleteApplicationOutput 770
DeleteApplicationReferenceDataSource 770
DeleteApplicationSnapshot 770
DeleteApplicationVpcConfiguration 771
DescribeApplication 771
DescribeApplicationSnapshot 771
DiscoverInputSchema 771
ListApplications 772
ListApplicationSnapshots 772
StartApplication 773
StopApplication 773
UpdateApplication 773
API リファレンス 775
..... 776

Amazon Managed Service for Apache Flink は、以前は Amazon Kinesis Data Analytics for Apache Flink と呼ばれていました。

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。

Apache Flink 向けの Amazon マネージドサービスとは何ですか?

Apache Flink 向けの Amazon マネージドサービスでは、Java、Scala、Python、または SQL 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、ストリーミングソースや静的ソースに対してコードを作成して実行し、時系列分析、リアルタイムダッシュボードのフィード、メトリクスのフィードを行うことができます。

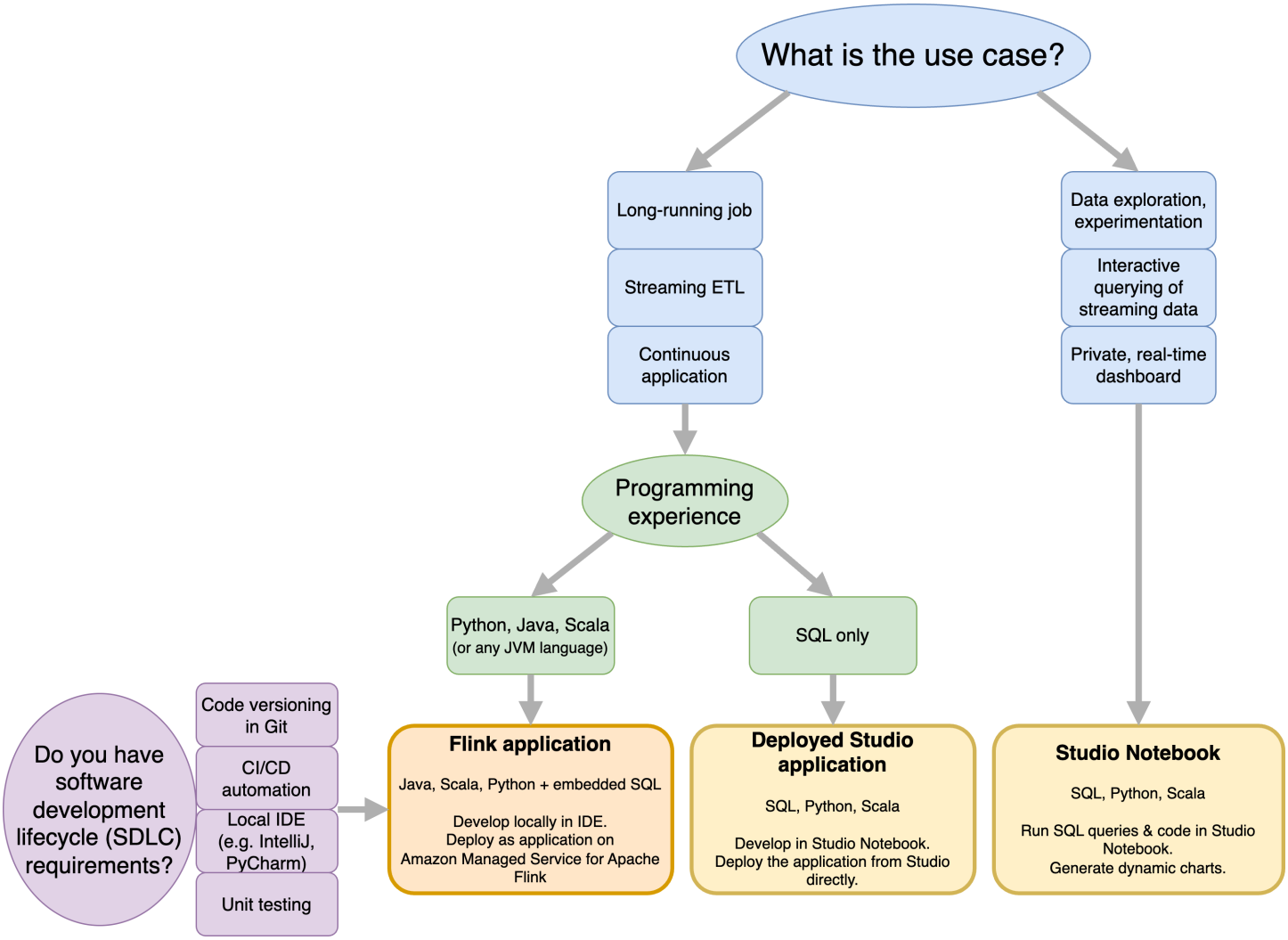
Apache Flink ベースのオープンソースライブラリを使用して、Apache Flink 用マネージドサービスで選択した言語でアプリケーションを構築できます。Apache Flink は、データストリームを処理するための一般的なフレームワークおよびエンジンです。

Apache Flink 用 Managed Serviceは、Apache Flink アプリケーションの基盤となるインフラストラクチャを提供します。コンピュートリソースのプロビジョニング、AZ フェイルオーバーレジリエンス、parallel 計算、自動スケーリング、アプリケーションバックアップ (チェックポイントとスナップショットとして実装) などのコア機能を処理します。ハイレベルの Flink プログラミング特徴 (オペレータ、関数、ソース、シンクなど) は、Flink インフラストラクチャーを自分でホストするときと同じように使用できます。

Apache Flink にはマネージドサービスを、Apache Flink Studio にはマネージドサービスを選択する

Apache Flink 用の Amazon マネージドサービスで Flink ジョブを実行するには、2 つのオプションがあります。Apache Flink 用マネージドサービスでは、任意の IDE と Apache Flink データストリームまたはテーブル API を使用して、Java、Scala、または Python (および組み込み SQL) で Flink アプリケーションを構築できます。Apache Flink Studio のマネージドサービスを使用すると、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、および 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および実行できます。

ユースケースに最も適した方法を選択できます。よくわからない場合は、このセクションで大まかなガイダンスを提供して参考にしてください。



Apache Flink 用の Amazon マネージドサービスと Apache Flink Studio 用の Amazon マネージドサービスのどちらを使用するかを決める前に、ユースケースを検討する必要があります。

ストリーミング ETL や継続的アプリケーションなどのワークロードを引き受ける長時間稼働アプリケーションの運用を計画している場合は、Apache Flink のマネージドサービスの使用を検討する必要があります。これは、選択した IDE で Flink API を使用して Flink アプリケーションを直接作成できるからです。IDE を使用してローカルで開発することで、Git でのコードバージョン管理、CI/CD 自動化、ユニットテストなどのソフトウェア開発ライフサイクル (SDLC) の共通プロセスやツールを活用できるようになります。

アドホックなデータ探索に興味がある場合や、ストリーミングデータをインタラクティブにクエリしたい場合、または専用のリアルタイムダッシュボードを作成したい場合は、Apache Flink Studio の Managed Service for Apache Flink Studio を使用すると、数回クリックするだけでこれらの目標を達成できます。SQL に精通しているユーザーは、長時間稼働するアプリケーションを Studio から直接デプロイすることを検討できます。

> **Note**
>
> Studio ノートブックを長時間実行アプリケーションに昇格させることができます。ただし、Git でのコードバージョン管理や CI/CD 自動化などの SDLC ツールや、ユニットテストなどの手法と統合したい場合は、選択した IDE を使用する Apache Flink のマネージドサービスをお勧めします。

Apache Flink のマネージドサービスで使用する Apache Flink API を選択する

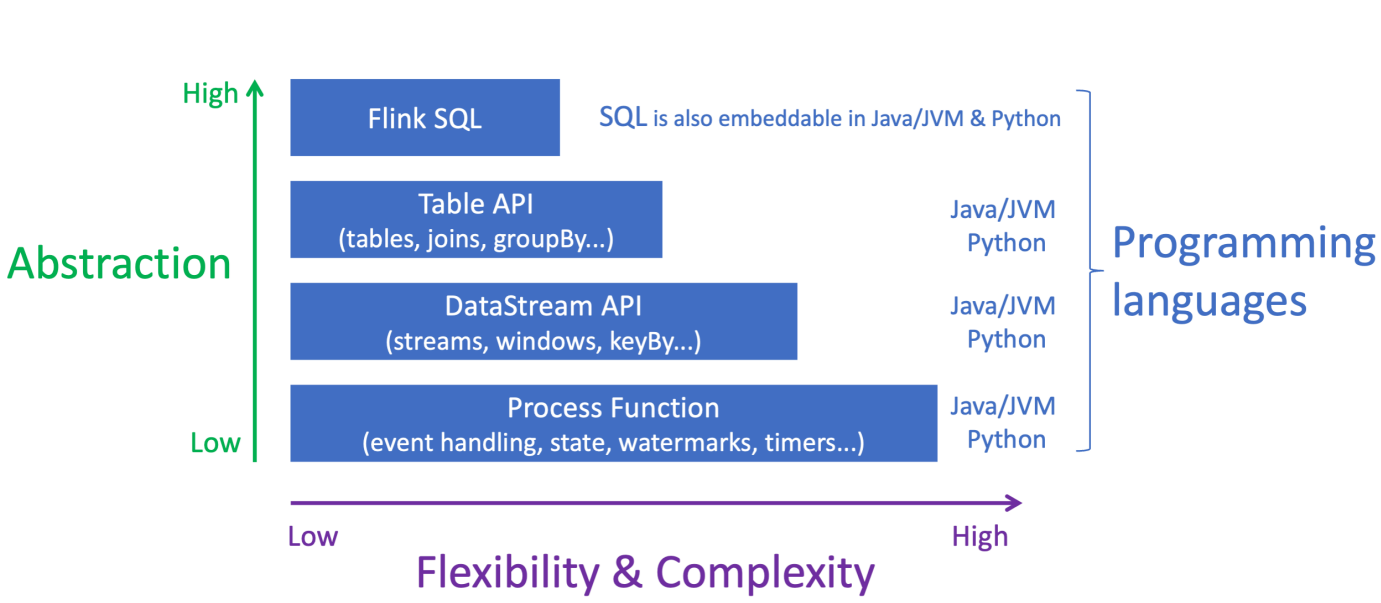
Apache Flink のマネージドサービスでは、任意の IDE で Apache Flink API を使用して Java、Python、および Scala を使用してアプリケーションを構築できます。Flink データストリームとテーブル API を使用してアプリケーションを構築する方法に関するガイダンスは、ドキュメントに記載されています。Flink アプリケーションを作成する言語と、アプリケーションと運用のニーズに最適な API を選択できます。よくわからない場合は、このセクションで大まかなガイダンスを参考にしてください。

Flink API を選択する

Apache Flink API の抽象化レベルはさまざまで、アプリケーションの構築方法に影響する可能性があります。表現力豊かで柔軟性があり、組み合わせてアプリケーションを構築できます。Flink API を 1 つだけ使用する必要はありません。Flink API の詳細については、Apache Flink のドキュメントをご覧ください。

Flink には、Flink SQL、テーブル API、API、プロセス関数という 4 つのレベルの DataStream API 抽象化があり、これらは API と組み合わせて使用されます。 DataStream これらはすべて Apache Flink 用の Amazon マネージドサービスでサポートされています。可能な限り高いレベルの抽象化から始めることをお勧めしますが、Flink の機能の中には Java、Python、または Scala でアプリケーションを作成できる Datastream API でしか使用できないものもあります。次のような場合は、データストリーム API の使用を検討してください。

- 状態をきめ細かく制御する必要がある
- 外部のデータベースやエンドポイントを非同期で (推論などで) 呼び出す機能を活用したい
- カスタムタイマーを使いたい (たとえば、カスタムウィンドウ処理やレイトイベント処理を実装するため)
- 状態をリセットせずにアプリケーションのフローを変更できるようにしたい。



Note

DataStreamAPI を使った言語の選択:

- SQL は、選択したプログラミング言語に関係なく、どの Flink アプリケーションにも埋め込むことができます。
- DataStream API の使用を計画している場合、Python ではすべてのコネクタがサポートされているわけではありません。
- 低レイテンシー/高スループットが必要な場合は、API に関係なく Java/Scala を検討すべきです。
- プロセス関数 API で非同期 IO を使用する予定がある場合は、Java を使用する必要があります。

API の選択は、状態をリセットせずにアプリケーションロジックを進化させる能力にも影響を与える可能性があります。これは、Java と Python の両方の DataStream API でのみ使用できる演算子に UID を設定する機能という特定の機能によって異なります。詳細については、Apache Flink ドキュメントの「すべてのオペレータに UUID を設定する」を参照してください。

ストリーミングデータアプリケーション入門

まず、ストリーミングデータを継続的に読み取って処理する Apache Flink アプリケーション用 Managed Service を作成します。次に、選択した IDE を使用してコードを書き、ライブストリーミングデータでテストします。Apache Flink 用 Managed Service で結果を送信する宛先を設定することもできます。

始める前に、以下のセクションを読んでおくことをお勧めします。

- Managed Service for Apache Flink: 仕組み
- Amazon Managed Service for Apache Flink の開始方法 (DataStream API)

あるいは、Apache Flink Studio ノートブック用のマネージドサービスを作成することから始めることもできます。これにより、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でき、標準 SQL、Python、および 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および実行できます。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。始める前に、以下のセクションを読んでおくことをお勧めします。

- Apache Flink 用 Managed Service で Studio ノートブックを使用する
- Studio ノートブックの作成

Managed Service for Apache Flink: 仕組み

Managed Service for Apache Flinkは、ストリーミングデータを処理するためにApache Flinkアプリケーションを使用できるフルマネージドAmazonサービスです。

Apache Flink アプリケーションのプログラミング

Apache Flink アプリケーションは、Apache Flink フレームワークを使用して作成された Java または Scala アプリケーションです。Apache Flink アプリケーションはローカルで作成してビルドします。

アプリケーションは主に DataStream API またはテーブル API を使用します。他の Apache Flink API も使用できますが、ストリーミングアプリケーションの構築にはあまり使用されません。

2 つの API の特徴は、次のとおりです。

DataStream API

Apache Flink DataStream API プログラミングモデルは、次の 2 つのコンポーネントに基づいています。

- 「データストリーム:」データレコードの連続フローを構造化して表現したものです。
- 「変換演算子:」1 つ以上のデータストリームを入力として受け取り、1 つ以上のデータストリームを出力として生成します。

DataStream API で作成されたアプリケーションは、次の操作を行います。

- データソース (Kinesis ストリームや Amazon MSK トピックなど) からデータを読み取ります。
- フィルタリング、集約、エンリッチメントなどの変換をデータに適用します。
- 変換したデータをデータシンクに書き込みます。

DataStream API を使用するアプリケーションは Java または Scala で記述でき、Kinesis データストリーム、Amazon MSK トピック、またはカスタムソースから読み取ることができます。

アプリケーションは「コネクタ」を使用してデータを処理します。Apache Flink は、次のタイプのコネクタを使用しています。

- 「ソース」:外部データの読み取りに使用されるコネクター。
- 「シンク」:外部への書き込みに使用されるコネクター。
- 「オペレータ」:アプリケーション内のデータを処理するために使用されるコネクタ。

一般的なアプリケーションは、ソース付きの少なくとも 1 つのデータストリーム、1 つ以上のオペレータを含むデータストリーム、および少なくとも 1 つのデータシンクで構成されます。

DataStream API の使用の詳細については、「」を参照してくださいDataStream API。

Table API

Apache Flink Table API プログラミングモデルは、以下のコンポーネントに基づいています。

- 「テーブル環境:」1 つ以上のテーブルを作成およびホストするために使用する基礎データへのインターフェースです。
- 「テーブル:」SQL テーブルまたはビューへのアクセスを提供するオブジェクト。
- 「テーブルソース:」Amazon MSK トピックなどの外部ソースからデータを読み取るために使用されます。
- 「テーブル関数:」データ変換に使用される SQL クエリまたは API 呼び出し。
- 「テーブルシンク:」Amazon S3 バケットなどの外部の場所にデータを書き込むために使用されます。

Table API で作成されたアプリケーションは次のことを行います。

- `Table Source` に接続して `TableEnvironment` を作成します。
- SQL クエリまたはテーブル API 関数を使用して、`TableEnvironment` にテーブルを作成します。
- テーブル API または SQL を使用してテーブルに対してクエリを実行します。
- テーブルファンクションまたは SQL クエリを使用して、クエリの結果に変換を適用します。
- クエリまたは関数の結果を `Table Sink` に書き込みます。

Table API を使用するアプリケーションは Java または Scala で作成でき、API 呼び出しまたは SQL クエリを使用してデータをクエリできます。

テーブル API の使用方法の詳細については、「Table API」 を参照してください。

Managed Service for Apache Flink アプリケーションの作成

Managed Service for Apache Flink は、Apache Flink アプリケーションをホストするための環境を作成し、次の設定を提供する AWS サービスです。

- 「ランタイムプロパティ: 」 アプリケーションに提供できるパラメータ。これらのパラメータは、アプリケーションコードを再コンパイルしなくても変更できます。
- 「耐障害性」: アプリケーションが中断や再起動から回復する方法。
- ロギングとモニタリング: アプリケーションがイベントを CloudWatch ログに記録する方法。
- 「スケーリング」: アプリケーションがコンピューティングリソースをプロビジョニングする方法。

Apache Flink アプリケーション用 Managed Serviceは、コンソールまたは AWS CLIを使用して作成します。Apache Flink 用 Managed Serviceの作成を開始するには、 開始方法 (DataStream API) を参照してください。

Managed Service for Apache Flink アプリケーションの作成

このトピックには、Apache Flink 用 Managed Service の作成に関する情報が含まれています。

このトピックには、次のセクションが含まれています。

- Managed Service for Apache Flink アプリケーションコードの構築
- Managed Service for Apache Flink アプリケーションの作成
- Managed Service for Apache Flink アプリケーションの起動
- Managed Service for Apache Flink アプリケーションの検証
- Managed Service for Apache Flink アプリケーションのシステムロールバックの有効化

Managed Service for Apache Flink アプリケーションコードの構築

このセクションでは、Managed Service for Apache Flink アプリケーションのアプリケーションコードの構築に使用するコンポーネントについて説明します。

アプリケーションコードに対してサポートされている最新バージョンの Apache Flink を使用することをお勧めします。Apache Flink アプリケーション用 Managed Service のアップグレードについては、??? を参照してください。

アプリケーションコードは「Apache Maven」を使用してビルドします。Apache Maven プロジェクトは「pom.xml」ファイルを使用して、使用するコンポーネントのバージョンを指定します。

> **Note**
>
> Apache Flink 用 Managed Service は、最大 512 MB の JAR ファイルをサポートします。これより大きい JAR ファイルを使用すると、アプリケーションは起動に失敗します。

アプリケーションが Scala の任意のバージョンから Java API を使用できるようになっています。選択した Scala 標準ライブラリを Scala アプリケーションにバンドルする必要があります。

「Apache Beam」を使用する Apache Flink アプリケーション用 Managed Service の作成については、 Apache Beams の使用 を参照してください。

アプリケーションの Apache Flink バージョンの指定

Apache Flink Runtime バージョン 1.1.0 以降の Managed Service を使用する場合は、アプリケーションをコンパイルするときにアプリケーションが使用する Apache Flink のバージョンを指定します。-Dflink.version パラメータを使用して Apache Flink のバージョンを指定します。例えば、Apache Flink 1.19.1 を使用している場合は、以下を指定します。

```
mvn package -Dflink.version=1.19.1
```

以前のバージョンの Apache Flink を使用してアプリケーションを構築する方法については、「」を参照してください以前のバージョン。

Managed Service for Apache Flink アプリケーションの作成

アプリケーションコードを作成したら、以下を実行して Managed Service for Apache Flink アプリケーションを作成します。

- 「アプリケーションコードのアップロード:」アプリケーションコードを Amazon S3 バケットにアップロードします。アプリケーションを作成する際は、アプリケーションコードの S3 バケット名とオブジェクト名を指定します。アプリケーションコードのアップロード方法を示すチュートリアルについては、 開始方法 (DataStream API) チュートリアルの the section called “Apache Flink ストリーミング Java コードをアップロードする” を参照してください。
- 「Apache Flink アプリケーション用 Managed Service の作成」:以下のいずれかの方法を使用して Apache Flink アプリケーション用 Managed Service を作成します。

- AWS コンソールを使用して Managed Service for Apache Flink アプリケーションを作成する：AWS コンソールを使用してアプリケーションを作成および設定できます。

  コンソールを使用してアプリケーションを作成すると、アプリケーションの依存リソース (CloudWatch ログストリーム、IAM ロール、IAM ポリシーなど) が自動的に作成されます。

  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、「Apache Flink 用 Managed Service - アプリケーションの作成」ページのプルダウンから選択して、アプリケーションが使用する Apache Flink のバージョンを指定します。

  コンソールを使用してアプリケーションを作成する方法に関するチュートリアルについては、「開始方法 (DataStream API)」チュートリアルの「the section called “アプリケーションの作成と実行 (コンソール) ”」を参照してください。
- AWS CLI を使用して Managed Service for Apache Flink アプリケーションを作成する： AWS CLI を使用してアプリケーションを作成および設定できます。

  CLI を使用してアプリケーションを作成する場合は、アプリケーションの依存リソース (CloudWatch ログストリーム、IAM ロール、IAM ポリシーなど) も手動で作成する必要があります。

  CLI を使用してアプリケーションを作成する場合、 `CreateApplication` アクションの `RuntimeEnvironment` パラメータを使用して、アプリケーションが使用する Apache Flink のバージョンを指定します。

  CLI を使用してアプリケーションを作成する方法に関するチュートリアルについては、 開始方法 (DataStream API) チュートリアルの the section called “CLI を使用してアプリケーションを作成して実行する” を参照してください。

> **Note**
>
> 既存のアプリケーションの `RuntimeEnvironment` を変更できます。この方法の詳細は、Apache Flink のインプレースバージョンアップグレードを参照してください。

Managed Service for Apache Flink アプリケーションの起動

アプリケーションコードを作成し、S3 にアップロードし、Apache Flink アプリケーション用 Managed Service を作成したら、アプリケーションを起動します。Apache Flink 用 Managed Service アプリケーションの起動には、通常数分かかります。

アプリケーションを起動するには、以下のいずれかの方法を使用します。

- AWS コンソールを使用して Managed Service for Apache Flink アプリケーションを起動する：AWS コンソールのアプリケーションのページで実行を選択して、アプリケーションを実行できます。
- AWS API を使用して Managed Service for Apache Flink アプリケーションを起動します。StartApplicationアクションを使用してアプリケーションを実行できます。

Managed Service for Apache Flink アプリケーションの検証

アプリケーションが動作していることを確認するには、次の方法があります。

- CloudWatch ログの使用： CloudWatch Logs と Logs Insights CloudWatch を使用して、アプリケーションが正しく実行されていることを確認できます。Managed Service for Apache Flink アプリケーションで CloudWatch ログを使用する方法については、「」を参照してくださいロギングとモニタリング。
- CloudWatch メトリクスの使用： CloudWatch メトリクスを使用して、アプリケーションのアクティビティ、またはアプリケーションが入力または出力に使用するリソース (Kinesis ストリーム、Firehose ストリーム、Amazon S3 バケットなど) のアクティビティをモニタリングできます。 CloudWatch メトリクスの詳細については、「Amazon ユーザーガイド」の「メトリクスの使用 CloudWatch 」を参照してください。
- 「出力ロケーションのモニタリング:」アプリケーションが出力を特定のロケーション (Amazon S3 バケットやデータベースなど) に書き込む場合、そのロケーションに書き込まれたデータを監視できます。

Managed Service for Apache Flink アプリケーションのシステムロールバックの有効化

システムロールバック機能を使用すると、Amazon Managed Service for Apache Flink で実行中の Apache Flink アプリケーションの可用性を高めることができます。この設定を選択すると、UpdateApplicationや などのアクションがコードや設定のバグにautoscaling陥ったときに、サービスがアプリケーションを自動的に以前の実行バージョンに戻すことができます。

> **Note**
>
> システムロールバック機能を使用するには、アプリケーションを更新してオプトインする必要があります。既存のアプリケーションは、デフォルトではシステムロールバックを自動的に使用しません。

仕組み

更新アクションやスケーリングアクションなどのアプリケーションオペレーションを開始すると、Amazon Managed Service for Apache Flink はそのオペレーションの実行を最初に試みます。コードのバグやアクセス許可の不足など、オペレーションが成功しない問題が検出されると、サービスは自動的にRollbackApplicationオペレーションを開始します。

ロールバックは、関連付けられたアプリケーションの状態とともに、正常に実行された以前のバージョンにアプリケーションを復元しようとします。ロールバックが成功すると、アプリケーションは以前のバージョンを使用して最小限のダウンタイムでデータの処理を続行します。自動ロールバックも失敗した場合、Amazon Managed Service for Apache Flink はアプリケーションを READYステータスに移行し、エラーの修正やオペレーションの再試行など、さらにアクションを実行できるようにします。

自動システムロールバックを使用するにはオプトインする必要があります。コンソールまたは API を使用して、この時点からアプリケーションのすべてのオペレーションで有効にできます。

次の UpdateApplicationアクションのリクエスト例では、アプリケーションのシステムロールバックを有効にします。

```
{
   "ApplicationName": "MyApplication",
   "CurrentApplicationVersionId": 1,
   "ApplicationConfigurationUpdate": {
      "ApplicationSystemRollbackConfigurationUpdate": {
         "RollbackEnabledUpdate": "true"
       }
    }
}
```

一般的なシナリオ

次のシナリオは、自動システムロールバックが有益な場所を示しています。

- アプリケーションの更新： メインメソッドを使用して Flink ジョブを初期化するときにバグがある新しいコードでアプリケーションを更新すると、自動ロールバックによって以前の動作バージョンを復元できます。システムロールバックが役立つその他の更新シナリオは次のとおりです。
  - アプリケーションが更新され、maxParallelism よりも高い並列処理で実行される場合。
  - Flink ジョブの起動中に障害が発生する VPC アプリケーションのサブネットが正しくない状態で実行されるようにアプリケーションが更新された場合。
- Flink バージョンのアップグレード： 新しい Apache Flink バージョンにアップグレードし、アップグレードされたアプリケーションにスナップショットの互換性の問題が発生すると、システムロールバックにより、以前の Flink バージョンに自動的に戻ることができます。
- AutoScaling： スナップショットと Flink ジョブグラフのオペレータの不一致により、アプリケーションがスケールアップしてもセーブポイントからの復元に問題が発生する場合。

オペレーション APIs

可視性を高めるために、Amazon Managed Service for Apache Flink には、障害や関連するシステムロールバックの追跡に役立つアプリケーションオペレーションに関連する 2 つの APIs があります。

`ListApplicationOperations`

この API は、、、 などUpdateApplication、アプリケーションで実行されたすべてのオペレーションを逆の時系列MaintenanceRollbackApplicationで一覧表示します。次のListApplicationOperationsアクションのリクエスト例では、アプリケーションの最初の 10 個のアプリケーションオペレーションを一覧表示します。

```
{
   "ApplicationName": "MyApplication",
   "Limit": 10
}
```

次の のリクエスト例は、リストをアプリケーションの以前の更新にフィルタリングするListApplicationOperationsのに役立ちます。

```
{
   "ApplicationName": "MyApplication",
   "operation": "UpdateApplication"
}
```

`DescribeApplicationOperation`

この API は、該当する場合、障害の理由などListApplicationOperations、にリストされている特定のオペレーションに関する詳細情報を提供します。次のDescribeApplicationOperationアクションのリクエスト例では、特定のアプリケーションオペレーションの詳細を一覧表示します。

```
{
   "ApplicationName": "MyApplication",
   "OperationId": "xyzoperation"
}
```

トラブルシューティング情報については、システムロールバックのベストプラクティスを参照してください。

Managed Service for Apache Flink アプリケーションの実行

このトピックには、Apache Flink 用 Managed Serviceの実行に関する情報が含まれています。

Apache Flink アプリケーション用 Managed Serviceを実行すると、サービスによって Apache Flink ジョブが作成されます。Apache Flink ジョブは、Apache Flink アプリケーション用 Managed Serviceの実行ライフサイクルです。Job の実行とそれが使用するリソースは、ジョブマネージャーによって管理されます。ジョブマネージャは、アプリケーションの実行をタスクに分割します。各タスクはタスクマネージャーによって管理されます。アプリケーションのパフォーマンスを監視する場合、各タスクマネージャーまたはジョブマネージャー全体のパフォーマンスを調べることができます。

Apache Flink ジョブの詳細については、Apache Flink ドキュメントの「ジョブとスケジューリング」を参照してください。

アプリケーションとジョブのステータス

アプリケーションとアプリケーションのジョブの両方に現在の実行ステータスがあります。

- 「アプリケーションステータス:」アプリケーションには、実行フェーズを説明する現在のステータスがあります。アプリケーション状態には以下のものがあります。
  - 「安定したアプリケーションステータス:」通常、ステータスを変更するまで、アプリケーションは次のステータスのままになります。
    - 「READY:」新規または停止中のアプリケーションは、実行するまで準備完了状態です。
    - 「RUNNING」正常に起動したアプリケーションは RUNNING ステータスになります。

- 「一時的なアプリケーションステータス:」これらのステータスのアプリケーションは、通常、別のステータスへの移行中です。アプリケーションが一定期間一時的なステータスのままである場合は、 Force パラメータを に設定して StopApplicationアクションを使用してアプリケーションを停止できますtrue。これらのステータスには以下のものが含まれる：
  - STARTING: StartApplicationアクションの後に発生します。アプリケーションは READY から RUNNING ステータスに移行中です。
  - STOPPING: StopApplicationアクションの後に発生します。アプリケーションは RUNNING から READY ステータスに移行中です。
  - DELETING: DeleteApplicationアクションの後に発生します。アプリケーション は削除中です。
  - UPDATING: UpdateApplicationアクションの後に発生します。アプリケーションは更新中で、RUNNING または READY ステータスに戻ります。
  - AUTOSCALING: アプリケーションには の AutoScalingEnabledプロパティが ParallelismConfigurationに設定されておりtrue、サービスはアプリケーションの並列処理を増やしています。アプリケーションがこのステータスの場合、使用できる唯一の有効な API アクションは、 Forceパラメータを に設定した StopApplicationアクションですtrue。自動スケーリングの詳細については、「Auto Scaling」 を参照してください。
  - FORCE_STOPPING: Forceパラメータを に設定して StopApplicationアクションが呼び出された後に発生しますtrue。アプリケーションは強制停止中です。アプリケーションはSTARTING 、UPDATING 、STOPPING または AUTOSCALING ステータスから READY ステータスに移行中です。
  - ROLLING_BACK: RollbackApplicationアクションが呼び出された後に発生します。アプリケーションを以前のバージョンにロールバックしています。アプリケーションは UPDATING または AUTOSCALING ステータスから RUNNING ステータスに移行します。
  - MAINTENANCE: は Managed Service for Apache Flinkがアプリケーションにパッチを適用しているときに発生します。詳細については、「メンテナンス」を参照してください。

  アプリケーションのステータスは、コンソールまたは DescribeApplicationアクションを使用して確認できます。
- 「Job ステータス:」 アプリケーションが RUNNING ステータスになると、ジョブには現在の実行フェーズを説明するステータスが表示されます。ジョブは CREATED ステータスで開始し、開始後 RUNNING ステータスに進みます。エラー状態が発生すると、アプリケーションは次のステータスになります。

- Apache Flink 1.11 以降を使用するアプリケーションでは、アプリケーションが RESTARTING ステータスになります。
- Apache Flink 1.8 以前を使用するアプリケーションでは、アプリケーションが FAILING ステータスに入ります。

その後、アプリケーションは、ジョブを再開できるかどうかに応じて、 RESTARTING または FAILED のステータスに進みます。

ジョブのステータスを確認するには、アプリケーション CloudWatch ログでステータスの変更を確認します。

バッチワークロード

Apache Flink 用 Managed ServiceApache Flink 用 Managed Serviceは、バッチワークロードの実行をサポートしています。バッチジョブでは、Apache Flink ジョブが 「FINISHED」 ステータスになると、Apache Flink アプリケーションス用 Managed Service のテータスは 「READY」 に設定されます。Flink ジョブのステータスについて詳しくは、「ジョブとスケジューリング」 を参照してください。

アプリケーションリソース

このセクションでは、アプリケーションが使用するシステムリソースについて説明します。Apache Flink 用 Managed Serviceがどのようにリソースをプロビジョニングして使用するかを理解しておくと、パフォーマンスが高く安定した Apache Flink アプリケーション用 Managed Serviceの設計、作成、維持に役立ちます。

Managed Service for Apache Flink アプリケーションリソース

Managed Service for Apache Flink は、Apache Flink アプリケーションをホストするための環境を作成する AWS サービスです。Apache Flink 用 Managed Serviceは、「Kinesis プロセッシングユニット (KPU)」と呼ばれるユニットを使用してリソースを提供します。

1 つの KPU は次のシステムリソースを表します。

- 1つのCPUコア
- 4 GBのメモリ(1 GBがネイティブメモリ、3 GBがヒープメモリ)
- 50 GB のディスクスペース

KPU は、「タスク」と「サブタスク」と呼ばれる別々の実行単位でアプリケーションを実行します。サブタスクはスレッドと同等と考えることができます。

アプリケーションで使用できる KPU の数は、アプリケーションの `Parallelism` 設定をアプリケーションの `ParallelismPerKPU` 設定で割った数です。

アプリケーションの並列処理については、 スケーリング をご参照ください。

Apache Flink アプリケーションリソース

Apache Flink 環境は、「タスクスロット」と呼ばれる単位を使用してアプリケーションのリソースを割り当てます。Apache Flink 用 Managed Serviceがアプリケーションにリソースを割り当てると、1 つ以上の Apache Flink タスクスロットが 1 つの KPU に割り当てられます。1 つの KPU に割り当てられるスロット数は、アプリケーションの `ParallelismPerKPU` 設定と同じです。タスクスロットの詳細については、「Apache Flink ドキュメント」の「ジョブのスケジュール」を参照してください。

演算子の並列処理

オペレータが使用できるサブタスクの最大数を設定できます。この値は「オペレータ並列度」と呼ばれます。デフォルトでは、アプリケーション内の各オペレータの並列度はアプリケーションの並列度と同じです。つまり、デフォルトでは、アプリケーション内の各オペレータは、必要に応じてアプリケーションで使用可能なすべてのサブタスクを使用できます。

`setParallelism` メソッドを使用して、アプリケーション内のオペレータの並列度を設定できます。この方法を使用すると、各オペレータが一度に使用できるサブタスクの数を制御できます。

演算子の詳細については、Apache Flink ドキュメントの「演算子」を参照してください。

オペレーターの連鎖

通常、各オペレータは別々のサブタスクを使用して実行しますが、複数のオペレータが常に順番に実行する場合、ランタイムはそれらすべてを同じタスクに割り当てることができます。このプロセスは「オペレータチェイニング」と呼ばれます。

複数のシーケンシャルオペレータがすべて同じデータを操作する場合、それらを 1 つのタスクにまとめることができます。そのために必要ないくつかの基準を以下に挙げます。

- オペレータは 1 対 1 の単純な転送を行います。
- オペレータの並列度はすべて同じです。

アプリケーションがオペレータを 1 つのサブタスクにチェーンすると、サービスがネットワーク操作を実行したり、オペレータごとにサブタスクを割り当てたりする必要がなくなるため、システムリソースを節約できます。アプリケーションがオペレータチェイニングを使用しているかどうかを確認するには、Apache Flink 用 Managed Serviceコンソールのジョブグラフを見てください。アプリケーションの各頂点は 1 つ以上のオペレータを表します。グラフには、1 つの頂点として連結されたオペレータが表示されます。

DataStream API

Apache Flink アプリケーションは、Apache Flink DataStream API を使用してデータストリーム内のデータを変換します。

このセクションは、以下のトピックで構成されます。

- コネクタを使用して DataStream API で Managed Service for Apache Flink のデータを移動する: これらのコンポーネントは、アプリケーションと外部データソースおよび宛先との間でデータを移動します。
- DataStream API を使用した Managed Service for Apache Flink の演算子を使用したデータの変換: これらのコンポーネントは、アプリケーション内のデータ要素を変換またはグループ化します。
- DataStream API を使用した Managed Service for Apache Flink でのイベントの追跡: このトピックでは、Managed Service for Apache Flink が DataStream API を使用するときにイベントを追跡する方法について説明します。

コネクタを使用して DataStream API で Managed Service for Apache Flink のデータを移動する

Amazon Managed Service for Apache Flink DataStream API では、コネクタは Managed Service for Apache Flink アプリケーションとの間でデータを移動するソフトウェアコンポーネントです。コネクタは、ファイルやディレクトリからの読み取りを可能にする柔軟な統合です。コネクタは、Amazon のサービスやサードパーティのシステムとやり取りするための完全なモジュールで構成されています。

コネクターの種類には、次のものがあります。

- [Sources] (出典): Kinesis データストリーム、ファイル、またはその他のデータソースからアプリケーションにデータを提供します。

- シンク: アプリケーションから Kinesis データストリーム、Firehose ストリーム、またはその他のデータ送信先にデータを送信します。
- 非同期 I/O: データソース (データベースなど) への非同期アクセスを提供し、ストリームイベントを充実させます。

使用可能なコネクタ

Apache Flink フレームワークには、さまざまなソースのデータにアクセスするためのコネクターが含まれています。Apache Flink フレームワークで使用できるコネクタについては、「Apache Flink ドキュメント」の「コネクタ」を参照してください。

Warning

Flink 1.6、1.8、1.11、または 1.13 で実行されているアプリケーションがあり、中東 (アラブ首長国連邦)、アジアパシフィック (ハイデラバード)、イスラエル (テルアビブ)、欧州 (チューリッヒ)、中東 (アラブ首長国連邦)、アジアパシフィック (メルボルン)、またはアジアパシフィック (ジャカルタ) の各リージョンで実行する場合は、更新されたコネクタを使用してアプリケーションアーカイブを再構築するか、Flink 1.18 にアップグレードする必要があります。
Apache Flink コネクタは、独自のオープンソースリポジトリに保存されます。バージョン 1.18 以降にアップグレードする場合は、依存関係を更新する必要があります。Apache Flink AWS コネクタのリポジトリにアクセスするには、「」を参照してくださいflink-connector-aws。
以下は推奨されるガイドラインです。

コネクタのアップグレード

F バ ジ ョ ン	使用されるコネクタ	解決方法
1	EFO	Amazon Managed Service for Apache Flink バージョン 1.15 にアップグレードする場合は、最新の EFO コネクタを使用していることを確認してください。バー

F バ ジ ョ ン	使用されるコネクタ	解決方法
		ジョン 1.15.3 以降である必要があります。詳細については、以下を参照してください。 FLINK-29324。
1	Amazon Data Firehose シンク	Amazon Managed Service for Apache Flink バージョン 1.15 にアップグレードするときは、最新の Amazon Data Firehose シンクを使用していることを確認してください。 Amazon Data Firehose シンク
1	Kafka コネクタ	Amazon Managed Service for Apache Flink バージョン 1.15 にアップグレードするときは、最新の Kafka コネクタ APIsを使用していることを確認してください。Apache Flink は FlinkKafkaConsumerと を廃止しましたFlinkKafkaProducer。Kafka シンクのこれらの APIsは、Flink 1.15 の Kafka にコミットできません。KafkaSource と を使用していることを確認しますKafkaSink。

F バ ョ ン	使用されるコネクタ	解決方法
1 - 1	Firehose	アプリケーションは、新しい AWS リージョンを認識しない古いバージョンの Firehose コネクタに依存します。Firehose コネクターバージョン 2.1.0 でアプリケーションアーカイブを再構築します。 「v2.1.0」
1	Kinesis	アプリケーションは、新しい AWS リージョンを認識しない古いバージョンの Flink Kinesis コネクタに依存します。Flink Kinesis コネクターバージョン 1.6.1 を使用してアプリケーションアーカイブを再構築します。 https://github.com/awslabs/amazon-kinesis-connector-flink/tree/1.6.1

F バ ジ ョ ン	使用されるコネクタ	解決方法
1	Kinesis	アプリケーションは、新しい AWS リージョンを認識しない古いバージョンの Flink Kinesis コネクタに依存します。Flink Kinesis コネクターバージョン 2.4.1 を使用してアプリケーションアーカイブを再構築します。 https://github.com/awslabs/amazon-kinesis-connector-flink/tree/2.4.1
1 と 1	Kinesis	アプリケーションは、新しい AWS リージョンを認識しない古いバージョンの Flink Kinesis コネクタに依存します。残念ながら、Flink は 1.6/1.13 コネクタのパッチやバグ修正をもうリリースしていません。Flink 1.15 でアプリケーションアーカイブを再構築して Flink 1.15 にアップデートすることをお勧めします。

Managed Service for Apache Flink へのストリーミングデータソースの追加

Apache Flink には、ファイル、ソケット、コレクション、カスタムソースから読み取るためのコネクタが用意されています。アプリケーションコードでは、「Apache Flink ソース」を使用してスト

リームからデータを受信します。このセクションでは、Amazon のサービスで使用できるソースについて説明します。

Kinesis Data Streams

FlinkKinesisConsumer ソースは Amazon Kinesis データストリームからアプリケーションにストリーミングデータを提供します。

「**FlinkKinesisConsumer**」の作成

次のコード例は、FlinkKinesisConsumer の作成を示しています。

```
Properties inputProperties = new Properties();
inputProperties.setProperty(ConsumerConfigConstants.AWS_REGION, region);
inputProperties.setProperty(ConsumerConfigConstants.STREAM_INITIAL_POSITION, "LATEST");

DataStream<string> input = env.addSource(new FlinkKinesisConsumer<>(inputStreamName,
 new SimpleStringSchema(), inputProperties));
```

FlinkKinesisConsumer の使用方法の詳細については、「Apache Flink ストリーミング Java コードをダウンロードして調べる」を参照してください。

EFO コンシューマーを使用する **FlinkKinesisConsumer** の作成

で拡張ファンアウト (EFO) がサポートされ FlinkKinesisConsumer るようになりました。

Kinesis コンシューマーが EFO を使用する場合、Kinesis Data Streams サービスは、コンシューマーがストリームの固定帯域幅を、ストリームから読み取る他のコンシューマーと共有するのではなく、独自の専用帯域幅を提供します。

Kinesis コンシューマーで EFO を使用する方法の詳細については、FLIP-128: Enhanced Fan Out for AWS Kinesis Consumers」を参照してください。

EFO コンシューマーを有効にするには、Kinesis コンシューマーで次のパラメータを設定します。

- RECORD_PUBLISHER_TYPE: アプリケーションが EFO Consumerを使用して Kinesis Data Streamsデータにアクセスできるようにするには、このパラメータを EFO に設定します。
- EFO_CONSUMER_NAME: このパラメータを、このストリームのコンシューマー間で一意の文字列値に設定します。同じ Kinesis Data Stream でコンシューマー名を再利用すると、その名前を使用していた以前のコンシューマーは終了します。

EFO を使用するように FlinkKinesisConsumer を設定するには、コンシューマーに以下のパラメータを追加します。

```
consumerConfig.putIfAbsent(RECORD_PUBLISHER_TYPE, "EFO");
consumerConfig.putIfAbsent(EFO_CONSUMER_NAME, "basic-efo-flink-app");
```

EFO コンシューマーを使用する Apache Flink 用 Managed Serviceの例については、 EFO Consumer を参照してください。

Amazon MSK

KafkaSource ソースは Amazon MSK トピックからアプリケーションにストリーミングデータを提供します。

「KafkaSource」の作成

次のコード例は、KafkaSource の作成を示しています。

```
KafkaSource<String> source = KafkaSource.<String>builder()
    .setBootstrapServers(brokers)
    .setTopics("input-topic")
    .setGroupId("my-group")
    .setStartingOffsets(OffsetsInitializer.earliest())
    .setValueOnlyDeserializer(new SimpleStringSchema())
    .build();

env.fromSource(source, WatermarkStrategy.noWatermarks(), "Kafka Source");
```

KafkaSource の使用方法の詳細については、「MSK レプリケーション」を参照してください。

Managed Service for Apache Flink でシンクを使用してデータを書き込む

アプリケーションコードでは、任意の Apache Flink シンクコネクタを使用して、Kinesis Data Streams や DynamoDB などの AWS サービスを含む外部システムに書き込むことができます。

Apache Flink はファイルとソケット用のシンクも提供しており、カスタムシンクを実装できます。サポートされている複数のシンクの中で、以下がよく使用されます。

Kinesis Data Streams

Apache Flink では、「Kinesis Data Streams Connector」に関する情報が Apache Flink ドキュメントに記載されています。

Kinesis データストリームを入力と出力に使用するアプリケーションの例については、開始方法 (DataStream API) を参照してください。

Apache Kafka と Amazon Managed Streaming for Apache Kafka (MSK)

Apache Flink Kafka コネクタは、1 回限りの保証を含め、Apache Kafka と Amazon MSK へのデータの公開を広範囲にサポートします。Kafka への書き込み方法については、Apache Flink ドキュメントの「Kafka Connectors の例」を参照してください。

Amazon S3

Amazon S3 バケットにオブジェクトを書き込むには、Apache Flink `StreamingFileSink` を使用できます。

S3 にオブジェクトを書き込む方法の例については、the section called “S3 シンク” を参照してください。

Firehose

`FlinkKinesisFirehoseProducer` は、Firehose サービスを使用してアプリケーション出力を保存するための、信頼性が高くスケーラブルな Apache Flink シンクです。このセクションでは、Maven プロジェクトをセットアップして `FlinkKinesisFirehoseProducer` を作成・使用するための設定方法について説明します。

トピック

- 「FlinkKinesisFirehoseProducer」の作成
- FlinkKinesisFirehoseProducer コード例

「**`FlinkKinesisFirehoseProducer`**」の作成

次のコード例は、`FlinkKinesisFirehoseProducer` の作成を示しています。

```
Properties outputProperties = new Properties();
outputProperties.setProperty(ConsumerConfigConstants.AWS_REGION, region);

FlinkKinesisFirehoseProducer<String> sink = new
 FlinkKinesisFirehoseProducer<>(outputStreamName, new SimpleStringSchema(),
 outputProperties);
```

`FlinkKinesisFirehoseProducer` コード例

次のコード例は、 を作成して設定`FlinkKinesisFirehoseProducer`し、Apache Flink データストリームから Firehose サービスにデータを送信する方法を示しています。

```
package com.amazonaws.services.kinesisanalytics;

import
 com.amazonaws.services.kinesisanalytics.flink.connectors.config.ProducerConfigConstants;
import
 com.amazonaws.services.kinesisanalytics.flink.connectors.producer.FlinkKinesisFirehoseProducer
import com.amazonaws.services.kinesisanalytics.runtime.KinesisAnalyticsRuntime;
import org.apache.flink.api.common.serialization.SimpleStringSchema;
import org.apache.flink.streaming.api.datastream.DataStream;
import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
import org.apache.flink.streaming.connectors.kinesis.FlinkKinesisConsumer;
import org.apache.flink.streaming.connectors.kinesis.FlinkKinesisProducer;

import org.apache.flink.streaming.connectors.kinesis.config.ConsumerConfigConstants;

import java.io.IOException;
import java.util.Map;
import java.util.Properties;

public class StreamingJob {

 private static final String region = "us-east-1";
 private static final String inputStreamName = "ExampleInputStream";
 private static final String outputStreamName = "ExampleOutputStream";

 private static DataStream<String>
 createSourceFromStaticConfig(StreamExecutionEnvironment env) {
  Properties inputProperties = new Properties();
  inputProperties.setProperty(ConsumerConfigConstants.AWS_REGION, region);
  inputProperties.setProperty(ConsumerConfigConstants.STREAM_INITIAL_POSITION,
 "LATEST");

  return env.addSource(new FlinkKinesisConsumer<>(inputStreamName, new
 SimpleStringSchema(), inputProperties));
 }

 private static DataStream<String>
 createSourceFromApplicationProperties(StreamExecutionEnvironment env)
```

```
  throws IOException {
 Map<String, Properties> applicationProperties =
KinesisAnalyticsRuntime.getApplicationProperties();
 return env.addSource(new FlinkKinesisConsumer<>(inputStreamName, new
SimpleStringSchema(),
   applicationProperties.get("ConsumerConfigProperties")));
}

private static FlinkKinesisFirehoseProducer<String>
createFirehoseSinkFromStaticConfig() {
 /*
  * com.amazonaws.services.kinesisanalytics.flink.connectors.config.
  * ProducerConfigConstants
  * lists of all of the properties that firehose sink can be configured with.
  */

 Properties outputProperties = new Properties();
 outputProperties.setProperty(ConsumerConfigConstants.AWS_REGION, region);

 FlinkKinesisFirehoseProducer<String> sink = new
FlinkKinesisFirehoseProducer<>(outputStreamName,
   new SimpleStringSchema(), outputProperties);
 ProducerConfigConstants config = new ProducerConfigConstants();
 return sink;
}

private static FlinkKinesisFirehoseProducer<String>
createFirehoseSinkFromApplicationProperties() throws IOException {
 /*
  * com.amazonaws.services.kinesisanalytics.flink.connectors.config.
  * ProducerConfigConstants
  * lists of all of the properties that firehose sink can be configured with.
  */

 Map<String, Properties> applicationProperties =
KinesisAnalyticsRuntime.getApplicationProperties();
 FlinkKinesisFirehoseProducer<String> sink = new
FlinkKinesisFirehoseProducer<>(outputStreamName,
   new SimpleStringSchema(),
   applicationProperties.get("ProducerConfigProperties"));
 return sink;
}

public static void main(String[] args) throws Exception {
```

```
  // set up the streaming execution environment
  final StreamExecutionEnvironment env =
 StreamExecutionEnvironment.getExecutionEnvironment();

  /*
   * if you would like to use runtime configuration properties, uncomment the
   * lines below
   * DataStream<String> input = createSourceFromApplicationProperties(env);
   */

  DataStream<String> input = createSourceFromStaticConfig(env);

  // Kinesis Firehose sink
  input.addSink(createFirehoseSinkFromStaticConfig());

  // If you would like to use runtime configuration properties, uncomment the
  // lines below
  // input.addSink(createFirehoseSinkFromApplicationProperties());

  env.execute("Flink Streaming Java API Skeleton");
 }
}
```

Firehose シンクの使用方法に関する完全なチュートリアルについては、「」を参照してくださいthe section called “Firehose シンク”。

Apache Flink 用 Managed Serviceでの非同期 I/O の使用率

非同期 I/O オペレータは、データベースなどの外部データソースを使用してストリームデータを強化します。Apache Flink 用 Managed Serviceはストリームイベントを非同期的に強化するため、リクエストを一括処理して効率を高めることができます。

詳細については、Apache Flink ドキュメントの「非同期 I/O」を参照してください。

DataStream API を使用した Managed Service for Apache Flink の演算子を使用したデータの変換

Apache Flink 用 Managed Service の受信データを変換するには、Apache Flink「オペレータ」を使用します。Apache Flink オペレータは 1 つ以上のデータストリームを新しいデータストリームに変換します。新しいデータストリームには、元のデータストリームから変更されたデータが含まれま

す。Apache Flink には 25 種類以上のストリーム処理オペレータがあらかじめ組み込まれています。詳細については、「Apache Flink ドキュメント」の「オペレータ」を参照してください。

このトピックには、次のセクションが含まれています。

- 変換演算子
- 集計演算子

変換演算子

JSON データストリームのいずれか 1 つのフィールドの簡単なテキスト変換の例を次に示します。

このコードは変換されたデータストリームを作成します。新しいデータストリームは、元のストリームと同じデータを持ち、 TICKER フィールドの内容に文字列　Company が追加されます。

```
DataStream<ObjectNode> output = input.map(
    new MapFunction<ObjectNode, ObjectNode>() {
        @Override
        public ObjectNode map(ObjectNode value) throws Exception {
            return value.put("TICKER", value.get("TICKER").asText() + " Company");
        }
    }
);
```

集計演算子

次は集約オペレータの例です。このコードは集約されたデータストリームを作成します。このオペレータは 5 秒間のタンブリングウィンドウを作成し、ウィンドウ内の同じ TICKER 値を持つレコードの PRICE 値の合計を返します。

```
DataStream<ObjectNode> output = input.keyBy(node -> node.get("TICKER").asText())
    .window(TumblingProcessingTimeWindows.of(Time.seconds(5)))
    .reduce((node1, node2) -> {
        double priceTotal = node1.get("PRICE").asDouble() +
 node2.get("PRICE").asDouble();
        node1.replace("PRICE", JsonNodeFactory.instance.numberNode(priceTotal));
    return node1;
});
```

その他のコード例については、「」を参照してください例。

DataStream API を使用した Managed Service for Apache Flink でのイベントの追跡

Apache Flink 用 Managed Service は、次のタイムスタンプを使用してイベントを追跡します。

- 「Processing Time:」それぞれの操作を実行しているマシンのシステム時間を指します。
- 「イベント時間:」各イベントが発生デバイスで発生した時刻を指します。
- 「取り込み時間:」Apache Flink サービス用 Managed Service にイベントが入るまでの時間を指します。

ストリーミング環境で使用される時間は、 を使用して設定しますsetStreamTimeCharacteristic。

```
env.setStreamTimeCharacteristic(TimeCharacteristic.ProcessingTime);
env.setStreamTimeCharacteristic(TimeCharacteristic.IngestionTime);
env.setStreamTimeCharacteristic(TimeCharacteristic.EventTime);
```

タイムスタンプの詳細については、Apache Flink ドキュメントの「透かしの生成」を参照してください。

Table API

Apache Flink アプリケーションは、「Apache Flink テーブル API」を使用して、リレーショナルモデルを使用してストリーム内のデータを操作します。Table API を使用してテーブルソースを使用してデータにアクセスし、次にテーブル関数を使用してテーブルデータを変換およびフィルタリングします。API 関数または SQL コマンドを使用して、表形式のデータを変換およびフィルタリングできます。

このセクションは、以下のトピックで構成されます。

- テーブル API コネクタ: これらのコンポーネントは、アプリケーションと外部データソースおよび宛先との間でデータを移動します。
- テーブル API の時間属性: このトピックでは、Managed Service for Apache Flinkが Table API を使用する際にイベントをトラッキングする方法について説明します。

テーブル API コネクタ

Apache Flink プログラミングモデルでは、コネクタは、アプリケーションが他の AWS サービスなどの外部ソースからデータを読み書きするために使用するコンポーネントです。

Apache Flink テーブル API では、以下のタイプのコネクタを使用できます。

- テーブル API ソース: テーブルAPIソースコネクタを使用して、APIコールまたはSQLクエリを使用して TableEnvironment 内にテーブルを作成します。
- テーブル API シンク: SQL コマンドを使用して、Amazon MSK トピックや Amazon S3 バケットなどの外部ソースにテーブルデータを書き込みます。

テーブル API ソース

データストリームからテーブルソースを作成します。次のコードは Amazon MSK トピックからテーブルを作成します。

```
//create the table
    final FlinkKafkaConsumer<StockRecord> consumer = new
 FlinkKafkaConsumer<StockRecord>(kafkaTopic, new KafkaEventDeserializationSchema(),
 kafkaProperties);
    consumer.setStartFromEarliest();
    //Obtain stream
    DataStream<StockRecord> events = env.addSource(consumer);

    Table table = streamTableEnvironment.fromDataStream(events);
```

テーブルソースの詳細については、Apache Flink ドキュメントの「テーブルと SQL コネクタ」を参照してください。

テーブル API シンク

テーブルデータをシンクに書き込むには、SQL でシンクを作成し、その StreamTableEnvironment オブジェクトで SQL ベースのシンクを実行します。

次のコードの例は、テーブルのデータを Amazon S3 シンクに書き込む方法を示しています。

```
final String s3Sink = "CREATE TABLE sink_table (" +
    "event_time TIMESTAMP," +
    "ticker STRING," +
```

```
    "price DOUBLE," +
    "dt STRING," +
    "hr STRING" +
    ")" +
    " PARTITIONED BY (ticker,dt,hr)" +
    " WITH" +
    "(" +
    " 'connector' = 'filesystem'," +
    " 'path' = '" + s3Path + "'," +
    " 'format' = 'json'" +
    ") ";

    //send to s3
    streamTableEnvironment.executeSql(s3Sink);
    filteredTable.executeInsert("sink_table");
```

この format パラメータを使用して、Apache Flink 用 Managed Serviceが出力をシンクに書き込む際に使用するフォーマットを制御できます。形式の詳細については、Apache Flink ドキュメントの「サポートされているコネクタ」を参照してください。

ユーザー定義のソースとシンク

既存の Apache Kafka コネクタを使用して、Amazon MSK や Amazon S3 などの他の AWS サービスとの間でデータを送受信できます。他のデータソースや送信先とやり取りする場合は、独自のソースとシンクを定義できます。詳細については、「Apache Flink ドキュメント」の「ユーザー定義のソースとシンク」を参照してください。

テーブル API の時間属性

データストリーム内の各レコードには、そのレコードに関連するイベントがいつ発生したかを定義する複数のタイムスタンプがあります。

- 「イベント時間」:レコードを作成したイベントがいつ発生したかを定義するユーザー定義のタイムスタンプ。
- 「取り込み時間」:アプリケーションがデータストリームからレコードを取得した時刻。
- 「処理時間」:アプリケーションがレコードを処理した時間。

Apache Flink Table API がレコード時間に基づいてウィンドウを作成する場合、setStreamTimeCharacteristicメソッドを使用して、これらのタイムスタンプのどれを使用するかを定義します。

Table API でのタイムスタンプの使用の詳細については、Apache Flink ドキュメントの「Time Attributes」と「Timely Stream Processing」を参照してください。

Apache Flinkのマネージド・サービスでPythonを使用する

> **Note**
>
> Apple Silicon チップを搭載した新しい Mac で Python Flink アプリケーションを開発している場合、Python の依存関係 PyFlink 1.15 で既知の問題が発生する可能性があります。この場合、Docker で Python インタープリターを実行することをお勧めします。 step-by-step 手順については、PyFlink 「Apple Silicon Mac での 1.15 開発」を参照してください。

Apache Flink バージョン 1.18.1 には、Python バージョン 3.10 を使用したアプリケーションの作成のサポートが含まれています。詳細については、「Flink Python Docs」を参照してください。Python を使用して Apache Flink アプリケーション用マネージドサービスを作成するには、次の手順を実行します。

- Python `main` アプリケーションコードをメソッドを含むテキストファイルとして作成します。
- アプリケーションコードファイルおよび Python または Java の依存関係を zip ファイルにバンドルし、Amazon S3 バケットにアップロードします。
- Amazon S3 コードの場所、アプリケーションプロパティ、およびアプリケーション設定を指定して、Apache Flink アプリケーション用 Managed Service を作成します。

大まかに言うと、Python テーブル API は Java テーブル API のラッパーのようなものです。Python テーブル API の詳細については、Apache Flink ドキュメントの「テーブル API チュートリアル」を参照してください。

Managed Service for Apache Flink for Python アプリケーションのプログラミング

Python アプリケーション向けのApache Flink 用 Managed Serviceは、Apache Flink Python テーブル API を使用してコーディングします。Apache Flink エンジンは Python テーブル API ステートメント (Python VM で実行されている) を Java テーブル API ステートメント (Java VM で実行されている) に変換します。

Python テーブル API を使用するには、次の操作を行います。

- StreamTableEnvironment へのリファレンスを作成します。
- StreamTableEnvironment リファレンスに対してクエリを実行して、table ソースストリーミングデータからオブジェクトを作成します。
- table オブジェクトに対してクエリを実行して出力テーブルを作成します。
- StatementSet を使用して出力テーブルを宛先に書き込みます。

Apache Flink 用 Managed Serviceで Python テーブル API を使い始めるには、Amazon Managed Service for Apache Flink for Python の開始方法 を参照してください。

ストリーミングデータの読み取りと書き込み

ストリーミングデータを読み書きするには、テーブル環境で SQL クエリを実行します。

テーブルの作成

次のコード例は、SQL クエリを作成するユーザー定義関数を示しています。SQL クエリは Kinesis ストリームと相互作用するテーブルを作成します。

```
def create_table(table_name, stream_name, region, stream_initpos):
   return """ CREATE TABLE {0} (
                `record_id` VARCHAR(64) NOT NULL,
                `event_time` BIGINT NOT NULL,
                `record_number` BIGINT NOT NULL,
                `num_retries` BIGINT NOT NULL,
                `verified` BOOLEAN NOT NULL
              )
              PARTITIONED BY (record_id)
              WITH (
                'connector' = 'kinesis',
                'stream' = '{1}',
                'aws.region' = '{2}',
                'scan.stream.initpos' = '{3}',
                'sink.partitioner-field-delimiter' = ';',
                'sink.producer.collection-max-count' = '100',
                'format' = 'json',
                'json.timestamp-format.standard' = 'ISO-8601'
              ) """.format(table_name, stream_name, region, stream_initpos)
```

ストリーミングデータの読み取り

次のコード例は、前述の CreateTable SQL クエリをテーブル環境参照に対して使用してデータを読み取る方法を示しています。

```
  table_env.execute_sql(create_table(input_table, input_stream, input_region,
stream_initpos))
```

ストリーミングデータの書き込み

次のコード例は、CreateTable 例の SQL クエリを使用して出力テーブル参照を作成する方法と、StatementSetを使用してテーブルを操作して宛先の Kinesis ストリームにデータを書き込む方法を示しています。

```
  table_result = table_env.execute_sql("INSERT INTO {0} SELECT * FROM {1}"
                    .format(output_table_name, input_table_name))
```

ランタイムプロパティの読み取り

ランタイムプロパティを使用すると、アプリケーションコードを変更せずにアプリケーションを設定できます。

アプリケーションのアプリケーションプロパティは、Java アプリケーション向けの Apache Flink 用 Managed Service と同じ方法で指定します。ランタイムプロパティは次の方法で指定できます。

- CreateApplication アクションの使用。
- UpdateApplication アクションの使用。
- コンソールを使ってアプリケーションを設定します。

コード内でアプリケーション・プロパティを取得するには、Managed Service for Apache Flinkランタイムが作成する application_properties.json と呼ばれるjsonファイルを読み込みます。

以下のサンプルコードは、application_properties.json ファイルからのアプリケーションプロパティの読み取りの例です。

```
file_path = '/etc/flink/application_properties.json'
   if os.path.isfile(file_path):
       with open(file_path, 'r') as file:
           contents = file.read()
           properties = json.loads(contents)
```

次のユーザー定義関数のコード例は、アプリケーションプロパティオブジェクト:retrieves からプロパティグループを読み取る方法を示しています。

```
def property_map(properties, property_group_id):
   for prop in props:
       if prop["PropertyGroupId"] == property_group_id:
           return prop["PropertyMap"]
```

次のコード例は、前の例で返されたプロパティグループから INPUT_STREAM_KEY というプロパティを読み取る方法を示しています。

```
input_stream = input_property_map[INPUT_STREAM_KEY]
```

アプリケーションのコードパッケージの作成

Python アプリケーションを作成したら、コードファイルと依存関係を zip ファイルにバンドルします。

zip ファイルには `main` メソッドを含む Python スクリプトが含まれている必要があり、オプションで以下を含めることができます。

- その他の Python コードファイル
- JAR ファイル内のユーザー定義 Java コード
- JAR ファイル内の Java ライブラリ

> **Note**
>
> アプリケーションの ZIP ファイルには、アプリケーションの依存関係がすべて含まれている必要があります。アプリケーションの他のソースからのライブラリを参照することはできません。

Managed Service for Apache Flink Python アプリケーションの作成

コードファイルの指定

アプリケーションのコードパッケージを作成したら、Amazon S3 バケットにアップロードします。次に、コンソールまたは CreateApplicationアクションを使用してアプリケーションを作成します。

CreateApplication アクションを使用してアプリケーションを作成するときは、 という特別なアプリケーションプロパティグループを使用して、zip ファイル内のコードファイルとアーカイブを指定しますkinesis.analytics.flink.run.options。以下のタイプファイルを定義できます。

- 「python」: Python のメインメソッドを含むテキストファイル。
- 「jarfile」:Java ユーザー定義関数を含む Java JAR ファイル。
- 「pyFiles」: アプリケーションが使用するリソースを含む Python リソースファイル。
- 「pyArchives:」 アプリケーションのリソースファイルを含む zip ファイル。

Apache Flink Python コードファイルタイプの詳細については、「Apache Flink ドキュメント」の「コマンドラインインターフェイス」を参照してください。

> **Note**
>
> Apache Flink のマネージドサービスは、pyModule、pyExecutable または pyRequirements ファイルタイプをサポートしていません。コード、要件、依存関係はすべて zip ファイルに含まれている必要があります。pip を使用してインストールする依存関係を指定することはできません。

次の JSON スニペットの例は、アプリケーションの zip ファイル内のファイルの場所を指定する方法を示しています。

```
"ApplicationConfiguration": {
    "EnvironmentProperties": {
      "PropertyGroups": [
        {
          "PropertyGroupId": "kinesis.analytics.flink.run.options",
          "PropertyMap": {
            "python": "MyApplication/main.py",
            "jarfile": "MyApplication/lib/myJarFile.jar",
            "pyFiles": "MyApplication/lib/myDependentFile.py",
            "pyArchives": "MyApplication/lib/myArchive.zip"
          }
        },
```

Python Managed Service for Apache Flink アプリケーションのモニタリング

アプリケーションの CloudWatch ログを使用して、Managed Service for Apache Flink Python アプリケーションをモニタリングします。

Apache Flink 用 Managed Service は Python アプリケーションの以下のメッセージをログに記録します。

- アプリケーションの `main` メソッドで `print()` を使用してコンソールに書き込まれるメッセージ。
- `logging` パッケージを使用してユーザー定義関数で送信されるメッセージ。次のコード例は、ユーザー定義関数からアプリケーションログへの書き込みを示しています。

```
import logging

@udf(input_types=[DataTypes.BIGINT()], result_type=DataTypes.BIGINT())
def doNothingUdf(i):
    logging.info("Got {} in the doNothingUdf".format(str(i)))
    return i
```

- アプリケーションから返されるエラーメッセージ。

  アプリケーションが `main` 関数内で例外を投げると、その例外はアプリケーションのログに記録されます。

  次の例は、Python コードから発生した例外のログエントリを示しています。

```
2021-03-15 16:21:20.000   --------------------------- Python Process Started
 --------------------------
2021-03-15 16:21:21.000   Traceback (most recent call last):
2021-03-15 16:21:21.000   "  File ""/tmp/flink-
web-6118109b-1cd2-439c-9dcd-218874197fa9/flink-web-upload/4390b233-75cb-4205-
a532-441a2de83db3_code/PythonKinesisSink/PythonUdfUndeclared.py"", line 101, in
 <module>"
2021-03-15 16:21:21.000       main()
2021-03-15 16:21:21.000   "  File ""/tmp/flink-
web-6118109b-1cd2-439c-9dcd-218874197fa9/flink-web-upload/4390b233-75cb-4205-
a532-441a2de83db3_code/PythonKinesisSink/PythonUdfUndeclared.py"", line 54, in main"
2021-03-15 16:21:21.000   "    table_env.register_function(""doNothingUdf"",
 doNothingUdf)"
```

```
2021-03-15 16:21:21.000   NameError: name 'doNothingUdf' is not defined
2021-03-15 16:21:21.000   ---------------------------- Python Process Exited
 ----------------------------
2021-03-15 16:21:21.000   Run python process failed
2021-03-15 16:21:21.000   Error occurred when trying to start the job
```

> **Note**
>
> パフォーマンス上の問題から、アプリケーション開発中はカスタムログメッセージのみを使用することをおすすめします。

CloudWatch Insights を使用したログのクエリ

次の CloudWatch Insights クエリは、アプリケーションの主要関数の実行中に Python エントリポイントによって作成されたログを検索します。

```
fields @timestamp, message
| sort @timestamp asc
| filter logger like /PythonDriver/
| limit 1000
```

Managed Service for Apache Flink のランタイムプロパティ

「ランタイムプロパティ」を使用すると、アプリケーションコードを再コンパイルせずにアプリケーションを設定できます。

このトピックには、次のセクションが含まれています。

- コンソールでのランタイムプロパティの使用
- CLI でのランタイムプロパティの使用
- Managed Service for Apache Flink アプリケーションのランタイムプロパティへのアクセス

コンソールでのランタイムプロパティの使用

を使用して、Managed Service for Apache Flink アプリケーションからランタイムプロパティを追加、更新、または削除できます AWS Management Console。

> **Note**
>
> 以前にサポートされているバージョンの Apache Flink を使用していて、既存のアプリケーションを Apache Flink 1.19.1 にアップグレードする場合は、インプレース Apache Flink バージョンアップグレードを使用してアップグレードできます。インプレースバージョンアップグレードでは、スナップショット、ログ、メトリクス、タグ、Flink 設定など、Apache Flink バージョン全体で 1 つの ARN に対するアプリケーションのトレーサビリティを維持します。この機能は RUNNINGおよび READY状態で使用できます。詳細については、「Apache Flink のインプレースバージョンアップグレード」を参照してください。

Apache Flink アプリケーション用 Managed Serviceのランタイムプロパティの更新

1. https://console.aws.amazon.com/flink で Apache Flink 用 Managed Serviceコンソールを開く
2. Apache Flink アプリケーション用 Managed Serviceを選択します。[Application details (アプリケーションの詳細)] を選択します。
3. アプリケーションのページで、構成 をクリックします。
4. 「プロパティ」セクションを展開します。
5. 「プロパティ」セクションのコントロールを使用して、キーと値のペアを含むプロパティグループを定義します。これらのコントロールを使用して、プロパティグループとランタイムプロパティを追加、更新、削除します。
6. [更新] を選択します。

CLI でのランタイムプロパティの使用

「AWS CLI」を使用してランタイムプロパティを追加、更新、削除できます。

このセクションには、アプリケーションのランタイムプロパティを設定するための API アクションのリクエスト例が含まれています。JSON ファイルを API アクションの入力に使用する方法の詳細については、Apache Flink API のマネージドサービスのサンプルコード を参照してください。

> **Note**
>
> 次の例のサンプルのアカウント ID (*012345678901*) をアカウント ID に置き換えます。

アプリケーション作成時のランタイムプロパティの追加

以下の「CreateApplication」アクションリクエスト例では、アプリケーションの作成時に 2 つのランタイムプロパティグループ (ProducerConfigProperties と ConsumerConfigProperties) を追加します。

```
{
    "ApplicationName": "MyApplication",
    "ApplicationDescription": "my java test app",
    "RuntimeEnvironment": "FLINK-1_19",
    "ServiceExecutionRole": "arn:aws:iam::012345678901:role/MF-stream-rw-role",
    "ApplicationConfiguration": {
        "ApplicationCodeConfiguration": {
            "CodeContent": {
                "S3ContentLocation": {
                    "BucketARN": "arn:aws:s3:::ka-app-code-username",
                    "FileKey": "java-getting-started-1.0.jar"
                }
            },
            "CodeContentType": "ZIPFILE"
        },
        "EnvironmentProperties":  {
         "PropertyGroups": [
            {
               "PropertyGroupId": "ProducerConfigProperties",
               "PropertyMap" : {
                    "flink.stream.initpos" : "LATEST",
                    "aws.region" : "us-west-2",
                    "AggregationEnabled" : "false"
               }
            },
            {
               "PropertyGroupId": "ConsumerConfigProperties",
               "PropertyMap" : {
                    "aws.region" : "us-west-2"
               }
            }
         ]
      }
    }
}
```

既存のアプリケーションでのランタイムプロパティの追加と更新

以下の「UpdateApplication」アクションリクエスト例は、既存のアプリケーションのランタイムプロパティを追加または更新します。

```
{
  "ApplicationName": "MyApplication",
  "CurrentApplicationVersionId": 2,
  "ApplicationConfigurationUpdate": {
    "EnvironmentPropertyUpdates": {
      "PropertyGroups": [
        {
          "PropertyGroupId": "ProducerConfigProperties",
          "PropertyMap" : {
            "flink.stream.initpos" : "LATEST",
            "aws.region" : "us-west-2",
            "AggregationEnabled" : "false"
          }
        },
        {
          "PropertyGroupId": "ConsumerConfigProperties",
          "PropertyMap" : {
            "aws.region" : "us-west-2"
          }
        }
      ]
    }
  }
}
```

> **Note**
>
> プロパティグループに対応するランタイムプロパティがないキーを使用すると、Apache Flink 用 Managed Service はそのキーと値のペアを新しいプロパティとして追加します。プロパティグループ内の既存のランタイムプロパティのキーを使用すると、Apache Flink 用 Managed Service はそのプロパティ値を更新します。

ランタイムプロパティの削除

以下の「UpdateApplication」アクションリクエスト例では、既存のアプリケーションからすべてのランタイムプロパティとプロパティグループを削除します。

```
{
  "ApplicationName": "MyApplication",
  "CurrentApplicationVersionId": 3,
  "ApplicationConfigurationUpdate": {
    "EnvironmentPropertyUpdates": {
      "PropertyGroups": []
    }
  }
}
```

> **Important**
>
> 既存のプロパティグループまたはプロパティグループ内の既存のプロパティキーを省略すると、そのプロパティグループまたはプロパティは削除されます。

Managed Service for Apache Flink アプリケーションのランタイムプロパティへのアクセス

Java アプリケーションコード内のランタイムプロパティは、`Map<String, Properties>` オブジェクトを返す静的 `KinesisAnalyticsRuntime.getApplicationProperties()` メソッドを使用して取得します。

次の Java コードの例では、アプリケーションのランタイムプロパティを取得します。

```
Map<String, Properties> applicationProperties =
KinesisAnalyticsRuntime.getApplicationProperties();
```

プロパティグループを (`Java.Util.Properties` オブジェクトとして) 次のように取得します。

```
Properties consumerProperties = applicationProperties.get("ConsumerConfigProperties");
```

Apache Flink のソースまたはシンクは、通常、個々のプロパティを取得せずに Properties オブジェクトを渡すことで設定します。以下のコード例は、ランタイムプロパティから取得した Properties オブジェクトを渡して Flink ソースを作成する方法を示しています。

```
private static FlinkKinesisProducer<String> createSinkFromApplicationProperties()
 throws IOException {
  Map<String, Properties> applicationProperties =
 KinesisAnalyticsRuntime.getApplicationProperties();
  FlinkKinesisProducer<String> sink = new FlinkKinesisProducer<String>(new
 SimpleStringSchema(),
    applicationProperties.get("ProducerConfigProperties"));

  sink.setDefaultStream(outputStreamName);
  sink.setDefaultPartition("0");
  return sink;
}
```

コード例については、Managed Service for Apache Flink：例 を参照してください。

Apache Flink コネクタの使用

Apache Flink コネクタは、Amazon Managed Service for Apache Flink アプリケーションとの間でデータを移動するソフトウェアコンポーネントです。コネクタは、ファイルやディレクトリから読み取ることができる柔軟な統合です。コネクタは、Amazon のサービスやサードパーティのシステムとやり取りするための完全なモジュールで構成されています。

コネクターの種類には、次のものがあります。

- ソース： Kinesis データストリーム、 ファイル、Apache Kafka トピック、 ファイル、またはその他のデータソースからアプリケーションにデータを提供します。
- シンク： アプリケーションから Kinesis データストリーム、Firehose ストリーム、Apache Kafka トピック、またはその他のデータ送信先にデータを送信します。
- 非同期 I/O: データベースなどのデータソースへの非同期アクセスを提供し、ストリームを強化します。

Apache Flink コネクタは、独自のソースリポジトリに保存されます。Apache Flink コネクタのバージョンとアーティファクトは、使用している Apache Flink のバージョン、および DataStream、テーブル、または SQL API を使用しているかどうかに応じて変わります。

Amazon Managed Service for Apache Flink は、40 を超える構築済みの Apache Flink ソースおよびシンクコネクタをサポートしています。次の表は、最も一般的なコネクタとそれに関連するバージョンの概要を示しています。非同期シンクフレームワークを使用してカスタムシンクを構築することもできます。詳細については、Apache Flink ドキュメントの「汎用非同期ベースシンク」を参照してください。

Apache Flink AWS コネクタのリポジトリにアクセスするには、「」を参照してくださいflink-connector-aws。

Flink バージョンのコネクタ

コネクタ	Flink バージョン 1.15	Flink バージョン 1.18	Flink バージョン 1.19
Kinesis Data Stream - ソース - DataStream およびテーブル API	flink-connector-kinesis、1.15.4	flink-connector-kinesis、4.3.0 ~ 1.18	flink-connector-kinesis、4.3.0 ~ 1.19
Kinesis Data Stream - シンク - DataStream テーブル API	flink-connector-aws-kinesis-streams、1.15.4	flink-connector-aws-kinesis-streams、4.3.0 ~ 1.18	flink-connector-aws-kinesis-streams、4.3.0 ~ 1.19
Kinesis Data Streams - ソース/シンク - SQL	flink-sql-connector-kinesis、1.15.4	flink-sql-connector-kinesis、4.3.0 ~ 1.18	flink-sql-connector-kinesis、4.3.0 ~ 1.19
Kafka - DataStream およびテーブル API	flink-connector-kafka、1.15.4	flink-connector-kafka、3.2.0 ~ 1.18	flink-connector-kafka、3.2.0 ~ 1.19
Kafka - SQL	flink-sql-connector-kafka、1.15.4	flink-sql-connector-kafka、3.2.0 ~ 1.18	flink-sql-connector-kafka、3.2.0 ~ 1.19
Firehose - DataStream およびテーブル API	flink-connector-aws-kinesis-firehose、1.15.4	flink-connector-aws-firehose、4.3.0 ~ 1.18	flink-connector-aws-firehose、4.3.0 ~ 1.19
Firehose - SQL	flink-sql-connector-aws-kinesis-firehose、1.15.4	flink-sql-connector-aws-firehose、4.3.0-1.18	flink-sql-connector-aws-firehose、4.3.0-1.19

コネクタ	Flink バージョン 1.15	Flink バージョン 1.18	Flink バージョン 1.19
DynamoDB - DataStream および テーブル API	flink-connector-dynamodb、3.0.0 ~ 1.15	flink-connector-dynamodb、4.3.0 ~ 1.18	flink-connector-dynamodb、4.3.0 ~ 1.19
DynamoDB - SQL	flink-sql-connector-dynamodb、3.0.0 ~ 1.15	flink-sql-connector-dynamodb、4.3.0 ~ 1.18	flink-sql-connector-dynamodb、4.3.0 ~ 1.19
OpenSearch DataStream および テーブル API	-	flink-connector-opensearch、1.2.0 ~ 1.18	flink-connector-opensearch、1.2.0 ~ 1.19
OpenSearch - SQL	-	flink-sql-connector-opensearch、1.2.0 ~ 1.18	flink-sql-connector-opensearch、1.2.0 ~ 1.19

Amazon Managed Service for Apache Flink のコネクタの詳細については、以下を参照してください。

- DataStream API コネクタ
- テーブル API コネクタ

Managed Service for Apache Flink での耐障害性の実装

チェックポインティングは、Amazon Managed Service for Apache Flink でフォールトトレランスを実装するために使用される方法です。チェックポイントは、実行中のアプリケーションの up-to-date バックアップであり、予期しないアプリケーションの中断やフェイルオーバーからすぐに復旧するために使用されます。

Apache Flink アプリケーションのチェックポイントの詳細については、Apache Flink ドキュメントの「チェックポイント」を参照してください。

「スナップショット」は、アプリケーションの状態を手動で作成して管理するバックアップです。スナップショットを使うと、「`UpdateApplication`」の呼び出しによってアプリケーションを以前

の状態に復元できます。詳細については、「スナップショットを使用したアプリケーションのバックアップの管理」を参照してください。

アプリケーションのチェックポイント機能が有効になっていると、アプリケーションが予期せず再起動した場合にアプリケーションデータのバックアップを作成して読み込むことができるため、耐障害性が確保されます。このような予期しないアプリケーションの再起動は、ジョブの予期しない再起動、インスタンスの障害などが原因である可能性があります。これにより、アプリケーションは、これらの再起動時に無障害実行と同じセマンティクスを持つことになります。

スナップショットがアプリケーションに対して有効で、アプリケーションの を使用して設定されている場合ApplicationRestoreConfiguration、サービスはアプリケーションの更新中、またはサービス関連のスケーリングまたはメンテナンス中に 1 回のみ処理セマンティクスを提供します。

Managed Service for Apache Flink でのチェックポイントの設定

アプリケーションのチェックポインティング動作を構成できます。チェックポイントの状態を持続させるかどうか、チェックポイントに状態を保存する頻度、1 つのチェックポイント操作の終了から別のチェックポイント操作の開始までの最小間隔を定義できます。

「`CreateApplication`」または「`UpdateApplication`」API 操作を使用して次の設定を行います。

- 「`CheckpointingEnabled`」— アプリケーションでチェックポイントが有効になっているかどうかを示します。
- 「`CheckpointInterval`」— チェックポイント (パーシスタンス) 操作の間隔をミリ秒単位で含みます。
- 「`ConfigurationType`」— デフォルトのチェックポイント動作を使用するには、この値を「`DEFAULT`」に設定します。この値を「`CUSTOM`」に設定すると、他の値を設定できます。

> **Note**
>
> デフォルトのチェックポイント動作は次のとおりです。
>
> - CheckpointingEnabled： true
> - CheckpointInterval： 60,000
> - MinPauseBetweenCheckpoints： 5000

ConfigurationType が に設定されている場合DEFAULT、 を使用するか、アプリケーションコードの値を設定して他の値に設定されていても AWS Command Line Interface、前述の値が使用されます。

> **Note**
>
> Flink 1.15 以降では、Apache Flink 用 Managed Serviceは、アプリケーションの更新、スケーリング、停止などの自動スナップショット作成時に `stop-with-savepoint` を使用します。

- `MinPauseBetweenCheckpoints` — 1 つのチェックポイント操作が終了してから別のチェックポイント操作が開始されるまでの最小時間 (ミリ秒単位)。この値を設定すると、チェックポイントオペレーションが `CheckpointInterval` よりも時間がかかる場合でも、アプリケーションは継続的にチェックポイント機能を実行できなくなります。

チェックポイント API の例

このセクションには、アプリケーションのチェックポイントを構成するための API アクションのリクエスト例が含まれています。JSON ファイルを API アクションの入力に使用する方法の詳細については、Apache Flink API のマネージドサービスのサンプルコード を参照してください。

新しいアプリケーションのチェックポイントを設定する

以下の「`CreateApplication`」アクションのリクエスト例では、アプリケーションの作成時にチェックポインティングを設定しています。

```
{
   "ApplicationName": "MyApplication",
   "RuntimeEnvironment":"FLINK-1_19",
   "ServiceExecutionRole":"arn:aws:iam::123456789123:role/myrole",
   "ApplicationConfiguration": {
      "ApplicationCodeConfiguration":{
      "CodeContent":{
        "S3ContentLocation":{
          "BucketARN":"arn:aws:s3:::mybucket",
          "FileKey":"myflink.jar",
          "ObjectVersion":"AbCdEfGhIjKlMnOpQrStUvWxYz12345"
        }
```

```
      },
      "FlinkApplicationConfiguration": {
         "CheckpointConfiguration": {
            "CheckpointingEnabled": "true",
            "CheckpointInterval": 20000,
            "ConfigurationType": "CUSTOM",
            "MinPauseBetweenCheckpoints": 10000
         }
      }
}
```

新しいアプリケーションのチェックポイントを無効にする

以下の「CreateApplication」アクションのリクエスト例では、アプリケーションの作成時にチェックポインティングを無効にします。

```
{
   "ApplicationName": "MyApplication",
   "RuntimeEnvironment":"FLINK-1_19",
   "ServiceExecutionRole":"arn:aws:iam::123456789123:role/myrole",
   "ApplicationConfiguration": {
      "ApplicationCodeConfiguration":{
      "CodeContent":{
        "S3ContentLocation":{
          "BucketARN":"arn:aws:s3:::mybucket",
          "FileKey":"myflink.jar",
          "ObjectVersion":"AbCdEfGhIjKlMnOpQrStUvWxYz12345"
        }
      },
      "FlinkApplicationConfiguration": {
         "CheckpointConfiguration": {
            "CheckpointingEnabled": "false"
         }
      }
}
```

既存のアプリケーションのチェックポイントを設定する

以下の「UpdateApplication」アクションリクエスト例では、既存のアプリケーションのチェックポインティングを設定しています。

```
{
```

```
   "ApplicationName": "MyApplication",
   "ApplicationConfigurationUpdate": {
      "FlinkApplicationConfigurationUpdate": {
         "CheckpointConfigurationUpdate": {
            "CheckpointingEnabledUpdate": true,
            "CheckpointIntervalUpdate": 20000,
            "ConfigurationTypeUpdate": "CUSTOM",
            "MinPauseBetweenCheckpointsUpdate": 10000
         }
      }
   }
}
```

既存のアプリケーションのチェックポイントを無効にする

以下の「UpdateApplication」アクションのリクエスト例では、既存のアプリケーションのチェックポインティングを無効にします。

```
{
   "ApplicationName": "MyApplication",
   "ApplicationConfigurationUpdate": {
      "FlinkApplicationConfigurationUpdate": {
         "CheckpointConfigurationUpdate": {
            "CheckpointingEnabledUpdate": false,
            "CheckpointIntervalUpdate": 20000,
            "ConfigurationTypeUpdate": "CUSTOM",
            "MinPauseBetweenCheckpointsUpdate": 10000
         }
      }
   }
}
```

スナップショットを使用したアプリケーションのバックアップの管理

「スナップショットは」、Apache Flink 「セーブポイント」の Apache Flink 用 Managed Service 実装です。スナップショットは、ユーザーまたはサービスによってトリガーされ、作成され、管理されるアプリケーション状態のバックアップです。Apache Flink セーブポイントの詳細については、「Apache Flink ドキュメント」の「セーブポイント」を参照してください。スナップショットを使

用すると、アプリケーション状態の特定のスナップショットからアプリケーションを再起動できます。

> **Note**
>
> アプリケーションが正しい状態データで正しく再起動できるように、1 日に数回スナップショットを作成することをおすすめします。スナップショットの正しい頻度は、アプリケーションのビジネスロジックによって異なります。スナップショットを頻繁に作成すると、より最近のデータを復元できますが、コストが増加し、より多くのシステムリソースが必要になります。

Apache Flink 用 Managed Serviceでは、次の API アクションを使用してスナップショットを管理します。

- CreateApplicationSnapshot
- DeleteApplicationSnapshot
- DescribeApplicationSnapshot
- ListApplicationSnapshots

アプリケーションごとのスナップショット数の制限については、クォータ を参照してください。アプリケーションがスナップショットの上限に達すると、スナップショットを手動で作成すると失敗し、「LimitExceededException」が表示されます。

Apache Flink 用 Managed Serviceは決してスナップショットを削除しません。これらのスナップショットは、 DeleteApplicationSnapshot アクションを使用して手動で削除する必要があります。

アプリケーションの起動時に、保存されているアプリケーションの状態のスナップショットを読み込むには、「StartApplication」または「UpdateApplication」アクションの「ApplicationRestoreConfiguration」パラメータを使用します。

このトピックには、次のセクションが含まれています。

- スナップショットの自動作成
- 互換性のない状態データを含むスナップショットからの復元
- スナップショット API の例

スナップショットの自動作成

ApplicationSnapshotConfiguration アプリケーションの `true`で `SnapshotsEnabled`が に設定されている場合、Managed Service for Apache Flink は、アプリケーションが更新、スケーリング、または停止されると、スナップショットを自動的に作成して使用し、厳密に 1 回限りの処理セマンティクスを提供します。

> **Note**
>
> `ApplicationSnapshotConfiguration::SnapshotsEnabled` が `false` に設定されると、アプリケーションの更新中にデータが失われます。

> **Note**
>
> Apache Flink 用 Managed Serviceは、スナップショット作成中に中間セーブポイントをトリガーします。Flink バージョン 1.15 以降では、中間セーブポイントによる副作用は発生しなくなりました。「セーブポイントのトリガー」を参照してください。

自動的に作成されたスナップショットには以下の特性があります。

- スナップショットは サービスによって管理されますが、 ListApplicationSnapshotsアクションを使用してスナップショットを表示できます。自動的に作成されたスナップショットは、スナップショットの制限に含まれます。
- アプリケーションがスナップショットの制限を超えると、手動で作成したスナップショットは失敗しますが、Apache Flink 用 Managed Service サービスは、アプリケーションの更新、スケーリング、または停止時に引き続き正常にスナップショットを作成します。スナップショットを手動で作成する前に、 DeleteApplicationSnapshotアクションを使用してスナップショットを手動で削除する必要があります。

互換性のない状態データを含むスナップショットからの復元

スナップショットにはオペレータに関する情報が含まれているため、以前のアプリケーションバージョン以降に変更されたオペレータの状態データをスナップショットから復元すると、予期しない結果が生じることがあります。現在のオペレータに対応していないスナップショットから状態データを

復元しようとすると、アプリケーションに障害が発生します。障害が発生したアプリケーションは、「STOPPING」または「UPDATING」のいずれかの状態のままになります。

互換性のない状態データを含むスナップショットからアプリケーションが復元できるようにするには、 UpdateApplicationアクションtrueを使用して の AllowNonRestoredStateパラメータFlinkRunConfigurationを に設定します。

古いスナップショットからアプリケーションを復元すると、次のような動作になります。

- 「オペレータ追加:」新しいオペレータが追加されても、セーブポイントには新しいオペレータの状態データはありません。障害は発生せず、「AllowNonRestoredState」を設定する必要はありません。
- 「オペレータが削除された:」既存のオペレータが削除されると、そのオペレータの状態データがセーブポイントに格納されます。AllowNonRestoredState が true に設定されていないと障害が発生します。
- 「オペレータ修正:」パラメータのタイプを互換性のあるタイプに変更するなど、互換性のある変更が行われた場合、アプリケーションは古いスナップショットから復元できます。スナップショットからの復元の詳細については、Apache Flink ドキュメントの「セーブポイント」を参照してください。Apache Flink バージョン 1.8 以降を使用するアプリケーションは、別のスキーマのスナップショットから復元できる可能性があります。Apache Flink バージョン 1.6 を使用するアプリケーションは復元できません。 two-phase-commit シンクの場合は、ユーザーが作成したスナップショット () の代わりにシステムスナップショット (SwS) を使用することをお勧めします CreateApplicationSnapshot。

  Flink の場合、Apache Flink 用 Managed Serviceは、スナップショットの作成中に中間セーブポイントをトリガーします。Flink 1.15 以降では、中間セーブポイントによる副作用は発生しなくなりました。「セーブポイントのトリガー」を参照してください。

既存のセーブポイントデータと互換性のないアプリケーションを再開する必要がある場合は、StartApplicationアクションの ApplicationRestoreTypeパラメータを に設定して、スナップショットからの復元をスキップすることをお勧めしますSKIP_RESTORE_FROM_SNAPSHOT。

Apache Flink が互換性のない状態データを処理する方法の詳細については、「Apache Flink ドキュメント」の「状態スキーマ進化」を参照してください。

スナップショット API の例

このセクションには、アプリケーションでスナップショットを使用するための API アクションのリクエスト例が含まれています。JSON ファイルを API アクションの入力に使用する方法の詳細については、Apache Flink API のマネージドサービスのサンプルコード を参照してください。

アプリケーションのスナップショットを有効にする

UpdateApplication アクションの以下のリクエスト例は、アプリケーションのスナップショットを有効にします。

```
{
   "ApplicationName": "MyApplication",
   "CurrentApplicationVersionId": 1,
   "ApplicationConfigurationUpdate": {
      "ApplicationSnapshotConfigurationUpdate": {
         "SnapshotsEnabledUpdate": "true"
       }
    }
}
```

スナップショットを作成する

以下の「CreateApplicationSnapshot」アクションのリクエスト例では、現在のアプリケーション状態のスナップショットを作成します。

```
{
   "ApplicationName": "MyApplication",
   "SnapshotName": "MyCustomSnapshot"
}
```

アプリケーションのスナップショットを一覧表示する

以下の「ListApplicationSnapshots」アクションリクエスト例では、現在のアプリケーション状態の最初の 50 個のスナップショットが一覧表示されます。

```
{
   "ApplicationName": "MyApplication",
   "Limit": 50
```

```
}
```

アプリケーションスナップショットの詳細を一覧表示する

DescribeApplicationSnapshot アクションの以下のリクエスト例では、特定のアプリケーションスナップショットの詳細を一覧表示します。

```
{
   "ApplicationName": "MyApplication",
   "SnapshotName": "MyCustomSnapshot"
}
```

スナップショットを削除する

以下の「DeleteApplicationSnapshot」アクションリクエスト例では、以前に保存したスナップショットを削除します。SnapshotCreationTimestamp 値は、「ListApplicationSnapshots」または「DeleteApplicationSnapshot」を使用して取得できます。

```
{
   "ApplicationName": "MyApplication",
   "SnapshotName": "MyCustomSnapshot",
   "SnapshotCreationTimestamp": 12345678901.0,
}
```

名前付きスナップショットを使用してアプリケーションを再起動する

以下の「StartApplication」アクションリクエスト例では、特定のスナップショットから保存された状態を使用してアプリケーションを起動します。

```
{
   "ApplicationName": "MyApplication",
   "RunConfiguration": {
      "ApplicationRestoreConfiguration": {
         "ApplicationRestoreType": "RESTORE_FROM_CUSTOM_SNAPSHOT",
         "SnapshotName": "MyCustomSnapshot"
      }
   }
}
```

最新のスナップショットを使用してアプリケーションを再起動する

以下の「StartApplication」アクションリクエスト例では、最新のスナップショットを使用してアプリケーションを起動します。

```
{
   "ApplicationName": "MyApplication",
   "RunConfiguration": {
      "ApplicationRestoreConfiguration": {
         "ApplicationRestoreType": "RESTORE_FROM_LATEST_SNAPSHOT"
      }
   }
}
```

スナップショットなしでアプリケーションを再起動する

以下の「StartApplication」アクションのリクエスト例では、スナップショットがあってもアプリケーションの状態をロードせずにアプリケーションを起動します。

```
{
   "ApplicationName": "MyApplication",
   "RunConfiguration": {
      "ApplicationRestoreConfiguration": {
         "ApplicationRestoreType": "SKIP_RESTORE_FROM_SNAPSHOT"
      }
   }
}
```

Apache Flink のインプレースバージョンアップグレード

Apache Flink のインプレースバージョンアップグレードでは、Apache Flink バージョン全体で単一の ARN に対するアプリケーションのトレーサビリティを維持します。これには、スナップショット、ログ、メトリクス、タグ、Flink 設定、リソース制限の引き上げ、VPCsなどが含まれます。Apache Flink のインプレースバージョンアップグレードを実行して、既存のアプリケーションを Amazon Managed Service for Apache Flink の新しい Flink バージョンにアップグレードできます。このタスクを実行するには、、 AWS CLI、 SDK AWS CloudFormation、 AWS または を使用できます AWS Management Console。

> **Note**
>
> Amazon Managed Service for Apache Flink Studio では、Apache Flink のインプレースバージョンアップグレードを使用できません。

このトピックには、次のセクションが含まれています。

- Apache Flink のインプレースバージョンアップグレードを使用したアプリケーションのアップグレード
- アプリケーションを新しい Apache Flink バージョンにアップグレードする
- ロールバック
- 一般的なベストプラクティスと推奨事項
- 注意事項と既知の問題

Apache Flink のインプレースバージョンアップグレードを使用したアプリケーションのアップグレード

開始する前に、「インプレースバージョンアップグレード」の動画を視聴することをお勧めします。

Apache Flink のインプレースバージョンアップグレードを実行するには、、 AWS CLI AWS CloudFormation、 AWS SDK、または を使用できます AWS Management Console。この機能は、READYまたは RUNNING状態の Managed Service for Apache Flink で使用する既存のアプリケーションで使用できます。 UpdateApplication API を使用して、Flink ランタイムを変更する機能を追加します。

アップグレード前: Apache Flink アプリケーションの更新

Flink アプリケーションを記述するときは、アプリケーションの JAR に依存関係をバンドルし、その JAR を Amazon S3 バケットにアップロードします。そこから、Amazon Managed Service for Apache Flink は、選択した新しい Flink ランタイムでジョブを実行します。アップグレードする Flink ランタイムとの互換性を実現するために、アプリケーションを更新する必要がある場合があります。Flink バージョン間に不整合があり、バージョンアップグレードが失敗する可能性があります。最も一般的には、これは送信元 (進入) または送信先 (シンク、出力) と Scala の依存関係のコネクタを使用します。Managed Service for Apache Flink の Flink 1.15 以降のバージョンは Scala に依存しないため、JAR には使用する Scala のバージョンが含まれている必要があります。

アプリケーションを更新するには

1. 状態のアプリケーションのアップグレードに関する Flink コミュニティからのアドバイスをお読みください。「アプリケーションと Flink バージョンのアップグレード」を参照してください。
2. 既知の問題と制限事項のリストをお読みください。注意事項と既知の問題 を参照してください。
3. 依存関係を更新し、アプリケーションをローカルでテストします。通常、これらの依存関係は次のとおりです。

    1. Flink ランタイムと API。
    2. 新しい Flink ランタイムに推奨されるコネクタ。これらは、更新する特定のランタイムのリリースバージョンで確認できます。
    3. Scala – Apache Flink は、Flink 1.15 以降、Scala に依存しません。アプリケーション JAR で使用する Scala の依存関係を含める必要があります。
4. zipfile に新しいアプリケーション JAR を構築し、Amazon S3 にアップロードします。以前の JAR/zip ファイルとは異なる名前を使用することをお勧めします。ロールバックする必要がある場合は、この情報を使用します。
5. ステートフルアプリケーションを実行している場合は、現在のアプリケーションのスナップショットを作成することを強くお勧めします。これにより、アップグレード中またはアップグレード後に問題が発生した場合に、ステートリーにロールバックできます。

アプリケーションを新しい Apache Flink バージョンにアップグレードする

UpdateApplication アクションを使用して Flink アプリケーションをアップグレードできます。

`UpdateApplication` API は複数の方法で呼び出すことができます。

- で既存の設定ワークフローを使用します AWS Management Console。
  - のアプリページに移動します AWS Management Console。
  - [設定] を選択します。
  - 新しいランタイムと、復元設定とも呼ばれる、開始するスナップショットを選択します。最新のスナップショットからアプリケーションを起動するには、最新の設定を復元設定として使用します。Amazon S3 で新しくアップグレードされたアプリケーション JAR/zip をポイントします。
- AWS CLI update-application アクションを使用します。
- AWS CloudFormation (CFN) を使用します。
  - RuntimeEnvironment フィールドを更新します。以前は、 がアプリケーション AWS CloudFormation を削除し、新しいアプリケーションを作成していたため、スナップショットや

その他のアプリケーション履歴が失われていました。これで RuntimeEnvironment 、 が所定の位置に AWS CloudFormation 更新され、アプリケーションは削除されません。

- AWS SDK を使用します。
  - 選択したプログラミング言語については、 SDK のドキュメントを参照してください。「UpdateApplication」を参照してください。

アップグレードは、アプリケーションが RUNNING状態の間、またはアプリケーションが READY状態で停止している間に実行できます。Amazon Managed Service for Apache Flink は、元のランタイムバージョンとターゲットランタイムバージョンの互換性を検証します。この互換性チェックは、RUNNING状態UpdateApplicationの間に を実行すると が実行され、 READY状態の間にアップグレードStartApplicationすると次の で実行されます。

RUNNING 状態のアプリケーションのアップグレード

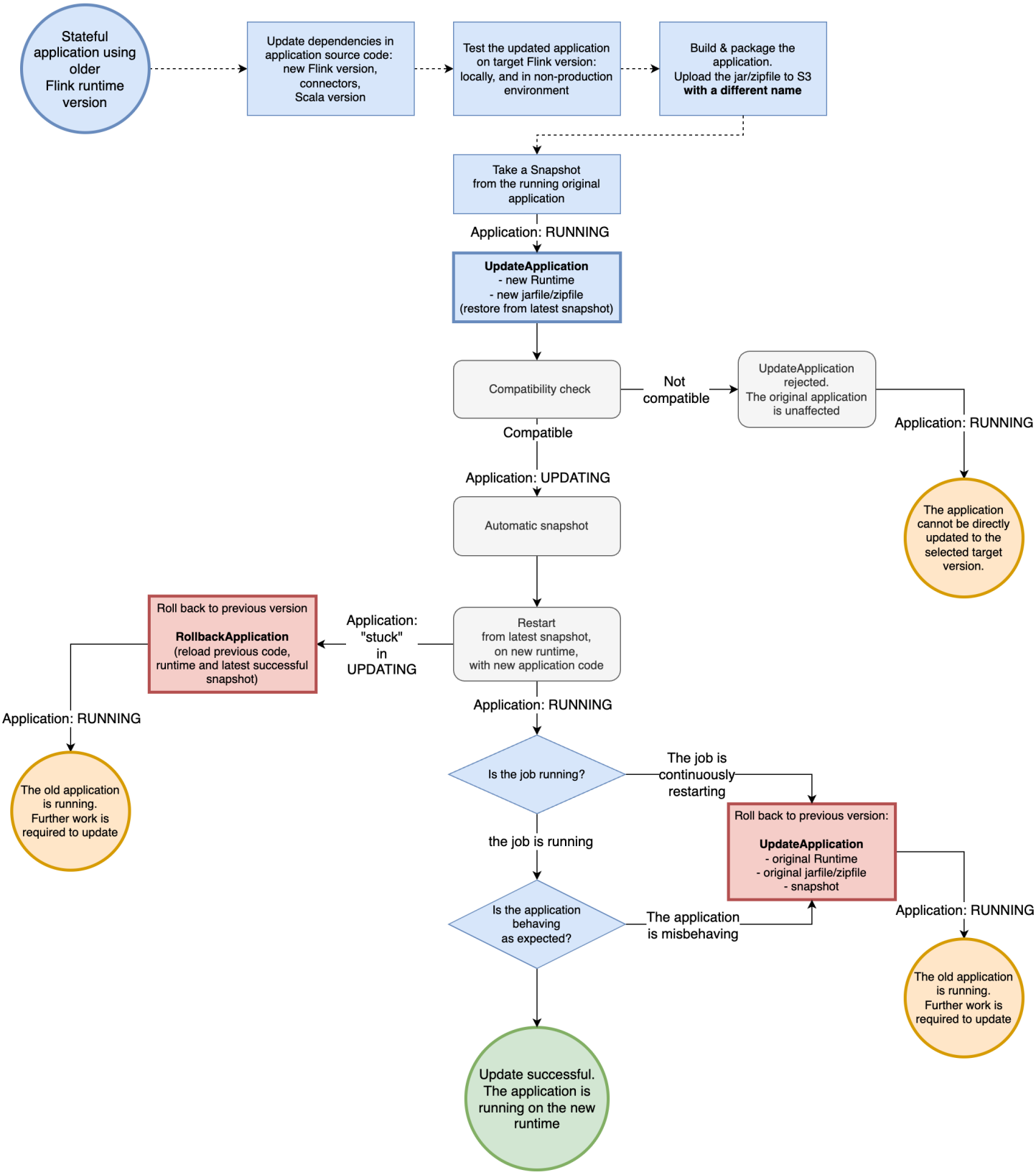
次の例は、 を使用して、 という名前RUNNINGのアプリケーションを米国東部 (バージニア北部) の UpgradeTest Flink 1.18 にアップグレード AWS CLI し、最新のスナップショットからアップグレードされたアプリケーションを起動することを示しています。

```
aws --region us-east-1 kinesisanalyticsv2 update-application \
--application-name UpgradeTest --runtime-environment-update "FLINK-1_18" \
--application-configuration-update '{"ApplicationCodeConfigurationUpdate": '\
'{"CodeContentUpdate": {"S3ContentLocationUpdate": '\
'{"FileKeyUpdate": "flink_1_18_app.jar"}}}}' \
 --run-configuration-update '{"ApplicationRestoreConfiguration": '\
 '{"ApplicationRestoreType": "RESTORE_FROM_LATEST_SNAPSHOT"}}' \
 --current-application-version-id ${current_application_version}
```

- サービススナップショットを有効にし、最新のスナップショットからアプリケーションを継続する場合、Amazon Managed Service for Apache Flink は、現在のRUNNINGアプリケーションのランタイムが選択したターゲットランタイムと互換性があることを確認します。
- ターゲットランタイムを継続するスナップショットを指定した場合、Amazon Managed Service for Apache Flink は、ターゲットランタイムが指定されたスナップショットと互換性があることを確認します。互換性チェックが失敗した場合、更新リクエストは拒否され、アプリケーションはRUNNING状態のままになります。

- スナップショットなしでアプリケーションを起動することを選択した場合、Amazon Managed Service for Apache Flink は互換性チェックを実行しません。
- アップグレードしたアプリケーションが失敗したり、推移的なUPDATING状態で停止したりした場合は、 ロールバックセクションの指示に従って正常な状態に戻ります。

ステートアプリケーションを実行するためのプロセスフロー


Stateful application using older Flink runtime version
Update dependencies in application source code: new Flink version, connectors, Scala version
Test the updated application on target Flink version: locally, and in non-production environment
Build & package the application. Upload the jar/zipfile to S3 with a different name
Take a Snapshot from the running original application
Application: RUNNING
UpdateApplication - new Runtime - new jarfile/zipfile (restore from latest snapshot)
Compatibility check
Not compatible
UpdateApplication rejected. The original application is unaffected
Application: RUNNING
The application cannot be directly updated to the selected target version.
Compatible
Application: UPDATING
Automatic snapshot
Restart from latest snapshot, on new runtime, with new application code
Application: "stuck" in UPDATING
Roll back to previous version
RollbackApplication (reload previous code, runtime and latest successful snapshot)
Application: RUNNING
The old application is running. Further work is required to update
Application: RUNNING
Is the job running?
The job is continuously restarting
Roll back to previous version:
UpdateApplication
- original Runtime
- original jarfile/zipfile
- snapshot
Application: RUNNING
The old application is running. Further work is required to update
the job is running
Is the application behaving as expected?
The application is misbehaving
Update successful. The application is running on the new runtime

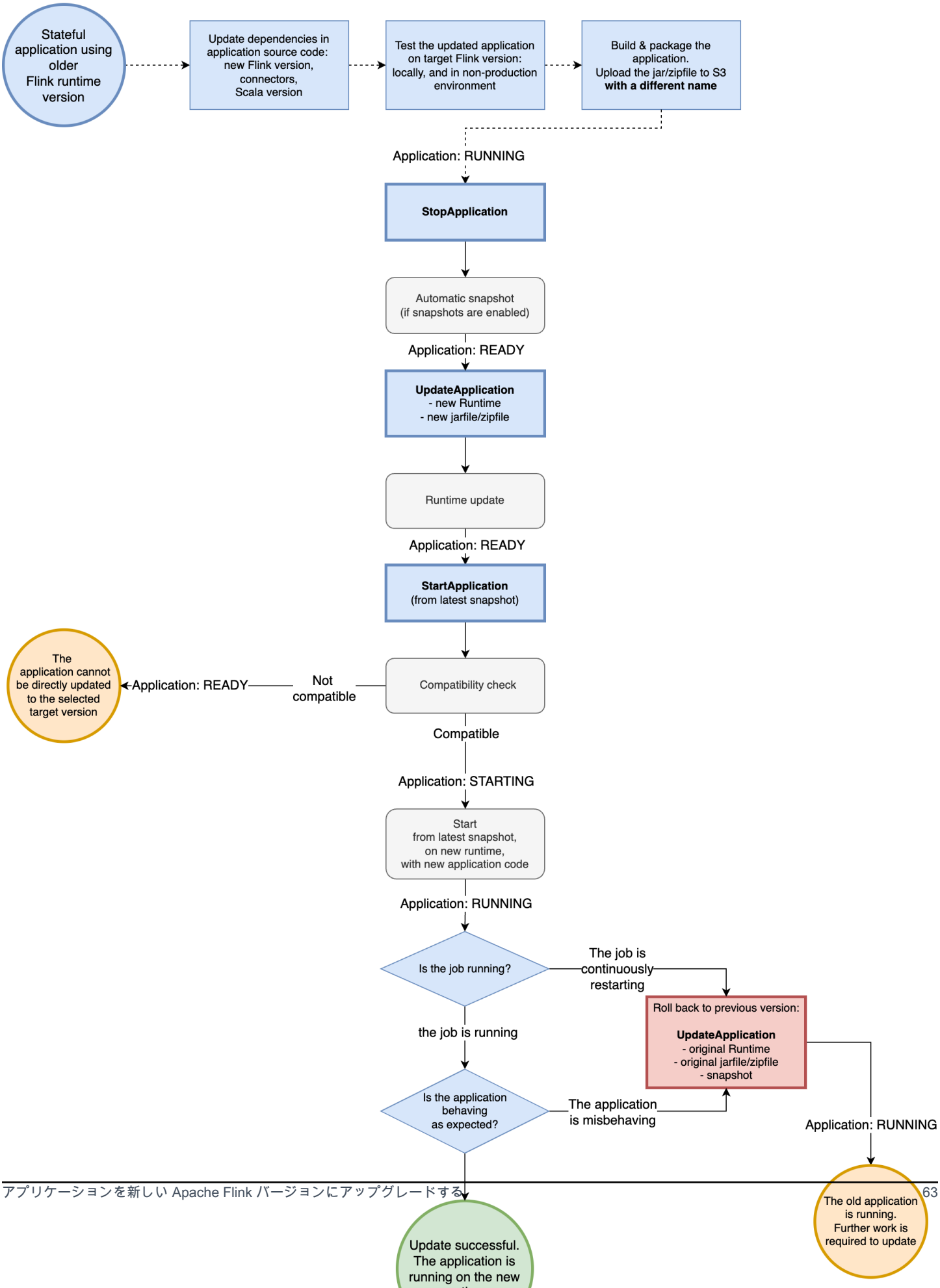
READY 状態のアプリケーションのアップグレード

次の例は、 を使用して、米国東部 (バージニア北部) UpgradeTest の という名前の READY 状態のアプリケーションを Flink 1.18 にアップグレードする例を示しています。 AWS CLIアプリケーションが実行されていないため、アプリケーションを起動するためのスナップショットが指定されていません。スナップショットは、アプリケーション開始リクエストを発行するときに指定できます。

```
aws --region us-east-1 kinesisanalyticsv2 update-application \
--application-name UpgradeTest --runtime-environment-update "FLINK-1_18" \
--application-configuration-update '{"ApplicationCodeConfigurationUpdate": '\
'{"CodeContentUpdate": {"S3ContentLocationUpdate": '\
'{"FileKeyUpdate": "flink_1_18_app.jar"}}}}' \
 --current-application-version-id ${current_application_version}
```

- READY 状態のアプリケーションのランタイムを任意の Flink バージョンに更新できます。Amazon Managed Service for Apache Flink は、アプリケーションを起動するまでチェックを実行しません。
- Amazon Managed Service for Apache Flink は、アプリケーションを起動するために選択したスナップショットに対してのみ互換性チェックを実行します。これらは、「Flink 互換性表」に従った基本的な互換性チェックです。スナップショットが作成された Flink バージョンとターゲットとする Flink バージョンのみをチェックします。選択したスナップショットの Flink ランタイムがアプリの新しいランタイムと互換性がない場合、開始リクエストが拒否される可能性があります。

準備完了状態のアプリケーションのプロセスフロー


Stateful application using older Flink runtime version
Update dependencies in application source code: new Flink version, connectors, Scala version
Test the updated application on target Flink version: locally, and in non-production environment
Build & package the application. Upload the jar/zipfile to S3 with a different name
Application: RUNNING
StopApplication
Automatic snapshot (if snapshots are enabled)
Application: READY
UpdateApplication - new Runtime - new jarfile/zipfile
Runtime update
Application: READY
StartApplication (from latest snapshot)
Compatibility check
Not compatible
Application: READY
The application cannot be directly updated to the selected target version
Compatible
Application: STARTING
Start from latest snapshot, on new runtime, with new application code
Application: RUNNING
Is the job running?
The job is continuously restarting
Roll back to previous version: UpdateApplication - original Runtime - original jarfile/zipfile - snapshot
the job is running
Is the application behaving as expected?
The application is misbehaving
Application: RUNNING
Update successful. The application is running on the new
The old application is running. Further work is required to update

ロールバック

アプリケーションに問題がある場合、または Flink バージョン間でアプリケーションコードに不整合がある場合は、 AWS CLI、、 AWS SDK AWS CloudFormation、または を使用してロールバックできます AWS Management Console。次の例は、さまざまな障害シナリオでのロールバックがどのように見えるかを示しています。

ランタイムのアップグレードは成功し、アプリケーションは **RUNNING**状態ですが、ジョブは失敗し、継続的に再起動しています

米国東部 (バージニア北部) で、 という名前のステートフルアプリケーションを Flink 1.15 `TestApplication`から Flink 1.18 にアップグレードしようとしているとします。ただし、アップグレードされた Flink 1.18 アプリケーションは、アプリケーションが RUNNING状態であっても、起動に失敗しているか、常に再起動しています。これは一般的な障害シナリオです。さらなるダウンタイムを避けるため、アプリケーションをすぐに以前の実行中のバージョン (Flink 1.15) にロールバックし、後で問題を診断することをお勧めします。

アプリケーションを以前の実行バージョンにロールバックするには、rollback-application AWS CLI コマンドまたは RollbackApplication API アクションを使用します。この API アクションは、最新バージョンになった変更をロールバックします。次に、最後に成功したスナップショットを使用してアプリケーションを再起動します。

アップグレードを試みる前に、既存のアプリでスナップショットを作成することを強くお勧めします。これにより、データ損失やデータの再処理を回避できます。

この障害シナリオでは、 はアプリケーションをロールバック AWS CloudFormation しません。テンプレートを更新 CloudFormation して、以前のランタイムを指し、 がアプリケーションを更新 CloudFormation するよう強制するには、前のコードを指す必要があります。それ以外の場合、CloudFormation はアプリケーションが RUNNING状態に移行したときに更新されたことを前提としています。

スタックしているアプリケーションをロールバックする **UPDATING**

アップグレードの試行後にアプリケーションが UPDATINGまたは AUTOSCALING状態でスタックした場合、Amazon Managed Service for Apache Flink は rollback-applications AWS CLI コマンド、またはアプリケーションをスタックUPDATINGまたは AUTOSCALING状態より前のバージョンにロールバックできる RollbackApplications API アクションを提供します。この API は、アプリケーションがUPDATINGまたはAUTOSCALING推移的な状態でスタックする原因となった変更をロールバックします。

一般的なベストプラクティスと推奨事項

- 本番稼働用アップグレードを試みる前に、本番稼働用以外の環境で 状態なしで新しいジョブ/ランタイムをテストします。
- まず、非本番稼働用アプリケーションでステートフルアップグレードをテストすることを検討してください。
- 新しいジョブグラフに、アップグレードされたアプリケーションの起動に使用するスナップショットと互換性がある状態があることを確認してください。
  - 演算子の状態に保存されている型が同じままであることを確認してください。タイプが変更された場合、Apache Flink は演算子の状態を復元できません。
  - `uid` メソッドを使用して設定したオペレーター IDsが同じままであることを確認してください。Apache Flink は、一意の IDsことを強くお勧めします。詳細については、Apache Flink ドキュメントの「オペレーター IDs」を参照してください。

    オペレーターに IDs を割り当てない場合、Flink は自動的に ID を生成します。その場合、プログラム構造に依存し、変更すると互換性の問題が発生する可能性があります。Flink は、オペレータ IDsを使用してスナップショットの状態をオペレータに一致させます。オペレータ IDs を変更すると、アプリケーションが起動しないか、スナップショットに保存されている状態が削除され、新しいオペレータが状態なしで開始されます。
  - キー付き状態の保存に使用されるキーを変更しないでください。
  - ウィンドウや結合などのステートフル演算子の入力タイプを変更しないでください。これにより、オペレーターの内部状態のタイプが暗黙的に変更され、状態が非互換性になります。

注意事項と既知の問題

Flink 1.19 以降からの設定変更は許可されていません

- Flink 1.18 以前から Flink 1.19 以降でランタイムを更新する場合、Flink ジョブコードを使用した Flink ジョブ設定の変更は許可されなくなります。その結果、アプリケーションはジョブの送信に失敗します。エラーログは、実行時にどの許可されていない設定が変更されたかを示します。詳細については、「FlinkRuntimeException：「許可されていない設定変更 (複数可) が検出されました」」を参照してください。

状態の互換性に関する既知の制限事項

- Table API を使用している場合、Apache Flink は Flink バージョン間の状態の互換性を保証しません。詳細については、Apache Flink ドキュメントの「ステートフルアップグレードと進化」を参照してください。
- Flink 1.6 の状態は Flink 1.18 と互換性がありません。状態が 1.6 から 1.18 以降にアップグレードしようとすると、API はリクエストを拒否します。1.8、1.11、1.13、1.15 にアップグレードしてスナップショットを作成し、1.18 以降にアップグレードできます。詳細については、Apache Flink ドキュメントの「アプリケーションと Flink バージョンのアップグレード」を参照してください。

Flink Kinesis Connector の既知の問題

- Flink 1.11 以前を使用していて、E nhanced-fan-out (EFO) サポート用の`amazon-kinesis-connector-flink`コネクタを使用している場合は、Flink 1.13 以降にステートフルアップグレードするための追加の手順を実行する必要があります。これは、コネクタのパッケージ名が変更されたためです。詳細については、「」を参照してくださいamazon-kinesis-connector-flink。

  Flink 1.11 以前の`amazon-kinesis-connector-flink`コネクタはパッケージ を使用し`software.amazon.kinesis`、Flink 1.13 以降の Kinesis コネクタは を使用します`org.apache.flink.streaming.connectors.kinesis`。移行をサポートするには、このツールを使用します: amazon-kinesis-connector-flink-state-migrator 。
- で Flink 1.13 以前を使用して`FlinkKinesisProducer`いて、Flink 1.15 以降にアップグレードする場合、ステートフルアップグレードでは、新しい ではなく Flink 1.15 以降`FlinkKinesisProducer`で引き続き を使用する必要がありまする`KinesisStreamsSink`。ただし、シンクにカスタム`uid`設定が既にある場合は、 が 状態を保持しない`KinesisStreamsSink`ため、 `FlinkKinesisProducer` に切り替えることができます。Flink は、カスタム`uid`が設定されているため、同じ演算子として扱います。

Scala で記述された Flink アプリケーション

- Flink 1.15 以降、Apache Flink はランタイムに Scala を含めません。Flink 1.15 以降にアップグレードするときは、使用する Scala のバージョンと、その他の Scala の依存関係をコード JAR/zip に含める必要があります。詳細については、「Amazon Managed Service for Apache Flink for Apache Flink 1.15.2 release 」を参照してください。
- アプリケーションで Scala を使用していて、Flink 1.11 以前 (Scala 2.11) から Flink 1.13 (Scala 2.12) にアップグレードする場合は、コードで Scala 2.12 を使用していることを確認してくださ

い。そうしないと、Flink 1.13 アプリケーションが Flink 1.13 ランタイムで Scala 2.11 クラスを見つけられない可能性があります。

Flink アプリケーションをダウングレードする際の考慮事項

- Flink アプリケーションのダウングレードは可能ですが、以前の Flink バージョンでアプリケーションが実行されていた場合に限られます。ステートフルアップグレードの場合、Managed Service for Apache Flink では、ダウングレードに一致する 以前のバージョンで作成されたスナップショットを使用する必要があります。
- ランタイムを Flink 1.13 以降から Flink 1.11 以前に更新していて、アプリケーションが HashMap 状態バックエンドを使用している場合、アプリケーションは継続的に失敗します。

Managed Service for Apache Flink でのアプリケーションスケーリング

スケーリングを実装するために、Apache Flink 用 Amazon Managed Service のタスクの並列実行とリソースの割り当てを設定できます。Apache Flink がタスクの並列インスタンスをスケジュールする方法については、Apache Flink ドキュメントの「並列実行」を参照してください。

トピック

- アプリケーション並列処理と ParallelismPerKPU の設定
- Kinesis 処理ユニットの割り当て
- アプリケーションの並列処理の更新
- Auto Scaling

アプリケーション並列処理と ParallelismPerKPU の設定

Apache Flink アプリケーション用 Managed Serviceタスク (ソースからの読み取りやオペレータの実行など) のparallel 実行は、次の「`ParallelismConfiguration`」プロパティを使用して設定します。

- `Parallelism` — このプロパティを使用して、デフォルトの Apache Flink アプリケーション並列処理を設定します。すべてのオペレータ、ソース、シンクは、アプリケーションコードでオーバーライドされない限り、この並列処理で実行されます。デフォルトは 1 で、最大値は 256 です。

- `ParallelismPerKPU` — このプロパティを使用して、使用しているアプリケーションの Kinesis Processing Unit (KPU) あたりにスケジュールできるparallel タスクの数を設定します。デフォルトは 1 で、最大は 8 です。ブロッキングオペレーション (I/O など) を行うアプリケーションでは、`ParallelismPerKPU` の値が大きいほど KPU リソースを最大限に活用できます。

> **Note**
>
> `Parallelism`の上限は、KPUの上限 (デフォルトは64) の `ParallelismPerKPU` 倍です (デフォルトは64)。KPU の上限は、制限の引き上げをリクエストすることで増やすことができます。制限の引き上げをリクエストする方法については、「Service Quotas」 の 「制限の引き上げをリクエストするには」 を参照してください。

特定の演算子のタスク並列処理の設定については、Apache Flink ドキュメントの「並列処理の設定: 演算子」を参照してください。

Kinesis 処理ユニットの割り当て

Apache Flink 用 Managed Service 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。割り当てられた KPU ごとに、50 GB の実行中のアプリケーションストレージも提供されます。

Apache Flink 用 Managed Serviceは、次のように `Parallelism` および `ParallelismPerKPU` プロパティを使用してアプリケーションの実行に必要な KPU を計算します。

```
Allocated KPUs for the application = Parallelism/ParallelismPerKPU
```

Apache Flink 用 Managed Service は、スループットや処理アクティビティの急増に応じて、アプリケーションリソースを迅速に提供します。アクティビティの急増が過ぎると、アプリケーションから徐々にリソースを削除します。リソースの自動割り当てを無効にするには、アプリケーションの並列処理の更新 で後述するように、`AutoScalingEnabled` 値を `false` に設定します。

アプリケーションの KPU のデフォルト制限は 64 です。制限の引き上げをリクエストする方法については、「サービス クォータ」 の 「制限の引き上げをリクエストするには」 を参照してください。

> **Note**
>
> オーケストレーションの目的で追加の KPU が課金されます。詳細については、「Managed Service for Apache Flink の料金」 を参照してください

アプリケーションの並列処理の更新

このセクションには、アプリケーションの並列処理を設定する API アクションのサンプルリクエストが含まれています。API アクションでリクエストブロックを使用する方法のその他の例と手順については、Apache Flink API のマネージドサービスのサンプルコード を参照してください。

以下の「CreateApplication」アクションのリクエスト例では、アプリケーションの作成時に並列処理を設定します。

```
{
   "ApplicationName": "string",
   "RuntimeEnvironment":"FLINK-1_18",
   "ServiceExecutionRole":"arn:aws:iam::123456789123:role/myrole",
   "ApplicationConfiguration": {
      "ApplicationCodeConfiguration":{
      "CodeContent":{
         "S3ContentLocation":{
            "BucketARN":"arn:aws:s3:::mybucket",
            "FileKey":"myflink.jar",
            "ObjectVersion":"AbCdEfGhIjKlMnOpQrStUvWxYz12345"
            }
         },
      "CodeContentType":"ZIPFILE"
   },
      "FlinkApplicationConfiguration": {
         "ParallelismConfiguration": {
            "AutoScalingEnabled": "true",
            "ConfigurationType": "CUSTOM",
            "Parallelism": 4,
            "ParallelismPerKPU": 4
         }
      }
   }
}
```

次の UpdateApplication アクションのリクエスト例では、既存のアプリケーションの並列処理を設定します。

```
{
   "ApplicationName": "MyApplication",
   "CurrentApplicationVersionId": 4,
   "ApplicationConfigurationUpdate": {
      "FlinkApplicationConfigurationUpdate": {
         "ParallelismConfigurationUpdate": {
            "AutoScalingEnabledUpdate": "true",
            "ConfigurationTypeUpdate": "CUSTOM",
            "ParallelismPerKPUUpdate": 4,
            "ParallelismUpdate": 4
         }
      }
   }
}
```

次の UpdateApplication アクションのリクエストの例では、既存のアプリケーションの並列処理が無効になります。

```
{
   "ApplicationName": "MyApplication",
   "CurrentApplicationVersionId": 4,
   "ApplicationConfigurationUpdate": {
      "FlinkApplicationConfigurationUpdate": {
         "ParallelismConfigurationUpdate": {
            "AutoScalingEnabledUpdate": "false"
         }
      }
   }
}
```

Auto Scaling

Apache Flink 用 Managed Service は、ほとんどのシナリオでソースのデータスループットとオペレーターの複雑さに対応できるように、アプリケーションの並列度を柔軟にスケーリングします。Apache Flink 用 Managed Service は、アプリケーションのリソース (CPU) 使用状況を監視し、それに応じてアプリケーションの並列度を柔軟にスケールアップまたはスケールダウンします。

- CloudWatch メトリクスの最大値が 15 分間 75% 以上になると、アプリケーションcontainerCPUUtilizationはスケールアップ (並列処理の増加) します。つまり、1 分間で 75 パーセント以上の連続したデータポイントが 15 個あると、ScaleUp アクションがトリガーされます。
- CPU 使用率が 6 時間にわたって 10% を下回ると、アプリケーションはスケールダウン (並列処理が減少) します。つまり、1 分間の期間が 10% 未満の連続するデータポイントが 360 個ある場合に、ScaleDown アクションがトリガーされます。

> **Note**
>
> 1分間の containerCPUUtilization の最大値は、スケーリング・アクションに使用するデータポイントとの相関を見つけるために参照できるが、アクションがトリガーされた正確な瞬間を反映する必要はありません。

Apache Flink 用 Managed Service では、アプリケーションの CurrentParallelism 値がアプリケーションの Parallelism 設定を下回ることはありません。

Apache Flink 用 Managed Serviceサービスがアプリケーションをスケーリングしているときは、AUTOSCALING 状態になります。 DescribeApplication または ListApplicationsアクションをAPI使用して、現在のアプリケーションのステータスを確認できます。サービスがアプリケーションをスケーリングしている間に使用できる唯一の有効な API アクションは、 Forceパラメータ StopApplicationを に設定することですtrue。

AutoScalingEnabled プロパティ (「FlinkApplicationConfiguration」の一部) を使用して、auto スケーリングの動作を有効または無効にすることができます。Managed Service for Apache Flink がプロビジョニングする KPUs については、アプリケーションparallelismとparallelismPerKPU設定の機能であるアカウントで AWS 課金されます。アクティビティが急増すると、Apache Flink 用 Managed Service のコストが増加します。

料金については、「Amazon Managed Service for Apache Flink の料金」を参照してください。

アプリケーションのスケーリングについて、以下のことに注意してください：

- 自動スケーリングはデフォルトで有効になっている。
- Studio ノートブックにはスケーリングは適用されません。ただし、Studio Notebook を永続的状態のアプリケーションとしてデプロイすると、スケーリングはデプロイされたアプリケーションに適用されます。

- 使用しているアプリケーションのデフォルトの上限は 64 KPU です。詳細については、「クォータ」を参照してください。
- 自動スケーリングによってアプリケーションの並列度が更新されると、アプリケーションのダウンタイムが発生します。このダウンタイムを回避するには、以下を実行します。
  - 自動スケーリングを無効にする
  - アプリケーションの `parallelism`および を UpdateApplicationアクション`parallelismPerKPU`で設定します。使用しているアプリケーションの並列処理の設定の詳細については、以下の the section called “アプリケーションの並列処理の更新” を参照してください。
  - アプリケーションのリソース使用状況を定期的に監視して、アプリケーションがワークロードに適した並列度設定になっていることを確認してください。割り当てリソースの使用状況を監視する方法については、the section called “Managed Service for Apache Flink のメトリクスとディメンション” を参照してください。

最大並列度に関する考慮事項

- Autoscale ロジックは、Flink ジョブを並列処理にスケーリングしてジョブやオペレータ `maxParallelism` に干渉することを防ぎます。たとえば、ソースとシンクのみの単純なジョブで、ソースが `maxParallelism` 16 で、`sink` が 8 の場合、ジョブは 8 以上にオートスケーリングされません。
- `maxParallelism` がジョブに設定されていない場合、Flink はデフォルトで 128 に設定されます。したがって、ジョブを 128 よりも高い並列処理で実行する必要があると思われる場合は、アプリケーション用にその数値を設定する必要があります。
- ジョブの自動スケーリングを期待していても表示されない場合は、`maxParallelism` 値がそれを考慮に入れていることを確認してください。

追加情報については、「Apache Flink の拡張モニタリングと自動スケーリング」を参照してください。

例については、 kda-flink-app-autoscaling「」を参照してください。

タグ付けを使用する

このセクションでは、アプリケーションにキーバリューメタデータタグをManaged Service for Apache Flink アプリケーションに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Apache Flink アプリケーション用 Managed Serviceの課金を決定。詳細については、Billing と Cost Management ユーザーガイドの 「コスト配分タグを使用する」 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、AWS Identity and Access Management 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タグを追加するか、タグを u 個追加します。
- アプリケーションのタグを一覧表示する。
- アプリケーションからタグを削除する。

アプリケーション作成時のタグの追加

タグは、tagsCreateApplicationアプリケーションの作成時にアクションのパラメータを使用して追加します。

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タグを追加するか、タグを u 個追加します。

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を一覧表示する。

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:kinesisanalyticsus-west-2:012345678901:application/
MyApplication"
}
```

アプリケーションからタグを削除する。

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:kinesisanalyticsus-west-2:012345678901:application/
MyApplication",
   "TagKeys": [ "KeyOfFirstTagToRemove", "KeyOfSecondTagToRemove" ]
}
```

Apache Flink CloudFormation 用マネージドサービスとの併用

次の演習では、同じスタック内の Lambda AWS CloudFormation 関数を使用して作成した Flink アプリケーションを起動する方法を示しています。

開始する前に

この演習を開始する前に、at を使用して Flink アプリケーションを作成する手順に従ってください。AWS CloudFormation AWS::KinesisAnalytics::Application

Lambda 関数の記述

作成または更新後に Flink アプリケーションを起動するには、kinesisanalyticsv2 「アプリケーションを起動」 API を使用します。この呼び出しは Flink AWS CloudFormation アプリケーション作成後のイベントによってトリガーされます。Lambda 関数をトリガーするためのスタックの設定方法については、このエクササイズの後半で説明しますが、まずは Lambda 関数の宣言とそのコードに焦点を当てます。この例では「Python3.8」ランタイムを使用しています。

```
StartApplicationLambda:
    Type: AWS::Lambda::Function
    DependsOn: StartApplicationLambdaRole
    Properties:
      Description: Starts an application when invoked.
      Runtime: python3.8
      Role: !GetAtt StartApplicationLambdaRole.Arn
      Handler: index.lambda_handler
      Timeout: 30
      Code:
```

```
        ZipFile: |
          import logging
          import cfnresponse
          import boto3

          logger = logging.getLogger()
          logger.setLevel(logging.INFO)

          def lambda_handler(event, context):
            logger.info('Incoming CFN event {}'.format(event))

            try:
              application_name = event['ResourceProperties']['ApplicationName']

              # filter out events other than Create or Update,
              # you can also omit Update in order to start an application on Create
 only.
              if event['RequestType'] not in ["Create", "Update"]:
                logger.info('No-op for Application {} because CFN RequestType {} is
 filtered'.format(application_name, event['RequestType']))
                cfnresponse.send(event, context, cfnresponse.SUCCESS, {})

                return

              # use kinesisanalyticsv2 API to start an application.
              client_kda = boto3.client('kinesisanalyticsv2',
 region_name=event['ResourceProperties']['Region'])

              # get application status.
              describe_response =
 client_kda.describe_application(ApplicationName=application_name)
              application_status = describe_response['ApplicationDetail']
['ApplicationStatus']

              # an application can be started from 'READY' status only.
              if application_status != 'READY':
                logger.info('No-op for Application {} because ApplicationStatus {} is
 filtered'.format(application_name, application_status))
                cfnresponse.send(event, context, cfnresponse.SUCCESS, {})

                return

              # create RunConfiguration.
              run_configuration = {
```

```
                'ApplicationRestoreConfiguration': {
                  'ApplicationRestoreType': 'RESTORE_FROM_LATEST_SNAPSHOT',
                }
              }

              logger.info('RunConfiguration for Application {}:
 {}'.format(application_name, run_configuration))

              # this call doesn't wait for an application to transfer to 'RUNNING'
 state.
              client_kda.start_application(ApplicationName=application_name,
 RunConfiguration=run_configuration)

              logger.info('Started Application: {}'.format(application_name))
              cfnresponse.send(event, context, cfnresponse.SUCCESS, {})
            except Exception as err:
              logger.error(err)
              cfnresponse.send(event,context, cfnresponse.FAILED, {"Data": str(err)})
```

前述のコードでは、Lambda AWS CloudFormation は受信イベントを処理し、Createおよび以外のすべてをフィルターで除外しUpdate、アプリケーションの状態を取得して、状態がである場合は起動します。READYアプリケーションの状態を取得するには、以下に示すように Lambda ロールを作成する必要があります。

Lambda ロールの作成

Lambda がアプリケーションと正常に「対話」してログを書き込むためのロールを作成します。このロールはデフォルトの管理ポリシーを使用しますが、カスタムポリシーを使用するように絞り込むこともできます。

```
StartApplicationLambdaRole:
    Type: AWS::IAM::Role
    DependsOn: TestFlinkApplication
    Properties:
      Description: A role for lambda to use while interacting with an application.
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: Allow
            Principal:
              Service:
                - lambda.amazonaws.com
```

```
            Action:
              - sts:AssumeRole
      ManagedPolicyArns:
        - arn:aws:iam::aws:policy/Amazonmanaged-flinkFullAccess
        - arn:aws:iam::aws:policy/CloudWatchLogsFullAccess
      Path: /
```

Lambda リソースは Flink アプリケーションに依存しているため、同じスタックで Flink アプリケーションを作成した後に作成されることに注意してください。

Lambda 関数を呼び出す

あとは、Lambda 関数を呼び出すだけです。そのためには、カスタムリソースを使用します。

```
StartApplicationLambdaInvoke:
    Description: Invokes StartApplicationLambda to start an application.
    Type: AWS::CloudFormation::CustomResource
    DependsOn: StartApplicationLambda
    Version: "1.0"
    Properties:
      ServiceToken: !GetAtt StartApplicationLambda.Arn
      Region: !Ref AWS::Region
      ApplicationName: !Ref TestFlinkApplication
```

Lambdaを使って Flink アプリケーションを起動するのに必要なものはこれだけです。これで、独自のスタックを作成するか、以下の完全な例を使用して、これらすべてのステップが実際にどのように機能するかを確認する準備ができました。

完全な例

以下の例は、前のステップを少し拡張したもので、RunConfigurationテンプレートパラメーターによる調整が加えられています。これは、試してみるための作業スタックです。添付の注意事項を必ずお読みください。

stack.yaml

```
Description: 'kinesisanalyticsv2 CloudFormation Test Application'
Parameters:
  ApplicationRestoreType:
    Description: ApplicationRestoreConfiguration option, can
 be SKIP_RESTORE_FROM_SNAPSHOT, RESTORE_FROM_LATEST_SNAPSHOT or
 RESTORE_FROM_CUSTOM_SNAPSHOT.
```

```
    Type: String
    Default: SKIP_RESTORE_FROM_SNAPSHOT
    AllowedValues: [ SKIP_RESTORE_FROM_SNAPSHOT, RESTORE_FROM_LATEST_SNAPSHOT,
 RESTORE_FROM_CUSTOM_SNAPSHOT ]
  SnapshotName:
    Description: ApplicationRestoreConfiguration option, name of a snapshot to restore
 to, used with RESTORE_FROM_CUSTOM_SNAPSHOT ApplicationRestoreType.
    Type: String
    Default: ''
  AllowNonRestoredState:
    Description: FlinkRunConfiguration option, can be true or false.
    Default: true
    Type: String
    AllowedValues: [ true, false ]
  CodeContentBucketArn:
    Description: ARN of a bucket with application code.
    Type: String
  CodeContentFileKey:
    Description: A jar filename with an application code inside a bucket.
    Type: String
Conditions:
  IsSnapshotNameEmpty: !Equals [ !Ref SnapshotName, '' ]
Resources:
  TestServiceExecutionRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: Allow
            Principal:
              Service:
                - kinesisanlaytics.amazonaws.com
            Action: sts:AssumeRole
      ManagedPolicyArns:
        - arn:aws:iam::aws:policy/AmazonKinesisFullAccess
        - arn:aws:iam::aws:policy/AmazonS3FullAccess
      Path: /
  InputKinesisStream:
    Type: AWS::Kinesis::Stream
    Properties:
      ShardCount: 1
  OutputKinesisStream:
    Type: AWS::Kinesis::Stream
```

```
    Properties:
      ShardCount: 1
  TestFlinkApplication:
    Type: 'AWS::kinesisanalyticsv2::Application'
    Properties:
      ApplicationName: 'CFNTestFlinkApplication'
      ApplicationDescription: 'Test Flink Application'
      RuntimeEnvironment: 'FLINK-1_18'
      ServiceExecutionRole: !GetAtt TestServiceExecutionRole.Arn
      ApplicationConfiguration:
        EnvironmentProperties:
          PropertyGroups:
            - PropertyGroupId: 'KinesisStreams'
              PropertyMap:
                INPUT_STREAM_NAME: !Ref InputKinesisStream
                OUTPUT_STREAM_NAME: !Ref OutputKinesisStream
                AWS_REGION: !Ref AWS::Region
        FlinkApplicationConfiguration:
          CheckpointConfiguration:
            ConfigurationType: 'CUSTOM'
            CheckpointingEnabled: True
            CheckpointInterval: 1500
            MinPauseBetweenCheckpoints: 500
          MonitoringConfiguration:
            ConfigurationType: 'CUSTOM'
            MetricsLevel: 'APPLICATION'
            LogLevel: 'INFO'
          ParallelismConfiguration:
            ConfigurationType: 'CUSTOM'
            Parallelism: 1
            ParallelismPerKPU: 1
            AutoScalingEnabled: True
        ApplicationSnapshotConfiguration:
          SnapshotsEnabled: True
        ApplicationCodeConfiguration:
          CodeContent:
            S3ContentLocation:
              BucketARN: !Ref CodeContentBucketArn
              FileKey: !Ref CodeContentFileKey
          CodeContentType: 'ZIPFILE'
  StartApplicationLambdaRole:
    Type: AWS::IAM::Role
    DependsOn: TestFlinkApplication
    Properties:
```

```
      Description: A role for lambda to use while interacting with an application.
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: Allow
            Principal:
              Service:
                - lambda.amazonaws.com
            Action:
              - sts:AssumeRole
      ManagedPolicyArns:
        - arn:aws:iam::aws:policy/Amazonmanaged-flinkFullAccess
        - arn:aws:iam::aws:policy/CloudWatchLogsFullAccess
      Path: /
  StartApplicationLambda:
    Type: AWS::Lambda::Function
    DependsOn: StartApplicationLambdaRole
    Properties:
      Description: Starts an application when invoked.
      Runtime: python3.8
      Role: !GetAtt StartApplicationLambdaRole.Arn
      Handler: index.lambda_handler
      Timeout: 30
      Code:
        ZipFile: |
          import logging
          import cfnresponse
          import boto3

          logger = logging.getLogger()
          logger.setLevel(logging.INFO)

          def lambda_handler(event, context):
            logger.info('Incoming CFN event {}'.format(event))

            try:
              application_name = event['ResourceProperties']['ApplicationName']

              # filter out events other than Create or Update,
              # you can also omit Update in order to start an application on Create
only.
              if event['RequestType'] not in ["Create", "Update"]:
                logger.info('No-op for Application {} because CFN RequestType {} is
filtered'.format(application_name, event['RequestType']))
```

```
                cfnresponse.send(event, context, cfnresponse.SUCCESS, {})

                return

            # use kinesisanalyticsv2 API to start an application.
            client_kda = boto3.client('kinesisanalyticsv2',
 region_name=event['ResourceProperties']['Region'])

            # get application status.
            describe_response =
 client_kda.describe_application(ApplicationName=application_name)
            application_status = describe_response['ApplicationDetail']
['ApplicationStatus']

            # an application can be started from 'READY' status only.
            if application_status != 'READY':
              logger.info('No-op for Application {} because ApplicationStatus {} is
 filtered'.format(application_name, application_status))
              cfnresponse.send(event, context, cfnresponse.SUCCESS, {})

              return

            # create RunConfiguration from passed parameters.
            run_configuration = {
              'FlinkRunConfiguration': {
                'AllowNonRestoredState': event['ResourceProperties']
['AllowNonRestoredState'] == 'true'
              },
              'ApplicationRestoreConfiguration': {
                'ApplicationRestoreType': event['ResourceProperties']
['ApplicationRestoreType'],
              }
            }

            # add SnapshotName to RunConfiguration if specified.
            if event['ResourceProperties']['SnapshotName'] != '':
              run_configuration['ApplicationRestoreConfiguration']['SnapshotName'] =
 event['ResourceProperties']['SnapshotName']

            logger.info('RunConfiguration for Application {}:
 {}'.format(application_name, run_configuration))

            # this call doesn't wait for an application to transfer to 'RUNNING'
 state.
```

```
              client_kda.start_application(ApplicationName=application_name,
 RunConfiguration=run_configuration)

              logger.info('Started Application: {}'.format(application_name))
              cfnresponse.send(event, context, cfnresponse.SUCCESS, {})
            except Exception as err:
              logger.error(err)
              cfnresponse.send(event,context, cfnresponse.FAILED, {"Data": str(err)})
  StartApplicationLambdaInvoke:
    Description: Invokes StartApplicationLambda to start an application.
    Type: AWS::CloudFormation::CustomResource
    DependsOn: StartApplicationLambda
    Version: "1.0"
    Properties:
      ServiceToken: !GetAtt StartApplicationLambda.Arn
      Region: !Ref AWS::Region
      ApplicationName: !Ref TestFlinkApplication
      ApplicationRestoreType: !Ref ApplicationRestoreType
      SnapshotName: !Ref SnapshotName
      AllowNonRestoredState: !Ref AllowNonRestoredState
```

繰り返しになりますが、アプリケーション自体だけでなく、Lambda のロールも調整したい場合があります。

上記のスタックを作成する前に、パラメータを指定することを忘れないでください。

parameters.json

```
[
  {
    "ParameterKey": "CodeContentBucketArn",
    "ParameterValue": "YOUR_BUCKET_ARN"
  },
  {
    "ParameterKey": "CodeContentFileKey",
    "ParameterValue": "YOUR_JAR"
  },
  {
    "ParameterKey": "ApplicationRestoreType",
    "ParameterValue": "SKIP_RESTORE_FROM_SNAPSHOT"
  },
  {
    "ParameterKey": "AllowNonRestoredState",
    "ParameterValue": "true"
```

```
  }
]
```

「YOUR_BUCKET_ARN」と「YOUR_JAR」を特定の要件に置き換えてください。この「ガイド」に従って Amazon S3 バケットとアプリケーション jar を作成できます。

次に、スタックを作成します (YOUR_REGION を us-east-1 などの任意のリージョンに置き換える)。

```
aws cloudformation create-stack --region YOUR_REGION --template-body "file://
stack.yaml" --parameters "file://parameters.json" --stack-name "TestManaged Service for
 Apache FlinkStack" --capabilities CAPABILITY_NAMED_IAM
```

これで「https://console.aws.amazon.com/cloudformation」に移行して進行状況を確認できるようになりました。作成されると、Flinkアプリケーションが「Starting」状態になるはずです。「Running」の起動には数分かかる場合があります。

詳細については、次を参照してください。

- AWS を使用して任意のサービスプロパティを取得する 4 つの方法 AWS CloudFormation (パート 1/3)。
- チュートリアル: Amazon マシンイメージ ID を参照する。

Apache Flink 用 Managed Service を搭載したApache Flink ダッシュボードの使用

アプリケーションの Apache Flink Dashboard を使用して Apache Flink 用 Managed Service アプリケーションの健全性を監視することができます。アプリケーションのダッシュボードには、以下の情報が表示されます。

- Task Managers や Task Slots など、使用中のリソース。
- 実行中、完了、キャンセル、失敗したジョブなど、ジョブに関する情報。

Apache Flink Task Managers、Task Slots、Jobsに関する詳細は、Apache Flink ウェブサイトの「Apache Flink Architecture」を参照してください。

Apache Flink アプリケーション用 Managed Service を搭載した Apache Flink Dashboard を使用する際には、次の点に注意してください。

- Apache Flink アプリケーションの Managed Service 用 Apache Flink Dashboard は読み取り専用です。Apache Flink Dashboard で Apache Flink アプリケーション用 Managed Service を変更することはできません。
- Apache Flink Dashboard は Microsoft Internet Explorer と互換性がありません。

アプリケーションの Apache Flink ダッシュボードへのアクセス

アプリケーションの Apache Flink Dashboard には、 Apache Flink コンソール用のManaged Service を使用するか、CLI を使用して安全な URL エンドポイントをリクエストすることでアクセスすることができます。

Apache Flink 管理サービスコンソールを使用してアプリケーションの Apache Flink ダッシュボードにアクセスする

コンソールからアプリケーションの Apache Flink Dashboard にアクセスするには、アプリケーションのページで「Apache Flink Dashboard」を選択します。

> **Note**
>
> Apache Flink コンソール用 Managed Service からダッシュボードを開くと、コンソールが生成する URL は 12 時間有効になります。

Apache Flink CLI 用 Managed Service を使用してアプリケーションの Apache Flink Dashboard にアクセスする

Apache Flink CLI 用 Managed Service を使用して、アプリケーションダッシュボードにアクセスするための URL を生成することができます。生成した URL は、指定された期間だけ有効になります。

> **Note**
>
> 生成された URL に 3 分以内にアクセスしなければ、その URL は無効となります。

アクションを使用してダッシュボード URL を生成します。 CreateApplicationPresignedUrlアクションには以下のパラメータを指定できます。

- アプリケーション名
- URL が有効になる時間 (秒単位)
- URLタイプとして FLINK_DASHBOARD_URL を指定します

リリースバージョン

このトピックには、Managed Service for Apache Flink の各リリースでサポートされている機能と推奨コンポーネントバージョンに関する情報が含まれています。

> **Note**
>
> 非推奨の Apache Flink のバージョンを使用している場合は、Managed Service for Apache Flink Apache Flink のインプレースバージョンアップグレードの機能を使用して、アプリケーションをサポートされている最新の Flink バージョンにアップグレードすることをお勧めします。

Apache Flink バージョン	ステータス - Amazon Managed Service for Apache Flink	ステータス - Apache Flink コミュニティ	リンク
1.19.1	サポート	サポート	Amazon Managed Service for Apache Flink 1.19
1.18.1	サポート	サポート	Amazon Managed Service for Apache Flink 1.18
1.15.2	サポート	サポートされていません	Amazon Managed Service for Apache Flink 1.15
1.13.1	サポート	サポートされていません	開始方法: Flink 1.13.2
1.11.1	非推奨	サポートされていません	Managed Service for Apache Flink の以前のバージョン情報 (2024 年 11 月 5 日に廃止)

Apache Flink バージョン	ステータス - Amazon Managed Service for Apache Flink	ステータス - Apache Flink コミュニティ	リンク
1.8.2	非推奨	サポートされていません	Managed Service for Apache Flink の以前のバージョン情報 (2024 年 11 月 5 日に廃止)
1.6.2	非推奨	サポートされていません	Managed Service for Apache Flink の以前のバージョン情報 (2024 年 11 月 5 日に廃止)

トピック

- Amazon Managed Service for Apache Flink 1.19
- Amazon Managed Service for Apache Flink 1.18
- Amazon Managed Service for Apache Flink 1.15
- Managed Service for Apache Flink の以前のバージョン情報

Amazon Managed Service for Apache Flink 1.19

Managed Service for Apache Flink が Apache Flink バージョン 1.19.1 をサポートするようになりました。このセクションでは、Apache Flink 1.19.1 の Managed Service for Apache Flink サポートで導入された主な新機能と変更点を紹介します。

> **Note**
>
> 以前にサポートされているバージョンの Apache Flink を使用していて、既存のアプリケーションを Apache Flink 1.19.1 にアップグレードする場合は、インプレース Apache Flink バージョンアップグレードを使用してアップグレードできます。詳細については、「Apache Flink のインプレースバージョンアップグレード」を参照してください。インプレースバージョンアップグレードでは、スナップショット、ログ、メトリクス、タグ、Flink 設定な

ど、Apache Flink バージョン全体で 1 つの ARN に対するアプリケーションのトレーサビリティを維持します。

サポートされている機能

Apache Flink 1.19.1 では、名前付きパラメータ、カスタムソース並列処理、さまざまな Flink 演算子のさまざまな状態 TTLs など、SQL API の改善が導入されています。

サポートされている機能および関連ドキュメント

サポートされている機能	説明	Apache Flink ドキュメントリファレンス
SQL API: SQL Hint を使用した異なるステート TTLs の設定をサポート	ユーザーは、ストリームの通常の結合とグループ集計で状態 TTL を設定できるようになりました。	FLIP-373: SQL Hint を使用した異なるステート TTLs の設定
SQL API: 関数と呼び出しプロシージャの名前付きパラメータのサポート	ユーザーは、パラメータの順序に依存するのではなく、名前付きパラメータを関数で使用できるようになりました。	FLIP-378: 関数と呼び出しプロシージャの名前付きパラメータのサポート
SQL API: SQL ソースの並列処理の設定	ユーザーは SQL ソースの並列処理を指定できるようになりました。	FLIP-367: テーブル/SQL ソースの並列処理の設定をサポート
SQL API: セッションウィンドウ TVF のサポート	ユーザーはセッションウィンドウのテーブル値関数を使用できるようになりました。	FLINK-24024: セッションウィンドウ TVF のサポート
SQL API: ウィンドウ TVF 集約が変更ログ入力をサポート	ユーザーは、変更ログ入力に対してウィンドウ集約を実行できるようになりました。	FLINK-20281: ウィンドウ集約が変更ログストリーム入力をサポート
Python 3.11 のサポート	Flink は Python 3.11 をサポートするようになりました。Python 3.10 と比較して 10 ~	FLINK-33030: Python 3.11 サポートの追加

サポートされている機能	説明	Apache Flink ドキュメントリファレンス
	60% 高速です。詳細については、「Python 3.11 の最新情報」を参照してください。	
TwoPhaseCommitting シンクのメトリクスを提供する	ユーザーは、コミッターのステータスに関する統計を 2 フェーズのコミットシンクで表示できます。	FLIP-371: でコミットを作成するための初期化コンテキストを提供する TwoPhaseCommittingSink
ジョブの再起動とチェックポイントのためのトレースレポーター	ユーザーはチェックポイントの期間と再現率の傾向に関するトレースをモニタリングできるようになりました。Amazon Managed Service for Apache Flink では、デフォルトで Slf4j トレースレポーターが有効になっているため、ユーザーはアプリケーション CloudWatch ログを通じてチェックポイントとジョブのトレースをモニタリングできます。	FLIP-384: チェックポイント TraceReporter トレースと復旧トレースの紹介と使用

> **Note**
>
> サポートケース を送信することで、次の機能をオプトインできます。

オプトイン機能と関連ドキュメント

オプトイン機能	説明	Apache Flink ドキュメントリファレンス
ソースがバックログを処理しているときに、チェックポイント間隔を長くするサポート	ユーザーは特定のジョブ要件に合わせて設定を調整する必要があるため、これはオプトイン機能です。	FLIP-309: ソースがバックログを処理しているときに、チェックポイント間隔を長くするサポート
System.out と System.err を Java ログにリダイレクトする	これはオプトイン機能です。Amazon Managed Service for Apache Flink では、本番環境でのベストプラクティスはネイティブ Java ロガーを使用することであるため、デフォルトの動作は System.out および System.err からの出力を無視することです。	FLIP-390: システム出力とエラーをサポートして LOG にリダイレクトするか、破棄する

Apache Flink 1.19.1 リリースドキュメントについては、「Apache Flink ドキュメント v1.19.1」を参照してください。

Amazon Managed Service for Apache Flink 1.19.1 の変更点

トレースレポーターのログ記録がデフォルトで有効になっている

Apache Flink 1.19.1 では、チェックポイントとリカバリのトレースが導入され、ユーザーはチェックポイントとジョブのリカバリの問題をより適切にデバッグできるようになりました。Amazon Managed Service for Apache Flink では、これらのトレースは CloudWatch ログストリームにログインされるため、ユーザーはジョブの初期化に費やされた時間を分割し、チェックポイントの履歴サイズを記録できます。

デフォルトの再起動戦略が指数遅延になりました

Apache Flink 1.19.1 では、指数遅延再起動戦略が大幅に改善されています。Flink 1.19.1 以降の Amazon Managed Service for Apache Flink では、Flink ジョブはデフォルトでエクスポネンシャル遅

延再起動戦略を使用します。つまり、ユーザージョブは一時的なエラーからより迅速に回復しますが、ジョブの再起動が続く場合、外部システムを過負荷にすることはありません。

Flink ジョブコードを使用する特定の Flink ジョブ設定が無効になっています

Apache Flink 1.18 以前のバージョンでは、プログラムによる設定の変更はすべて Amazon Managed Service for Apache Flink で許容され、一部の設定はサイレントに上書きされます。Apache Flink 1.19 以降では、Flink ジョブコードを使用した Flink ジョブ設定の変更を無効にしました。ユーザーがこの設定を指定すると、ジョブは ProgramInvocationException リストをスローします。詳細については、「FlinkRuntimeException：「許可されていない設定変更 (複数可) が検出されました」」を参照してください。

コンポーネント

コンポーネント	Version
Java	11 (推奨)
Python	3.11
Kinesis Data Analytics Flink ランタイム (aws-kinesisanalytics-runtime)	1.2.0
Connector	使用可能なコネクタの詳細については、「Apache Flink コネクタ」を参照してください。
「Apache Beam （ Beamアプリケーションのみ ）」	Flink 1.19 と互換性のある Apache Flink Runner はありません。詳細については、「Flink バージョンの互換性」を参照してください。

既知の問題

Apache ビーム

現在、Apache Beam には Flink 1.19 用の互換性のある Apache Flink Runner はありません。詳細については、「Flink バージョンの互換性」を参照してください。

Amazon Managed Service for Apache Flink Studio

Studio は Apache Zeppelin ノートブックを使用して、Apache Flink ストリーム処理アプリケーションの開発、デバッグ、実行のための単一インターフェイスの開発エクスペリエンスを提供します。Flink 1.19 のサポートを有効にするには、Zeppelin の Flink インタープリタのアップグレードが必要です。この作業は Zeppelin コミュニティでスケジュールされており、完了したらこれらのメモを更新します。Amazon Managed Service for Apache Flink Studio で Flink 1.15 を引き続き使用できます。詳細については、「Studio ノートブックの作成」を参照してください。

Amazon Managed Service for Apache Flink 1.18

Managed Service for Apache Flink が Apache Flink バージョン 1.18.1 をサポートするようになりました。Apache Flink 1.18.1 の Managed Service for Apache Flink サポートで導入された主な新機能と変更点について説明します。

> **Note**
>
> 以前にサポートされているバージョンの Apache Flink を使用していて、既存のアプリケーションを Apache Flink 1.18.1 にアップグレードする場合は、インプレース Apache Flink バージョンアップグレードを使用してアップグレードできます。インプレースバージョンアップグレードでは、スナップショット、ログ、メトリクス、タグ、Flink 設定など、Apache Flink バージョン全体で 1 つの ARN に対するアプリケーションのトレーサビリティを維持します。この機能は RUNNINGおよび READY状態で使用できます。詳細については、「Apache Flink のインプレースバージョンアップグレード」を参照してください。

サポートされている機能	説明	Apache Flink ドキュメントリファレンス
Opensearch コネクタ	このコネクタには、 を保証する at-least-onceシンクが含まれています。	github: Opensearch Connector
Amazon DynamoDB コネクタ	このコネクタには、 を保証する at-least-onceシンクが含まれています。	Amazon DynamoDB シンク

サポートされている機能	説明	Apache Flink ドキュメントリファレンス
MongoDB コネクタ	このコネクタには、保証を提供する at-least-onceソースとシンクが含まれています。	MongoDB コネクタ
Flink プランナーで Hive をデカップリングする	Hive ダイアレクトは、追加の JAR スワップなしで直接使用できます。	FLINK-26603: Flink プランナーで Hive をデカップリングする
RocksDBWriteBatchWrapper で WAL をデフォルトで無効にする	これにより、復旧時間が短縮されます。	FLINK-32326: RocksDBWriteBatchWrapper で WAL をデフォルトで無効にする
透かしの配置を有効にすると、透かしの集約パフォーマンスが向上します。	ウォーターマークの配置を有効にする際のウォーターマークの集約パフォーマンスを向上させ、関連するベンチマークを追加します。	FLINK-32524: 透かし集約パフォーマンス
ウォーターマークの配置を本番環境で使用できるようにする	大規模なジョブが過負荷になるリスクを排除 JobManager	FLINK-32548: ウォーターマークの配置の準備を整える
非同期シンク RateLimitingStratey 用に設定可能	RateLimitingStrategy では、スケーリングする対象、スケーリングするタイミング、スケーリングする量の決定を設定できます。	FLIP-242: 非同期シンク RateLimitingStrategy 用に設定可能な を導入
テーブルと列の統計を一括取得する	クエリのパフォーマンスが向上しました。	FLIP-247: 特定のパーティションのテーブルおよび列統計の一括フェッチ

Apache Flink 1.18.1 リリースドキュメントについては、「Apache Flink 1.18.1 Release Announcement」を参照してください。

Apache Flink 1.18 を使用した Amazon Managed Service for Apache Flink の変更点

Akka を Pekko に置き換えました

Apache Flink は、Apache Flink 1.18 で Akka を Pekko に置き換えました。この変更は、Apache Flink 1.18.1 以降の Managed Service for Apache Flink で完全にサポートされています。この変更の結果としてアプリケーションを変更する必要はありません。詳細については、FLINK-32468: Akka を Pekko に置き換える」を参照してください。

スレッドモードでの PyFlink ランタイム実行のサポート

この Apache Flink の変更により、Pyflink ランタイムフレームワークの新しい実行モードであるプロセスモードが導入されました。プロセスモードは、別のプロセスではなく、同じスレッドで Python ユーザー定義関数を実行できるようになりました。

コンポーネント

コンポーネント	Version
Java	11 (推奨)
Scala	バージョン 1.15 以降、Flink は Scala に依存しません。参考までに、MSF Flink 1.18 は Scala 3.3 (LTS) に対して検証されています。
Managed Service for Apache Flink Flink Runtime (aws-kinesisanalytics-runtime)	1.2.0
AWS Kinesis Connector (flink-connector-kinesis)[ソース〕	4.2.0 ~ 1.18
AWS Kinesis Connector (flink-connector-kinesis)[シンク〕	4.2.0 ~ 1.18
「Apache Beam （ Beamアプリケーションのみ ）」	バージョン 2.75.0 以前。詳細については、「Flink バージョンの互換性」を参照してください。

バグ修正

Apache Flink 1.18.1 の状態圧縮

Apache Flink は、すべてのチェックポイントとセーブポイントに対してオプションの圧縮 (デフォルト: オフ) を提供します。Apache Flink は、スナップショット圧縮が有効になっているときにオペレータの状態を適切に復元できなかった Flink 1.18.1 のバグを特定しました。これにより、データが失われたり、チェックポイントから復元できなくなる可能性があります。詳細については、FLINK-34063: スナップショット圧縮が有効になっている場合、ソース演算子のスケーリングを変更すると、一部の分割が失われます」を参照してください。

これを解決するために、Amazon Managed Service for Apache Flink は、将来のバージョンの Apache Flink に含まれる修正をバックポートしました。詳細については、「github: Always flush compression buffers」を参照してください。

既知の問題

Amazon Managed Service for Apache Flink Studio

Studio は Apache Zeppelin ノートブックを使用して、Apache Flink ストリーム処理アプリケーションの開発、デバッグ、実行のための単一インターフェイスの開発エクスペリエンスを提供します。Flink 1.18 のサポートを有効にするには、Zeppelin の Flink インタープリタのアップグレードが必要です。この作業は Zeppelin コミュニティでスケジュールされており、完了したらこれらのメモを更新します。Amazon Managed Service for Apache Flink Studio で Flink 1.15 を引き続き使用できます。詳細については、「Studio ノートブックの作成」を参照してください。

Amazon Managed Service for Apache Flink 1.15

Managed Service for Apache Flink は、Apache 1.15.2 の以下の新機能をサポートしています。

機能	説明	Apache Flip リファレンス
Async Sink	デベロッパーが以前の作業の半分未満でカスタム AWS コネクタを構築できるようにする非同期送信先を構築するための AWS 、貢献したフレームワーク。詳細については、	「FLIP-171: 非同期シンク」。

機能	説明	Apache Flip リファレンス
	「汎用非同期ベースシンク」を参照してください。	
Kinesis Data Firehose Sink	AWS は、非同期フレームワークを使用して新しい Amazon Kinesis Firehose シンクを提供しました。	Amazon Kinesis Data Firehose Sink
セーブポイントでの停止	セーブポイントでの停止によりクリーンな停止操作が保証され、さらに最も重要な利点としてセーブポイントに依存している顧客のために、1 回限りのセマンティクスをサポートします。	「FLIP-34: セーブポイントでのJob 終了/サスペンド」。
Scala デカップリング	ユーザーは Scala 3 を含む、すべての Scala バージョンから Java API を利用できるようになりました。顧客は、選択した Scala 標準ライブラリーを Scala アプリケーションにバンドルする必要があります。	「FLIP-28: フリンクテーブルを Scala フリーにするという長期的な目標」。
Scala	上記の Scala デカップリングを参照してください。	「FLIP-28: フリンクテーブルを Scala フリーにするという長期的な目標」。

機能	説明	Apache Flip リファレンス
Unified Connector Metrics	Flink はジョブ、タスク、オペレータの「スタンダードメトリクス」を定義しています。Managed Service for Apache Flink は引き続きシンクとソースのメトリクスをサポートし、1.15 では Availability Metrics の `fullRestarts` と並行して `numRestarts` を導入します。	「FLIP-33: Standardize Connector Metrics」および「FLIP-179: Expose Standardized Operator Metrics」。
完了したタスクのチェックポイント機能	この機能は Flink 1.15 ではデフォルトで有効になっており、ジョブグラフの一部がすべてのデータの処理を終了してもチェックポイントの実行を継続できるようになっています。それにバインドされた（バッチ）ソースが含まれている場合に発生する可能性があります。	「FLIP-147: タスク終了後のチェックポイントのサポート」。

Apache Flink 1.15 における Amazon Managed Service for Apache Flink の変更点

Studio のノートブック

Managed Service for Apache Flink Studio は、Apache Flink 1.15 をサポートするようになりました。Managed Service for Apache Flink Studio は、Apache Zeppelin ノートブックを利用して、Apache Flink ストリーム処理アプリケーションの開発、コード・デバッグ、実行のための単一インターフェースの開発体験を提供します。Managed Service for Apache Flink Studio の詳細と入門については、「Apache Flink 用 Managed Service で Studio ノートブックを使用する」を参照してください。

「EFO コネクター」

Managed Service for Apache Flink バージョン 1.15 にアップグレードする際は、必ず最新の EFO コネクタ (バージョン 1.15.3 以降) を使用してください。理由の詳細については、「FLINK-29324」を参照してください。

「Scala デカップリング」

Flink 1.15.2 以降では、任意の Scala スタンダードライブラリを Scala アプリケーションにバンドルする必要が出てきます。

Kinesis Data Firehose Sink

Managed Service for Apache Flink バージョン 1.15 にアップグレードする場合は、最新の Amazon Kinesis Data Firehose Sinkを使用していることを確認してください。

Kafka Connectors

Apache Flink バージョン 1.15 の Amazon Managed Service for Apache Flink にアップグレードする場合は、最新の Kafka コネクタ API を使用していることを確認してください。Apache Flink は非推奨FlinkKafkaConsumerとなり、Kafka シンクのFlinkKafkaProducerこれらの APIs は Flink 1.15 の Kafka にコミットできません。KafkaSource と を使用していることを確認しますKafkaSink。

コンポーネント

コンポーネント	Version
Java	11 (推奨)
Scala	2.12
Managed Service for Apache Flink Flink Runtime (aws-kinesisanalytics-runtime)	1.2.0
AWS Kinesis Connector (flink-connector-kinesis)	1.15.4
「Apache Beam （ Beamアプリケーションのみ ）」	Jackson バージョン 2.12.2 を搭載した 2.33.0

Managed Service for Apache Flink の以前のバージョン情報

> **Note**
>
> Apache Flink バージョン 1.6、1.8、および 1.11 は、Apache Flink コミュニティで 3 年以上サポートされていません。Amazon Managed Service for Apache Flink では、2024 年 11 月 5 日にこれらのバージョンを廃止する予定です。この日以降、これらの Flink バージョン用の新しいアプリケーションを作成することはできません。現時点では、既存のアプリケーションの実行を継続できます。Amazon Managed Service for Apache Flink のインプレースバージョンアップグレード機能を使用して、アプリケーションをステートリーにアップグレードできます。詳細については、「」を参照してくださいApache Flink のインプレースバージョンアップグレード。

Apache Flink のバージョン 1.15.2 および 1.13.2 は Managed Service for Apache Flink でサポートされていますが、Apache Flink コミュニティではサポートされなくなりました。

このトピックには、次のセクションが含まれています。

- 以前の Apache Flink バージョンでの Apache Flink Kinesis Streams コネクタの使用
- Apache Flink 1.8.2 を使用したアプリケーションの構築
- Apache Flink 1.6.2 を使用したアプリケーションの構築
- アプリケーションのアップグレード
- Apache Flink 1.6.2 および 1.8.2 で利用可能なコネクタ
- 開始方法: Flink 1.13.2
- 開始方法: Flink 1.11.1 - 非推奨
- 開始方法: Flink 1.8.2 - 非推奨
- 開始方法: Flink 1.6.2 - 非推奨
- Managed Service for Apache Flink の以前のバージョン (レガシー) の例

以前の Apache Flink バージョンでの Apache Flink Kinesis Streams コネクタの使用

Apache Flink Kinesis Streams コネクタは、バージョン 1.11 以前の Apache Flink には含まれていませんでした。以前のバージョンの Apache Flink で Apache Flink Kinesis コネクタを使用するには、

アプリケーションが使用する Apache Flink のバージョンをダウンロード、コンパイル、インストールする必要があります。このコネクタは、アプリケーションのソースとして使用される Kinesis ストリームからデータを消費したり、アプリケーションの出力に使用される Kinesis ストリームにデータを書き込んだりするために使用されます。

> **Note**
>
> 「KPL バージョン 0.14.0」以降でコネクタを構築していることを確認してください。

Apache Flink バージョン 1.8.2 のソースコードをダウンロードしてインストールするには、以下の手順に従います。

1. 「Apache Maven」がインストールされていて、 JAVA_HOME 環境変数が JRE ではなく JDK を指していることを確認してください。次のコマンドで Apache Maven のインストールをテストすることができます。

   ```
   mvn -version
   ```

2. Apache Flink バージョン 1.8.2 のソースコードをダウンロードします。

   ```
   wget https://archive.apache.org/dist/flink/flink-1.8.2/flink-1.8.2-src.tgz
   ```

3. Apache Flink ソースコードを解凍します。

   ```
   tar -xvf flink-1.8.2-src.tgz
   ```

4. Apache Flink ソースコードディレクトリに移動します。

   ```
   cd flink-1.8.2
   ```

5. Apache Flink をコンパイルしてインストールします。

   ```
   mvn clean install -Pinclude-kinesis -DskipTests
   ```

   > **Note**
   >
   > Microsoft Windows で Flink をコンパイルする場合は、 -Drat.skip=true パラメータを追加する必要があります。

Apache Flink 1.8.2 を使用したアプリケーションの構築

このセクションには、Apache Flink 1.8.2 で動作する Apache Flink アプリケーション用 Managed Service を構築するために使用するコンポーネントに関する情報が含まれています。

Apache Flink 用 Managed Service アプリケーションには、次のコンポーネントバージョンを使用してください。

コンポーネント	Version
Java	1.8 (推奨)
Apache Flink	1.8.2
Managed Service for Apache Flink for Flink Runtime (aws-kinesisanalytics-runtime)	1.0.1
Apache Flink Flink コネクタ用 Managed Service (aws-kinesisanalytics-flink)	1.0.1
Apache Maven	3.1

Apache Flink 1.8.2 を使用してアプリケーションをコンパイルするには、以下のパラメーターを指定して Maven を実行します。

```
mvn package -Dflink.version=1.8.2
```

Apache Flink バージョン 1.8.2 を使用する Apache Flink アプリケーション用 Managed Service の `pom.xml` ファイルの例については、「Apache Flink 1.8.2 用 Managed Service 入門アプリケーション」を参照してください。

Apache Flink アプリケーション用 Managed Service のアプリケーションコードを構築して使用する方法については、 アプリケーションの作成 を参照してください。

Apache Flink 1.6.2 を使用したアプリケーションの構築

このセクションには、Apache Flink 1.6.2 で動作する Apache Flink アプリケーション用 Managed Service を構築するために使用するコンポーネントに関する情報が含まれています。

Apache Flink 用 Managed Service アプリケーションには、次のコンポーネントバージョンを使用してください。

コンポーネント	Version
Java	1.8 (推奨)
AWS Java SDK	1.11.379
Apache Flink	1.6.2
Managed Service for Apache Flink for Flink Runtime (aws-kinesisanalytics-runtime)	1.0.1
Apache Flink Flink コネクタ用 Managed Service (aws-kinesisanalytics-flink)	1.0.1
Apache Maven	3.1
Apache Beam	Apache Flink 1.6.2 ではサポートされていません。

> **Note**
>
> Apache Flink Runtime バージョン「1.0.1」用 Managed Service を使用する場合は、アプリケーションコードをコンパイルする際に `-Dflink.version` パラメータを使用するのではなく、 `pom.xml` ファイルに Apache Flink のバージョンを指定します。

Apache Flink バージョン 1.6.2 を使用する Apache Flink アプリケーション 用 Managed Service の「`pom.xml`」ファイルの例については、「Apache Flink 1.6.2 用 Managed Service 入門アプリケーション」を参照してください。

Apache Flink アプリケーション用 Managed Service のアプリケーションコードを構築して使用する方法については、 アプリケーションの作成 を参照してください。

アプリケーションのアップグレード

Amazon Managed Service for Apache Flink アプリケーションの Apache Flink バージョンをアップグレードするには、 AWS CLI、 AWS SDK AWS CloudFormation、または を使用して、インプレース Apache Flink バージョンアップグレード機能を使用します AWS Management Console。詳細については、「Apache Flink のインプレースバージョンアップグレード」を参照してください。

この機能は、 READYまたは RUNNING状態の Amazon Managed Service for Apache Flink で使用する既存のアプリケーションで使用できます。

Apache Flink 1.6.2 および 1.8.2 で利用可能なコネクタ

Apache Flink フレームワークには、さまざまなソースのデータにアクセスするためのコネクターが含まれています。

- Apache Flink 1.6.2 フレームワークで使用可能なコネクタの情報については、「Apache Flink ドキュメント (1.6.2)」の「Connectors (1.6.2)」を参照してください。
- Apache Flink 1.8.2 フレームワークで利用可能なコネクタの情報については、「Apache Flink ドキュメント(1.8.2)」の「Connectors (1.8.2)」を参照してください。

開始方法: Flink 1.13.2

このセクションでは、Managed Service for Apache Flink と DataStream API の基本概念を紹介します。アプリケーションの作成とテストに使用できるオプションについて説明します。また、このガイドのチュートリアルを完了し、初めてアプリケーションを作成するのに必要なツールのインストール方法についても説明します。

トピック

- Managed Service for Apache Flink アプリケーションのコンポーネント
- 演習を完了するための前提条件
- ステップ 1: AWS アカウントを設定し、管理者ユーザーを作成する
- 次のステップ
- ステップ 2: AWS Command Line Interface （ AWS CLI) を設定する
- ステップ 3: Managed Service for Apache Flink アプリケーションを作成して実行する
- ステップ 4: リソースをクリーンアップ AWS する
- ステップ 5: 次のステップ

Managed Service for Apache Flink アプリケーションのコンポーネント

Apache Flink アプリケーション用 Managed Service では、Apache Flink ランタイムを使用して入力を処理し、出力を生成する Java/Apache Maven または Scala アプリケーションを使用してデータを処理します。

Apache Flink アプリケーション用 Managed Serviceには、以下のコンポーネントがあります。

- 「ランタイムプロパティ:」「ランタイムプロパティ」を使用すると、アプリケーションコードを再コンパイルせずにアプリケーションを設定できます。
- 「ソース:」 アプリケーションは 「ソース」 を使用してデータを消費します。ソースコネクタは、Kinesis データストリーム、Amazon S3 バケットなどからデータを読み取ります。詳細については、「[Sources] (出典)」を参照してください。
- 「オペレータ:」アプリケーションは 1 つ以上の「オペレータ」を使用してデータを処理します。オペレータはデータを変換、強化、または集約できます。詳細については、 DataStream API 演算子 を参照してください。
- 「シンク:」アプリケーションは「シンク」を使用して外部ソースにデータを生成します。シンクコネクタは、Kinesis データストリーム、Firehose ストリーム、Amazon S3 バケットなどにデータを書き込みます。詳細については、「シンク」を参照してください。

アプリケーションコードを作成、コンパイル、パッケージ化したら、コードパッケージを Amazon Simple Storage Service (Amazon S3) バケットにアップロードします。次に、Apache Flink アプリケーション用 Managed Serviceを作成します。コードパッケージの場所、ストリーミングデータソースとして Kinesis データストリームを渡し、通常はアプリケーションの処理済みデータを受け取るストリーミングまたはファイルの場所を渡します。

演習を完了するための前提条件

このガイドの手順を完了するには、以下が必要です。

- Java 開発キット (JDK) バージョン 11。JAVA_HOME 環境変数を、JDK のインストール場所を指すように設定します。
- 開発環境 (Eclipse Java Neon や IntelliJ Idea など) を使用してアプリケーションを開発し、コンパイルすることをお勧めします。
- Git クライアント。Git クライアントをまだインストールしていない場合は、インストールします。

- Apache Maven Compiler Plugin。Maven が作業パスに含まれている必要があります。Apache Maven のインストールをテストするには、次のように入力します。

```
$ mvn -version
```

開始するには、ステップ 1: AWS アカウントを設定し、管理者ユーザーを作成するに進みます。

ステップ 1: AWS アカウントを設定し、管理者ユーザーを作成する

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環として、通話呼び出しを受け取り、電話キーパッドで検証コードを入力するように求められます。

   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。セキュリティのベストプラクティスとして、ユーザーに管理アクセスを割り当て、ルートユーザーのみを使用してルートユーザーアクセスが必要なタスクを実行してください。

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

のセキュリティ保護 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 「ルートユーザーの仮想 MFA デバイスを有効にする (コンソール)」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、「 ユーザーガイド」の「デフォルト で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AWS IAM Identity Center 」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「グループの参加」を参照してください。

プログラム的なアクセス権を付与する

ユーザーが の AWS 外部で を操作する場合は、プログラムによるアクセスが必要です AWS Management Console。プログラムによるアクセスを許可する方法は、 にアクセスするユーザーのタイプによって異なります AWS。

ユーザーにプログラマチックアクセス権を付与するには、以下のいずれかのオプションを選択します。

プログラマチックアクセス権を必要とするユーザー	目的	方法
ワークフォースアイデンティティ (IAM Identity Center で管理されているユーザー)	一時的な認証情報を使用して、、 AWS SDKs AWS CLI、または AWS APIs。	使用するインターフェイス用の手引きに従ってください。 • については AWS CLI、「 ユーザーガイド」の AWS CLI 「 を使用するための の設定 AWS IAM Identity CenterAWS Command Line Interface 」を参照してください。 • AWS SDKs 、ツール、AWS APIs「 SDK とツールのリファレンスガイド」の「IAM Identity Center 認証」を参照してください。AWS SDKs

プログラマチックアクセス権を必要とするユーザー	目的	方法
IAM	一時的な認証情報を使用して、、 AWS SDKs AWS CLI、または AWS APIs。	「IAM ユーザーガイド」の「 AWS リソースでの一時的な認証情報の使用」の手順に従います。
IAM	(非推奨) 長期認証情報を使用して、、 AWS SDKs AWS CLI、または AWS APIs。	使用するインターフェイス用の手引きに従ってください。 • については AWS CLI、「 AWS Command Line Interface ユーザーガイド」の「IAM ユーザー認証情報を使用した認証」を参照してください。 • AWS SDKs「 SDK とツールのリファレンスガイド」の「長期的な認証情報を使用した認証」を参照してください。 AWS SDKs • AWS APIsユーザーガイド」の「IAM ユーザーのアクセスキーの管理」を参照してください。

次のステップ

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

次のステップ

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

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

このステップでは、Managed Service for Apache Flink で使用する をダウンロードして設定 AWS CLI します。

> **Note**
>
> このガイドの使用開始実習では、操作を実行するために、アカウントの管理者の認証情報 (adminuser) を使用していることが前提となっています。

> **Note**
>
> が既に AWS CLI インストールされている場合は、アップグレードして最新の機能を取得する必要がある場合があります。詳細については、「AWS Command Line Interface ユーザーガイド」の「AWS Command Line Interfaceのインストール」を参照してください。のバージョンを確認するには AWS CLI、次のコマンドを実行します。
>
> ```
> aws --version
> ```
>
> このチュートリアルの演習では、次の AWS CLI バージョン 以降が必要です。
>
> ```
> aws-cli/1.16.63
> ```

を設定するには 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。
   - AWS Command Line Interfaceのインストール
   - AWS CLIの設定
2. 管理者ユーザーの名前付きプロファイルを config ファイルに追加します AWS CLI 。 AWS CLI コマンドを実行するときに、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 AWS リージョンのリストについては、「」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。

> **Note**
>
> このチュートリアルのサンプルコードとコマンドでは、米国西部 (オレゴン) リージョンを使用しています。別の リージョンを使用するには、このチュートリアルのコードとコマンドのリージョンを、使用したいリージョンに変更します。

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
```

AWS アカウントと をセットアップしたら AWS CLI、次の演習を試してサンプルアプリケーションを設定し、 end-to-end セットアップをテストできます。

次のステップ

ステップ 3: Managed Service for Apache Flink アプリケーションを作成して実行する

ステップ 3: Managed Service for Apache Flink アプリケーションを作成して実行する

この演習では、データストリームをソースおよびシンクとして使用して、Managed Service for Apache Flink アプリケーションを作成します。

このセクションには、以下のステップが含まれています。

- 2 つの Amazon Kinesis データストリームを作成する
- サンプルレコードを入力ストリームに書き込む
- Apache Flink ストリーミング Java コードをダウンロードして調べる
- アプリケーションコードをコンパイルする
- Apache Flink ストリーミング Java コードをアップロードする
- Managed Service for Apache Flink アプリケーションを作成して実行する
- 次のステップ

2 つの Amazon Kinesis データストリームを作成する

この演習で Apache Flink アプリケーションのマネージドサービスを作成する前に、2 つの Kinesis データストリーム (ExampleInputStream と ExampleOutputStream) を作成する必要があります。アプリケーションでは、これらのストリームを使用してアプリケーションの送信元と送信先のストリームを選択します。

これらのストリームは Amazon Kinesis コンソールまたは次の AWS CLI コマンドを使用して作成できます。コンソールでの操作方法については、「Amazon Kinesis Data Streams デベロッパーガイド」 の 「データストリームの作成および更新」 を参照してください。

データストリームを作成するには (AWS CLI)

1. 最初のストリーム (ExampleInputStream) を作成するには、次の Amazon Kinesis create-stream AWS CLI コマンドを使用します。

```
$ aws kinesis create-stream \
--stream-name ExampleInputStream \
--shard-count 1 \
--region us-west-2 \
--profile adminuser
```

2. 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 ExampleOutputStream \
--shard-count 1 \
--region us-west-2 \
--profile adminuser
```

サンプルレコードを入力ストリームに書き込む

このセクションでは、Python スクリプトを使用して、アプリケーションが処理するサンプルレコードをストリームに書き込みます。

> **Note**
>
> このセクションでは AWS SDK for Python (Boto) が必要です。

1. 次の内容で、stock.py という名前のファイルを作成します。

```
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', region_name='us-west-2'))
```

2. このチュートリアルの後半では、アプリケーションにデータを送信する stock.py スクリプトを実行します。

```
$ python stock.py
```

Apache Flink ストリーミング Java コードをダウンロードして調べる

この例の Java アプリケーションコードは、 から入手できます GitHub。アプリケーションコードをダウンロードするには、次の操作を行います。

1. 次のコマンドを使用してリモートリポジトリのクローンを作成します。

```
git clone https://github.com/aws-samples/amazon-kinesis-data-analytics-examples.git
```

2. amazon-kinesis-data-analytics-java-examples/GettingStarted ディレクトリに移動します。

アプリケーションコードに関して、以下の点に注意してください。

- 「Project Object Model (pom.xml)」ファイルには、Managed Service for Apache Flink 用ライブラリなど、アプリケーションの設定と依存関係に関する情報が含まれています。
- BasicStreamingJob.java ファイルには、アプリケーションの機能を定義する main メソッドが含まれています。
- アプリケーションは Kinesis ソースを使用して、ソースストリームから読み取りを行います。次のスニペットでは、Kinesis ソースが作成されます。

```
return env.addSource(new FlinkKinesisConsumer<>(inputStreamName,
                new SimpleStringSchema(), inputProperties));
```

- アプリケーションでは、ソースおよびシンクコネクタを作成し、StreamExecutionEnvironment オブジェクトを使用して外部リソースにアクセスします。
- アプリケーションでは、静的プロパティを使用してソースおよびシンクコネクタを作成します。動的なアプリケーションプロパティを使用するには、createSourceFromApplicationProperties および createSinkFromApplicationProperties メソッドを使用してコネクタを作成します。これらのメソッドは、アプリケーションのプロパティを読み取ってコネクタを設定します。

  ランタイムプロパティの詳細については、ランタイムプロパティ を参照してください。

アプリケーションコードをコンパイルする

このセクションでは、Apache Maven コンパイラを使用してアプリケーション用の Java コードを作成します。Apache Maven と Java 開発キット (JDK) をインストールする方法については、演習を完了するための前提条件を満たすを参照してください。

アプリケーションコードをコンパイルするには

1. アプリケーションコードを使用するには、コードをコンパイルして JAR ファイルにパッケージ化します。コードのコンパイルとパッケージ化には次の 2 通りの方法があります。

   - コマンドライン Maven ツールを使用します。pom.xml ファイルが格納されているディレクトリで次のコマンドを実行して JAR ファイルを作成します。

     ```
     mvn package -Dflink.version=1.13.2
     ```

   - 開発環境を使用します。詳細については、開発環境のドキュメントを参照してください。

> **Note**
>
> 提供されているソースコードは Java 11 のライブラリーに依存しています。

パッケージは JAR ファイルとしてアップロードすることも、圧縮して ZIP ファイルとしてアップロードすることもできします。を使用してアプリケーションを作成する場合は AWS CLI、コードコンテンツタイプ (JAR または ZIP) を指定します。

2. コンパイル中にエラーが発生した場合は、`JAVA_HOME` 環境変数が正しく設定されていることを確認します。

アプリケーションのコンパイルに成功すると、次のファイルが作成されます。

```
target/aws-kinesis-analytics-java-apps-1.0.jar
```

Apache Flink ストリーミング Java コードをアップロードする

このセクションでは、Amazon Simple Storage Service (Amazon S3) バケットを作成し、アプリケーションコードをアップロードします。

アプリケーションコードをアップロードするには

1. Amazon S3 コンソール (https://console.aws.amazon.com/s3/) を開きます。
2. [バケットを作成] を選択します。
3. [Bucket name (バケット名)] フィールドに**ka-app-code-*<username>***と入力します。バケット名にユーザー名などのサフィックスを追加して、グローバルに一意にします。[次へ] をクリックします。
4. 設定オプションのステップでは、設定をそのままにし、[次へ] を選択します。
5. アクセス許可の設定のステップでは、設定をそのままにし、[次へ] を選択します。
6. [バケットを作成] を選択します。
7. Amazon S3 コンソールで、ka-app-code-***<username>*** バケットを選択し、アップロード を選択します。
8. ファイルの選択のステップで、[ファイルを追加] を選択します。前のステップで作成した `aws-kinesis-analytics-java-apps-1.0.jar` ファイルに移動します。[次へ] をクリックします。
9. オブジェクトの設定を変更する必要はないので、[アップロード] を選択してください。

アプリケーションコードが Amazon S3 バケットに保存され、アプリケーションからアクセスできるようになります。

Managed Service for Apache Flink アプリケーションを作成して実行する

コンソールまたは AWS CLIのいずれかを使用してManaged Service for Apache Flink を作成し、実行することができます。

> **Note**
>
> コンソールを使用してアプリケーションを作成すると、 AWS Identity and Access Management (IAM) と Amazon CloudWatch Logs リソースが自動的に作成されます。を使用してアプリケーションを作成するときは AWS CLI、これらのリソースを個別に作成します。

トピック

- アプリケーションの作成と実行 (コンソール)
- アプリケーションの作成と実行 (AWS CLI)

アプリケーションの作成と実行 (コンソール)

以下の手順を実行し、コンソールを使用してアプリケーションを作成、設定、更新、および実行します。

アプリケーションの作成

1. https://console.aws.amazon.com/flink で Apache Flink 用 Managed Serviceコンソールを開く
2. Managed Service for Apache Flinkのダッシュボードで、「分析アプリケーションの作成」 を選択します。
3. 「Managed Service for Apache Flink-アプリケーションの作成」ページで、次のようにアプリケーションの詳細を入力します。
   - [アプリケーション名] には **`MyApplication`** と入力します。
   - [Description (説明)] に **`My java test app`** と入力します。
   - [ランタイム] には、[Apache Flink] を選択します。
   - バージョンプルダウンは「Apache Flink バージョン 1.13」のままにしておきます。
4. [アクセス許可] には、[IAM ロールの作成 / 更新**`kinesis-analytics-MyApplication-us-west-2`**] を選択します。

5. [Create application] を選択します。

> **Note**
>
> コンソールを使用して Apache Flink アプリケーション用 Managed Service を作成する場合は、IAM ロールとポリシーをアプリケーションが自動的に作成するオプションを選択できます。アプリケーションではこのロールとポリシーを使用して、依存リソースにアクセスします。これらの IAM リソースは、次のようにアプリケーション名とリージョンを使用して命名されます。
>
> - ポリシー: `kinesis-analytics-service-MyApplication-us-west-2`
> - ロール: `kinesisanalytics-MyApplication-us-west-2`

IAM ポリシーを編集する

IAM ポリシーを編集し、Kinesis Data Streamsにアクセスするための許可を追加します。

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. [ポリシー] を選択します。前のセクションでコンソールによって作成された **kinesis-analytics-service-MyApplication-us-west-2** ポリシーを選択します。
3. [概要] ページで、[ポリシーの編集] を選択します。[JSON] タブを選択します。
4. 次のポリシー例で強調表示されているセクションをポリシーに追加します。サンプルのアカウント ID (*012345678901*) を自分のアカウント ID に置き換えます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ReadCode",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": [
                "arn:aws:s3:::ka-app-code-username/aws-kinesis-analytics-java-apps-1.0.jar"
            ]
```

```
        },
        {
            "Sid": "DescribeLogGroups",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:*"
            ]
        },
        {
            "Sid": "DescribeLogStreams",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogStreams"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:*"
            ]
        },
        {
            "Sid": "PutLogEvents",
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:kinesis-analytics-log-stream"
            ]
        },
        {
            "Sid": "ReadInputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
        },
        {
            "Sid": "WriteOutputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
```

```
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleOutputStream"
        }
    ]
}
```

アプリケーションを設定する

1. MyApplication ページで、 の設定 を選択します。
2. [Configure application] ページで、[Code location] を次のように指定します。
   - [Amazon S3 バケット] で、**ka-app-code-*<username>***と入力します。
   - [Amazon S3 オブジェクトへのパス] で、**aws-kinesis-analytics-java-apps-1.0.jar**と入力します。
3. [Access to application resources] の [Access permissions] では、[Create / update IAM role**kinesis-analytics-MyApplication-us-west-2**] を選択します。
4. 次のように入力します。

グループ ID	キー	値
ProducerConfigProperties	**flink.inputstream.initpos**	**LATEST**
ProducerConfigProperties	**aws.region**	**us-west-2**
ProducerConfigProperties	**AggregationEnabled**	**false**

5. [Monitoring] の [Monitoring metrics level] が [Application] に設定されていることを確認します。
6. CloudWatch のログ記録では、有効化チェックボックスをオンにします。
7. [更新] を選択します。

> **Note**
>
> Amazon CloudWatch ログ記録を有効にすると、Managed Service for Apache Flink によってロググループとログストリームが作成されます。これらのリソースの名前は次のとおりです。
>
> - ロググループ: `/aws/kinesis-analytics/MyApplication`
> - ログストリーム: `kinesis-analytics-log-stream`

アプリケーションを実行する

Flink ジョブグラフは、アプリケーションを実行し、Apache Flink ダッシュボードを開き、目的の Flink ジョブを選択すると表示できます。

アプリケーションの停止

MyApplication ページで、停止 を選択します。アクションを確認します。

アプリケーションの更新

コンソールを使用して、アプリケーションのプロパティ、モニタリング設定、アプリケーション JAR の場所またはファイル名などのアプリケーション設定を更新できます。アプリケーションコードを更新する必要がある場合は、アプリケーション JAR を Amazon S3 バケットから再ロードすることもできます。

MyApplication ページで、 の設定 を選択します。アプリケーションの設定を更新し、[更新] を選択します。

アプリケーションの作成と実行 (AWS CLI)

このセクションでは、 を使用して Managed Service for Apache Flink アプリケーション AWS CLI を作成して実行します。Managed Service for Apache Flink は、 `kinesisanalyticsv2` AWS CLI コマンドを使用して Managed Service for Apache Flink アプリケーションを作成して操作します。

許可ポリシーを作成する

> **Note**
>
> アプリケーションのアクセス権限ポリシーとロールを作成する必要があります。これらの IAM リソースを作成しない場合、アプリケーションはそのデータストリームやログストリームにアクセスできません。

まず、2 つのステートメントを含むアクセス許可ポリシーを作成します。1 つは、ソースストリームの `read` アクションに対するアクセス許可を付与し、もう 1 つはシンクストリームの `write` アクションに対するアクセス許可を付与します。次に、IAM ロール (次のセクションで作成) にポリシーをアタッチします。そのため、 Managed Service for Apache Flinkがこのロールを引き受けると、ソースストリームからの読み取りとシンクストリームへの書き込みを行うために必要なアクセス許可がサービスに付与されます。

次のコードを使用して `AKReadSourceStreamWriteSinkStream` アクセス許可ポリシーを作成します。*`username`* を Amazon S3 バケットの作成に使用したユーザー名に置き換え、アプリケーションコードを保存します。Amazon リソースネーム (ARN) のアカウント ID (*`012345678901`*) を自分のアカウント ID に置き換えます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "S3",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": ["arn:aws:s3:::ka-app-code-username",
                "arn:aws:s3:::ka-app-code-username/*"
            ]
        },
        {
            "Sid": "ReadInputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
```

```
        },
        {
            "Sid": "WriteOutputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleOutputStream"
        }
    ]
}
```

アクセス許可ポリシーを作成する step-by-step 手順については、IAM ユーザーガイドの「チュートリアル: 最初のカスタマー管理ポリシーの作成とアタッチ」を参照してください。

> **Note**
>
> その他のアマゾンサービスにアクセスするには、 AWS SDK for Javaを使用します。Managed Service for Apache Flink は、アプリケーションに関連付けられているサービス実行 IAM ロールに、SDK が必要とする認証情報を自動的に設定します。追加の手順は必要ありません。

IAM ロールを作成する

このセクションでは、Managed Service for Apache Flink アプリケーションがソースストリームを読み取り、シンクストリームに書き込むために想定できる IAM ロールを作成します。

Apache Flink 用 Managed Service は、許可なしにはストリームにアクセスできません。IAM ロールを介してこれらの許可を付与します。各 IAM ロールには、2 つのポリシーがアタッチされます。信頼ポリシーは、ロールを引き受けるための許可を Managed Service for Apache Flink 付与し、許可ポリシーは、ロールを引き受けた後に Managed Service for Apache Flink が実行できる事柄を決定します。

前のセクションで作成したアクセス許可ポリシーをこのロールにアタッチします。

IAM ロールを作成するには

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. ナビゲーションペインで [Roles (ロール)]、[Create Role (ロールの作成)] の順に選択します。

3. [信頼されるエンティティの種類を選択] で、[AWS のサービス] を選択します。[このロールを使用するサービスを選択] で、[Kinesis Analytics] を選択します。[ユースケースの選択] で、[Kinesis Analytics] を選択します。

   [Next: Permissions] (次のステップ: 許可) を選択します。

4. [アクセス権限ポリシーをアタッチする] ページで、[Next: Review] (次: 確認) を選択します。ロールを作成した後に、アクセス許可ポリシーをアタッチします。

5. [Create role (ロールの作成)] ページで、ロールの名前に **`MF-stream-rw-role`** を入力します。[ロールの作成] を選択します。

   これで、`MF-stream-rw-role` と呼ばれる新しい IAM ロールが作成されます。次に、ロールの信頼ポリシーとアクセス許可ポリシーを更新します。

6. アクセス許可ポリシーをロールにアタッチします。

   > **Note**
   >
   > この演習では、Managed Service for Apache Flink が、Kinesis データストリーム (ソース) からのデータの読み取りと、別の Kinesis データストリームへの出力の書き込みの両方を実行するためにこのロールを引き受けます。このため、前のステップで作成したポリシー、the section called “許可ポリシーを作成する” をアタッチします。

   a. [概要] ページで、[アクセス許可] タブを選択します。

   b. [Attach Policies (ポリシーのアタッチ)] を選択します。

   c. 検索ボックスに**`AKReadSourceStreamWriteSinkStream`**(前のセクションで作成したポリシー) と入力します。

   d. AKReadSourceStreamWriteSinkStream ポリシーを選択し、ポリシー をアタッチ を選択します。

これで、アプリケーションがリソースにアクセスするために使用するサービスの実行ロールが作成されました。新しいロールの ARN を書き留めておきます。

ロールを作成する step-by-step 手順については、IAM ユーザーガイドの「IAM ロールの作成 (コンソール)」を参照してください。

Managed Service for Apache Flink アプリケーションを作成する

1. 次の JSON コードを create_request.json という名前のファイルに保存します。サンプルロールの ARN を、前に作成したロールの ARN に置き換えます。バケット ARN のサフィックス (*username*) を、前のセクションで選択したサフィックスに置き換えます。サービス実行ロールのサンプルのアカウント ID (*012345678901*) を、自分のアカウント ID に置き換えます。

```
{
    "ApplicationName": "test",
    "ApplicationDescription": "my java test app",
    "RuntimeEnvironment": "FLINK-1_15",
    "ServiceExecutionRole": "arn:aws:iam::012345678901:role/MF-stream-rw-role",
    "ApplicationConfiguration": {
        "ApplicationCodeConfiguration": {
            "CodeContent": {
                "S3ContentLocation": {
                    "BucketARN": "arn:aws:s3:::ka-app-code-username",
                    "FileKey": "aws-kinesis-analytics-java-apps-1.0.jar"
                }
            },
            "CodeContentType": "ZIPFILE"
        },
        "EnvironmentProperties":  {
         "PropertyGroups": [
            {
               "PropertyGroupId": "ProducerConfigProperties",
               "PropertyMap" : {
                    "flink.stream.initpos" : "LATEST",
                    "aws.region" : "us-west-2",
                    "AggregationEnabled" : "false"
               }
            },
            {
               "PropertyGroupId": "ConsumerConfigProperties",
               "PropertyMap" : {
                    "aws.region" : "us-west-2"
               }
            }
         ]
      }
    }
}
```

2. 前述のリクエストを指定して CreateApplication アクションを実行し、アプリケーションを作成します。

```
aws kinesisanalyticsv2 create-application --cli-input-json file://
create_request.json
```

これでアプリケーションが作成されました。次のステップでは、アプリケーションを起動します。

アプリケーションの起動

このセクションでは、StartApplication アクションを使用してアプリケーションを起動します。

アプリケーションを起動するには

1. 次の JSON コードを start_request.json という名前のファイルに保存します。

```
{
    "ApplicationName": "test",
    "RunConfiguration": {
        "ApplicationRestoreConfiguration": {
         "ApplicationRestoreType": "RESTORE_FROM_LATEST_SNAPSHOT"
         }
    }
}
```

2. 前述のリクエストを指定して StartApplication アクションを実行し、アプリケーションを起動します。

```
aws kinesisanalyticsv2 start-application --cli-input-json file://start_request.json
```

アプリケーションが実行されます。Amazon CloudWatch コンソールで Managed Service for Apache Flink メトリクスをチェックして、アプリケーションが動作していることを確認できます。

アプリケーションの停止

このセクションでは、StopApplication アクションを使用してアプリケーションを停止します。

アプリケーションを停止するには

1. 次の JSON コードを stop_request.json という名前のファイルに保存します。

```
{
   "ApplicationName": "test"
}
```

2. 次のリクエストを指定して StopApplication アクションを実行し、アプリケーションを停止します。

```
aws kinesisanalyticsv2 stop-application --cli-input-json file://stop_request.json
```

アプリケーションが停止します。

CloudWatch ログ記録オプションを追加する

を使用して AWS CLI 、Amazon CloudWatch ログストリームをアプリケーションに追加できます。アプリケーションで CloudWatch ログを使用する方法については、「」を参照してくださいthe section called “ログ記録の設定”。

環境プロパティを更新します

このセクションでは、「UpdateApplication」アクションを使用して、アプリケーションコードを再コンパイルせずにアプリケーションの環境プロパティを変更します。この例では、ソースストリームおよびレプリケート先ストリームのリージョンを変更します。

アプリケーションの環境プロパティを更新します

1. 次の JSON コードを update_properties_request.json という名前のファイルに保存します。

```
{"ApplicationName": "test",
   "CurrentApplicationVersionId": 1,
   "ApplicationConfigurationUpdate": {
      "EnvironmentPropertyUpdates": {
         "PropertyGroups": [
            {
               "PropertyGroupId": "ProducerConfigProperties",
               "PropertyMap" : {
                    "flink.stream.initpos" : "LATEST",
```

```
                    "aws.region" : "us-west-2",
                    "AggregationEnabled" : "false"
                }
            },
            {
                "PropertyGroupId": "ConsumerConfigProperties",
                "PropertyMap" : {
                    "aws.region" : "us-west-2"
                }
            }
        ]
      }
   }
}
```

2. 前述のリクエストでUpdateApplicationアクションを実行し、環境プロパティを更新します。

```
aws kinesisanalyticsv2 update-application --cli-input-json file://
update_properties_request.json
```

アプリケーションコードの更新

アプリケーションコードを新しいバージョンのコードパッケージで更新する必要がある場合は、UpdateApplication AWS CLI アクションを使用します。

> **Note**
>
> 同じファイル名のアプリケーションコードの新しいバージョンをロードするには、新しいオブジェクトバージョンを指定する必要があります。Amazon S3 オブジェクトバージョンを使用する方法の詳細については、「バージョニングの有効化または無効化」を参照してください。

を使用するには AWS CLI、Amazon S3 バケットから以前のコードパッケージを削除し、新しいバージョンをアップロードして、同じ Amazon S3 バケットとオブジェクト名、および新しいオブジェクトバージョンUpdateApplicationを指定して を呼び出します。アプリケーションは新しいコードパッケージで再起動します。

以下の UpdateApplication アクションのサンプル・リクエストは、アプリケーション・コードを再読み込 み、アプリケーションを再起動します。CurrentApplicationVersionId を現在のアプリケーションバージョンに更新します。ListApplications または DescribeApplication アクションを使用して、現在のアプリケーションバージョンを確認できます。バケット名のサフィックス (「*<username>*」) を、 the section called “2 つの Amazon Kinesis データストリームを作成する” セクションで選択したサフィックスで更新します。

```
{
    "ApplicationName": "test",
    "CurrentApplicationVersionId": 1,
    "ApplicationConfigurationUpdate": {
        "ApplicationCodeConfigurationUpdate": {
            "CodeContentUpdate": {
                "S3ContentLocationUpdate": {
                    "BucketARNUpdate": "arn:aws:s3:::ka-app-code-username",
                    "FileKeyUpdate": "aws-kinesis-analytics-java-apps-1.0.jar",
                    "ObjectVersionUpdate": "SAMPLEUehYngP87ex1nzYIGYgfhypvDU"
                }
            }
        }
    }
}
```

次のステップ

ステップ 4: リソースをクリーンアップ AWS する

ステップ 4: リソースをクリーンアップ AWS する

このセクションでは、入門チュートリアルで作成した AWS リソースをクリーンアップする手順について説明します。

このトピックには、次のセクションが含まれています。

- Managed Service for Apache Flink アプリケーションを削除する
- Kinesis データストリームを削除する
- Amazon S3 オブジェクトとバケットを削除する
- IAM リソースを削除する
- CloudWatch リソースを削除する
- 次のステップ

Managed Service for Apache Flink アプリケーションを削除する

1. Kinesis コンソール (https://console.aws.amazon.com/kinesis) を開きます。
2. Managed Service for Apache Flink パネルで、 を選択しますMyApplication。
3. アプリケーションのページで[削除]を選択し、削除を確認します。

Kinesis データストリームを削除する

1. https://console.aws.amazon.com/flink で Apache Flink 用 Managed Serviceコンソールを開く
2. Kinesis Data Streams パネルで、 を選択しますExampleInputStream。
3. ExampleInputStream ページで、Kinesis ストリームの削除を選択し、削除を確定します。
4. Kinesis ストリームページで、 を選択しExampleOutputStream、アクション を選択し、削除 を選択し、削除を確定します。

Amazon S3 オブジェクトとバケットを削除する

1. https://console.aws.amazon.com/s3/でAmazon S3 コンソールを開きます。
2. ka-app-code-***<username>*** バケットを選択します。
3. [削除] を選択し、バケット名を入力して削除を確認します。

IAM リソースを削除する

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. ナビゲーションバーで、[ポリシー] を選択します。
3. フィルターコントロールに「kinesis」と入力します。
4. kinesis-analytics-service-MyApplication-us-west-2 ポリシーを選択します。
5. [ポリシーアクション]、[削除] の順に選択します。
6. ナビゲーションバーで [ロール] を選択します。
7. kinesis-analytics-MyApplication-us-west-2 ロールを選択します。
8. [ロールの削除] を選択し、削除を確定します。

CloudWatch リソースを削除する

1. https://console.aws.amazon.com/cloudwatch/ で CloudWatch コンソールを開きます。

2. ナビゲーションバーで [ログ] を選択します。
3. /aws/kinesis-analytics/MyApplication ロググループを選択します。
4. [ロググループの削除]を選択し、削除を確認してください。

次のステップ

ステップ 5: 次のステップ

ステップ 5: 次のステップ

Apache Flink 用 Managed Serviceの基本的なアプリケーションを作成して実行したので、より高度な Apache Flink 用 Managed Serviceソリューションについては、以下のリソースを参照してください。

- Amazon Kinesis 用 AWS ストリーミングデータソリューション： Amazon Kinesis 用 AWS ストリーミングデータソリューションは、ストリーミングデータを簡単にキャプチャ、保存、処理、配信するために必要な AWS サービスを自動的に設定します。このソリューションには、ストリーミングデータのユースケースを解決するための複数のオプションが用意されています。Managed Service for Apache Flink オプションは、 end-to-endシミュレートされたニューヨークのタクシーデータに対して分析オペレーションを実行する実際のアプリケーションを示すストリーミング ETL の例を提供します。このソリューションは、IAM ロールとポリシー、 CloudWatch ダッシュボード、 CloudWatch アラームなど、必要なすべての AWS リソースを設定します。
- AWS Amazon MSK のストリーミングデータソリューション： Amazon MSK の AWS ストリーミングデータソリューションは、データがプロデューサー、ストリーミングストレージ、コンシューマー、および送信先を通過する AWS CloudFormation テンプレートを提供します。
- 「Apache Flink と Apache Kafka によるクリックストリームラボ」:ストリーミングストレージには Apache Kafka 用の Amazon マネージドストリーミングを使用し、ストリーム処理には Apache Flink アプリケーション向けの Apache Flink 用 Managed Serviceを使用する、クリックストリームのユースケースを対象としたエンドツーエンドラボです。
- Amazon Managed Service for Apache Flink Workshop: このワークショップでは、 end-to-end ストリーミングデータをほぼリアルタイムで取り込み、分析、視覚化するためのストリーミングアーキテクチャを構築します。あなたは、ニューヨーク市のあるタクシー会社の業務改善に着手しました。ニューヨーク市のタクシー車両のテレメトリデータをほぼリアルタイムで分析して、車両運用を最適化します。
- 「Learn Flink: ハンズオントレーニング:」スケーラブルなストリーミング ETL、分析、イベント駆動型アプリケーションの作成を開始するための Apache Flink の公式入門トレーニングです。

> **Note**
>
> Apache Flink 用 Managed Serviceは、このトレーニングで使用されている Apache Flink バージョン (1.12) をサポートしていないことに注意してください。Flink Managed Service for Apache Flink で Flink 1.15.2 を使用できます。

開始方法: Flink 1.11.1 - 非推奨

> **Note**
>
> Apache Flink バージョン 1.6、1.8、および 1.11 は、Apache Flink コミュニティで 3 年以上サポートされていません。Amazon Managed Service for Apache Flink では、2024 年 11 月 5 日にこれらのバージョンを廃止する予定です。この日以降、これらの Flink バージョン用の新しいアプリケーションを作成することはできません。現時点では、既存のアプリケーションの実行を継続できます。Amazon Managed Service for Apache Flink のインプレースバージョンアップグレード機能を使用して、アプリケーションをステートリーにアップグレードできます。詳細については、「」を参照してくださいApache Flink のインプレースバージョンアップグレード。

このトピックには、Apache Flink 1.11.1 を使用する開始方法 (DataStream API)チュートリアルのバージョンが含まれています。

このセクションでは、Managed Service for Apache Flink と DataStream API の基本概念を紹介します。アプリケーションの作成とテストに使用できるオプションについて説明します。また、このガイドのチュートリアルを完了し、初めてアプリケーションを作成するのに必要なツールのインストール方法についても説明します。

トピック

- Managed Service for Apache Flink アプリケーションのコンポーネント
- 演習を完了するための前提条件
- ステップ 1: AWS アカウントを設定し、管理者ユーザーを作成する
- ステップ 2: AWS Command Line Interface (AWS CLI) をセットアップする
- ステップ 3: Managed Service for Apache Flink アプリケーションを作成して実行する
- ステップ 4: リソースをクリーンアップ AWS する

- ステップ 5: 次のステップ

Managed Service for Apache Flink アプリケーションのコンポーネント

Apache Flink アプリケーション用 Managed Service では、Apache Flink ランタイムを使用して入力を処理し、出力を生成する Java/Apache Maven または Scala アプリケーションを使用してデータを処理します。

Apache Flink アプリケーション用 Managed Serviceには、以下のコンポーネントがあります。

- 「ランタイムプロパティ:」「ランタイムプロパティ」を使用すると、アプリケーションコードを再コンパイルせずにアプリケーションを設定できます。
- 「ソース:」 アプリケーションは 「ソース」 を使用してデータを消費します。ソースコネクタは、Kinesis データストリーム、Amazon S3 バケットなどからデータを読み取ります。詳細については、「[Sources] (出典)」を参照してください。
- 「オペレータ:」アプリケーションは 1 つ以上の「オペレータ」を使用してデータを処理します。オペレータはデータを変換、強化、または集約できます。詳細については、 DataStream API 演算子 を参照してください。
- 「シンク:」アプリケーションは「シンク」を使用して外部ソースにデータを生成します。シンクコネクタは、Kinesis データストリーム、Firehose ストリーム、Amazon S3 バケットなどにデータを書き込みます。詳細については、「シンク」を参照してください。

アプリケーションコードを作成、コンパイル、パッケージ化したら、コードパッケージを Amazon Simple Storage Service (Amazon S3) バケットにアップロードします。次に、Apache Flink アプリケーション用 Managed Serviceを作成します。コードパッケージの場所、ストリーミングデータソースとして Kinesis データストリームを渡し、通常はアプリケーションの処理済みデータを受け取るストリーミングまたはファイルの場所を渡します。

演習を完了するための前提条件

このガイドの手順を完了するには、以下が必要です。

- Java 開発キット (JDK) バージョン 11。`JAVA_HOME` 環境変数を、JDK のインストール場所を指すように設定します。
- 開発環境 (Eclipse Java Neon や IntelliJ Idea など) を使用してアプリケーションを開発し、コンパイルすることをお勧めします。

- Git クライアント。Git クライアントをまだインストールしていない場合は、インストールします。
- Apache Maven Compiler Plugin。Maven が作業パスに含まれている必要があります。Apache Maven のインストールをテストするには、次のように入力します。

```
$ mvn -version
```

開始するには、ステップ 1: AWS アカウントを設定し、管理者ユーザーを作成するに進みます。

ステップ 1: AWS アカウントを設定し、管理者ユーザーを作成する

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、次のステップ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環として、通話呼び出しを受け取り、電話キーパッドで検証コードを入力するように求められます。

   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。セキュリティのベストプラクティスとして、ユーザーに管理アクセスを割り当て、ルートユーザーのみを使用してルートユーザーアクセスが必要なタスクを実行してください。

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

のセキュリティ保護 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 「ルートユーザーの仮想 MFA デバイスを有効にする (コンソール) 」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、「 ユーザーガイド」の「デフォルト でユーザーアクセス IAM アイデンティティセンターディレクトリを設定するAWS IAM Identity Center 」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「グループの参加」を参照してください。

プログラム的なアクセス権を付与する

ユーザーが の AWS 外部で を操作する場合は、プログラムによるアクセスが必要です AWS Management Console。プログラムによるアクセスを許可する方法は、 にアクセスするユーザーのタイプによって異なります AWS。

ユーザーにプログラマチックアクセス権を付与するには、以下のいずれかのオプションを選択します。

プログラマチックアクセス権を必要とするユーザー	目的	方法
ワークフォースアイデンティティ (IAM Identity Center で管理されているユーザー)	一時的な認証情報を使用して、、 AWS SDKs AWS CLI、または AWS APIs。	使用するインターフェイス用の手引きに従ってください。 • については AWS CLI、「 ユーザーガイド」の AWS CLI 「 を使用するための の設定 AWS IAM Identity CenterAWS Command Line Interface 」を参照してください。 • AWS SDKs 、ツール、AWS APIs「 SDK とツールのリファレンスガイド」の「IAM Identity Center 認証」を参照してください。AWS SDKs

プログラマチックアクセス権を必要とするユーザー	目的	方法
IAM	一時的な認証情報を使用して、、 AWS SDKs AWS CLI、または AWS APIs。	「IAM ユーザーガイド」の「 AWS リソースでの一時的な認証情報の使用」の手順に従います。
IAM	(非推奨) 長期認証情報を使用して、、 AWS SDKs AWS CLI、または AWS APIs。	使用するインターフェイス用の手引きに従ってください。 • については AWS CLI、「 AWS Command Line Interface ユーザーガイド」の「IAM ユーザー認証情報を使用した認証」を参照してください。 • AWS SDKs「 SDK とツールのリファレンスガイド」の「長期的な認証情報を使用した認証」を参照してください。 AWS SDKs • AWS APIsユーザーガイド」の「IAM ユーザーのアクセスキーの管理」を参照してください。

次のステップ

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

ステップ 2: AWS Command Line Interface (AWS CLI) をセットアップする

このステップでは、Managed Service for Apache Flink で使用する をダウンロードして設定 AWS CLI します。

> **Note**
>
> このガイドの使用開始実習では、操作を実行するために、アカウントの管理者の認証情報 (adminuser) を使用していることが前提となっています。

> **Note**
>
> が既に AWS CLI インストールされている場合は、アップグレードして最新の機能を取得する必要がある場合があります。詳細については、「AWS Command Line Interface ユーザーガイド」の「AWS Command Line Interfaceのインストール」を参照してください。のバージョンを確認するには AWS CLI、次のコマンドを実行します。
>
> ```
> aws --version
> ```
>
> このチュートリアルの演習では、次の AWS CLI バージョン 以降が必要です。
>
> ```
> aws-cli/1.16.63
> ```

を設定するには 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。

   - AWS Command Line Interfaceのインストール
   - AWS CLIの設定

2. 管理者ユーザーの名前付きプロファイルを config ファイルに追加します AWS CLI 。 AWS CLI コマンドを実行するときに、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 AWS リージョンのリストについては、「」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。

> **Note**
>
> このチュートリアルのサンプルコードとコマンドでは、米国西部 (オレゴン) リージョンを使用しています。別の リージョンを使用するには、このチュートリアルのコードとコマンドのリージョンを、使用したいリージョンに変更します。

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
```

AWS アカウントと をセットアップしたら AWS CLI、次の演習を試してサンプルアプリケーションを設定し、 end-to-end セットアップをテストできます。

次のステップ

ステップ 3: Managed Service for Apache Flink アプリケーションを作成して実行する

ステップ 3: Managed Service for Apache Flink アプリケーションを作成して実行する

この演習では、データストリームをソースおよびシンクとして使用して、Managed Service for Apache Flink アプリケーションを作成します。

このセクションには、以下のステップが含まれています。

- 2 つの Amazon Kinesis データストリームを作成する
- サンプルレコードを入力ストリームに書き込む
- Apache Flink ストリーミング Java コードをダウンロードして調べる
- アプリケーションコードをコンパイルする
- Apache Flink ストリーミング Java コードをアップロードする
- Managed Service for Apache Flink アプリケーションを作成して実行する
- 次のステップ

2 つの Amazon Kinesis データストリームを作成する

この演習で Apache Flink アプリケーションのマネージドサービスを作成する前に、2 つの Kinesis データストリーム (`ExampleInputStream` と `ExampleOutputStream`) を作成する必要があります。アプリケーションでは、これらのストリームを使用してアプリケーションの送信元と送信先のストリームを選択します。

これらのストリームは Amazon Kinesis コンソールまたは次の AWS CLI コマンドを使用して作成できます。コンソールでの操作方法については、「Amazon Kinesis Data Streams デベロッパーガイド」 の 「データストリームの作成および更新」 を参照してください。

データストリームを作成するには (AWS CLI)

1. 最初のストリーム (`ExampleInputStream`) を作成するには、次の Amazon Kinesis `create-stream` AWS CLI コマンドを使用します。

```
$ aws kinesis create-stream \
--stream-name ExampleInputStream \
--shard-count 1 \
--region us-west-2 \
--profile adminuser
```

2. 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 ExampleOutputStream \
--shard-count 1 \
--region us-west-2 \
--profile adminuser
```

サンプルレコードを入力ストリームに書き込む

このセクションでは、Python スクリプトを使用して、アプリケーションが処理するサンプルレコードをストリームに書き込みます。

> **Note**
>
> このセクションでは AWS SDK for Python (Boto) が必要です。

1. 次の内容で、stock.py という名前のファイルを作成します。

```
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"))
```

2. このチュートリアルの後半では、アプリケーションにデータを送信する stock.py スクリプトを実行します。

```
$ python stock.py
```

Apache Flink ストリーミング Java コードをダウンロードして調べる

この例の Java アプリケーションコードは、 から入手できます GitHub。アプリケーションコードをダウンロードするには、次の操作を行います。

1. 次のコマンドを使用してリモートリポジトリのクローンを作成します。

```
git clone https://github.com/aws-samples/amazon-kinesis-data-analytics-examples.git
```

2. amazon-kinesis-data-analytics-java-examples/GettingStarted ディレクトリに移動します。

アプリケーションコードに関して、以下の点に注意してください。

- 「Project Object Model (pom.xml)」ファイルには、Managed Service for Apache Flink 用ライブラリなど、アプリケーションの設定と依存関係に関する情報が含まれています。
- BasicStreamingJob.java ファイルには、アプリケーションの機能を定義する main メソッドが含まれています。
- アプリケーションは Kinesis ソースを使用して、ソースストリームから読み取りを行います。次のスニペットでは、Kinesis ソースが作成されます。

```
return env.addSource(new FlinkKinesisConsumer<>(inputStreamName,
                new SimpleStringSchema(), inputProperties));
```

- アプリケーションでは、ソースおよびシンクコネクタを作成し、StreamExecutionEnvironment オブジェクトを使用して外部リソースにアクセスします。
- アプリケーションでは、静的プロパティを使用してソースおよびシンクコネクタを作成します。動的なアプリケーションプロパティを使用するには、createSourceFromApplicationProperties および createSinkFromApplicationProperties メソッドを使用してコネクタを作成します。これらのメソッドは、アプリケーションのプロパティを読み取ってコネクタを設定します。

  ランタイムプロパティの詳細については、ランタイムプロパティ を参照してください。

アプリケーションコードをコンパイルする

このセクションでは、Apache Maven コンパイラを使用してアプリケーション用の Java コードを作成します。Apache Maven と Java 開発キット (JDK) をインストールする方法については、演習を完了するための前提条件を満たすを参照してください。

アプリケーションコードをコンパイルするには

1. アプリケーションコードを使用するには、コードをコンパイルして JAR ファイルにパッケージ化します。コードのコンパイルとパッケージ化には次の 2 通りの方法があります。

   - コマンドライン Maven ツールを使用します。pom.xml ファイルが格納されているディレクトリで次のコマンドを実行して JAR ファイルを作成します。

     ```
     mvn package -Dflink.version=1.11.3
     ```

   - 開発環境を使用します。詳細については、開発環境のドキュメントを参照してください。

   > **Note**
   >
   > 提供されているソースコードは Java 11 のライブラリーに依存しています。プロジェクトの Java バージョンが 11 であることを確認してください。

   パッケージは JAR ファイルとしてアップロードすることも、圧縮して ZIP ファイルとしてアップロードすることもできします。を使用してアプリケーションを作成する場合は AWS CLI、コードコンテンツタイプ (JAR または ZIP) を指定します。
2. コンパイル中にエラーが発生した場合は、JAVA_HOME 環境変数が正しく設定されていることを確認します。

アプリケーションのコンパイルに成功すると、次のファイルが作成されます。

```
target/aws-kinesis-analytics-java-apps-1.0.jar
```

Apache Flink ストリーミング Java コードをアップロードする

このセクションでは、Amazon Simple Storage Service (Amazon S3) バケットを作成し、アプリケーションコードをアップロードします。

アプリケーションコードをアップロードするには

1. Amazon S3 コンソール (https://console.aws.amazon.com/s3/) を開きます。
2. [バケットを作成] を選択します。

3. [Bucket name (バケット名)] フィールドに**ka-app-code-*<username>***と入力します。バケット名にユーザー名などのサフィックスを追加して、グローバルに一意にします。[次へ] をクリックします。
4. 設定オプションのステップでは、設定をそのままにし、[次へ] を選択します。
5. アクセス許可の設定のステップでは、設定をそのままにし、[次へ] を選択します。
6. [バケットを作成] を選択します。
7. Amazon S3 コンソールで、ka-app-code-***<username>*** バケットを選択し、アップロード を選択します。
8. ファイルの選択のステップで、[ファイルを追加] を選択します。前のステップで作成した `aws-kinesis-analytics-java-apps-1.0.jar` ファイルに移動します。[次へ] をクリックします。
9. オブジェクトの設定を変更する必要はないので、[アップロード] を選択してください。

アプリケーションコードが Amazon S3 バケットに保存され、アプリケーションからアクセスできるようになります。

Managed Service for Apache Flink アプリケーションを作成して実行する

コンソールまたは AWS CLIのいずれかを使用してManaged Service for Apache Flink を作成し、実行することができます。

> **Note**
>
> コンソールを使用してアプリケーションを作成すると、 AWS Identity and Access Management (IAM) と Amazon CloudWatch Logs リソースが自動的に作成されます。を使用してアプリケーションを作成するときは AWS CLI、これらのリソースを個別に作成します。

トピック

- アプリケーションの作成と実行 (コンソール)
- アプリケーションの作成と実行 (AWS CLI)

アプリケーションの作成と実行 (コンソール)

以下の手順を実行し、コンソールを使用してアプリケーションを作成、設定、更新、および実行します。

アプリケーションの作成

1. https://console.aws.amazon.com/flink で Apache Flink 用 Managed Serviceコンソールを開く
2. Managed Service for Apache Flinkのダッシュボードで、「分析アプリケーションの作成」 を選択します。
3. 「Managed Service for Apache Flink-アプリケーションの作成」ページで、次のようにアプリケーションの詳細を入力します。
   - [アプリケーション名] には **MyApplication** と入力します。
   - [Description (説明)] に **My java test app** と入力します。
   - [ランタイム] には、[Apache Flink] を選択します。
   - バージョンプルダウンは「Apache Flink バージョン 1.11 (推奨バージョン)」のままにしておきます。
4. [アクセス許可] には、[IAM ロールの作成 / 更新**kinesis-analytics-MyApplication-us-west-2**] を選択します。
5. [Create application] を選択します。

> **Note**
>
> コンソールを使用して Apache Flink アプリケーション用 Managed Service を作成する場合は、IAM ロールとポリシーをアプリケーションが自動的に作成するオプションを選択できます。アプリケーションではこのロールとポリシーを使用して、依存リソースにアクセスします。これらの IAM リソースは、次のようにアプリケーション名とリージョンを使用して命名されます。
>
> - ポリシー: `kinesis-analytics-service-MyApplication-us-west-2`
> - ロール: `kinesisanalytics-MyApplication-us-west-2`

IAM ポリシーを編集する

IAM ポリシーを編集し、Kinesis Data Streamsにアクセスするための許可を追加します。

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. [ポリシー] を選択します。前のセクションでコンソールによって作成された **kinesis-analytics-service-MyApplication-us-west-2** ポリシーを選択します。

3. [概要] ページで、[ポリシーの編集] を選択します。[JSON] タブを選択します。
4. 次のポリシー例で強調表示されているセクションをポリシーに追加します。サンプルのアカウント ID (*012345678901*) を自分のアカウント ID に置き換えます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ReadCode",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": [
                "arn:aws:s3:::ka-app-code-username/aws-kinesis-analytics-java-
apps-1.0.jar"
            ]
        },
        {
            "Sid": "DescribeLogGroups",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:*"
            ]
        },
        {
            "Sid": "DescribeLogStreams",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogStreams"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:*"
            ]
        },
        {
            "Sid": "PutLogEvents",
            "Effect": "Allow",
```

```
            "Action": [
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:kinesis-analytics-log-stream"
            ]
        },
        {
            "Sid": "ReadInputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
        },
        {
            "Sid": "WriteOutputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleOutputStream"
        }
    ]
}
```

アプリケーションを設定する

1. MyApplication ページで、 の設定 を選択します。
2. [Configure application] ページで、[Code location] を次のように指定します。
   - [Amazon S3 バケット] で、**ka-app-code-*<username>***と入力します。
   - [Amazon S3 オブジェクトへのパス] で、**aws-kinesis-analytics-java-apps-1.0.jar**と入力します。
3. [Access to application resources] の [Access permissions] では、[Create / update IAM role**kinesis-analytics-MyApplication-us-west-2**] を選択します。
4. [Properties] の [Group ID] には、**ProducerConfigProperties**と入力します。
5. 次のアプリケーションのプロパティと値を入力します。

グループ ID	キー	値
ProducerConfigProperties	**flink.inputstream.initpos**	**LATEST**
ProducerConfigProperties	**aws.region**	**us-west-2**
ProducerConfigProperties	**AggregationEnabled**	**false**

6. [Monitoring] の [Monitoring metrics level] が [Application] に設定されていることを確認します。
7. CloudWatch のログ記録では、有効化チェックボックスをオンにします。
8. [更新] を選択します。

> **Note**
>
> Amazon CloudWatch ログ記録を有効にすると、Managed Service for Apache Flink によってロググループとログストリームが作成されます。これらのリソースの名前は次のとおりです。
>
> - ロググループ: `/aws/kinesis-analytics/MyApplication`
> - ログストリーム: `kinesis-analytics-log-stream`

アプリケーションを実行する

Flink ジョブグラフは、アプリケーションを実行し、Apache Flink ダッシュボードを開き、目的の Flink ジョブを選択すると表示できます。

アプリケーションの停止

MyApplication ページで、停止 を選択します。アクションを確認します。

アプリケーションの更新

コンソールを使用して、アプリケーションのプロパティ、モニタリング設定、アプリケーション JAR の場所またはファイル名などのアプリケーション設定を更新できます。アプリケーションコー

ドを更新する必要がある場合は、アプリケーション JAR を Amazon S3 バケットから再ロードすることもできます。

MyApplication ページで、 の設定 を選択します。アプリケーションの設定を更新し、[更新] を選択します。

アプリケーションの作成と実行 (AWS CLI)

このセクションでは、 を使用して Managed Service for Apache Flink アプリケーション AWS CLI を作成して実行します。Managed Service for Apache Flink は `kinesisanalyticsv2` AWS CLI コマンドを使用して Managed Service for Apache Flink アプリケーションを作成して操作します。

アクセス許可ポリシーを作成する

> **Note**
>
> アプリケーションのアクセス権限ポリシーとロールを作成する必要があります。これらの IAM リソースを作成しない場合、アプリケーションはそのデータストリームやログストリームにアクセスできません。

まず、2 つのステートメントを含むアクセス許可ポリシーを作成します。1 つは、ソースストリームの `read` アクションに対するアクセス許可を付与し、もう 1 つはシンクストリームの `write` アクションに対するアクセス許可を付与します。次に、IAM ロール (次のセクションで作成) にポリシーをアタッチします。そのため、 Managed Service for Apache Flinkがこのロールを引き受けると、ソースストリームからの読み取りとシンクストリームへの書き込みを行うために必要なアクセス許可がサービスに付与されます。

次のコードを使用して `AKReadSourceStreamWriteSinkStream` アクセス許可ポリシーを作成します。*`username`* を Amazon S3 バケットの作成に使用したユーザー名に置き換え、アプリケーションコードを保存します。Amazon リソースネーム (ARN) のアカウント ID (*`012345678901`*) を自分のアカウント ID に置き換えます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "S3",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
```

```
                "s3:GetObjectVersion"
            ],
            "Resource": ["arn:aws:s3:::ka-app-code-username",
                "arn:aws:s3:::ka-app-code-username/*"
            ]
        },
        {
            "Sid": "ReadInputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
        },
        {
            "Sid": "WriteOutputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleOutputStream"
        }
    ]
}
```

アクセス許可ポリシーを作成する step-by-step 手順については、IAM ユーザーガイドの「チュートリアル: 最初のカスタマー管理ポリシーの作成とアタッチ」を参照してください。

> **Note**
>
> その他のアマゾンサービスにアクセスするには、 AWS SDK for Javaを使用します。Managed Service for Apache Flink は、アプリケーションに関連付けられているサービス実行 IAM ロールに、SDK が必要とする認証情報を自動的に設定します。追加の手順は必要ありません。

IAM ロールを作成します。

このセクションでは、Managed Service for Apache Flink アプリケーションがソースストリームを読み取り、シンクストリームに書き込むために想定できる IAM ロールを作成します。

Apache Flink 用 Managed Service は、許可なしにはストリームにアクセスできません。IAM ロールを介してこれらの許可を付与します。各 IAM ロールには、2 つのポリシーがアタッチされます。信頼ポリシーは、ロールを引き受けるための許可を Managed Service for Apache Flink 付与し、許可ポ

リシーは、ロールを引き受けた後に Managed Service for Apache Flink が実行できる事柄を決定します。

前のセクションで作成したアクセス許可ポリシーをこのロールにアタッチします。

IAM ロールを作成するには

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. ナビゲーションペインで [Roles (ロール)]、[Create Role (ロールの作成)] の順に選択します。
3. [信頼されるエンティティの種類を選択] で、[AWS のサービス] を選択します。[このロールを使用するサービスを選択] で、[Kinesis Analytics] を選択します。[ユースケースの選択] で、[Kinesis Analytics] を選択します。

   [Next: Permissions] (次のステップ: 許可) を選択します。
4. [アクセス権限ポリシーをアタッチする] ページで、[Next: Review] (次: 確認) を選択します。ロールを作成した後に、アクセス許可ポリシーをアタッチします。
5. [Create role (ロールの作成)] ページで、ロールの名前に **`MF-stream-rw-role`** を入力します。[ロールの作成] を選択します。

   これで、`MF-stream-rw-role` と呼ばれる新しい IAM ロールが作成されます。次に、ロールの信頼ポリシーとアクセス許可ポリシーを更新します。
6. アクセス許可ポリシーをロールにアタッチします。

   > **Note**
   >
   > この演習では、Managed Service for Apache Flink が、Kinesis データストリーム (ソース) からのデータの読み取りと、別の Kinesis データストリームへの出力の書き込みの両方を実行するためにこのロールを引き受けます。このため、前のステップで作成したポリシー、the section called “アクセス許可ポリシーを作成する” をアタッチします。

   a. [概要] ページで、[アクセス許可] タブを選択します。
   b. [Attach Policies (ポリシーのアタッチ)] を選択します。
   c. 検索ボックスに**`AKReadSourceStreamWriteSinkStream`**(前のセクションで作成したポリシー) と入力します。
   d. AKReadSourceStreamWriteSinkStream ポリシーを選択し、ポリシー をアタッチ を選択します。

これで、アプリケーションがリソースにアクセスするために使用するサービスの実行ロールが作成されました。新しいロールの ARN を書き留めておきます。

ロールを作成する step-by-step 手順については、IAM ユーザーガイドの「IAM ロールの作成 (コンソール)」を参照してください。

Managed Service for Apache Flink アプリケーションを作成する

1. 次の JSON コードを create_request.json という名前のファイルに保存します。サンプルロールの ARN を、前に作成したロールの ARN に置き換えます。バケット ARN のサフィックス (*username*) を、前のセクションで選択したサフィックスに置き換えます。サービス実行ロールのサンプルのアカウント ID (*012345678901*) を、自分のアカウント ID に置き換えます。

```
{
    "ApplicationName": "test",
    "ApplicationDescription": "my java test app",
    "RuntimeEnvironment": "FLINK-1_11",
    "ServiceExecutionRole": "arn:aws:iam::012345678901:role/MF-stream-rw-role",
    "ApplicationConfiguration": {
        "ApplicationCodeConfiguration": {
            "CodeContent": {
                "S3ContentLocation": {
                    "BucketARN": "arn:aws:s3:::ka-app-code-username",
                    "FileKey": "aws-kinesis-analytics-java-apps-1.0.jar"
                }
            },
            "CodeContentType": "ZIPFILE"
        },
        "EnvironmentProperties":  {
         "PropertyGroups": [
            {
               "PropertyGroupId": "ProducerConfigProperties",
               "PropertyMap" : {
                    "flink.stream.initpos" : "LATEST",
                    "aws.region" : "us-west-2",
                    "AggregationEnabled" : "false"
               }
            },
            {
               "PropertyGroupId": "ConsumerConfigProperties",
               "PropertyMap" : {
                    "aws.region" : "us-west-2"
               }
```

```
            }
         ]
      }
   }
}
```

2. 前述のリクエストを指定して CreateApplication アクションを実行し、アプリケーションを作成します。

```
aws kinesisanalyticsv2 create-application --cli-input-json file://
create_request.json
```

これでアプリケーションが作成されました。次のステップでは、アプリケーションを起動します。

アプリケーションを起動する

このセクションでは、StartApplication アクションを使用してアプリケーションを起動します。

アプリケーションを起動するには

1. 次の JSON コードを start_request.json という名前のファイルに保存します。

```
{
    "ApplicationName": "test",
    "RunConfiguration": {
        "ApplicationRestoreConfiguration": {
         "ApplicationRestoreType": "RESTORE_FROM_LATEST_SNAPSHOT"
         }
    }
}
```

2. 前述のリクエストを指定して StartApplication アクションを実行し、アプリケーションを起動します。

```
aws kinesisanalyticsv2 start-application --cli-input-json file://start_request.json
```

アプリケーションが実行されます。Amazon CloudWatch コンソールで Managed Service for Apache Flink メトリクスをチェックして、アプリケーションが動作していることを確認できます。

アプリケーションの停止

このセクションでは、StopApplication アクションを使用してアプリケーションを停止します。

アプリケーションを停止するには

1. 次の JSON コードを stop_request.json という名前のファイルに保存します。

```
{
    "ApplicationName": "test"
}
```

2. 次のリクエストを指定して StopApplication アクションを実行し、アプリケーションを停止します。

```
aws kinesisanalyticsv2 stop-application --cli-input-json file://stop_request.json
```

アプリケーションが停止します。

ログ記録オプションを追加する CloudWatch

を使用して AWS CLI 、Amazon CloudWatch ログストリームをアプリケーションに追加できます。アプリケーションで CloudWatch ログを使用する方法については、「」を参照してくださいthe section called “ログ記録の設定”。

環境プロパティを更新する

このセクションでは、「UpdateApplication」アクションを使用して、アプリケーションコードを再コンパイルせずにアプリケーションの環境プロパティを変更します。この例では、ソースストリームおよびレプリケート先ストリームのリージョンを変更します。

アプリケーションの環境プロパティを更新します

1. 次の JSON コードを update_properties_request.json という名前のファイルに保存します。

```
{"ApplicationName": "test",
   "CurrentApplicationVersionId": 1,
   "ApplicationConfigurationUpdate": {
      "EnvironmentPropertyUpdates": {
```

```
         "PropertyGroups": [
            {
               "PropertyGroupId": "ProducerConfigProperties",
               "PropertyMap" : {
                    "flink.stream.initpos" : "LATEST",
                    "aws.region" : "us-west-2",
                    "AggregationEnabled" : "false"
               }
            },
            {
               "PropertyGroupId": "ConsumerConfigProperties",
               "PropertyMap" : {
                    "aws.region" : "us-west-2"
               }
            }
         ]
      }
   }
}
```

2. 前述のリクエストでUpdateApplicationアクションを実行し、環境プロパティを更新します。

```
aws kinesisanalyticsv2 update-application --cli-input-json file://
update_properties_request.json
```

アプリケーションコードの更新

アプリケーションコードを新しいバージョンのコードパッケージで更新する必要がある場合は、UpdateApplication AWS CLI アクションを使用します。

> **Note**
>
> 同じファイル名のアプリケーションコードの新しいバージョンをロードするには、新しいオブジェクトバージョンを指定する必要があります。Amazon S3 オブジェクトバージョンを使用する方法の詳細については、「バージョニングの有効化または無効化」を参照してください。

を使用するには AWS CLI、Amazon S3 バケットから以前のコードパッケージを削除し、新しいバージョンをアップロードして、同じ Amazon S3 バケットとオブジェクト名、および新しいオブジェク

トバージョンUpdateApplicationを指定して を呼び出します。アプリケーションは新しいコードパッケージで再起動します。

以下の UpdateApplication アクションのサンプル・リクエストは、アプリケーション・コードを再読み込 み、アプリケーションを再起動します。CurrentApplicationVersionId を現在のアプリケーションバージョンに更新します。ListApplications または DescribeApplication アクションを使用して、現在のアプリケーションバージョンを確認できます。バケット名のサフィックス (「*<username>*」) を、 the section called “2 つの Amazon Kinesis データストリームを作成する” セクションで選択したサフィックスで更新します。

```
{
    "ApplicationName": "test",
    "CurrentApplicationVersionId": 1,
    "ApplicationConfigurationUpdate": {
        "ApplicationCodeConfigurationUpdate": {
            "CodeContentUpdate": {
                "S3ContentLocationUpdate": {
                    "BucketARNUpdate": "arn:aws:s3:::ka-app-code-username",
                    "FileKeyUpdate": "aws-kinesis-analytics-java-apps-1.0.jar",
                    "ObjectVersionUpdate": "SAMPLEUehYngP87ex1nzYIGYgfhypvDU"
                }
            }
        }
    }
}
```

次のステップ

ステップ 4: リソースをクリーンアップ AWS する

ステップ 4: リソースをクリーンアップ AWS する

このセクションでは、入門チュートリアルで作成した AWS リソースをクリーンアップする手順について説明します。

このトピックには、次のセクションが含まれています。

- Managed Service for Apache Flink アプリケーションを削除する
- Kinesis データストリームを削除する
- Amazon S3 オブジェクトとバケットを削除する
- rour IAM リソースを削除する

- CloudWatch リソースを削除する
- 次のステップ

Managed Service for Apache Flink アプリケーションを削除する

1. Kinesis コンソール (https://console.aws.amazon.com/kinesis) を開きます。
2. Managed Service for Apache Flink パネルで、 を選択しますMyApplication。
3. アプリケーションのページで[削除]を選択し、削除を確認します。

Kinesis データストリームを削除する

1. https://console.aws.amazon.com/flink で Apache Flink 用 Managed Serviceコンソールを開く
2. Kinesis Data Streams パネルで、 を選択しますExampleInputStream。
3. ExampleInputStream ページで、Kinesis ストリームの削除を選択し、削除を確定します。
4. Kinesis ストリームページで、 を選択しExampleOutputStream、アクション を選択し、削除 を選択し、削除を確定します。

Amazon S3 オブジェクトとバケットを削除する

1. https://console.aws.amazon.com/s3/でAmazon S3 コンソールを開きます。
2. ka-app-code-***<username>*** バケットを選択します。
3. [削除] を選択し、バケット名を入力して削除を確認します。

rour IAM リソースを削除する

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. ナビゲーションバーで、[ポリシー] を選択します。
3. フィルターコントロールに「kinesis」と入力します。
4. kinesis-analytics-service-MyApplication-us-west-2 ポリシーを選択します。
5. [ポリシーアクション]、[削除] の順に選択します。
6. ナビゲーションバーで [ロール] を選択します。
7. kinesis-analytics-MyApplication-us-west-2 ロールを選択します。
8. [ロールの削除] を選択し、削除を確定します。

CloudWatch リソースを削除する

1. https://console.aws.amazon.com/cloudwatch/ で CloudWatch コンソールを開きます。
2. ナビゲーションバーで [ログ] を選択します。
3. /aws/kinesis-analytics/MyApplication ロググループを選択します。
4. [ロググループの削除]を選択し、削除を確認してください。

次のステップ

ステップ 5: 次のステップ

ステップ 5: 次のステップ

Apache Flink 用 Managed Serviceの基本的なアプリケーションを作成して実行したので、より高度な Apache Flink 用 Managed Serviceソリューションについては、以下のリソースを参照してください。

- Amazon Kinesis 用 AWS ストリーミングデータソリューション： Amazon Kinesis 用 AWS ストリーミングデータソリューションは、ストリーミングデータを簡単にキャプチャ、保存、処理、配信するために必要な AWS サービスを自動的に設定します。このソリューションには、ストリーミングデータのユースケースを解決するための複数のオプションが用意されています。Managed Service for Apache Flink オプションは、 end-to-endシミュレートされたニューヨークのタクシーデータに対して分析オペレーションを実行する実際のアプリケーションを示すストリーミング ETL の例を提供します。このソリューションは、IAM ロールとポリシー、 CloudWatch ダッシュボード、 CloudWatch アラームなど、必要なすべての AWS リソースを設定します。
- AWS Amazon MSK のストリーミングデータソリューション： Amazon MSK の AWS ストリーミングデータソリューションは、データがプロデューサー、ストリーミングストレージ、コンシューマー、および送信先を通過する AWS CloudFormation テンプレートを提供します。
- 「Apache Flink と Apache Kafka によるクリックストリームラボ」:ストリーミングストレージには Apache Kafka 用の Amazon マネージドストリーミングを使用し、ストリーム処理には Apache Flink アプリケーション向けの Apache Flink 用 Managed Serviceを使用する、クリックストリームのユースケースを対象としたエンドツーエンドラボです。
- Amazon Managed Service for Apache Flink Workshop: このワークショップでは、 end-to-end ストリーミングデータをほぼリアルタイムで取り込み、分析、視覚化するためのストリーミングアーキテクチャを構築します。あなたは、ニューヨーク市のあるタクシー会社の業務改善に着手しました。ニューヨーク市のタクシー車両のテレメトリデータをほぼリアルタイムで分析して、車両運用を最適化します。

- 「Learn Flink: ハンズオントレーニング:」スケーラブルなストリーミング ETL、分析、イベント駆動型アプリケーションの作成を開始するための Apache Flink の公式入門トレーニングです。

> **Note**
>
> Apache Flink 用 Managed Serviceは、このトレーニングで使用されている Apache Flink バージョン (1.12) をサポートしていないことに注意してください。Flink Managed Service for Apache Flink で Flink 1.15.2 を使用できます。

- Apache Flink コード例： さまざまな Apache Flink アプリケーション例の GitHub リポジトリ。

開始方法: Flink 1.8.2 - 非推奨

> **Note**
>
> Apache Flink バージョン 1.6、1.8、および 1.11 は、Apache Flink コミュニティで 3 年以上サポートされていません。Amazon Managed Service for Apache Flink では、2024 年 11 月 5 日にこれらのバージョンを廃止する予定です。この日以降、これらの Flink バージョン用の新しいアプリケーションを作成することはできません。現時点では、既存のアプリケーションの実行を継続できます。Amazon Managed Service for Apache Flink のインプレースバージョンアップグレード機能を使用して、アプリケーションをステートリーにアップグレードできます。詳細については、「」を参照してくださいApache Flink のインプレースバージョンアップグレード。

このトピックには、Apache Flink 1.8.2 開始方法 (DataStream API)を使用するチュートリアルのバージョンが含まれています。

トピック

- Managed Service for Apache Flink アプリケーションのコンポーネント
- 演習を完了するための前提条件
- ステップ 1: AWS アカウントを設定し、管理者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: Managed Service for Apache Flink アプリケーションを作成して実行する
- ステップ 4: リソースをクリーンアップ AWS する

Managed Service for Apache Flink アプリケーションのコンポーネント

Apache Flink アプリケーション用 Managed Service では、Apache Flink ランタイムを使用して入力を処理し、出力を生成する Java/Apache Maven または Scala アプリケーションを使用してデータを処理します。

Apache Flink アプリケーション用 Managed Serviceには、以下のコンポーネントがあります。

- 「ランタイムプロパティ:」「ランタイムプロパティ」を使用すると、アプリケーションコードを再コンパイルせずにアプリケーションを設定できます。
- 「ソース:」 アプリケーションは 「ソース」 を使用してデータを消費します。ソースコネクタは、Kinesis データストリーム、Amazon S3 バケットなどからデータを読み取ります。詳細については、「[Sources] (出典)」を参照してください。
- 「オペレータ:」アプリケーションは 1 つ以上の「オペレータ」を使用してデータを処理します。オペレータはデータを変換、強化、または集約できます。詳細については、 DataStream API 演算子 を参照してください。
- 「シンク:」アプリケーションは「シンク」を使用して外部ソースにデータを生成します。シンクコネクタは、Kinesis データストリーム、Firehose ストリーム、Amazon S3 バケットなどにデータを書き込みます。詳細については、「シンク」を参照してください。

アプリケーションコードを作成、コンパイル、パッケージ化したら、コードパッケージを Amazon Simple Storage Service (Amazon S3) バケットにアップロードします。次に、Apache Flink アプリケーション用 Managed Serviceを作成します。コードパッケージの場所、ストリーミングデータソースとして Kinesis データストリームを渡し、通常はアプリケーションの処理済みデータを受け取るストリーミングまたはファイルの場所を渡します。

演習を完了するための前提条件

このガイドの手順を完了するには、以下が必要です。

- Java 開発キット (JDK) バージョン 8。`JAVA_HOME` 環境変数を、JDK のインストール場所を指すように設定します。
- このチュートリアルで Apache Flink Kinesis コネクタを使用するには、Apache Flink をダウンロードしてインストールする必要があります。詳細については、 以前の Apache Flink バージョンでの Apache Flink Kinesis Streams コネクタの使用 を参照してください。
- 開発環境 (Eclipse Java Neon や IntelliJ Idea など) を使用してアプリケーションを開発し、コンパイルすることをお勧めします。

- Git クライアント。Git クライアントをまだインストールしていない場合は、インストールします。
- Apache Maven Compiler Plugin。Maven が作業パスに含まれている必要があります。Apache Maven のインストールをテストするには、次のように入力します。

```
$ mvn -version
```

開始するには、ステップ 1: AWS アカウントを設定し、管理者ユーザーを作成するに進みます。

ステップ 1: AWS アカウントを設定し、管理者ユーザーを作成する

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、次のステップ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環として、通話呼び出しを受け取り、電話キーパッドで検証コードを入力するように求められます。

   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。セキュリティのベストプラクティスとして、ユーザーに管理アクセスを割り当て、ルートユーザーのみを使用してルートユーザーアクセスが必要なタスクを実行してください。

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

のセキュリティ保護 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 「ルートユーザーの仮想 MFA デバイスを有効にする (コンソール)」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、「 ユーザーガイド」の「デフォルト でユーザーアクセス IAM アイデンティティセンターディレクトリを設定するAWS IAM Identity Center 」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「グループの参加」を参照してください。

プログラム的なアクセス権を付与する

ユーザーが の AWS 外部で を操作する場合は、プログラムによるアクセスが必要です AWS Management Console。プログラムによるアクセスを許可する方法は、 にアクセスするユーザーのタイプによって異なります AWS。

ユーザーにプログラマチックアクセス権を付与するには、以下のいずれかのオプションを選択します。

プログラマチックアクセス権を必要とするユーザー	目的	方法
ワークフォースアイデンティティ (IAM Identity Center で管理されているユーザー)	一時的な認証情報を使用して、、 AWS SDKs AWS CLI、または AWS APIs。	使用するインターフェイス用の手引きに従ってください。 • については AWS CLI、「 ユーザーガイド」の AWS CLI 「 を使用するための の設定 AWS IAM Identity CenterAWS Command Line Interface 」を参照してください。 • AWS SDKs 、ツール、AWS APIs「 SDK とツールのリファレンスガイド」の「IAM Identity Center 認証」を参照してください。AWS SDKs

プログラマチックアクセス権を必要とするユーザー	目的	方法
IAM	一時的な認証情報を使用して、、 AWS SDKs AWS CLI、または AWS APIs。	「IAM ユーザーガイド」の「 AWS リソースでの一時的な認証情報の使用」の手順に従います。
IAM	(非推奨) 長期認証情報を使用して、、 AWS SDKs AWS CLI、または AWS APIs。	使用するインターフェイス用の手引きに従ってください。 • については AWS CLI、「 AWS Command Line Interface ユーザーガイド」の「IAM ユーザー認証情報を使用した認証」を参照してください。 • AWS SDKs「 SDK とツールのリファレンスガイド」の「長期的な認証情報を使用した認証」を参照してください。 AWS SDKs • AWS APIsユーザーガイド」の「IAM ユーザーのアクセスキーの管理」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

このステップでは、Managed Service for Apache Flink で使用する をダウンロードして設定 AWS CLI します。

> **Note**
>
> このガイドの使用開始実習では、操作を実行するために、アカウントの管理者の認証情報 (`adminuser`) を使用していることが前提となっています。

> **Note**
>
> が既に AWS CLI インストールされている場合は、アップグレードして最新の機能を取得する必要がある場合があります。詳細については、「AWS Command Line Interface ユーザーガイド」の「AWS Command Line Interfaceのインストール」を参照してください。のバージョンを確認するには AWS CLI、次のコマンドを実行します。
>
> ```
> aws --version
> ```
>
> このチュートリアルの演習では、次の AWS CLI バージョン 以降が必要です。
>
> ```
> aws-cli/1.16.63
> ```

を設定するには 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。

   - AWS Command Line Interfaceのインストール
   - AWS CLIの設定

2. 管理者ユーザーの名前付きプロファイルを config ファイルに追加します AWS CLI 。 AWS CLI コマンドを実行するときに、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
   ```

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

> **Note**
>
> このチュートリアルのサンプルコードとコマンドでは、米国西部 (オレゴン) リージョンを使用しています。別の AWS リージョンを使用するには、このチュートリアルのコードとコマンドのリージョンを、使用するリージョンに変更します。

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
```

AWS アカウントと をセットアップしたら AWS CLI、次の演習を試してサンプルアプリケーションを設定し、 end-to-end セットアップをテストできます。

次のステップ

ステップ 3: Managed Service for Apache Flink アプリケーションを作成して実行する

ステップ 3: Managed Service for Apache Flink アプリケーションを作成して実行する

この演習では、データストリームをソースおよびシンクとして使用して、Managed Service for Apache Flink アプリケーションを作成します。

このセクションには、以下のステップが含まれています。

- 2 つの Amazon Kinesis データストリームを作成する
- サンプルレコードを入力ストリームに書き込む
- Apache Flink ストリーミング Java コードをダウンロードして調べる
- アプリケーションコードをコンパイルする
- Apache Flink ストリーミング Java コードをアップロードする
- Managed Service for Apache Flink アプリケーションを作成して実行する
- 次のステップ

2 つの Amazon Kinesis データストリームを作成する

この演習で Apache Flink アプリケーションのマネージドサービスを作成する前に、2 つの Kinesis データストリーム (`ExampleInputStream` と `ExampleOutputStream`) を作成する必要がありま

す。アプリケーションでは、これらのストリームを使用してアプリケーションの送信元と送信先のストリームを選択します。

これらのストリームは Amazon Kinesis コンソールまたは次の AWS CLI コマンドを使用して作成できます。コンソールでの操作方法については、「Amazon Kinesis Data Streams デベロッパーガイド」 の 「データストリームの作成および更新」 を参照してください。

データストリームを作成するには (AWS CLI)

1. 最初のストリーム (ExampleInputStream) を作成するには、次の Amazon Kinesis create-stream AWS CLI コマンドを使用します。

```
$ aws kinesis create-stream \
--stream-name ExampleInputStream \
--shard-count 1 \
--region us-west-2 \
--profile adminuser
```

2. 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 ExampleOutputStream \
--shard-count 1 \
--region us-west-2 \
--profile adminuser
```

サンプルレコードを入力ストリームに書き込む

このセクションでは、Python スクリプトを使用して、アプリケーションが処理するサンプルレコードをストリームに書き込みます。

> **Note**
>
> このセクションでは AWS SDK for Python (Boto) が必要です。

1. 次の内容で、stock.py という名前のファイルを作成します。

```
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"))
```

2. このチュートリアルの後半では、アプリケーションにデータを送信する stock.py スクリプトを実行します。

```
$ python stock.py
```

Apache Flink ストリーミング Java コードをダウンロードして調べる

この例の Java アプリケーションコードは、 から入手できます GitHub。アプリケーションコードをダウンロードするには、次の操作を行います。

1. 次のコマンドを使用してリモートリポジトリのクローンを作成します。

```
git clone https://github.com/aws-samples/amazon-kinesis-data-analytics-examples.git
```

2. amazon-kinesis-data-analytics-java-examples/GettingStarted_1_8 ディレクトリに移動します。

アプリケーションコードに関して、以下の点に注意してください。

- 「Project Object Model (pom.xml)」ファイルには、Managed Service for Apache Flink 用ライブラリなど、アプリケーションの設定と依存関係に関する情報が含まれています。
- BasicStreamingJob.java ファイルには、アプリケーションの機能を定義する main メソッドが含まれています。
- アプリケーションは Kinesis ソースを使用して、ソースストリームから読み取りを行います。次のスニペットでは、Kinesis ソースが作成されます。

```
return env.addSource(new FlinkKinesisConsumer<>(inputStreamName,
                new SimpleStringSchema(), inputProperties));
```

- アプリケーションでは、ソースおよびシンクコネクタを作成し、StreamExecutionEnvironment オブジェクトを使用して外部リソースにアクセスします。
- アプリケーションでは、静的プロパティを使用してソースおよびシンクコネクタを作成します。動的なアプリケーションプロパティを使用するには、createSourceFromApplicationProperties および createSinkFromApplicationProperties メソッドを使用してコネクタを作成します。これらのメソッドは、アプリケーションのプロパティを読み取ってコネクタを設定します。

  ランタイムプロパティの詳細については、ランタイムプロパティ を参照してください。

アプリケーションコードをコンパイルする

このセクションでは、Apache Maven コンパイラを使用してアプリケーション用の Java コードを作成します。Apache Maven と Java 開発キット (JDK) をインストールする方法については、演習を完了するための前提条件を参照してください。

> **Note**
>
> 「1.11 より前のバージョンの Apache Flink で Kinesis コネクタを使用するには、Apache Maven をダウンロード、ビルド、インストールする必要があります。」 詳細について

は、the section called “以前の Apache Flink バージョンでの Apache Flink Kinesis Streams コネクタの使用” を参照してください。

アプリケーションコードをコンパイルするには

1. アプリケーションコードを使用するには、コードをコンパイルして JAR ファイルにパッケージ化します。コードのコンパイルとパッケージ化には次の 2 通りの方法があります。

   - コマンドライン Maven ツールを使用します。pom.xml ファイルが格納されているディレクトリで次のコマンドを実行して JAR ファイルを作成します。

     ```
     mvn package -Dflink.version=1.8.2
     ```

   - 開発環境を使用します。詳細については、開発環境のドキュメントを参照してください。

     > **Note**
     >
     > 提供されているソースコードは Java 1.8 のライブラリに依存しています。プロジェクトの Java バージョンが 1.8 であることを確認してください。

   パッケージは JAR ファイルとしてアップロードすることも、圧縮して ZIP ファイルとしてアップロードすることもできします。を使用してアプリケーションを作成する場合は AWS CLI、コードコンテンツタイプ (JAR または ZIP) を指定します。
2. コンパイル中にエラーが発生した場合は、JAVA_HOME 環境変数が正しく設定されていることを確認します。

アプリケーションのコンパイルに成功すると、次のファイルが作成されます。

```
target/aws-kinesis-analytics-java-apps-1.0.jar
```

Apache Flink ストリーミング Java コードをアップロードする

このセクションでは、Amazon Simple Storage Service (Amazon S3) バケットを作成し、アプリケーションコードをアップロードします。

アプリケーションコードをアップロードするには

1. Amazon S3 コンソール (https://console.aws.amazon.com/s3/) を開きます。

2. [バケットを作成] を選択します。
3. [Bucket name (バケット名)] フィールドに**ka-app-code-*<username>***と入力します。バケット名にユーザー名などのサフィックスを追加して、グローバルに一意にします。[次へ] をクリックします。
4. 設定オプションのステップでは、設定をそのままにし、[次へ] を選択します。
5. アクセス許可の設定のステップでは、設定をそのままにし、[次へ] を選択します。
6. [バケットを作成] を選択します。
7. Amazon S3 コンソールで、ka-app-code-***<username>*** バケットを選択し、アップロードを選択します。
8. ファイルの選択のステップで、[ファイルを追加] を選択します。前のステップで作成した `aws-kinesis-analytics-java-apps-1.0.jar` ファイルに移動します。[次へ] をクリックします。
9. オブジェクトの設定を変更する必要はないので、[アップロード] を選択してください。

アプリケーションコードが Amazon S3 バケットに保存され、アプリケーションからアクセスできるようになります。

Managed Service for Apache Flink アプリケーションを作成して実行する

コンソールまたは AWS CLIのいずれかを使用してManaged Service for Apache Flink を作成し、実行することができます。

> **Note**
>
> コンソールを使用してアプリケーションを作成すると、 AWS Identity and Access Management (IAM) と Amazon CloudWatch Logs リソースが自動的に作成されます。を使用してアプリケーションを作成するときは AWS CLI、これらのリソースを個別に作成します。

トピック

- アプリケーションの作成と実行 (コンソール)
- アプリケーションの作成と実行 (AWS CLI)

アプリケーションの作成と実行 (コンソール)

以下の手順を実行し、コンソールを使用してアプリケーションを作成、設定、更新、および実行します。

アプリケーションの作成

1. https://console.aws.amazon.com/flink で Apache Flink 用 Managed Serviceコンソールを開く
2. Managed Service for Apache Flinkのダッシュボードで、「分析アプリケーションの作成」 を選択します。
3. 「Managed Service for Apache Flink-アプリケーションの作成」ページで、次のようにアプリケーションの詳細を入力します。
   - [アプリケーション名] には **`MyApplication`** と入力します。
   - [Description (説明)] に **`My java test app`** と入力します。
   - [ランタイム] には、[Apache Flink] を選択します。
   - バージョンプルダウンは「Apache Flink 1.8 (推奨バージョン)」のままにしておきます。
4. [アクセス許可] には、[IAM ロールの作成 / 更新**`kinesis-analytics-MyApplication-us-west-2`**] を選択します。
5. [Create application] を選択します。

> **Note**
>
> コンソールを使用して Apache Flink アプリケーション用 Managed Service を作成する場合は、IAM ロールとポリシーをアプリケーションが自動的に作成するオプションを選択できます。アプリケーションではこのロールとポリシーを使用して、依存リソースにアクセスします。これらの IAM リソースは、次のようにアプリケーション名とリージョンを使用して命名されます。
>
> - ポリシー: `kinesis-analytics-service-`*`MyApplication`*`-`*`us-west-2`*
> - ロール: `kinesisanalytics-`*`MyApplication`*`-`*`us-west-2`*

IAM ポリシーを編集する

IAM ポリシーを編集し、Kinesis Data Streamsにアクセスするための許可を追加します。

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. [ポリシー] を選択します。前のセクションでコンソールによって作成された **kinesis-analytics-service-MyApplication-us-west-2** ポリシーを選択します。
3. [概要] ページで、[ポリシーの編集] を選択します。[JSON] タブを選択します。
4. 次のポリシー例で強調表示されているセクションをポリシーに追加します。サンプルのアカウント ID (*012345678901*) を自分のアカウント ID に置き換えます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ReadCode",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": [
                "arn:aws:s3:::ka-app-code-username/aws-kinesis-analytics-java-
apps-1.0.jar"
            ]
        },
        {
            "Sid": "DescribeLogGroups",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:*"
            ]
        },
        {
            "Sid": "DescribeLogStreams",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogStreams"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:*"
            ]
```

```
        },
        {
            "Sid": "PutLogEvents",
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:kinesis-analytics-log-stream"
            ]
        },
        {
            "Sid": "ReadInputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
        },
        {
            "Sid": "WriteOutputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleOutputStream"
        }
    ]
}
```

アプリケーションを設定する

1. MyApplication ページで、 の設定 を選択します。
2. [Configure application] ページで、[Code location] を次のように指定します。
   - [Amazon S3 バケット] で、**ka-app-code-*<username>***と入力します。
   - [Amazon S3 オブジェクトへのパス] で、**aws-kinesis-analytics-java-apps-1.0.jar**と入力します。
3. [Access to application resources] の [Access permissions] では、[Create / update IAM role**kinesis-analytics-MyApplication-us-west-2**] を選択します。
4. 次のアプリケーションのプロパティと値を入力します。

グループ ID	キー	値
ProducerConfigProperties	**flink.inputstream.initpos**	**LATEST**
ProducerConfigProperties	**aws.region**	**us-west-2**
ProducerConfigProperties	**AggregationEnabled**	**false**

5. [Monitoring] の [Monitoring metrics level] が [Application] に設定されていることを確認します。
6. CloudWatch のログ記録では、有効化チェックボックスをオンにします。
7. [更新] を選択します。

> **Note**
>
> Amazon CloudWatch ログ記録を有効にすると、Managed Service for Apache Flink によってロググループとログストリームが作成されます。これらのリソースの名前は次のとおりです。
>
> - ロググループ: `/aws/kinesis-analytics/MyApplication`
> - ログストリーム: `kinesis-analytics-log-stream`

アプリケーションを実行する

1. MyApplication ページで、「 を実行」を選択します。アクションを確認します。
2. アプリケーションが実行されたら、ページを更新します。コンソールには [Application graph] が示されます。

アプリケーションの停止

MyApplication ページで、停止 を選択します。アクションを確認します。

アプリケーションの更新

コンソールを使用して、アプリケーションのプロパティ、モニタリング設定、アプリケーション JAR の場所またはファイル名などのアプリケーション設定を更新できます。アプリケーションコードを更新する必要がある場合は、アプリケーション JAR を Amazon S3 バケットから再ロードすることもできます。

MyApplication ページで、 の設定 を選択します。アプリケーションの設定を更新し、[更新] を選択します。

アプリケーションの作成と実行 (AWS CLI)

このセクションでは、 を使用して Managed Service for Apache Flink アプリケーション AWS CLI を作成して実行します。Managed Service for Apache Flink は、 `kinesisanalyticsv2` AWS CLI コマンドを使用して Managed Service for Apache Flink アプリケーションを作成して操作します。

アクセス許可ポリシーを作成する

> **Note**
>
> アプリケーションのアクセス権限ポリシーとロールを作成する必要があります。これらの IAM リソースを作成しない場合、アプリケーションはそのデータストリームやログストリームにアクセスできません。

まず、2 つのステートメントを含むアクセス許可ポリシーを作成します。1 つは、ソースストリームの `read` アクションに対するアクセス許可を付与し、もう 1 つはシンクストリームの `write` アクションに対するアクセス許可を付与します。次に、IAM ロール (次のセクションで作成) にポリシーをアタッチします。そのため、 Managed Service for Apache Flinkがこのロールを引き受けると、ソースストリームからの読み取りとシンクストリームへの書き込みを行うために必要なアクセス許可がサービスに付与されます。

次のコードを使用して `AKReadSourceStreamWriteSinkStream` アクセス許可ポリシーを作成します。*`username`* を Amazon S3 バケットの作成に使用したユーザー名に置き換え、アプリケーションコードを保存します。Amazon リソースネーム (ARN) のアカウント ID (*`012345678901`*) を自分のアカウント ID に置き換えます。

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

```
            "Sid": "S3",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": ["arn:aws:s3:::ka-app-code-username",
                "arn:aws:s3:::ka-app-code-username/*"
            ]
        },
        {
            "Sid": "ReadInputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
        },
        {
            "Sid": "WriteOutputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleOutputStream"
        }
    ]
}
```

アクセス許可ポリシーを作成する step-by-step 手順については、IAM ユーザーガイドの「チュートリアル: 最初のカスタマー管理ポリシーの作成とアタッチ」を参照してください。

> **Note**
>
> その他のアマゾンサービスにアクセスするには、 AWS SDK for Javaを使用します。Managed Service for Apache Flink は、アプリケーションに関連付けられているサービス実行 IAM ロールに、SDK が必要とする認証情報を自動的に設定します。追加の手順は必要ありません。

IAM ロールを作成する

このセクションでは、Managed Service for Apache Flink アプリケーションがソースストリームを読み取り、シンクストリームに書き込むために想定できる IAM ロールを作成します。

Apache Flink 用 Managed Service は、許可なしにはストリームにアクセスできません。IAM ロールを介してこれらの許可を付与します。各 IAM ロールには、2 つのポリシーがアタッチされます。信頼ポリシーは、ロールを引き受けるための許可を Managed Service for Apache Flink 付与し、許可ポリシーは、ロールを引き受けた後に Managed Service for Apache Flink が実行できる事柄を決定します。

前のセクションで作成したアクセス許可ポリシーをこのロールにアタッチします。

IAM ロールを作成するには

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. ナビゲーションペインで [Roles (ロール)]、[Create Role (ロールの作成)] の順に選択します。
3. [信頼されるエンティティの種類を選択] で、[AWS のサービス] を選択します。[このロールを使用するサービスを選択] で、[Kinesis Analytics] を選択します。[ユースケースの選択] で、[Kinesis Analytics] を選択します。

   [Next: Permissions] (次のステップ: 許可) を選択します。
4. [アクセス権限ポリシーをアタッチする] ページで、[Next: Review] (次: 確認) を選択します。ロールを作成した後に、アクセス許可ポリシーをアタッチします。
5. [Create role (ロールの作成)] ページで、ロールの名前に **`MF-stream-rw-role`** を入力します。[ロールの作成] を選択します。

   これで、`MF-stream-rw-role` と呼ばれる新しい IAM ロールが作成されます。次に、ロールの信頼ポリシーとアクセス許可ポリシーを更新します。
6. アクセス許可ポリシーをロールにアタッチします。

   > **Note**
   >
   > この演習では、Managed Service for Apache Flink が、Kinesis データストリーム (ソース) からのデータの読み取りと、別の Kinesis データストリームへの出力の書き込みの両方を実行するためにこのロールを引き受けます。このため、前のステップで作成したポリシー、the section called “アクセス許可ポリシーを作成する” をアタッチします。

   a. [概要] ページで、[アクセス許可] タブを選択します。

   b. [Attach Policies (ポリシーのアタッチ)] を選択します。

c. 検索ボックスに**AKReadSourceStreamWriteSinkStream**(前のセクションで作成したポリシー) と入力します。

d. AKReadSourceStreamWriteSinkStream ポリシーを選択し、ポリシーのアタッチ を選択します。

これで、アプリケーションがリソースにアクセスするために使用するサービスの実行ロールが作成されました。新しいロールの ARN を書き留めておきます。

ロールの作成 step-by-step 手順については、IAM ユーザーガイドの「IAM ロールの作成 (コンソール)」を参照してください。

Managed Service for Apache Flink アプリケーションを作成する

1. 次の JSON コードを create_request.json という名前のファイルに保存します。サンプルロールの ARN を、前に作成したロールの ARN に置き換えます。バケット ARN のサフィックス (*username*) を、前のセクションで選択したサフィックスに置き換えます。サービス実行ロールのサンプルのアカウント ID (*012345678901*) を、自分のアカウント ID に置き換えます。

```
{
    "ApplicationName": "test",
    "ApplicationDescription": "my java test app",
    "RuntimeEnvironment": "FLINK-1_8",
    "ServiceExecutionRole": "arn:aws:iam::012345678901:role/MF-stream-rw-role",
    "ApplicationConfiguration": {
        "ApplicationCodeConfiguration": {
            "CodeContent": {
                "S3ContentLocation": {
                    "BucketARN": "arn:aws:s3:::ka-app-code-username",
                    "FileKey": "aws-kinesis-analytics-java-apps-1.0.jar"
                }
            },
            "CodeContentType": "ZIPFILE"
        },
        "EnvironmentProperties":  {
         "PropertyGroups": [
            {
               "PropertyGroupId": "ProducerConfigProperties",
               "PropertyMap" : {
                    "flink.stream.initpos" : "LATEST",
                    "aws.region" : "us-west-2",
                    "AggregationEnabled" : "false"
```

```
                }
            },
            {
                "PropertyGroupId": "ConsumerConfigProperties",
                "PropertyMap" : {
                     "aws.region" : "us-west-2"
                }
            }
        ]
      }
    }
}
```

2. 前述のリクエストを指定して CreateApplication アクションを実行し、アプリケーションを作成します。

```
aws kinesisanalyticsv2 create-application --cli-input-json file://
create_request.json
```

これでアプリケーションが作成されました。次のステップでは、アプリケーションを起動します。

アプリケーションを起動する

このセクションでは、StartApplication アクションを使用してアプリケーションを起動します。

アプリケーションを起動するには

1. 次の JSON コードを start_request.json という名前のファイルに保存します。

```
{
    "ApplicationName": "test",
    "RunConfiguration": {
        "ApplicationRestoreConfiguration": {
         "ApplicationRestoreType": "RESTORE_FROM_LATEST_SNAPSHOT"
         }
    }
}
```

2. 前述のリクエストを指定して StartApplication アクションを実行し、アプリケーションを起動します。

```
aws kinesisanalyticsv2 start-application --cli-input-json file://start_request.json
```

アプリケーションが実行されます。Amazon CloudWatch コンソールで Managed Service for Apache Flink メトリクスをチェックして、アプリケーションが動作していることを確認できます。

アプリケーションの停止

このセクションでは、StopApplication アクションを使用してアプリケーションを停止します。

アプリケーションを停止するには

1. 次の JSON コードを stop_request.json という名前のファイルに保存します。

```
{
   "ApplicationName": "test"
}
```

2. 次のリクエストを指定して StopApplication アクションを実行し、アプリケーションを停止します。

```
aws kinesisanalyticsv2 stop-application --cli-input-json file://stop_request.json
```

アプリケーションが停止します。

CloudWatch ログ記録オプションを追加する

を使用して AWS CLI 、Amazon CloudWatch ログストリームをアプリケーションに追加できます。アプリケーションで CloudWatch ログを使用する方法については、「」を参照してくださいthe section called “ログ記録の設定”。

環境プロパティを更新する

このセクションでは、「UpdateApplication」アクションを使用して、アプリケーションコードを再コンパイルせずにアプリケーションの環境プロパティを変更します。この例では、ソースストリームおよびレプリケート先ストリームのリージョンを変更します。

アプリケーションの環境プロパティを更新します

1. 次の JSON コードを update_properties_request.json という名前のファイルに保存します。

```
{"ApplicationName": "test",
   "CurrentApplicationVersionId": 1,
   "ApplicationConfigurationUpdate": {
      "EnvironmentPropertyUpdates": {
         "PropertyGroups": [
            {
               "PropertyGroupId": "ProducerConfigProperties",
               "PropertyMap" : {
                    "flink.stream.initpos" : "LATEST",
                    "aws.region" : "us-west-2",
                    "AggregationEnabled" : "false"
               }
            },
            {
               "PropertyGroupId": "ConsumerConfigProperties",
               "PropertyMap" : {
                    "aws.region" : "us-west-2"
               }
            }
         ]
      }
   }
}
```

2. 前述のリクエストでUpdateApplicationアクションを実行し、環境プロパティを更新します。

```
aws kinesisanalyticsv2 update-application --cli-input-json file://
update_properties_request.json
```

アプリケーションコードの更新

アプリケーションコードを新しいバージョンのコードパッケージで更新する必要がある場合は、UpdateApplication AWS CLI アクションを使用します。

> **Note**
>
> 同じファイル名のアプリケーションコードの新しいバージョンをロードするには、新しいオブジェクトバージョンを指定する必要があります。Amazon S3 オブジェクトバージョンを使用する方法の詳細については、「バージョニングの有効化または無効化」を参照してください。

を使用するには AWS CLI、Amazon S3 バケットから以前のコードパッケージを削除し、新しいバージョンをアップロードして、同じ Amazon S3 バケットとオブジェクト名、および新しいオブジェクトバージョンUpdateApplicationを指定して を呼び出します。アプリケーションは新しいコードパッケージで再起動します。

以下の UpdateApplication アクションのサンプル・リクエストは、アプリケーション・コードを再読み込 み、アプリケーションを再起動します。CurrentApplicationVersionId を現在のアプリケーションバージョンに更新します。ListApplications または DescribeApplication アクションを使用して、現在のアプリケーションバージョンを確認できます。バケット名のサフィックス (「*<username>*」) を、 the section called “2 つの Amazon Kinesis データストリームを作成する” セクションで選択したサフィックスで更新します。

```
{
    "ApplicationName": "test",
    "CurrentApplicationVersionId": 1,
    "ApplicationConfigurationUpdate": {
        "ApplicationCodeConfigurationUpdate": {
            "CodeContentUpdate": {
                "S3ContentLocationUpdate": {
                    "BucketARNUpdate": "arn:aws:s3:::ka-app-code-username",
                    "FileKeyUpdate": "aws-kinesis-analytics-java-apps-1.0.jar",
                    "ObjectVersionUpdate": "SAMPLEUehYngP87ex1nzYIGYgfhypvDU"
                }
            }
        }
    }
}
```

次のステップ

ステップ 4: リソースをクリーンアップ AWS する

ステップ 4: リソースをクリーンアップ AWS する

このセクションでは、入門チュートリアルで作成した AWS リソースをクリーンアップする手順について説明します。

このトピックには、次のセクションが含まれています。

- Managed Service for Apache Flink アプリケーションを削除する
- Kinesis データストリームを削除する
- Amazon S3 オブジェクトとバケットを削除する
- IAM リソースを削除する
- CloudWatch リソースを削除する

Managed Service for Apache Flink アプリケーションを削除する

1. Kinesis コンソール (https://console.aws.amazon.com/kinesis) を開きます。
2. Managed Service for Apache Flink パネルで、 を選択しますMyApplication。
3. [設定] を選択します。
4. スナップショットセクションで「無効」を選択して、更新を選択します。
5. アプリケーションのページで[削除]を選択し、削除を確認します。

Kinesis データストリームを削除する

1. https://console.aws.amazon.com/flink で Apache Flink 用 Managed Serviceコンソールを開く
2. Kinesis Data Streams パネルで、 を選択しますExampleInputStream。
3. ExampleInputStream ページで、Kinesis ストリームの削除を選択し、削除を確定します。
4. Kinesis ストリームページで、 を選択しExampleOutputStream、アクション を選択し、削除 を選択し、削除を確定します。

Amazon S3 オブジェクトとバケットを削除する

1. https://console.aws.amazon.com/s3/でAmazon S3 コンソールを開きます。
2. ka-app-code-***<username>*** バケットを選択します。
3. [削除] を選択し、バケット名を入力して削除を確認します。

IAM リソースを削除する

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. ナビゲーションバーで、[ポリシー] を選択します。
3. フィルターコントロールに「kinesis」と入力します。
4. kinesis-analytics-service-MyApplication-us-west-2 ポリシーを選択します。
5. [ポリシーアクション]、[削除] の順に選択します。
6. ナビゲーションバーで [ロール] を選択します。
7. kinesis-analytics-MyApplication-us-west-2 ロールを選択します。
8. [ロールの削除] を選択し、削除を確定します。

CloudWatch リソースを削除する

1. https://console.aws.amazon.com/cloudwatch/ で CloudWatch コンソールを開きます。
2. ナビゲーションバーで [ログ] を選択します。
3. /aws/kinesis-analytics/MyApplication ロググループを選択します。
4. [ロググループの削除]を選択し、削除を確認してください。

開始方法: Flink 1.6.2 - 非推奨

> **Note**
>
> Apache Flink バージョン 1.6、1.8、および 1.11 は、Apache Flink コミュニティで 3 年以上サポートされていません。Amazon Managed Service for Apache Flink では、2024 年 11 月 5 日にこれらのバージョンを廃止する予定です。この日以降、これらの Flink バージョン用の新しいアプリケーションを作成することはできません。現時点では、既存のアプリケーションの実行を継続できます。Amazon Managed Service for Apache Flink のインプレースバージョンアップグレード機能を使用して、アプリケーションをステートリーにアップグレードできます。詳細については、「」を参照してくださいApache Flink のインプレースバージョンアップグレード。

このトピックには、Apache Flink 1.6.2 開始方法 (DataStream API)を使用するチュートリアルのバージョンが含まれています。

トピック

- Managed Service for Apache Flink アプリケーションのコンポーネント
- 演習を完了するための前提条件
- ステップ 1: AWS アカウントを設定し、管理者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: Managed Service for Apache Flink アプリケーションを作成して実行する
- ステップ 4: リソースをクリーンアップ AWS する

Managed Service for Apache Flink アプリケーションのコンポーネント

Apache Flink アプリケーション用 Managed Service では、Apache Flink ランタイムを使用して入力を処理し、出力を生成する Java/Apache Maven または Scala アプリケーションを使用してデータを処理します。

Apache Flink 用 Managed Service のコンポーネントは次のとおりです。

- 「ランタイムプロパティ:」「ランタイムプロパティ」を使用すると、アプリケーションコードを再コンパイルせずにアプリケーションを設定できます。
- 「ソース:」 アプリケーションは 「ソース」 を使用してデータを消費します。ソースコネクタは、Kinesis データストリーム、Amazon S3 バケットなどからデータを読み取ります。詳細については、「[Sources] (出典)」を参照してください。
- 「オペレータ:」アプリケーションは 1 つ以上の「オペレータ」を使用してデータを処理します。オペレータはデータを変換、強化、または集約できます。詳細については、 DataStream API 演算子 を参照してください。
- 「シンク:」アプリケーションは「シンク」を使用して外部ソースにデータを生成します。シンクコネクタは、Kinesis データストリーム、Firehose ストリーム、Amazon S3 バケットなどにデータを書き込みます。詳細については、「シンク」を参照してください。

アプリケーションを作成、コンパイル、パッケージ化したら、コードパッケージを Amazon Simple Storage Service (Amazon S3) バケットにアップロードします。次に、Apache Flink アプリケーション用 Managed Serviceを作成します。コードパッケージの場所、ストリーミングデータソースとして Kinesis データストリームを渡し、通常はアプリケーションの処理済みデータを受け取るストリーミングまたはファイルの場所を渡します。

演習を完了するための前提条件

このガイドの手順を完了するには、以下が必要です。

- Java 開発キット (JDK) バージョン 8。JAVA_HOME 環境変数を、JDK のインストール場所を指すように設定します。
- 開発環境 (Eclipse Java Neon や IntelliJ Idea など) を使用してアプリケーションを開発し、コンパイルすることをお勧めします。
- Git クライアント。Git クライアントをまだインストールしていない場合は、インストールします。
- Apache Maven Compiler Plugin。Maven が作業パスに含まれている必要があります。Apache Maven のインストールをテストするには、次のように入力します。

```
$ mvn -version
```

開始するには、ステップ 1: AWS アカウントを設定し、管理者ユーザーを作成するに進みます。

ステップ 1: AWS アカウントを設定し、管理者ユーザーを作成する

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環として、通話呼び出しを受け取り、電話キーパッドで検証コードを入力するように求められます。

   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。セキュリティのベストプラクティスとして、ユーザーに管理アクセスを割り当て、ルートユーザーのみを使用してルートユーザーアクセスが必要なタスクを実行してください。

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

のセキュリティ保護 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 「ルートユーザーの仮想 MFA デバイスを有効にする (コンソール)」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、「 ユーザーガイド」の「デフォルト でユーザーアクセス IAM アイデンティティセンターディレクトリを設定するAWS IAM Identity Center 」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「グループの参加」を参照してください。

プログラム的なアクセス権を付与する

ユーザーが の AWS 外部で を操作する場合は、プログラムによるアクセスが必要です AWS Management Console。プログラムによるアクセスを許可する方法は、 にアクセスするユーザーのタイプによって異なります AWS。

ユーザーにプログラマチックアクセス権を付与するには、以下のいずれかのオプションを選択します。

プログラマチックアクセス権を必要とするユーザー	目的	方法
ワークフォースアイデンティティ (IAM Identity Center で管理されているユーザー)	一時的な認証情報を使用して、、 AWS SDKs AWS CLI、または AWS APIs。	使用するインターフェイス用の手引きに従ってください。 • については AWS CLI、「 ユーザーガイド」の AWS CLI 「 を使用するための の設定 AWS IAM Identity CenterAWS Command Line

プログラマチックアクセス権を必要とするユーザー	目的	方法
		Interface 」を参照してください。 • AWS SDKs 、ツール、AWS APIs「 SDK とツールのリファレンスガイド」の「IAM Identity Center 認証」を参照してください。 AWS SDKs
IAM	一時的な認証情報を使用して、、 AWS SDKs AWS CLI、または AWS APIs。	「IAM ユーザーガイド」の「 AWS リソースでの一時的な認証情報の使用」の手順に従います。
IAM	(非推奨) 長期認証情報を使用して、、 AWS SDKs AWS CLI、または AWS APIs。	使用するインターフェイス用の手引きに従ってください。 • については AWS CLI、「 AWS Command Line Interface ユーザーガイド」の「IAM ユーザー認証情報を使用した認証」を参照してください。 • AWS SDKs「 SDK とツールのリファレンスガイド」の「長期的な認証情報を使用した認証」を参照してください。 AWS SDKs • AWS APIsユーザーガイド」の「IAM ユーザーのアクセスキーの管理」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

このステップでは、Apache Flink 用 Managed Service で使用する をダウンロードして設定 AWS CLI します。

> **Note**
>
> このガイドの使用開始実習では、操作を実行するために、アカウントの管理者の認証情報 (adminuser) を使用していることが前提となっています。

> **Note**
>
> が既に AWS CLI インストールされている場合は、アップグレードして最新の機能を取得する必要がある場合があります。詳細については、「AWS Command Line Interface ユーザーガイド」の「AWS Command Line Interfaceのインストール」を参照してください。のバージョンを確認するには AWS CLI、次のコマンドを実行します。
>
> ```
> aws --version
> ```
>
> このチュートリアルの演習では、次の AWS CLI バージョン 以降が必要です。
>
> ```
> aws-cli/1.16.63
> ```

を設定するには 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。
   - AWS Command Line Interfaceのインストール
   - AWS CLIの設定
2. 管理者ユーザーの名前付きプロファイルを config ファイルに追加します AWS CLI 。 AWS CLI コマンドを実行するときに、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 AWS リージョンのリストについては、「」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。

> **Note**
>
> このチュートリアルのサンプルコードとコマンドでは、米国西部 (オレゴン) リージョンを使用しています。別の リージョンを使用するには、このチュートリアルのコードとコマンドのリージョンを、使用したいリージョンに変更します。

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
```

AWS アカウントと をセットアップしたら AWS CLI、次の演習を試してサンプルアプリケーションを設定し、 end-to-end セットアップをテストできます。

次のステップ

ステップ 3: Managed Service for Apache Flink アプリケーションを作成して実行する

ステップ 3: Managed Service for Apache Flink アプリケーションを作成して実行する

この演習では、データストリームをソースおよびシンクとして使用して、Managed Service for Apache Flink アプリケーションを作成します。

このセクションには、以下のステップが含まれています。

- 2 つの Amazon Kinesis データストリームを作成する
- サンプルレコードを入力ストリームに書き込む
- Apache Flink ストリーミング Java コードをダウンロードして調べる
- アプリケーションコードをコンパイルする
- Apache Flink ストリーミング Java コードをアップロードする
- Managed Service for Apache Flink アプリケーションを作成して実行する

2 つの Amazon Kinesis データストリームを作成する

この演習で Apache Flink アプリケーションのマネージドサービスを作成する前に、2 つの Kinesis データストリーム (ExampleInputStream と ExampleOutputStream) を作成する必要があります。アプリケーションでは、これらのストリームを使用してアプリケーションの送信元と送信先のストリームを選択します。

これらのストリームは Amazon Kinesis コンソールまたは次の AWS CLI コマンドを使用して作成できます。コンソールでの操作方法については、「Amazon Kinesis Data Streams デベロッパーガイド」 の 「データストリームの作成および更新」 を参照してください。

データストリームを作成するには (AWS CLI)

1. 最初のストリーム (ExampleInputStream) を作成するには、次の Amazon Kinesis create-stream AWS CLI コマンドを使用します。

```
$ aws kinesis create-stream \
--stream-name ExampleInputStream \
--shard-count 1 \
--region us-west-2 \
--profile adminuser
```

2. 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 ExampleOutputStream \
--shard-count 1 \
--region us-west-2 \
--profile adminuser
```

サンプルレコードを入力ストリームに書き込む

このセクションでは、Python スクリプトを使用して、アプリケーションが処理するサンプルレコードをストリームに書き込みます。

> **Note**
>
> このセクションでは AWS SDK for Python (Boto) が必要です。

1. 次の内容で、stock.py という名前のファイルを作成します。

```
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"))
```

2. このチュートリアルの後半では、アプリケーションにデータを送信する stock.py スクリプトを実行します。

```
$ python stock.py
```

Apache Flink ストリーミング Java コードをダウンロードして調べる

この例の Java アプリケーションコードは、 から入手できます GitHub。アプリケーションコードをダウンロードするには、次の操作を行います。

1. 次のコマンドを使用してリモートリポジトリのクローンを作成します。

```
git clone https://github.com/aws-samples/amazon-kinesis-data-analytics-examples.git
```

2. amazon-kinesis-data-analytics-java-examples/GettingStarted_1_6 ディレクトリに移動します。

アプリケーションコードに関して、以下の点に注意してください。

- 「Project Object Model (pom.xml)」ファイルには、Managed Service for Apache Flink 用ライブラリなど、アプリケーションの設定と依存関係に関する情報が含まれています。
- BasicStreamingJob.java ファイルには、アプリケーションの機能を定義する main メソッドが含まれています。
- アプリケーションは Kinesis ソースを使用して、ソースストリームから読み取りを行います。次のスニペットでは、Kinesis ソースが作成されます。

```
return env.addSource(new FlinkKinesisConsumer<>(inputStreamName,
                new SimpleStringSchema(), inputProperties));
```

- アプリケーションでは、ソースおよびシンクコネクタを作成し、StreamExecutionEnvironment オブジェクトを使用して外部リソースにアクセスします。
- アプリケーションでは、静的プロパティを使用してソースおよびシンクコネクタを作成します。動的なアプリケーションプロパティを使用するには、createSourceFromApplicationProperties および createSinkFromApplicationProperties メソッドを使用してコネクタを作成します。これらのメソッドは、アプリケーションのプロパティを読み取ってコネクタを設定します。

  ランタイムプロパティの詳細については、ランタイムプロパティ を参照してください。

アプリケーションコードをコンパイルする

このセクションでは、Apache Maven コンパイラを使用してアプリケーション用の Java コードを作成します。Apache Maven と Java 開発キット (JDK) をインストールする方法については、演習を完了するための前提条件を参照してください。

> **Note**
>
> 1.11 より前のバージョンの Apache Flink で Kinesis コネクタを使用するには、コネクタのソース コードをダウンロードし、Apache Flink ドキュメント の説明に従ってビルドする必要があります。

アプリケーションコードをコンパイルするには

1. アプリケーションコードを使用するには、コードをコンパイルして JAR ファイルにパッケージ化します。コードのコンパイルとパッケージ化には次の 2 通りの方法があります。

   - コマンドライン Maven ツールを使用します。pom.xml ファイルが格納されているディレクトリで次のコマンドを実行して JAR ファイルを作成します。

     ```
     mvn package
     ```

     > **Note**
     >
     > -DFlink.version パラメーターは、Apache Flink ランタイムのマネージドサービスバージョン 1.0.1 には必要ありません。バージョン 1.1.0 以降でのみ必要です。詳細については、「the section called “アプリケーションの Apache Flink バージョンの指定”」を参照してください。

   - 開発環境を使用します。詳細については、開発環境のドキュメントを参照してください。

   パッケージは JAR ファイルとしてアップロードすることも、圧縮して ZIP ファイルとしてアップロードすることもできします。を使用してアプリケーションを作成する場合は AWS CLI、コードコンテンツタイプ (JAR または ZIP) を指定します。

2. コンパイル中にエラーが発生した場合は、JAVA_HOME 環境変数が正しく設定されていることを確認します。

アプリケーションのコンパイルに成功すると、次のファイルが作成されます。

```
target/aws-kinesis-analytics-java-apps-1.0.jar
```

Apache Flink ストリーミング Java コードをアップロードする

このセクションでは、Amazon Simple Storage Service (Amazon S3) バケットを作成し、アプリケーションコードをアップロードします。

アプリケーションコードをアップロードするには

1. Amazon S3 コンソール (https://console.aws.amazon.com/s3/) を開きます。
2. [バケットを作成] を選択します。
3. [Bucket name (バケット名)] フィールドに**ka-app-code-*<username>***と入力します。バケット名にユーザー名などのサフィックスを追加して、グローバルに一意にします。[次へ] をクリックします。
4. 設定オプションのステップでは、設定をそのままにし、[次へ] を選択します。
5. アクセス許可の設定のステップでは、設定をそのままにし、[次へ] を選択します。
6. [バケットを作成] を選択します。
7. Amazon S3 コンソールで、ka-app-code-***<username>*** バケットを選択し、アップロード を選択します。
8. ファイルの選択のステップで、[ファイルを追加] を選択します。前のステップで作成した `aws-kinesis-analytics-java-apps-1.0.jar` ファイルに移動します。[次へ] をクリックします。
9. アクセス許可の設定のステップでは、設定をそのままにします。[次へ] をクリックします。
10. プロパティの設定のステップでは、設定をそのままにします。[アップロード] を選択します。

アプリケーションコードが Amazon S3 バケットに保存され、アプリケーションからアクセスできるようになります。

Managed Service for Apache Flink アプリケーションを作成して実行する

コンソールまたは AWS CLIのいずれかを使用してManaged Service for Apache Flink を作成し、実行することができます。

> **Note**
>
> コンソールを使用してアプリケーションを作成すると、 AWS Identity and Access Management (IAM) と Amazon CloudWatch Logs リソースが自動的に作成されます。を使用してアプリケーションを作成するときは AWS CLI、これらのリソースを個別に作成します。

トピック

- アプリケーションの作成と実行 (コンソール)
- アプリケーションの作成と実行 (AWS CLI)

アプリケーションの作成と実行 (コンソール)

以下の手順を実行し、コンソールを使用してアプリケーションを作成、設定、更新、および実行します。

アプリケーションの作成

1. https://console.aws.amazon.com/flink で Apache Flink 用 Managed Serviceコンソールを開く
2. Managed Service for Apache Flinkのダッシュボードで、「分析アプリケーションの作成」 を選択します。
3. 「Managed Service for Apache Flink-アプリケーションの作成」ページで、次のようにアプリケーションの詳細を入力します。
   - [アプリケーション名] には **MyApplication** と入力します。
   - [Description (説明)] に **My java test app** と入力します。
   - [ランタイム] には、[Apache Flink] を選択します。

     > **Note**
     >
     > Apache Flink 用 Managed Serviceは Apache Flink バージョン 1.8.2 または 1.6.2 を使用します。

   - バージョンプルダウンを「Apache Flink 1.6」に変更します。
4. [アクセス許可] には、[IAM ロールの作成 / 更新**kinesis-analytics-MyApplication-us-west-2**] を選択します。
5. [Create application] を選択します。

> **Note**
>
> コンソールを使用して Apache Flink アプリケーション用 Managed Service を作成する場合は、IAM ロールとポリシーをアプリケーションが自動的に作成するオプションを選択できます。アプリケーションではこのロールとポリシーを使用して、依存リソースにアクセスしま

す。これらの IAM リソースは、次のようにアプリケーション名とリージョンを使用して命名されます。

- ポリシー: kinesis-analytics-service-*MyApplication*-*us-west-2*
- ロール: kinesisanalytics-*MyApplication*-*us-west-2*

IAM ポリシーを編集する

IAM ポリシーを編集し、Kinesis Data Streamsにアクセスするための許可を追加します。

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. [ポリシー] を選択します。前のセクションでコンソールによって作成された **kinesis-analytics-service-MyApplication-us-west-2** ポリシーを選択します。
3. [概要] ページで、[ポリシーの編集] を選択します。[JSON] タブを選択します。
4. 次のポリシー例で強調表示されているセクションをポリシーに追加します。サンプルのアカウント ID (*012345678901*) を自分のアカウント ID に置き換えます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ReadCode",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": [
                "arn:aws:s3:::ka-app-code-username/java-getting-started-1.0.jar"
            ]
        },
        {
            "Sid": "DescribeLogGroups",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:*"
            ]
```

```
        },
        {
            "Sid": "DescribeLogStreams",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogStreams"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:*"
            ]
        },
        {
            "Sid": "PutLogEvents",
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:kinesis-analytics-log-stream"
            ]
        },
        {
            "Sid": "ReadInputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
        },
        {
            "Sid": "WriteOutputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleOutputStream"
        }
    ]
}
```

アプリケーションを設定する

1. MyApplication ページで、 の設定 を選択します。

2. [Configure application] ページで、[Code location] を次のように指定します。
   - [Amazon S3 バケット] で、**ka-app-code-*<username>***と入力します。
   - [Amazon S3 オブジェクトへのパス] で、**java-getting-started-1.0.jar**と入力します。
3. [Access to application resources] の [Access permissions] では、[Create / update IAM role**kinesis-analytics-MyApplication-us-west-2**] を選択します。
4. 次のアプリケーションのプロパティと値を入力します。

グループ ID	キー	値
ProducerConfigProperties	**flink.inputstream.initpos**	**LATEST**
ProducerConfigProperties	**aws.region**	**us-west-2**
ProducerConfigProperties	**AggregationEnabled**	**false**

5. [Monitoring] の [Monitoring metrics level] が [Application] に設定されていることを確認します。
6. CloudWatch のログ記録では、有効化チェックボックスをオンにします。
7. [更新] を選択します。

> **Note**
>
> Amazon CloudWatch ログ記録を有効にすると、Managed Service for Apache Flink によってロググループとログストリームが作成されます。これらのリソースの名前は次のとおりです。
>
> - ロググループ: `/aws/kinesis-analytics/MyApplication`
> - ログストリーム: `kinesis-analytics-log-stream`

アプリケーションを実行する

1. MyApplication ページで、 の実行 を選択します。アクションを確認します。

2. アプリケーションが実行されたら、ページを更新します。コンソールには [Application graph] が示されます。

アプリケーションの停止

MyApplication ページで、停止 を選択します。アクションを確認します。

アプリケーションの更新

コンソールを使用して、アプリケーションのプロパティ、モニタリング設定、アプリケーション JAR の場所またはファイル名などのアプリケーション設定を更新できます。アプリケーションコードを更新する必要がある場合は、アプリケーション JAR を Amazon S3 バケットから再ロードすることもできます。

MyApplication ページで、 の設定 を選択します。アプリケーションの設定を更新し、[更新] を選択します。

アプリケーションの作成と実行 (AWS CLI)

このセクションでは、 を使用して Managed Service for Apache Flink アプリケーション AWS CLI を作成して実行します。Managed Service for Apache Flink は、 `kinesisanalyticsv2` AWS CLI コマンドを使用して Managed Service for Apache Flink アプリケーションを作成して操作します。

許可ポリシーを作成する

まず、2 つのステートメントを含むアクセス許可ポリシーを作成します。1 つは、ソースストリームの `read` アクションに対するアクセス許可を付与し、もう 1 つはシンクストリームの `write` アクションに対するアクセス許可を付与します。次に、IAM ロール (次のセクションで作成) にポリシーをアタッチします。そのため、 Managed Service for Apache Flinkがこのロールを引き受けると、ソースストリームからの読み取りとシンクストリームへの書き込みを行うために必要なアクセス許可がサービスに付与されます。

次のコードを使用して `AKReadSourceStreamWriteSinkStream` アクセス許可ポリシーを作成します。*`username`* を Amazon S3 バケットの作成に使用したユーザー名に置き換え、アプリケーションコードを保存します。Amazon リソースネーム (ARN) のアカウント ID (*`012345678901`*) を自分のアカウント ID に置き換えます。

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

```
            "Sid": "S3",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": ["arn:aws:s3:::ka-app-code-username",
                "arn:aws:s3:::ka-app-code-username/*"
            ]
        },
        {
            "Sid": "ReadInputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
        },
        {
            "Sid": "WriteOutputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleOutputStream"
        }
    ]
}
```

アクセス許可ポリシーを作成する step-by-step 手順については、IAM ユーザーガイドの「チュートリアル: 最初のカスタマー管理ポリシーの作成とアタッチ」を参照してください。

> **Note**
>
> その他のアマゾンサービスにアクセスするには、 AWS SDK for Javaを使用します。Managed Service for Apache Flink は、アプリケーションに関連付けられているサービス実行 IAM ロールに、SDK が必要とする認証情報を自動的に設定します。追加の手順は必要ありません。

IAM ロールを作成する

このセクションでは、Managed Service for Apache Flink アプリケーションがソースストリームを読み取り、シンクストリームに書き込むために想定できる IAM ロールを作成します。

Apache Flink 用 Managed Service は、許可なしにはストリームにアクセスできません。IAM ロールを介してこれらの許可を付与します。各 IAM ロールには、2 つのポリシーがアタッチされます。信頼ポリシーは、ロールを引き受けるための許可を Managed Service for Apache Flink 付与し、許可ポリシーは、ロールを引き受けた後に Managed Service for Apache Flink が実行できる事柄を決定します。

前のセクションで作成したアクセス許可ポリシーをこのロールにアタッチします。

IAM ロールを作成するには

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. ナビゲーションペインで [Roles (ロール)]、[Create Role (ロールの作成)] の順に選択します。
3. [信頼されるエンティティの種類を選択] で、[AWS のサービス] を選択します。[このロールを使用するサービスを選択] で、[Kinesis Analytics] を選択します。[ユースケースの選択] で、[Kinesis Analytics] を選択します。

   [Next: Permissions] (次のステップ: 許可) を選択します。
4. [アクセス権限ポリシーをアタッチする] ページで、[Next: Review] (次: 確認) を選択します。ロールを作成した後に、アクセス許可ポリシーをアタッチします。
5. [Create role (ロールの作成)] ページで、ロールの名前に **`MF-stream-rw-role`** を入力します。[ロールの作成] を選択します。

   これで、`MF-stream-rw-role` と呼ばれる新しい IAM ロールが作成されます。次に、ロールの信頼ポリシーとアクセス許可ポリシーを更新します。
6. アクセス許可ポリシーをロールにアタッチします。

   > **Note**
   >
   > この演習では、Managed Service for Apache Flink が、Kinesis データストリーム (ソース) からのデータの読み取りと、別の Kinesis データストリームへの出力の書き込みの両方を実行するためにこのロールを引き受けます。このため、前のステップで作成したポリシー、the section called “許可ポリシーを作成する” をアタッチします。

   a. [概要] ページで、[アクセス許可] タブを選択します。

   b. [Attach Policies (ポリシーのアタッチ)] を選択します。

c. 検索ボックスに**AKReadSourceStreamWriteSinkStream**(前のセクションで作成したポリシー) と入力します。

d. AKReadSourceStreamWriteSinkStream ポリシーを選択し、ポリシー をアタッチ を選択します。

これで、アプリケーションがリソースにアクセスするために使用するサービスの実行ロールが作成されました。新しいロールの ARN を書き留めておきます。

ロールを作成する step-by-step 手順については、IAM ユーザーガイドの「IAM ロールの作成 (コンソール)」を参照してください。

Managed Service for Apache Flink アプリケーションを作成する

1. 次の JSON コードを create_request.json という名前のファイルに保存します。サンプルロールの ARN を、前に作成したロールの ARN に置き換えます。バケット ARN のサフィックス (*username*) を、前のセクションで選択したサフィックスに置き換えます。サービス実行ロールのサンプルのアカウント ID (*012345678901*) を、自分のアカウント ID に置き換えます。

```
{
    "ApplicationName": "test",
    "ApplicationDescription": "my java test app",
    "RuntimeEnvironment": "FLINK-1_6",
    "ServiceExecutionRole": "arn:aws:iam::012345678901:role/MF-stream-rw-role",
    "ApplicationConfiguration": {
        "ApplicationCodeConfiguration": {
            "CodeContent": {
                "S3ContentLocation": {
                    "BucketARN": "arn:aws:s3:::ka-app-code-username",
                    "FileKey": "java-getting-started-1.0.jar"
                }
            },
            "CodeContentType": "ZIPFILE"
        },
        "EnvironmentProperties":  {
         "PropertyGroups": [
            {
               "PropertyGroupId": "ProducerConfigProperties",
               "PropertyMap" : {
                    "flink.stream.initpos" : "LATEST",
                    "aws.region" : "us-west-2",
                    "AggregationEnabled" : "false"
```

```
                }
            },
            {
                "PropertyGroupId": "ConsumerConfigProperties",
                "PropertyMap" : {
                     "aws.region" : "us-west-2"
                }
            }
        ]
      }
    }
}
```

2. 前述のリクエストを指定して CreateApplication アクションを実行し、アプリケーションを作成します。

```
aws kinesisanalyticsv2 create-application --cli-input-json file://
create_request.json
```

これでアプリケーションが作成されました。次のステップでは、アプリケーションを起動します。

アプリケーションを起動する

このセクションでは、StartApplication アクションを使用してアプリケーションを起動します。

アプリケーションを起動するには

1. 次の JSON コードを start_request.json という名前のファイルに保存します。

```
{
    "ApplicationName": "test",
    "RunConfiguration": {
        "ApplicationRestoreConfiguration": {
         "ApplicationRestoreType": "RESTORE_FROM_LATEST_SNAPSHOT"
         }
    }
}
```

2. 前述のリクエストを指定して StartApplication アクションを実行し、アプリケーションを起動します。

```
aws kinesisanalyticsv2 start-application --cli-input-json file://start_request.json
```

アプリケーションが実行されます。Amazon CloudWatch コンソールで Managed Service for Apache Flink メトリクスをチェックして、アプリケーションが動作していることを確認できます。

アプリケーションの停止

このセクションでは、StopApplication アクションを使用してアプリケーションを停止します。

アプリケーションを停止するには

1. 次の JSON コードを stop_request.json という名前のファイルに保存します。

```
{
   "ApplicationName": "test"
}
```

2. 次のリクエストを指定して StopApplication アクションを実行し、アプリケーションを停止します。

```
aws kinesisanalyticsv2 stop-application --cli-input-json file://stop_request.json
```

アプリケーションが停止します。

CloudWatch ログ記録オプションを追加する

を使用して AWS CLI 、Amazon CloudWatch ログストリームをアプリケーションに追加できます。アプリケーションで CloudWatch ログを使用する方法については、「」を参照してくださいthe section called “ログ記録の設定”。

環境プロパティを更新する

このセクションでは、「UpdateApplication」アクションを使用して、アプリケーションコードを再コンパイルせずにアプリケーションの環境プロパティを変更します。この例では、ソースストリームおよびレプリケート先ストリームのリージョンを変更します。

アプリケーションの環境プロパティを更新します

1. 次の JSON コードを update_properties_request.json という名前のファイルに保存します。

```
{"ApplicationName": "test",
   "CurrentApplicationVersionId": 1,
   "ApplicationConfigurationUpdate": {
      "EnvironmentPropertyUpdates": {
         "PropertyGroups": [
            {
               "PropertyGroupId": "ProducerConfigProperties",
               "PropertyMap" : {
                    "flink.stream.initpos" : "LATEST",
                    "aws.region" : "us-west-2",
                    "AggregationEnabled" : "false"
               }
            },
            {
               "PropertyGroupId": "ConsumerConfigProperties",
               "PropertyMap" : {
                    "aws.region" : "us-west-2"
               }
            }
         ]
      }
   }
}
```

2. 前述のリクエストでUpdateApplicationアクションを実行し、環境プロパティを更新します。

```
aws kinesisanalyticsv2 update-application --cli-input-json file://
update_properties_request.json
```

アプリケーションコードの更新

アプリケーションコードを新しいバージョンのコードパッケージで更新する必要がある場合は、UpdateApplication AWS CLI アクションを使用します。

を使用するには AWS CLI、Amazon S3 バケットから以前のコードパッケージを削除し、新しいバージョンをアップロードして、同じ Amazon S3 バケットとオブジェクト名UpdateApplicationを指定して を呼び出します。 Amazon S3 アプリケーションは新しいコードパッケージで再起動します。

以下の UpdateApplication アクションのサンプル· リクエストは、アプリケーション· コードを再読み込 み、アプリケーションを再起動します。CurrentApplicationVersionId を現在のアプリケーションバージョンに更新します。ListApplications または DescribeApplication アクションを使用して、現在のアプリケーションバージョンを確認できます。バケット名のサフィックス (「*<username>*」) を、 the section called “2 つの Amazon Kinesis データストリームを作成する” セクションで選択したサフィックスで更新します。

```
{
    "ApplicationName": "test",
    "CurrentApplicationVersionId": 1,
    "ApplicationConfigurationUpdate": {
        "ApplicationCodeConfigurationUpdate": {
            "CodeContentUpdate": {
                "S3ContentLocationUpdate": {
                    "BucketARNUpdate": "arn:aws:s3:::ka-app-code-username",
                    "FileKeyUpdate": "java-getting-started-1.0.jar"
                }
            }
        }
    }
}
```

ステップ 4: リソースをクリーンアップ AWS する

このセクションでは、入門チュートリアルで作成した AWS リソースをクリーンアップする手順について説明します。

このトピックには、次のセクションが含まれています。

- Managed Service for Apache Flink アプリケーションを削除する
- Kinesis データストリームを削除する
- Amazon S3 オブジェクトとバケットを削除する
- IAM リソースを削除する
- CloudWatch リソースを削除する

Managed Service for Apache Flink アプリケーションを削除する

1. Kinesis コンソール (https://console.aws.amazon.com/kinesis) を開きます。
2. Managed Service for Apache Flink パネルで、 を選択しますMyApplication。
3. [設定] を選択します。
4. スナップショットセクションで「無効」を選択して、更新を選択します。
5. アプリケーションのページで[削除]を選択し、削除を確認します。

Kinesis データストリームを削除する

1. https://console.aws.amazon.com/flink で Apache Flink 用 Managed Serviceコンソールを開く
2. Kinesis Data Streams パネルで、 を選択しますExampleInputStream。
3. ExampleInputStream ページで、Kinesis ストリームの削除を選択し、削除を確定します。
4. Kinesis ストリームページで、 を選択しExampleOutputStream、アクション を選択し、削除 を選択し、削除を確定します。

Amazon S3 オブジェクトとバケットを削除する

1. https://console.aws.amazon.com/s3/でAmazon S3 コンソールを開きます。
2. ka-app-code-***<username>*** バケットを選択します。
3. [削除] を選択し、バケット名を入力して削除を確認します。

IAM リソースを削除する

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. ナビゲーションバーで、[ポリシー] を選択します。
3. フィルターコントロールに「kinesis」と入力します。
4. kinesis-analytics-service-MyApplication-us-west-2 ポリシーを選択します。
5. [ポリシーアクション]、[削除] の順に選択します。
6. ナビゲーションバーで [ロール] を選択します。
7. kinesis-analytics-MyApplication-us-west-2 ロールを選択します。
8. [ロールの削除] を選択し、削除を確定します。

CloudWatch リソースを削除する

1. https://console.aws.amazon.com/cloudwatch/ で CloudWatch コンソールを開きます。
2. ナビゲーションバーで [ログ] を選択します。
3. /aws/kinesis-analytics/MyApplication ロググループを選択します。
4. [ロググループの削除]を選択し、削除を確認してください。

Managed Service for Apache Flink の以前のバージョン (レガシー) の例

> **Note**
>
> 現在の例については、「」を参照してください例。

このセクションでは、 Managed Service for Apache Flink 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、Managed Service for Apache Flink アプリケーションの作成と結果のテストに役立つサンプルコードと step-by-step 手順が含まれています。

例に進む前に、以下に目を通しておくことをお勧めします。

- 仕組み
- 開始方法 (DataStream API)

> **Note**
>
> これらの例は、米国西部 (オレゴン) リージョン (us-west-2) を使用していると仮定しています。別のリージョンを使用している場合は、アプリケーションコード、コマンド、IAM ロールを適切に更新してください。

トピック

- DataStream API の例
- Python の例
- Scala の例

DataStream API の例

次の例は、Apache Flink DataStream API を使用してアプリケーションを作成する方法を示しています。

トピック

- 例: タンブリングウィンドウ
- 例: スライディングウィンドウ
- 例: Amazon S3 バケットへの書き込み
- チュートリアル: Managed Service for Apache Flink アプリケーションを使用して、MSK クラスター内の 1 つのトピックから VPC 内の別のトピックにデータをレプリケートする
- 例: Kinesis データストリームで EFO コンシューマーを使用する
- 例: Firehose への書き込み
- 例: 別のアカウントの Kinesis ストリームからの読み取り
- チュートリアル: Amazon MSK でのカスタムトラストストアの使用

例: タンブリングウィンドウ

> **Note**
>
> 現在の例については、「」を参照してください例。

この練習では、タンブリングウィンドウを使用してデータを集約する Managed Service for Apache Flink アプリケーションを作成します。Flink では、集約はデフォルトで有効になっています。 以下を無効にするには次のコマンドを使用します。

```
sink.producer.aggregation-enabled' = 'false'
```

> **Note**
>
> この演習に必要な前提条件を設定するには、まず開始方法 (DataStream API)演習を完了してください。

このトピックには、次のセクションが含まれています。

- 依存リソースを作成する
- サンプルレコードを入力ストリームに書き込む
- アプリケーションコードをダウンロードして調べる
- アプリケーションコードをコンパイルする
- Apache Flink ストリーミング Java コードをアップロードする
- Managed Service for Apache Flink アプリケーションを作成して実行する
- AWS リソースをクリーンアップする

依存リソースを作成する

この練習用の Managed Service for Apache Flink を作成する前に、以下の依存リソースを作成します。

- 2 つの Kinesis Data Streams (`ExampleInputStream`と`ExampleOutputStream`)
- アプリケーションのコードを保存するためのAmazon S3バケット (`ka-app-code-<username>`)

Kinesis ストリームと Amazon S3 バケットは、コンソールを使用して作成できます。これらのリソースの作成手順については、次の各トピックを参照してください。

- 「Amazon Kinesis Data Streamsデベロッパーガイド」の「データストリームの作成および更新」データストリーム**`ExampleInputStream`**と**`ExampleOutputStream`**に名前を付けます。
- Amazon Simple Storage Service ユーザーガイドの「S3 バケットを作成する方法」を参照してください。ログイン名 (**`ka-app-code-<username>`** など) を追加して、Amazon S3 バケットにグローバルに一意の名前を付けます。

サンプルレコードを入力ストリームに書き込む

このセクションでは、Python スクリプトを使用して、アプリケーションが処理するサンプルレコードをストリームに書き込みます。

> **Note**
>
> このセクションでは AWS SDK for Python (Boto) が必要です。

1. 次の内容で、`stock.py` という名前のファイルを作成します。

```
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', region_name='us-west-2'))
```

2. stock.py スクリプトを実行します。

```
$ python stock.py
```

チュートリアルの残りの部分を完了する間、スクリプトを実行し続けてください。

アプリケーションコードをダウンロードして調べる

この例の Java アプリケーションコードは、 から入手できます GitHub。アプリケーションコードをダウンロードするには、次の操作を行います。

1. Git クライアントをまだインストールしていない場合は、インストールします。詳細については、「Git のインストール」をご参照ください。
2. 次のコマンドを使用してリモートリポジトリのクローンを作成します。

```
git clone https://github.com/aws-samples/amazon-kinesis-data-analytics-examples.git
```

3. amazon-kinesis-data-analytics-java-examples/TumblingWindow ディレクトリに移動します。

アプリケーションコードはTumblingWindowStreamingJob.javaファイルに含まれています。アプリケーションコードに関して、以下の点に注意してください。

- アプリケーションは Kinesis ソースを使用して、ソースストリームから読み取りを行います。次のスニペットでは、Kinesis ソースが作成されます。

```
return env.addSource(new FlinkKinesisConsumer<>(inputStreamName,
                new SimpleStringSchema(), inputProperties));
```

- 以下のインポートステートメントを追加します。

```
import
 org.apache.flink.streaming.api.windowing.assigners.TumblingProcessingTimeWindows; //
flink 1.13 onward
```

- このアプリケーションでは、timeWindow演算子を使用して 5 秒間のタンブリングウィンドウにおける各ストックコードのカウント値を求めます。次のコードは演算子を作成し、集約されたデータを新しい Kinesis Data Streamsシンクに送信します。

```
input.flatMap(new Tokenizer()) // Tokenizer for generating words
                    .keyBy(0) // Logically partition the stream for each word

                    .window(TumblingProcessingTimeWindows.of(Time.seconds(5))) //
Flink 1.13 onward
                    .sum(1) // Sum the number of words per partition
                    .map(value -> value.f0 + "," + value.f1.toString() + "\n")
                    .addSink(createSinkFromStaticConfig());
```

アプリケーションコードをコンパイルする

アプリケーションをコンパイルするには、次の操作を行います。

1. Java と Maven がまだインストールされていない場合は、インストールします。詳細については、開始方法 (DataStream API)チュートリアルの「前提条件」を参照してください。
2. 次のコマンドを使用して、アプリケーションをコンパイルします。

```
mvn package -Dflink.version=1.15.3
```

> **Note**
>
> 提供されているソースコードは Java 11 のライブラリーに依存しています。

アプリケーションをコンパイルすると、アプリケーション JAR ファイル (`target/aws-kinesis-analytics-java-apps-1.0.jar`) が作成されます。

Apache Flink ストリーミング Java コードをアップロードする

このセクションでは、依存リソースを作成する のセクションで作成した Amazon S3 バケットにアプリケーションコードをアップロードします。

1. Amazon S3 コンソールで、ka-app-code-***<username>*** バケットを選択し、アップロード を選択します。
2. ファイルの選択のステップで、[ファイルを追加] を選択します。前のステップで作成した `aws-kinesis-analytics-java-apps-1.0.jar` ファイルに移動します。
3. オブジェクトの設定を変更する必要はないので、[アップロード] を選択してください。

アプリケーションコードが Amazon S3 バケットに保存され、アプリケーションからアクセスできるようになります。

Managed Service for Apache Flink アプリケーションを作成して実行する

以下の手順を実行し、コンソールを使用してアプリケーションを作成、設定、更新、および実行します。

アプリケーションの作成

1. https://console.aws.amazon.com/flink で Apache Flink 用 Managed Serviceコンソールを開く
2. Managed Service for Apache Flinkのダッシュボードで、「分析アプリケーションの作成」 を選択します。
3. 「Managed Service for Apache Flink-アプリケーションの作成」ページで、次のようにアプリケーションの詳細を入力します。
   - [アプリケーション名] には **MyApplication** と入力します。
   - [ランタイム] には、[Apache Flink] を選択します。

     > **Note**
     >
     > Apache Flink 用 Managed Serviceは Apache Flink バージョン 1.15.2 を使用しています。

   - バージョンプルダウンは Apache Flink バージョン 1.15.2 (推奨バージョン) のままにしておきます。
4. [アクセス許可] には、[IAM ロールの作成 / 更新**kinesis-analytics-MyApplication-us-west-2**] を選択します。
5. [Create application] を選択します。

> **Note**
>
> コンソールを使用して Apache Flink アプリケーション用 Managed Service を作成する場合は、IAM ロールとポリシーをアプリケーションが自動的に作成するオプションを選択できます。アプリケーションではこのロールとポリシーを使用して、依存リソースにアクセスします。これらの IAM リソースは、次のようにアプリケーション名とリージョンを使用して命名されます。
>
> - ポリシー: `kinesis-analytics-service-MyApplication-us-west-2`
> - ロール: `kinesisanalytics-MyApplication-us-west-2`

IAM ポリシーを編集する

IAM ポリシーを編集し、Kinesis Data Streamsにアクセスするための許可を追加します。

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. [ポリシー] を選択します。前のセクションでコンソールによって作成された **kinesis-analytics-service-MyApplication-us-west-2** ポリシーを選択します。
3. [概要] ページで、[ポリシーの編集] を選択します。[JSON] タブを選択します。
4. 次のポリシー例で強調表示されているセクションをポリシーに追加します。サンプルのアカウント ID (*012345678901*) を自分のアカウント ID に置き換えます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ReadCode",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "logs:DescribeLogGroups",
                "s3:GetObjectVersion"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:*",
                "arn:aws:s3:::ka-app-code-<username>/aws-kinesis-analytics-java-
apps-1.0.jar"
            ]
        },
        {
            "Sid": "DescribeLogStreams",
            "Effect": "Allow",
            "Action": "logs:DescribeLogStreams",
            "Resource": "arn:aws:logs:us-west-2:012345678901:log-group:/aws/
kinesis-analytics/MyApplication:log-stream:*"
        },
        {
            "Sid": "PutLogEvents",
            "Effect": "Allow",
            "Action": "logs:PutLogEvents",
            "Resource": "arn:aws:logs:us-west-2:012345678901:log-group:/aws/
kinesis-analytics/MyApplication:log-stream:kinesis-analytics-log-stream"
        },
        {
            "Sid": "ListCloudwatchLogGroups",
            "Effect": "Allow",
            "Action": [
```

```
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:*"
            ]
        },
        {
            "Sid": "ReadInputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
        },
        {
            "Sid": "WriteOutputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleOutputStream"
        }
    ]
}
```

アプリケーションを設定する

1. MyApplication ページで、 の設定 を選択します。
2. [Configure application] ページで、[Code location] を次のように指定します。
   - [Amazon S3 バケット] で、**ka-app-code-*<username>***と入力します。
   - [Amazon S3 オブジェクトへのパス] で、**aws-kinesis-analytics-java-apps-1.0.jar**と入力します。
3. [Access to application resources] の [Access permissions] では、[Create / update IAM role**kinesis-analytics-MyApplication-us-west-2**] を選択します。
4. [Monitoring] の [Monitoring metrics level] が [Application] に設定されていることを確認します。
5. CloudWatch のログ記録では、有効化チェックボックスをオンにします。
6. [更新] を選択します。

> **Note**
>
> CloudWatch ログ記録を有効にすると、Managed Service for Apache Flink によってロググループとログストリームが作成されます。これらのリソースの名前は次のとおりです。
>
> - ロググループ: `/aws/kinesis-analytics/MyApplication`
> - ログストリーム: `kinesis-analytics-log-stream`
>
> このログストリームはアプリケーションのモニターに使用されます。このログストリームは、アプリケーションの結果の送信に使用されたログストリームとは異なります。

アプリケーションを実行する

1. MyApplication ページで、 の実行 を選択します。「スナップショットなしで実行」オプションを選択したままにして、確定します。
2. アプリケーションが実行されたら、ページを更新します。コンソールには [Application graph] が示されます。

CloudWatch コンソールで Managed Service for Apache Flink メトリクスをチェックして、アプリケーションが動作していることを確認できます。

AWS リソースをクリーンアップする

このセクションでは、「タンブリングウィンドウ」チュートリアルで作成した AWS リソースをクリーンアップする手順について説明します。

このトピックには、次のセクションが含まれています。

- Managed Service for Apache Flink アプリケーションを削除する
- Kinesis データストリームを削除する
- Amazon S3 オブジェクトとバケットを削除する
- IAM リソースを削除する
- CloudWatch リソースを削除する

Managed Service for Apache Flink アプリケーションを削除する

1. https://console.aws.amazon.com/flink で Apache Flink 用 Managed Serviceコンソールを開く
2. Managed Service for Apache Flink パネルで、 を選択しますMyApplication。
3. アプリケーションのページで[削除]を選択し、削除を確認します。

Kinesis データストリームを削除する

1. 「https://console.aws.amazon.com/kinesis」で Kinesis コンソールを開きます。
2. Kinesis Data Streams パネルで、 を選択しますExampleInputStream。
3. ExampleInputStream ページで、Kinesis ストリームの削除を選択し、削除を確定します。
4. Kinesis ストリームページで、 を選択しExampleOutputStream、アクション を選択し、削除 を選択し、削除を確定します。

Amazon S3 オブジェクトとバケットを削除する

1. https://console.aws.amazon.com/s3/でAmazon S3 コンソールを開きます。
2. ka-app-code-***<username>*** バケットを選択します。
3. [削除] を選択し、バケット名を入力して削除を確認します。

IAM リソースを削除する

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. ナビゲーションバーで、[ポリシー] を選択します。
3. フィルターコントロールに「kinesis」と入力します。
4. kinesis-analytics-service-MyApplication-us-west-2 ポリシーを選択します。
5. [ポリシーアクション]、[削除] の順に選択します。
6. ナビゲーションバーで [ロール] を選択します。
7. kinesis-analytics-MyApplication-us-west-2 ロールを選択します。
8. [ロールの削除] を選択し、削除を確定します。

CloudWatch リソースを削除する

1. https://console.aws.amazon.com/cloudwatch/ で CloudWatch コンソールを開きます。

2. ナビゲーションバーで [ログ] を選択します。
3. /aws/kinesis-analytics/MyApplication ロググループを選択します。
4. [ロググループの削除]を選択し、削除を確認してください。

例: スライディングウィンドウ

> **Note**
>
> 現在の例については、「」を参照してください例。

> **Note**
>
> この演習に必要な前提条件を設定するには、まず開始方法 (DataStream API)演習を完了してください。

このトピックには、次のセクションが含まれています。

- 依存リソースを作成する
- サンプルレコードを入力ストリームに書き込む
- アプリケーションコードをダウンロードして調べる
- アプリケーションコードをコンパイルする
- Apache Flink ストリーミング Java コードをアップロードする
- Managed Service for Apache Flink アプリケーションを作成して実行する
- AWS リソースをクリーンアップする

依存リソースを作成する

この練習用の Managed Service for Apache Flink を作成する前に、以下の依存リソースを作成します。

- 2 つの Kinesis Data Streams (`ExampleInputStream`と`ExampleOutputStream`)
- アプリケーションのコードを保存するためのAmazon S3バケット (`ka-app-code-<username>`)

Kinesis ストリームと Amazon S3 バケットは、コンソールを使用して作成できます。これらのリソースの作成手順については、次の各トピックを参照してください。

- 「Amazon Kinesis Data Streamsデベロッパーガイド」の「データストリームの作成および更新」データストリーム**ExampleInputStream**と**ExampleOutputStream**に名前を付けます。
- Amazon Simple Storage Service ユーザーガイドの「S3 バケットを作成する方法」を参照してください。ログイン名 (**ka-app-code-*<username>*** など) を追加して、Amazon S3 バケットにグローバルに一意の名前を付けます。

サンプルレコードを入力ストリームに書き込む

このセクションでは、Python スクリプトを使用して、アプリケーションが処理するサンプルレコードをストリームに書き込みます。

> **Note**
>
> このセクションでは AWS SDK for Python (Boto) が必要です。

1. 次の内容で、stock.py という名前のファイルを作成します。

```
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"))
```

2. stock.py スクリプトを実行します。

```
$ python stock.py
```

チュートリアルの残りの部分を完了する間、スクリプトを実行し続けてください。

アプリケーションコードをダウンロードして調べる

この例の Java アプリケーションコードは、 から入手できます GitHub。アプリケーションコードをダウンロードするには、次の操作を行います。

1. Git クライアントをまだインストールしていない場合は、インストールします。詳細については、「Git のインストール」をご参照ください。
2. 次のコマンドを使用してリモートリポジトリのクローンを作成します。

```
git clone https://github.com/aws-samples/amazon-kinesis-data-analytics-examples.git
```

3. amazon-kinesis-data-analytics-java-examples/SlidingWindow ディレクトリに移動します。

アプリケーションコードはSlidingWindowStreamingJobWithParallelism.javaファイルに含まれています。アプリケーションコードに関して、以下の点に注意してください。

- アプリケーションは Kinesis ソースを使用して、ソースストリームから読み取りを行います。次のスニペットでは、Kinesis ソースが作成されます。

```
return env.addSource(new FlinkKinesisConsumer<>(inputStreamName,
                new SimpleStringSchema(), inputProperties));
```

- このアプリケーションでは、timeWindow演算子を使用して、5 秒ずつスライドする 10 秒のウィンドウで各銘柄コードの最小値を求めます。次のコードは演算子を作成し、集約されたデータを新しい Kinesis Data Streamsシンクに送信します。
- 以下のインポートステートメントを追加します。

```
import
 org.apache.flink.streaming.api.windowing.assigners.TumblingProcessingTimeWindows; //
flink 1.13 onward
```

- このアプリケーションでは、timeWindow演算子を使用して 5 秒間のタンブリングウィンドウにおける各ストックコードのカウント値を求めます。次のコードは演算子を作成し、集約されたデータを新しい Kinesis Data Streamsシンクに送信します。

```
input.flatMap(new Tokenizer()) // Tokenizer for generating words
                .keyBy(0) // Logically partition the stream for each word

  .window(TumblingProcessingTimeWindows.of(Time.seconds(5))) //Flink 1.13 onward
                .sum(1) // Sum the number of words per partition
                .map(value -> value.f0 + "," + value.f1.toString() + "\n")
                .addSink(createSinkFromStaticConfig());
```

アプリケーションコードをコンパイルする

アプリケーションをコンパイルするには、次の操作を行います。

1. Java と Maven がまだインストールされていない場合は、インストールします。詳細については、開始方法 (DataStream API)チュートリアルの「前提条件」を参照してください。
2. 次のコマンドを使用して、アプリケーションをコンパイルします。

```
mvn package -Dflink.version=1.15.3
```

> **Note**
>
> 提供されているソースコードは Java 11 のライブラリーに依存しています。

アプリケーションをコンパイルすると、アプリケーション JAR ファイル (target/aws-kinesis-analytics-java-apps-1.0.jar) が作成されます。

Apache Flink ストリーミング Java コードをアップロードする

このセクションでは、依存リソースを作成する のセクションで作成した Amazon S3 バケットにアプリケーションコードをアップロードします。

1. Amazon S3 コンソールで、ka-app-code-***<username>*** バケットを選択し、アップロードを選択します。
2. ファイルの選択のステップで、[ファイルを追加] を選択します。前のステップで作成した `aws-kinesis-analytics-java-apps-1.0.jar` ファイルに移動します。
3. オブジェクトの設定を変更する必要はないので、[アップロード] を選択してください。

アプリケーションコードが Amazon S3 バケットに保存され、アプリケーションからアクセスできるようになります。

Managed Service for Apache Flink アプリケーションを作成して実行する

以下の手順を実行し、コンソールを使用してアプリケーションを作成、設定、更新、および実行します。

アプリケーションの作成

1. https://console.aws.amazon.com/flink で Apache Flink 用 Managed Serviceコンソールを開く
2. Managed Service for Apache Flinkのダッシュボードで、「分析アプリケーションの作成」 を選択します。
3. 「Managed Service for Apache Flink-アプリケーションの作成」ページで、次のようにアプリケーションの詳細を入力します。
   - [アプリケーション名] には **`MyApplication`** と入力します。
   - [ランタイム] には、[Apache Flink] を選択します。
   - バージョンプルダウンは Apache Flink バージョン 1.15.2 (推奨バージョン) のままにしておきます。
4. [アクセス許可] には、[IAM ロールの作成 / 更新**`kinesis-analytics-MyApplication-us-west-2`**] を選択します。
5. [Create application] を選択します。

> **Note**
>
> コンソールを使用して Apache Flink アプリケーション用 Managed Service を作成する場合は、IAM ロールとポリシーをアプリケーションが自動的に作成するオプションを選択できます。アプリケーションではこのロールとポリシーを使用して、依存リソースにアクセスします。これらの IAM リソースは、次のようにアプリケーション名とリージョンを使用して命名されます。
>
> - ポリシー: `kinesis-analytics-service-MyApplication-us-west-2`
> - ロール: `kinesisanalytics-MyApplication-us-west-2`

IAM ポリシーを編集する

IAM ポリシーを編集し、Kinesis Data Streamsにアクセスするための許可を追加します。

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. [ポリシー] を選択します。前のセクションでコンソールによって作成された **kinesis-analytics-service-MyApplication-us-west-2** ポリシーを選択します。
3. [概要] ページで、[ポリシーの編集] を選択します。[JSON] タブを選択します。
4. 次のポリシー例で強調表示されているセクションをポリシーに追加します。サンプルのアカウント ID (*012345678901*) を自分のアカウント ID に置き換えます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ReadCode",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "logs:DescribeLogGroups",
                "s3:GetObjectVersion"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:*",
                "arn:aws:s3:::ka-app-code-<username>/aws-kinesis-analytics-java-
apps-1.0.jar"
            ]
        },
```

```
        {
            "Sid": "DescribeLogStreams",
            "Effect": "Allow",
            "Action": "logs:DescribeLogStreams",
            "Resource": "arn:aws:logs:us-west-2:012345678901:log-group:/aws/
kinesis-analytics/MyApplication:log-stream:*"
        },
        {
            "Sid": "PutLogEvents",
            "Effect": "Allow",
            "Action": "logs:PutLogEvents",
            "Resource": "arn:aws:logs:us-west-2:012345678901:log-group:/aws/
kinesis-analytics/MyApplication:log-stream:kinesis-analytics-log-stream"
        },
        {
            "Sid": "ListCloudwatchLogGroups",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:*"
            ]
        },
        {
            "Sid": "ReadInputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
        },
        {
            "Sid": "WriteOutputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleOutputStream"
        }
    ]
}
```

アプリケーションを設定する

1. MyApplication ページで、 の設定 を選択します。
2. [Configure application] ページで、[Code location] を次のように指定します。
   - [Amazon S3 バケット] で、**ka-app-code-*<username>***と入力します。
   - [Amazon S3 オブジェクトへのパス] で、**aws-kinesis-analytics-java-apps-1.0.jar**と入力します。
3. [Access to application resources] の [Access permissions] では、[Create / update IAM role**kinesis-analytics-MyApplication-us-west-2**] を選択します。
4. [Monitoring] の [Monitoring metrics level] が [Application] に設定されていることを確認します。
5. CloudWatch のログ記録では、有効化チェックボックスをオンにします。
6. [更新] を選択します。

> **Note**
>
> Amazon CloudWatch ログ記録を有効にすると、Managed Service for Apache Flink によってロググループとログストリームが作成されます。これらのリソースの名前は次のとおりです。
>
> - ロググループ: `/aws/kinesis-analytics/MyApplication`
> - ログストリーム: `kinesis-analytics-log-stream`
>
> このログストリームはアプリケーションのモニターに使用されます。このログストリームは、アプリケーションの結果の送信に使用されたログストリームとは異なります。

アプリケーションの並列処理を設定する

このアプリケーション例では、タスクの並列実行を使用しています。次のアプリケーションコードはmin演算子の並列処理を設定します。

```
.setParallelism(3) // Set parallelism for the min operator
```

アプリケーションの並列処理は、提供された並列処理 (デフォルトは 1) を超えることはできません。アプリケーションの並列処理を増やすには、次の AWS CLI アクションを使用します。

```
aws kinesisanalyticsv2 update-application
      --application-name MyApplication
      --current-application-version-id <VersionId>
      --application-configuration-update "{\"FlinkApplicationConfigurationUpdate
\": { \"ParallelismConfigurationUpdate\": {\"ParallelismUpdate\": 5,
 \"ConfigurationTypeUpdate\": \"CUSTOM\" }}}"
```

DescribeApplication または ListApplicationsアクションをAPI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。

アプリケーションを実行する

Flink ジョブグラフは、アプリケーションを実行し、Apache Flink ダッシュボードを開き、目的の Flink ジョブを選択すると表示できます。

CloudWatch コンソールで Managed Service for Apache Flink メトリクスをチェックして、アプリケーションが動作していることを確認できます。

AWS リソースをクリーンアップする

このセクションでは、スライディングウィンドウチュートリアルで作成した AWS リソースをクリーンアップする手順について説明します。

このトピックには、次のセクションが含まれています。

- Managed Service for Apache Flink アプリケーションを削除する
- Kinesis データストリームを削除する
- Amazon S3 オブジェクトとバケットを削除する
- IAM リソースを削除する
- CloudWatch リソースを削除する

Managed Service for Apache Flink アプリケーションを削除する

1. https://console.aws.amazon.com/flink で Apache Flink 用 Managed Serviceコンソールを開く
2. Managed Service for Apache Flink パネルで、 を選択しますMyApplication。
3. アプリケーションのページで[削除]を選択し、削除を確認します。

Kinesis データストリームを削除する

1. 「https://console.aws.amazon.com/kinesis」で Kinesis コンソールを開きます。
2. Kinesis Data Streams パネルで、 を選択しますExampleInputStream。
3. ExampleInputStream ページで、Kinesis ストリームの削除を選択し、削除を確定します。
4. Kinesis ストリームページで、 を選択しExampleOutputStream、アクション を選択し、削除 を選択し、削除を確定します。

Amazon S3 オブジェクトとバケットを削除する

1. https://console.aws.amazon.com/s3/でAmazon S3 コンソールを開きます。
2. ka-app-code-***<username>*** バケットを選択します。
3. [削除] を選択し、バケット名を入力して削除を確認します。

IAM リソースを削除する

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. ナビゲーションバーで、[ポリシー] を選択します。
3. フィルターコントロールに「kinesis」と入力します。
4. kinesis-analytics-service-MyApplication-us-west-2 ポリシーを選択します。
5. [ポリシーアクション]、[削除] の順に選択します。
6. ナビゲーションバーで [ロール] を選択します。
7. kinesis-analytics-MyApplication-us-west-2 ロールを選択します。
8. [ロールの削除] を選択し、削除を確定します。

CloudWatch リソースを削除する

1. https://console.aws.amazon.com/cloudwatch/ で CloudWatch コンソールを開きます。
2. ナビゲーションバーで [ログ] を選択します。
3. /aws/kinesis-analytics/MyApplication ロググループを選択します。
4. [ロググループの削除]を選択し、削除を確認してください。

例: Amazon S3 バケットへの書き込み

この練習では、Kinesis データストリーム をソースとして、Amazon S3 バケットをシンクとして、Managed Service for Apache Flink を作成します。シンクを使用すると、Amazon S3 コンソール内のアプリケーションの出力を検証できます。

> **Note**
>
> この演習に必要な前提条件を設定するには、まず開始方法 (DataStream API)演習を完了してください。

このトピックには、次のセクションが含まれています。

- 依存リソースを作成する
- サンプルレコードを入力ストリームに書き込む
- アプリケーションコードをダウンロードして調べる
- アプリケーションコードの変更
- アプリケーションコードをコンパイルする
- Apache Flink ストリーミング Java コードをアップロードする
- Managed Service for Apache Flink アプリケーションを作成して実行する
- アプリケーションの出力を確認する
- オプション: ソースとシンクをカスタマイズする
- AWS リソースをクリーンアップする

依存リソースを作成する

このエクササイズで Apache Flink 用 Managed Service を作成する前に、以下の依存リソースを作成します。

- Kinesis データストリーム (`ExampleInputStream`)
- アプリケーションのコードと出力を格納する Amazon S3 バケット (`ka-app-code-<username>`)

> **Note**
>
> Managed Service for Apache Flinkでサーバー側の暗号化が有効になる場合、Managed Service for Apache Flink は Amazon S3 にデータを書き込むことができません。

Kinesis ストリームと Amazon S3 バケットは、コンソールを使用して作成できます。これらのリソースの作成手順については、次の各トピックを参照してください。

- 「Amazon Kinesis Data Streamsデベロッパーガイド」の「データストリームの作成および更新」データストリームに**ExampleInputStream**と名前を付けます。
- Amazon Simple Storage Service ユーザーガイドの「S3 バケットを作成する方法」を参照してください。ログイン名 (**ka-app-code-*<username>*** など) を追加して、Amazon S3 バケットにグローバルに一意の名前を付けます。Amazon S3 バケットで 2 つのフォルダ (**code**と**data**) を作成します。

アプリケーションは、次の CloudWatch リソースがまだ存在しない場合、作成します。

- `/AWS/KinesisAnalytics-java/MyApplication`という名前のロググループ。
- `kinesis-analytics-log-stream` というログストリーム。

サンプルレコードを入力ストリームに書き込む

このセクションでは、Python スクリプトを使用して、アプリケーションが処理するサンプルレコードをストリームに書き込みます。

> **Note**
>
> このセクションでは AWS SDK for Python (Boto) が必要です。

1. 次の内容で、`stock.py` という名前のファイルを作成します。

```
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', region_name='us-west-2'))
```

2. stock.py スクリプトを実行します。

```
$ python stock.py
```

チュートリアルの残りの部分を完了する間、スクリプトを実行し続けてください。

アプリケーションコードをダウンロードして調べる

この例の Java アプリケーションコードは、 から入手できます GitHub。アプリケーションコードをダウンロードするには、次の操作を行います。

1. Git クライアントをまだインストールしていない場合は、インストールします。詳細については、「Git のインストール」をご参照ください。
2. 次のコマンドを使用してリモートリポジトリのクローンを作成します。

```
git clone https://github.com/aws-samples/amazon-kinesis-data-analytics-examples.git
```

3. amazon-kinesis-data-analytics-java-examples/S3Sink ディレクトリに移動します。

アプリケーションコードはS3StreamingSinkJob.javaファイルに含まれています。アプリケーションコードに関して、以下の点に注意してください。

- アプリケーションは Kinesis ソースを使用して、ソースストリームから読み取りを行います。次のスニペットでは、Kinesis ソースが作成されます。

```
return env.addSource(new FlinkKinesisConsumer<>(inputStreamName,
                new SimpleStringSchema(), inputProperties));
```

- 以下のインポートステートメントを追加してください。

```
import
 org.apache.flink.streaming.api.windowing.assigners.TumblingProcessingTimeWindows;
```

- アプリケーションは Apache Flink S3 シンクを使用して Amazon S3 に書き込みます。

  シンクはタンブリングウィンドウでメッセージを読み取り、メッセージを S3 バケットオブジェクトにエンコードし、エンコードされたオブジェクトを S3 シンクに送信します。次のコードは、Amazon S3 に送信するオブジェクトをエンコードします。

```
input.map(value -> { // Parse the JSON
                JsonNode jsonNode = jsonParser.readValue(value, JsonNode.class);
                return new Tuple2<>(jsonNode.get("ticker").toString(), 1);
            }).returns(Types.TUPLE(Types.STRING, Types.INT))
            .keyBy(v -> v.f0) // Logically partition the stream for each word
            .window(TumblingProcessingTimeWindows.of(Time.minutes(1)))
            .sum(1) // Count the appearances by ticker per partition
            .map(value -> value.f0 + " count: " + value.f1.toString() + "\n")
            .addSink(createS3SinkFromStaticConfig());
```

> **Note**
>
> アプリケーションは Flink StreamingFileSink オブジェクトを使用して Amazon S3 に書き込みます。の詳細についてはStreamingFileSink、Apache Flink StreamingFileSink

ドキュメントの「」を参照してください。 https://nightlies.apache.org/flink/flink-docs-release-1.13/

アプリケーションコードの変更

このセクションでは、Amazon S3 バケットに出力を書き込むようにアプリケーションコードを変更します。

アプリケーションの出力場所を指定するように次の行をユーザー名で更新してください。

```
private static final String s3SinkPath = "s3a://ka-app-code-<username>/data";
```

アプリケーションコードをコンパイルする

アプリケーションをコンパイルするには、次の操作を行います。

1. Java と Maven がまだインストールされていない場合は、インストールします。詳細については、開始方法 (DataStream API)チュートリアルの「前提条件」を参照してください。
2. 次のコマンドを使用して、アプリケーションをコンパイルします。

```
mvn package -Dflink.version=1.15.3
```

アプリケーションをコンパイルすると、アプリケーション JAR ファイル (`target/aws-kinesis-analytics-java-apps-1.0.jar`) が作成されます。

> **Note**
> 提供されているソースコードは Java 11 のライブラリーに依存しています。

Apache Flink ストリーミング Java コードをアップロードする

このセクションでは、依存リソースを作成する のセクションで作成した Amazon S3 バケットにアプリケーションコードをアップロードします。

1. Amazon S3 コンソールで、ka-app-code-***<username>*** バケットを選択し、コードフォルダに移動して、アップロードを選択します。

2. ファイルの選択のステップで、[ファイルを追加] を選択します。前のステップで作成した `aws-kinesis-analytics-java-apps-1.0.jar` ファイルに移動します。
3. オブジェクトの設定を変更する必要はないので、[アップロード] を選択してください。

アプリケーションコードが Amazon S3 バケットに保存され、アプリケーションからアクセスできるようになります。

Managed Service for Apache Flink アプリケーションを作成して実行する

以下の手順を実行し、コンソールを使用してアプリケーションを作成、設定、更新、および実行します。

アプリケーションの作成

1. https://console.aws.amazon.com/flink で Apache Flink 用 Managed Serviceコンソールを開く
2. Managed Service for Apache Flinkのダッシュボードで、「分析アプリケーションの作成」 を選択します。
3. 「Managed Service for Apache Flink-アプリケーションの作成」ページで、次のようにアプリケーションの詳細を入力します。
   - [アプリケーション名] には **`MyApplication`** と入力します。
   - [ランタイム] には、[Apache Flink] を選択します。
   - バージョンプルダウンは Apache Flink バージョン 1.15.2 (推奨バージョン) のままにしておきます。
4. [アクセス許可] には、[IAM ロールの作成 / 更新**`kinesis-analytics-MyApplication-us-west-2`**] を選択します。
5. [Create application] を選択します。

> **Note**
>
> コンソールを使用して Apache Flink アプリケーション用 Managed Service を作成する場合は、IAM ロールとポリシーをアプリケーションが自動的に作成するオプションを選択できます。アプリケーションではこのロールとポリシーを使用して、依存リソースにアクセスします。これらの IAM リソースは、次のようにアプリケーション名とリージョンを使用して命名されます。
>
> - [アプリケーション名] には**`MyApplication`**と入力します。

- [ランタイム] には、[Apache Flink] を選択します。
- バージョンはApache Flink バージョン 1.15.2 (推薦バージョン)のままにしておきます。

6. [アクセス許可] には、[IAM ロールの作成 / 更新**kinesis-analytics-MyApplication-us-west-2**] を選択します。
7. [Create application] を選択します。

> **Note**
>
> コンソールを使用して Managed Service for Apache Flink を作成する場合は、IAM ロールとポリシーをアプリケーションが自動的に作成するオプションを選択できます。アプリケーションではこのロールとポリシーを使用して、依存リソースにアクセスします。これらの IAM リソースは、次のようにアプリケーション名とリージョンを使用して命名されます。
>
> - ポリシー: `kinesis-analytics-service-MyApplication-us-west-2`
> - ロール: `kinesisanalytics-MyApplication-us-west-2`

IAM ポリシーを編集する

IAM ポリシーを編集し、Kinesis データストリームにアクセスするための許可を追加します。

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. [ポリシー] を選択します。前のセクションでコンソールによって作成された **kinesis-analytics-service-MyApplication-us-west-2** ポリシーを選択します。
3. [概要] ページで、[ポリシーの編集] を選択します。[JSON] タブを選択します。
4. 次のポリシー例で強調表示されているセクションをポリシーに追加します。サンプルのアカウント ID (*012345678901*) を自分のアカウント ID に置き換えます。<username> を自身のユーザー名に置き換えます。

```
{
            "Sid": "S3",
            "Effect": "Allow",
            "Action": [
                "s3:Abort*",
                "s3:DeleteObject*",
```

```
                "s3:GetObject*",
                "s3:GetBucket*",
                "s3:List*",
                "s3:ListBucket",
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::ka-app-code-<username>",
                "arn:aws:s3:::ka-app-code-<username>/*"
            ]
          },
        {
            "Sid": "ListCloudwatchLogGroups",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "arn:aws:logs:region:account-id:log-group:*"
            ]
        },
        {
            "Sid": "ListCloudwatchLogStreams",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogStreams"
            ],
            "Resource": [
                "arn:aws:logs:region:account-id:log-group:%LOG_GROUP_PLACEHOLDER
%:log-stream:*"
            ]
        },
        {
            "Sid": "PutCloudwatchLogs",
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:region:account-id:log-group:%LOG_GROUP_PLACEHOLDER
%:log-stream:%LOG_STREAM_PLACEHOLDER%"
            ]
        }
        ,
```

```
        {
            "Sid": "ReadInputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
        },

    ]
}
```

アプリケーションを設定する

1. MyApplication ページで、 の設定 を選択します。
2. [Configure application] ページで、[Code location] を次のように指定します。
   - [Amazon S3 バケット] で、**ka-app-code-*<username>***と入力します。
   - [Amazon S3 オブジェクトへのパス] で、**code/aws-kinesis-analytics-java-apps-1.0.jar**と入力します。
3. [Access to application resources] の [Access permissions] では、[Create / update IAM role**kinesis-analytics-MyApplication-us-west-2**] を選択します。
4. [Monitoring] の [Monitoring metrics level] が [Application] に設定されていることを確認します。
5. CloudWatch のログ記録では、有効化チェックボックスをオンにします。
6. [更新] を選択します。

> **Note**
>
> CloudWatch ログ記録を有効にすると、Managed Service for Apache Flink によってロググループとログストリームが作成されます。これらのリソースの名前は次のとおりです。
>
> - ロググループ: `/aws/kinesis-analytics/MyApplication`
> - ログストリーム: `kinesis-analytics-log-stream`
>
> このログストリームはアプリケーションのモニターに使用されます。このログストリームは、アプリケーションの結果の送信に使用されたログストリームとは異なります。

アプリケーションを実行する

1. MyApplication ページで、実行 を選択します。「スナップショットなしで実行」オプションを選択したままにして、確定します。
2. アプリケーションが実行されたら、ページを更新します。コンソールには [Application graph] が示されます。

アプリケーションの出力を確認する

Amazon S3 コンソールで、S3 バケット内のデータフォルダを開きます。

数分後、アプリケーションからの集約データを含むオブジェクトが表示されます。

> **Note**
>
> Flink では、集約はデフォルトで有効になっています。　以下を無効にするには次のコマンドを使用します。
>
> ```
> sink.producer.aggregation-enabled' = 'false'
> ```

オプション: ソースとシンクをカスタマイズする

このセクションでは、ソースオブジェクトおよびシンクオブジェクトの設定をカスタマイズします。

> **Note**
>
> 以下のセクションで説明するコードセクションを変更した後、次の操作を行ってアプリケーションコードをリロードします。
>
> - このthe section called “アプリケーションコードをコンパイルする”セクションの手順を繰り返して、更新したアプリケーションコードをコンパイルします。
> - このthe section called “Apache Flink ストリーミング Java コードをアップロードする”セクションの手順を繰り返して、更新したアプリケーションコードをアップロードします。
> - コンソールのアプリケーションページで 設定 を選択し、更新 を選択して、更新したアプリケーションコードをアプリケーションにリロードします。

このセクションには、次の項目が含まれています。

- データパーティショニングを設定する
- 読み取り頻度を設定する
- 書き込みバッファリングを設定する

データパーティショニングを設定する

このセクションでは、ストリーミングファイルシンクが S3 バケットに作成するフォルダーの名前を設定します。ストリーミングファイルシンクにバケットアサイナーを追加します。

S3 バケットで作成されたフォルダー名をカスタマイズするには、次の操作を行ってください。

1. S3StreamingSinkJob.javaファイルの先頭に次のインポートステートメントを追加してください。

```
import
 org.apache.flink.streaming.api.functions.sink.filesystem.rollingpolicies.DefaultRollingPol
import
 org.apache.flink.streaming.api.functions.sink.filesystem.bucketassigners.DateTimeBucketAss
```

2. 次のようにcreateS3SinkFromStaticConfig()コード内のメソッドを更新します。

```
private static StreamingFileSink<String> createS3SinkFromStaticConfig() {

    final StreamingFileSink<String> sink = StreamingFileSink
        .forRowFormat(new Path(s3SinkPath), new
 SimpleStringEncoder<String>("UTF-8"))
        .withBucketAssigner(new DateTimeBucketAssigner("yyyy-MM-dd--HH"))
        .withRollingPolicy(DefaultRollingPolicy.create().build())
        .build();
    return sink;
}
```

前のコード例では、DateTimeBucketAssignerとカスタム日付形式を使用して S3 バケットにフォルダを作成しました。DateTimeBucketAssignerは現在のシステム時刻を使用してバケット名を作成します。カスタムバケット割り当てを作成して、作成したフォルダ名をさらにカスタマイズする場合は、 を実装するクラスを作成できますBucketAssigner。getBucketIdメソッドを使用してカスタムロジックを実行します。

BucketAssignerのカスタム実装では、Context パラメータを使用してレコードに関する詳細情報を取得して、保存先フォルダを決定することができます。

読み取り頻度を設定する

このセクションでは、ソースストリームの読み取り頻度を設定します。

Kinesis Streams コンシューマは、デフォルトで 1 秒あたり 5 回にソースストリームから読み取りを行います。ストリームからデータを読み取るクライアントが複数ある場合や、アプリケーションがレコードの読み取りを再試行する必要がある場合は、この頻度により問題が発生します。コンシューマの読み取り頻度を設定することで、このような問題を回避することができます。

Kinesis コンシューマの読み取り頻度を設定するには、SHARD_GETRECORDS_INTERVAL_MILLIS設定を行います。

次のコード例では、SHARD_GETRECORDS_INTERVAL_MILLIS設定を 1 秒に設定しています。

```
kinesisConsumerConfig.setProperty(ConsumerConfigConstants.SHARD_GETRECORDS_INTERVAL_MILLIS,
 "1000");
```

書き込みバッファリングを設定する

このセクションでは、書き込み頻度やそのほかのシンクの設定を行います。

デフォルトでは、アプリケーションは 1 分ごとに宛先バケットに書き込みます。この間隔やその他の設定は、DefaultRollingPolicyオブジェクトを設定することで変更できます。

> **Note**
>
> Apache Flink ストリーミングファイルシンクは、アプリケーションがチェックポイントを作成するたびに出力バケットに書き込みます。デフォルトでは、アプリケーションは 1 分ごとにチェックポイントを作成します。S3 シンクの書き込み間隔を延長するには、チェックポイント間隔を延長する必要があります。

DefaultRollingPolicyオブジェクトを設定するには、次の操作を行います。

1. アプリケーションのCheckpointInterval設定を増やしてください。UpdateApplication アクションの次の入力は、チェックポイント間隔を 10 分に設定します。

```
{
   "ApplicationConfigurationUpdate": {
      "FlinkApplicationConfigurationUpdate": {
         "CheckpointConfigurationUpdate": {
            "ConfigurationTypeUpdate" : "CUSTOM",
            "CheckpointIntervalUpdate": 600000
         }
      }
   },
   "ApplicationName": "MyApplication",
   "CurrentApplicationVersionId": 5
}
```

上記のコードを使用するには、現在のアプリケーションバージョンを指定します。ListApplications アクションを使用してアプリケーションバージョンを取得できます。

2. S3StreamingSinkJob.javaファイルの先頭に次のインポートステートメントを追加します。

```
import java.util.concurrent.TimeUnit;
```

3. S3StreamingSinkJob.javaファイル内のcreateS3SinkFromStaticConfigメソッドを以下のように更新します。

```
private static StreamingFileSink<String> createS3SinkFromStaticConfig() {

        final StreamingFileSink<String> sink = StreamingFileSink
                .forRowFormat(new Path(s3SinkPath), new
 SimpleStringEncoder<String>("UTF-8"))
                .withBucketAssigner(new DateTimeBucketAssigner("yyyy-MM-dd--HH"))
                .withRollingPolicy(
                        DefaultRollingPolicy.create()
                            .withRolloverInterval(TimeUnit.MINUTES.toMillis(8))
                            .withInactivityInterval(TimeUnit.MINUTES.toMillis(5))
                            .withMaxPartSize(1024 * 1024 * 1024)
                            .build())
                .build();
        return sink;
    }
```

前述のコード例では、Amazon S3 バケットへの書き込み頻度を 8 分に設定しています。

Apache Flink ストリーミングファイルシンクの詳細については、Apache Flink ドキュメント内のRow-encoded Formatsを参照してください。

AWS リソースをクリーンアップする

このセクションでは、Amazon S3 チュートリアルで作成した AWS リソースをクリーンアップする手順について説明します。

このトピックには、次のセクションが含まれています。

- Managed Service for Apache Flink アプリケーションを削除する
- Kinesis データストリームを削除する
- Amazon S3 オブジェクトとバケットを削除する
- IAM リソースを削除する
- CloudWatch リソースを削除する

Managed Service for Apache Flink アプリケーションを削除する

1. https://console.aws.amazon.com/flink で Apache Flink 用 Managed Serviceコンソールを開く
2. Managed Service for Apache Flink パネルで、 を選択しますMyApplication。
3. アプリケーションページで[削除]を選択し、削除を確定します。

Kinesis データストリームを削除する

1. Kinesis コンソール (https://console.aws.amazon.com/kinesis) を開きます。
2. Kinesis Data Streams パネルで、 を選択しますExampleInputStream。
3. ExampleInputStream ページで、Kinesis ストリームの削除を選択し、削除を確定します。

Amazon S3 オブジェクトとバケットを削除する

1. https://console.aws.amazon.com/s3/でAmazon S3 コンソールを開きます。
2. ka-app-code-***<username>*** バケットを選択します。
3. [削除] を選択し、バケット名を入力して削除を確認します。

IAM リソースを削除する

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。

2. ナビゲーションバーで、[ポリシー] を選択します。
3. フィルターコントロールに「kinesis」と入力します。
4. kinesis-analytics-service-MyApplication-us-west-2 ポリシーを選択します。
5. [ポリシーアクション]、[削除] の順に選択します。
6. ナビゲーションバーで ロールを選択します。
7. kinesis-analytics-MyApplication-us-west-2 ロールを選択します。
8. [ロールの削除] を選択し、削除を確定します。

CloudWatch リソースを削除する

1. https://console.aws.amazon.com/cloudwatch/ で CloudWatch コンソールを開きます。
2. ナビゲーションバーで [ログ] を選択します。
3. /aws/kinesis-analytics/MyApplication ロググループを選択します。
4. [ロググループの削除]を選択し、削除を確認してください。

チュートリアル: Managed Service for Apache Flink アプリケーションを使用して、MSK クラスター内の 1 つのトピックから VPC 内の別のトピックにデータをレプリケートする

> **Note**
>
> 現在の例については、「」を参照してください例。

次のチュートリアルでは、Amazon MSK クラスターと 2 つのトピックを含む Amazon VPC を作成する方法と、ある Amazon MSK トピックから読み取り、別の Amazon MSK トピックに書き込む Managed Service for Apache Flinkを作成する方法を示しています。

> **Note**
>
> この演習に必要な前提条件を設定するには、まず開始方法 (DataStream API)演習を完了してください。

このチュートリアルには、次のセクションが含まれています。

- Amazon MSK クラスターを使用してAmazon VPC を作成します

- アプリケーションコードを作成する
- Apache Flink ストリーミング Java コードをアップロードする
- アプリケーションの作成
- アプリケーションを設定する
- アプリケーションを実行する
- アプリケーションをテストする

Amazon MSK クラスターを使用してAmazon VPC を作成します

Managed Service for Apache Flinkアプリケーションからアクセスするサンプル VPC と Amazon MSK クラスターを作成するには、Amazon MSK の使用開始チュートリアルに従ってください。

チュートリアルを完了する際は、以下の点に注意してください。

- ステップ 3: トピックの作成で、kafka-topics.sh --createコマンドを繰り返してAWSKafkaTutorialTopicDestinationという名前の宛先トピックを作成します。

```
bin/kafka-topics.sh --create --zookeeper ZooKeeperConnectionString --replication-
factor 3 --partitions 1 --topic AWS KafkaTutorialTopicDestination
```

- クラスターのブートストラップサーバーリストを記録します。ブートストラップサーバーのリストは、次のコマンドで取得できます (を MSK クラスターの ARN *ClusterArn*に置き換えます) 。

```
aws kafka get-bootstrap-brokers --region us-west-2 --cluster-arn ClusterArn
{...
    "BootstrapBrokerStringTls": "b-2.awskafkatutorialcluste.t79r6y.c4.kafka.us-
west-2.amazonaws.com:9094,b-1.awskafkatutorialcluste.t79r6y.c4.kafka.us-
west-2.amazonaws.com:9094,b-3.awskafkatutorialcluste.t79r6y.c4.kafka.us-
west-2.amazonaws.com:9094"
}
```

- チュートリアルのステップに従うときは、選択した AWS リージョンをコード、コマンド、コンソールエントリで使用してください。

アプリケーションコードを作成する

このセクションでは、アプリケーション JAR ファイルをダウンロードしてコンパイルします。Java 11 を使用することをお勧めします。

この例の Java アプリケーションコードは、 から入手できます GitHub。アプリケーションコードをダウンロードするには、次の操作を行います。

1. Git クライアントをまだインストールしていない場合は、インストールします。詳細については、「Git のインストール」をご参照ください。
2. 次のコマンドを使用してリモートリポジトリのクローンを作成します。

```
git clone https://github.com/aws-samples/amazon-kinesis-data-analytics-examples.git
```

3. アプリケーションコードは`amazon-kinesis-data-analytics-java-examples/KafkaConnectors/KafkaGettingStartedJob.java`ファイルに含まれています。コードを調べて、Managed Service for Apache Flink アプリケーションコードの構造を了解することができます。
4. コマンドライン Maven ツールまたは優先的な開発環境を使用して JAR ファイルを作成します。コマンドライン Maven ツールを使用して JAR ファイルをコンパイルするには、次のように入力します。

```
mvn package -Dflink.version=1.15.3
```

ビルドが成功する場合、次のファイルが作成されます。

```
target/KafkaGettingStartedJob-1.0.jar
```

> **Note**
>
> 提供されているソースコードは Java 11 のライブラリーに依存しています。開発環境を使用している場合は、

Apache Flink ストリーミング Java コードをアップロードする

このセクションでは、開始方法 (DataStream API) チュートリアルで作成した Amazon S3 バケットにアプリケーションコードをアップロードします。

> **Note**
>
> 入門チュートリアルから Amazon S3 バケットを削除した場合は、the section called “Apache Flink ストリーミング Java コードをアップロードする”手順をもう一度実行してください。

1. Amazon S3 コンソールで、ka-app-code-***<username>*** バケットを選択し、アップロードを選択します。
2. ファイルの選択のステップで、[ファイルを追加] を選択します。前のステップで作成した `KafkaGettingStartedJob-1.0.jar` ファイルに移動します。
3. オブジェクトの設定を変更する必要はないので、[アップロード] を選択してください。

アプリケーションコードが Amazon S3 バケットに保存され、アプリケーションからアクセスできるようになります。

アプリケーションの作成

1. https://console.aws.amazon.com/flink. で Managed Service for Apache Flink コンソールを開きます。
2. Managed Service for Apache Flinkのダッシュボードで、「分析アプリケーションの作成」 を選択します。
3. 「Managed Service for Apache Flink-アプリケーションの作成」ページで、次のようにアプリケーションの詳細を入力します。
   - [アプリケーション名] には **`MyApplication`** と入力します。
   - [ランタイム] には、[Apache Flink 1.15.2] を選択します。
4. [アクセス許可] には、[IAM ロールの作成 / 更新**`kinesis-analytics-MyApplication-us-west-2`**] を選択します。
5. [Create application] を選択します。

> **Note**
>
> コンソールを使用して Apache Flink アプリケーション用 Managed Service を作成する場合は、IAM ロールとポリシーをアプリケーションが自動的に作成するオプションを選択できます。アプリケーションではこのロールとポリシーを使用して、依存リソースにアクセスしま

す。これらの IAM リソースは、次のようにアプリケーション名とリージョンを使用して命名されます。

- ポリシー: `kinesis-analytics-service-MyApplication-us-west-2`
- ロール: `kinesisanalytics-MyApplication-us-west-2`

アプリケーションを設定する

1. MyApplication ページで、 の設定 を選択します。
2. [Configure application] ページで、[Code location] を次のように指定します。
   - [Amazon S3 バケット] で、**ka-app-code-*<username>***と入力します。
   - [Amazon S3 オブジェクトへのパス] で、**KafkaGettingStartedJob-1.0.jar**と入力します。
3. [Access to application resources] の [Access permissions] では、[Create / update IAM role**kinesis-analytics-MyApplication-us-west-2**] を選択します。

> **Note**
>
> コンソール (CloudWatch ログや Amazon VPC など) を使用してアプリケーションリソースを指定すると、コンソールはアプリケーション実行ロールを変更して、それらのリソースへのアクセス許可を付与します。

4. [プロパティ] で [グループの追加]を選択します。以下のプロパティを入力します。

グループ ID	キー	値
KafkaSource	トピック	AWS KafkaTutorialTopic
KafkaSource	bootstrap.servers	###########################
KafkaSource	security.protocol	SSL
KafkaSource	ssl.truststore.location	/usr/lib/jvm/java-11-amazon-corretto/lib/security/cacerts

グループ ID	キー	値
KafkaSource	ssl.truststore.password	changeit

> **Note**
>
> デフォルト証明書の ssl.truststore.password は「changeit」です。デフォルト証明書を使用している場合は、この値を変更する必要はありません。

[グループの追加] をもう一度選択します。以下のプロパティを入力します。

グループ ID	キー	値
KafkaSink	トピック	AWS KafkaTutorialTopic Destination
KafkaSink	bootstrap.servers	###########################
KafkaSink	security.protocol	SSL
KafkaSink	ssl.truststore.location	/usr/lib/jvm/java-11-amazon-corretto/lib/security/cacerts
KafkaSink	ssl.truststore.password	changeit
KafkaSink	transaction.timeout.ms	1,000

アプリケーションコードは上記のアプリケーションプロパティを読み取って、VPC と Amazon MSK クラスターとのやり取りに使用されるソースとシンクを設定します。プロパティ使用の詳細については、「ランタイムプロパティ」を参照してください。

5. スナップショットで 無効 を選択します。これにより、無効なアプリケーション状態データを読み込まずにアプリケーションを簡単に更新できます。
6. [Monitoring] の [Monitoring metrics level] が [Application] に設定されていることを確認します。
7. CloudWatch ロギング で、有効化チェックボックスを選択します。

8. [仮想プライベートクラウド (VPC)] セクションで、アプリケーションに関連する VPC を選択します。アプリケーションが VPC リソースにアクセスするために使用する VPC に関連付けられているサブネットとセキュリティグループを選択します。
9. [更新] を選択します。

> **Note**
>
> CloudWatch ログ記録を有効にすると、Managed Service for Apache Flink によってロググループとログストリームが作成されます。これらのリソースの名前は次のとおりです。
>
> - ロググループ: `/aws/kinesis-analytics/MyApplication`
> - ログストリーム: `kinesis-analytics-log-stream`
>
> このログストリームはアプリケーションのモニターに使用されます。

アプリケーションを実行する

Flink ジョブグラフは、アプリケーションを実行し、Apache Flink ダッシュボードを開き、目的の Flink ジョブを選択すると表示できます。

アプリケーションをテストする

このセクションでは、レコードをソーストピックに書き込みます。アプリケーションはソーストピックからレコードを読み取り、宛先トピックに書き込みます。アプリケーションが動作していることを確認するには、ソーストピックにレコードを書き込み、宛先トピックからレコードを読み取ります。

トピックからレコードの書き込みと読み取りを行うには、「Amazon MSK の使用開始チュートリアルの「ステップ 6: データの生成と利用」の手順に従います。

ターゲットトピックから読み込むには、クラスターへの 二番目の接続で、ソーストピックの代わりに宛先トピック名を使用してください。

```
bin/kafka-console-consumer.sh --bootstrap-server BootstrapBrokerString --
consumer.config client.properties --topic AWS KafkaTutorialTopicDestination --from-
beginning
```

宛先トピックにレコードが表示されない場合は、トラブルシューティングトピックのVPC 内のリソースにアクセスできないセクションを参照してください。

例: Kinesis データストリームで EFO コンシューマーを使用する

> **Note**
>
> 現在の例については、「」を参照してください例。

この演習では、拡張ファンアウト (EFO) コンシューマーを使用して Kinesis データストリームから読み取る Managed Service for Apache Flink アプリケーションを作成します。Kinesis コンシューマーが EFO を使用する場合、Kinesis Data Streams サービスは、コンシューマーがストリームの固定帯域幅を、ストリームから読み取る他のコンシューマーと共有するのではなく、独自の専用帯域幅を提供します。

Kinesis コンシューマーで EFO を使用する方法の詳細については、FLIP-128: Kinesis コンシューマー向けの拡張ファンアウトを参照してください。

この例で作成したアプリケーションは、 AWS Kinesis コネクタ (flink-connector-kinesis) 1.15.3 を使用します。

> **Note**
>
> この演習に必要な前提条件を設定するには、まず開始方法 (DataStream API)演習を完了してください。

このトピックには、次のセクションが含まれています。

- 依存リソースを作成する
- サンプルレコードを入力ストリームに書き込む
- アプリケーションコードをダウンロードして調べる
- アプリケーションコードをコンパイルする
- Apache Flink ストリーミング Java コードをアップロードする
- Managed Service for Apache Flink アプリケーションを作成して実行する
- AWS リソースをクリーンアップする

依存リソースを作成する

この練習用の Managed Service for Apache Flink を作成する前に、以下の依存リソースを作成します。

- 2 つの Kinesis Data Streams (`ExampleInputStream`と`ExampleOutputStream`)
- アプリケーションのコードを保存するためのAmazon S3バケット (`ka-app-code-<username>`)

Kinesis ストリームと Amazon S3 バケットは、コンソールを使用して作成できます。これらのリソースの作成手順については、次の各トピックを参照してください。

- 「Amazon Kinesis Data Streamsデベロッパーガイド」の「データストリームの作成および更新」データストリーム**`ExampleInputStream`**と**`ExampleOutputStream`**に名前を付けます。
- Amazon Simple Storage Service ユーザーガイドの「S3 バケットを作成する方法」を参照してください。ログイン名 (**`ka-app-code-<username>`** など) を追加して、Amazon S3 バケットにグローバルに一意の名前を付けます。

サンプルレコードを入力ストリームに書き込む

このセクションでは、Python スクリプトを使用して、アプリケーションが処理するサンプルレコードをストリームに書き込みます。

> **Note**
>
> このセクションでは AWS SDK for Python (Boto) が必要です。

1. 次の内容で、`stock.py` という名前のファイルを作成します。

```
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', region_name='us-west-2'))
```

2. stock.py スクリプトを実行します。

```
$ python stock.py
```

チュートリアルの残りの部分を完了する間、スクリプトを実行し続けてください。

アプリケーションコードをダウンロードして調べる

この例の Java アプリケーションコードは、 から入手できます GitHub。アプリケーションコードをダウンロードするには、次の操作を行います。

1. Git クライアントをまだインストールしていない場合は、インストールします。詳細については、「Git のインストール」をご参照ください。
2. 次のコマンドを使用してリモートリポジトリのクローンを作成します。

```
git clone https://github.com/aws-samples/amazon-kinesis-data-analytics-examples.git
```

3. amazon-kinesis-data-analytics-java-examples/EfoConsumer ディレクトリに移動します。

アプリケーションコードはEfoApplication.javaファイルに含まれています。アプリケーションコードに関して、以下の点に注意してください。

- EFO コンシューマーを有効にするには、Kinesis コンシューマーで次のパラメータを設定します。
  - RECORD_PUBLISHER_TYPE: アプリケーションが EFO Consumerを使用して Kinesis Data Streamsデータにアクセスできるようにするには、このパラメータを EFO に設定します。
  - EFO_CONSUMER_NAME: このパラメータを、このストリームのコンシューマー間で一意の文字列値に設定します。同じ Kinesis Data Stream でコンシューマー名を再利用すると、その名前を使用していた以前のコンシューマーは終了します。
- 次のコード例は、EFO Consumerを使用してソースストリームから読み取るために、コンシューマー設定プロパティに値を割り当てる方法を示しています。

```
consumerConfig.putIfAbsent(RECORD_PUBLISHER_TYPE, "EFO");
consumerConfig.putIfAbsent(EFO_CONSUMER_NAME, "basic-efo-flink-app");
```

アプリケーションコードをコンパイルする

アプリケーションをコンパイルするには、次の操作を行います。

1. Java と Maven がまだインストールされていない場合は、インストールします。詳細については、開始方法 (DataStream API)チュートリアルの「前提条件」を参照してください。
2. 次のコマンドを使用して、アプリケーションをコンパイルします。

```
mvn package -Dflink.version=1.15.3
```

> **Note**
>
> 提供されているソースコードは Java 11 のライブラリーに依存しています。

アプリケーションをコンパイルすると、アプリケーション JAR ファイル (target/aws-kinesis-analytics-java-apps-1.0.jar) が作成されます。

Apache Flink ストリーミング Java コードをアップロードする

このセクションでは、依存リソースを作成する のセクションで作成した Amazon S3 バケットにアプリケーションコードをアップロードします。

1. Amazon S3 コンソールで、ka-app-code-***<username>*** バケットを選択し、アップロードを選択します。
2. ファイルの選択のステップで、[ファイルを追加] を選択します。前のステップで作成した `aws-kinesis-analytics-java-apps-1.0.jar` ファイルに移動します。
3. オブジェクトの設定を変更する必要はないので、[アップロード] を選択してください。

アプリケーションコードが Amazon S3 バケットに保存され、アプリケーションからアクセスできるようになります。

Managed Service for Apache Flink アプリケーションを作成して実行する

以下の手順を実行し、コンソールを使用してアプリケーションを作成、設定、更新、および実行します。

アプリケーションの作成

1. https://console.aws.amazon.com/flink で Apache Flink 用 Managed Serviceコンソールを開く
2. Managed Service for Apache Flinkのダッシュボードで、「分析アプリケーションの作成」 を選択します。
3. 「Managed Service for Apache Flink-アプリケーションの作成」ページで、次のようにアプリケーションの詳細を入力します。
   - [アプリケーション名] には **`MyApplication`** と入力します。
   - [ランタイム] には、[Apache Flink] を選択します。

     > **Note**
     >
     > Apache Flink 用 Managed Serviceは Apache Flink バージョン 1.15.2 を使用しています。

   - バージョンプルダウンは Apache Flink バージョン 1.15.2 (推奨バージョン) のままにしておきます。
4. [アクセス許可] には、[IAM ロールの作成 / 更新**`kinesis-analytics-MyApplication-us-west-2`**] を選択します。
5. [Create application] を選択します。

> **Note**
>
> コンソールを使用して Apache Flink アプリケーション用 Managed Service を作成する場合は、IAM ロールとポリシーをアプリケーションが自動的に作成するオプションを選択できます。アプリケーションではこのロールとポリシーを使用して、依存リソースにアクセスします。これらの IAM リソースは、次のようにアプリケーション名とリージョンを使用して命名されます。
>
> - ポリシー: `kinesis-analytics-service-MyApplication-us-west-2`
> - ロール: `kinesisanalytics-MyApplication-us-west-2`

IAM ポリシーを編集する

IAM ポリシーを編集し、Kinesis Data Streamsにアクセスするための許可を追加します。

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. [ポリシー] を選択します。前のセクションでコンソールによって作成された **kinesis-analytics-service-MyApplication-us-west-2** ポリシーを選択します。
3. [概要] ページで、[ポリシーの編集] を選択します。[JSON] タブを選択します。
4. 次のポリシー例で強調表示されているセクションをポリシーに追加します。サンプルのアカウント ID (*012345678901*) を自分のアカウント ID に置き換えます。

> **Note**
>
> これらの権限により、アプリケーションには EFO Consumerへのアクセス権限が付与されます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ReadCode",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "logs:DescribeLogGroups",
```

```
                "s3:GetObjectVersion"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:*",
                "arn:aws:s3:::ka-app-code-<username>/aws-kinesis-analytics-java-
apps-1.0.jar"
            ]
        },
        {
            "Sid": "DescribeLogStreams",
            "Effect": "Allow",
            "Action": "logs:DescribeLogStreams",
            "Resource": "arn:aws:logs:us-west-2:012345678901:log-group:/aws/
kinesis-analytics/MyApplication:log-stream:*"
        },
        {
            "Sid": "PutLogEvents",
            "Effect": "Allow",
            "Action": "logs:PutLogEvents",
            "Resource": "arn:aws:logs:us-west-2:012345678901:log-group:/aws/
kinesis-analytics/MyApplication:log-stream:kinesis-analytics-log-stream"
        },
        {
            "Sid": "ListCloudwatchLogGroups",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:*"
            ]
        },
        {
            "Sid": "AllStreams",
            "Effect": "Allow",
            "Action": [
                "kinesis:ListShards",
                "kinesis:ListStreamConsumers",
                "kinesis:DescribeStreamSummary"
            ],
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/*"
        },
        {
            "Sid": "Stream",
```

```
            "Effect": "Allow",
            "Action": [
                "kinesis:DescribeStream",
                "kinesis:RegisterStreamConsumer",
                "kinesis:DeregisterStreamConsumer"
            ],
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
        },
        {
            "Sid": "WriteOutputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleOutputStream"
        },
        {
            "Sid": "Consumer",
            "Effect": "Allow",
            "Action": [
                "kinesis:DescribeStreamConsumer",
                "kinesis:SubscribeToShard"
            ],
            "Resource": [
                "arn:aws:kinesis:us-west-2:012345678901:stream/ExampleInputStream/
consumer/my-efo-flink-app",
                "arn:aws:kinesis:us-west-2:012345678901:stream/ExampleInputStream/
consumer/my-efo-flink-app:*"
            ]
        }
    ]
}
```

アプリケーションを設定する

1. MyApplication ページで、 の設定 を選択します。
2. [Configure application] ページで、[Code location] を次のように指定します。
   - [Amazon S3 バケット] で、**ka-app-code-*<username>***と入力します。
   - [Amazon S3 オブジェクトへのパス] で、**aws-kinesis-analytics-java-apps-1.0.jar**と入力します。

3. [Access to application resources] の [Access permissions] では、[Create / update IAM role**kinesis-analytics-MyApplication-us-west-2**] を選択します。
4. [プロパティ] で [グループの作成]を選択します。
5. 次のアプリケーションのプロパティと値を入力します。

グループ ID	キー	値
ConsumerConfigProperties	**flink.stream.recordpublisher**	**EFO**
ConsumerConfigProperties	**flink.stream.efo.consumername**	**basic-efo-flink-app**
ConsumerConfigProperties	**INPUT_STREAM**	**ExampleInputStream**
ConsumerConfigProperties	**flink.inputstream.initpos**	**LATEST**
ConsumerConfigProperties	**AWS_REGION**	**us-west-2**

6. [プロパティ] で [グループの作成] を選択します。
7. 次のアプリケーションのプロパティと値を入力します。

グループ ID	キー	値
ProducerConfigProperties	**OUTPUT_STREAM**	**ExampleOutputStream**
ProducerConfigProperties	**AWS_REGION**	**us-west-2**
ProducerConfigProperties	**AggregationEnabled**	**false**

8. [Monitoring] の [Monitoring metrics level] が [Application] に設定されていることを確認します。
9. CloudWatch のログ記録では、有効化チェックボックスをオンにします。

10. [更新] を選択します。

> **Note**
>
> CloudWatch ログ記録を有効にすると、Managed Service for Apache Flink によってロググループとログストリームが作成されます。これらのリソースの名前は次のとおりです。
>
> - ロググループ: `/aws/kinesis-analytics/MyApplication`
> - ログストリーム: `kinesis-analytics-log-stream`
>
> このログストリームはアプリケーションのモニターに使用されます。このログストリームは、アプリケーションの結果の送信に使用されたログストリームとは異なります。

アプリケーションを実行する

Flink ジョブグラフは、アプリケーションを実行し、Apache Flink ダッシュボードを開き、目的の Flink ジョブを選択すると表示できます。

CloudWatch コンソールで Managed Service for Apache Flink メトリクスをチェックして、アプリケーションが動作していることを確認できます。

データストリームの拡張ファンアウトタブで、Kinesis Data Streams コンソールでコンシューマーの名前 () を確認することもできますbasic-efo-flink-app。

AWS リソースをクリーンアップする

このセクションでは、efo Window チュートリアルで作成された AWS リソースをクリーンアップする手順について説明します。

このトピックには、次のセクションが含まれています。

- Managed Service for Apache Flink アプリケーションを削除する
- Kinesis データストリームを削除する
- Amazon S3 オブジェクトとバケットの削除
- IAM リソースを削除する
- CloudWatch リソースを削除する

Managed Service for Apache Flink アプリケーションを削除する

1. https://console.aws.amazon.com/flink で Apache Flink 用 Managed Serviceコンソールを開く
2. Managed Service for Apache Flink パネルで、 を選択しますMyApplication。
3. アプリケーションのページで[削除]を選択し、削除を確認します。

Kinesis データストリームを削除する

1. 「https://console.aws.amazon.com/kinesis」で Kinesis コンソールを開きます。
2. Kinesis Data Streams パネルで、 を選択しますExampleInputStream。
3. ExampleInputStream ページで、Kinesis ストリームの削除を選択し、削除を確定します。
4. Kinesis ストリームページで、 を選択しExampleOutputStream、アクション を選択し、削除 を選択し、削除を確定します。

Amazon S3 オブジェクトとバケットの削除

1. https://console.aws.amazon.com/s3/でAmazon S3 コンソールを開きます。
2. ka-app-code-***<username>*** バケットを選択します。
3. [削除] を選択し、バケット名を入力して削除を確認します。

IAM リソースを削除する

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. ナビゲーションバーで、[ポリシー] を選択します。
3. フィルターコントロールに「kinesis」と入力します。
4. kinesis-analytics-service-MyApplication-us-west-2 ポリシーを選択します。
5. [ポリシーアクション]、[削除] の順に選択します。
6. ナビゲーションバーで [ロール] を選択します。
7. kinesis-analytics-MyApplication-us-west-2 ロールを選択します。
8. [ロールの削除] を選択し、削除を確定します。

CloudWatch リソースを削除する

1. https://console.aws.amazon.com/cloudwatch/ で CloudWatch コンソールを開きます。

2. ナビゲーションバーで [ログ] を選択します。
3. /aws/kinesis-analytics/MyApplication ロググループを選択します。
4. [ロググループの削除]を選択し、削除を確認してください。

例: Firehose への書き込み

> **Note**
>
> 現在の例については、「」を参照してください例。

この演習では、ソースとして Kinesis データストリーム、シンクとして Firehose ストリームを持つ Managed Service for Apache Flink アプリケーションを作成します。シンクを使用すると、Amazon S3 バケット内のアプリケーションの出力を検証できます。

> **Note**
>
> この演習に必要な前提条件を設定するには、まず開始方法 (DataStream API)演習を完了してください。

このセクションには、以下のステップが含まれています。

- 依存リソースを作成する
- サンプルレコードを入力ストリームに書き込む
- Apache Flink ストリーミング Java コードをダウンロードして調べる
- アプリケーションコードをコンパイルする
- Apache Flink ストリーミング Java コードをアップロードする
- Managed Service for Apache Flink アプリケーションを作成して実行する
- AWS リソースをクリーンアップする

依存リソースを作成する

このエクササイズで Apache Flink 用 Managed Service を作成する前に、以下の依存リソースを作成します。

- A Kinesis data stream (`ExampleInputStream`)

- アプリケーションが出力を書き込む Firehose ストリーム (`ExampleDeliveryStream`) 。
- アプリケーションのコードを保存するためのAmazon S3バケット (`ka-app-code-<username>`)

コンソールを使用して、Kinesis ストリーム、Amazon S3 バケット、Firehose ストリームを作成できます。これらのリソースの作成手順については、次の各トピックを参照してください。

- 「Amazon Kinesis Data Streamsデベロッパーガイド」の「データストリームの作成および更新」データストリームに**ExampleInputStream**と名前を付けます。
- 「Amazon Data Firehose デベロッパーガイド」の「Amazon Kinesis Data Firehose 配信ストリームの作成」。 Firehose ストリームに という名前を付けます**ExampleDeliveryStream**。Firehose ストリームを作成するときは、Firehose ストリームの S3 送信先 と IAM ロール も作成します。
- Amazon Simple Storage Service ユーザーガイドの「S3 バケットを作成する方法」を参照してください。ログイン名 (**ka-app-code-*<username>*** など) を追加して、Amazon S3 バケットにグローバルに一意の名前を付けます。

サンプルレコードを入力ストリームに書き込む

このセクションでは、Python スクリプトを使用して、アプリケーションが処理するサンプルレコードをストリームに書き込みます。

> **Note**
>
> このセクションでは AWS SDK for Python (Boto) が必要です。

1. 次の内容で、`stock.py` という名前のファイルを作成します。

```
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', region_name='us-west-2'))
```

2. stock.py スクリプトを実行します。

```
$ python stock.py
```

チュートリアルの残りの部分を完了する間、スクリプトを実行し続けてください。

Apache Flink ストリーミング Java コードをダウンロードして調べる

この例の Java アプリケーションコードは、 から入手できます GitHub。アプリケーションコードをダウンロードするには、次の操作を行います。

1. 次のコマンドを使用してリモートリポジトリのクローンを作成します。

```
git clone https://github.com/aws-samples/amazon-kinesis-data-analytics-examples.git
```

2. amazon-kinesis-data-analytics-java-examples/FirehoseSink ディレクトリに移動します。

アプリケーションコードはFirehoseSinkStreamingJob.javaファイルに含まれています。アプリケーションコードに関して、以下の点に注意してください。

- アプリケーションは Kinesis ソースを使用して、ソースストリームから読み取りを行います。次のスニペットでは、Kinesis ソースが作成されます。

```
return env.addSource(new FlinkKinesisConsumer<>(inputStreamName,
                new SimpleStringSchema(), inputProperties));
```

- アプリケーションは Firehose シンクを使用して Firehose ストリームにデータを書き込みます。次のスニペットは Firehose シンクを作成します。

```
private static KinesisFirehoseSink<String> createFirehoseSinkFromStaticConfig() {
        Properties sinkProperties = new Properties();
        sinkProperties.setProperty(AWS_REGION, region);

        return KinesisFirehoseSink.<String>builder()
                .setFirehoseClientProperties(sinkProperties)
                .setSerializationSchema(new SimpleStringSchema())
                .setDeliveryStreamName(outputDeliveryStreamName)
                .build();
    }
```

アプリケーションコードをコンパイルする

アプリケーションをコンパイルするには、次の操作を行います。

1. Java と Maven がまだインストールされていない場合は、インストールします。詳細については、開始方法 (DataStream API)チュートリアルの「前提条件」を参照してください。
2. 次のアプリケーションの Kinesis コネクタを使用するには、Apache Maven をダウンロード、ビルド、インストールする必要があります。詳細については、the section called “以前の Apache Flink バージョンでの Apache Flink Kinesis Streams コネクタの使用” を参照してください。
3. 次のコマンドを使用して、アプリケーションをコンパイルします。

```
mvn package -Dflink.version=1.15.3
```

> **Note**
>
> 提供されているソースコードは Java 11 のライブラリーに依存しています。

アプリケーションをコンパイルすると、アプリケーション JAR ファイル (`target/aws-kinesis-analytics-java-apps-1.0.jar`) が作成されます。

Apache Flink ストリーミング Java コードをアップロードする

このセクションでは、依存リソースを作成する のセクションで作成した Amazon S3 バケットにアプリケーションコードをアップロードします。

アプリケーションコードをアップロードするには

1. https://console.aws.amazon.com/s3/でAmazon S3 コンソールを開きます。
2. コンソールで、ka-app-code-***<username>*** バケットを選択し、アップロードを選択します。
3. ファイルの選択のステップで、[ファイルを追加] を選択します。前のステップで作成した `java-getting-started-1.0.jar` ファイルに移動します。
4. オブジェクトの設定を変更する必要はないので、[アップロード] を選択してください。

アプリケーションコードが Amazon S3 バケットに保存され、アプリケーションからアクセスできるようになります。

Managed Service for Apache Flink アプリケーションを作成して実行する

コンソールまたは AWS CLIのいずれかを使用してManaged Service for Apache Flink を作成し、実行することができます。

> **Note**
>
> コンソールを使用してアプリケーションを作成すると、 AWS Identity and Access Management (IAM) と Amazon CloudWatch Logs リソースが自動的に作成されます。を使用してアプリケーションを作成するときは AWS CLI、これらのリソースを個別に作成します。

トピック

- アプリケーションの作成と実行 (コンソール)
- アプリケーションの作成と実行 (AWS CLI)

アプリケーションの作成と実行 (コンソール)

以下の手順を実行し、コンソールを使用してアプリケーションを作成、設定、更新、および実行します。

アプリケーションの作成

1. https://console.aws.amazon.com/flink で Apache Flink 用 Managed Serviceコンソールを開く
2. Managed Service for Apache Flinkのダッシュボードで、「分析アプリケーションの作成」 を選択します。
3. 「Managed Service for Apache Flink-アプリケーションの作成」ページで、次のようにアプリケーションの詳細を入力します。
   - [アプリケーション名] には **MyApplication** と入力します。
   - [Description (説明)] に **My java test app** と入力します。
   - [ランタイム] には、[Apache Flink] を選択します。

     > **Note**
     >
     > Apache Flink 用 Managed Serviceは Apache Flink バージョン 1.15.2 を使用しています。

   - バージョンプルダウンは Apache Flink バージョン 1.15.2 (推奨バージョン) のままにしておきます。
4. [アクセス許可] には、[IAM ロールの作成 / 更新**kinesis-analytics-MyApplication-us-west-2**] を選択します。
5. [Create application] を選択します。

> **Note**
>
> 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は、IAM ロールとポリシーをアプリケーションが自動的に作成するオプションを選択できます。アプリケーションではこのロールとポリシーを使用して、依存リソースにアクセスします。これらの IAM リソースは、次のようにアプリケーション名とリージョンを使用して命名されます。
>
> - ポリシー: `kinesis-analytics-service-`*`MyApplication`*`-`*`us-west-2`*
> - ロール: `kinesisanalytics-`*`MyApplication`*`-`*`us-west-2`*

IAM ポリシーを編集する

IAM ポリシーを編集して、Kinesis データストリームと Firehose ストリームにアクセスするためのアクセス許可を追加します。

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. [ポリシー] を選択します。前のセクションでコンソールによって作成された **kinesis-analytics-service-MyApplication-us-west-2** ポリシーを選択します。
3. [概要] ページで、[ポリシーの編集] を選択します。[JSON] タブを選択します。
4. 次のポリシー例で強調表示されているセクションをポリシーに追加します。サンプルのアカウント ID (*012345678901*) の全てのインスタンスを自分のアカウント ID に置き換えます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ReadCode",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": [
                "arn:aws:s3:::ka-app-code-username/java-getting-started-1.0.jar"
            ]
        },
        {
            "Sid": "DescribeLogGroups",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:*"
            ]
        },
        {
            "Sid": "DescribeLogStreams",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogStreams"
           ],
```

```
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:*"
            ]
        },
        {
            "Sid": "PutLogEvents",
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:kinesis-analytics-log-stream"
            ]
        },
        {
            "Sid": "ReadInputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
        },
        {
            "Sid": "WriteDeliveryStream",
            "Effect": "Allow",
            "Action": "firehose:*",
            "Resource": "arn:aws:firehose:us-west-2:012345678901:deliverystream/
ExampleDeliveryStream"
        }
    ]
}
```

アプリケーションを設定する

1. MyApplication ページで、 の設定 を選択します。
2. [Configure application] ページで、[Code location] を次のように指定します。
   - [Amazon S3 バケット] で、**ka-app-code-*<username>***と入力します。
   - [Amazon S3 オブジェクトへのパス] で、**java-getting-started-1.0.jar**と入力します。

3. [Access to application resources] の [Access permissions] では、[Create / update IAM role**kinesis-analytics-MyApplication-us-west-2**] を選択します。
4. [Monitoring] の [Monitoring metrics level] が [Application] に設定されていることを確認します。
5. CloudWatch のログ記録では、有効化チェックボックスをオンにします。
6. [更新] を選択します。

> **Note**
>
> CloudWatch ログ記録を有効にすると、Managed Service for Apache Flink によってロググループとログストリームが作成されます。これらのリソースの名前は次のとおりです。
>
> - ロググループ: `/aws/kinesis-analytics/MyApplication`
> - ログストリーム: `kinesis-analytics-log-stream`

アプリケーションを実行する

Flink ジョブグラフは、アプリケーションを実行し、Apache Flink ダッシュボードを開き、目的の Flink ジョブを選択すると表示できます。

アプリケーションの停止

MyApplication ページで、停止 を選択します。アクションを確認します。

アプリケーションの更新

コンソールを使用して、アプリケーションのプロパティ、モニタリング設定、アプリケーション JAR の場所またはファイル名などのアプリケーション設定を更新できます。

MyApplication ページで、 の設定 を選択します。アプリケーションの設定を更新し、[更新] を選択します。

> **Note**
>
> コンソールでアプリケーションのコードを更新するには、JAR のオブジェクト名を変更するか、別の S3 バケットを使用するか、またはthe section called “アプリケーションコードの更新”セクションで説明されている AWS CLI を使用する必要があります。ファイル名またはバ

ケットが変更されない場合、Configure ページで アップデート を選択してもアプリケーションコードはリロードされません。

アプリケーションの作成と実行 (AWS CLI)

このセクションでは、 を使用して Managed Service for Apache Flink アプリケーション AWS CLI を作成して実行します。

許可ポリシーを作成する

まず、2 つのステートメントを含むアクセス許可ポリシーを作成します。1 つは、ソースストリームの read アクションに対するアクセス許可を付与し、もう 1 つはシンクストリームの write アクションに対するアクセス許可を付与します。次に、IAM ロール (次のセクションで作成) にポリシーをアタッチします。そのため、 Managed Service for Apache Flinkがこのロールを引き受けると、ソースストリームからの読み取りとシンクストリームへの書き込みを行うために必要なアクセス許可がサービスに付与されます。

次のコードを使用して AKReadSourceStreamWriteSinkStream アクセス許可ポリシーを作成します。 を Amazon S3 バケットの作成に使用した*#####*に置き換え、アプリケーションコードを保存します。Amazon リソースネーム (ARN) のアカウント ID (*012345678901*) を自分のアカウント ID に置き換えます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "S3",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": ["arn:aws:s3:::ka-app-code-username",
                "arn:aws:s3:::ka-app-code-username/*"
            ]
        },
        {
            "Sid": "ReadInputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
```

```
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
        },
        {
            "Sid": "WriteDeliveryStream",
            "Effect": "Allow",
            "Action": "firehose:*",
            "Resource": "arn:aws:firehose:us-west-2:012345678901:deliverystream/
ExampleDeliveryStream"
        }
    ]
}
```

アクセス許可ポリシーを作成する step-by-step 手順については、IAM ユーザーガイドの「チュートリアル: 最初のカスタマー管理ポリシーの作成とアタッチ」を参照してください。

> **Note**
>
> その他のアマゾンサービスにアクセスするには、 AWS SDK for Javaを使用します。Managed Service for Apache Flink は、アプリケーションに関連付けられているサービス実行 IAM ロールに、SDK が必要とする認証情報を自動的に設定します。追加の手順は必要ありません。

IAM ロールを作成する

このセクションでは、Managed Service for Apache Flink アプリケーションがソースストリームを読み取り、シンクストリームに書き込むために想定できる IAM ロールを作成します。

権限がない場合、Managed Service for Apache Flinkはストリームにアクセスできません。IAM ロールを介してこれらの許可を付与します。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、Managed Service for Apache Flink のロールを引き受けるアクセス許可を Apache Flink に付与します。権限ポリシーは、 Managed Service for Apache Flinkがロールを引き受けた後に実行できる内容を決定します。

前のセクションで作成したアクセス許可ポリシーをこのロールにアタッチします。

IAM ロールを作成するには

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. ナビゲーションペインで [Roles (ロール)]、[Create Role (ロールの作成)] の順に選択します。

3. [信頼されるエンティティの種類を選択] で、[AWS のサービス] を選択します。[このロールを使用するサービスを選択] で、[Kinesis Analytics] を選択します。[ユースケースの選択] で、[Kinesis Analytics] を選択します。

   [Next: Permissions] (次のステップ: 許可) を選択します。

4. [アクセス権限ポリシーをアタッチする] ページで、[Next: Review] (次: 確認) を選択します。ロールを作成した後に、アクセス許可ポリシーをアタッチします。

5. [Create role (ロールの作成)] ページで、ロールの名前に **`MF-stream-rw-role`** を入力します。[ロールの作成] を選択します。

   これで、`MF-stream-rw-role` と呼ばれる新しい IAM ロールが作成されます。次に、ロールの信頼ポリシーとアクセス許可ポリシーを更新します。

6. アクセス許可ポリシーをロールにアタッチします。

   > **Note**
   >
   > この演習では、Managed Service for Apache Flink が、Kinesis データストリーム (ソース) からのデータの読み取りと、別の Kinesis データストリームへの出力の書き込みの両方を実行するためにこのロールを引き受けます。このため、前のステップで作成したポリシー、the section called “許可ポリシーを作成する” をアタッチします。

   a. [概要] ページで、[アクセス許可] タブを選択します。

   b. [Attach Policies (ポリシーのアタッチ)] を選択します。

   c. 検索ボックスに**`AKReadSourceStreamWriteSinkStream`**(前のセクションで作成したポリシー) と入力します。

   d. AKReadSourceStreamWriteSinkStream ポリシーを選択し、ポリシー をアタッチ を選択します。

これで、アプリケーションがリソースにアクセスするために使用するサービスの実行ロールが作成されました。新しいロールの ARN を書き留めておきます。

ロールを作成する step-by-step 手順については、IAM ユーザーガイドの「IAM ロールの作成 (コンソール)」を参照してください。

Managed Service for Apache Flink アプリケーションを作成する

1. 次の JSON コードを create_request.json という名前のファイルに保存します。サンプルロールの ARN を、前に作成したロールの ARN に置き換えます。バケット ARN のサフィックスを、前のthe section called “依存リソースを作成する”セクションka-app-code-*<username>*で選択したサフィックスに置き換えます。サービス実行ロールのサンプルのアカウント ID (*012345678901*) を、自分のアカウント ID に置き換えます。

```
{
    "ApplicationName": "test",
    "ApplicationDescription": "my java test app",
    "RuntimeEnvironment": "FLINK-1_15",
    "ServiceExecutionRole": "arn:aws:iam::012345678901:role/MF-stream-rw-role",
    "ApplicationConfiguration": {
        "ApplicationCodeConfiguration": {
            "CodeContent": {
                "S3ContentLocation": {
                    "BucketARN": "arn:aws:s3:::ka-app-code-username",
                    "FileKey": "java-getting-started-1.0.jar"
                }
            },
            "CodeContentType": "ZIPFILE"
        }
      }
    }
}
```

2. 前述のリクエストを指定して CreateApplication アクションを実行し、アプリケーションを作成します。

```
aws kinesisanalyticsv2 create-application --cli-input-json file://
create_request.json
```

これでアプリケーションが作成されました。次のステップでは、アプリケーションを起動します。

アプリケーションを起動する

このセクションでは、StartApplication アクションを使用してアプリケーションを起動します。

アプリケーションを起動するには

1. 次の JSON コードを start_request.json という名前のファイルに保存します。

```
{
    "ApplicationName": "test",
    "RunConfiguration": {
        "ApplicationRestoreConfiguration": {
         "ApplicationRestoreType": "RESTORE_FROM_LATEST_SNAPSHOT"
         }
    }
}
```

2. 前述のリクエストを指定して StartApplication アクションを実行し、アプリケーションを起動します。

```
aws kinesisanalyticsv2 start-application --cli-input-json file://start_request.json
```

アプリケーションが実行されます。Amazon CloudWatch コンソールで Managed Service for Apache Flink メトリクスをチェックして、アプリケーションが動作していることを確認できます。

アプリケーションの停止

このセクションでは、StopApplication アクションを使用してアプリケーションを停止します。

アプリケーションを停止するには

1. 次の JSON コードを stop_request.json という名前のファイルに保存します。

```

{
    "ApplicationName": "test"
}
```

2. 次のリクエストを指定して StopApplication アクションを実行し、アプリケーションを停止します。

```
aws kinesisanalyticsv2 stop-application --cli-input-json file://stop_request.json
```

アプリケーションが停止します。

CloudWatch ログ記録オプションを追加する

を使用して AWS CLI 、Amazon CloudWatch ログストリームをアプリケーションに追加できます。アプリケーションで CloudWatch ログを使用する方法については、「」を参照してくださいthe section called “ログ記録の設定”。

アプリケーションコードの更新

アプリケーションコードを新しいバージョンのコードパッケージで更新する必要がある場合は、`UpdateApplication` AWS CLI アクションを使用します。

を使用するには AWS CLI、Amazon S3 バケットから以前のコードパッケージを削除し、新しいバージョンをアップロードして、同じ Amazon S3 バケットとオブジェクト名`UpdateApplication`を指定して を呼び出します。 Amazon S3

以下の `UpdateApplication` アクションのサンプル· リクエストは、アプリケーション· コードを再読み込 み、アプリケーションを再起動します。`CurrentApplicationVersionId` を現在のアプリケーションバージョンに更新します。`ListApplications` または `DescribeApplication` アクションを使用して、現在のアプリケーションバージョンを確認できます。バケット名のサフィックス (「*<username>*」) を、 the section called “依存リソースを作成する” セクションで選択したサフィックスで更新します。

```
{
    "ApplicationName": "test",
    "CurrentApplicationVersionId": 1,
    "ApplicationConfigurationUpdate": {
        "ApplicationCodeConfigurationUpdate": {
            "CodeContentUpdate": {
                "S3ContentLocationUpdate": {
                    "BucketARNUpdate": "arn:aws:s3:::ka-app-code-username",
                    "FileKeyUpdate": "java-getting-started-1.0.jar"
                }
            }
        }
    }
}
```

AWS リソースをクリーンアップする

このセクションでは、入門チュートリアルで作成した AWS リソースをクリーンアップする手順について説明します。

このトピックには、次のセクションが含まれています。

- Managed Service for Apache Flink アプリケーションを削除する
- Kinesis データストリームを削除する
- Firehose ストリームを削除する
- Amazon S3 オブジェクトとバケットを削除する
- IAM リソースを削除する
- CloudWatch リソースを削除する

Managed Service for Apache Flink アプリケーションを削除する

1. https://console.aws.amazon.com/flink で Apache Flink 用 Managed Serviceコンソールを開く
2. Managed Service for Apache Flink パネルで、 を選択しますMyApplication。
3. [設定] を選択します。
4. スナップショットセクションで「無効」を選択して、更新を選択します。
5. アプリケーションのページで[削除]を選択し、削除を確認します。

Kinesis データストリームを削除する

1. Kinesis コンソール (https://console.aws.amazon.com/kinesis) を開きます。
2. Kinesis Data Streams パネルで、 を選択しますExampleInputStream。
3. ExampleInputStream ページで、「Kinesis ストリームの削除」を選択し、削除を確定します。

Firehose ストリームを削除する

1. Kinesis コンソール (https://console.aws.amazon.com/kinesis) を開きます。
2. Firehose パネルで、 を選択しますExampleDeliveryStream。
3. ExampleDeliveryStream ページで、Firehose ストリームの削除を選択し、削除を確定します。

Amazon S3 オブジェクトとバケットを削除する

1. https://console.aws.amazon.com/s3/でAmazon S3 コンソールを開きます。
2. ka-app-code-***<username>*** バケットを選択します。

3. [削除] を選択し、バケット名を入力して削除を確認します。
4. Firehose ストリームの送信先に Amazon S3 バケットを作成した場合は、そのバケットも削除します。

IAM リソースを削除する

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. ナビゲーションバーで、[ポリシー] を選択します。
3. フィルターコントロールに「kinesis」と入力します。
4. kinesis-analytics-service-MyApplication-us-west-2 ポリシーを選択します。
5. [ポリシーアクション]、[削除] の順に選択します。
6. Firehose ストリームの新しいポリシーを作成した場合は、そのポリシーも削除します。
7. ナビゲーションバーで [ロール] を選択します。
8. kinesis-analytics-MyApplication-us-west-2 ロールを選択します。
9. [ロールの削除] を選択し、削除を確定します。
10. Firehose ストリーム用に新しいロールを作成した場合は、そのロールも削除します。

CloudWatch リソースを削除する

1. https://console.aws.amazon.com/cloudwatch/ で CloudWatch コンソールを開きます。
2. ナビゲーションバーで [ログ] を選択します。
3. /aws/kinesis-analytics/MyApplication ロググループを選択します。
4. [ロググループの削除]を選択し、削除を確認してください。

例: 別のアカウントの Kinesis ストリームからの読み取り

> **Note**
>
> 現在の例については、「」を参照してください例。

この例は、別のアカウントの Kinesis ストリームからデータを読み取る Managed Service for Apache Flinkアプリケーションを作成する方法を示しています。この例では、1 つのアカウントをソース

Kinesis ストリームに使用し、もう 1 つのアカウントを Managed Service for Apache Flinkとsink Kinesis streamに使用します。

このトピックには、次のセクションが含まれています。

- 前提条件
- セットアップ
- ソース Kinesis ストリームを作成する
- IAM ロールとポリシーの作成と更新
- Python スクリプトを更新する
- Java アプリケーションを更新する
- アプリケーションを構築、アップロード、実行する

前提条件

- このチュートリアルでは、「Getting Started」の例を変更して、別のアカウントの Kinesis ストリームからデータを読み取ります。続行する前に開始方法 (DataStream API)チュートリアルを完了してください。
- このチュートリアルを完了するには、2 つの AWS アカウントが必要です。1 つはソースストリーム用、もう 1 つはアプリケーションとシンクストリーム用です。アプリケーションとシンクストリーム AWS の入門チュートリアルに使用したアカウントを使用します。ソースストリームには別の AWS アカウントを使用してください。

セットアップ

名前付きプロファイルを使用して 2 つの AWS アカウントにアクセスします。 AWS 認証情報と設定ファイルを変更して、2 つのアカウントのリージョンと接続情報を含む 2 つのプロファイルを含めます。

次の認証情報ファイルの例には、ka-source-stream-account-profileとka-sink-stream-account-profile2つの名前付きプロファイルが含まれています。シンクストリームアカウントには、入門チュートリアルで使用したアカウントを使用してください。

```
[ka-source-stream-account-profile]
aws_access_key_id=AKIAIOSFODNN7EXAMPLE
aws_secret_access_key=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY

[ka-sink-stream-account-profile]
```

```
aws_access_key_id=AKIAI44QH8DHBEXAMPLE
aws_secret_access_key=je7MtGbClwBF/2Zp9Utk/h3yCo8nvbEXAMPLEKEY
```

次の設定ファイルの例には、地域と出力形式の情報を含む同じ名前付きのプロファイルが含まれています。

```
[profile ka-source-stream-account-profile]
region=us-west-2
output=json

[profile ka-sink-stream-account-profile]
region=us-west-2
output=json
```

> **Note**
>
> このチュートリアルは`ka-sink-stream-account-profile`を使用しません。プロファイルを使用して 2 つの異なる AWS アカウントにアクセスする方法の例として含まれています。

で名前付きプロファイルを使用する方法の詳細については AWS CLI、 AWS Command Line Interfaceドキュメントの「名前付きプロファイル」を参照してください。

ソース Kinesis ストリームを作成する

このセクションでは、ソースアカウントに Kinesis ストリームを作成します。

次のコマンドを入力して、アプリケーションの入力に使用されたKinesis ストリームを作成します。この`--profile`パラメーターは、使用するアカウントプロファイルを指定することに注意してください。

```
$ aws kinesis create-stream \
--stream-name SourceAccountExampleInputStream \
--shard-count 1 \
--profile ka-source-stream-account-profile
```

IAM ロールとポリシーの作成と更新

AWS アカウント間でオブジェクトへのアクセスを許可するには、ソースアカウントに IAM ロールとポリシーを作成します。次に、シンクアカウントの IAM ポリシーを変更します。IAM ロールとポリ

シーの作成と管理の詳細については、AWS Identity and Access Management ユーザーガイドの次のトピックを参照してください。

- IAM; ロールの作成
- IAM ポリシーの作成

シンクアカウントのロールとポリシー

1. 入門チュートリアルからkinesis-analytics-service-MyApplication-us-west-2ポリシーを編集します。このポリシーにより、ソース ストリームを読み取るためにソースアカウントのロールを引き受けることができます。

> **Note**
>
> 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kinesis-analytics-service-*<application name>*-*<application region>*というポリシーとkinesisanalytics-*<application name>*-*<application region>*というロールを作成します。

以下の強調表示されたセクションをポリシーに追加します。サンプルアカウント ID (*SOURCE01234567*) をソースストリームに使用するアカウントID に置き換えます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AssumeRoleInSourceAccount",
            "Effect": "Allow",
            "Action": "sts:AssumeRole",
            "Resource": "arn:aws:iam::SOURCE01234567:role/KA-Source-Stream-Role"
        },
        {
            "Sid": "ReadCode",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": [
```

```
                "arn:aws:s3:::ka-app-code-username/aws-kinesis-analytics-java-
apps-1.0.jar"
            ]
        },
        {
            "Sid": "ListCloudwatchLogGroups",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:SINK012345678:log-group:*"
            ]
        },
        {
            "Sid": "ListCloudwatchLogStreams",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogStreams"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:SINK012345678:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:*"
            ]
        },
        {
            "Sid": "PutCloudwatchLogs",
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:SINK012345678:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:kinesis-analytics-log-stream"
            ]
        }
    ]
}
```

2. サービスロールの Amazon リソースネーム (ARN) をメモしておきます。これは次のセクションで必要になります。[the IAM role ARN number (IAM ロールの ARN 番号)] は、次のようになります。

```
arn:aws:iam::SINK012345678:role/service-role/kinesis-analytics-MyApplication-us-
west-2
```

ソースアカウントのロールとポリシー

1. KA-Source-Stream-Policyというソースアカウントにポリシーを作成します。このポリシーでは、次の形式を使用します。サンプルアカウント番号をソースアカウントのアカウント番号に置き換えます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ReadInputStream",
            "Effect": "Allow",
            "Action": [
                "kinesis:DescribeStream",
                "kinesis:GetRecords",
                "kinesis:GetShardIterator",
                "kinesis:ListShards"
            ],
            "Resource":
               "arn:aws:kinesis:us-west-2:SOURCE123456784:stream/
SourceAccountExampleInputStream"
        }
    ]
}
```

2. MF-Source-Stream-Roleという名前のソースアカウントでロールを作成します。Managed Flinkユースケースを使用してロールを作成するには、次の手順を実行します。

    1. IAM 管理コンソールで、ロールの作成を選択します。
    2. [Create role] (ロールの作成) ページで、AWS [AWS Service] (AWS サービス) を選択します。サービスリストで Kinesisを選択します。
    3. ユースケースの選択 セクションで、Managed Service for Apache Flinkを選択します。
    4. [Next: Permissions] (次のステップ: 許可) を選択します。
    5. 次の手順を実行して、前のステップで作成した KA-Source-Stream-Policy 許可ポリシーを追加します。[Next:Tags] (次のステップ: タグ) を選択します。

6. [次へ: レビュー] を選択します。
7. ロールに KA-Source-Stream-Role という名前を付けます。アプリケーションは、 へのアクセスにこのロールを使用します。

3. シンクアカウントのkinesis-analytics-MyApplication-us-west-2ARNをソースアカウントのKA-Source-Stream-Role ロールの信頼関係に追加します。

1. IAM コンソールの KA-Source-Stream-Role を開きます。
2. [Trust Relationships] タブを選択します。
3. [Edit trust relationship (信頼関係の編集)] を選択します。
4. 信頼関係には、次のコードを使用します。サンプルのアカウント ID (*SINK012345678*) を自分のアカウント ID に置き換えます。

```
{
    "Version": "2012-10-17",
    "Statement": [
      {
        "Effect": "Allow",
        "Principal": {
          "AWS": "arn:aws:iam::SINK012345678:role/service-role/kinesis-analytics-
MyApplication-us-west-2"
        },
        "Action": "sts:AssumeRole"
      }
    ]
}
```

Python スクリプトを更新する

このセクションでは、ソースアカウントプロファイルを使用するようにサンプルデータを生成する Python scriptを更新します。

stock.py以下に強調表示された変更でスクリプトを更新します。

```
import json
import boto3
import random
import datetime
import os
```

```
os.environ['AWS_PROFILE'] ='ka-source-stream-account-profile'
os.environ['AWS_DEFAULT_REGION'] = 'us-west-2'


kinesis = boto3.client('kinesis')
def getReferrer():
    data = {}
    now = datetime.datetime.now()
    str_now = now.isoformat()
    data['event_time'] = str_now
    data['ticker'] = random.choice(['AAPL', 'AMZN', 'MSFT', 'INTC', 'TBV'])
    price = random.random() * 100
    data['price'] = round(price, 2)
    return data

while True:
        data = json.dumps(getReferrer())
        print(data)
        kinesis.put_record(
                StreamName="SourceAccountExampleInputStream",
                Data=data,
                PartitionKey="partitionkey")
```

Java アプリケーションを更新する

このセクションでは、ソースストリームから読み取る時に、ソースアカウントロールを引き受けるように Java アプリケーションコードを更新します。

BasicStreamingJob.javaファイルに以下の変更を加えます。サンプルソースアカウント番号 (*SOURCE01234567*) をソースアカウント番号に置き換えてください。

```
package com.amazonaws.services.managed-flink;

import com.amazonaws.services.managed-flink.runtime.KinesisAnalyticsRuntime;
import org.apache.flink.api.common.serialization.SimpleStringSchema;
import org.apache.flink.streaming.api.datastream.DataStream;
import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
import org.apache.flink.streaming.connectors.kinesis.FlinkKinesisConsumer;
import org.apache.flink.streaming.connectors.kinesis.FlinkKinesisProducer;
import org.apache.flink.streaming.connectors.kinesis.config.ConsumerConfigConstants;
import org.apache.flink.streaming.connectors.kinesis.config.AWSConfigConstants;

import java.io.IOException;
```

```
import java.util.Map;
import java.util.Properties;

 /**
 * A basic Managed Service for Apache Flink for Java application with Kinesis data
 streams
 * as source and sink.
 */
public class BasicStreamingJob {
    private static final String region = "us-west-2";
    private static final String inputStreamName = "SourceAccountExampleInputStream";
    private static final String outputStreamName = ExampleOutputStream;
    private static final String roleArn = "arn:aws:iam::SOURCE01234567:role/KA-Source-
Stream-Role";
    private static final String roleSessionName = "ksassumedrolesession";

    private static DataStream<String>
 createSourceFromStaticConfig(StreamExecutionEnvironment env) {
        Properties inputProperties = new Properties();
        inputProperties.setProperty(AWSConfigConstants.AWS_CREDENTIALS_PROVIDER,
 "ASSUME_ROLE");
        inputProperties.setProperty(AWSConfigConstants.AWS_ROLE_ARN, roleArn);
        inputProperties.setProperty(AWSConfigConstants.AWS_ROLE_SESSION_NAME,
 roleSessionName);
        inputProperties.setProperty(ConsumerConfigConstants.AWS_REGION, region);
        inputProperties.setProperty(ConsumerConfigConstants.STREAM_INITIAL_POSITION,
 "LATEST");

        return env.addSource(new FlinkKinesisConsumer<>(inputStreamName, new
 SimpleStringSchema(), inputProperties));
    }

    private static KinesisStreamsSink<String> createSinkFromStaticConfig() {
        Properties outputProperties = new Properties();
        outputProperties.setProperty(AWSConfigConstants.AWS_REGION, region);

        return KinesisStreamsSink.<String>builder()
                .setKinesisClientProperties(outputProperties)
                .setSerializationSchema(new SimpleStringSchema())
                .setStreamName(outputProperties.getProperty("OUTPUT_STREAM",
 "ExampleOutputStream"))
                .setPartitionKeyGenerator(element ->
 String.valueOf(element.hashCode()))
                .build();
```

```
    }

    public static void main(String[] args) throws Exception {
        // set up the streaming execution environment
        final StreamExecutionEnvironment env =
 StreamExecutionEnvironment.getExecutionEnvironment();

        DataStream<String> input = createSourceFromStaticConfig(env);

        input.addSink(createSinkFromStaticConfig());

        env.execute("Flink Streaming Java API Skeleton");
    }
}
```

アプリケーションを構築、アップロード、実行する

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. pom.xmlファイルのあるディレクトリで次のコマンドを実行して、アプリケーションを再構築します。

   ```
   mvn package -Dflink.version=1.15.3
   ```

2. Amazon Simple Storage Service (Amazon S3) バケットから以前の JAR ファイルを削除して、S3 バケットに新しいaws-kinesis-analytics-java-apps-1.0.jarファイルをアップロードします。
3. Managed Service for Apache Flink consoleコンソールのアプリケーションページで、設定、更新を選択してアプリケーション JAR ファイルをリロードします。
4. stock.pyスクリプトを実行してソースストリームにデータを送信します。

   ```
   python stock.py
   ```

アプリケーションは別のアカウントの Kinesis ストリームからデータを読み取ります。

ExampleOutputStream コンソールで PutRecords.Bytes メトリックスをチェックして、アプリケーションが動作していることを確認します。出力ストリームに活動があれば、アプリケーションは正常に運行しています。

チュートリアル: Amazon MSK でのカスタムトラストストアの使用

> **Note**
>
> 現在の例については、「」を参照してください例。

現在のデータソース API

現在のデータソース APIs を使用している場合、アプリケーションはここで説明する Amazon MSK Config Providers ユーティリティを活用できます。これにより、 KafkaSource 関数は Amazon S3 の相互 TLS のキーストアと信頼ストアにアクセスできます。

```
...
// define names of config providers:
builder.setProperty("config.providers", "secretsmanager,s3import");

// provide implementation classes for each provider:
builder.setProperty("config.providers.secretsmanager.class",
 "com.amazonaws.kafka.config.providers.SecretsManagerConfigProvider");
builder.setProperty("config.providers.s3import.class",
 "com.amazonaws.kafka.config.providers.S3ImportConfigProvider");

String region = appProperties.get(Helpers.S3_BUCKET_REGION_KEY).toString();
String keystoreS3Bucket = appProperties.get(Helpers.KEYSTORE_S3_BUCKET_KEY).toString();
String keystoreS3Path = appProperties.get(Helpers.KEYSTORE_S3_PATH_KEY).toString();
String truststoreS3Bucket =
 appProperties.get(Helpers.TRUSTSTORE_S3_BUCKET_KEY).toString();
String truststoreS3Path = appProperties.get(Helpers.TRUSTSTORE_S3_PATH_KEY).toString();
String keystorePassSecret =
 appProperties.get(Helpers.KEYSTORE_PASS_SECRET_KEY).toString();
String keystorePassSecretField =
 appProperties.get(Helpers.KEYSTORE_PASS_SECRET_FIELD_KEY).toString();

// region, etc..
builder.setProperty("config.providers.s3import.param.region", region);

// properties
builder.setProperty("ssl.truststore.location", "${s3import:" + region + ":" +
 truststoreS3Bucket + "/" + truststoreS3Path + "}");
builder.setProperty("ssl.keystore.type", "PKCS12");
builder.setProperty("ssl.keystore.location", "${s3import:" + region + ":" +
 keystoreS3Bucket + "/" + keystoreS3Path + "}");
```

```
builder.setProperty("ssl.keystore.password", "${secretsmanager:" + keystorePassSecret +
 ":" + keystorePassSecretField + "}");
builder.setProperty("ssl.key.password", "${secretsmanager:" + keystorePassSecret + ":"
 + keystorePassSecretField + "}");
...
```

詳細とウォークスルーについては、こちらをご覧ください。

レガシー SourceFunction APIs

レガシー SourceFunction APIs を使用している場合、アプリケーションはカスタム信頼ストアをロードするために openメソッドを上書きするカスタムシリアル化スキーマと逆シリアル化スキーマを使用します。これにより、アプリケーションが再起動したり、スレッドを置き換えたりした後でも、アプリケーションでトラストストアを使用できるようになります。

カスタムトラストストアは、次のコードを使用して取得および保存されます。

```
public static void initializeKafkaTruststore() {
    ClassLoader classLoader = Thread.currentThread().getContextClassLoader();
    URL inputUrl = classLoader.getResource("kafka.client.truststore.jks");
    File dest = new File("/tmp/kafka.client.truststore.jks");

    try {
        FileUtils.copyURLToFile(inputUrl, dest);
    } catch (Exception ex) {
        throw new FlinkRuntimeException("Failed to initialize Kakfa truststore", ex);
    }
}
```

> **Note**
>
> Apache Flink では、トラストストアがJKS 形式である必要があります。

> **Note**
>
> この開始方法 (DataStream API)演習に必要な前提条件を設定するには、まず演習を完了してください。

次のチュートリアルでは、カスタム、プライベート、またはセルフホストの認証局 (CA) が発行したサーバー証明書を使用する Kafka クラスターに安全に接続する (転送中の暗号化) 方法を示しています。

Kafka クライアントを TLS 経由で Kafka クラスターに安全に接続するには、Kafka クライアント (Flink アプリケーションの例など) は、Kafka クラスターのサーバー証明書 (発行 CA からルートレベル CA まで) によって提示される完全な信頼チェーンを信頼する必要があります。カスタムトラストストアの例として、相互 TLS (MTLS) 認証が有効になっている Amazon MSK クラスターを使用します。これは、MSK クラスターノードが AWS 、 Certificate Manager Private Certificate Authority (ACM Private CA) によって発行されたサーバー証明書を使用することを意味します。この証明書は、アカウントとリージョンに対してプライベートであるため、Flink アプリケーションを実行する Java 仮想マシン (JVM) のデフォルトの信頼ストアでは信頼されません。

> **Note**
>
> - キーストアは、検証のためにアプリケーションがサーバーまたはクライアントの両方に提示する必要があるプライベートキーと ID 証明書を保存するために使用されます。
> - トラストストアは、サーバーによって SSL 接続で提示された証明書を検証する認定機関 (CA) からの証明書を保存するために使用されます。

このチュートリアルの技術は、Managed Service for Apache Flinkアプリケーションと、次のようなその他の Apache Kafka ソースとのやり取りにも使用されます。

- でホストされているカスタム Apache Kafka クラスター AWS （ Amazon EC2 または Amazon EKS ）
- でホストされている Confluent Kafka クラスター AWS
- AWS Direct Connectまたは VPN 経由でアクセスされるオンプレミスの Kafka クラスター

このチュートリアルには、次のセクションが含まれています。

- Amazon MSK クラスターで VPC を作成する
- カスタムトラストストアを作成してクラスターに適用する
- アプリケーションコードを作成する
- Apache Flink ストリーミング Java コードをアップロードする
- アプリケーションの作成

- アプリケーションを設定する
- アプリケーションを実行する
- アプリケーションをテストする

Amazon MSK クラスターで VPC を作成する

Managed Service for Apache Flinkアプリケーションからアクセスするサンプル VPC と Amazon MSK クラスターを作成するには、Amazon MSK の使用開始チュートリアルに従ってください。

チュートリアルを完了したら、次の操作を実行します。

- ステップ 3: トピックの作成で、kafka-topics.sh --createコマンドを繰り返してAWS KafkaTutorialTopicDestinationという名前の宛先トピックを作成します。

```
bin/kafka-topics.sh --create --bootstrap-server ZooKeeperConnectionString --
replication-factor 3 --partitions 1 --topic AWSKafkaTutorialTopicDestination
```

> **Note**
>
> kafka-topics.shコマンドがZooKeeperClientTimeoutExceptionを返す場合は、Kafka クラスターのセキュリティグループに、クライアントインスタンスのプライベート IP アドレスからのすべてのトラフィックを許可するインバウンドルールがあることを確認します。

- クラスターのブートストラップサーバーリストを記録します。ブートストラップサーバーのリストは、次のコマンドで取得できます (を MSK クラスターの ARN *ClusterArn*に置き換えます)。

```
aws kafka get-bootstrap-brokers --region us-west-2 --cluster-arn ClusterArn
{...
    "BootstrapBrokerStringTls": "b-2.awskafkatutorialcluste.t79r6y.c4.kafka.us-
west-2.amazonaws.com:9094,b-1.awskafkatutorialcluste.t79r6y.c4.kafka.us-
west-2.amazonaws.com:9094,b-3.awskafkatutorialcluste.t79r6y.c4.kafka.us-
west-2.amazonaws.com:9094"
}
```

- このチュートリアルのステップと前提条件となるチュートリアルに従うときは、選択した AWS リージョンをコード、コマンド、コンソールエントリで使用してください。

カスタムトラストストアを作成してクラスターに適用する

このセクションでは、カスタム認証局 (CA) を作成し、それを使用してカスタムトラストストアを生成し、それを MSK クラスターに適用します。

カスタムトラストストアを作成して適用するには、Amazon Managed Streaming for Apache Kafka Developer Guideでクライアント認証チュートリアルに従ってください。

アプリケーションコードを作成する

このセクションでは、アプリケーション JAR ファイルをダウンロードしてコンパイルします。

この例の Java アプリケーションコードは、 から入手できます GitHub。アプリケーションコードをダウンロードするには、次の操作を行います。

1. Git クライアントをまだインストールしていない場合は、インストールします。詳細については、「Git のインストール」をご参照ください。
2. 次のコマンドを使用してリモートリポジトリのクローンを作成します。

```
git clone https://github.com/aws-samples/amazon-kinesis-data-analytics-examples.git
```

3. アプリケーションコードは`amazon-kinesis-data-analytics-java-examples/CustomKeystore`ファイルに含まれています。コードを調べて、Managed Service for Apache Flinkのコードの構造を了解することができます。
4. コマンドライン Maven ツールまたは優先的な開発環境を使用して JAR ファイルを作成します。コマンドライン Maven ツールを使用して JAR ファイルをコンパイルするには、次のように入力します。

```
mvn package -Dflink.version=1.15.3
```

ビルドが成功する場合、次のファイルが作成されます。

```
target/flink-app-1.0-SNAPSHOT.jar
```

> **Note**
>
> 提供されているソースコードは Java 11 のライブラリーに依存しています。

Apache Flink ストリーミング Java コードをアップロードする

このセクションでは、開始方法 (DataStream API) チュートリアルで作成した Amazon S3 バケットにアプリケーションコードをアップロードします。

> **Note**
>
> 入門チュートリアルから Amazon S3 バケットを削除した場合は、the section called “Apache Flink ストリーミング Java コードをアップロードする”手順をもう一度実行してください。

1. Amazon S3 コンソールで、ka-app-code-***<username>*** バケットを選択し、アップロード を選択します。
2. ファイルの選択のステップで、[ファイルを追加] を選択します。前のステップで作成した `flink-app-1.0-SNAPSHOT.jar` ファイルに移動します。
3. オブジェクトの設定を変更する必要はないので、[アップロード] を選択してください。

アプリケーションコードが Amazon S3 バケットに保存され、アプリケーションからアクセスできるようになります。

アプリケーションの作成

1. https://console.aws.amazon.com/flink で Apache Flink 用 Managed Serviceコンソールを開く
2. Managed Service for Apache Flinkのダッシュボードで、「分析アプリケーションの作成」 を選択します。
3. 「Managed Service for Apache Flink-アプリケーションの作成」ページで、次のようにアプリケーションの詳細を入力します。
   - [アプリケーション名] には **MyApplication** と入力します。
   - [ランタイム] には、[Apache Flink 1.15.2] を選択します。
4. [アクセス許可] には、[IAM ロールの作成 / 更新**kinesis-analytics-MyApplication-us-west-2**] を選択します。
5. [Create application] を選択します。

> **Note**
>
> コンソールを使用して Managed Service for Apache Flink を作成する場合は、IAM ロールとポリシーをアプリケーションが自動的に作成するオプションを選択できます。アプリケーションではこのロールとポリシーを使用して、依存リソースにアクセスします。これらの IAM リソースは、次のようにアプリケーション名とリージョンを使用して命名されます。
>
> - ポリシー: `kinesis-analytics-service-MyApplication-us-west-2`
> - ロール: `kinesisanalytics-MyApplication-us-west-2`

アプリケーションを設定する

1. MyApplication ページで、 の設定 を選択します。
2. [Configure application] ページで、[Code location] を次のように指定します。
   - [Amazon S3 バケット] で、**ka-app-code-*<username>***と入力します。
   - [Amazon S3 オブジェクトへのパス] で、**flink-app-1.0-SNAPSHOT.jar**と入力します。
3. [Access to application resources] の [Access permissions] では、[Create / update IAM role**kinesis-analytics-MyApplication-us-west-2**] を選択します。

   > **Note**
   >
   > コンソールを使用してアプリケーションリソース (ログや VPC など) を指定すると、コンソールはアプリケーション実行ロールを変更して、それらのリソースにアクセスする権限を付与します。

4. [プロパティ] で [グループの追加]を選択します。以下のプロパティを入力します。

グループ ID	キー	値
KafkaSource	トピック	AWS KafkaTutorialTopic
KafkaSource	bootstrap.servers	###########################
KafkaSource	security.protocol	SSL

グループ ID	キー	値
KafkaSource	ssl.truststore.location	/usr/lib/jvm/java-11-amazon-corretto/lib/security/cacerts
KafkaSource	ssl.truststore.password	changeit

> **Note**
>
> デフォルト証明書の ssl.truststore.password は「変更してください」です。デフォルト証明書を使用している場合は、この値を変更する必要はありません。

グループの追加をもう一度選択します。以下のプロパティを入力します。

グループ ID	キー	値
KafkaSink	トピック	AWS KafkaTutorialTopic Destination
KafkaSink	bootstrap.servers	###########################
KafkaSink	security.protocol	SSL
KafkaSink	ssl.truststore.location	/usr/lib/jvm/java-11-amazon-corretto/lib/security/cacerts
KafkaSink	ssl.truststore.password	changeit
KafkaSink	transaction.timeout.ms	1,000

アプリケーションコードは上記のアプリケーションプロパティを読み取って、VPC と Amazon MSK クラスターとのやり取りに使用されるソースとシンクを設定します。プロパティ使用の詳細については、「ランタイムプロパティ」を参照してください。

5. スナップショットで 無効 を選択します。これにより、無効なアプリケーション状態データを読み込まずにアプリケーションを簡単に更新できます。

6. [Monitoring] の [Monitoring metrics level] が [Application] に設定されていることを確認します。
7. CloudWatch のログ記録では、有効化チェックボックスを選択します。
8. [仮想プライベートクラウド (VPC)] セクションで、アプリケーションに関連する VPC を選択します。アプリケーションが VPC リソースにアクセスするために使用する VPC に関連付けられているサブネットとセキュリティグループを選択します。
9. [更新] を選択します。

> **Note**
>
> CloudWatch ログ記録を有効にすると、Managed Service for Apache Flink によってロググループとログストリームが作成されます。これらのリソースの名前は次のとおりです。
>
> - ロググループ: `/aws/kinesis-analytics/MyApplication`
> - ログストリーム: `kinesis-analytics-log-stream`
>
> このログストリームはアプリケーションのモニターに使用されます。

アプリケーションを実行する

Flink ジョブグラフは、アプリケーションを実行し、Apache Flink ダッシュボードを開き、目的の Flink ジョブを選択すると表示できます。

アプリケーションをテストする

このセクションでは、レコードをソーストピックに書き込みます。アプリケーションはソーストピックからレコードを読み取り、宛先トピックに書き込みます。アプリケーションが動作していることを確認するには、ソーストピックにレコードを書き込み、宛先トピックからレコードを読み取ります。

トピックからレコードの書き込みと読み取りを行うには、「Amazon MSK の使用開始チュートリアルの「ステップ 6: データの生成と利用」の手順に従います。

ターゲットトピックから読み込むには、クラスターへの 二番目の接続で、ソーストピックの代わりに宛先トピック名を使用してください。

```
bin/kafka-console-consumer.sh --bootstrap-server BootstrapBrokerString --
consumer.config client.properties --topic AWS KafkaTutorialTopicDestination --from-
beginning
```

宛先トピックにレコードが表示されない場合は、トラブルシューティングトピックのVPC 内のリソースにアクセスできないセクションを参照してください。

Python の例

次の例では、Apache Flink Table API をしようた Python でアプリケーションを作成する方法を示しています。

トピック

- 例: Python でのタンブリングウィンドウの作成
- 例: Python でのスライディングウィンドウの作成
- 例: Python で Amazon S3 にストリーミングデータを送信する

例: Python でのタンブリングウィンドウの作成

> **Note**
> 現在の例については、「」を参照してください例。

この演習では、タンブリングウィンドウを使用してデータを集約する Python Managed Service for Apache Flinkを作成します。

> **Note**
> この演習に必要な前提条件を設定するには、まず開始方法 (Python)演習を完了してください。

このトピックには、次のセクションが含まれています。

- 依存リソースを作成する
- サンプルレコードを入力ストリームに書き込む
- アプリケーションコードをダウンロードして調べる
- Apache Flink ストリーミング Python コードを圧縮してアップロードする
- Managed Service for Apache Flink アプリケーションを作成して実行する
- AWS リソースをクリーンアップする

依存リソースを作成する

この練習用の Managed Service for Apache Flink を作成する前に、以下の依存リソースを作成します。

- 2 つの Kinesis Data Streams (`ExampleInputStream`と`ExampleOutputStream`)
- アプリケーションのコードを保存するためのAmazon S3バケット (`ka-app-code-<username>`)

Kinesis ストリームと Amazon S3 バケットは、コンソールを使用して作成できます。これらのリソースの作成手順については、次の各トピックを参照してください。

- 「Amazon Kinesis Data Streamsデベロッパーガイド」の「データストリームの作成および更新」データストリーム**`ExampleInputStream`**と**`ExampleOutputStream`**に名前を付けます。
- Amazon Simple Storage Service ユーザーガイドの「S3 バケットを作成する方法」を参照してください。ログイン名 (**`ka-app-code-<username>`** など) を追加して、Amazon S3 バケットにグローバルに一意の名前を付けます。

サンプルレコードを入力ストリームに書き込む

このセクションでは、Python スクリプトを使用して、アプリケーションが処理するサンプルレコードをストリームに書き込みます。

> **Note**
>
> このセクションでは AWS SDK for Python (Boto) が必要です。

> **Note**
>
> このセクションの Python スクリプトでは、 AWS CLIを使用しています。アカウント認証情報とデフォルトのリージョンを使用する AWS CLI ように を設定する必要があります。を設定するには AWS CLI、次のように入力します。
>
> ```
> aws configure
> ```

1. 次の内容で、`stock.py` という名前のファイルを作成します。

```
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', region_name='us-west-2'))
```

2. stock.py スクリプトを実行します。

```
$ python stock.py
```

チュートリアルの残りの部分を完了する間、スクリプトを実行し続けてください。

アプリケーションコードをダウンロードして調べる

この例の Python アプリケーションコードは、 から入手できます GitHub。アプリケーションコードをダウンロードするには、次の操作を行います。

1. Git クライアントをまだインストールしていない場合は、インストールします。詳細については、「Git のインストール」をご参照ください。
2. 次のコマンドを使用してリモートリポジトリのクローンを作成します。

```
git clone https://github.com/aws-samples/amazon-kinesis-data-analytics-examples.git
```

3. amazon-kinesis-data-analytics-java-examples/python/TumblingWindow ディレクトリに移動します。

アプリケーションコードはtumbling-windows.pyファイルに含まれています。アプリケーションコードに関して、以下の点に注意してください。

- アプリケーションは Kinesis テーブルソースを使用して、ソースストリームから読み取りを行います。次のスニペットは、 create_table 関数を呼び出して Kinesis テーブルソースを作成します。

```
table_env.execute_sql(
        create_input_table(input_table_name, input_stream, input_region,
 stream_initpos)
    )
```

このcreate_table関数は SQL コマンドを使用して、ストリーミングソースに裏付けられたテーブルを作成します。

```
def create_input_table(table_name, stream_name, region, stream_initpos):
    return """ CREATE TABLE {0} (
                ticker VARCHAR(6),
                price DOUBLE,
                event_time TIMESTAMP(3),
                WATERMARK FOR event_time AS event_time - INTERVAL '5' SECOND
              )
              PARTITIONED BY (ticker)
              WITH (
                'connector' = 'kinesis',
                'stream' = '{1}',
                'aws.region' = '{2}',
                'scan.stream.initpos' = '{3}',
                'format' = 'json',
                'json.timestamp-format.standard' = 'ISO-8601'
```

```
            ) """.format(table_name, stream_name, region, stream_initpos)
```

- アプリケーションはこのTumble演算子を使用して、指定されたタンブリングウィンドウ内のレコードを集約し、集計されたレコードをテーブルオブジェクトとして返します。

```
tumbling_window_table = (
        input_table.window(
            Tumble.over("10.seconds").on("event_time").alias("ten_second_window")
        )
        .group_by("ticker, ten_second_window")
        .select("ticker, price.min as price, to_string(ten_second_window.end) as
 event_time")
```

- このアプリケーションは、flink-sql-connector-kinesis-1.15.2.jarからの Kinesis Flink コネクタを使用します。

Apache Flink ストリーミング Python コードを圧縮してアップロードする

このセクションでは、依存リソースを作成する のセクションで作成した Amazon S3 バケットにアプリケーションコードをアップロードします。

1. 任意の圧縮アプリケーションを使用してtumbling-windows.pyおよびflink-sql-connector-kinesis-1.15.2.jarファイルを圧縮します。アーカイブ myapp.zip に名をつけます。
2. Amazon S3 コンソールで、ka-app-code-***<username>*** バケットを選択し、アップロード を選択します。
3. ファイルの選択のステップで、[ファイルを追加] を選択します。前のステップで作成した myapp.zip ファイルに移動します。
4. オブジェクトの設定を変更する必要はないので、[アップロード] を選択してください。

アプリケーションコードが Amazon S3 バケットに保存され、アプリケーションからアクセスできるようになります。

Managed Service for Apache Flink アプリケーションを作成して実行する

以下の手順を実行し、コンソールを使用してアプリケーションを作成、設定、更新、および実行します。

アプリケーションの作成

1. https://console.aws.amazon.com/flink で Apache Flink 用 Managed Serviceコンソールを開く
2. Managed Service for Apache Flinkのダッシュボードで、「分析アプリケーションの作成」 を選択します。
3. 「Managed Service for Apache Flink-アプリケーションの作成」ページで、次のようにアプリケーションの詳細を入力します。
   - [アプリケーション名] には **MyApplication** と入力します。
   - [ランタイム] には、[Apache Flink] を選択します。

     > **Note**
     >
     > Apache Flink 用 Managed Serviceは Apache Flink バージョン 1.15.2 を使用しています。

   - バージョンプルダウンは Apache Flink バージョン 1.15.2 (推奨バージョン) のままにしておきます。
4. [アクセス許可] には、[IAM ロールの作成 / 更新**kinesis-analytics-MyApplication-us-west-2**] を選択します。
5. [Create application] を選択します。

> **Note**
>
> コンソールを使用して Apache Flink アプリケーション用 Managed Service を作成する場合は、IAM ロールとポリシーをアプリケーションが自動的に作成するオプションを選択できます。アプリケーションではこのロールとポリシーを使用して、依存リソースにアクセスします。これらの IAM リソースは、次のようにアプリケーション名とリージョンを使用して命名されます。
>
> - ポリシー: `kinesis-analytics-service-MyApplication-us-west-2`
> - ロール: `kinesisanalytics-MyApplication-us-west-2`

アプリケーションを設定する

1. MyApplication ページで、 の設定 を選択します。

2. [Configure application] ページで、[Code location] を次のように指定します。
   - [Amazon S3 バケット] で、**ka-app-code-*<username>***と入力します。
   - [Amazon S3 オブジェクトへのパス] で、**myapp.zip**と入力します。
3. [Access to application resources] の [Access permissions] では、[Create / update IAM role**kinesis-analytics-MyApplication-us-west-2**] を選択します。
4. [プロパティ] で [グループの追加]を選択します。
5. 次のように入力します。

グループ ID	キー	値
consumer.config.0	**input.stream.name**	**ExampleInputStream**
consumer.config.0	**aws.region**	**us-west-2**
consumer.config.0	**scan.stream.initpos**	**LATEST**

   [保存] を選択します。
6. プロパティ で、グループの追加をもう一度選択します。
7. 次のように入力します。

グループ ID	キー	値
producer.config.0	**output.stream.name**	**ExampleOutputStream**
producer.config.0	**aws.region**	**us-west-2**
producer.config.0	**shard.count**	**1**

8. プロパティ で、グループの追加をもう一度選択します。[グループ ID] に、「**kinesis.analytics.flink.run.options**」と入力します。この特別なプロパティグループは、アプリケーションにコードリソースの場所を指定します。詳細については、「コードファイルの指定」を参照してください。
9. 次のように入力します。

グループ ID	キー	値
kinesis.analytics.flink.run.options	**python**	**tumbling-windows.py**
kinesis.analytics.flink.run.options	**jarfile**	**flink-sql-connector-kinesis-1.15.2.jar**

10. [Monitoring] の [Monitoring metrics level] が [Application] に設定されていることを確認します。
11. CloudWatch のログ記録では、有効化チェックボックスをオンにします。
12. [更新] を選択します。

> **Note**
>
> CloudWatch ログ記録を有効にすると、Managed Service for Apache Flink によってロググループとログストリームが作成されます。これらのリソースの名前は次のとおりです。
>
> - ロググループ: `/aws/kinesis-analytics/MyApplication`
> - ログストリーム: `kinesis-analytics-log-stream`
>
> このログストリームはアプリケーションのモニターに使用されます。このログストリームは、アプリケーションの結果の送信に使用されたログストリームとは異なります。

IAM ポリシーを編集する

IAM ポリシーを編集し、Kinesis Data Streamsにアクセスするための許可を追加します。

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. [ポリシー] を選択します。前のセクションでコンソールによって作成された **kinesis-analytics-service-MyApplication-us-west-2** ポリシーを選択します。
3. [概要] ページで、[ポリシーの編集] を選択します。[JSON] タブを選択します。
4. 次のポリシー例で強調表示されているセクションをポリシーに追加します。サンプルのアカウント ID (*012345678901*) を自分のアカウント ID に置き換えます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ReadCode",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "logs:DescribeLogGroups",
                "s3:GetObjectVersion"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:*",
                "arn:aws:s3:::ka-app-code-<username>/myapp.zip"
            ]
        },
        {
            "Sid": "DescribeLogStreams",
            "Effect": "Allow",
            "Action": "logs:DescribeLogStreams",
            "Resource": "arn:aws:logs:us-west-2:012345678901:log-group:/aws/
kinesis-analytics/MyApplication:log-stream:*"
        },
        {
            "Sid": "PutLogEvents",
            "Effect": "Allow",
            "Action": "logs:PutLogEvents",
            "Resource": "arn:aws:logs:us-west-2:012345678901:log-group:/aws/
kinesis-analytics/MyApplication:log-stream:kinesis-analytics-log-stream"
        },
        {
            "Sid": "ListCloudwatchLogGroups",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:*"
            ]
        },
        {
            "Sid": "ReadInputStream",
            "Effect": "Allow",
```

```
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
        },
        {
            "Sid": "WriteOutputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleOutputStream"
        }
    ]
}
```

アプリケーションを実行する

Flink ジョブグラフは、アプリケーションを実行し、Apache Flink ダッシュボードを開き、目的の Flink ジョブを選択すると表示できます。

CloudWatch コンソールで Managed Service for Apache Flink メトリクスをチェックして、アプリケーションが動作していることを確認できます。

AWS リソースをクリーンアップする

このセクションでは、「タンブリングウィンドウ」チュートリアルで作成した AWS リソースをクリーンアップする手順について説明します。

このトピックには、次のセクションが含まれています。

- Managed Service for Apache Flink アプリケーションを削除する
- Kinesis データストリームを削除する
- Amazon S3 オブジェクトとバケットを削除する
- IAM リソースを削除する
- CloudWatch リソースを削除する

Managed Service for Apache Flink アプリケーションを削除する

1. https://console.aws.amazon.com/flink で Apache Flink 用 Managed Serviceコンソールを開く
2. Managed Service for Apache Flink パネルで、 を選択しますMyApplication。

3. アプリケーションのページで[削除]を選択し、削除を確認します。

Kinesis データストリームを削除する

1. 「https://console.aws.amazon.com/kinesis」で Kinesis コンソールを開きます。
2. Kinesis Data Streams パネルで、 を選択しますExampleInputStream。
3. ExampleInputStream ページで、Kinesis ストリームの削除を選択し、削除を確定します。
4. Kinesis ストリームページで、 を選択しExampleOutputStream、アクション を選択し、削除 を選択し、削除を確定します。

Amazon S3 オブジェクトとバケットを削除する

1. https://console.aws.amazon.com/s3/でAmazon S3 コンソールを開きます。
2. ka-app-code-***<username>*** バケットを選択します。
3. [削除] を選択し、バケット名を入力して削除を確認します。

IAM リソースを削除する

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. ナビゲーションバーで、[ポリシー] を選択します。
3. フィルターコントロールに「kinesis」と入力します。
4. kinesis-analytics-service-MyApplication-us-west-2 ポリシーを選択します。
5. [ポリシーアクション]、[削除] の順に選択します。
6. ナビゲーションバーで [ロール] を選択します。
7. kinesis-analytics-MyApplication-us-west-2 ロールを選択します。
8. [ロールの削除] を選択し、削除を確定します。

CloudWatch リソースを削除する

1. https://console.aws.amazon.com/cloudwatch/ で CloudWatch コンソールを開きます。
2. ナビゲーションバーで [ログ] を選択します。
3. /aws/kinesis-analytics/MyApplication ロググループを選択します。
4. [ロググループの削除]を選択し、削除を確認してください。

例: Python でのスライディングウィンドウの作成

> **Note**
>
> 現在の例については、「」を参照してください例。

> **Note**
>
> この演習に必要な前提条件を設定するには、まず開始方法 (Python)演習を完了してください。

このトピックには、次のセクションが含まれています。

- 依存リソースを作成する
- サンプルレコードを入力ストリームに書き込む
- アプリケーションコードをダウンロードして調べる
- Apache Flink ストリーミング Python コードを圧縮してアップロードする
- Managed Service for Apache Flink アプリケーションを作成して実行する
- AWS リソースをクリーンアップする

依存リソースを作成する

この練習用の Managed Service for Apache Flink を作成する前に、以下の依存リソースを作成します。

- 2 つの Kinesis Data Streams (`ExampleInputStream`と`ExampleOutputStream`)
- アプリケーションのコードを保存するためのAmazon S3バケット (`ka-app-code-<username>`)

Kinesis ストリームと Amazon S3 バケットは、コンソールを使用して作成できます。これらのリソースの作成手順については、次の各トピックを参照してください。

- 「Amazon Kinesis Data Streamsデベロッパーガイド」の「データストリームの作成および更新」データストリーム**`ExampleInputStream`**と**`ExampleOutputStream`**に名前を付けます。

- Amazon Simple Storage Service ユーザーガイドの「S3 バケットを作成する方法」を参照してください。ログイン名 (**ka-app-code-*<username>*** など) を追加して、Amazon S3 バケットにグローバルに一意の名前を付けます。

サンプルレコードを入力ストリームに書き込む

このセクションでは、Python スクリプトを使用して、アプリケーションが処理するサンプルレコードをストリームに書き込みます。

> **Note**
>
> このセクションでは AWS SDK for Python (Boto) が必要です。

> **Note**
>
> このセクションの Python スクリプトでは、 AWS CLIを使用しています。アカウント認証情報とデフォルトのリージョンを使用する AWS CLI ように を設定する必要があります。を設定するには AWS CLI、次のように入力します。
>
> ```
> aws configure
> ```

1. 次の内容で、stock.py という名前のファイルを作成します。

```
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', region_name='us-west-2'))
```

2. stock.py スクリプトを実行します。

```
$ python stock.py
```

チュートリアルの残りの部分を完了する間、スクリプトを実行し続けてください。

アプリケーションコードをダウンロードして調べる

この例の Python アプリケーションコードは、 から入手できます GitHub。アプリケーションコードをダウンロードするには、次の操作を行います。

1. Git クライアントをまだインストールしていない場合は、インストールします。詳細については、「Git のインストール」をご参照ください。
2. 次のコマンドを使用してリモートリポジトリのクローンを作成します。

```
git clone https://github.com/aws-samples/>amazon-kinesis-data-analytics-java-
examples
```

3. amazon-kinesis-data-analytics-java-examples/python/SlidingWindow ディレクトリに移動します。

アプリケーションコードはsliding-windows.pyファイルに含まれています。アプリケーションコードに関して、以下の点に注意してください。

- アプリケーションは Kinesis テーブルソースを使用して、ソースストリームから読み取りを行います。次のスニペットは、 create_input_table 関数を呼び出して Kinesis テーブルソースを作成します。

```
table_env.execute_sql(
        create_input_table(input_table_name, input_stream, input_region,
 stream_initpos)
    )
```

このcreate_input_table関数は SQL コマンドを使用して、ストリーミングソースに裏付けられたテーブルを作成します。

```
def create_input_table(table_name, stream_name, region, stream_initpos):
    return """ CREATE TABLE {0} (
                ticker VARCHAR(6),
                price DOUBLE,
                event_time TIMESTAMP(3),
                WATERMARK FOR event_time AS event_time - INTERVAL '5' SECOND
              )
              PARTITIONED BY (ticker)
              WITH (
                'connector' = 'kinesis',
                'stream' = '{1}',
                'aws.region' = '{2}',
                'scan.stream.initpos' = '{3}',
                'format' = 'json',
                'json.timestamp-format.standard' = 'ISO-8601'
              ) """.format(table_name, stream_name, region, stream_initpos)
 }
```

- アプリケーションはこのSlide演算子を使用して、指定されたスライディングウィンドウ内のレコードを集約し、集計されたレコードをテーブルオブジェクトとして返します。

```
sliding_window_table = (
        input_table
            .window(
                Slide.over("10.seconds")
                .every("5.seconds")
                .on("event_time")
                .alias("ten_second_window")
            )
            .group_by("ticker, ten_second_window")
```

```
            .select("ticker, price.min as price, to_string(ten_second_window.end) as
event_time")
    )
```

- アプリケーションは、flink-sql-connector-kinesis-1.15.2.jar ファイルから Kinesis Flink コネクタを使用します。

Apache Flink ストリーミング Python コードを圧縮してアップロードする

このセクションでは、依存リソースを作成する のセクションで作成した Amazon S3 バケットにアプリケーションコードをアップロードします。

このセクションでは、Python アプリケーションをパッケージ化する方法について説明します。

1. 任意の圧縮アプリケーションを使用してsliding-windows.pyおよびflink-sql-connector-kinesis-1.15.2.jarファイルを圧縮します。アーカイブ myapp.zip に名をつけます。
2. Amazon S3 コンソールで、ka-app-code-*<username>* バケットを選択し、アップロード を選択します。
3. ファイルの選択のステップで、[ファイルを追加] を選択します。前のステップで作成した myapp.zip ファイルに移動します。
4. オブジェクトの設定を変更する必要はないので、[アップロード] を選択してください。

アプリケーションコードが Amazon S3 バケットに保存され、アプリケーションからアクセスできるようになります。

Managed Service for Apache Flink アプリケーションを作成して実行する

以下の手順を実行し、コンソールを使用してアプリケーションを作成、設定、更新、および実行します。

アプリケーションの作成

1. https://console.aws.amazon.com/flink で Apache Flink 用 Managed Serviceコンソールを開く
2. Managed Service for Apache Flinkのダッシュボードで、「分析アプリケーションの作成」 を選択します。
3. 「Managed Service for Apache Flink-アプリケーションの作成」ページで、次のようにアプリケーションの詳細を入力します。

- [アプリケーション名] には **MyApplication** と入力します。
- [ランタイム] には、[Apache Flink] を選択します。

> **Note**
>
> Apache Flink 用 Managed Serviceは Apache Flink バージョン 1.15.2 を使用しています。

- バージョンプルダウンは Apache Flink バージョン 1.15.2 (推奨バージョン) のままにしておきます。

4. [アクセス許可] には、[IAM ロールの作成 / 更新**kinesis-analytics-MyApplication-us-west-2**] を選択します。
5. [Create application] を選択します。

> **Note**
>
> コンソールを使用して Apache Flink アプリケーション用 Managed Service を作成する場合は、IAM ロールとポリシーをアプリケーションが自動的に作成するオプションを選択できます。アプリケーションではこのロールとポリシーを使用して、依存リソースにアクセスします。これらの IAM リソースは、次のようにアプリケーション名とリージョンを使用して命名されます。
>
> - ポリシー: `kinesis-analytics-service-MyApplication-us-west-2`
> - ロール: `kinesisanalytics-MyApplication-us-west-2`

アプリケーションを設定する

1. MyApplication ページで、 の設定 を選択します。
2. [Configure application] ページで、[Code location] を次のように指定します。
   - [Amazon S3 バケット] で、**ka-app-code-*<username>***と入力します。
   - [Amazon S3 オブジェクトへのパス] で、**myapp.zip**と入力します。
3. [Access to application resources] の [Access permissions] では、[Create / update IAM role**kinesis-analytics-MyApplication-us-west-2**] を選択します。

4. [プロパティ] で [グループの追加]を選択します。
5. 次のアプリケーションのプロパティと値を入力します。

グループ ID	キー	値
consumer.config.0	**input.stream.name**	**ExampleInputStream**
consumer.config.0	**aws.region**	**us-west-2**
consumer.config.0	**scan.stream.initpos**	**LATEST**

[保存] を選択します。

6. プロパティで、グループの追加をもう一度選択します。
7. 次のアプリケーションのプロパティと値を入力します。

グループ ID	キー	値
producer.config.0	**output.stream.name**	**ExampleOutputStream**
producer.config.0	**aws.region**	**us-west-2**
producer.config.0	**shard.count**	**1**

8. プロパティ で、グループの追加をもう一度選択します。[グループ ID] に、「**kinesis.analytics.flink.run.options**」と入力します。この特別なプロパティグループは、アプリケーションにコードリソースの場所を指定します。詳細については、「コードファイルの指定」を参照してください。
9. 次のアプリケーションのプロパティと値を入力します。

グループ ID	キー	値
kinesis.analytics.flink.run.options	**python**	**sliding-windows.py**

グループ ID	キー	値
kinesis.analytics.flink.run.options	**jarfile**	**flink-sql-connector-kinesis_1.15.2.jar**

10. [Monitoring] の [Monitoring metrics level] が [Application] に設定されていることを確認します。
11. CloudWatch のログ記録では、有効化チェックボックスをオンにします。
12. [更新] を選択します。

> **Note**
>
> CloudWatch ログ記録を有効にすると、Managed Service for Apache Flink によってロググループとログストリームが作成されます。これらのリソースの名前は次のとおりです。
>
> - ロググループ: `/aws/kinesis-analytics/MyApplication`
> - ログストリーム: `kinesis-analytics-log-stream`
>
> このログストリームはアプリケーションのモニターに使用されます。このログストリームは、アプリケーションの結果の送信に使用されたログストリームとは異なります。

IAM ポリシーを編集する

IAM ポリシーを編集し、Kinesis Data Streamsにアクセスするための許可を追加します。

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. [ポリシー] を選択します。前のセクションでコンソールによって作成された **kinesis-analytics-service-MyApplication-us-west-2** ポリシーを選択します。
3. [概要] ページで、[ポリシーの編集] を選択します。[JSON] タブを選択します。
4. 次のポリシー例で強調表示されているセクションをポリシーに追加します。サンプルのアカウント ID (*012345678901*) を自分のアカウント ID に置き換えます。

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

```
            "Sid": "ReadCode",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "logs:DescribeLogGroups",
                "s3:GetObjectVersion"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:*",
                "arn:aws:s3:::ka-app-code-<username>/myapp.zip"
            ]
        },
        {
            "Sid": "DescribeLogStreams",
            "Effect": "Allow",
            "Action": "logs:DescribeLogStreams",
            "Resource": "arn:aws:logs:us-west-2:012345678901:log-group:/aws/
kinesis-analytics/MyApplication:log-stream:*"
        },
        {
            "Sid": "PutLogEvents",
            "Effect": "Allow",
            "Action": "logs:PutLogEvents",
            "Resource": "arn:aws:logs:us-west-2:012345678901:log-group:/aws/
kinesis-analytics/MyApplication:log-stream:kinesis-analytics-log-stream"
        },
        {
            "Sid": "ListCloudwatchLogGroups",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:*"
            ]
        },
        {
            "Sid": "ReadInputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
        },
        {
```

```
            "Sid": "WriteOutputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleOutputStream"
        }
    ]
}
```

アプリケーションを実行する

Flink ジョブグラフは、アプリケーションを実行し、Apache Flink ダッシュボードを開き、目的の Flink ジョブを選択すると表示できます。

CloudWatch コンソールで Managed Service for Apache Flink メトリクスをチェックして、アプリケーションが動作していることを確認できます。

AWS リソースをクリーンアップする

このセクションでは、スライディングウィンドウチュートリアルで作成した AWS リソースをクリーンアップする手順について説明します。

このトピックには、次のセクションが含まれています。

- Managed Service for Apache Flink アプリケーションを削除する
- Kinesis データストリームを削除する
- Amazon S3 オブジェクトとバケットを削除する
- IAM リソースを削除する
- CloudWatch リソースを削除する

Managed Service for Apache Flink アプリケーションを削除する

1. https://console.aws.amazon.com/flink で Apache Flink 用 Managed Serviceコンソールを開く
2. Managed Service for Apache Flink パネルで、 を選択しますMyApplication。
3. アプリケーションのページで[削除]を選択し、削除を確認します。

Kinesis データストリームを削除する

1. 「https://console.aws.amazon.com/kinesis」で Kinesis コンソールを開きます。

2. Kinesis Data Streams パネルで、 を選択しますExampleInputStream。
3. ExampleInputStream ページで、Kinesis ストリームの削除を選択し、削除を確定します。
4. Kinesis ストリームページで、 を選択しExampleOutputStream、アクション を選択し、削除 を選択し、削除を確定します。

Amazon S3 オブジェクトとバケットを削除する

1. https://console.aws.amazon.com/s3/でAmazon S3 コンソールを開きます。
2. ka-app-code-***<username>*** バケットを選択します。
3. [削除] を選択し、バケット名を入力して削除を確認します。

IAM リソースを削除する

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. ナビゲーションバーで、[ポリシー] を選択します。
3. フィルターコントロールに「kinesis」と入力します。
4. kinesis-analytics-service-MyApplication-us-west-2 ポリシーを選択します。
5. [ポリシーアクション]、[削除] の順に選択します。
6. ナビゲーションバーで [ロール] を選択します。
7. kinesis-analytics-MyApplication-us-west-2 ロールを選択します。
8. [ロールの削除] を選択し、削除を確定します。

CloudWatch リソースを削除する

1. https://console.aws.amazon.com/cloudwatch/ で CloudWatch コンソールを開きます。
2. ナビゲーションバーで [ログ] を選択します。
3. /aws/kinesis-analytics/MyApplication ロググループを選択します。
4. [ロググループの削除]を選択し、削除を確認してください。

例: Python で Amazon S3 にストリーミングデータを送信する

> **Note**
>
> 現在の例については、「」を参照してください例。

この演習では、Amazon Simple Storage Service シンクにデータをストリーミングするPython Managed Service for Apache Flinkアプリケーションを作成します。

> **Note**
>
> この演習に必要な前提条件を設定するには、まず開始方法 (Python)演習を完了してください。

このトピックには、次のセクションが含まれています。

- 依存リソースを作成する
- サンプルレコードを入力ストリームに書き込む
- アプリケーションコードをダウンロードして調べる
- Apache Flink ストリーミング Python コードを圧縮してアップロードする
- Managed Service for Apache Flink アプリケーションを作成して実行する
- AWS リソースをクリーンアップする

依存リソースを作成する

この練習用の Managed Service for Apache Flink を作成する前に、以下の依存リソースを作成します。

- A Kinesis data stream (`ExampleInputStream`)
- アプリケーションのコードと出力を格納する Amazon S3 バケット (`ka-app-code-<username>`)

> **Note**
>
> Managed Service for Apache Flinkでサーバー側の暗号化が有効になる場合、Managed Service for Apache Flink は Amazon S3 にデータを書き込むことができません。

Kinesis ストリームと Amazon S3 バケットは、コンソールを使用して作成できます。これらのリソースの作成手順については、次の各トピックを参照してください。

- 「Amazon Kinesis Data Streamsデベロッパーガイド」の「データストリームの作成および更新」データストリームに**ExampleInputStream**と名前を付けます。
- Amazon Simple Storage Service ユーザーガイドの「S3 バケットを作成する方法」を参照してください。ログイン名 (**ka-app-code-*<username>*** など) を追加して、Amazon S3 バケットにグローバルに一意の名前を付けます。

サンプルレコードを入力ストリームに書き込む

このセクションでは、Python スクリプトを使用して、アプリケーションが処理するサンプルレコードをストリームに書き込みます。

> **Note**
>
> このセクションでは AWS SDK for Python (Boto) が必要です。

> **Note**
>
> このセクションの Python スクリプトでは、 AWS CLIを使用しています。アカウント認証情報とデフォルトのリージョンを使用する AWS CLI ように を設定する必要があります。を設定するには AWS CLI、次のように入力します。
>
> ```
> aws configure
> ```

1. 次の内容で、stock.py という名前のファイルを作成します。

```
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', region_name='us-west-2'))
```

2. stock.py スクリプトを実行します。

```
$ python stock.py
```

チュートリアルの残りの部分を完了する間、スクリプトを実行し続けてください。

アプリケーションコードをダウンロードして調べる

この例の Python アプリケーションコードは、 から入手できます GitHub。アプリケーションコードをダウンロードするには、次の操作を行います。

1. Git クライアントをまだインストールしていない場合は、インストールします。詳細については、「Git のインストール」をご参照ください。
2. 次のコマンドを使用してリモートリポジトリのクローンを作成します。

```
git clone https://github.com/aws-samples/amazon-kinesis-data-analytics-examples.git
```

3. amazon-kinesis-data-analytics-java-examples/python/S3Sink ディレクトリに移動します。

アプリケーションコードはstreaming-file-sink.pyファイルに含まれています。アプリケーションコードに関して、以下の点に注意してください。

- アプリケーションは Kinesis テーブルソースを使用して、ソースストリームから読み取りを行います。次のスニペットは、 create_source_table 関数を呼び出して Kinesis テーブルソースを作成します。

```
table_env.execute_sql(
        create_source_table(input_table_name, input_stream, input_region,
 stream_initpos)
    )
```

このcreate_source_table関数は SQL コマンドを使用して、ストリーミングソースに裏付けられたテーブルを作成します。

```
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', region_name='us-west-2'))
```

- アプリケーションはfilesystem演算子を使用して Amazon S3 バケットにレコードを送信します。

```
def create_sink_table(table_name, bucket_name):
    return """ CREATE TABLE {0} (
                ticker VARCHAR(6),
                price DOUBLE,
                event_time VARCHAR(64)
              )
              PARTITIONED BY (ticker)
              WITH (
                  'connector'='filesystem',
                  'path'='s3a://{1}/',
                  'format'='json',
                  'sink.partition-commit.policy.kind'='success-file',
                  'sink.partition-commit.delay' = '1 min'
              ) """.format(table_name, bucket_name)
```

- アプリケーションは、flink-sql-connector-kinesis-1.15.2.jar ファイルから Kinesis Flink コネクタを使用します。

Apache Flink ストリーミング Python コードを圧縮してアップロードする

このセクションでは、依存リソースを作成する のセクションで作成した Amazon S3 バケットにアプリケーションコードをアップロードします。

1. お好みの圧縮アプリケーションを使用して、 streaming-file-sink.pyおよび flink-sql-connector-kinesis-1.15.2.jar ファイルを圧縮します。アーカイブ myapp.zip に名をつけます。
2. Amazon S3 コンソールで、ka-app-code-***<username>*** バケットを選択し、アップロード を選択します。
3. ファイルの選択のステップで、[ファイルを追加] を選択します。前のステップで作成した myapp.zip ファイルに移動します。
4. オブジェクトの設定を変更する必要はないので、[アップロード] を選択してください。

アプリケーションコードが Amazon S3 バケットに保存され、アプリケーションからアクセスできるようになります。

Managed Service for Apache Flink アプリケーションを作成して実行する

以下の手順を実行し、コンソールを使用してアプリケーションを作成、設定、更新、および実行します。

アプリケーションの作成

1. https://console.aws.amazon.com/flink で Apache Flink 用 Managed Serviceコンソールを開く
2. Managed Service for Apache Flinkのダッシュボードで、「分析アプリケーションの作成」 を選択します。
3. 「Managed Service for Apache Flink-アプリケーションの作成」ページで、次のようにアプリケーションの詳細を入力します。
   - [アプリケーション名] には **MyApplication** と入力します。
   - [ランタイム] には、[Apache Flink] を選択します。

     > **Note**
     >
     > Apache Flink 用 Managed Serviceは Apache Flink バージョン 1.15.2 を使用しています。

   - バージョンプルダウンは Apache Flink バージョン 1.15.2 (推奨バージョン) のままにしておきます。
4. [アクセス許可] には、[IAM ロールの作成 / 更新**kinesis-analytics-MyApplication-us-west-2**] を選択します。
5. [Create application] を選択します。

> **Note**
>
> コンソールを使用して Apache Flink アプリケーション用 Managed Service を作成する場合は、IAM ロールとポリシーをアプリケーションが自動的に作成するオプションを選択できます。アプリケーションではこのロールとポリシーを使用して、依存リソースにアクセスします。これらの IAM リソースは、次のようにアプリケーション名とリージョンを使用して命名されます。
>
> - ポリシー: `kinesis-analytics-service-`*`MyApplication`*`-`*`us-west-2`*
> - ロール: `kinesisanalytics-`*`MyApplication`*`-`*`us-west-2`*

アプリケーションを設定する

1. MyApplication ページで、 の設定 を選択します。
2. [Configure application] ページで、[Code location] を次のように指定します。
   - [Amazon S3 バケット] で、**ka-app-code-*<username>***と入力します。
   - [Amazon S3 オブジェクトへのパス] で、**myapp.zip**と入力します。
3. [Access to application resources] の [Access permissions] では、[Create / update IAM role**kinesis-analytics-MyApplication-us-west-2**] を選択します。
4. [プロパティ] で [グループの追加]を選択します。
5. 次のアプリケーションのプロパティと値を入力します。

グループ ID	キー	値
consumer.config.0	**input.stream.name**	**ExampleInputStream**
consumer.config.0	**aws.region**	**us-west-2**
consumer.config.0	**scan.stream.initpos**	**LATEST**

   [保存] を選択します。
6. プロパティ で、グループの追加をもう一度選択します。[グループ ID] に、「**kinesis.analytics.flink.run.options**」と入力します。この特別なプロパティグループは、アプリケーションにコードリソースの場所を指定します。詳細については、「コードファイルの指定」を参照してください。
7. 次のアプリケーションのプロパティと値を入力します。

グループ ID	キー	値
kinesis.analytics.flink.run.options	**python**	**streaming-file-sink.py**
kinesis.analytics.flink.run.options	**jarfile**	**S3Sink/lib/flink-sql-connector-kinesis-1.15.2.jar**

8. プロパティ で、グループの追加をもう一度選択します。[グループ ID] に、「**sink.config.0**」と入力します。この特別なプロパティグループは、アプリケーションにコードリソースの場所を指定します。詳細については、「コードファイルの指定」を参照してください。
9. 次のアプリケーションプロパティと値を入力します (*bucket-name*をAmazon S3 バケットの実際の名前に置き換えてください)。

グループ ID	キー	値
sink.config.0	**output.bucket.name**	***bucket-name***

10. [Monitoring] の [Monitoring metrics level] が [Application] に設定されていることを確認します。
11. CloudWatch のログ記録では、有効化チェックボックスをオンにします。
12. [更新] を選択します。

> **Note**
>
> CloudWatch ログ記録を有効にすると、Managed Service for Apache Flink によってロググループとログストリームが作成されます。これらのリソースの名前は次のとおりです。
>
> - ロググループ: `/aws/kinesis-analytics/MyApplication`
> - ログストリーム: `kinesis-analytics-log-stream`
>
> このログストリームはアプリケーションのモニターに使用されます。このログストリームは、アプリケーションの結果の送信に使用されたログストリームとは異なります。

IAM ポリシーを編集する

IAM ポリシーを編集し、Kinesis Data Streamsにアクセスするための許可を追加します。

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. [ポリシー] を選択します。前のセクションでコンソールによって作成された **kinesis-analytics-service-MyApplication-us-west-2** ポリシーを選択します。
3. [概要] ページで、[ポリシーの編集] を選択します。[JSON] タブを選択します。

4. 次のポリシー例で強調表示されているセクションをポリシーに追加します。サンプルのアカウント ID (*012345678901*) を自分のアカウント ID に置き換えます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ReadCode",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "logs:DescribeLogGroups",
                "s3:GetObjectVersion"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:*",
                "arn:aws:s3:::ka-app-code-<username>/myapp.zip"
            ]
        },
        {
            "Sid": "DescribeLogStreams",
            "Effect": "Allow",
            "Action": "logs:DescribeLogStreams",
            "Resource": "arn:aws:logs:us-west-2:012345678901:log-group:/aws/
kinesis-analytics/MyApplication:log-stream:*"
        },
        {
            "Sid": "PutLogEvents",
            "Effect": "Allow",
            "Action": "logs:PutLogEvents",
            "Resource": "arn:aws:logs:us-west-2:012345678901:log-group:/aws/
kinesis-analytics/MyApplication:log-stream:kinesis-analytics-log-stream"
        },
        {
            "Sid": "ListCloudwatchLogGroups",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:*"
            ]
        },
```

```
        {
            "Sid": "ReadInputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
        },
        {
            "Sid": "WriteObjects",
            "Effect": "Allow",
            "Action": [
                "s3:Abort*",
                "s3:DeleteObject*",
                "s3:GetObject*",
                "s3:GetBucket*",
                "s3:List*",
                "s3:ListBucket",
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::ka-app-code-<username>",
                "arn:aws:s3:::ka-app-code-<username>/*"
            ]
        }
    ]
}
```

アプリケーションを実行する

Flink ジョブグラフは、アプリケーションを実行し、Apache Flink ダッシュボードを開き、目的の Flink ジョブを選択すると表示できます。

CloudWatch コンソールで Managed Service for Apache Flink メトリクスをチェックして、アプリケーションが動作していることを確認できます。

AWS リソースをクリーンアップする

このセクションでは、スライディングウィンドウチュートリアルで作成した AWS リソースをクリーンアップする手順について説明します。

このトピックには、次のセクションが含まれています。

- Managed Service for Apache Flink アプリケーションを削除する

- Kinesis データストリームを削除する
- Amazon S3 オブジェクトとバケットを削除する
- IAM リソースを削除する
- CloudWatch リソースを削除する

Managed Service for Apache Flink アプリケーションを削除する

1. https://console.aws.amazon.com/flink で Apache Flink 用 Managed Serviceコンソールを開く
2. Managed Service for Apache Flink パネルで、 を選択しますMyApplication。
3. アプリケーションのページで[削除]を選択し、削除を確認します。

Kinesis データストリームを削除する

1. Kinesis コンソール (https://console.aws.amazon.com/kinesis) を開きます。
2. Kinesis Data Streams パネルで、 を選択しますExampleInputStream。
3. ExampleInputStream ページで、Kinesis ストリームの削除を選択し、削除を確定します。

Amazon S3 オブジェクトとバケットを削除する

1. https://console.aws.amazon.com/s3/でAmazon S3 コンソールを開きます。
2. ka-app-code-***<username>*** バケットを選択します。
3. [削除] を選択し、バケット名を入力して削除を確認します。

IAM リソースを削除する

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. ナビゲーションバーで、[ポリシー] を選択します。
3. フィルターコントロールに「kinesis」と入力します。
4. kinesis-analytics-service-MyApplication-us-west-2 ポリシーを選択します。
5. [ポリシーアクション]、[削除] の順に選択します。
6. ナビゲーションバーで [ロール] を選択します。
7. kinesis-analytics-MyApplication-us-west-2 ロールを選択します。

8. [ロールの削除] を選択し、削除を確定します。

CloudWatch リソースを削除する

1. https://console.aws.amazon.com/cloudwatch/ で CloudWatch コンソールを開きます。
2. ナビゲーションバーで [ログ] を選択します。
3. /aws/kinesis-analytics/MyApplication ロググループを選択します。
4. [ロググループの削除]を選択し、削除を確認してください。

Scala の例

以下の例では、Scala を使用した Apache Flink を使用してアプリケーションを作成する方法を示しています。

トピック

- 例: Scala でのタンブリングウィンドウの作成
- 例: Scala でのスライディングウィンドウの作成
- 例: Scala で Amazon S3 にストリーミングデータを送信する

例: Scala でのタンブリングウィンドウの作成

> **Note**
> 現在の例については、「」を参照してください例。

> **Note**
> バージョン 1.15 以降、Flink は Scala フリーになりました。アプリケーションが Scala の任意のバージョンから Java API を使用できるようになっています。Flink は今でも内部的にいくつかの主要コンポーネントで Scala を使用していますが、Scala をユーザーコードのクラスローダーに公開していません。そのため、ユーザーは Scala の依存関係を自分の JAR アーカイブに追加する必要があります。
> Flink 1.15 での Scala の変更についての詳しい情報は、Scala Free in One Fifteenを参照してください。

この演習では、Scala 3.2.0 と Flink の Java DataStream API を使用するシンプルなストリーミングアプリケーションを作成します。アプリケーションは Kinesis ストリームからデータを読み取り、スライディングウィンドウを使用して集約し、結果を出力Kinesisストリームに書き込みます。

> **Note**
>
> この練習に必要な前提条件を設定するには、まずGetting Started (Scala)の練習を完了してください。

このトピックには、次のセクションが含まれています。

- アプリケーションコードをダウンロードして調べる
- アプリケーション・コードをコンパイルしてアップロードするには
- アプリケーションの作成と実行 (コンソール)
- アプリケーションの作成と実行 (CLI)
- アプリケーションコードの更新
- AWS リソースをクリーンアップする

アプリケーションコードをダウンロードして調べる

この例の Python アプリケーションコードは、 から入手できます GitHub。アプリケーションコードをダウンロードするには、次の操作を行います。

1. Git クライアントをまだインストールしていない場合は、インストールします。詳細については、「Git のインストール」をご参照ください。
2. 次のコマンドを使用してリモートリポジトリのクローンを作成します。

```
git clone https://github.com/aws-samples/amazon-kinesis-data-analytics-examples.git
```

3. amazon-kinesis-data-analytics-java-examples/scala/TumblingWindow ディレクトリに移動します。

アプリケーションコードに関して、以下の点に注意してください。

- build.sbtファイルには、Managed Service for Apache Flink ライブラリなど、アプリケーションの設定と依存関係に関する情報が含まれています。

- この BasicStreamingJob.scala ファイルには、アプリケーションの機能を定義するメインメソッドが含まれています。
- アプリケーションは Kinesis ソースを使用して、ソースストリームから読み取りを行います。次のスニペットでは、Kinesis ソースが作成されます。

```
private def createSource: FlinkKinesisConsumer[String] = {
  val applicationProperties = KinesisAnalyticsRuntime.getApplicationProperties
  val inputProperties = applicationProperties.get("ConsumerConfigProperties")

  new FlinkKinesisConsumer[String](inputProperties.getProperty(streamNameKey,
 defaultInputStreamName),
    new SimpleStringSchema, inputProperties)
}
```

また、アプリケーションは Kinesis シンクを使用して結果ストリームに書き込みます。次のスニペットでは、Kinesis シンクが作成されます。

```
private def createSink: KinesisStreamsSink[String] = {
  val applicationProperties = KinesisAnalyticsRuntime.getApplicationProperties
  val outputProperties = applicationProperties.get("ProducerConfigProperties")

  KinesisStreamsSink.builder[String]
    .setKinesisClientProperties(outputProperties)
    .setSerializationSchema(new SimpleStringSchema)
    .setStreamName(outputProperties.getProperty(streamNameKey,
 defaultOutputStreamName))
    .setPartitionKeyGenerator((element: String) => String.valueOf(element.hashCode))
    .build
}
```

- アプリケーションはウィンドウ演算子を使用して、5 秒間のタンブリングウィンドウにおける各銘柄記号の値を求めます。次のコードは演算子を作成し、集約されたデータを新しい Kinesis Data Streamsシンクに送信します。

```
environment.addSource(createSource)
  .map { value =>
    val jsonNode = jsonParser.readValue(value, classOf[JsonNode])
    new Tuple2[String, Int](jsonNode.get("ticker").toString, 1)
  }
  .returns(Types.TUPLE(Types.STRING, Types.INT))
  .keyBy(v => v.f0) // Logically partition the stream for each ticker
```

```
    .window(TumblingProcessingTimeWindows.of(Time.seconds(10)))
    .sum(1) // Sum the number of tickers per partition
    .map { value => value.f0 + "," + value.f1.toString + "\n" }
    .sinkTo(createSink)
```

- アプリケーションは、 StreamExecutionEnvironment オブジェクトを使用して外部リソースにアクセスするためのソースコネクタとシンクコネクタを作成します。
- アプリケーションは、動的アプリケーションプロパティを使用してソースコネクタとシンクコネクタを作成します。アプリケーションのプロパティを読み取ってコネクタを設定します。ランタイムプロパティの詳細については、ランタイムプロパティを参照してください。

アプリケーション・コードをコンパイルしてアップロードするには

このセクションでは、アプリケーションコードをコンパイルして Amazon S3 バケットにアップロードします。

アプリケーションコードのコンパイル

SBT ビルドツールを使用して、アプリケーションの Scala コードをビルドします。SBTをインストールするには、Install sbt with cs setupを参照してください。また、Java 開発キット (JDK) をインストールする必要があります。演習を完了するための前提条件 を参照してください。

1. アプリケーションコードを使用するには、コードをコンパイルして JAR ファイルにパッケージ化します。SBT を使用してコードをコンパイルしてパッケージ化できます。

   ```
   sbt assembly
   ```

2. アプリケーションのコンパイルに成功すると、次のファイルが作成されます。

   ```
   target/scala-3.2.0/tumbling-window-scala-1.0.jar
   ```

Apache Flink Streaming Scala Code のアップロード

このセクションでは、Amazon S3 バケットを作成し、アプリケーションコードをアップロードします。

1. https://console.aws.amazon.com/s3/でAmazon S3 コンソールを開きます。
2. [バケットを作成] を選択します。

3. [Bucket name (バケット名)] フィールドにka-app-code-<username>と入力します。バケット名にユーザー名などのサフィックスを追加して、グローバルに一意にします。[次へ] をクリックします。
4. 設定オプションのステップでは、設定をそのままにし、[次へ] を選択します。
5. アクセス許可の設定のステップでは、設定をそのままにし、[次へ] を選択します。
6. [バケットを作成] を選択します。
7. ka-app-code-<username>バケットを選択し、アップロード を選択します。
8. ファイルの選択のステップで、[ファイルを追加] を選択します。前のステップで作成した tumbling-window-scala-1.0.jar ファイルに移動します。
9. オブジェクトの設定を変更する必要はないので、[アップロード] を選択してください。

アプリケーションコードが Amazon S3 バケットに保存され、アプリケーションからアクセスできるようになります。

アプリケーションの作成と実行 (コンソール)

以下の手順を実行し、コンソールを使用してアプリケーションを作成、設定、更新、および実行します。

アプリケーションの作成

1. https://console.aws.amazon.com/flink で Apache Flink 用 Managed Serviceコンソールを開く
2. Managed Service for Apache Flinkのダッシュボードで、「分析アプリケーションの作成」 を選択します。
3. 「Managed Service for Apache Flink-アプリケーションの作成」ページで、次のようにアプリケーションの詳細を入力します。
   - [アプリケーション名] には **MyApplication** と入力します。
   - [Description (説明)] に **My Scala test app** と入力します。
   - [ランタイム] には、[Apache Flink] を選択します。
   - バージョンはApache Flink バージョン 1.15.2 (推薦バージョン)のままにしておきます。
4. [アクセス許可] には、[IAM ロールの作成 / 更新**kinesis-analytics-MyApplication-us-west-2**] を選択します。
5. [Create application] を選択します。

> **Note**
>
> コンソールを使用して Apache Flink アプリケーション用 Managed Service を作成する場合は、IAM ロールとポリシーをアプリケーションが自動的に作成するオプションを選択できます。アプリケーションではこのロールとポリシーを使用して、依存リソースにアクセスします。これらの IAM リソースは、次のようにアプリケーション名とリージョンを使用して命名されます。
>
> - ポリシー: `kinesis-analytics-service-MyApplication-us-west-2`
> - ロール: `kinesisanalytics-MyApplication-us-west-2`

アプリケーションを設定する

アプリケーションを設定するには、次の手順に従います。

アプリケーションを構成するには

1. MyApplication ページで、 の設定 を選択します。
2. [Configure application] ページで、[Code location] を次のように指定します。
   - [Amazon S3 バケット] で、**ka-app-code-*<username>***と入力します。
   - [Amazon S3 オブジェクトへのパス] で、**tumbling-window-scala-1.0.jar**と入力します。
3. [Access to application resources] の [Access permissions] では、[Create / update IAM role**kinesis-analytics-MyApplication-us-west-2**] を選択します。
4. [プロパティ] で [グループの追加]を選択します。
5. 次のように入力します。

グループ ID	キー	値
ConsumerConfigProperties	**input.stream.name**	**ExampleInputStream**
ConsumerConfigProperties	**aws.region**	**us-west-2**

グループ ID	キー	値
ConsumerConfigProperties	**flink.stream.initpos**	**LATEST**

[保存] を選択します。

6. プロパティ で、グループの追加をもう一度選択します。
7. 次のように入力します。

グループ ID	キー	値
ProducerConfigProperties	**output.stream.name**	**ExampleOutputStream**
ProducerConfigProperties	**aws.region**	**us-west-2**

8. [Monitoring] の [Monitoring metrics level] が [Application] に設定されていることを確認します。
9. CloudWatch ログ記録 で、有効化チェックボックスを選択します。
10. [更新] を選択します。

> **Note**
>
> Amazon CloudWatch ログ記録を有効にすると、Managed Service for Apache Flink によってロググループとログストリームが作成されます。これらのリソースの名前は次のとおりです。
>
> - ロググループ: `/aws/kinesis-analytics/MyApplication`
> - ログストリーム: `kinesis-analytics-log-stream`

IAM ポリシーを編集する

Amazon S3 バケットにアクセスする許可を追加するように IAM ポリシーを編集します。

IAM ポリシーを編集して S3 バケット権限を追加するには

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. [ポリシー] を選択します。前のセクションでコンソールによって作成された **kinesis-analytics-service-MyApplication-us-west-2** ポリシーを選択します。
3. [概要] ページで、[ポリシーの編集] を選択します。[JSON] タブを選択します。
4. 次のポリシー例で強調表示されているセクションをポリシーに追加します。サンプルのアカウント ID (*012345678901*) を自分のアカウント ID に置き換えます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ReadCode",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": [
                "arn:aws:s3:::ka-app-code-username/tumbling-window-scala-1.0.jar"
            ]
        },
        {
            "Sid": "DescribeLogGroups",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:*"
            ]
        },
        {
            "Sid": "DescribeLogStreams",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogStreams"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:*"
```

```
            ]
        },
        {
            "Sid": "PutLogEvents",
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:kinesis-analytics-log-stream"
            ]
        },
        {
            "Sid": "ReadInputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
        },
        {
            "Sid": "WriteOutputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleOutputStream"
        }
    ]
}
```

アプリケーションを実行する

Flink ジョブグラフは、アプリケーションを実行し、Apache Flink ダッシュボードを開き、目的の Flink ジョブを選択すると表示できます。

アプリケーションの停止

アプリケーションを停止するには、MyApplicationページで停止 を選択します。アクションを確認します。

アプリケーションの作成と実行 (CLI)

このセクションでは、 を使用して Managed Service for Apache Flink アプリケーション AWS Command Line Interface を作成して実行します。kinesisanalyticsv2 AWS CLI コマンドを使用して、Managed Service for Apache Flink アプリケーションを作成して操作します。

許可ポリシーを作成する

> **Note**
>
> アプリケーションのアクセス権限ポリシーとロールを作成する必要があります。これらの IAM リソースを作成しない場合、アプリケーションはそのデータストリームやログストリームにアクセスできません。

まず、2 つのステートメントを含むアクセス許可ポリシーを作成します。1 つは、ソースストリームの読み取りアクションに対するアクセス許可を付与し、もう 1 つはシンクストリームの書き込みアクションに対するアクセス許可を付与します。次に、IAM ロール (次のセクションで作成) にポリシーをアタッチします。そのため、 Managed Service for Apache Flinkがこのロールを引き受けると、ソースストリームからの読み取りとシンクストリームへの書き込みを行うために必要なアクセス許可がサービスに付与されます。

次のコードを使用して AKReadSourceStreamWriteSinkStream アクセス許可ポリシーを作成します。**username** を Amazon S3 バケットの作成に使用したユーザー名に置き換え、アプリケーションコードを保存します。Amazon リソースネーム (ARN) **(012345678901)**のアカウント ID を自分のアカウント ID に置き換えます。**MF-stream-rw-role**サービス実行ロールは、顧客固有のロールに合わせて調整する必要があります。

```
{
    "ApplicationName": "tumbling_window",
    "ApplicationDescription": "Scala tumbling window application",
    "RuntimeEnvironment": "FLINK-1_15",
    "ServiceExecutionRole": "arn:aws:iam::012345678901:role/MF-stream-rw-role",
    "ApplicationConfiguration": {
        "ApplicationCodeConfiguration": {
            "CodeContent": {
                "S3ContentLocation": {
                    "BucketARN": "arn:aws:s3:::ka-app-code-username",
                    "FileKey": "tumbling-window-scala-1.0.jar"
                }
            },
```

```
            "CodeContentType": "ZIPFILE"
        },
        "EnvironmentProperties":  {
         "PropertyGroups": [
            {
               "PropertyGroupId": "ConsumerConfigProperties",
               "PropertyMap" : {
                    "aws.region" : "us-west-2",
                    "stream.name" : "ExampleInputStream",
                    "flink.stream.initpos" : "LATEST"
               }
            },
            {
               "PropertyGroupId": "ProducerConfigProperties",
               "PropertyMap" : {
                    "aws.region" : "us-west-2",
                    "stream.name" : "ExampleOutputStream"
               }
            }
         ]
      }
    },
    "CloudWatchLoggingOptions": [
      {
         "LogStreamARN": "arn:aws:logs:us-west-2:012345678901:log-
group:MyApplication:log-stream:kinesis-analytics-log-stream"
      }
   ]
}
```

アクセス許可ポリシーを作成する step-by-step 手順については、IAM ユーザーガイドの「チュートリアル: 最初のカスタマー管理ポリシーの作成とアタッチ」を参照してください。

IAM ロールを作成する

このセクションでは、Managed Service for Apache Flink アプリケーションがソースストリームを読み取り、シンクストリームに書き込むために想定できる IAM ロールを作成します。

Apache Flink 用 Managed Service は、許可なしにはストリームにアクセスできません。IAM ロールを介してこれらの許可を付与します。各 IAM ロールには、2 つのポリシーがアタッチされます。信頼ポリシーは、ロールを引き受けるための許可を Managed Service for Apache Flink 付与し、許可ポリシーは、ロールを引き受けた後に Managed Service for Apache Flink が実行できる事柄を決定します。

前のセクションで作成したアクセス許可ポリシーをこのロールにアタッチします。

IAM ロールを作成するには

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. ナビゲーションペインで [ロール] を選択し、続いて [ロールを作成] を選択します。
3. [信頼されるエンティティの種類を選択] で、[AWS のサービス] を選択します。
4. [このロールを使用するサービスを選択] で、[Kinesis Analytics] を選択します。
5. [ユースケースの選択]で、[Managed Service for Apache Flink]を選択します。
6. [Next: Permissions] (次のステップ: 許可) を選択します。
7. [アクセス権限ポリシーをアタッチする] ページで、[Next: Review] (次: 確認) を選択します。ロールを作成した後に、アクセス許可ポリシーをアタッチします。
8. [Create role (ロールの作成)] ページで、ロールの名前に **`MF-stream-rw-role`** を入力します。[ロールの作成] を選択します。

   これで、`MF-stream-rw-role` と呼ばれる新しい IAM ロールが作成されます。次に、ロールの信頼ポリシーとアクセス許可ポリシーを更新します。
9. アクセス許可ポリシーをロールにアタッチします。

   > **Note**
   >
   > この演習では、Managed Service for Apache Flink が、Kinesis データストリーム (ソース) からのデータの読み取りと、別の Kinesis データストリームへの出力の書き込みの両方を実行するためにこのロールを引き受けます。前のステップである「許可ポリシーの作成」で作成したロールを添付します。

   a. [概要] ページで、[アクセス許可] タブを選択します。
   b. [Attach Policies (ポリシーのアタッチ)] を選択します。
   c. 検索ボックスに**`AKReadSourceStreamWriteSinkStream`**(前のセクションで作成したポリシー) と入力します。
   d. `AKReadSourceStreamWriteSinkStream`ポリシーを選択し、[ポリシーを添付]を選択します。

これで、アプリケーションがリソースにアクセスするために使用するサービスの実行ロールが作成されました。新しいロールの ARN を書き留めておきます。

ロールの作成 step-by-step 手順については、IAM ユーザーガイドの「IAM ロールの作成 (コンソール)」を参照してください。

アプリケーションの作成

次の JSON コードを create_request.json という名前のファイルに保存します。サンプルロールの ARN を、前に作成したロールの ARN に置き換えます。バケット ARN のサフィックス (ユーザー名) を、前のセクションで選択したサフィックスに置き換えます。サービス実行ロールのサンプルのアカウント ID (012345678901) を、自分のアカウント ID に置き換えます。ServiceExecutionRoleには、前のセクションで作成した IAMユーザーロールを含める必要があります。

```
"ApplicationName": "tumbling_window",
    "ApplicationDescription": "Scala getting started application",
    "RuntimeEnvironment": "FLINK-1_15",
    "ServiceExecutionRole": "arn:aws:iam::012345678901:role/MF-stream-rw-role",
    "ApplicationConfiguration": {
        "ApplicationCodeConfiguration": {
            "CodeContent": {
                "S3ContentLocation": {
                    "BucketARN": "arn:aws:s3:::ka-app-code-username",
                    "FileKey": "tumbling-window-scala-1.0.jar"
                }
            },
            "CodeContentType": "ZIPFILE"
        },
        "EnvironmentProperties":  {
         "PropertyGroups": [
            {
               "PropertyGroupId": "ConsumerConfigProperties",
               "PropertyMap" : {
                    "aws.region" : "us-west-2",
                    "stream.name" : "ExampleInputStream",
                    "flink.stream.initpos" : "LATEST"
               }
            },
            {
               "PropertyGroupId": "ProducerConfigProperties",
               "PropertyMap" : {
```

```
                    "aws.region" : "us-west-2",
                    "stream.name" : "ExampleOutputStream"
                }
            }
        ]
      }
    },
    "CloudWatchLoggingOptions": [
      {
         "LogStreamARN": "arn:aws:logs:us-west-2:012345678901:log-
group:MyApplication:log-stream:kinesis-analytics-log-stream"
      }
   ]
}
```

次のリクエストCreateApplicationで を実行してアプリケーションを作成します。

```
aws kinesisanalyticsv2 create-application --cli-input-json file://create_request.json
```

これでアプリケーションが作成されました。次のステップでは、アプリケーションを起動します。

アプリケーションを起動する

このセクションでは、 StartApplicationアクションを使用してアプリケーションを起動します。

アプリケーションを起動するには

1. 次の JSON コードを start_request.json という名前のファイルに保存します。

```
{
    "ApplicationName": "tumbling_window",
    "RunConfiguration": {
        "ApplicationRestoreConfiguration": {
         "ApplicationRestoreType": "RESTORE_FROM_LATEST_SNAPSHOT"
         }
    }
}
```

2. 前述のリクエストを指定して StartApplication アクションを実行し、アプリケーションを起動します。

```
aws kinesisanalyticsv2 start-application --cli-input-json file://start_request.json
```

アプリケーションが実行されます。Amazon CloudWatch コンソールで Managed Service for Apache Flink メトリクスをチェックして、アプリケーションが動作していることを確認できます。

アプリケーションの停止

このセクションでは、 StopApplicationアクションを使用してアプリケーションを停止します。

アプリケーションを停止するには

1. 次の JSON コードを stop_request.json という名前のファイルに保存します。

```
{
   "ApplicationName": "tumbling_window"
}
```

2. 前述のリクエストを指定して StopApplication アクションを実行し、アプリケーションを起動します。

```
aws kinesisanalyticsv2 stop-application --cli-input-json file://stop_request.json
```

アプリケーションが停止します。

CloudWatch ログ記録オプションを追加する

を使用して AWS CLI 、Amazon CloudWatch ログストリームをアプリケーションに追加できます。アプリケーションで CloudWatch ログを使用する方法については、「アプリケーションログ記録のセットアップ」を参照してください。

環境プロパティを更新する

このセクションでは、 UpdateApplicationアクションを使用して、アプリケーションコードを再コンパイルせずにアプリケーションの環境プロパティを変更します。この例では、ソースストリームおよびレプリケート先ストリームのリージョンを変更します。

アプリケーションの環境プロパティを更新します

1. 次の JSON コードを update_properties_request.json という名前のファイルに保存します。

```
{"ApplicationName": "tumbling_window",
   "CurrentApplicationVersionId": 1,
```

```
   "ApplicationConfigurationUpdate": {
      "EnvironmentPropertyUpdates": {
         "PropertyGroups": [
            {
               "PropertyGroupId": "ConsumerConfigProperties",
               "PropertyMap" : {
                    "aws.region" : "us-west-2",
                    "stream.name" : "ExampleInputStream",
                    "flink.stream.initpos" : "LATEST"
               }
            },
            {
               "PropertyGroupId": "ProducerConfigProperties",
               "PropertyMap" : {
                    "aws.region" : "us-west-2",
                    "stream.name" : "ExampleOutputStream"
               }
            }
         ]
      }
   }
}
```

2. 前述のリクエストでUpdateApplicationアクションを実行し、環境プロパティを更新します。

```
aws kinesisanalyticsv2 update-application --cli-input-json file://
update_properties_request.json
```

アプリケーションコードの更新

アプリケーションコードを新しいバージョンのコードパッケージで更新する必要がある場合は、 CLI UpdateApplication アクションを使用します。

> **Note**
>
> 同じファイル名のアプリケーションコードの新しいバージョンをロードするには、新しいオブジェクトバージョンを指定する必要があります。Amazon S3 オブジェクトバージョンを使用する方法の詳細については、「バージョニングの有効化または無効化」を参照してください。

を使用するには AWS CLI、Amazon S3 バケットから以前のコードパッケージを削除し、新しいバージョンをアップロードして、同じ Amazon S3 バケットとオブジェクト名、および新しいオブジェクトバージョンUpdateApplicationを指定して を呼び出します。アプリケーションは新しいコードパッケージで再起動します。

以下の UpdateApplication アクションのサンプル・リクエストは、アプリケーション・コードを再読み込 み、アプリケーションを再起動します。CurrentApplicationVersionId を現在のアプリケーションバージョンに更新します。ListApplications または DescribeApplication アクションを使用して、現在のアプリケーションバージョンを確認できます。バケット名のサフィックス (「<username>」) を、 依存リソースを作成する セクションで選択したサフィックスで更新します。

```
{
    "ApplicationName": "tumbling_window",
    "CurrentApplicationVersionId": 1,
    "ApplicationConfigurationUpdate": {
        "ApplicationCodeConfigurationUpdate": {
            "CodeContentUpdate": {
                "S3ContentLocationUpdate": {
                    "BucketARNUpdate": "arn:aws:s3:::ka-app-code-username",
                    "FileKeyUpdate": "tumbling-window-scala-1.0.jar",
                    "ObjectVersionUpdate": "SAMPLEUehYngP87ex1nzYIGYgfhypvDU"
                }
            }
        }
    }
}
```

AWS リソースをクリーンアップする

このセクションでは、タンブリングウィンドウチュートリアルで作成された AWS リソースをクリーンアップする手順について説明します。

このトピックには、次のセクションが含まれています。

- Managed Service for Apache Flink アプリケーションを削除する
- Kinesis データストリームを削除する
- Amazon S3 オブジェクトとバケットを削除する
- IAM リソースを削除する
- CloudWatch リソースを削除する

Managed Service for Apache Flink アプリケーションを削除する

1. https://console.aws.amazon.com/flink で Apache Flink 用 Managed Serviceコンソールを開く
2. Managed Service for Apache Flink パネルで、 を選択しますMyApplication。
3. アプリケーションのページで[削除]を選択し、削除を確認します。

Kinesis データストリームを削除する

1. 「https://console.aws.amazon.com/kinesis」で Kinesis コンソールを開きます。
2. Kinesis Data Streams パネルで、 を選択しますExampleInputStream。
3. ExampleInputStream ページで、Kinesis ストリームの削除を選択し、削除を確定します。
4. Kinesis ストリームページで、 を選択しExampleOutputStream、アクション を選択し、削除 を選択し、削除を確定します。

Amazon S3 オブジェクトとバケットを削除する

1. https://console.aws.amazon.com/s3/でAmazon S3 コンソールを開きます。
2. ka-app-code-***<username>*** バケットを選択します。
3. [削除] を選択し、バケット名を入力して削除を確認します。

IAM リソースを削除する

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. ナビゲーションバーで、[ポリシー] を選択します。
3. フィルターコントロールに「kinesis」と入力します。
4. kinesis-analytics-service-MyApplication-us-west-2 ポリシーを選択します。
5. [ポリシーアクション]、[削除] の順に選択します。
6. ナビゲーションバーで [ロール] を選択します。
7. kinesis-analytics-MyApplication-us-west-2 ロールを選択します。
8. [ロールの削除] を選択し、削除を確定します。

CloudWatch リソースを削除する

1. https://console.aws.amazon.com/cloudwatch/ で CloudWatch コンソールを開きます。

2. ナビゲーションバーで [ログ] を選択します。
3. /aws/kinesis-analytics/MyApplication ロググループを選択します。
4. [ロググループの削除]を選択し、削除を確認してください。

例: Scala でのスライディングウィンドウの作成

> **Note**
>
> 現在の例については、「」を参照してください例。

> **Note**
>
> バージョン 1.15 以降、Flink は Scala フリーになりました。アプリケーションが Scala の任意のバージョンから Java API を使用できるようになっています。Flink は今でも内部的にいくつかの主要コンポーネントで Scala を使用していますが、Scala をユーザーコードのクラスローダーに公開していません。そのため、ユーザーは Scala の依存関係を自分の JAR アーカイブに追加する必要があります。
> Flink 1.15 での Scala の変更についての詳しい情報は、Scala Free in One Fifteenを参照してください。

この演習では、Scala 3.2.0 と Flink の Java DataStream API を使用するシンプルなストリーミングアプリケーションを作成します。アプリケーションは Kinesis ストリームからデータを読み取り、スライディングウィンドウを使用して集約し、結果を出力Kinesisストリームに書き込みます。

> **Note**
>
> この練習に必要な前提条件を設定するには、まずGetting Started (Scala)の練習を完了してください。

このトピックには、次のセクションが含まれています。

- アプリケーションコードをダウンロードして調べる
- アプリケーション・コードをコンパイルしてアップロードするには
- アプリケーションの作成と実行 (コンソール)

- アプリケーションの作成と実行 (CLI)
- アプリケーションコードの更新
- AWS リソースをクリーンアップする

アプリケーションコードをダウンロードして調べる

この例の Python アプリケーションコードは、 から入手できます GitHub。アプリケーションコードをダウンロードするには、次の操作を行います。

1. Git クライアントをまだインストールしていない場合は、インストールします。詳細については、「Git のインストール」をご参照ください。
2. 次のコマンドを使用してリモートリポジトリのクローンを作成します。

```
git clone https://github.com/aws-samples/amazon-kinesis-data-analytics-examples.git
```

3. amazon-kinesis-data-analytics-java-examples/scala/SlidingWindow ディレクトリに移動します。

アプリケーションコードに関して、以下の点に注意してください。

- build.sbtファイルには、Managed Service for Apache Flink ライブラリなど、アプリケーションの設定と依存関係に関する情報が含まれています。
- この BasicStreamingJob.scala ファイルには、アプリケーションの機能を定義するメインメソッドが含まれています。
- アプリケーションは Kinesis ソースを使用して、ソースストリームから読み取りを行います。次のスニペットでは、Kinesis ソースが作成されます。

```
private def createSource: FlinkKinesisConsumer[String] = {
  val applicationProperties = KinesisAnalyticsRuntime.getApplicationProperties
  val inputProperties = applicationProperties.get("ConsumerConfigProperties")

  new FlinkKinesisConsumer[String](inputProperties.getProperty(streamNameKey,
 defaultInputStreamName),
    new SimpleStringSchema, inputProperties)
}
```

また、アプリケーションは Kinesis シンクを使用して結果ストリームに書き込みます。次のスニペットでは、Kinesis シンクが作成されます。

```
private def createSink: KinesisStreamsSink[String] = {
  val applicationProperties = KinesisAnalyticsRuntime.getApplicationProperties
  val outputProperties = applicationProperties.get("ProducerConfigProperties")

  KinesisStreamsSink.builder[String]
    .setKinesisClientProperties(outputProperties)
    .setSerializationSchema(new SimpleStringSchema)
    .setStreamName(outputProperties.getProperty(streamNameKey,
 defaultOutputStreamName))
    .setPartitionKeyGenerator((element: String) => String.valueOf(element.hashCode))
    .build
}
```

- アプリケーションはウィンドウ演算子を使用して、5 秒ずつスライドする 10 秒間のウィンドウ内の各銘柄記号の値を求めます。次のコードは演算子を作成し、集約されたデータを新しい Kinesis Data Streamsシンクに送信します。

```
environment.addSource(createSource)
    .map { value =>
      val jsonNode = jsonParser.readValue(value, classOf[JsonNode])
      new Tuple2[String, Double](jsonNode.get("ticker").toString,
 jsonNode.get("price").asDouble)
    }
    .returns(Types.TUPLE(Types.STRING, Types.DOUBLE))
    .keyBy(v => v.f0) // Logically partition the stream for each word
    .window(SlidingProcessingTimeWindows.of(Time.seconds(10), Time.seconds(5)))
    .min(1) // Calculate minimum price per ticker over the window
    .map { value => value.f0 + String.format(",%.2f", value.f1) + "\n" }
    .sinkTo(createSink)
```

- アプリケーションは、 StreamExecutionEnvironment オブジェクトを使用して外部リソースにアクセスするためのソースコネクタとシンクコネクタを作成します。
- アプリケーションは、動的アプリケーションプロパティを使用してソースコネクタとシンクコネクタを作成します。アプリケーションのプロパティを読み取ってコネクタを設定します。ランタイムプロパティの詳細については、ランタイムプロパティを参照してください。

アプリケーション· コードをコンパイルしてアップロードするには

このセクションでは、アプリケーションコードをコンパイルして Amazon S3 バケットにアップロードします。

アプリケーションコードのコンパイル

SBT ビルドツールを使用して、アプリケーションの Scala コードをビルドします。SBTをインストールするには、Install sbt with cs setupを参照してください。また、Java 開発キット (JDK) をインストールする必要があります。演習を完了するための前提条件 を参照してください。

1. アプリケーションコードを使用するには、コードをコンパイルして JAR ファイルにパッケージ化します。SBT を使用してコードをコンパイルしてパッケージ化できます。

```
sbt assembly
```

2. アプリケーションのコンパイルに成功すると、次のファイルが作成されます。

```
target/scala-3.2.0/sliding-window-scala-1.0.jar
```

Apache Flink Streaming Scala Code のアップロード

このセクションでは、Amazon S3 バケットを作成し、アプリケーションコードをアップロードします。

1. https://console.aws.amazon.com/s3/でAmazon S3 コンソールを開きます。
2. [バケットを作成] を選択します。
3. [Bucket name (バケット名)] フィールドにka-app-code-<username>と入力します。バケット名にユーザー名などのサフィックスを追加して、グローバルに一意にします。[次へ] をクリックします。
4. 設定オプションのステップでは、設定をそのままにし、[次へ] を選択します。
5. アクセス許可の設定のステップでは、設定をそのままにし、[次へ] を選択します。
6. [バケットを作成] を選択します。
7. ka-app-code-<username>バケットを選択し、アップロード を選択します。
8. ファイルの選択のステップで、[ファイルを追加] を選択します。前のステップで作成した sliding-window-scala-1.0.jar ファイルに移動します。
9. オブジェクトの設定を変更する必要はないので、[アップロード] を選択してください。

アプリケーションコードが Amazon S3 バケットに保存され、アプリケーションからアクセスできるようになります。

アプリケーションの作成と実行 (コンソール)

以下の手順を実行し、コンソールを使用してアプリケーションを作成、設定、更新、および実行します。

アプリケーションの作成

1. https://console.aws.amazon.com/flink で Apache Flink 用 Managed Serviceコンソールを開く
2. Managed Service for Apache Flinkのダッシュボードで、「分析アプリケーションの作成」 を選択します。
3. 「Managed Service for Apache Flink-アプリケーションの作成」ページで、次のようにアプリケーションの詳細を入力します。
   - [アプリケーション名] には **MyApplication** と入力します。
   - [Description (説明)] に **My Scala test app** と入力します。
   - [ランタイム] には、[Apache Flink] を選択します。
   - バージョンはApache Flink バージョン 1.15.2 (推薦バージョン)のままにしておきます。
4. [アクセス許可] には、[IAM ロールの作成 / 更新**kinesis-analytics-MyApplication-us-west-2**] を選択します。
5. [Create application] を選択します。

> **Note**
>
> コンソールを使用して Apache Flink アプリケーション用 Managed Service を作成する場合は、IAM ロールとポリシーをアプリケーションが自動的に作成するオプションを選択できます。アプリケーションではこのロールとポリシーを使用して、依存リソースにアクセスします。これらの IAM リソースは、次のようにアプリケーション名とリージョンを使用して命名されます。
>
> - ポリシー: `kinesis-analytics-service-MyApplication-us-west-2`
> - ロール: `kinesisanalytics-MyApplication-us-west-2`

アプリケーションを設定する

アプリケーションを設定するには、次の手順に従います。

アプリケーションを構成するには

1. MyApplication ページで、 の設定 を選択します。
2. [Configure application] ページで、[Code location] を次のように指定します。
   - [Amazon S3 バケット] で、**ka-app-code-*<username>***と入力します。
   - [Amazon S3 オブジェクトへのパス] で、**sliding-window-scala-1.0.jar.**と入力します。
3. [Access to application resources] の [Access permissions] では、[Create / update IAM role**kinesis-analytics-MyApplication-us-west-2**] を選択します。
4. [プロパティ] で [グループの追加]を選択します。
5. 次のように入力します。

グループ ID	キー	値
ConsumerConfigProperties	**input.stream.name**	**ExampleInputStream**
ConsumerConfigProperties	**aws.region**	**us-west-2**
ConsumerConfigProperties	**flink.stream.initpos**	**LATEST**

   [保存] を選択します。
6. プロパティ で、グループの追加をもう一度選択します。
7. 次のように入力します。

グループ ID	キー	値
ProducerConfigProperties	**output.stream.name**	**ExampleOutputStream**
ProducerConfigProperties	**aws.region**	**us-west-2**

8. [Monitoring] の [Monitoring metrics level] が [Application] に設定されていることを確認します。
9. CloudWatch のログ記録では、有効化チェックボックスを選択します。
10. [更新] を選択します。

> **Note**
>
> Amazon CloudWatch ログ記録を有効にすると、Managed Service for Apache Flink によってロググループとログストリームが作成されます。これらのリソースの名前は次のとおりです。
>
> - ロググループ: `/aws/kinesis-analytics/MyApplication`
> - ログストリーム: `kinesis-analytics-log-stream`

IAM ポリシーを編集する

Amazon S3 バケットにアクセスする許可を追加するように IAM ポリシーを編集します。

IAM ポリシーを編集して S3 バケット権限を追加するには

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. [ポリシー] を選択します。前のセクションでコンソールによって作成された **kinesis-analytics-service-MyApplication-us-west-2** ポリシーを選択します。
3. [概要] ページで、[ポリシーの編集] を選択します。[JSON] タブを選択します。
4. 次のポリシー例で強調表示されているセクションをポリシーに追加します。サンプルのアカウント ID (*012345678901*) を自分のアカウント ID に置き換えます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ReadCode",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": [
```

```
                "arn:aws:s3:::ka-app-code-username/sliding-window-scala-1.0.jar"
            ]
        },
        {
            "Sid": "DescribeLogGroups",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:*"
            ]
        },
        {
            "Sid": "DescribeLogStreams",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogStreams"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:*"
            ]
        },
        {
            "Sid": "PutLogEvents",
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:kinesis-analytics-log-stream"
            ]
        },
        {
            "Sid": "ReadInputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
        },
        {
            "Sid": "WriteOutputStream",
```

```
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleOutputStream"
        }
    ]
}
```

アプリケーションを実行する

Flink ジョブグラフは、アプリケーションを実行し、Apache Flink ダッシュボードを開き、目的の Flink ジョブを選択すると表示できます。

アプリケーションの停止

アプリケーションを停止するには、MyApplicationページで停止 を選択します。アクションを確認します。

アプリケーションの作成と実行 (CLI)

このセクションでは、 を使用して Managed Service for Apache Flink アプリケーション AWS Command Line Interface を作成して実行します。kinesisanalyticsv2 AWS CLI コマンドを使用して、Managed Service for Apache Flink アプリケーションを作成して操作します。

許可ポリシーを作成する

> **Note**
>
> アプリケーションのアクセス権限ポリシーとロールを作成する必要があります。これらの IAM リソースを作成しない場合、アプリケーションはそのデータストリームやログストリームにアクセスできません。

まず、2 つのステートメントを含むアクセス許可ポリシーを作成します。1 つは、ソースストリームの読み取りアクションに対するアクセス許可を付与し、もう 1 つはシンクストリームの書き込みアクションに対するアクセス許可を付与します。次に、IAM ロール (次のセクションで作成) にポリシーをアタッチします。そのため、 Managed Service for Apache Flinkがこのロールを引き受けると、ソースストリームからの読み取りとシンクストリームへの書き込みを行うために必要なアクセス許可がサービスに付与されます。

次のコードを使用して AKReadSourceStreamWriteSinkStream アクセス許可ポリシーを作成します。**username** を Amazon S3 バケットの作成に使用したユーザー名に置き換え、アプリケーションコードを保存します。Amazon リソースネーム (ARN) **(012345678901)**のアカウント ID を自分のアカウント ID に置き換えます。

```
{
    "ApplicationName": "sliding_window",
    "ApplicationDescription": "Scala sliding window application",
    "RuntimeEnvironment": "FLINK-1_15",
    "ServiceExecutionRole": "arn:aws:iam::012345678901:role/MF-stream-rw-role",
    "ApplicationConfiguration": {
        "ApplicationCodeConfiguration": {
            "CodeContent": {
                "S3ContentLocation": {
                    "BucketARN": "arn:aws:s3:::ka-app-code-username",
                    "FileKey": "sliding-window-scala-1.0.jar"
                }
            },
            "CodeContentType": "ZIPFILE"
        },
        "EnvironmentProperties":  {
         "PropertyGroups": [
            {
               "PropertyGroupId": "ConsumerConfigProperties",
               "PropertyMap" : {
                    "aws.region" : "us-west-2",
                    "stream.name" : "ExampleInputStream",
                    "flink.stream.initpos" : "LATEST"
               }
            },
            {
               "PropertyGroupId": "ProducerConfigProperties",
               "PropertyMap" : {
                    "aws.region" : "us-west-2",
                    "stream.name" : "ExampleOutputStream"
               }
            }
         ]
      }
    },
    "CloudWatchLoggingOptions": [
      {
```

```
        "LogStreamARN": "arn:aws:logs:us-west-2:012345678901:log-
group:MyApplication:log-stream:kinesis-analytics-log-stream"
      }
   ]
}
```

アクセス許可ポリシーを作成する step-by-step 手順については、IAM ユーザーガイドの「チュートリアル: 最初のカスタマー管理ポリシーの作成とアタッチ」を参照してください。

IAM ロールを作成する

このセクションでは、Managed Service for Apache Flink アプリケーションがソースストリームを読み取り、シンクストリームに書き込むために想定できる IAM ロールを作成します。

Apache Flink 用 Managed Service は、許可なしにはストリームにアクセスできません。IAM ロールを介してこれらの許可を付与します。各 IAM ロールには、2 つのポリシーがアタッチされます。信頼ポリシーは、ロールを引き受けるための許可を Managed Service for Apache Flink 付与し、許可ポリシーは、ロールを引き受けた後に Managed Service for Apache Flink が実行できる事柄を決定します。

前のセクションで作成したアクセス許可ポリシーをこのロールにアタッチします。

IAM ロールを作成するには

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. ナビゲーションペインで [ロール] を選択し、続いて [ロールを作成] を選択します。
3. [信頼されるエンティティの種類を選択] で、[AWS のサービス] を選択します。
4. [このロールを使用するサービスを選択] で、[Kinesis Analytics] を選択します。
5. [ユースケースの選択]で、[Managed Service for Apache Flink]を選択します。
6. [Next: Permissions] (次のステップ: 許可) を選択します。
7. [アクセス権限ポリシーをアタッチする] ページで、[Next: Review] (次: 確認) を選択します。ロールを作成した後に、アクセス許可ポリシーをアタッチします。
8. [Create role (ロールの作成)] ページで、ロールの名前に **`MF-stream-rw-role`** を入力します。[ロールの作成] を選択します。

   これで、`MF-stream-rw-role` と呼ばれる新しい IAM ロールが作成されます。次に、ロールの信頼ポリシーとアクセス許可ポリシーを更新します。

9. アクセス許可ポリシーをロールにアタッチします。

> **Note**
>
> この演習では、Managed Service for Apache Flink が、Kinesis データストリーム (ソース) からのデータの読み取りと、別の Kinesis データストリームへの出力の書き込みの両方を実行するためにこのロールを引き受けます。前のステップである「許可ポリシーの作成」で作成したロールを添付します。

a. [概要] ページで、[アクセス許可] タブを選択します。
b. [Attach Policies (ポリシーのアタッチ)] を選択します。
c. 検索ボックスに**AKReadSourceStreamWriteSinkStream**(前のセクションで作成したポリシー) と入力します。
d. AKReadSourceStreamWriteSinkStreamポリシーを選択し、[ポリシーを添付]を選択します。

これで、アプリケーションがリソースにアクセスするために使用するサービスの実行ロールが作成されました。新しいロールの ARN を書き留めておきます。

ロールの作成 step-by-step 手順については、IAM ユーザーガイドの「IAM ロールの作成 (コンソール)」を参照してください。

アプリケーションの作成

次の JSON コードを create_request.json という名前のファイルに保存します。サンプルロールの ARN を、前に作成したロールの ARN に置き換えます。バケット ARN のサフィックス (ユーザー名) を、前のセクションで選択したサフィックスに置き換えます。サービス実行ロールのサンプルのアカウント ID (012345678901) を、自分のアカウント ID に置き換えます。

```
{
    "ApplicationName": "sliding_window",
    "ApplicationDescription": "Scala sliding_window application",
    "RuntimeEnvironment": "FLINK-1_15",
    "ServiceExecutionRole": "arn:aws:iam::012345678901:role/MF-stream-rw-role",
    "ApplicationConfiguration": {
        "ApplicationCodeConfiguration": {
            "CodeContent": {
                "S3ContentLocation": {
```

```
                    "BucketARN": "arn:aws:s3:::ka-app-code-username",
                    "FileKey": "sliding-window-scala-1.0.jar"
                }
            },
            "CodeContentType": "ZIPFILE"
        },
        "EnvironmentProperties":  {
         "PropertyGroups": [
            {
               "PropertyGroupId": "ConsumerConfigProperties",
               "PropertyMap" : {
                    "aws.region" : "us-west-2",
                    "stream.name" : "ExampleInputStream",
                    "flink.stream.initpos" : "LATEST"
               }
            },
            {
               "PropertyGroupId": "ProducerConfigProperties",
               "PropertyMap" : {
                    "aws.region" : "us-west-2",
                    "stream.name" : "ExampleOutputStream"
               }
            }
         ]
      }
    },
    "CloudWatchLoggingOptions": [
      {
         "LogStreamARN": "arn:aws:logs:us-west-2:012345678901:log-
group:MyApplication:log-stream:kinesis-analytics-log-stream"
      }
   ]
}
```

次のリクエストCreateApplicationで を実行してアプリケーションを作成します。

```
aws kinesisanalyticsv2 create-application --cli-input-json file://create_request.json
```

これでアプリケーションが作成されました。次のステップでは、アプリケーションを起動します。

アプリケーションを起動する

このセクションでは、 StartApplicationアクションを使用してアプリケーションを起動します。

アプリケーションを起動するには

1. 次の JSON コードを start_request.json という名前のファイルに保存します。

```
{
    "ApplicationName": "sliding_window",
    "RunConfiguration": {
        "ApplicationRestoreConfiguration": {
         "ApplicationRestoreType": "RESTORE_FROM_LATEST_SNAPSHOT"
         }
    }
}
```

2. 前述のリクエストを指定して StartApplication アクションを実行し、アプリケーションを起動します。

```
aws kinesisanalyticsv2 start-application --cli-input-json file://start_request.json
```

アプリケーションが実行されます。Amazon CloudWatch コンソールで Managed Service for Apache Flink メトリクスをチェックして、アプリケーションが動作していることを確認できます。

アプリケーションの停止

このセクションでは、 StopApplicationアクションを使用してアプリケーションを停止します。

アプリケーションを停止するには

1. 次の JSON コードを stop_request.json という名前のファイルに保存します。

```
{
   "ApplicationName": "sliding_window"
}
```

2. 前述のリクエストを指定して StopApplication アクションを実行し、アプリケーションを起動します。

```
aws kinesisanalyticsv2 stop-application --cli-input-json file://stop_request.json
```

アプリケーションが停止します。

CloudWatch ログ記録オプションを追加する

を使用して AWS CLI 、Amazon CloudWatch ログストリームをアプリケーションに追加できます。アプリケーションで CloudWatch ログを使用する方法については、「アプリケーションログ記録のセットアップ」を参照してください。

環境プロパティを更新する

このセクションでは、 UpdateApplicationアクションを使用して、アプリケーションコードを再コンパイルせずにアプリケーションの環境プロパティを変更します。この例では、ソースストリームおよびレプリケート先ストリームのリージョンを変更します。

アプリケーションの環境プロパティを更新します

1. 次の JSON コードを update_properties_request.json という名前のファイルに保存します。

```
{"ApplicationName": "sliding_window",
   "CurrentApplicationVersionId": 1,
   "ApplicationConfigurationUpdate": {
      "EnvironmentPropertyUpdates": {
         "PropertyGroups": [
            {
               "PropertyGroupId": "ConsumerConfigProperties",
               "PropertyMap" : {
                    "aws.region" : "us-west-2",
                    "stream.name" : "ExampleInputStream",
                    "flink.stream.initpos" : "LATEST"
               }
            },
            {
               "PropertyGroupId": "ProducerConfigProperties",
               "PropertyMap" : {
                    "aws.region" : "us-west-2",
                    "stream.name" : "ExampleOutputStream"
               }
            }
         ]
      }
   }
}
```

2. 前述のリクエストでUpdateApplicationアクションを実行し、環境プロパティを更新します。

```
aws kinesisanalyticsv2 update-application --cli-input-json file://
update_properties_request.json
```

アプリケーションコードの更新

アプリケーションコードを新しいバージョンのコードパッケージで更新する必要がある場合は、 CLI UpdateApplication アクションを使用します。

> **Note**
>
> 同じファイル名のアプリケーションコードの新しいバージョンをロードするには、新しいオブジェクトバージョンを指定する必要があります。Amazon S3 オブジェクトバージョンを使用する方法の詳細については、「バージョニングの有効化または無効化」を参照してください。

を使用するには AWS CLI、Amazon S3 バケットから以前のコードパッケージを削除し、新しいバージョンをアップロードして、同じ Amazon S3 バケットとオブジェクト名、および新しいオブジェクトバージョンUpdateApplicationを指定して を呼び出します。アプリケーションは新しいコードパッケージで再起動します。

以下の UpdateApplication アクションのサンプル・リクエストは、アプリケーション・コードを再読み込 み、アプリケーションを再起動します。CurrentApplicationVersionId を現在のアプリケーションバージョンに更新します。ListApplications または DescribeApplication アクションを使用して、現在のアプリケーションバージョンを確認できます。バケット名のサフィックス (「<username>」) を、 依存リソースを作成する セクションで選択したサフィックスで更新します。

```
{
    "ApplicationName": "sliding_window",
    "CurrentApplicationVersionId": 1,
    "ApplicationConfigurationUpdate": {
        "ApplicationCodeConfigurationUpdate": {
            "CodeContentUpdate": {
                "S3ContentLocationUpdate": {
```

```
                    "BucketARNUpdate": "arn:aws:s3:::ka-app-code-username",
                    "FileKeyUpdate": "-1.0.jar",
                    "ObjectVersionUpdate": "SAMPLEUehYngP87ex1nzYIGYgfhypvDU"
                }
            }
        }
    }
}
```

AWS リソースをクリーンアップする

このセクションでは、スライディングウィンドウチュートリアルで作成した AWS リソースをクリーンアップする手順について説明します。

このトピックには、次のセクションが含まれています。

- Managed Service for Apache Flink アプリケーションを削除する
- Kinesis データストリームを削除する
- Amazon S3 オブジェクトとバケットを削除する
- IAM リソースを削除する
- CloudWatch リソースを削除する

Managed Service for Apache Flink アプリケーションを削除する

1. https://console.aws.amazon.com/flink で Apache Flink 用 Managed Serviceコンソールを開く
2. Managed Service for Apache Flink パネルで、 を選択しますMyApplication。
3. アプリケーションのページで[削除]を選択し、削除を確認します。

Kinesis データストリームを削除する

1. 「https://console.aws.amazon.com/kinesis」で Kinesis コンソールを開きます。
2. Kinesis Data Streams パネルで、 を選択しますExampleInputStream。
3. ExampleInputStream ページで、Kinesis Streams の削除を選択し、削除を確定します。
4. Kinesis ストリームページで、 を選択しExampleOutputStream、アクション を選択し、削除 を選択し、削除を確定します。

Amazon S3 オブジェクトとバケットを削除する

1. https://console.aws.amazon.com/s3/でAmazon S3 コンソールを開きます。
2. ka-app-code-***<username>*** バケットを選択します。
3. [削除] を選択し、バケット名を入力して削除を確認します。

IAM リソースを削除する

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. ナビゲーションバーで、[ポリシー] を選択します。
3. フィルターコントロールに「kinesis」と入力します。
4. kinesis-analytics-service-MyApplication-us-west-2 ポリシーを選択します。
5. [ポリシーアクション]、[削除] の順に選択します。
6. ナビゲーションバーで [ロール] を選択します。
7. kinesis-analytics-MyApplication-us-west-2 ロールを選択します。
8. [ロールの削除] を選択し、削除を確定します。

CloudWatch リソースを削除する

1. https://console.aws.amazon.com/cloudwatch/ で CloudWatch コンソールを開きます。
2. ナビゲーションバーで [ログ] を選択します。
3. /aws/kinesis-analytics/MyApplication ロググループを選択します。
4. [ロググループの削除]を選択し、削除を確認してください。

例: Scala で Amazon S3 にストリーミングデータを送信する

> **Note**
> 現在の例については、「」を参照してください例。

> **Note**
> バージョン 1.15 以降、Flink は Scala フリーになりました。アプリケーションが Scala の任意のバージョンから Java API を使用できるようになっています。Flink は今でも内部的にい

くつかの主要コンポーネントで Scala を使用していますが、Scala をユーザーコードのクラスローダーに公開していません。そのため、ユーザーは Scala の依存関係を自分の JAR アーカイブに追加する必要があります。
Flink 1.15 での Scala の変更についての詳しい情報は、Scala Free in One Fifteenを参照してください。

この演習では、Scala 3.2.0 と Flink の Java DataStream API を使用するシンプルなストリーミングアプリケーションを作成します。アプリケーションは Kinesis ストリームからデータを読み取り、スライディングウィンドウを使用してデータを集約し、結果を S3 に書き込みます。

> **Note**
>
> この練習に必要な前提条件を設定するには、まずGetting Started (Scala) の練習を完了してください。Amazon S3 バケット ka-app-code-<username> **data/**に追加フォルダを作成するだけで済みます。

このトピックには、次のセクションが含まれています。

- アプリケーションコードをダウンロードして調べる
- アプリケーション・コードをコンパイルしてアップロードするには
- アプリケーションの作成と実行 (コンソール)
- アプリケーションの作成と実行 (CLI)
- アプリケーションコードの更新
- AWS リソースをクリーンアップする

アプリケーションコードをダウンロードして調べる

この例の Python アプリケーションコードは、 から入手できます GitHub。アプリケーションコードをダウンロードするには、次の操作を行います。

1. Git クライアントをまだインストールしていない場合は、インストールします。詳細については、「Git のインストール」をご参照ください。
2. 次のコマンドを使用してリモートリポジトリのクローンを作成します。

```
git clone https://github.com/aws-samples/amazon-kinesis-data-analytics-examples.git
```

3. amazon-kinesis-data-analytics-java-examples/scala/S3Sink ディレクトリに移動します。

アプリケーションコードに関して、以下の点に注意してください。

- build.sbtファイルには、Managed Service for Apache Flink ライブラリなど、アプリケーションの設定と依存関係に関する情報が含まれています。
- この BasicStreamingJob.scala ファイルには、アプリケーションの機能を定義するメインメソッドが含まれています。
- アプリケーションは Kinesis ソースを使用して、ソースストリームから読み取りを行います。次のスニペットでは、Kinesis ソースが作成されます。

```
private def createSource: FlinkKinesisConsumer[String] = {
  val applicationProperties = KinesisAnalyticsRuntime.getApplicationProperties
  val inputProperties = applicationProperties.get("ConsumerConfigProperties")

  new FlinkKinesisConsumer[String](inputProperties.getProperty(streamNameKey,
 defaultInputStreamName),
    new SimpleStringSchema, inputProperties)
}
```

また、アプリケーションは を使用して Amazon S3 バケットに StreamingFileSink 書き込みます。

```
def createSink: StreamingFileSink[String] = {
  val applicationProperties = KinesisAnalyticsRuntime.getApplicationProperties
  val s3SinkPath =
 applicationProperties.get("ProducerConfigProperties").getProperty("s3.sink.path")

  StreamingFileSink
    .forRowFormat(new Path(s3SinkPath), new SimpleStringEncoder[String]("UTF-8"))
    .build()
}
```

- アプリケーションは、 StreamExecutionEnvironment オブジェクトを使用して外部リソースにアクセスするためのソースコネクタとシンクコネクタを作成します。
- アプリケーションは、動的アプリケーションプロパティを使用してソースコネクタとシンクコネクタを作成します。アプリケーションのプロパティを読み取ってコネクタを設定します。ランタイムプロパティの詳細については、ランタイムプロパティを参照してください。

アプリケーション· コードをコンパイルしてアップロードするには

このセクションでは、アプリケーションコードをコンパイルして Amazon S3 バケットにアップロードします。

アプリケーションコードのコンパイル

SBT ビルドツールを使用して、アプリケーションの Scala コードをビルドします。SBTをインストールするには、Install sbt with cs setupを参照してください。また、Java 開発キット (JDK) をインストールする必要があります。演習を完了するための前提条件 を参照してください。

1. アプリケーションコードを使用するには、コードをコンパイルして JAR ファイルにパッケージ化します。SBT を使用してコードをコンパイルしてパッケージ化できます。

```
sbt assembly
```

2. アプリケーションのコンパイルに成功すると、次のファイルが作成されます。

```
target/scala-3.2.0/s3-sink-scala-1.0.jar
```

Apache Flink Streaming Scala Code のアップロード

このセクションでは、Amazon S3 バケットを作成し、アプリケーションコードをアップロードします。

1. https://console.aws.amazon.com/s3/でAmazon S3 コンソールを開きます。
2. [バケットを作成] を選択します。
3. [Bucket name (バケット名)] フィールドにka-app-code-<username>と入力します。バケット名にユーザー名などのサフィックスを追加して、グローバルに一意にします。[次へ] をクリックします。
4. 設定オプションのステップでは、設定をそのままにし、[次へ] を選択します。
5. アクセス許可の設定のステップでは、設定をそのままにし、[次へ] を選択します。
6. [バケットを作成] を選択します。
7. ka-app-code-<username>バケットを選択し、アップロード を選択します。
8. ファイルの選択のステップで、[ファイルを追加] を選択します。前のステップで作成した s3-sink-scala-1.0.jar ファイルに移動します。

9. オブジェクトの設定を変更する必要はないので、[アップロード] を選択してください。

アプリケーションコードが Amazon S3 バケットに保存され、アプリケーションからアクセスできるようになります。

アプリケーションの作成と実行 (コンソール)

以下の手順を実行し、コンソールを使用してアプリケーションを作成、設定、更新、および実行します。

アプリケーションの作成

1. https://console.aws.amazon.com/flink で Apache Flink 用 Managed Serviceコンソールを開く
2. Managed Service for Apache Flinkのダッシュボードで、「分析アプリケーションの作成」 を選択します。
3. 「Managed Service for Apache Flink-アプリケーションの作成」ページで、次のようにアプリケーションの詳細を入力します。
   - [アプリケーション名] には **MyApplication** と入力します。
   - [Description (説明)] に **My java test app** と入力します。
   - [ランタイム] には、[Apache Flink] を選択します。
   - バージョンはApache Flink バージョン 1.15.2 (推薦バージョン)のままにしておきます。
4. [アクセス許可] には、[IAM ロールの作成 / 更新**kinesis-analytics-MyApplication-us-west-2**] を選択します。
5. [Create application] を選択します。

> **Note**
>
> コンソールを使用して Apache Flink アプリケーション用 Managed Service を作成する場合は、IAM ロールとポリシーをアプリケーションが自動的に作成するオプションを選択できます。アプリケーションではこのロールとポリシーを使用して、依存リソースにアクセスします。これらの IAM リソースは、次のようにアプリケーション名とリージョンを使用して命名されます。
>
> - ポリシー: `kinesis-analytics-service-MyApplication-us-west-2`
> - ロール: `kinesisanalytics-MyApplication-us-west-2`

アプリケーションを設定する

アプリケーションを設定するには、次の手順に従います。

アプリケーションを構成するには

1. MyApplication ページで、 の設定 を選択します。
2. [Configure application] ページで、[Code location] を次のように指定します。
   - [Amazon S3 バケット] で、**ka-app-code-*<username>***と入力します。
   - [Amazon S3 オブジェクトへのパス] で、**s3-sink-scala-1.0.jar**と入力します。
3. [Access to application resources] の [Access permissions] では、[Create / update IAM role**kinesis-analytics-MyApplication-us-west-2**] を選択します。
4. [プロパティ] で [グループの追加]を選択します。
5. 次のように入力します。

グループ ID	キー	値
ConsumerConfigProperties	**input.stream.name**	**ExampleInputStream**
ConsumerConfigProperties	**aws.region**	**us-west-2**
ConsumerConfigProperties	**flink.stream.initpos**	**LATEST**

   [保存] を選択します。
6. [プロパティ] で [グループの追加]を選択します。す。
7. 次のように入力します。

グループ ID	キー	値
ProducerConfigProperties	**s3.sink.path**	**s3a://ka-app-code-*<user-name>* /data**

8. [Monitoring] の [Monitoring metrics level] が [Application] に設定されていることを確認します。

9. CloudWatch ログ記録 で、有効化チェックボックスを選択します。
10. [更新] を選択します。

> **Note**
>
> Amazon CloudWatch ログ記録を有効にすると、Managed Service for Apache Flink によってロググループとログストリームが作成されます。これらのリソースの名前は次のとおりです。
>
> - ロググループ: `/aws/kinesis-analytics/MyApplication`
> - ログストリーム: `kinesis-analytics-log-stream`

IAM ポリシーを編集する

Amazon S3 バケットにアクセスする許可を追加するように IAM ポリシーを編集します。

IAM ポリシーを編集して S3 バケット権限を追加するには

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. [ポリシー] を選択します。前のセクションでコンソールによって作成された **kinesis-analytics-service-MyApplication-us-west-2** ポリシーを選択します。
3. [概要] ページで、[ポリシーの編集] を選択します。[JSON] タブを選択します。
4. 次のポリシー例で強調表示されているセクションをポリシーに追加します。サンプルのアカウント ID (*012345678901*) を自分のアカウント ID に置き換えます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ReadCode",
            "Effect": "Allow",
            "Action": [
                "s3:Abort*",
                "s3:DeleteObject*",
                "s3:GetObject*",
                "s3:GetBucket*",
                "s3:List*",
                "s3:ListBucket",
```

```
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::ka-app-code-<username>",
                "arn:aws:s3:::ka-app-code-<username>/*"
            ]
        },
        {
            "Sid": "DescribeLogGroups",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:*"
            ]
        },
        {
            "Sid": "DescribeLogStreams",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogStreams"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:*"
            ]
        },
        {
            "Sid": "PutLogEvents",
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:kinesis-analytics-log-stream"
            ]
        },
        {
            "Sid": "ReadInputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
```

```
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
        }
    ]
}
```

アプリケーションを実行する

Flink ジョブグラフは、アプリケーションを実行し、Apache Flink ダッシュボードを開き、目的の Flink ジョブを選択すると表示できます。

アプリケーションの停止

アプリケーションを停止するには、MyApplicationページで停止 を選択します。アクションを確認します。

アプリケーションの作成と実行 (CLI)

このセクションでは、 を使用して Managed Service for Apache Flink アプリケーション AWS Command Line Interface を作成して実行します。kinesisanalyticsv2 AWS CLI コマンドを使用して、Managed Service for Apache Flink アプリケーションを作成して操作します。

許可ポリシーを作成する

> **Note**
>
> アプリケーションのアクセス権限ポリシーとロールを作成する必要があります。これらの IAM リソースを作成しない場合、アプリケーションはそのデータストリームやログストリームにアクセスできません。

まず、2 つのステートメントを含むアクセス許可ポリシーを作成します。1 つは、ソースストリームの読み取りアクションに対するアクセス許可を付与し、もう 1 つはシンクストリームの書き込みアクションに対するアクセス許可を付与します。次に、IAM ロール (次のセクションで作成) にポリシーをアタッチします。そのため、 Managed Service for Apache Flinkがこのロールを引き受けると、ソースストリームからの読み取りとシンクストリームへの書き込みを行うために必要なアクセス許可がサービスに付与されます。

次のコードを使用して AKReadSourceStreamWriteSinkStream アクセス許可ポリシーを作成します。**username** を Amazon S3 バケットの作成に使用したユーザー名に置き換え、アプリケーショ

ンコードを保存します。Amazon リソースネーム (ARN) **(012345678901)**のアカウント ID を自分のアカウント ID に置き換えます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ReadCode",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": [
                "arn:aws:s3:::ka-app-code-username/getting-started-scala-1.0.jar"
            ]
        },
        {
            "Sid": "DescribeLogGroups",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:*"
            ]
        },
        {
            "Sid": "DescribeLogStreams",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogStreams"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-analytics/
MyApplication:log-stream:*"
            ]
        },
        {
            "Sid": "PutLogEvents",
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents"
```

```
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-analytics/
MyApplication:log-stream:kinesis-analytics-log-stream"
            ]
        },
        {
            "Sid": "ReadInputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
        },
        {
            "Sid": "WriteOutputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleOutputStream"
        }
    ]
}
```

アクセス許可ポリシーを作成する step-by-step 手順については、IAM ユーザーガイドの「チュートリアル: 最初のカスタマー管理ポリシーの作成とアタッチ」を参照してください。

IAM ロールを作成する

このセクションでは、Managed Service for Apache Flink アプリケーションがソースストリームを読み取り、シンクストリームに書き込むために想定できる IAM ロールを作成します。

Apache Flink 用 Managed Service は、許可なしにはストリームにアクセスできません。IAM ロールを介してこれらの許可を付与します。各 IAM ロールには、2 つのポリシーがアタッチされます。信頼ポリシーは、ロールを引き受けるための許可を Managed Service for Apache Flink 付与し、許可ポリシーは、ロールを引き受けた後に Managed Service for Apache Flink が実行できる事柄を決定します。

前のセクションで作成したアクセス許可ポリシーをこのロールにアタッチします。

IAM ロールを作成するには

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。

2. ナビゲーションペインで [ロール] を選択し、続いて [ロールを作成] を選択します。
3. [信頼されるエンティティの種類を選択] で、[AWS のサービス] を選択します。
4. [このロールを使用するサービスを選択] で、[Kinesis Analytics] を選択します。
5. [ユースケースの選択]で、[Managed Service for Apache Flink]を選択します。
6. [Next: Permissions] (次のステップ: 許可) を選択します。
7. [アクセス権限ポリシーをアタッチする] ページで、[Next: Review] (次: 確認) を選択します。ロールを作成した後に、アクセス許可ポリシーをアタッチします。
8. [Create role (ロールの作成)] ページで、ロールの名前に **`MF-stream-rw-role`** を入力します。[ロールの作成] を選択します。

   これで、`MF-stream-rw-role` と呼ばれる新しい IAM ロールが作成されます。次に、ロールの信頼ポリシーとアクセス許可ポリシーを更新します。
9. アクセス許可ポリシーをロールにアタッチします。

   > **Note**
   >
   > この演習では、Managed Service for Apache Flink が、Kinesis データストリーム (ソース) からのデータの読み取りと、別の Kinesis データストリームへの出力の書き込みの両方を実行するためにこのロールを引き受けます。前のステップである「許可ポリシーの作成」で作成したロールを添付します。

   a. [概要] ページで、[アクセス許可] タブを選択します。
   b. [Attach Policies (ポリシーのアタッチ)] を選択します。
   c. 検索ボックスに**`AKReadSourceStreamWriteSinkStream`**(前のセクションで作成したポリシー) と入力します。
   d. `AKReadSourceStreamWriteSinkStream`ポリシーを選択し、[ポリシーを添付]を選択します。

これで、アプリケーションがリソースにアクセスするために使用するサービスの実行ロールが作成されました。新しいロールの ARN を書き留めておきます。

ロールを作成する step-by-step 手順については、IAM ユーザーガイドの「IAM ロールの作成 (コンソール)」を参照してください。

アプリケーションの作成

次の JSON コードを create_request.json という名前のファイルに保存します。サンプルロールの ARN を、前に作成したロールの ARN に置き換えます。バケット ARN のサフィックス (ユーザー名) を、前のセクションで選択したサフィックスに置き換えます。サービス実行ロールのサンプルのアカウント ID (012345678901) を、自分のアカウント ID に置き換えます。

```
{
    "ApplicationName": "s3_sink",
    "ApplicationDescription": "Scala tumbling window application",
    "RuntimeEnvironment": "FLINK-1_15",
    "ServiceExecutionRole": "arn:aws:iam::012345678901:role/MF-stream-rw-role",
    "ApplicationConfiguration": {
        "ApplicationCodeConfiguration": {
            "CodeContent": {
                "S3ContentLocation": {
                    "BucketARN": "arn:aws:s3:::ka-app-code-username",
                    "FileKey": "s3-sink-scala-1.0.jar"
                }
            },
            "CodeContentType": "ZIPFILE"
        },
        "EnvironmentProperties":  {
         "PropertyGroups": [
            {
               "PropertyGroupId": "ConsumerConfigProperties",
               "PropertyMap" : {
                    "aws.region" : "us-west-2",
                    "stream.name" : "ExampleInputStream",
                    "flink.stream.initpos" : "LATEST"
               }
            },
            {
               "PropertyGroupId": "ProducerConfigProperties",
               "PropertyMap" : {
                    "s3.sink.path" : "s3a://ka-app-code-<username>/data"
               }
            }
         ]
      }
    },
    "CloudWatchLoggingOptions": [
      {
```

```
         "LogStreamARN": "arn:aws:logs:us-west-2:012345678901:log-
group:MyApplication:log-stream:kinesis-analytics-log-stream"
      }
   ]
}
```

次のリクエストCreateApplicationで を実行してアプリケーションを作成します。

```
aws kinesisanalyticsv2 create-application --cli-input-json file://create_request.json
```

これでアプリケーションが作成されました。次のステップでは、アプリケーションを起動します。

アプリケーションを起動する

このセクションでは、 StartApplicationアクションを使用してアプリケーションを起動します。

アプリケーションを起動するには

1. 次の JSON コードを start_request.json という名前のファイルに保存します。

```
{{
    "ApplicationName": "s3_sink",
    "RunConfiguration": {
        "ApplicationRestoreConfiguration": {
         "ApplicationRestoreType": "RESTORE_FROM_LATEST_SNAPSHOT"
         }
    }
}
```

2. 前述のリクエストを指定して StartApplication アクションを実行し、アプリケーションを起動します。

```
aws kinesisanalyticsv2 start-application --cli-input-json file://start_request.json
```

アプリケーションが実行されます。Amazon CloudWatch コンソールで Managed Service for Apache Flink メトリクスをチェックして、アプリケーションが動作していることを確認できます。

アプリケーションの停止

このセクションでは、 StopApplicationアクションを使用してアプリケーションを停止します。

アプリケーションを停止するには

1. 次の JSON コードを stop_request.json という名前のファイルに保存します。

```
{
   "ApplicationName": "s3_sink"
}
```

2. 前述のリクエストを指定して StopApplication アクションを実行し、アプリケーションを起動します。

```
aws kinesisanalyticsv2 stop-application --cli-input-json file://stop_request.json
```

アプリケーションが停止します。

CloudWatch ログ記録オプションを追加する

を使用して AWS CLI 、Amazon CloudWatch ログストリームをアプリケーションに追加できます。アプリケーションで CloudWatch ログを使用する方法については、「アプリケーションログ記録のセットアップ」を参照してください。

環境プロパティを更新する

このセクションでは、 UpdateApplicationアクションを使用して、アプリケーションコードを再コンパイルせずにアプリケーションの環境プロパティを変更します。この例では、ソースストリームおよびレプリケート先ストリームのリージョンを変更します。

アプリケーションの環境プロパティを更新します

1. 次の JSON コードを update_properties_request.json という名前のファイルに保存します。

```
{"ApplicationName": "s3_sink",
   "CurrentApplicationVersionId": 1,
   "ApplicationConfigurationUpdate": {
      "EnvironmentPropertyUpdates": {
         "PropertyGroups": [
            {
               "PropertyGroupId": "ConsumerConfigProperties",
               "PropertyMap" : {
                    "aws.region" : "us-west-2",
```

```
                    "stream.name" : "ExampleInputStream",
                    "flink.stream.initpos" : "LATEST"
                }
            },
            {
                "PropertyGroupId": "ProducerConfigProperties",
                "PropertyMap" : {
                    "s3.sink.path" : "s3a://ka-app-code-<username>/data"
                }
            }
        ]
      }
   }
}
```

2. 前述のリクエストでUpdateApplicationアクションを実行し、環境プロパティを更新します。

```
aws kinesisanalyticsv2 update-application --cli-input-json file://
update_properties_request.json
```

アプリケーションコードの更新

アプリケーションコードを新しいバージョンのコードパッケージで更新する必要がある場合は、 CLI UpdateApplication アクションを使用します。

> **Note**
>
> 同じファイル名のアプリケーションコードの新しいバージョンをロードするには、新しいオブジェクトバージョンを指定する必要があります。Amazon S3 オブジェクトバージョンを使用する方法の詳細については、「バージョニングの有効化または無効化」を参照してください。

を使用するには AWS CLI、Amazon S3 バケットから以前のコードパッケージを削除し、新しいバージョンをアップロードして、同じ Amazon S3 バケットとオブジェクト名、および新しいオブジェクトバージョンUpdateApplicationを指定して を呼び出します。アプリケーションは新しいコードパッケージで再起動します。

以下の UpdateApplication アクションのサンプル・リクエストは、アプリケーション・コードを再読み込 み、アプリケーションを再起動します。CurrentApplicationVersionId を現在のアプリケーションバージョンに更新します。ListApplications または DescribeApplication アクションを使用して、現在のアプリケーションバージョンを確認できます。バケット名のサフィックス (「<username>」) を、 依存リソースを作成する セクションで選択したサフィックスで更新します。

```
{
    "ApplicationName": "s3_sink",
    "CurrentApplicationVersionId": 1,
    "ApplicationConfigurationUpdate": {
        "ApplicationCodeConfigurationUpdate": {
            "CodeContentUpdate": {
                "S3ContentLocationUpdate": {
                    "BucketARNUpdate": "arn:aws:s3:::ka-app-code-username",
                    "FileKeyUpdate": "s3-sink-scala-1.0.jar",
                    "ObjectVersionUpdate": "SAMPLEUehYngP87ex1nzYIGYgfhypvDU"
                }
            }
        }
    }
}
```

AWS リソースをクリーンアップする

このセクションでは、「タンブリングウィンドウ」チュートリアルで作成した AWS リソースをクリーンアップする手順について説明します。

このトピックには、次のセクションが含まれています。

- Managed Service for Apache Flink アプリケーションを削除する
- Kinesis データストリームを削除する
- Amazon S3 オブジェクトとバケットを削除する
- IAM リソースを削除する
- CloudWatch リソースを削除する

Managed Service for Apache Flink アプリケーションを削除する

1. https://console.aws.amazon.com/flink で Apache Flink 用 Managed Serviceコンソールを開く
2. Managed Service for Apache Flink パネルで、 を選択しますMyApplication。

3. アプリケーションのページで[削除]を選択し、削除を確認します。

Kinesis データストリームを削除する

1. 「https://console.aws.amazon.com/kinesis」で Kinesis コンソールを開きます。
2. Kinesis Data Streams パネルで、 を選択しますExampleInputStream。
3. ExampleInputStream ページで、Kinesis ストリームの削除を選択し、削除を確定します。
4. Kinesis ストリームページで、 を選択しExampleOutputStream、アクション を選択し、削除 を選択し、削除を確定します。

Amazon S3 オブジェクトとバケットを削除する

1. https://console.aws.amazon.com/s3/でAmazon S3 コンソールを開きます。
2. ka-app-code-***<username>*** バケットを選択します。
3. [削除] を選択し、バケット名を入力して削除を確認します。

IAM リソースを削除する

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. ナビゲーションバーで、[ポリシー] を選択します。
3. フィルターコントロールに「kinesis」と入力します。
4. kinesis-analytics-service-MyApplication-us-west-2 ポリシーを選択します。
5. [ポリシーアクション]、[削除] の順に選択します。
6. ナビゲーションバーで [ロール] を選択します。
7. kinesis-analytics-MyApplication-us-west-2 ロールを選択します。
8. [ロールの削除] を選択し、削除を確定します。

CloudWatch リソースを削除する

1. https://console.aws.amazon.com/cloudwatch/ で CloudWatch コンソールを開きます。
2. ナビゲーションバーで [ログ] を選択します。
3. /aws/kinesis-analytics/MyApplication ロググループを選択します。
4. [ロググループの削除]を選択し、削除を確認してください。

Apache Flink 用 Managed Service で Studio ノートブックを使用する

Managed Service for Apache Flink用 Studio ノートブックを使用すると、リアルタイムでインタラクティブにデータストリームをクエリし、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。 AWS マネジメントコンソールで数回クリックするだけで、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。ノートブックを使用すると、Apache Flink が提供する高度な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体験が得られます。Studio ノートブックは、「Apache Zeppelin」で動作する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、あらゆるスキルセットの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。

- データの可視化
- ファイルにデータをエクスポートする
- 分析しやすい出力形式のコントロール

Apache Flink と Apache Zeppelin 用 Managed Service の使用を開始するには、 Studio ノートブックチュートリアルの作成 を参照してください。Zeppelin の詳細については、「Apache Zeppelin ドキュメント」を参照してください。

ノートブックでは、SQL、Python、Scala の Apache Flink Table API と SQL、または Scala の DataStream API を使用してクエリをモデル化します。数回クリックするだけで、Studio ノートブックを、継続的に実行される非インタラクティブな Apache Flink ストリーム処理アプリケーション用 Managed Service に昇格させ、本番ワークロードに使用できます。

このトピックには、次のセクションが含まれています。

- Studio ノートブックのランタイムバージョン
- Studio ノートブックの作成
- ストリーミングデータのインタラクティブな分析
- 永続的な状態のアプリケーションとしてデプロイする

- Studio ノートブックの IAM パーミッション
- コネクタと依存関係の使用
- ユーザー定義関数の実装
- チェックポイントの有効化
- Studio ランタイムのアップグレード
- の使用 AWS Glue
- サンプルおよびチュートリアル
- トラブルシューティング
- 付録:カスタム IAM ポリシーの作成

Studio ノートブックのランタイムバージョン

Amazon Managed Service for Apache Flink Studio を使用すると、データストリームをリアルタイムでクエリし、インタラクティブなノートブックで標準の SQL、Python、Scala を使用してストリーム処理アプリケーションを構築して実行できます。Studio ノートブックは Apache Zeppelin を搭載しており、ストリーム処理エンジンとして Apache Flink を使用します。

> **Note**
>
> 2024 年 11 月 5 日に、Apache Flink バージョン 1.11 で Studio Runtime を非推奨にします。この日以降、このバージョンを使用して新しいノートブックを実行したり、新しいアプリケーションを作成したりすることはできません。その前に最新のランタイム (Apache Flink 1.15 および Apache Zeppelin 0.10) にアップグレードすることをお勧めします。ノートブックをアップグレードする方法については、「」を参照してくださいStudio ランタイムのアップグレード。

Studio ランタイム

Apache Flink バージョン	Apache Zeppelin バージョン	Python バージョン	
1.15	0.10	3.8	推奨
1.13	0.9	3.8	サポート

Apache Flink バージョン	Apache Zeppelin バージョン	Python バージョン	
1.11	0.9	37	2024 年 11 月 5 日に廃止

Studio ノートブックの作成

Studio ノートブックには、ストリーミングデータ上で実行され、分析結果を返す SQL、Python、または Scala で記述されたクエリまたはプログラムが含まれています。コンソールまたは CLI のいずれかを使用してアプリケーションを作成し、データソースからのデータを分析するためのクエリを提供します。

アプリケーションには次のコンポーネントがあります。

- Amazon MSK クラスター、Kinesis データストリーム、または Amazon S3 バケットなどのデータソース。
- AWS Glue データベース。このデータベースには、データソースとデスティネーションスキーマとエンドポイントを格納するテーブルが含まれています。詳細については、「の使用 AWS Glue」を参照してください。
- アプリケーションコード。顧客のコードは分析クエリまたはプログラムを実装します。
- アプリケーション設定とランタイムプロパティ。アプリケーション設定とランタイムプロパティについては、「 Apache Flink Applications 用デベロッパーガイド」の以下のトピックを参照してください。
  - 「アプリケーションの並列度とスケーリング： 」アプリケーションの並列度設定を使用して、アプリケーションが同時に実行できるクエリの数を制御します。また、以下のような場合は、複数の実行経路を持つクエリは、並列度の向上を利用することができます。
    - Kinesis データストリームの複数のシャードを処理する場合
    - KeyBy オペレータを使用してデータを分割する場合。
    - 複数のウィンドウオペレータを使用する場合

    アプリケーションスケーリングの詳細情報については、「Apache Flink 用 Managed Service のアプリケーションスケーリング」を参照してください。

- ロギングとモニタリング: アプリケーションのロギングとモニタリングについては、「Apache Flink 向けの Amazon Managed Service for Apache Flink でのロギングとモニタリング」を参照してください。
- アプリケーションでは、チェックポイントとセーブポイントを使用してフォールトトレランスを行います。Studio ノートブックでは、チェックポイントとセーブポイントはデフォルトでは有効になっていません。

Studio ノートブックは、 AWS Management Console または を使用して作成できます AWS CLI。

コンソールからアプリケーションを作成する場合、以下のオプションがあります：。

- Amazon MSK コンソールでクラスターを選択し、[データをリアルタイムで処理]を選択します。
- Kinesis Data Streams コンソールでデータストリームを選択し、次に [アプリケーション] タブで [データをリアルタイムで処理] を選択します。
- Apache Flink 用 Managed Service コンソールで [Studio] タブを選択し、次に [Studio ノートブックの作成] を選択します。

チュートリアルについては、「Managed Service for Apache Flink によるイベント検出」を参照してください。

アドバンスト Studio ノートブックソリューションの例については、「Amazon Managed Service for Apache Flink Studio の Apache Flink」を参照してください。

ストリーミングデータのインタラクティブな分析

Apache Zeppelin を搭載したサーバーレス· ノートブックを使用して、ストリーミングデータとやり取りします。ノートブックには複数のノートを書くことができ、各ノートにはコードを書く段落を 1 つ以上書くことができます。

次の SQL クエリの例は、データソースからデータを取得する方法を示しています。

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

Flink Streaming SQL クエリのその他の例については、サンプルおよびチュートリアル以下および Apache Flink ドキュメントの「クエリ」を参照してください。

Studio ノートブックの Flink SQL クエリを使用してストリーミングデータをクエリできます。また、Python (Table API) や Scala (Table API と Datastream API) を使って、ストリーミングデータをインタラクティブにクエリするプログラムを書くこともできます。クエリやプログラムの結果を表示し、数秒で更新して再実行し、更新された結果を表示することができます。

Flink インタプリタ

Apache Flink 用 Managed Service がアプリケーションの実行に使用する言語は、「インタプリタ」を使用して指定します。Apache Flink 用 Managed Serviceでは、以下のインタプリタを使用できます。

名前	Class	説明
%flink	FlinkInterpreter	ExecutionEnvironment/StreamExecutionEnvironment/BatchTableEnvironment/ StreamTableEnvironment を作成し、Scala 環境を提供します
%flink.pyflink	PyFlinkInterpreter	Python 環境を提供する
%flink.ipyflink	IPyFlinkInterpreter	ipython 環境を提供します。
%flink.ssql	FlinkStreamSqlInterpreter	ストリーム SQL 環境を提供する
%flink.bsql	FlinkBatchSqlInterpreter	バッチ SQL 環境を提供する

Flink インタープリタの詳細情報については、「Apache Zeppelin 用 Flink インタープリタ」を参照してください。

インタプリタとして `%flink.pyflink` または `%flink.ipyflink` を使用している場合は、`ZeppelinContext`を使用してノートブック内で結果を視覚化する必要があります。

より PyFlink 具体的な例については、「Managed Service for Apache Flink Studio と Python を使用してデータストリームをインタラクティブにクエリする」を参照してください。

Apache Flink テーブルの環境変数

Apache Zeppelin では、環境変数を使用してテーブル環境リソースにアクセスできます。

以下の変数を使用して Scala テーブル環境リソースにアクセスします。

変数	リソース
`senv`	`StreamExecutionEnvironment`
`stenv`	`StreamTableEnvironment for blink planner`

以下の変数を使用して Python テーブル環境リソースにアクセスします。

変数	リソース
`s_env`	`StreamExecutionEnvironment`
`st_env`	`StreamTableEnvironment for blink planner`

テーブル環境の使用の詳細については、Apache Flink ドキュメントの「概念と共通 API」を参照してください。

永続的な状態のアプリケーションとしてデプロイする

コードを構築して Amazon S3 にエクスポートできます。ノートに書いたコードを、継続的に実行されるストリーム処理アプリケーションに昇格させることができます。Apache Flink 用 Managed Service で Apache Flink アプリケーションを実行するには、2 つのモードがあります。Studio ノートブックでは、コードをインタラクティブに開発し、コードの結果をリアルタイムで表示し、ノート内で可視化することができます。ノートをストリーミング・モードで実行するようにデプロイすると、Apache Flink 用 Managed Service は、継続的に実行され、ソースからデータを読み取り、デスティネーションに書き込み、長時間実行されるアプリケーションの状態を維持し、ソースストリームのスループットに基づいて自動的にオートスケールするアプリケーションを作成します。

> **Note**
>
> アプリケーションコードをエクスポートする S3 バケットは、スタジオノートブックと同じリージョンにある必要があります。

Studio ノートパソコンからノートパソコンを導入するには、次の条件を満たす必要があります。

- 段落は順番に並べる必要があります。アプリケーションをデプロイすると、メモ内のすべての段落が、メモに表示されるように順番に (left-to-right、 top-to-bottom) 実行されます。この順序は、ノートの [すべての段落を実行] を選択することで確認できます。
- 顧客のコードは Python と SQL、または Scala と SQL の組み合わせです。現時点では、Python と Scala はサポートされていません deploy-as-application。
- ノートには、「`%flink` `%flink.ssql` `%flink.pyflink` `%flink.ipyflink` `%md`」のインタープリタのみを使用してください。
- 「Zeppelin コンテキスト」オブジェクト `z` の使用はサポートされていません。何も返さないメソッドは、警告をログに記録する以外に何もしません。その他のメソッドは Python の例外を発生させたり、Scala でのコンパイルに失敗したりします。
- 1 つのノートの結果が 1 つの Apache Flink ジョブになる必要があります。
- 「動的フォーム」を持つ Notes は、アプリケーションとしてデプロイすることはできません。
- %md (Markdown) 段落は、アプリケーションとしてデプロイする際にスキップされます。なぜなら、これらの段落には、結果のアプリケーションの一部として実行するのに適さない、人間が読める文書が含まれていると考えられるからです。
- Zeppelin 内で実行するために無効化された段落は、アプリケーションとしてデプロイする際にスキップされます。無効化された段落が互換性のないインタプリタを使用していても、例えば `%flink and %flink.ssql` インタプリタを含むノートの `%flink.ipyflink` では、アプリケーションとしてノートをデプロイする際にスキップされ、エラーにはなりません。
- アプリケーションのデプロイを成功させるには、 の実行が有効になっているソースコード (Flink SQL、 PyFlink または Flink Scala) を含む段落が少なくとも 1 つ存在している必要があります。
- 段落内のインタプリタディレクティブで並列度を設定しても (例えば `%flink.ssql(parallelism=32)`)、ノートからデプロイされたアプリケーションでは無視されます。代わりに、、 AWS Command Line Interface 、または AWS API を使用してデプロイされたアプリケーションを更新して AWS Management Console、アプリケーションに必要な並列度のレベルに応じて並列処理や ParallelismPerKPU 設定を変更したり、デプロイされたアプリケーションの自動スケーリングを有効にしたりできます。

- 永続的な状態のアプリケーションとしてデプロイする場合は、VPC がインターネットにアクセスできる必要があります。VPC がインターネットにアクセスできない場合は、 インターネットにアクセスできない VPC に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。

Scala/Python の基準

- Scala または Python のコードでは、古い「Flink」プランナー (Scala 用 `stenv_2` 、Python 用 `st_env_2`) ではなく、「Blinkプランナー」(Scala 用 `senv` 、 `stenv` 、Python 用 `s_env` 、`st_env`) を使用してください。Apache Flink プロジェクトでは、Zeppelin および Flink のデフォルトのプランナーである Blink プランナーを本番ユースケースで使用することを推奨しています。
- 顧客の Python の段落は、アプリケーションとしてデプロイされることを意図したノートの `%timeit` または `%conda` で、 ！または「IPython マジックコマンド」を使用した「シェルの呼び出し/割り当て」を使ってはいけません。
- Scala ケースクラスは、 `map` や `filter` などの高階データフローオペレータに渡される関数のパラメーターとして使用することはできません。Scala ケースクラスについては、Scala ドキュメントの「CASE CLASSES」を参照してください。

SQL 基準

- 段落の出力セクションに相当する、データを渡す場所がないため、単純な SELECT ステートメントは使用できません。
- どの段落でも、DDL ステートメント (USE、CREATE ALTER、DROP、SET、RESET) は DML (INSERT) ステートメントの前に置く必要があります。これは、1 つの段落内の DML ステートメントを 1 つの Flink ジョブとしてまとめて送信する必要があるためです。
- DML ステートメントを含む段落は 1 つまでにしてください。これは、この機能 deploy-as-application では、Flink への 1 つのジョブの送信のみがサポートされているためです。

詳細と例については、「Amazon Managed Service for Apache Flink、Amazon Translate、Amazon Comprehend で SQL 関数を使用してストリーミングデータを翻訳、編集、分析する」を参照してください。

Studio ノートブックの IAM パーミッション

Apache Flink 用 Managed Service では、 AWS Management Consoleで Studio ノートブックを作成すると、IAM ロールが作成されます。また、以下のアクセスを許可するポリシーをそのロールに関連付けます。

サービス	アクセス
CloudWatch ログ	リスト
Amazon EC2	リスト
AWS Glue	読み取り、書き込み
Managed Service for Apache Flink	読み取り
Managed Service for Apache Flink V2	読み取り
Amazon S3	読み取り、書き込み

コネクタと依存関係の使用

コネクタを使用すると、さまざまなテクノロジーにわたってデータの読み書きが可能になります。Apache Flink 用 Managed Service では、Studio ノートブックに 3 つのデフォルトコネクタがバンドルされています。カスタムコネクタを使用することもできます。コネクタの詳細については、Apache Flink ドキュメントの「テーブルコネクタと SQL コネクタ」を参照してください。

デフォルトコネクター

を使用して Studio ノートブック AWS Management Console を作成する場合、Apache Flink 用 Managed Service には、デフォルトで `flink-sql-connector-kinesis`、 `flink-connector-kafka_2.12`のカスタムコネクタが含まれています`aws-msk-iam-auth`。これらのカスタムコネクタを使用せずにコンソールから Studio ノートブックを作成するには、[カスタム設定で作成] オプションを選択します。次に、[Configurations] ページに移動し、2つのコネクターの横にあるチェックボックスをオフにします。

CreateApplication API を使用して Studio ノートブックを作成する場合、 `flink-sql-connector-flink`および `flink-connector-kafka` コネクタはデフォルトでは含まれていません。これ

らを追加するには、以下の例のように CustomArtifactsConfiguration データタイプに MavenReference を指定します。

この aws-msk-iam-auth コネクタは、IAM で自動的に認証する機能を含む、Amazon MSK で使用するコネクタです。

> **Note**
>
> 以下の例に示されているコネクタバージョンは、当社がサポートしている唯一のバージョンです。

```
For the Kinesis connector:

"CustomArtifactsConfiguration": [{
"ArtifactType": "DEPENDENCY_JAR",
   "MavenReference": {
"GroupId": "org.apache.flink",

      "ArtifactId": "flink-sql-connector-kinesis",
      "Version": "1.15.4"

   }
}]

For authenticating with AWS MSK through AWS IAM:

"CustomArtifactsConfiguration": [{
"ArtifactType": "DEPENDENCY_JAR",
   "MavenReference": {
"GroupId": "software.amazon.msk",
      "ArtifactId": "aws-msk-iam-auth",
      "Version": "1.1.6"
   }
}]

For the Apache Kafka connector:

"CustomArtifactsConfiguration": [{
"ArtifactType": "DEPENDENCY_JAR",
   "MavenReference": {
"GroupId": "org.apache.flink",
```

```
      "ArtifactId": "flink-connector-kafka",
      "Version": "1.15.4"

   }
}]
```

これらのコネクタを既存のノートブックに追加するには、 UpdateApplication API オペレーションを使用して、CustomArtifactsConfigurationUpdateデータ型MavenReferenceに として指定します。

> **Note**
>
> テーブル API の flink-sql-connector-kinesis コネクタでは、failOnErrorを true に設定できます。

依存関係とカスタムコネクタの追加

を使用して Studio ノートブックに AWS Management Console 依存関係またはカスタムコネクタを追加するには、次の手順に従います。

1. カスタムコネクタのファイルを Amazon S3 にアップロードします。
2. で AWS Management Console、Studio ノートブックを作成するためのカスタム作成オプションを選択します。
3. Studio ノートブック作成ワークフローに従って 設定 ステップまで進みます。
4. [カスタムコネクタ] セクションで [カスタムコネクタを追加] を選択します。
5. 依存関係またはカスタムコネクタの Amazon S3 のロケーションを指定します。
6. [変更の保存] を選択します。

CreateApplication API を使用して新しい Studio ノートブックを作成するときに依存関係 JAR またはカスタムコネクタを追加するには、依存関係 JAR の Amazon S3 の場所またはカスタムコネクタを CustomArtifactsConfiguration データ型に指定します。既存の Studio ノートブックに依存関係またはカスタムコネクタを追加するには、 UpdateApplication API オペレーションを呼び出し、依存関係 JAR の Amazon S3 の場所またはカスタムコネクタを CustomArtifactsConfigurationUpdate データ型に指定します。

> **Note**
>
> 依存関係またはカスタムコネクタを含める場合は、その中にバンドルされていない推移的な依存関係もすべて含める必要があります。

ユーザー定義関数の実装

ユーザー定義関数 (UDF) は、頻繁に使用されるロジックや、他の方法ではクエリで表現できないカスタムロジックを呼び出すことができる拡張ポイントです。Python または Java や Scala などの JVM 言語を使用して、Studio ノートブック内の段落に UDF を実装できます。JVM 言語で実装された UDF を含む外部 JAR ファイルを Studio ノートブックに追加することもできます。

`UserDefinedFunction` （または独自の抽象クラス ）をサブクラス化する抽象クラスを登録する JAR を実装する場合は、Apache Maven で提供されている範囲、Gradle の `compileOnly` 依存宣言、SBT で提供されている範囲、または UDF プロジェクトのビルド設定で同等の命令を使用します。これにより、UDF ソースコードは Flink API に対してコンパイルできますが、Flink API クラス自体はビルドアーティファクトに含まれません。Maven プロジェクトでこのような前提条件を守っている UDF jar の例の「pom」を参照してください。

> **Note**
>
> セットアップの例については、「AWS Machine Learning Blog」の「Amazon Managed Service for Apache Flink、Amazon Translate および Amazon Comprehend を搭載した SQL 関数を使用してストリーミングデータを翻訳、修正、分析する」を参照してください。

コンソールを使用して UDF JAR ファイルを Studio ノートブックに追加するには、以下の手順に従います。

1. UDF JAR ファイルを Amazon S3 にアップロードします。
2. で AWS Management Console、Studio ノートブックを作成するためのカスタム作成オプションを選択します。
3. Studio ノートブック作成ワークフローに従って 設定 ステップまで進みます。
4. [ユーザー定義関数] セクションで、[ユーザー定義関数を追加] を選択します。
5. UDF が実装されている JAR ファイルまたは ZIP ファイルの Amazon S3 ロケーションを指定します。

6. [変更の保存] を選択します。

CreateApplication API を使用して新しい Studio ノートブックを作成するときに UDF JAR を追加するには、CustomArtifactConfigurationデータ型で JAR の場所を指定します。既存の Studio ノートブックに UDF JAR を追加するには、 UpdateApplication API オペレーションを呼び出し、CustomArtifactsConfigurationUpdateデータ型で JAR の場所を指定します。または、を使用して UDF JAR ファイルを Studio ノートブック AWS Management Console に追加することもできます。

ユーザー定義関数に関する考慮事項

- Apache Flink Studio 用 Managed Service では、「Apache Zeppelin の用語」が使われています。ここで、ノートブックは、複数のノートを含むことができる Zeppelin インスタンスです。これにより、各ノートには複数の段落を含めることができます。Apache Flink Studio 用 Managed Service では、インタープリタープロセスはノートブックのすべてのノートで共有されます。したがって、あるメモで createTemporarySystemFunction を使用して明示的な関数登録を実行すると、同じノートブックの別のメモで同じ関数をそのまま参照できます。

  ただし、「アプリケーションとしてデプロイ」操作は、ノートブック内のすべてのノートではなく、「個々」のノートに適用されます。アプリケーションとしてデプロイすると、アクティブノートの内容のみがアプリケーションの生成に使用されます。他のノートブックで行われる明示的な関数登録は、生成されるアプリケーションの依存関係の一部ではありません。さらに、アプリケーション・オプションとしてデプロイする際に、JAR のメインクラス名を小文字の文字列に変換することで、暗黙的な関数登録が行われます。

  たとえば、 TextAnalyticsUDF が UDF JAR のメインクラスである場合、暗黙的に登録すると関数名 textanalyticsudf になります。そのため、Studio のノート 1 に次のような明示的な関数登録があった場合、共有インタープリタがあるため、そのノート (例えばノート 2) の他のすべてのノートは、名前 myNewFuncNameForClass でその関数を参照することができ

  ```
  stenv.createTemporarySystemFunction("myNewFuncNameForClass", new
  TextAnalyticsUDF())
  ```

  しかし、ノート 2 のアプリケーションとしてデプロイする操作では、この明示的な登録は依存関係に「含まれない」ため、デプロイされたアプリケーションは期待通りに動作しません。暗黙的に登録されるため、デフォルトでは、この関数へのすべての参照は myNewFuncNameForClass ではなく textanalyticsudf であることが想定されます。

カスタム関数名を登録する必要がある場合は、ノート 2 自体に別の段落を設け、次のように明示的に登録することが想定されます。

```
%flink(parallelism=l)
import com.amazonaws.kinesis.udf.textanalytics.TextAnalyticsUDF
# re-register the JAR for UDF with custom name
stenv.createTemporarySystemFunction("myNewFuncNameForClass", new TextAnalyticsUDF())
```

```
%flink. ssql(type=update, parallelism=1)
INSERT INTO
    table2
SELECT
    myNewFuncNameForClass(column_name)
FROM
    table1
;
```

- UDF JAR に Flink SDK が含まれている場合は、UDF ソースコードが Flink SDK に対してコンパイルできるように Java プロジェクトを設定しますが、Flink SDK クラス自体はビルドアーティファクト (JAR など) に含まれません。

  Apache Maven の `provided` スコープ、Gradle の `compileOnly` 依存宣言、SBT の `provided` スコープ、または UDF プロジェクトのビルド設定で同等の命令を使用できます。この「pom」は、Maven プロジェクトでこのような前提条件を満たす UDF jar の例から参照できます。完全な step-by-step チュートリアルについては、「Amazon Managed Service for Apache Flink、Amazon Translate、および Amazon Comprehend で SQL 関数を使用してストリーミングデータを翻訳、編集、分析する」を参照してください。

チェックポイントの有効化

環境設定を使ってチェックポイント機能を有効にします。チェックポイント機能の詳細については、「Apache Flink 用 Managed Service デベロッパーガイド」の「フォールトトレランス」を参照してください。

チェックポイント間隔の設定

次の Scala コード例では、アプリケーションのチェックポイント間隔を 1 分に設定しています。

```
// start a checkpoint every 1 minute
stenv.enableCheckpointing(60000)
```

次の Python コード例では、アプリケーションのチェックポイント間隔を 1 分に設定しています。

```
st_env.get_config().get_configuration().set_string(
    "execution.checkpointing.interval", "1min"
)
```

チェックポインティングタイプの設定

次の Scala コードの例では、アプリケーションのチェックポイントモードを EXACTLY_ONCE (デフォルト) に設定しています。

```
// set mode to exactly-once (this is the default)
stenv.getCheckpointConfig.setCheckpointingMode(CheckpointingMode.EXACTLY_ONCE)
```

次の Python コードの例では、アプリケーションのチェックポイントモードを EXACTLY_ONCE (デフォルト) に設定しています。

```
st_env.get_config().get_configuration().set_string(
    "execution.checkpointing.mode", "EXACTLY_ONCE"
)
```

Studio ランタイムのアップグレード

このセクションでは、Studio ノートブックランタイムをアップグレードする方法について説明します。常にサポートされている最新の Studio ランタイムにアップグレードすることをお勧めします。

ノートブックを新しい Studio ランタイムにアップグレードする

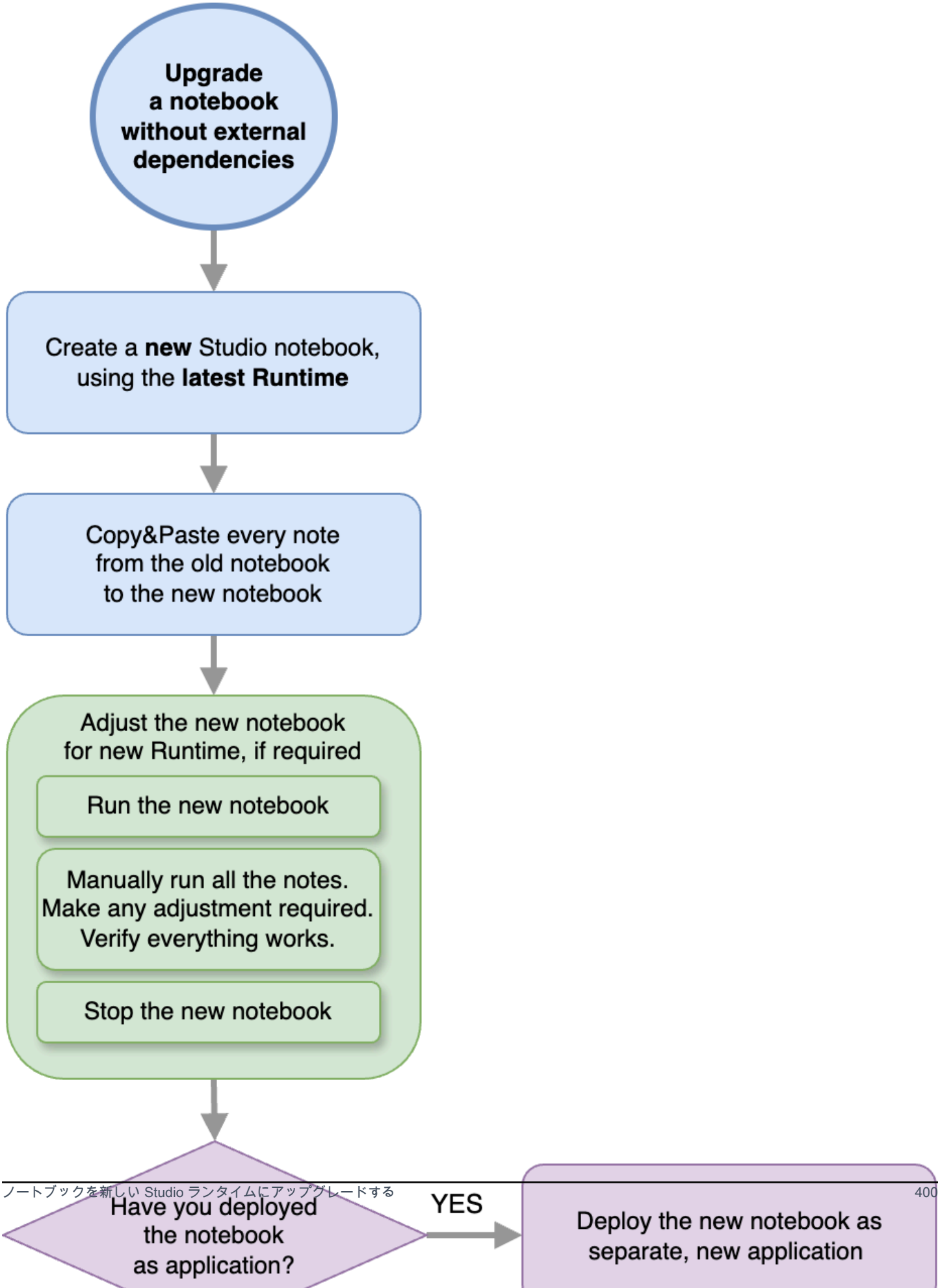
Studio の使用方法によって、ランタイムをアップグレードするステップは異なります。ユースケースに合ったオプションを選択します。

外部依存関係のない SQL クエリまたは Python コード

外部依存関係なしで SQL または Python を使用している場合は、次のランタイムアップグレードプロセスを使用します。最新のランタイムバージョンにアップグレードすることをお勧めします。アップグレードプロセスは、アップグレード元のランタイムバージョンと同じです。

1. 最新のランタイムを使用して新しい Studio ノートブックを作成します。
2. 古いノートブックから新しいノートブックにすべてのメモのコードをコピーして貼り付けます。
3. 新しいノートブックで、以前のバージョンから変更された Apache Flink 機能と互換性があるようにコードを調整します。
   - 新しいノートブックを実行します。ノートブックを開き、メモごとに順番に実行し、機能するかどうかをテストします。
   - コードに必要な変更を加えます。
   - 新しいノートブックを停止します。
4. 古いノートブックをアプリケーションとしてデプロイした場合：
   - 新しいノートブックを別の新しいアプリケーションとしてデプロイします。
   - 古いアプリケーションを停止します。
   - スナップショットなしで新しいアプリケーションを実行します。
5. 古いノートブックが実行されている場合は停止します。必要に応じて、インタラクティブな使用のために新しいノートブックを起動します。

外部依存関係なしでアップグレードするプロセスフロー

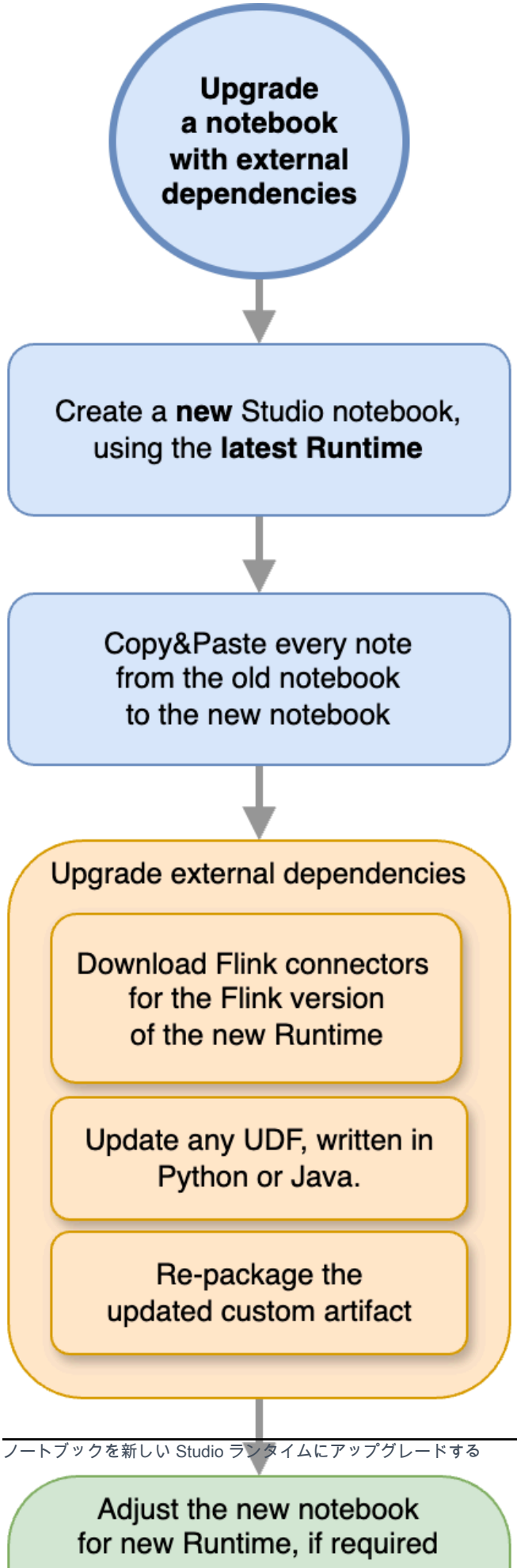

Upgrade a notebook without external dependencies
Create a new Studio notebook, using the latest Runtime
Copy&Paste every note from the old notebook to the new notebook
Adjust the new notebook for new Runtime, if required
Run the new notebook
Manually run all the notes. Make any adjustment required. Verify everything works.
Stop the new notebook
Have you deployed the notebook as application?
YES
Deploy the new notebook as separate, new application

外部依存関係を持つ SQL クエリまたは Python コード

SQL または Python を使用していて、Python または Java に実装されたユーザー定義関数などのコネクタやカスタムアーティファクトなどの外部依存関係を使用している場合は、このプロセスに従います。最新のランタイムにアップグレードすることをお勧めします。アップグレード元のランタイムバージョンに関係なく、プロセスは同じです。

1. 最新のランタイムを使用して新しい Studio ノートブックを作成します。
2. 古いノートブックから新しいノートブックにすべてのメモのコードをコピーして貼り付けます。
3. 外部依存関係とカスタムアーティファクトを更新します。
   - 新しいランタイムの Apache Flink バージョンと互換性のある新しいコネクタを探します。Flink バージョンの正しいコネクタを見つけるには、Apache Flink ドキュメントの「テーブルと SQL コネクタ」を参照してください。
   - Apache Flink API の変更と、ユーザー定義関数で使用される Python または JAR の依存関係と一致するように、ユーザー定義関数のコードを更新します。更新されたカスタムアーティファクトを再パッケージ化します。
   - これらの新しいコネクタとアーティファクトを新しいノートブックに追加します。
4. 新しいノートブックで、以前のバージョンから変更された Apache Flink 機能と互換性があるようにコードを調整します。
   - 新しいノートブックを実行します。ノートブックを開き、メモごとに順番に実行し、機能するかどうかをテストします。
   - コードに必要な変更を加えます。
   - 新しいノートブックを停止します。
5. 古いノートブックをアプリケーションとしてデプロイした場合：
   - 新しいノートブックを別の新しいアプリケーションとしてデプロイします。
   - 古いアプリケーションを停止します。
   - スナップショットなしで新しいアプリケーションを実行します。
6. 古いノートブックが実行されている場合は停止します。必要に応じて、インタラクティブな使用のために新しいノートブックを起動します。

外部依存関係を使用してアップグレードするプロセスフロー


Upgrade a notebook with external dependencies
Create a new Studio notebook, using the latest Runtime
Copy&Paste every note from the old notebook to the new notebook
Upgrade external dependencies
Download Flink connectors for the Flink version of the new Runtime
Update any UDF, written in Python or Java.
Re-package the updated custom artifact
Adjust the new notebook for new Runtime, if required

の使用 AWS Glue

Studio ノートブックは、データソースとシンクに関する情報を から保存して取得します AWS Glue。Studio ノートブックを作成するときは、接続情報を含む AWS Glue データベースを指定します。データソースとシンクにアクセスするときは、データベースに含まれる AWS Glue テーブルを指定します。 AWS Glue テーブルは、データソースと送信先の場所、スキーマ、パラメータを定義する AWS Glue 接続へのアクセスを提供します。

Studio ノートブックはテーブルプロパティを使用してアプリケーション固有のデータを保存します。詳細については、「テーブルプロパティ」を参照してください。

Studio ノートブックで使用する AWS Glue 接続、データベース、およびテーブルを設定する方法の例については、データベースを作成します。 AWS GlueStudio ノートブックチュートリアルの作成チュートリアルの「」を参照してください。

テーブルプロパティ

データフィールドに加えて、 AWS Glue テーブルはテーブルプロパティを使用して Studio ノートブックに他の情報を提供します。Managed Service for Apache Flink では、次の AWS Glue テーブルプロパティを使用します。

- Apache Flink 時間値の使用: これらのプロパティは、Apache Flink 用 Managed Service が Apache Flink の内部データ処理時間値をどのように出力するかを定義します。
- Flink コネクタとフォーマットプロパティの使用: これらのプロパティはデータストリームに関する情報を提供します。

AWS Glue テーブルにプロパティを追加するには、次の手順を実行します。

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/glue/ で AWS Glue コンソールを開きます。
2. テーブルのリストから、アプリケーションがデータ接続情報を保存するために使用するテーブルを選択します。[Action]、[テーブル詳細の編集] を選択します。
3. [テーブルプロパティ] で、[キー] に **`managed-flink.proctime`**、[値] に **`user_action_time`** を入力します。

Apache Flink 時間値の使用

Apache Flink は、「Processing Time」や「Event Time」など、ストリーム処理イベントの発生時刻を記述する時間値を提供します。これらの値をアプリケーション出力に含めるには、Managed Service for Apache Flink ランタイムにこれらの値を指定されたフィールドに出力するように指示するプロパティを AWS Glue テーブルに定義します。

テーブルプロパティで使用するキーと値は次のとおりです。

Timestamp タイプ	キー	値
処理時間	マネージド flink.proctime	AWS Glue が値を公開するために使用する列名。この列名は既存のテーブル列に対応していません。
イベント時刻	managed-flink.rowtime	AWS Glue が値を公開するために使用する列名。この列名は既存のテーブル列に対応します。
	managed-flink.watermark.「*column_name* 」.ミリ秒	ミリ秒単位の透かし間隔

Flink コネクタとフォーマットプロパティの使用

AWS Glue テーブルプロパティを使用して、データソースに関する情報をアプリケーションの Flink コネクタに提供します。Apache Flink 用 Managed Service がコネクタに使用するプロパティの例は、次のとおりです。

Connector Type	キー	値
Kafka	`format`	Kafka メッセージの逆シリアル化とシリアル化に使用される形式。例: `json`または `csv`。

Connector Type	キー	値
	`scan.startup.mode`	Kafka コンシューマーのスタートアップモード。例: `earliest-offset` または `timestamp` 。
Kinesis	`format`	Kinesis データストリームレコードの逆シリアル化とシリアル化に使用される形式。例: `json`または `csv`。
	`aws.region`	ストリームが定義されている AWS リージョン。
S3 (ファイルシステム)	format	ファイルの逆シリアル化とシリアル化に使用される形式。例: `json`または `csv`。
	`path`	Amazon S3 パス。例: `s3://mybucket/` 。

Kinesis と Apache Kafka 以外のコネクタの詳細情報については、コネクタのマニュアルを参照してください。

サンプルおよびチュートリアル

トピック

- チュートリアル: Apache Flink 用 Managed Service での Studio ノートブックの作成
- チュートリアル：永続的な状態を持つアプリケーションとしてデプロイする
- 例

チュートリアル: Apache Flink 用 Managed Service での Studio ノートブックの作成

次のチュートリアルでは、Kinesis データストリームまたは Amazon MSK クラスターからデータを読み取る Studio ノートブックを作成する方法を示しています。

このチュートリアルには、次のセクションが含まれています。

- セットアップ
- データベースを作成します。 AWS Glue
- 次のステップ
- Kinesis Data Streams を使用した Studio ノートブックの作成
- Amazon MSK によるスタジオノートブックの作成
- アプリケーションと依存リソースをクリーンアップします。

セットアップ

バージョン 2 AWS CLI 以降であることを確認してください。最新のものをインストールするには AWS CLI、「AWS CLI バージョン 2 のインストール、更新、アンインストール」を参照してください。

データベースを作成します。 AWS Glue

Studio ノートブックは、Amazon MSK データソースに関するメタデータ用の「AWS Glue」データベースを使用します。

AWS Glue データベースを作成する

1. https://console.aws.amazon.com/glue/ AWS Glue でコンソールを開きます。
2. [Add database] (データベースの追加) を選択します。[データベースの追加] ウィンドウで、[データベース名] に **`default`** を入力します。[作成] を選択します。

次のステップ

このチュートリアルでは、Kinesis Data Streams または Amazon MSK のいずれかを使用する Studio ノートブックを作成できます。

- Kinesis Data Streams : Kinesis Data Streams を使用すると、ソースとして Kinesis Data Streams を使用するアプリケーションをすばやく作成できます。依存リソースとして Kinesis データストリームを作成するだけで済みます。
- Amazon MSK : Amazon MSK で、Amazon MSK クラスターをソースとして使用するアプリケーションを作成します。依存リソースとして Amazon VPC、Amazon EC2 クライアントインスタンス、および Amazon MSK クラスターを作成する必要があります。

Kinesis Data Streams を使用した Studio ノートブックの作成

このチュートリアルでは、Kinesis Data Stream をソースとして使用する Studio ノートブックを作成する方法について説明します。

このチュートリアルには、次のセクションが含まれています。

- セットアップ
- テーブルを作成します。 AWS Glue
- Kinesis Data Streams を使用した Studio ノートブックの作成
- Kinesis データストリームへのデータ送信
- Studio ノートブックをテストします。

セットアップ

Studio ノートブックを作成する前に、Kinesis データストリーム (`ExampleInputStream`) を作成します。アプリケーションはこのストリームをアプリケーションソースとして使用します。

このストリームは Amazon Kinesis コンソールまたは次の AWS CLI コマンドを使用して作成できます。コンソールでの操作方法については、「Amazon Kinesis Data Streams デベロッパーガイド」 の 「データストリームの作成および更新」 を参照してください。ストリーム **`ExampleInputStream`** に名前を付け、[オープンシャード数] を **`1`** に設定します。

を使用してストリーム (`ExampleInputStream`) を作成するには AWS CLI、次の Amazon Kinesis `create-stream` AWS CLI コマンドを使用します。

```
$ aws kinesis create-stream \
--stream-name ExampleInputStream \
--shard-count 1 \
--region us-east-1 \
```

```
--profile adminuser
```

テーブルを作成します。 AWS Glue

Studio ノートブックは、Kinesis Data Streams データソースに関するメタデータに「AWS Glue」データベースを使用します。

> **Note**
>
> データベースは最初に手動で作成することも、ノートブックの作成時に Apache Flink 用 Managed Service に自動的に作成させることもできます。同様に、このセクションで説明されているように手動でテーブルを作成することも、Apache Zeppelin 内のノートブックで Apache Flink 用 Managed Service のテーブル作成コネクタコードで DDL ステートメントを使用してテーブルを作成することもできます。その後、 AWS Glue チェックインしてテーブルが正しく作成されたことを確認できます。

テーブルを作成する

1. AWS Management Console にサインインし、https://console.aws.amazon.com/glue/ AWS Glue のコンソールを開きます。
2. AWS Glue データベースをまだお持ちでない場合は、左側のナビゲーションバーから [データベース] を選択します。[Add database] (データベースの追加) を選択します。[データベースの追加] ウィンドウで、[データベース名] に **`default`** を入力します。[作成] を選択します。
3. 左のナビゲーションバーで、[テーブル] を選択します。「テーブル」ページで、「テーブルを追加」、「テーブルを手動で追加」を選択します。
4. 「テーブルのプロパティの設定」ページで、「テーブル名」に **`stock`** を入力します。以前に作成したデータベースを選択していることを確認してください。[次へ] をクリックします。
5. [データストアの追加] ページで、[Kinesis] を選択します。[Stream name] (ストリーム名) に **`ExampleInputStream`** を入力します。[Kinesis ソース URL] には、 **`https://kinesis.us-east-1.amazonaws.com`** の入力を選択します。[Kinesis ソース URL] をコピーして貼り付ける場合は、先頭または末尾のスペースを必ず削除してください。[次へ] をクリックします。
6. 「分類」ページで「JSON」を選択します。[次へ] をクリックします。
7. スキーマを定義するで、[Add column] を編集して列を追加します。以下のプロパティを持つ列を追加します。

列名	データ型
ticker	**string**
price	**double**

[次へ] をクリックします。

8. 次のページで設定を確認し、「終了」を選択します。
9. テーブルの一覧で、新しく作成したテーブルを選択します。
10. 「テーブル編集」を選択し、キー `managed-flink.proctime` と値 `proctime` を含むプロパティを追加します。
11. [適用] を選択します。

Kinesis Data Streams を使用した Studio ノートブックの作成

アプリケーションで使用するリソースを作成したので、次は Studio ノートブックを作成します。

アプリケーションを作成するには、 AWS Management Console またはを使用できます AWS CLI。

- を使用して Studio ノートブックを作成します。 AWS Management Console
- を使用して Studio ノートブックを作成します。 AWS CLI

を使用して Studio ノートブックを作成します。 AWS Management Console

1. 「https://console.aws.amazon.com/managed-flink/home?region=us-east-1#/applications/dashboard」にある Apache Flink コンソール用 Managed Service を開きます。
2. 「Apache Flink アプリケーション用 Managed Service」ページで、「Studio」タブを選択します。[Studio ノートブックの作成] を選択します。

> **Note**
>
> 入力する Amazon MSK クラスターまたは Kinesis Data Streams を選択して [リアルタイムでデータを処理] を選択することで、Amazon MSK または Kinesis Data Streams コンソールから Studio ノートブックを作成することもできます。

3. [Studio ノートブックの作成] ページで、次の情報を入力します。

- ノートブックの名前に「**MyNotebook**」を入力します。
- 「AWS Glue データベース」の「デフォルト」を選択します。

[Studio ノートブックの作成] を選択します。

4. MyNotebookページで [実行] を選択します。「ステータス」に「実行中」が表示されるまで待ちます。ノートブックの実行中は料金が発生します。

を使用して Studio ノートブックを作成します。 AWS CLI

を使用して Studio ノートブックを作成するには AWS CLI、次の操作を行います。

1. アカウント ID を確認します。アプリケーションを作成する際にこの値が必要になります。
2. ロール `arn:aws:iam::AccountID:role/ZeppelinRole` を作成し、コンソールで自動作成されたロールに以下の権限を追加します。

   `"kinesis:GetShardIterator",`

   `"kinesis:GetRecords",`

   `"kinesis:ListShards"`
3. `create.json` というファイルを次の内容で作成します。プレースホルダー値を、ユーザー自身の情報に置き換えます。

```
{
    "ApplicationName": "MyNotebook",
    "RuntimeEnvironment": "ZEPPELIN-FLINK-3_0",
    "ApplicationMode": "INTERACTIVE",
    "ServiceExecutionRole": "arn:aws:iam::AccountID:role/ZeppelinRole",
    "ApplicationConfiguration": {
        "ApplicationSnapshotConfiguration": {
            "SnapshotsEnabled": false
        },
        "ZeppelinApplicationConfiguration": {
            "CatalogConfiguration": {
                "GlueDataCatalogConfiguration": {
                    "DatabaseARN": "arn:aws:glue:us-east-1:AccountID:database/
default"
                }
            }
```

```
        }
    }
}
```

4. アプリケーションを作成するには、次のコマンドを実行します。

```
aws kinesisanalyticsv2 create-application --cli-input-json file://create.json
```

5. コマンドが完了すると、新しい Studio ノートブックの詳細を示す出力結果が表示されます。次は出力の例です。

```
{
    "ApplicationDetail": {
        "ApplicationARN": "arn:aws:kinesisanalyticsus-
east-1:012345678901:application/MyNotebook",
        "ApplicationName": "MyNotebook",
        "RuntimeEnvironment": "ZEPPELIN-FLINK-3_0",
        "ApplicationMode": "INTERACTIVE",
        "ServiceExecutionRole": "arn:aws:iam::012345678901:role/ZeppelinRole",
...
```

6. アプリケーションを起動するには、次のコマンドを実行します。サンプル値をアカウント ID に置き換えます。

```
aws kinesisanalyticsv2 start-application --application-arn
 arn:aws:kinesisanalyticsus-east-1:012345678901:application/MyNotebook\
```

Kinesis データストリームへのデータ送信

Kinesis データストリームにテストデータを送信するには、次の手順に従います。

1. Kinesis Data Generator を開きます。
2. 「でCognitoユーザーを作成」を選択します。 CloudFormation
3. AWS CloudFormation コンソールが開き、Kinesis データジェネレーターテンプレートが表示されます。[次へ] をクリックします。
4. [Specify component details] (コンポーネントの詳細の指定) ページで、Cognito ユーザーのユーザー名とパスワードを入力します。[次へ] をクリックします。
5. [スタックオプションの設定] ページで、[次へ] を選択します。

6. Kinesis-データジェネレーター-Cognit-Userのレビューページで、IAMリソースを作成する可能性のある [同意します] を選択します。 AWS CloudFormation チェックボックス。[Create Stack] (スタックの作成) を選択します。
7. AWS CloudFormation スタックの作成が完了するまでお待ちください。スタックが完了したら、コンソールで Kinesis-データジェネレーター-Cognito-Userスタックを開き、[出力] タブを選択します。 AWS CloudFormation 出力値としてリストされている URL KinesisDataGeneratorUrlを開きます。
8. [Amazon Kinesis Data Generator] ページで、ステップ 4 で作成した認証情報を使用してログインします。
9. 次のページで、次の値を入力します。

リージョン	**us-east-1**
ストリーム/Firehose ストリーム	**ExampleInputStream**
1 秒あたりのレコード数	**1**

[Record Template] (記録テンプレート) に、次の内容を貼り付けます。

```
{
    "ticker": "{{random.arrayElement(
        ["AMZN","MSFT","GOOG"]
    )}}",
    "price": {{random.number(
        {
            "min":10,
            "max":150
        }
    )}}
}
```

10. [データ送信] を選択します。
11. ジェネレータは、Kinesis データストリームにデータを送信します。

次のセクションを完了する間、ジェネレータを作動させたままにしておきます。

Studio ノートブックをテストします。

このセクションでは、Studio ノートブックを使用して Kinesis データストリームのデータをクエリします。

1. 「https://console.aws.amazon.com/managed-flink/home?region=us-east-1#/applications/dashboard」にある Apache Flink コンソール用 Managed Service を開きます。
2. [Apache Flink アプリケーション用 Managed Service] ページで、[Studio ノートブック] タブを選択します。を選択しますMyNotebook。
3. MyNotebookページで「Apache ツェッペリンで開く」を選択します。

   新しいタブで Apache Zeppelin インターフェイスが開きます。
4. [Zeppelinへようこそ！]ページで [Zeppelin Note] を選択します。
5. 「Zeppelin Note」ページで、新しいノートに次のクエリを入力します。

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

   実行アイコンを選択します。

   しばらくすると、ノートには Kinesis データストリームのデータが表示されます。

アプリケーションの Apache Flink ダッシュボードを開いて運用状況を表示するには、「FLINK JOB」を選択します。Flink Dashboard の詳細については、「Managed Service for Apache Flink デベロッパーガイド」の「Apache Flink ダッシュボード」を参照してください。

Flink ストリーミング SQL クエリの他の例については、「Apache Flink ドキュメント」の「クエリ」を参照してください。

Amazon MSK によるスタジオノートブックの作成

このチュートリアルでは、Amazon MSK クラスターをソースとして使用する Studio ノートブックを作成する方法について説明します。

このチュートリアルには、次のセクションが含まれています。

- セットアップ
- VPC に NAT ゲートウェイを追加

- AWS Glue 接続とテーブルを作成します。
- Amazon MSK による Studio ノートブックの作成
- Amazon MSK クラスターにデータを送信します。
- Studio ノートブックをテストします。

セットアップ

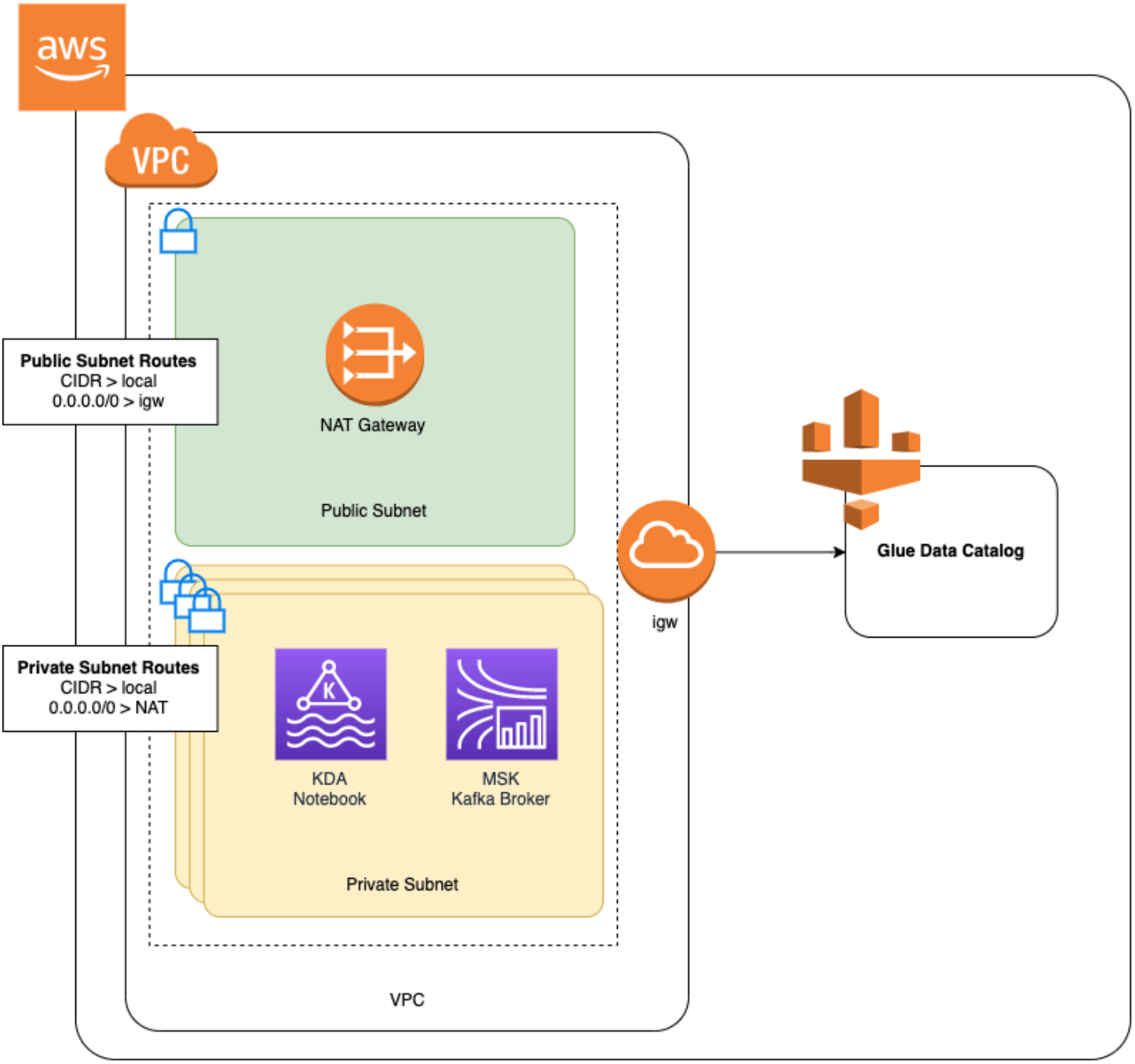
このチュートリアルでは、プレーンテキストでアクセスできる Amazon MSK クラスターが必要です。Amazon MSK クラスターをまだセットアップしていない場合は、「Amazon MSK の使用入門」チュートリアルに従って、Amazon VPC、Amazon MSK クラスター、トピック、および Amazon EC2 クライアントインスタンスを作成してください。

チュートリアルを実行するときは、以下の手順を実行します。

- 「ステップ 3: Amazon MSK クラスターを作成する」のステップ 4 で、 `ClientBroker` 値を `TLS` から **`PLAINTEXT`** に変更します。

VPC に NAT ゲートウェイを追加

「Amazon MSK の使用入門」チュートリアルに従って Amazon MSK クラスターを作成した場合、または既存の Amazon VPC にプライベートサブネット用の NAT ゲートウェイがまだない場合は、Amazon VPC に NAT ゲートウェイを追加する必要があります。アーキテクチャを次の図に示します。



Amazon VPC 用の NAT ゲートウェイを作成するには、以下を実行します。

1. Amazon VPC コンソール (https://console.aws.amazon.com/vpc/) を開きます。
2. 左のナビゲーションバーから、[NAT ゲートウェイ] を選択します。
3. 「NAT ゲートウェイ」ページで「NAT ゲートウェイの作成」を選択します。
4. [NAT ゲートウェイの作成] ページで、以下の値を入力します。

名前-オプション	**`ZeppelinGateway`**
サブネット	AWS KafkaTutorialSubnet1
エラスティック IP アロケーション ID	使用可能な Elastic IP を選択してください。使用可能なエラスティック IP がない場合は、[エラスティック IP の割り当て] を選択してから、コンソールが作成する Elastic IP を選択します。

[Create NAT Gateway] (NAT ゲートウェイの作成) を選択します。

5. 左のナビゲーションバーで、[Route Tables] (ルートテーブル) を選択します。
6. [ルートテーブルの作成] を選択します。
7. [ルートテーブルの作成] ページで、以下の情報を指定します。

   - 名前タグ: **`ZeppelinRouteTable`**
   - VPC: 自分の VPC (VPC などAWS KafkaTutorial) を選択してください。

   [作成] を選択します。
8. ルートテーブルのリストで、を選択します。ZeppelinRouteTable[ルート] タブを選択し、[ルート編集] を選択します。
9. [ルートの編集] ページで、[ルートの追加] を選択します。
10. [送信先] に「**`0.0.0.0/0`**」と入力します。[ターゲット] で [NAT ゲートウェイ] を選択しますZeppelinGateway。[ルートの保存] を選択します。[閉じる] を選びます。
11. 「ルートテーブル」ページで、「サブネットアソシエーション」ZeppelinRouteTableタブを選択した状態で、「サブネットアソシエーション」タブを選択します。「サブネット関連付けの編集」を選択します。
12. 「サブネットアソシエーションの編集」 ページで、「AWS KafkaTutorialSubnet2」と「AWS KafkaTutorialSubnet3」を選択します。[保存] を選択します。

AWS Glue 接続とテーブルを作成します。

Studio ノートブックは、Amazon MSK データソースに関するメタデータ用の「AWS Glue」データベースを使用します。このセクションでは、Amazon MSK AWS Glue クラスターへのアクセス方法を説明する接続と、Studio Notebook AWS Glue などのクライアントにデータソース内のデータを表示する方法を説明するテーブルを作成します。

接続を作成する

1. AWS Management Console にサインインし、https://console.aws.amazon.com/glue/ AWS Glue のコンソールを開きます。
2. AWS Glue データベースをまだお持ちでない場合は、左側のナビゲーションバーから [データベース] を選択します。[Add database] (データベースの追加) を選択します。[データベースの追加] ウィンドウで、[データベース名] に **`default`** を入力します。[作成] を選択します。
3. 左のナビゲーションバーから、[接続]を選択します。[Add connection (接続の追加)] を選択します。
4. 「接続を追加」ウィンドウで、次の値を入力します。

   - [Connection Name] (接続名) に、**`ZeppelinConnection`** と入力します。
   - [接続タイプ] で、[Kafka] を選択します。
   - 「Kafka ブートストラップサーバー URL」には、クラスターのブートストラップブローカーの文字列を指定します。ブートストラップブローカーは、MSK コンソールから、または次の CLI コマンドを入力して取得できます。

     ```
     aws kafka get-bootstrap-brokers --region us-east-1 --cluster-arn ClusterArn
     ```

   - 「SSL 接続が必要」チェックボックスをオフにします。

   [次へ] をクリックします。
5. [VPC] ページで、次の値を入力します。

   - VPC の場合は、VPC の名前 (VPC など AWS KafkaTutorial) を選択します。
   - [サブネット] には 2 を選択します。AWS KafkaTutorialSubnet
   - 「セキュリティグループ」では、使用可能なすべてのグループを選択します。

   [次へ] をクリックします。

6. 「接続プロパティ」/「接続アクセス」ページで 「完了」を選択します。

テーブルを作成する

> **Note**
>
> 次の手順で説明するように手動でテーブルを作成することも、Apache Zeppelin 内のノートブックにある Apache Flink 用 Managed Service のテーブル作成コネクタコードを使用して DDL ステートメントでテーブルを作成することもできます。その後、 AWS Glue チェックインしてテーブルが正しく作成されたことを確認できます。

1. 左のナビゲーションバーで、[テーブル] を選択します。「テーブル」ページで、「テーブルを追加」、「テーブルを手動で追加」を選択します。
2. 「テーブルのプロパティの設定」ページで、「テーブル名」に **stock** を入力します。以前に作成したデータベースを選択していることを確認してください。[次へ] をクリックします。
3. 「データストアの追加」ページで「Kafka」を選択します。[トピック名] には、トピック名 (例:AWS KafkaTutorialTopic) を入力します。[接続] には、を選択しますZeppelinConnection。
4. 「分類」ページで「JSON」を選択します。[次へ] をクリックします。
5. スキーマを定義するで、[Add column] を編集して列を追加します。以下のプロパティを持つ列を追加します。

列名	データ型
ticker	**string**
price	**double**

   [次へ] をクリックします。
6. 次のページで設定を確認し、「終了」を選択します。
7. テーブルの一覧で、新しく作成したテーブルを選択します。
8. 「テーブル編集」を選択し、キー `managed-flink.proctime` と値 `proctime` を含むプロパティを追加します。
9. [適用] を選択します。

Amazon MSK による Studio ノートブックの作成

アプリケーションで使用するリソースを作成したので、次は Studio ノートブックを作成します。

AWS Management Console またはを使用してアプリケーションを作成できます AWS CLI。

- を使用して Studio ノートブックを作成します。 AWS Management Console
- を使用して Studio ノートブックを作成します。 AWS CLI

> **Note**
>
> Amazon MSK コンソールから既存のクラスターを選択し、「データをリアルタイムで処理」を選択することで Studio ノートブックを作成することもできます。

を使用して Studio ノートブックを作成します。 AWS Management Console

1. 「https://console.aws.amazon.com/managed-flink/home?region=us-east-1#/applications/dashboard」にある Apache Flink コンソール用 Managed Service を開きます。
2. 「Apache Flink アプリケーション用 Managed Service」ページで、「Studio」タブを選択します。「Studio ノートブックの作成」を選択します。

   > **Note**
   >
   > Amazon MSK または Kinesis Data Streams コンソールから Studio ノートブックを作成するには、入力の Amazon MSK クラスターまたは Kinesis データストリームを選択し、「データをリアルタイムで処理」を選択します。

3. [Studio ノートブックの作成] ページで、次の情報を入力します。

   - 「Studio ノートブック名」に **`MyNotebook`** を入力します。
   - 「AWS Glue データベース」の「デフォルト」を選択します。

   [Studio ノートブックの作成] を選択します。
4. MyNotebookページで [設定] タブを選択します。「Networking」セクションで、「編集」を選択します。

5. [ネットワークの編集] MyNotebook ページで、Amazon MSK クラスターに基づく VPC 設定を選択します。「Amazon MSK クラスター」には Amazon MSK クラスターを選択します。[変更を保存] を選択します。
6. MyNotebookページで [Run] を選択します。「ステータス」に「実行中」が表示されるまで待ちます。

を使用して Studio ノートブックを作成します。 AWS CLI

を使用して Studio ノートブックを作成するには AWS CLI、次の操作を行います。

1. 次の情報があることを確認します。アプリケーションを作成するにはこれらの値が必要です。
   - アカウント ID。
   - Amazon MSK クラスターを含む Amazon VPC 用のサブネット ID やセキュリティグループ ID。
2. `create.json` というファイルを次の内容で作成します。プレースホルダー値を、ユーザー自身の情報に置き換えます。

```
{
    "ApplicationName": "MyNotebook",
    "RuntimeEnvironment": "ZEPPELIN-FLINK-3_0",
    "ApplicationMode": "INTERACTIVE",
    "ServiceExecutionRole": "arn:aws:iam::AccountID:role/ZeppelinRole",
    "ApplicationConfiguration": {
        "ApplicationSnapshotConfiguration": {
            "SnapshotsEnabled": false
        },
        "VpcConfigurations": [
            {
                "SubnetIds": [
                    "SubnetID 1",
                    "SubnetID 2",
                    "SubnetID 3"
                ],
                "SecurityGroupIds": [
                    "VPC Security Group ID"
                ]
            }
        ],
        "ZeppelinApplicationConfiguration": {
```

```
            "CatalogConfiguration": {
                "GlueDataCatalogConfiguration": {
                    "DatabaseARN": "arn:aws:glue:us-east-1:AccountID:database/
default"
                }
            }
        }
    }
}
```

3. アプリケーションを作成するには、次のコマンドを実行します。

```
aws kinesisanalyticsv2 create-application --cli-input-json file://create.json
```

4. コマンドが完了すると、次のような出力が表示され、新しい Studio ノートブックの詳細が表示されます。

```
{
    "ApplicationDetail": {
        "ApplicationARN": "arn:aws:kinesisanalyticsus-
east-1:012345678901:application/MyNotebook",
        "ApplicationName": "MyNotebook",
        "RuntimeEnvironment": "ZEPPELIN-FLINK-3_0",
        "ApplicationMode": "INTERACTIVE",
        "ServiceExecutionRole": "arn:aws:iam::012345678901:role/ZeppelinRole",
...
```

5. アプリケーションを起動するには、次のコマンドを実行します。サンプル値をアカウント ID に置き換えます。

```
aws kinesisanalyticsv2 start-application --application-arn
 arn:aws:kinesisanalyticsus-east-1:012345678901:application/MyNotebook\
```

Amazon MSK クラスターにデータを送信します。

このセクションでは、Amazon EC2 クライアントで Python スクリプトを実行して Amazon MSK データソースにデータを送信します。

1. Amazon EC2 クライアントに接続します。
2. 以下のコマンドを実行して Python バージョン 3、Pip、および Kafka for Python パッケージをインストールし、アクションを確認します。

```
sudo yum install python37
curl -O https://bootstrap.pypa.io/get-pip.py
python3 get-pip.py --user
pip install kafka-python
```

3. 次のコマンドを入力して、 AWS CLI クライアントマシン上でを設定します。

```
aws configure
```

アカウントの認証情報と **us-east-1** を region に入力します。

4. stock.py というファイルを次の内容で作成します。サンプル値を Amazon MSK クラスターの Bootstrap Brokers 文字列に置き換え、トピックがそうでない場合はトピック名を更新してください。AWS KafkaTutorialTopic

```
from kafka import KafkaProducer
import json
import random
from datetime import datetime

BROKERS = "<<Bootstrap Broker List>>"
producer = KafkaProducer(
    bootstrap_servers=BROKERS,
    value_serializer=lambda v: json.dumps(v).encode('utf-8'),
    retry_backoff_ms=500,
    request_timeout_ms=20000,
    security_protocol='PLAINTEXT')


def getStock():
    data = {}
    now = datetime.now()
    str_now = now.strftime("%Y-%m-%d %H:%M:%S")
    data['event_time'] = str_now
    data['ticker'] = random.choice(['AAPL', 'AMZN', 'MSFT', 'INTC', 'TBV'])
    price = random.random() * 100
    data['price'] = round(price, 2)
    return data


while True:
    data =getStock()
```

```
    # print(data)
    try:
        future = producer.send("AWSKafkaTutorialTopic", value=data)
        producer.flush()
        record_metadata = future.get(timeout=10)
        print("sent event to Kafka! topic {} partition {} offset
{}".format(record_metadata.topic, record_metadata.partition,
record_metadata.offset))
    except Exception as e:
        print(e.with_traceback())
```

5. 次のコマンドを使用してスクリプトを実行します。

```
$ python3 stock.py
```

6. 以下のセクションを実行している間は、スクリプトを実行したままにしておきます。

Studio ノートブックをテストします。

このセクションでは、Studio ノートブックを使用して Amazon MSK クラスターのデータをクエリします。

1. 「https://console.aws.amazon.com/managed-flink/home?region=us-east-1#/applications/dashboard」にある Apache Flink 用 Managed Serviceコンソールを開きます。
2. [Apache Flink アプリケーション用 Managed Service] ページで、[Studio ノートブック] タブを選択します。を選択してくださいMyNotebook。
3. MyNotebookページで、「Apache ツェッペリンで開く」を選択します。

   新しいタブで Apache Zeppelin インターフェイスが開きます。
4. 「Zeppelinへようこそ！」でページで「Zeppelinの新ノート」を選択します。
5. 「Zeppelin Note」ページで、新しいノートに次のクエリを入力します。

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

   実行アイコンを選択します。

   アプリケーションは Amazon MSK クラスターのデータを表示します。

アプリケーションの Apache Flink ダッシュボードを開いて運用状況を表示するには、「FLINK JOB」を選択します。Flink Dashboard の詳細については、「Managed Service for Apache Flink デベロッパーガイド」の「Apache Flink ダッシュボード」を参照してください。

Flink ストリーミング SQL クエリの他の例については、「Apache Flink ドキュメント」の「クエリ」を参照してください。

アプリケーションと依存リソースをクリーンアップします。

Studio ノートブックの削除

1. Apache Flink コンソール用 Managed Service を開きます。
2. を選択しますMyNotebook。
3. [Actions (アクション)] を選択してから [Delete (削除)] を選択します。

AWS Glue データベースと接続を削除します。

1. https://console.aws.amazon.com/glue/ AWS Glue でコンソールを開きます。
2. 左のナビゲーションペインで、[Databases] (データベース) を選択します。[Default] (デフォルト) の横にあるチェックボックスをチェックして選択します。[Action] (アクション)、[Delete Database] (データベースの削除) を選択します。[Confirm] (確認) をクリックして、選択内容を確認します。
3. 左のナビゲーションバーから、[接続]を選択します。ZeppelinConnection横のチェックボックスをチェックして選択します。[Action] (アクション)、[Delete Connection] (接続の削除) を選択します。[Confirm] (確認) をクリックして、選択内容を確認します。

ポリシーと IAM ロールを削除するには

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. 左のナビゲーションメニューから [Roles] (ロール) を選択します。
3. ZeppelinRole検索バーを使用してロールを検索します。
4. ZeppelinRoleロールを選択します。ロールの削除 を選択します。削除を確定します。

CloudWatch ロググループを削除します。

コンソールを使用してアプリケーションを作成すると、 CloudWatch コンソールによって自動的にロググループとログストリームが作成されます。 AWS CLIを使用してアプリケーションを作成した場合、ロググループとストリームはありません。

1. https://console.aws.amazon.com/cloudwatch/ CloudWatch でコンソールを開きます。
2. 左側のナビゲーションバーで、[Log groups] (ロググループ) を選択します。
3. /AWSKinesisAnalytics/MyNotebookロググループを選択します。
4. [アクション]、[ロググループの削除] の順にクリックします。削除を確定します。

Kinesis Data Streams リソースのクリーンアップ

Kinesis ストリームを削除するには、Kinesis Data Streams コンソールを開き、Kinesis ストリームを選択して、[Actions]、[Delete] を選択します。

MSK リソースをクリーンアップする

このチュートリアル用の Amazon MSK クラスターを作成した場合は、このセクションのステップに従います。このセクションでは、Amazon EC2 クライアントインスタンス、Amazon VPC、および Amazon MSK クラスターをクリーンアップする方法について説明します。

Amazon MSK クラスターを削除する

このチュートリアル用に Amazon MSK クラスターを作成した場合は、以下の手順に従います。

1. https://console.aws.amazon.com/msk/home?region=us-east-1#/home/ で Amazon MSK コンソールを開きます。
2. を選択してくださいAWS KafkaTutorialCluster。[削除] を選択します。表示されるウィンドウに **`delete`** を入力し、選択を確定します。

クライアントインスタンスの終了

このチュートリアル用に Amazon EC2 クライアントインスタンスを作成した場合は、次の手順に従います。

1. Amazon EC2 コンソール (https://console.aws.amazon.com/ec2/) を開きます。
2. 左側のナビゲーションペインから、[Instances] (インスタンス) を選択します。

3. ZeppelinClient横のチェックボックスを選択して選択します。
4. [Instance state] (インスタンスの状態)、[Terminate instance] (インスタンスの終了)の順に選択します。

Amazon VPC の削除

このチュートリアル用に Amazon VPC を作成した場合は、次の手順に従います。

1. Amazon EC2 コンソール (https://console.aws.amazon.com/ec2/) を開きます。
2. 左のナビゲーションバーから [Network Interfaces] (ネットワークインターフェース) を選択します。
3. 検索ボックスに VPC ID を入力し、Enter キーを押します。
4. テーブルヘッダーのチェックボックスを選択して、表示されているすべてのネットワークインターフェースを選択します。
5. [アクション]、[デタッチ] の順にクリックします。表示されるウィンドウで、[Force detachment] (強制デタッチメント) の [Enable] (有効化) を選択します。[Detach] (デタッチ) を選択し、すべてのネットワークインターフェースが [Available] (利用可能) ステータスになるまで待ちます。
6. テーブルヘッダーのチェックボックスを選択して、表示されているすべてのネットワークインターフェースを再び選択します。
7. [アクション]、[削除] の順に選択します。アクションを確認します。
8. Amazon VPC コンソール (https://console.aws.amazon.com/vpc/) を開きます。
9. AWS KafkaTutorialVPC を選択します。[Actions] (アクション)、[Delete VPC] (VPC を削除) の順に選択します。**`delete`** を入力して、削除を確定します。

チュートリアル：永続的な状態を持つアプリケーションとしてデプロイする

以下のチュートリアルでは、Apache Flink アプリケーション用 Managed Service として、 Studio ノートブックを永続的な状態でデプロイする方法を示します。

このチュートリアルには、次のセクションが含まれています。

- セットアップ
- AWS Management Consoleを使用して永続的な状態のアプリケーションをデプロイします。

- AWS CLIを使用して永続的な状態のアプリケーションをデプロイします。

セットアップ

Kinesis Data Streams または Amazon MSK のいずれかを使用して、 Studio ノートブックチュートリアルの作成 に従って新しい Studio ノートブックを作成します。Studio ノートブック `ExampleTestDeploy` に名前を付けます。

AWS Management Consoleを使用して永続的な状態のアプリケーションをデプロイします。

1. パッケージ化されたコードを保存する S3 バケットの場所を、コンソールの「アプリケーションコードの場所 - オプショナル」に追加します。これにより、ノートブックから直接アプリケーションをデプロイして実行できるようになります。
2. アプリケーションの役割に必要な権限を追加して、Amazon S3 バケットの読み取りと書き込みに使用している役割を有効にし、 Apache Flink アプリケーションのホスティングサービスを起動します。
   - Amazon S3 FullAccess
   - アマゾンが管理-flinkFullAccess
   - ソース、宛先、VPC へのアクセス (該当する場合)。詳細については、 Studio ノートブックの IAM パーミッション を参照してください。
3. 次のサンプルコードを使用してください。

```
%flink.ssql(type=update)
CREATE TABLE exampleoutput (
  'ticket' VARCHAR,
  'price' DOUBLE
)
WITH (
  'connector' = 'kinesis',
  'stream' = 'ExampleOutputStream',
  'aws.region' = 'us-east-1',
  'scan.stream.initpos' = 'LATEST',
  'format' = 'json'
);

INSERT INTO exampleoutput SELECT ticker, price FROM exampleinputstream
```

4. この機能を有効にすると、ノートブック内の各ノートの右上隅に、ノートブックの名前が記載された新しいドロップダウンが表示されます。以下の操作を行うことができます。

   - Studio ノートブックの設定は、 AWS Management Consoleで確認できます。
   - Zeppelin Note を作成して、Amazon S3 にエクスポートします。この時点で、アプリケーションの名前を入力し、「Build および Export」を選択します。エクスポートが完了すると、通知が届きます。
   - 必要に応じて、Amazon S3 で実行可能ファイルの追加テストを表示して実行することができます。
   - 構築が完了すると、永続的な状態と自動スケーリング機能を備えた Kinesis ストリーミングアプリケーションとしてコードをデプロイできるようになります。
   - ドロップダウンを使用して、「Zeppelin Note を Kinesis ストリーミングアプリケーションとしてデプロイ」を選択します。アプリケーション名を確認し、[AWS コンソール経由でデプロイ] を選択します。
   - Apache Flink AWS Management Console アプリケーション用マネージドサービスを作成するページが表示されます。アプリケーション名、並列処理、コードの場所、デフォルトの Glue DB、VPC (該当する場合)、IAM ロールが事前に入力されていることに注意してください。IAM ロールがソースと宛先に対して必要な権限を持っていることを確認します。永続的なアプリケーション状態管理のため、スナップショットはデフォルトで有効になっています。
   - [Create application] を選択します。
   - 「コンフィグ」を選択して任意の設定を変更し、「Run」を選択してストリーミング・アプリケーションを起動することができます。

AWS CLIを使用して永続的な状態のアプリケーションをデプロイします。

を使用してアプリケーションをデプロイするには AWS CLI、Beta 2 AWS CLI 情報で提供されるサービスモデルを使用するようにを更新する必要があります。更新されたサービスモデルの使用方法については、 セットアップ を参照してください。

次のコード例で、新規の Studio ノートブックを作成します。

```
aws kinesisanalyticsv2 create-application \
     --application-name <app-name> \
     --runtime-environment ZEPPELIN-FLINK-3_0 \
     --application-mode INTERACTIVE \
     --service-execution-role <iam-role>
     --application-configuration '{
```

```
       "ZeppelinApplicationConfiguration": {
         "CatalogConfiguration": {
           "GlueDataCatalogConfiguration": {
             "DatabaseARN": "arn:aws:glue:us-east-1:<account>:database/<glue-database-
name>"
           }
         }
       },
       "FlinkApplicationConfiguration": {
         "ParallelismConfiguration": {
           "ConfigurationType": "CUSTOM",
           "Parallelism": 4,
           "ParallelismPerKPU": 4
         }
       },
       "DeployAsApplicationConfiguration": {
            "S3ContentLocation": {
               "BucketARN": "arn:aws:s3:::<s3bucket>",
               "BasePath": "/something/"
            }
        },
       "VpcConfigurations": [
         {
           "SecurityGroupIds": [
             "<security-group>"
           ],
           "SubnetIds": [
             "<subnet-1>",
             "<subnet-2>"
           ]
         }
       ]
     }' \
     --region us-east-1
```

次のコード例で、Studio ノートブックを起動します。

```
aws kinesisanalyticsv2 start-application \
    --application-name <app-name> \
    --region us-east-1 \
    --no-verify-ssl
```

次のコードは、アプリケーションの Apache Zeppelin ノートブックページの URL を返します。

```
aws kinesisanalyticsv2 create-application-presigned-url \
    --application-name <app-name> \
    --url-type ZEPPELIN_UI_URL \

    --region us-east-1 \
    --no-verify-ssl
```

例

以下のクエリ例は、Studio ノートブックでウィンドウクエリを使用してデータを分析する方法を示しています。

- Amazon MSK/Apache Kafka によるテーブルの作成
- Kinesis によるテーブルの作成
- タンブリングウィンドウ
- スライディングウィンドウ
- Interactive SQL
- BlackHole SQL コネクタ
- データジェネレーター
- Interactive Scala
- Interactive Python
- インタラクティブな Python、SQL、Scala
- Cross-account Kinesis データストリーム

Apache Flink SQL クエリ設定の情報については、「インタラクティブなデータ分析のための Zeppelin ノートブック上の Flink」を参照してください。

Apache Flink ダッシュボードでアプリケーションを表示するには、アプリケーションの「Zeppelin Note」ページで「FLINK JOB」を選択します。

ウィンドウクエリの詳細については、「Apache Flink ドキュメント」の「Windows」を参照してください。

Apache Flink Streaming SQL クエリの他の例については、「Apache Flink ドキュメント」の「クエリ」を参照してください。

Amazon MSK/Apache Kafka によるテーブルの作成

Apache Flink Studio 用 Managed Service を搭載した Amazon MSK Flink コネクタを使用して、Plaintext、SSL または IAM 認証で接続を認証できます。要件に応じて特定のプロパティを使用してテーブルを作成します。

```
-- Plaintext connection

CREATE TABLE your_table (
  `column1` STRING,
  `column2` BIGINT
) WITH (
  'connector' = 'kafka',
  'topic' = 'your_topic',
  'properties.bootstrap.servers' = '<bootstrap servers>',
  'scan.startup.mode' = 'earliest-offset',
  'format' = 'json'
);

-- SSL connection

CREATE TABLE your_table (
  `column1` STRING,
  `column2` BIGINT
) WITH (
  'connector' = 'kafka',
   'topic' = 'your_topic',
  'properties.bootstrap.servers' = '<bootstrap servers>',
  'properties.security.protocol' = 'SSL',
  'properties.ssl.truststore.location' = '/usr/lib/jvm/java-11-amazon-corretto/lib/
security/cacerts',
  'properties.ssl.truststore.password' = 'changeit',
  'properties.group.id' = 'myGroup',
  'scan.startup.mode' = 'earliest-offset',
  'format' = 'json'
);

-- IAM connection (or for MSK Serverless)

CREATE TABLE your_table (
  `column1` STRING,
  `column2` BIGINT
) WITH (
```

```
  'connector' = 'kafka',
  'topic' = 'your_topic',
  'properties.bootstrap.servers' = '<bootstrap servers>',
  'properties.security.protocol' = 'SASL_SSL',
  'properties.sasl.mechanism' = 'AWS_MSK_IAM',
  'properties.sasl.jaas.config' = 'software.amazon.msk.auth.iam.IAMLoginModule
 required;',
  'properties.sasl.client.callback.handler.class' =
 'software.amazon.msk.auth.iam.IAMClientCallbackHandler',
  'properties.group.id' = 'myGroup',
  'scan.startup.mode' = 'earliest-offset',
  'format' = 'json'
);
```

これらのプロパティは、「Apache Kafka SQL Connector」の他のプロパティと組み合わせることができます。

Kinesis によるテーブルの作成

次の例では、Kinesis を使用してテーブルを作成します。

```
CREATE TABLE KinesisTable (
  `column1` BIGINT,
  `column2` BIGINT,
  `column3` BIGINT,
  `column4` STRING,
  `ts` TIMESTAMP(3)
)
PARTITIONED BY (column1, column2)
WITH (
  'connector' = 'kinesis',
  'stream' = 'test_stream',
  'aws.region' = '<region>',
  'scan.stream.initpos' = 'LATEST',
  'format' = 'csv'
);
```

使用可能な他のプロパティの詳細については、「Amazon Kinesis Data Streams SQL Connector」を参照してください。

タンブリングウィンドウ

次の Flink Streaming SQL クエリは、 ZeppelinTopic テーブルから 5 秒ごとのタンブリングウィンドウの最大値を選択します。

```
%flink.ssql(type=update)
SELECT TUMBLE_END(event_time, INTERVAL '5' SECOND) as winend, MAX(price) as
 five_second_high, ticker
FROM ZeppelinTopic
GROUP BY ticker, TUMBLE(event_time, INTERVAL '5' SECOND)
```

スライディングウィンドウ

次の Apache Flink Streaming SQL クエリは、 ZeppelinTopic テーブルから 5 秒ごとのスライディングウィンドウの最大値を選択します。

```
%flink.ssql(type=update)
SELECT HOP_END(event_time, INTERVAL '3' SECOND, INTERVAL '5' SECOND) AS winend,
 MAX(price) AS sliding_five_second_max
FROM ZeppelinTopic//or your table name in AWS Glue
GROUP BY HOP(event_time, INTERVAL '3' SECOND, INTERVAL '5' SECOND)
```

Interactive SQL

この例では、イベント時間と処理時間の最大値、キー値テーブルの値の合計を出力します。the section called “データジェネレーター” のサンプル・データ生成スクリプトが実行されていることを確認します。Studio ノートブックでフィルタリングや結合などの他の SQL クエリを試すには、Apache Flink ドキュメントのApache Flink ドキュメント:「クエリ」を参照してください。

```
%flink.ssql(type=single, parallelism=4, refreshInterval=1000, template=<h1>{2}</h1>
 records seen until <h1>Processing Time: {1}</h1> and <h1>Event Time: {0}</h1>)

-- An interactive query prints how many records from the `key-value-stream` we have
 seen so far, along with the current processing and event time.
SELECT
  MAX(`et`) as `et`,
  MAX(`pt`) as `pt`,
  SUM(`value`) as `sum`
FROM
  `key-values`
```

```
%flink.ssql(type=update, parallelism=4, refreshInterval=1000)

-- An interactive tumbling window query that displays the number of records observed
 per (event time) second.
-- Browse through the chart views to see different visualizations of the streaming
 result.
SELECT
  TUMBLE_START(`et`, INTERVAL '1' SECONDS) as `window`,
  `key`,
  SUM(`value`) as `sum`
FROM
  `key-values`
GROUP BY
  TUMBLE(`et`, INTERVAL '1' SECONDS),
  `key`;
```

BlackHole SQL コネクタ

BlackHole SQL コネクタでは、クエリをテストするために Kinesis データストリームまたは Amazon MSK クラスターを作成する必要はありません。SQL コネクタの詳細については、 BlackHoleApache Flink ドキュメントのBlackHole 「SQL Connector」を参照してください。この例では、デフォルトカタログはインメモリカタログです。

```
%flink.ssql

CREATE TABLE default_catalog.default_database.blackhole_table (
 `key` BIGINT,
 `value` BIGINT,
 `et` TIMESTAMP(3)
) WITH (
 'connector' = 'blackhole'
)
```

```
%flink.ssql(parallelism=1)

INSERT INTO `test-target`
SELECT
  `key`,
  `value`,
  `et`
FROM
```

```
  `test-source`
WHERE
  `key` > 3
```

```
%flink.ssql(parallelism=2)

INSERT INTO `default_catalog`.`default_database`.`blackhole_table`
SELECT
  `key`,
  `value`,
  `et`
FROM
  `test-target`
WHERE
  `key` > 7
```

データジェネレーター

この例では Scala を使用してサンプルデータを生成します。このサンプルデータを使用して、さまざまなクエリをテストできます。テーブル作成ステートメントを使用して key-values テーブルを作成します。

```
import org.apache.flink.streaming.api.functions.source.datagen.DataGeneratorSource
import org.apache.flink.streaming.api.functions.source.datagen.RandomGenerator
import org.apache.flink.streaming.api.scala.DataStream

import java.sql.Timestamp

// ad-hoc convenience methods to be defined on Table
implicit class TableOps[T](table: DataStream[T]) {
    def asView(name: String): DataStream[T] = {
      if (stenv.listTemporaryViews.contains(name)) {
        stenv.dropTemporaryView("`" + name + "`")
      }
      stenv.createTemporaryView("`" + name + "`", table)
      return table;
    }
}
```

```
%flink(parallelism=4)
val stream = senv
```

```
 .addSource(new DataGeneratorSource(RandomGenerator.intGenerator(1, 10), 1000))
 .map(key => (key, 1, new Timestamp(System.currentTimeMillis)))
 .asView("key-values-data-generator")
```

```
%flink.ssql(parallelism=4)
-- no need to define the paragraph type with explicit parallelism (such as
 "%flink.ssql(parallelism=2)")
-- in this case the INSERT query will inherit the parallelism of the of the above
 paragraph
INSERT INTO `key-values`
SELECT
 `_1` as `key`,
 `_2` as `value`,
 `_3` as `et`
FROM
 `key-values-data-generator`
```

Interactive Scala

これは the section called “Interactive SQL” の Scala 翻訳です。Scala の他の例については、Apache Flink ドキュメントの「Table API」を参照してください。

```
%flink
import org.apache.flink.api.scala._
import org.apache.flink.table.api._
import org.apache.flink.table.api.bridge.scala._

// ad-hoc convenience methods to be defined on Table
implicit class TableOps(table: Table) {
    def asView(name: String): Table = {
      if (stenv.listTemporaryViews.contains(name)) {
        stenv.dropTemporaryView(name)
      }
      stenv.createTemporaryView(name, table)
      return table;
    }
}
```

```
%flink(parallelism=4)

// A view that computes many records from the `key-values` we have seen so far, along
 with the current processing and event time.
```

```
val query01 = stenv
  .from("`key-values`")
  .select(
    $"et".max().as("et"),
    $"pt".max().as("pt"),
    $"value".sum().as("sum")
  ).asView("query01")
```

```
%flink.ssql(type=single, parallelism=16, refreshInterval=1000, template=<h1>{2}</h1>
 records seen until <h1>Processing Time: {1}</h1> and <h1>Event Time: {0}</h1>)

-- An interactive query prints the query01 output.
SELECT * FROM query01
```

```
%flink(parallelism=4)

// An tumbling window view that displays the number of records observed per (event
 time) second.
val query02 = stenv
  .from("`key-values`")
  .window(Tumble over 1.seconds on $"et" as $"w")
  .groupBy($"w", $"key")
  .select(
    $"w".start.as("window"),
    $"key",
    $"value".sum().as("sum")
  ).asView("query02")
```

```
%flink.ssql(type=update, parallelism=4, refreshInterval=1000)

-- An interactive query prints the query02 output.
-- Browse through the chart views to see different visualizations of the streaming
 result.
SELECT * FROM `query02`
```

Interactive Python

これは the section called “Interactive SQL” の Python 翻訳です。Python の他のサンプルについては、Apache Flink ドキュメントの「Table API」を参照してください。

```
%flink.pyflink
```

```
from pyflink.table.table import Table

def as_view(table, name):
  if (name in st_env.list_temporary_views()):
    st_env.drop_temporary_view(name)
  st_env.create_temporary_view(name, table)
  return table

Table.as_view = as_view
```

```
%flink.pyflink(parallelism=16)

# A view that computes many records from the `key-values` we have seen so far, along
 with the current processing and event time
st_env \
  .from_path("`keyvalues`") \
  .select(", ".join([
    "max(et) as et",
    "max(pt) as pt",
    "sum(value) as sum"
  ])) \
  .as_view("query01")
```

```
%flink.ssql(type=single, parallelism=16, refreshInterval=1000, template=<h1>{2}</h1>
 records seen until <h1>Processing Time: {1}</h1> and <h1>Event Time: {0}</h1>)

-- An interactive query prints the query01 output.
SELECT * FROM query01
```

```
%flink.pyflink(parallelism=16)

# A view that computes many records from the `key-values` we have seen so far, along
 with the current processing and event time
st_env \
  .from_path("`key-values`") \
  .window(Tumble.over("1.seconds").on("et").alias("w")) \
  .group_by("w, key") \
  .select(", ".join([
    "w.start as window",
    "key",
    "sum(value) as sum"
  ])) \
```

```
  .as_view("query02")
```

```
%flink.ssql(type=update, parallelism=16, refreshInterval=1000)

-- An interactive query prints the query02 output.
-- Browse through the chart views to see different visualizations of the streaming
 result.
SELECT * FROM `query02`
```

インタラクティブな Python、SQL、Scala

ノートブックでは、SQL、Python、Scala を自由に組み合わせてインタラクティブな分析を行うことができます。永続的な状態を持つアプリケーションとしてデプロイする予定の Studio ノートブックでは、SQL と Scala を組み合わせて使用できます。この例では、無視されるセクションと、永続的な状態でアプリケーションにデプロイされるセクションを示しています。

```
%flink.ssql
CREATE TABLE `default_catalog`.`default_database`.`my-test-source` (
  `key` BIGINT NOT NULL,
  `value` BIGINT NOT NULL,
  `et` TIMESTAMP(3) NOT NULL,
  `pt` AS PROCTIME(),
  WATERMARK FOR `et` AS `et` - INTERVAL '5' SECOND
)
WITH (
  'connector' = 'kinesis',
  'stream' = 'kda-notebook-example-test-source-stream',
  'aws.region' = 'eu-west-1',
  'scan.stream.initpos' = 'LATEST',
  'format' = 'json',
  'json.timestamp-format.standard' = 'ISO-8601'
)
```

```
%flink.ssql
CREATE TABLE `default_catalog`.`default_database`.`my-test-target` (
  `key` BIGINT NOT NULL,
  `value` BIGINT NOT NULL,
  `et` TIMESTAMP(3) NOT NULL,
  `pt` AS PROCTIME(),
  WATERMARK FOR `et` AS `et` - INTERVAL '5' SECOND
)
```

```
WITH (
  'connector' = 'kinesis',
  'stream' = 'kda-notebook-example-test-target-stream',
  'aws.region' = 'eu-west-1',
  'scan.stream.initpos' = 'LATEST',
  'format' = 'json',
  'json.timestamp-format.standard' = 'ISO-8601'
)
```

```
%flink()

// ad-hoc convenience methods to be defined on Table
implicit class TableOps(table: Table) {
  def asView(name: String): Table = {
    if (stenv.listTemporaryViews.contains(name)) {
      stenv.dropTemporaryView(name)
    }
    stenv.createTemporaryView(name, table)
    return table;
  }
}
```

```
%flink(parallelism=1)
val table = stenv
  .from("`default_catalog`.`default_database`.`my-test-source`")
  .select($"key", $"value", $"et")
  .filter($"key" > 10)
  .asView("query01")
```

```
%flink.ssql(parallelism=1)

-- forward data
INSERT INTO `default_catalog`.`default_database`.`my-test-target`
SELECT * FROM `query01`
```

```
%flink.ssql(type=update, parallelism=1, refreshInterval=1000)

-- forward data to local stream (ignored when deployed as application)
SELECT * FROM `query01`
```

```
%flink
```

```
// tell me the meaning of life (ignored when deployed as application!)
print("42!")
```

Cross-account Kinesis データストリーム

Studio ノートブックを所有するアカウント以外のアカウントにおける Kinesis データ・ストリームを使用するには、Studio ノートブックが実行されているアカウントにサービス実行ロールを作成し、データストリームを所有するアカウントにロール信頼ポリシーを作成します。Create table DDL ステートメントの Kinesis コネクタで `aws.credentials.provider`、`aws.credentials.role.arn`、`aws.credentials.role.sessionN` を使用して、データストリームに対してテーブルを作成します。

Studio ノートブックアカウントには、次のサービス実行ロールを使用します。

```
{
 "Sid": "AllowNotebookToAssumeRole",
 "Effect": "Allow",
 "Action": "sts:AssumeRole"
 "Resource": "*"
}
```

データストリームアカウントには、 `AmazonKinesisFullAccess` ポリシーと以下のロール信頼ポリシーを使用してください。

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Principal": {
 "AWS": "arn:aws:iam::<accountID>:root"
 },
 "Action": "sts:AssumeRole",
 "Condition": {}
 }
 ]
}
```

create table ステートメントには以下の段落を使用してます。

```
%flink.ssql
CREATE TABLE test1 (
name VARCHAR,
age BIGINT
) WITH (
'connector' = 'kinesis',
'stream' = 'stream-assume-role-test',
'aws.region' = 'us-east-1',
'aws.credentials.provider' = 'ASSUME_ROLE',
'aws.credentials.role.arn' = 'arn:aws:iam::<accountID>:role/stream-assume-role-test-
role',
'aws.credentials.role.sessionName' = 'stream-assume-role-test-session',
'scan.stream.initpos' = 'TRIM_HORIZON',
'format' = 'json'
)
```

トラブルシューティング

このセクションには、Studio ノートブックのトラブルシューティング情報が記載されています。

動かなくなったアプリケーションの停止

一時的な状態で停止しているアプリケーションを停止するには、 Forceパラメータを に設定して StopApplicationアクションを呼び出しますtrue。詳細については、「Apache Flink 用 Managed Service デベロッパーガイド」内の「Running Applications」を参照してください。

インターネットにアクセスできない VPC に永続的な状態のアプリケーションとしてデプロイする

Managed Service for Apache Flink Studio deploy-as-application 関数は、インターネットアクセスのない VPC アプリケーションをサポートしていません。Studio でアプリケーションを構築し、Apache Flink 用 Managed Service を使用して Flink アプリケーションを手動で作成し、Notebookで構築した zip ファイルを選択することをお勧めします。

以下のステップは、この方法の概要を説明します。

1. Studio アプリケーションをビルドして Amazon S3 にエクスポートします。これは zip ファイルである必要があります。
2. Amazon S3 にある zip ファイルのロケーションを参照するコードパスを使用して、Apache Flink アプリケーション用 Managed Service を手動で作成します。さらに、以下の env 変数

（合計 2 つ のgroupID、3 つの var ）を使用してアプリケーションを設定する必要があります。

3. kinesis.analytics.flink.run.options

   a. python: source/note.py

   b. jarfile: lib/PythonApplicationDependencies.jar

4. managed.deploy_as_app.options

   - DatabaseARN:「*<glue database ARN (Amazon Resource Name) >*」

5. アプリケーションが使用するサービスについて、Apache Flink Studio 用 Managed Service と Apache Flink IAM ロール用 Managed Service にアクセス許可を与える必要がある場合があります。両方のアプリに同じ IAM ロールを使用できます。

D eploy-as-app サイズとビルド時間の短縮

Studio deploy-as-app for Python アプリケーションは、必要なライブラリを特定できないため、Python 環境で利用可能なすべてをパッケージ化します。これにより、必要以上の deploy-as-app サイズになる可能性があります。次の手順では、依存関係をアンインストールして deploy-as-app Python アプリケーションのサイズを縮小する方法を示します。

Studio deploy-as-app の機能を使用して Python アプリケーションを構築する場合は、アプリケーションが依存していない場合、プリインストールされた Python パッケージをシステムから削除することを検討してください。これにより、最終的なアーティファクトサイズを小さくしてアプリケーションサイズのサービス制限を超えないようにするだけでなく、 deploy-as-app 機能を使用してアプリケーションの構築時間を短縮できます。

次のコマンドを実行すると、インストールされているすべての Python パッケージとそれぞれのインストールサイズを一覧表示し、サイズの大きいパッケージを選択的に削除できます。

```
%flink.pyflink

!pip list --format freeze | awk -F = {'print $1'} | xargs pip show | grep -E
 'Location:|Name:' | cut -d ' ' -f 2 | paste -d ' ' - - | awk '{gsub("-","_",$1); print
 $2 "/" tolower($1)}' | xargs du -sh 2> /dev/null | sort -hr
```

> **Note**
>
> Flink Python が動作するためには apache-beam が必要です。このパッケージとその依存関係は絶対に削除しないでください。

以下は、Studio V2 にプリインストールされている Python パッケージの一覧です。これらの削除を検討できます。

```
scipy
statsmodels
plotnine
seaborn
llvmlite
bokeh
pandas
matplotlib
botocore
boto3
numba
```

Zeppelin ノートブックから Python パッケージを削除するには:

1. 削除する前に、アプリケーションがそのパッケージやそれを利用するパッケージに依存しているかどうかを確認してください。pipdeptree を使うと、パッケージの依存パッケージを特定できます。
2. 以下のコマンドを実行してパッケージを削除します。

   ```
   %flink.pyflink
   !pip uninstall -y <package-to-remove>
   ```

3. 誤って削除したパッケージを取り戻す必要がある場合は、以下のコマンドを実行します。

   ```
   %flink.pyflink
   !pip install <package-to-install>
   ```

Example 例: Python アプリケーションを deploy-as-app 機能でデプロイする前に、 **scipy**パッケージを削除します。

1. pipdeptree を使用して、scipy に依存している他のパッケージやプロジェクトを検出し、scipy を安全に削除できるかどうかを確認します。

   - ノートブックからツールをインストールします。

   ```
   %flink.pyflink
   !pip install pipdeptree
   ```

   - 以下を実行して、scipy の逆依存関係ツリーを取得します。

   ```
   %flink.pyflink
   !pip -r -p scipy
   ```

   次のような出力が表示されます (これは要約版です):

   ```
   ...
   ------------------------------------------------------------------------
   scipy==1.8.0
   ### plotnine==0.5.1 [requires: scipy>=1.0.0]
   ### seaborn==0.9.0 [requires: scipy>=0.14.0]
   ### statsmodels==0.12.2 [requires: scipy>=1.1]
       ### plotnine==0.5.1 [requires: statsmodels>=0.8.0]
   ```

2. アプリケーションでの seaborn、statsmodels および plotnine の使用法を注意深く確認してください。アプリケーションが scipy、seaborn、statemodels、plotnine のいずれにも依存していない場合は、これらのパッケージをすべて削除することも、アプリケーションが必要としないパッケージだけを削除することもできます。
3. 次のコマンドを実行してパッケージを削除します。

   ```
   !pip uninstall -y scipy plotnine seaborn statemodels
   ```

ジョブのキャンセル

このセクションでは、Apache Zeppelin から実行できない Apache Flink ジョブをキャンセルする方法を説明します。このようなジョブをキャンセルしたい場合は、Apache Flink ダッシュボードに移動し、ジョブ ID をコピーして、以下の例のいずれかでそれを使用してください。

個々のジョブをキャンセルするには:

```
%flink.pyflink
import requests

requests.patch("https://zeppelin-flink:8082/jobs/[job_id]", verify=False)
```

実行中のジョブをすべてキャンセルするには:

```
%flink.pyflink
import requests

r = requests.get("https://zeppelin-flink:8082/jobs", verify=False)
jobs = r.json()['jobs']

for job in jobs:
    if (job["status"] == "RUNNING"):
        print(requests.patch("https://zeppelin-flink:8082/jobs/{}".format(job["id"]),
 verify=False))
```

すべてのジョブをキャンセルするには:

```
%flink.pyflink
import requests

r = requests.get("https://zeppelin-flink:8082/jobs", verify=False)
jobs = r.json()['jobs']

for job in jobs:
    requests.patch("https://zeppelin-flink:8082/jobs/{}".format(job["id"]),
 verify=False)
```

Apache Flink インタープリタの再起動

Studio ノートブック内の Apache Flink インタープリタを再起動するには

1. 画面の右上にある [Configuration] を選択します。
2. [Interpreter] を選択します。
3. [再起動] を選択してから [OK] を選択します。

付録:カスタム IAM ポリシーの作成

通常、マネージド IAM ポリシーを使用して、アプリケーションが依存リソースにアクセスできるようにします。アプリケーションの権限をより細かく制御する必要がある場合は、カスタム IAM ポリシーを使用できます。このセクションには、カスタム IAM ポリシーの例が含まれています。

> **Note**
>
> 次のポリシーの例では、プレースホルダーテキストをアプリケーションの値に置き換えます。

このトピックには、次のセクションが含まれています。

- AWS Glue
- CloudWatch ログ
- Kinesis Streams
- Amazon MSK クラスター

AWS Glue

次のポリシー例では、 AWS Glue データベースにアクセスするためのアクセス許可を付与します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "GlueTable",
            "Effect": "Allow",
            "Action": [
                "glue:GetConnection",
                "glue:GetTable",
                "glue:GetTables",
                "glue:GetDatabase",
                "glue:CreateTable",
                "glue:UpdateTable"
            ],
            "Resource": [
                "arn:aws:glue:<region>:<accountId>:connection/*",
                "arn:aws:glue:<region>:<accountId>:table/<database-name>/*",
```

```
                "arn:aws:glue:<region>:<accountId>:database/<database-name>",
                "arn:aws:glue:<region>:<accountId>:database/hive",
                "arn:aws:glue:<region>:<accountId>:catalog"
            ]
        },
        {
            "Sid": "GlueDatabase",
            "Effect": "Allow",
            "Action": "glue:GetDatabases",
            "Resource": "*"
        }
    ]
}
```

CloudWatch ログ

次のポリシーは、 CloudWatch ログへのアクセス許可を付与します。

```
{
      "Sid": "ListCloudwatchLogGroups",
      "Effect": "Allow",
      "Action": [
        "logs:DescribeLogGroups"
      ],
      "Resource": [
        "arn:aws:logs:<region>:<accountId>:log-group:*"
      ]
    },
    {
      "Sid": "ListCloudwatchLogStreams",
      "Effect": "Allow",
      "Action": [
        "logs:DescribeLogStreams"
      ],
      "Resource": [
        "<logGroupArn>:log-stream:*"
      ]
    },
    {
      "Sid": "PutCloudwatchLogs",
      "Effect": "Allow",
      "Action": [
        "logs:PutLogEvents"
```

```
      ],
      "Resource": [
        "<logStreamArn>"
      ]
    }
```

> **Note**
>
> 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 CloudWatch ログにアクセスするために必要なポリシーをアプリケーションロールに追加します。

Kinesis Streams

アプリケーションは、ソースまたはデスティネーションに Kinesis Stream を使用できます。アプリケーションには、ソースストリームから読み取るための読み取り許可と、デスティネーションストリームに書き込むための書き込み許可が必要です。

以下のポリシーは、ソースとして使用される Kinesis Stream からの読み取り権限を付与します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "KinesisShardDiscovery",
      "Effect": "Allow",
      "Action": "kinesis:ListShards",
      "Resource": "*"
    },
    {
      "Sid": "KinesisShardConsumption",
      "Effect": "Allow",
      "Action": [
        "kinesis:GetShardIterator",
        "kinesis:GetRecords",
        "kinesis:DescribeStream",
        "kinesis:DescribeStreamSummary",
        "kinesis:RegisterStreamConsumer",
        "kinesis:DeregisterStreamConsumer"
      ],
      "Resource": "arn:aws:kinesis:<region>:<accountId>:stream/<stream-name>"
    },
```

```
    {
      "Sid": "KinesisEfoConsumer",
      "Effect": "Allow",
      "Action": [
        "kinesis:DescribeStreamConsumer",
        "kinesis:SubscribeToShard"
      ],
      "Resource": "arn:aws:kinesis:<region>:<account>:stream/<stream-name>/consumer/*"
    }
  ]
}
```

次のポリシーは、デスティネーションとして使用される Kinesis Stream への書き込み権限を付与します。

```
{
    "Version": "2012-10-17",
    "Statement": [
         {
            "Sid": "KinesisStreamSink",
            "Effect": "Allow",
            "Action": [
                "kinesis:PutRecord",
                "kinesis:PutRecords",
                "kinesis:DescribeStreamSummary",
                "kinesis:DescribeStream"
            ],
            "Resource": "arn:aws:kinesis:<region>:<accountId>:stream/<stream-name>"
        }
    ]
}
```

アプリケーションが暗号化された Kinesis ストリームにアクセスする場合、ストリームとストリームの暗号化キーにアクセスするための追加権限を付与する必要があります。

次のポリシーは、暗号化されたソースストリームとストリームの暗号化キーへのアクセス権限を付与します。

```
{
      "Sid": "ReadEncryptedKinesisStreamSource",
      "Effect": "Allow",
      "Action": [
```

```
        "kms:Decrypt"
      ],
      "Resource": [
        "<inputStreamKeyArn>"
      ]
    }
    ,
```

次のポリシーは、暗号化されたデスティネーションストリームとストリームの暗号化キーへのアクセス権限を付与します。

```
{
      "Sid": "WriteEncryptedKinesisStreamSink",
      "Effect": "Allow",
      "Action": [
        "kms:GenerateDataKey"
      ],
      "Resource": [
        "<outputStreamKeyArn>"
      ]
    }
```

Amazon MSK クラスター

Amazon MSK クラスターへのアクセスを許可するには、クラスターの VPC へのアクセスを許可します。Amazon VPC にアクセスするためのポリシーの例については、「VPC Application Permissions」を参照してください。

Amazon Managed Service for Apache Flink の開始方法 (DataStream API)

このセクションでは、Managed Service for Apache Flink と DataStream API の基本概念を紹介します。アプリケーションの作成とテストに使用できるオプションについて説明します。また、このガイドのチュートリアルを完了し、初めてアプリケーションを作成するのに必要なツールのインストール方法についても説明します。

トピック

- Managed Service for Apache Flink アプリケーションのコンポーネントを確認する
- 演習を完了するための前提条件を満たす
- ステップ 1: AWS アカウントを設定し、管理者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: Managed Service for Apache Flink アプリケーションを作成して実行する
- ステップ 4: リソースをクリーンアップ AWS する
- ステップ 5: リソースを使用して次のステップを完了する

Managed Service for Apache Flink アプリケーションのコンポーネントを確認する

Apache Flink アプリケーション用 Managed Service では、Apache Flink ランタイムを使用して入力を処理し、出力を生成する Java/Apache Maven または Scala アプリケーションを使用してデータを処理します。

Apache Flink アプリケーション用 Managed Serviceには、以下のコンポーネントがあります。

- 「ランタイムプロパティ:」「ランタイムプロパティ」を使用すると、アプリケーションコードを再コンパイルせずにアプリケーションを設定できます。
- 「ソース:」 アプリケーションは 「ソース」 を使用してデータを消費します。ソースコネクタは、Kinesis データストリーム、Amazon S3 バケットなどからデータを読み取ります。詳細については、「[Sources] (出典)」を参照してください。
- 「オペレータ:」アプリケーションは 1 つ以上の「オペレータ」を使用してデータを処理します。オペレータはデータを変換、強化、または集約できます。詳細については、 DataStream API 演算子 を参照してください。

- 「シンク:」アプリケーションは「シンク」を使用して外部ソースにデータを生成します。シンクコネクタは、Kinesis データストリーム、Firehose ストリーム、Amazon S3 バケットなどにデータを書き込みます。詳細については、「シンク」を参照してください。

アプリケーションコードを作成、コンパイル、パッケージ化したら、コードパッケージを Amazon Simple Storage Service (Amazon S3) バケットにアップロードします。次に、Apache Flink アプリケーション用 Managed Serviceを作成します。コードパッケージの場所、ストリーミングデータソースとして Kinesis データストリームを渡し、通常はアプリケーションの処理済みデータを受け取るストリーミングまたはファイルの場所を渡します。

演習を完了するための前提条件を満たす

このガイドの手順を完了するには、以下が必要です。

- Java 開発キット (JDK) バージョン 11。JAVA_HOME 環境変数を、JDK のインストール場所を指すように設定します。
- 開発環境 (Eclipse Java Neon や IntelliJ Idea など) を使用してアプリケーションを開発し、コンパイルすることをお勧めします。
- Git クライアント。Git クライアントをまだインストールしていない場合は、インストールします。
- Apache Maven Compiler Plugin。Maven が作業パスに含まれている必要があります。Apache Maven のインストールをテストするには、次のように入力します。

```
$ mvn -version
```

開始するには、ステップ 1: AWS アカウントを設定し、管理者ユーザーを作成するに進みます。

ステップ 1: AWS アカウントを設定し、管理者ユーザーを作成する

Managed Service for Apache Flink を初めて使用する前に、以下のタスクを完了してください：

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、次のステップ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環として、通話呼び出しを受け取り、電話キーパッドで検証コードを入力するように求められます。

   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。セキュリティのベストプラクティスとして、ユーザーに管理アクセスを割り当て、ルートユーザーのみを使用してルートユーザーアクセスが必要なタスクを実行してください。

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

のセキュリティ保護 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 「ルートユーザーの仮想 MFA デバイスを有効にする (コンソール)」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、「 ユーザーガイド」の「デフォルト でユーザーアクセス IAM アイデンティティセンターディレクトリを設定するAWS IAM Identity Center 」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 「 アクセスポータルへのサインイン」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「グループの参加」を参照してください。

プログラム的なアクセス権を付与する

ユーザーが の AWS 外部で を操作する場合は、プログラムによるアクセスが必要です AWS Management Console。プログラムによるアクセスを許可する方法は、 にアクセスするユーザーのタイプによって異なります AWS。

ユーザーにプログラマチックアクセス権を付与するには、以下のいずれかのオプションを選択します。

プログラマチックアクセス権を必要とするユーザー	目的	方法
ワークフォースアイデンティティ (IAM Identity Center で管理されているユーザー)	一時的な認証情報を使用して、、 AWS SDKs AWS CLI、または AWS APIs。	使用するインターフェイス用の手引きに従ってください。 • については AWS CLI、「 ユーザーガイド」の AWS CLI 「 を使用するための の設定 AWS IAM Identity CenterAWS Command Line Interface 」を参照してください。 • AWS SDKs 、ツール、AWS APIs「 SDK とツールのリファレンスガイド」の「IAM Identity Center 認証」を参照してください。AWS SDKs
IAM	一時的な認証情報を使用して、、 AWS SDKs AWS CLI、または AWS APIs。	「IAM ユーザーガイド」の「 AWS リソースでの一時的な認証情報の使用」の手順に従います。
IAM	(非推奨) 長期認証情報を使用して、、 AWS SDKs AWS CLI、または AWS APIs。	使用するインターフェイス用の手引きに従ってください。 • については AWS CLI、「 AWS Command Line Interface ユーザーガイド」の「IAM ユーザー認証情報を使用した認証」を参照してください。 • AWS SDKs「 SDK とツールのリファレンスガイド」の「長期的な認証情報を使

プログラマチックアクセス権を必要とするユーザー	目的	方法
		用した認証」を参照してください。 AWS SDKs • AWS APIsユーザーガイド」の「IAM ユーザーのアクセスキーの管理」を参照してください。

次のステップ

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

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

このステップでは、Managed Service for Apache Flink で使用する をダウンロードして設定 AWS CLI します。

> **Note**
>
> このガイドの使用開始実習では、操作を実行するために、アカウントの管理者の認証情報 (`adminuser`) を使用していることが前提となっています。

> **Note**
>
> が既に AWS CLI インストールされている場合は、アップグレードして最新の機能を取得する必要がある場合があります。詳細については、「AWS Command Line Interface ユーザーガイド」の「AWS Command Line Interfaceのインストール」を参照してください。のバージョンを確認するには AWS CLI、次のコマンドを実行します。
>
> ```
> aws --version
> ```
>
> このチュートリアルの演習では、次の AWS CLI バージョン 以降が必要です。

```
aws-cli/1.16.63
```

を設定するには 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。

   - AWS Command Line Interfaceのインストール
   - AWS CLIの設定

2. 管理者ユーザーの名前付きプロファイルを config ファイルに追加します AWS CLI 。 AWS CLI コマンドを実行するときに、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 AWS リージョンのリストについては、「」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。

   > **Note**
   >
   > このチュートリアルのサンプルコードとコマンドでは、米国西部 (オレゴン) リージョンを使用しています。別の リージョンを使用するには、このチュートリアルのコードとコマンドのリージョンを、使用したいリージョンに変更します。

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
   ```

AWS アカウントと をセットアップしたら AWS CLI、次の演習を試してサンプルアプリケーションを設定し、 end-to-end セットアップをテストできます。

次のステップ

ステップ 3: Managed Service for Apache Flink アプリケーションを作成して実行する

ステップ 3: Managed Service for Apache Flink アプリケーションを作成して実行する

この演習では、データストリームをソースおよびシンクとして使用して、Managed Service for Apache Flink アプリケーションを作成します。

このセクションには、以下のステップが含まれています。

- 2 つの Amazon Kinesis データストリームを作成する
- サンプルレコードを入力ストリームに書き込む
- Apache Flink ストリーミング Java コードをダウンロードして調べる
- アプリケーションコードをコンパイルする
- Apache Flink ストリーミング Java コードをアップロードする
- Managed Service for Apache Flink アプリケーションを作成して実行する
- 次のステップ

2 つの Amazon Kinesis データストリームを作成する

この演習で Apache Flink アプリケーションのマネージドサービスを作成する前に、2 つの Kinesis データストリーム (`ExampleInputStream` と `ExampleOutputStream`) を作成する必要があります。アプリケーションでは、これらのストリームを使用してアプリケーションの送信元と送信先のストリームを選択します。

これらのストリームは Amazon Kinesis コンソールまたは次の AWS CLI コマンドを使用して作成できます。コンソールでの操作方法については、「Amazon Kinesis Data Streams デベロッパーガイド」 の 「データストリームの作成および更新」 を参照してください。

データストリームを作成するには (AWS CLI)

1. 最初のストリーム (`ExampleInputStream`) を作成するには、次の Amazon Kinesis `create-stream` AWS CLI コマンドを使用します。

```
$ aws kinesis create-stream \
```

```
--stream-name ExampleInputStream \
--shard-count 1 \
--region us-west-2 \
--profile adminuser
```

2. 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 ExampleOutputStream \
--shard-count 1 \
--region us-west-2 \
--profile adminuser
```

サンプルレコードを入力ストリームに書き込む

このセクションでは、Python スクリプトを使用して、アプリケーションが処理するサンプルレコードをストリームに書き込みます。

> **Note**
>
> このセクションでは AWS SDK for Python (Boto) が必要です。

1. 次の内容で、stock.py という名前のファイルを作成します。

```
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', region_name='us-west-2'))
```

2. このチュートリアルの後半では、アプリケーションにデータを送信する stock.py スクリプトを実行します。

```
$ python stock.py
```

Apache Flink ストリーミング Java コードをダウンロードして調べる

この例の Java アプリケーションコードは、 から入手できます GitHub。アプリケーションコードをダウンロードするには、次の操作を行います。

1. 次のコマンドを使用してリモートリポジトリのクローンを作成します。

```
git clone https://github.com/aws-samples/amazon-managed-service-for-apache-flink-examples.git
```

2. amazon-managed-service-for-apache-flink-examples/tree/main/java/GettingStarted ディレクトリに移動します。

アプリケーションコードに関して、以下の点に注意してください。

- 「Project Object Model (pom.xml)」ファイルには、Managed Service for Apache Flink 用ライブラリなど、アプリケーションの設定と依存関係に関する情報が含まれています。
- BasicStreamingJob.java ファイルには、アプリケーションの機能を定義する main メソッドが含まれています。

- アプリケーションは Kinesis ソースを使用して、ソースストリームから読み取りを行います。次のスニペットでは、Kinesis ソースが作成されます。

```
return env.addSource(new FlinkKinesisConsumer<>(inputStreamName,
                new SimpleStringSchema(), inputProperties));
```

- アプリケーションでは、ソースおよびシンクコネクタを作成し、StreamExecutionEnvironment オブジェクトを使用して外部リソースにアクセスします。
- アプリケーションでは、静的プロパティを使用してソースおよびシンクコネクタを作成します。動的なアプリケーションプロパティを使用するには、createSourceFromApplicationProperties および createSinkFromApplicationProperties メソッドを使用してコネクタを作成します。これらのメソッドは、アプリケーションのプロパティを読み取ってコネクタを設定します。

  ランタイムプロパティの詳細については、ランタイムプロパティ を参照してください。

アプリケーションコードをコンパイルする

このセクションでは、Apache Maven コンパイラを使用してアプリケーション用の Java コードを作成します。Apache Maven と Java 開発キット (JDK) をインストールする方法については、演習を完了するための前提条件を満たすを参照してください。

アプリケーションコードをコンパイルするには

1. アプリケーションコードを使用するには、コードをコンパイルして JAR ファイルにパッケージ化します。コードのコンパイルとパッケージ化には次の 2 通りの方法があります。

   - コマンドライン Maven ツールを使用します。pom.xml ファイルが格納されているディレクトリで次のコマンドを実行して JAR ファイルを作成します。

     ```
     mvn package -Dflink.version=1.19.1
     ```

   - 開発環境を使用します。詳細については、開発環境のドキュメントを参照してください。

     > **Note**
     >
     > 提供されているソースコードは Java 11 のライブラリーに依存しています。

パッケージは JAR ファイルとしてアップロードすることも、圧縮して ZIP ファイルとしてアップロードすることもできします。を使用してアプリケーションを作成する場合は AWS CLI、コードコンテンツタイプ (JAR または ZIP) を指定します。

2. コンパイル中にエラーが発生した場合は、JAVA_HOME 環境変数が正しく設定されていることを確認します。

アプリケーションのコンパイルに成功すると、次のファイルが作成されます。

```
target/amazon-msf-java-stream-app-1.0.jar
```

Apache Flink ストリーミング Java コードをアップロードする

このセクションでは、Amazon Simple Storage Service (Amazon S3) バケットを作成し、アプリケーションコードをアップロードします。

アプリケーションコードをアップロードするには

1. Amazon S3 コンソール (https://console.aws.amazon.com/s3/) を開きます。
2. [バケットを作成] を選択します。
3. [Bucket name (バケット名)] フィールドに**ka-app-code-*<username>***と入力します。バケット名にユーザー名などのサフィックスを追加して、グローバルに一意にします。[次へ] をクリックします。
4. 設定オプションのステップでは、設定をそのままにし、[次へ] を選択します。
5. アクセス許可の設定のステップでは、設定をそのままにし、[次へ] を選択します。
6. [バケットを作成] を選択します。
7. Amazon S3 コンソールで、ka-app-code-***<username>*** バケットを選択し、アップロード を選択します。
8. ファイルの選択のステップで、[ファイルを追加] を選択します。前のステップで作成した amazon-msf-java-stream-app-1.0.jar ファイルに移動します。[次へ] をクリックします。
9. オブジェクトの設定を変更する必要はないので、[アップロード] を選択してください。

アプリケーションコードが Amazon S3 バケットに保存され、アプリケーションからアクセスできるようになります。

Managed Service for Apache Flink アプリケーションを作成して実行する

コンソールまたは AWS CLIのいずれかを使用してManaged Service for Apache Flink を作成し、実行することができます。

> **Note**
>
> コンソールを使用してアプリケーションを作成すると、 AWS Identity and Access Management (IAM) と Amazon CloudWatch Logs リソースが自動的に作成されます。を使用してアプリケーションを作成するときは AWS CLI、これらのリソースを個別に作成します。

トピック

- アプリケーションの作成と実行 (コンソール)
- アプリケーションの作成と実行 (AWS CLI)

アプリケーションの作成と実行 (コンソール)

以下の手順を実行し、コンソールを使用してアプリケーションを作成、設定、更新、および実行します。

アプリケーションの作成

1. https://console.aws.amazon.com/flink で Apache Flink 用 Managed Serviceコンソールを開く
2. Managed Service for Apache Flinkのダッシュボードで、「分析アプリケーションの作成」 を選択します。
3. 「Managed Service for Apache Flink-アプリケーションの作成」ページで、次のようにアプリケーションの詳細を入力します。
   - [アプリケーション名] には **`MyApplication`** と入力します。
   - [Description (説明)] に **`My java test app`** と入力します。
   - [ランタイム] には、[Apache Flink] を選択します。
   - バージョンプルダウンは Apache Flink バージョン 1.19 のままにします。
4. [アクセス許可] には、[IAM ロールの作成 / 更新**`kinesis-analytics-MyApplication-us-west-2`**] を選択します。
5. [Create application] を選択します。

> **Note**
>
> コンソールを使用して Apache Flink アプリケーション用 Managed Service を作成する場合は、IAM ロールとポリシーをアプリケーションが自動的に作成するオプションを選択できます。アプリケーションではこのロールとポリシーを使用して、依存リソースにアクセスします。これらの IAM リソースは、次のようにアプリケーション名とリージョンを使用して命名されます。
>
> - ポリシー: `kinesis-analytics-service-MyApplication-us-west-2`
> - ロール: `kinesisanalytics-MyApplication-us-west-2`

IAM ポリシーを編集する

IAM ポリシーを編集し、Kinesis Data Streamsにアクセスするための許可を追加します。

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. [ポリシー] を選択します。前のセクションでコンソールによって作成された **kinesis-analytics-service-MyApplication-us-west-2** ポリシーを選択します。
3. [概要] ページで、[ポリシーの編集] を選択します。[JSON] タブを選択します。
4. 次のポリシー例で強調表示されているセクションをポリシーに追加します。サンプルのアカウント ID (*012345678901*) を自分のアカウント ID に置き換えます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ReadCode",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": [
                "arn:aws:s3:::ka-app-code-username/amazon-msf-java-stream-
app-1.0.jar"
            ]
        },
        {
            "Sid": "DescribeLogGroups",
```

```
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:*"
            ]
        },
        {
            "Sid": "DescribeLogStreams",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogStreams"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:*"
            ]
        },
        {
            "Sid": "PutLogEvents",
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:kinesis-analytics-log-stream"
            ]
        },
        {
            "Sid": "ReadInputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
        },
        {
            "Sid": "WriteOutputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleOutputStream"
        }
```

```
    ]
}
```

アプリケーションを設定する

1. MyApplication ページで、 の設定 を選択します。
2. [Configure application] ページで、[Code location] を次のように指定します。
   - [Amazon S3 バケット] で、**ka-app-code-*<username>***と入力します。
   - [Amazon S3 オブジェクトへのパス] で、**amazon-msf-java-stream-app-1.0.jar**と入力します。
3. [Access to application resources] の [Access permissions] では、[Create / update IAM role**kinesis-analytics-MyApplication-us-west-2**] を選択します。
4. [Properties] の [Group ID] には、**ProducerConfigProperties**と入力します。
5. 次のアプリケーションのプロパティと値を入力します。

グループ ID	キー	値
FlinkApplicationProperties	**flink.inputstream.initpos**	**LATEST**
FlinkApplicationProperties	**aws.region**	**us-west-2**
FlinkApplicationProperties	**AggregationEnabled**	**false**

6. [Monitoring] の [Monitoring metrics level] が [Application] に設定されていることを確認します。
7. CloudWatch のログ記録では、有効化チェックボックスをオンにします。
8. [更新] を選択します。

> **Note**
>
> Amazon CloudWatch ログ記録を有効にすると、Managed Service for Apache Flink によってロググループとログストリームが作成されます。これらのリソースの名前は次のとおりです。

- ロググループ: `/aws/kinesis-analytics/MyApplication`
- ログストリーム: `kinesis-analytics-log-stream`

アプリケーションを実行する

Flink ジョブグラフは、アプリケーションを実行し、Apache Flink ダッシュボードを開き、目的の Flink ジョブを選択すると表示できます。

アプリケーションの停止

MyApplication ページで、停止 を選択します。アクションを確認します。

アプリケーションの更新

コンソールを使用して、アプリケーションのプロパティ、モニタリング設定、アプリケーション JAR の場所またはファイル名などのアプリケーション設定を更新できます。アプリケーションコードを更新する必要がある場合は、アプリケーション JAR を Amazon S3 バケットから再ロードすることもできます。

MyApplication ページで、 の設定 を選択します。アプリケーションの設定を更新し、[更新] を選択します。

アプリケーションの作成と実行 (AWS CLI)

このセクションでは、 を使用して Managed Service for Apache Flink アプリケーション AWS CLI を作成して実行します。Managed Service for Apache Flink は、 `kinesisanalyticsv2` AWS CLI コマンドを使用して Managed Service for Apache Flink アプリケーションを作成して操作します。

許可ポリシーを作成する

> **Note**
>
> アプリケーションのアクセス権限ポリシーとロールを作成する必要があります。これらの IAM リソースを作成しない場合、アプリケーションはそのデータストリームやログストリームにアクセスできません。

まず、2 つのステートメントを含むアクセス許可ポリシーを作成します。1 つは、ソースストリームの `read` アクションに対するアクセス許可を付与し、もう 1 つはシンクストリームの `write` アク

ションに対するアクセス許可を付与します。次に、IAM ロール (次のセクションで作成) にポリシーをアタッチします。そのため、 Managed Service for Apache Flinkがこのロールを引き受けると、ソースストリームからの読み取りとシンクストリームへの書き込みを行うために必要なアクセス許可がサービスに付与されます。

次のコードを使用して AKReadSourceStreamWriteSinkStream アクセス許可ポリシーを作成します。*username* を Amazon S3 バケットの作成に使用したユーザー名に置き換え、アプリケーションコードを保存します。Amazon リソースネーム (ARN) のアカウント ID (*012345678901*) を自分のアカウント ID に置き換えます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "S3",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": ["arn:aws:s3:::ka-app-code-username",
                "arn:aws:s3:::ka-app-code-username/*"
            ]
        },
        {
            "Sid": "ReadInputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
        },
        {
            "Sid": "WriteOutputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleOutputStream"
        }
    ]
}
```

アクセス許可ポリシーを作成する step-by-step 手順については、IAM ユーザーガイドの「チュートリアル: 最初のカスタマー管理ポリシーの作成とアタッチ」を参照してください。

> **Note**
>
> その他のアマゾンサービスにアクセスするには、 AWS SDK for Javaを使用します。Managed Service for Apache Flink は、アプリケーションに関連付けられているサービス実行 IAM ロールに、SDK が必要とする認証情報を自動的に設定します。追加の手順は必要ありません。

IAM ロールを作成する

このセクションでは、Managed Service for Apache Flink アプリケーションがソースストリームを読み取り、シンクストリームに書き込むために想定できる IAM ロールを作成します。

Apache Flink 用 Managed Service は、許可なしにはストリームにアクセスできません。IAM ロールを介してこれらの許可を付与します。各 IAM ロールには、2 つのポリシーがアタッチされます。信頼ポリシーは、ロールを引き受けるための許可を Managed Service for Apache Flink 付与し、許可ポリシーは、ロールを引き受けた後に Managed Service for Apache Flink が実行できる事柄を決定します。

前のセクションで作成したアクセス許可ポリシーをこのロールにアタッチします。

IAM ロールを作成するには

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. ナビゲーションペインで [Roles (ロール)]、[Create Role (ロールの作成)] の順に選択します。
3. [信頼されるエンティティの種類を選択] で、[AWS のサービス] を選択します。[このロールを使用するサービスを選択] で、[Kinesis Analytics] を選択します。[ユースケースの選択] で、[Kinesis Analytics] を選択します。

   [Next: Permissions] (次のステップ: 許可) を選択します。
4. [アクセス権限ポリシーをアタッチする] ページで、[Next: Review] (次: 確認) を選択します。ロールを作成した後に、アクセス許可ポリシーをアタッチします。
5. [Create role (ロールの作成)] ページで、ロールの名前に **`MF-stream-rw-role`** を入力します。[ロールの作成] を選択します。

これで、MF-stream-rw-role と呼ばれる新しい IAM ロールが作成されます。次に、ロールの信頼ポリシーとアクセス許可ポリシーを更新します。

6. アクセス許可ポリシーをロールにアタッチします。

> **Note**
>
> この演習では、Managed Service for Apache Flink が、Kinesis データストリーム (ソース) からのデータの読み取りと、別の Kinesis データストリームへの出力の書き込みの両方を実行するためにこのロールを引き受けます。このため、前のステップで作成したポリシー、the section called “許可ポリシーを作成する” をアタッチします。

   a. [概要] ページで、[アクセス許可] タブを選択します。
   b. [Attach Policies (ポリシーのアタッチ)] を選択します。
   c. 検索ボックスに**AKReadSourceStreamWriteSinkStream**(前のセクションで作成したポリシー) と入力します。
   d. AKReadSourceStreamWriteSinkStream ポリシーを選択し、ポリシー をアタッチ を選択します。

これで、アプリケーションがリソースにアクセスするために使用するサービスの実行ロールが作成されました。新しいロールの ARN を書き留めておきます。

ロールの作成 step-by-step 手順については、IAM ユーザーガイドの「IAM ロールの作成 (コンソール)」を参照してください。

Managed Service for Apache Flink アプリケーションを作成する

1. 次の JSON コードを create_request.json という名前のファイルに保存します。サンプルロールの ARN を、前に作成したロールの ARN に置き換えます。バケット ARN のサフィックス (*username*) を、前のセクションで選択したサフィックスに置き換えます。サービス実行ロールのサンプルのアカウント ID (*012345678901*) を、自分のアカウント ID に置き換えます。

```
{
    "ApplicationName": "test",
    "ApplicationDescription": "my java test app",
    "RuntimeEnvironment": "FLINK-1_19",
    "ServiceExecutionRole": "arn:aws:iam::012345678901:role/MF-stream-rw-role",
```

```
    "ApplicationConfiguration": {
        "ApplicationCodeConfiguration": {
            "CodeContent": {
                "S3ContentLocation": {
                    "BucketARN": "arn:aws:s3:::ka-app-code-username",
                    "FileKey": "amazon-msf-java-stream-app-1.0.jar"
                }
            },
            "CodeContentType": "ZIPFILE"
        },
        "EnvironmentProperties":  {
         "PropertyGroups": [
            {
               "PropertyGroupId": "ProducerConfigProperties",
               "PropertyMap" : {
                    "flink.stream.initpos" : "LATEST",
                    "aws.region" : "us-west-2",
                    "AggregationEnabled" : "false"
               }
            },
            {
               "PropertyGroupId": "ConsumerConfigProperties",
               "PropertyMap" : {
                    "aws.region" : "us-west-2"
               }
            }
         ]
      }
    }
}
```

2. 前述のリクエストを指定して CreateApplication アクションを実行し、アプリケーションを作成します。

```
aws kinesisanalyticsv2 create-application --cli-input-json file://
create_request.json
```

これでアプリケーションが作成されました。次のステップでは、アプリケーションを起動します。

アプリケーションを起動する

このセクションでは、StartApplication アクションを使用してアプリケーションを起動します。

アプリケーションを起動するには

1. 次の JSON コードを start_request.json という名前のファイルに保存します。

```
{
    "ApplicationName": "test",
    "RunConfiguration": {
        "ApplicationRestoreConfiguration": {
         "ApplicationRestoreType": "RESTORE_FROM_LATEST_SNAPSHOT"
         }
    }
}
```

2. 前述のリクエストを指定して StartApplication アクションを実行し、アプリケーションを起動します。

```
aws kinesisanalyticsv2 start-application --cli-input-json file://start_request.json
```

アプリケーションが実行されます。Amazon CloudWatch コンソールで Managed Service for Apache Flink メトリクスをチェックして、アプリケーションが動作していることを確認できます。

アプリケーションの停止

このセクションでは、StopApplication アクションを使用してアプリケーションを停止します。

アプリケーションを停止するには

1. 次の JSON コードを stop_request.json という名前のファイルに保存します。

```
{
   "ApplicationName": "test"
}
```

2. 次のリクエストを指定して StopApplication アクションを実行し、アプリケーションを停止します。

```
aws kinesisanalyticsv2 stop-application --cli-input-json file://stop_request.json
```

アプリケーションが停止します。

CloudWatch ログ記録オプションを追加する

を使用して AWS CLI 、Amazon CloudWatch ログストリームをアプリケーションに追加できます。アプリケーションで CloudWatch ログを使用する方法については、「」を参照してくださいthe section called “ログ記録の設定”。

環境プロパティを更新する

このセクションでは、「UpdateApplication」アクションを使用して、アプリケーションコードを再コンパイルせずにアプリケーションの環境プロパティを変更します。この例では、ソースストリームおよびレプリケート先ストリームのリージョンを変更します。

アプリケーションの環境プロパティを更新します

1. 次の JSON コードを update_properties_request.json という名前のファイルに保存します。

```
{"ApplicationName": "test",
   "CurrentApplicationVersionId": 1,
   "ApplicationConfigurationUpdate": {
      "EnvironmentPropertyUpdates": {
         "PropertyGroups": [
            {
               "PropertyGroupId": "ProducerConfigProperties",
               "PropertyMap" : {
                    "flink.stream.initpos" : "LATEST",
                    "aws.region" : "us-west-2",
                    "AggregationEnabled" : "false"
               }
            },
            {
               "PropertyGroupId": "ConsumerConfigProperties",
               "PropertyMap" : {
                    "aws.region" : "us-west-2"
               }
            }
         ]
      }
   }
}
```

2. 前述のリクエストでUpdateApplicationアクションを実行し、環境プロパティを更新します。

```
aws kinesisanalyticsv2 update-application --cli-input-json file://
update_properties_request.json
```

アプリケーションコードの更新

アプリケーションコードを新しいバージョンのコードパッケージで更新する必要がある場合は、UpdateApplication AWS CLI アクションを使用します。

> **Note**
>
> 同じファイル名のアプリケーションコードの新しいバージョンをロードするには、新しいオブジェクトバージョンを指定する必要があります。Amazon S3 オブジェクトバージョンを使用する方法の詳細については、「バージョニングの有効化または無効化」を参照してください。

を使用するには AWS CLI、Amazon S3 バケットから以前のコードパッケージを削除し、新しいバージョンをアップロードして、同じ Amazon S3 バケットとオブジェクト名、および新しいオブジェクトバージョンUpdateApplicationを指定して を呼び出します。アプリケーションは新しいコードパッケージで再起動します。

以下の UpdateApplication アクションのサンプル・リクエストは、アプリケーション・コードを再読み込 み、アプリケーションを再起動します。CurrentApplicationVersionId を現在のアプリケーションバージョンに更新します。ListApplications または DescribeApplication アクションを使用して、現在のアプリケーションバージョンを確認できます。バケット名のサフィックス (「*<username>*」) を、 the section called “2 つの Amazon Kinesis データストリームを作成する” セクションで選択したサフィックスで更新します。

```
{
    "ApplicationName": "test",
    "CurrentApplicationVersionId": 1,
    "ApplicationConfigurationUpdate": {
        "ApplicationCodeConfigurationUpdate": {
            "CodeContentUpdate": {
                "S3ContentLocationUpdate": {
```

```
                    "BucketARNUpdate": "arn:aws:s3:::ka-app-code-username",
                    "FileKeyUpdate": "amazon-msf-java-stream-app-1.0.jar",
                    "ObjectVersionUpdate": "SAMPLEUehYngP87ex1nzYIGYgfhypvDU"
                }
            }
        }
    }
}
```

次のステップ

ステップ 4: リソースをクリーンアップ AWS する

ステップ 4: リソースをクリーンアップ AWS する

このセクションでは、入門チュートリアルで作成した AWS リソースをクリーンアップする手順について説明します。

このトピックには、次のセクションが含まれています。

- Managed Service for Apache Flink アプリケーションを削除する
- Kinesis データストリームを削除する
- Amazon S3 オブジェクトとバケットを削除する
- IAM リソースを削除する
- CloudWatch リソースを削除する
- 次のステップ

Managed Service for Apache Flink アプリケーションを削除する

1. Kinesis コンソール (https://console.aws.amazon.com/kinesis) を開きます。
2. Managed Service for Apache Flink パネルで、 を選択しますMyApplication。
3. アプリケーションのページで[削除]を選択し、削除を確認します。

Kinesis データストリームを削除する

1. https://console.aws.amazon.com/flink で Apache Flink 用 Managed Serviceコンソールを開く

2. Kinesis Data Streams パネルで、 を選択しますExampleInputStream。
3. ExampleInputStream ページで、Kinesis ストリームの削除を選択し、削除を確定します。
4. Kinesis ストリームページで、 を選択しExampleOutputStream、アクション を選択し、削除 を選択し、削除を確定します。

Amazon S3 オブジェクトとバケットを削除する

1. https://console.aws.amazon.com/s3/でAmazon S3 コンソールを開きます。
2. ka-app-code-***<username>*** バケットを選択します。
3. [削除] を選択し、バケット名を入力して削除を確認します。

IAM リソースを削除する

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. ナビゲーションバーで、[ポリシー] を選択します。
3. フィルターコントロールに「kinesis」と入力します。
4. kinesis-analytics-service-MyApplication-us-west-2 ポリシーを選択します。
5. [ポリシーアクション]、[削除] の順に選択します。
6. ナビゲーションバーで [ロール] を選択します。
7. kinesis-analytics-MyApplication-us-west-2 ロールを選択します。
8. [ロールの削除] を選択し、削除を確定します。

CloudWatch リソースを削除する

1. https://console.aws.amazon.com/cloudwatch/ で CloudWatch コンソールを開きます。
2. ナビゲーションバーで [ログ] を選択します。
3. /aws/kinesis-analytics/MyApplication ロググループを選択します。
4. [ロググループの削除]を選択し、削除を確認してください。

次のステップ

ステップ 5: リソースを使用して次のステップを完了する

ステップ 5: リソースを使用して次のステップを完了する

Apache Flink 用 Managed Serviceの基本的なアプリケーションを作成して実行したので、より高度な Apache Flink 用 Managed Serviceソリューションについては、以下のリソースを参照してください。

- Amazon Kinesis 用 AWS ストリーミングデータソリューション： Amazon Kinesis 用 AWS ストリーミングデータソリューションは、ストリーミングデータを簡単にキャプチャ、保存、処理、配信するために必要な AWS サービスを自動的に設定します。このソリューションには、ストリーミングデータのユースケースを解決するための複数のオプションが用意されています。Managed Service for Apache Flink オプションは、 end-to-endシミュレートされたニューヨークのタクシーデータに対して分析オペレーションを実行する実際のアプリケーションを示すストリーミング ETL の例を提供します。このソリューションは、IAM ロールとポリシー、 CloudWatch ダッシュボード、 CloudWatch アラームなど、必要なすべての AWS リソースを設定します。
- AWS Amazon MSK のストリーミングデータソリューション： Amazon MSK の AWS ストリーミングデータソリューションは、データがプロデューサー、ストリーミングストレージ、コンシューマー、および送信先を通過する AWS CloudFormation テンプレートを提供します。
- 「Apache Flink と Apache Kafka によるクリックストリームラボ」:ストリーミングストレージには Apache Kafka 用の Amazon マネージドストリーミングを使用し、ストリーム処理には Apache Flink アプリケーション向けの Apache Flink 用 Managed Serviceを使用する、クリックストリームのユースケースを対象としたエンドツーエンドラボです。
- Amazon Managed Service for Apache Flink Workshop: このワークショップでは、 end-to-end ストリーミングデータをほぼリアルタイムで取り込み、分析、視覚化するためのストリーミングアーキテクチャを構築します。あなたは、ニューヨーク市のあるタクシー会社の業務改善に着手しました。ニューヨーク市のタクシー車両のテレメトリデータをほぼリアルタイムで分析して、車両運用を最適化します。
- 「Managed Service for Apache Flink：例:」この開発者ガイドのこのセクションでは、Apache Flink のマネージドサービスでアプリケーションを作成および操作する例を紹介します。これには、Managed Service for Apache Flink アプリケーションの作成と結果のテストに役立つサンプルコードと step-by-step 手順が含まれています。
- 「Learn Flink: ハンズオントレーニング:」スケーラブルなストリーミング ETL、分析、イベント駆動型アプリケーションの作成を開始するための Apache Flink の公式入門トレーニングです。

Amazon Managed Service for Apache Flink の開始方法 (テーブル API)

このセクションでは、Apache Flink 用 Managed Service と Table API の基本概念を紹介します。アプリケーションの作成とテストに使用できるオプションについて説明します。また、このガイドのチュートリアルを完了し、初めてアプリケーションを作成するのに必要なツールのインストール方法についても説明します。

トピック

- Managed Service for Apache Flink アプリケーションのコンポーネント
- 前提条件
- Managed Service for Apache Flink アプリケーションを作成して実行する
- AWS リソースをクリーンアップする
- 次のステップ

Managed Service for Apache Flink アプリケーションのコンポーネント

Apache Flink アプリケーション用 Managed Service では、Apache Flink ランタイムを使用して入力を処理し、出力を生成する Java/Apache Maven または Scala アプリケーションを使用してデータを処理します。

Apache Flink アプリケーション用 Managed Serviceには、以下のコンポーネントがあります。

- 「ランタイムプロパティ:」「ランタイムプロパティ」を使用すると、アプリケーションコードを再コンパイルせずにアプリケーションを設定できます。
- 「テーブルソース:」アプリケーションはソースを使用してデータを消費します。「ソース」コネクタは、Kinesis データストリーム、Amazon MSK トピックなどからデータを読み取ります。詳細については、「テーブル API ソース」を参照してください。
- 「関数:」アプリケーションは 1 つ以上の関数を使用してデータを処理します。「関数」はデータを変換、拡張、または集約できます。
- 「シンク:」アプリケーションはシンクを使用して外部ソースにデータを生成します。シンクコネクタは、Kinesis データストリーム、Firehose Firehose ストリーム、Amazon MSK トピッ

ク、Amazon S3 バケットなどにデータを書き込みます。詳細については、「テーブル API シンク」を参照してください。

アプリケーションのコードを作成、コンパイル、パッケージ化した後、コードパッケージを Amazon S3 バケットにアップロードします。次に、Apache Flink アプリケーション用 Managed Serviceを作成します。コードパッケージのロケーション、ストリーミングデータソースとしての Amazon MSK トピック、および通常、アプリケーションの処理済みデータを受け取るストリーミングまたはファイルのロケーションを渡します。

前提条件

このチュートリアルを開始する前に、Amazon Managed Service for Apache Flink の開始方法 (DataStream API) の最初の 2 つのステップを完了してください。

- ステップ 1: AWS アカウントを設定し、管理者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

開始するには、 アプリケーションを作成します。 を参照してください。

Managed Service for Apache Flink アプリケーションを作成して実行する

このエクササイズでは、Amazon MSK トピックをソースとして、Amazon S3 バケットをシンクとして、 Apache Flink アプリケーション用 Managed Service を作成します。

このセクションには、以下のステップが含まれています。

- 依存リソースを作成する
- samplerRecords を入力ストリームに書き込む
- Apache Flink ストリーミング Java コードをダウンロードして調べる
- アプリケーションコードをコンパイルする
- Apache Flink ストリーミング Java コードをアップロードする
- Managed Service for Apache Flink アプリケーションを作成して実行する
- 次のステップ

依存リソースを作成する

このエクササイズで Apache Flink 用 Managed Service を作成する前に、以下の依存リソースを作成します。

- Amazon VPC と Amazon MSK クラスターをベースにした 仮想プライベートクラウド (VPC)
- アプリケーションのコードと出力を格納する Amazon S3 バケット (ka-app-code-*<username>*)

VPC と Amazon MSK クラスターを作成する

Apache Flink 用 Managed Service からアクセスするための VPC と Amazon MSK クラスターを作成するには、「Amazon MSK の使用入門」チュートリアルに従ってください。

チュートリアルを完了する際は、以下の点に注意してください。

- クラスターのブートストラップサーバーリストを記録します。以下のコマンドでブートストラップ・サーバのリストを取得し、「*ClusterArn*」を MSK クラスターの Amazon リソースネーム (ARN) に置き換えます。

```
aws kafka get-bootstrap-brokers --region us-west-2 --cluster-arn ClusterArn
{...
    "BootstrapBrokerStringTls": "b-2.awskafkatutorialcluste.t79r6y.c4.kafka.us-
west-2.amazonaws.com:9094,b-1.awskafkatutorialcluste.t79r6y.c4.kafka.us-
west-2.amazonaws.com:9094,b-3.awskafkatutorialcluste.t79r6y.c4.kafka.us-
west-2.amazonaws.com:9094"
}
```

- チュートリアルのステップに従うときは、選択した AWS リージョンをコード、コマンド、コンソールエントリで使用してください。

Amazon S3 バケットを作成する

Amazon S3 バケットは、コンソールを使用して作成できます。このリソースの作成手順については、次のトピックを参照してください。

- Amazon Simple Storage Service ユーザーガイドの「S3 バケットを作成する方法」を参照してください。ログイン名 (**ka-app-code-*<username>*** など) を追加して、Amazon S3 バケットにグローバルに一意の名前を付けます。

その他のリソース

アプリケーションを作成すると、Managed Service for Apache Flink は、以下の Amazon CloudWatch リソースがまだ存在しない場合、それらを作成します。

- /AWS/KinesisAnalytics-java/MyApplicationという名前のロググループ。
- kinesis-analytics-log-stream というログストリーム。

samplerRecords を入力ストリームに書き込む

このセクションでは、Python スクリプトを使用して、アプリケーションが処理するサンプルレコードをAmazon MSK トピックに書き込みます。

1. 「Amazon MSK 入門」チュートリアルの「ステップ 4: クライアントマシンを作成する」で作成したクライアントインスタンスに接続します。
2. Python 3、Pip、および Kafka Python ライブラリーをインストールします。

```
$ sudo yum install python37
$ curl -O https://bootstrap.pypa.io/get-pip.py
$ python3 get-pip.py --user
$ pip install kafka-python
```

3. 次の内容で、stock.py という名前のファイルを作成します。「BROKERS」値を、以前に記録したブートストラップブローカーリストに置き換えます。

```
from kafka import KafkaProducer
import json
import random
from datetime import datetime

# BROKERS = "b-1.stocks.8e6izk.c12.kafka.us-
east-1.amazonaws.com:9092,b-2.stocks.8e6izk.c12.kafka.us-east-1.amazonaws.com:9092"
BROKERS = "localhost:9092"
producer = KafkaProducer(
    bootstrap_servers=BROKERS,
    value_serializer=lambda v: json.dumps(v).encode('utf-8'),
    key_serializer=str.encode,
    retry_backoff_ms=500,
    request_timeout_ms=20000,
    security_protocol='PLAINTEXT')
```

```
def getReferrer():
    data = {}
    now = datetime.now()
    str_now = now.strftime("%Y-%m-%d %H:%M:%S")
    data['event_time'] = str_now
    data['ticker'] = random.choice(['AAPL', 'AMZN', 'MSFT', 'INTC', 'TBV'])
    price = random.random() * 100
    data['price'] = round(price, 2)
    return data


while True:
    data =getReferrer()
    # print(data)
    try:
        future = producer.send("stocktopic", value=data,key=data['ticker'])
        producer.flush()
        record_metadata = future.get(timeout=10)
        print("sent event to Kafka! topic {} partition {} offset
 {}".format(record_metadata.topic, record_metadata.partition,
 record_metadata.offset))
    except Exception as e:
        print(e.with_traceback())
```

4. このチュートリアルの後半では、アプリケーションにデータを送信する stock.py スクリプトを実行します。

```
$ python3 stock.py
```

Apache Flink ストリーミング Java コードをダウンロードして調べる

この例の Java アプリケーションコードは、 から入手できます GitHub。

Java アプリケーションコードのダウンロード

1. 次のコマンドを使用してリモートリポジトリのクローンを作成します。

```
git clone https://github.com/aws-samples/amazon-kinesis-data-analytics-examples.git
```

2. amazon-kinesis-data-analytics-java-examples/GettingStartedTable ディレクトリに移動します。

アプリケーションコードに関して、以下の点に注意してください。

- 「Project Object Model (pom.xml)」ファイルには、Managed Service for Apache Flink 用ライブラリなど、アプリケーションの設定と依存関係に関する情報が含まれています。
- StreamingJob.java ファイルには、アプリケーションの機能を定義する main メソッドが含まれています。
- アプリケーションは、 FlinkKafkaConsumer を使用して Amazon MSK トピックから読み取りを行います。次のスニペットでは、FlinkKafkaConsumer オブジェクトが作成されます。

```
final FlinkKafkaConsumer<StockRecord> consumer = new
 FlinkKafkaConsumer<StockRecord>(kafkaTopic, new KafkaEventDeserializationSchema(),
 kafkaProps);
```

- アプリケーションでは、ソースおよびシンクコネクタを作成し、StreamExecutionEnvironment と TableEnvironment を使用して外部リソースにアクセスします。
- アプリケーションは動的なアプリケーション・プロパティを使用してソース・コネクタとシンク・コネクタを作成するため、コードを再コンパイルすることなくアプリケーションパラメータ (S3 バケットなど) を指定できます。

```
//read the parameters from the Managed Service for Apache Flink environment
Map<String, Properties> applicationProperties =
 KinesisAnalyticsRuntime.getApplicationProperties();
Properties flinkProperties = null;

String kafkaTopic = parameter.get("kafka-topic", "AWSKafkaTutorialTopic");
String brokers = parameter.get("brokers", "");
String s3Path = parameter.get("s3Path", "");

if (applicationProperties != null) {
    flinkProperties = applicationProperties.get("FlinkApplicationProperties");
}

if (flinkProperties != null) {
    kafkaTopic = flinkProperties.get("kafka-topic").toString();
    brokers = flinkProperties.get("brokers").toString();
```

```
    s3Path = flinkProperties.get("s3Path").toString();
}
```

ランタイムプロパティの詳細については、ランタイムプロパティ を参照してください。

> **Note**
>
> アプリケーションを構築するときは、Amazon MSK クラスターと同じリージョンで Apache Flink アプリケーション用 Managed Service を作成して実行することを強くお勧めします。これは、Flink Kafka コネクタがデフォルトで低レイテンシー環境に最適化されているためです。クロスリージョン Kafka クラスターから利用する必要がある場合は、2097152 など、`receive.buffer.byte` の設定値を増やすことを検討してください。
> 詳細については、「カスタム MSK 設定」を参照してください。

アプリケーションコードをコンパイルする

このセクションでは、Apache Maven コンパイラを使用してアプリケーション用の Java コードを作成します。Apache Maven と Java 開発キット (JDK) をインストールする方法については、演習を完了するための前提条件を満たすを参照してください。

アプリケーションコードをコンパイルするには

1. アプリケーションコードを使用するには、コードをコンパイルして JAR ファイルにパッケージ化します。コードのコンパイルとパッケージ化には次の 2 通りの方法があります。

   - コマンドライン Maven ツールを使用します。`pom.xml` ファイルが格納されているディレクトリで次のコマンドを実行して JAR ファイルを作成します。

     ```
     mvn package -Dflink.version=1.19.1
     ```

   - 開発環境を使用します。詳細については、開発環境のドキュメントを参照してください。

     > **Note**
     >
     > 提供されているソースコードは Java 11 のライブラリーに依存しています。

パッケージは JAR ファイルとしてアップロードすることも、圧縮して ZIP ファイルとしてアップロードすることもできします。を使用してアプリケーションを作成する場合は AWS CLI、コードコンテンツタイプ (JAR または ZIP) を指定します。

2. コンパイル中にエラーが発生した場合は、JAVA_HOME 環境変数が正しく設定されていることを確認します。

アプリケーションのコンパイルに成功すると、次のファイルが作成されます。

```
target/aws-kinesis-analytics-java-apps-1.0.jar
```

Apache Flink ストリーミング Java コードをアップロードする

このセクションでは、Amazon S3 バケットを作成し、アプリケーションコードをアップロードします。

アプリケーションコードをアップロードするには

1. Amazon S3 コンソール (https://console.aws.amazon.com/s3/) を開きます。
2. [バケットを作成] を選択します。
3. [Bucket name (バケット名)] フィールドに**ka-app-code-*<username>***と入力します。バケット名にユーザー名などのサフィックスを追加して、グローバルに一意にします。[次へ] をクリックします。
4. 設定オプションのステップでは、設定をそのままにし、[次へ] を選択します。
5. アクセス許可の設定のステップでは、設定をそのままにし、[次へ] を選択します。
6. [バケットを作成] を選択します。
7. Amazon S3 コンソールで、ka-app-code-***<username>*** バケットを選択し、アップロード を選択します。
8. ファイルの選択のステップで、[ファイルを追加] を選択します。前のステップで作成した aws-kinesis-analytics-java-apps-1.0.jar ファイルに移動します。[次へ] をクリックします。
9. オブジェクトの設定を変更する必要はないので、[アップロード] を選択してください。

アプリケーションコードが Amazon S3 バケットに保存され、アプリケーションからアクセスできるようになります。

Managed Service for Apache Flink アプリケーションを作成して実行する

以下の手順を実行し、コンソールを使用してアプリケーションを作成、設定、更新、および実行します。

アプリケーションの作成

1. https://console.aws.amazon.com/flink で Apache Flink 用 Managed Serviceコンソールを開く
2. Managed Service for Apache Flinkのダッシュボードで、「分析アプリケーションの作成」 を選択します。
3. 「Managed Service for Apache Flink-アプリケーションの作成」ページで、次のようにアプリケーションの詳細を入力します。
   - [アプリケーション名] には **`MyApplication`** と入力します。
   - [Description (説明)] に **`My java test app`** と入力します。
   - [ランタイム] には、[Apache Flink] を選択します。
   - バージョンは Apache Flink バージョン 1.19.1 のままにします。
4. [アクセス許可] には、[IAM ロールの作成 / 更新**`kinesis-analytics-MyApplication-us-west-2`**] を選択します。
5. [Create application] を選択します。

> **Note**
>
> コンソールを使用して Apache Flink アプリケーション用 Managed Service を作成する場合は、IAM ロールとポリシーをアプリケーションが自動的に作成するオプションを選択できます。アプリケーションではこのロールとポリシーを使用して、依存リソースにアクセスします。これらの IAM リソースは、次のようにアプリケーション名とリージョンを使用して命名されます。
>
> - ポリシー: `kinesis-analytics-service-`*`MyApplication`*`-`*`us-west-2`*
> - ロール: `kinesisanalytics-`*`MyApplication`*`-`*`us-west-2`*

IAM ポリシーを編集する

Amazon S3 バケットにアクセスする許可を追加するように IAM ポリシーを編集します。

IAM ポリシーを編集して S3 バケット権限を追加するには

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. [ポリシー] を選択します。前のセクションでコンソールによって作成された **kinesis-analytics-service-MyApplication-us-west-2** ポリシーを選択します。
3. [概要] ページで、[ポリシーの編集] を選択します。[JSON] タブを選択します。
4. 次のポリシー例で強調表示されているセクションをポリシーに追加します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
        "Sid": "S3",
        "Effect": "Allow",
        "Action": [
            "s3:Abort*",
            "s3:DeleteObject*",
            "s3:GetObject*",
            "s3:GetBucket*",
            "s3:List*",
            "s3:ListBucket",
            "s3:PutObject"
        ],
        "Resource": [
            "arn:aws:s3:::ka-app-code-<username>",
            "arn:aws:s3:::ka-app-code-<username>/*"
        ]
        },
        {
            "Sid": "DescribeLogGroups",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:*"
            ]
        },
        {
            "Sid": "DescribeLogStreams",
            "Effect": "Allow",
            "Action": [
```

```
                "logs:DescribeLogStreams"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:*"
            ]
        },
        {
            "Sid": "PutLogEvents",
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:kinesis-analytics-log-stream"
            ]
        }
    ]
}
```

アプリケーションを設定する

アプリケーションを設定するには、次の手順に従います。

アプリケーションを構成するには

1. MyApplication ページで、 の設定 を選択します。
2. [Configure application] ページで、[Code location] を次のように指定します。
   - [Amazon S3 バケット] で、**ka-app-code-*<username>***と入力します。
   - [Amazon S3 オブジェクトへのパス] で、**aws-kinesis-analytics-java-apps-1.0.jar**と入力します。
3. [Access to application resources] の [Access permissions] では、[Create / update IAM role**kinesis-analytics-MyApplication-us-west-2**] を選択します。
4. 「プロパティ」で「グループを作成」を選択します。
5. 次のように入力します。

グループ ID	キー	値
FlinkApplicationProperties	**kafka-topic**	**AWSKafkaTutorialTopic**
FlinkApplicationProperties	**brokers**	***Your Amazon MSK cluster's Bootstrap Brokers list***
FlinkApplicationProperties	**s3Path**	**ka-app-code- *<username>***
FlinkApplicationProperties	**security.protocol**	**SSL**
FlinkApplicationProperties	**ssl.truststore.location**	**/usr/lib/jvm/java-11-amazon-corretto/lib/security/cacerts**
FlinkApplicationProperties	**ssl.truststore.password**	**changeit**

6. [Monitoring] の [Monitoring metrics level] が [Application] に設定されていることを確認します。
7. CloudWatch のログ記録では、有効化チェックボックスをオンにします。
8. 「仮想プライベートクラウド (VPC)」セクションで、「Amazon MSK クラスターに基づく VPC 設定」を選択します。を選択しますAWSKafkaTutorialCluster。
9. [更新] を選択します。

> **Note**
>
> Amazon CloudWatch ログ記録を有効にすると、Managed Service for Apache Flink によってロググループとログストリームが作成されます。これらのリソースの名前は次のとおりです。
>
> - ロググループ: `/aws/kinesis-analytics/MyApplication`

- ログストリーム: kinesis-analytics-log-stream

アプリケーションを実行する

アプリケーションを実行するには、次の手順に従います。

アプリケーションを実行するには

1. MyApplication ページで、実行 を選択します。アクションを確認します。
2. アプリケーションが実行されたら、ページを更新します。コンソールには [Application graph] が示されます。
3. Amazon EC2 クライアントから、以前に作成した Python スクリプトを実行して、アプリケーションが処理するレコードを Amazon MSK クラスターに書き込みます。

```
$ python3 stock.py
```

アプリケーションの停止

アプリケーションを停止するには、MyApplicationページで停止 を選択します。アクションを確認します。

次のステップ

AWS リソースをクリーンアップする

AWS リソースをクリーンアップする

このセクションでは、入門 (テーブル API) チュートリアルで作成した AWS リソースをクリーンアップする手順について説明します。

このトピックには、次のセクションが含まれています。

- Managed Service for Apache Flink アプリケーションを削除する
- Amazon MSK クラスターを削除する
- VPC の削除
- Amazon S3 オブジェクトとバケットを削除する

- IAM リソースを削除する
- CloudWatch リソースを削除する
- 次のステップ

Managed Service for Apache Flink アプリケーションを削除する

アプリケーションを削除するには、次の手順に従います。

アプリケーションを削除するには

1. Kinesis コンソール (https://console.aws.amazon.com/kinesis) を開きます。
2. Managed Service for Apache Flink パネルで、 を選択しますMyApplication。
3. アプリケーションページで「Delete」を選択し、削除を確定します。

Amazon MSK クラスターを削除する

Amazon MSK クラスターを削除するには、「Apache Kafka 用 Amazon マネージドストリーミング デベロッパーガイド」の「ステップ 8: Amazon MSK クラスターを削除する」に従ってください。

VPC の削除

Amazon VPC を削除するには、以下の操作を行います。

- Amazon VPC コンソールを開きます。
- [Your VPC] (お使いの VPC) を選択します。
- [アクション] で、[VPC の削除] を選択します。

Amazon S3 オブジェクトとバケットを削除する

S3 オブジェクトとバケットを削除するには、次の手順に従います。

S3 オブジェクトとバケットを削除するには

1. https://console.aws.amazon.com/s3/でAmazon S3 コンソールを開きます。
2. ka-app-code-***<username>*** バケットを選択します。
3. [削除] を選択し、バケット名を入力して削除を確認します。

IAM リソースを削除する

以下の手順を使用して IAM リソースを削除します。

IAM リソースを削除するには

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. ナビゲーションバーで、[ポリシー] を選択します。
3. フィルターコントロールに「kinesis」と入力します。
4. kinesis-analytics-service-MyApplication-us-west-2 ポリシーを選択します。
5. [ポリシーアクション]、[削除] の順に選択します。
6. ナビゲーションバーで [ロール] を選択します。
7. kinesis-analytics-MyApplication-us-west-2 ロールを選択します。
8. [ロールの削除] を選択し、削除を確定します。

CloudWatch リソースを削除する

CloudWatch リソースを削除するには、次の手順に従います。

CloudWatch リソースを削除するには

1. https://console.aws.amazon.com/cloudwatch/ で CloudWatch コンソールを開きます。
2. ナビゲーションバーで [ログ] を選択します。
3. /aws/kinesis-analytics/MyApplication ロググループを選択します。
4. [ロググループの削除]を選択し、削除を確認してください。

次のステップ

次のステップ

次のステップ

テーブル API を使用する Apache Flink アプリケーション用 Managed Service の作成と実行が完了したので、ステップ 5: リソースを使用して次のステップを完了する の Amazon Managed Service for Apache Flink の開始方法 (DataStream API) を参照してください。

Amazon Managed Service for Apache Flink for Python の開始方法

このセクションでは、Python とテーブル API を使用した Apache Flink 向けマネージドサービスの基本概念を紹介します。アプリケーションの作成とテストに使用できるオプションについて説明します。また、このガイドのチュートリアルを完了し、初めてアプリケーションを作成するのに必要なツールのインストール方法についても説明します。

トピック

- Pyflink 入門-Apache 用 Python インタープリター | Amazon Web Services
- Managed Service for Apache Flink アプリケーションのコンポーネント
- 前提条件
- Managed Service for Apache Flink for Python アプリケーションを作成して実行する
- AWS リソースをクリーンアップする

Pyflink 入門-Apache 用 Python インタープリター | Amazon Web Services

開始する前に、以下のビデオをご覧になることをお勧めします。

「Pyflink 入門-Apache 用 Python インタープリター | Amazon Web Services」

Managed Service for Apache Flink アプリケーションのコンポーネント

データを処理するために、Apache Flink アプリケーション用 Managed Serviceは、Apache Flink ランタイムを使用して入力を処理し、出力を生成する Python アプリケーションを使用します。

Apache Flink アプリケーション用 Managed Serviceには、以下のコンポーネントがあります。

- 「ランタイムプロパティ:」「ランタイムプロパティ」を使用すると、アプリケーションコードを再コンパイルせずにアプリケーションを設定できます。

- 「テーブルソース:」アプリケーションはソースを使用してデータを消費します。「ソース」コネクタは、Kinesis データストリーム、Amazon MSK トピックなどからデータを読み取ります。詳細については、「テーブル API ソース」を参照してください。
- 「関数:」アプリケーションは 1 つ以上の関数を使用してデータを処理します。「関数」はデータを変換、拡張、または集約できます。
- 「シンク:」アプリケーションはシンクを使用して外部ソースにデータを生成します。シンクコネクタは、Kinesis データストリーム、Firehose Firehose ストリーム、Amazon MSK トピック、Amazon S3 バケットなどにデータを書き込みます。詳細については、「テーブル API シンク」を参照してください。

アプリケーションコードを作成してパッケージ化したら、Amazon S3 バケットにコードパッケージをアップロードします。次に、Apache Flink アプリケーション用 Managed Serviceを作成します。コードパッケージの場所、ストリーミングデータソース、通常はアプリケーションの処理済みデータを受け取るストリーミングまたはファイルの場所を渡します。

前提条件

このチュートリアルを開始する前に、Amazon Managed Service for Apache Flink の開始方法 (DataStream API) の最初の 2 つのステップを完了してください。

- ステップ 1: AWS アカウントを設定し、管理者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

開始するには、 アプリケーションを作成します。 を参照してください。

Managed Service for Apache Flink for Python アプリケーションを作成して実行する

このエクササイズでは、ソースとシンクとして、Python アプリケーション用の Apache Flink を作成します。

このセクションには、以下のステップが含まれています。

- 依存リソースを作成する
- サンプルレコードを入力ストリームに書き込む
- Apache Flink ストリーミング Python コードを作成して調べる

- Python アプリへのサードパーティの依存関係の追加
- Apache Flink ストリーミング Python コードをアップロードする
- Managed Service for Apache Flink アプリケーションを作成して実行する
- 次のステップ

依存リソースを作成する

このエクササイズで Apache Flink 用 Managed Service を作成する前に、以下の依存リソースを作成します。

- 入力用と出力用の 2 つの Kinesis ストリーム。
- アプリケーションのコードと出力を格納する Amazon S3 バケット (ka-app-code-*<username>*)

2 つの Kinesis ストリームを作成する

この演習で Apache Flink アプリケーションのマネージドサービスを作成する前に、2 つの Kinesis データストリーム (ExampleInputStream と ExampleOutputStream) を作成する必要があります。アプリケーションでは、これらのストリームを使用してアプリケーションの送信元と送信先のストリームを選択します。

これらのストリームは Amazon Kinesis コンソールまたは次の AWS CLI コマンドを使用して作成できます。コンソールでの操作方法については、「Amazon Kinesis Data Streams デベロッパーガイド」 の 「データストリームの作成および更新」 を参照してください。

データストリームを作成するには (AWS CLI)

1. 最初のストリーム (ExampleInputStream) を作成するには、次の Amazon Kinesis create-stream AWS CLI コマンドを使用します。

```
$ aws kinesis create-stream \
--stream-name ExampleInputStream \
--shard-count 1 \
--region us-west-2 \
--profile adminuser
```

2. 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 ExampleOutputStream \
--shard-count 1 \
--region us-west-2 \
--profile adminuser
```

Amazon S3 バケットを作成する

Amazon S3 バケットは、コンソールを使用して作成できます。このリソースの作成手順については、次のトピックを参照してください。

- Amazon Simple Storage Service ユーザーガイドの「S3 バケットを作成する方法」を参照してください。ログイン名 (**ka-app-code-*<username>*** など) を追加して、Amazon S3 バケットにグローバルに一意の名前を付けます。

その他のリソース

アプリケーションを作成すると、Managed Service for Apache Flink は、以下の Amazon CloudWatch リソースがまだ存在しない場合、それらを作成します。

- `/AWS/KinesisAnalytics-java/MyApplication`という名前のロググループ。
- `kinesis-analytics-log-stream` というログストリーム。

サンプルレコードを入力ストリームに書き込む

このセクションでは、Python スクリプトを使用して、アプリケーションが処理するサンプルレコードをストリームに書き込みます。

> **Note**
>
> このセクションでは AWS SDK for Python (Boto) が必要です。

> **Note**
>
> このセクションの Python スクリプトでは、 AWS CLIを使用しています。アカウント認証情報とデフォルトのリージョンを使用する AWS CLI ように を設定する必要があります。を設定するには AWS CLI、次のように入力します。
>
> ```
> aws configure
> ```

1. 次の内容で、stock.py という名前のファイルを作成します。

```
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', region_name='us-west-2'))
```

2. stock.py スクリプトを実行します。

```
$ python stock.py
```

チュートリアルの残りの部分を完了する間、スクリプトを実行し続けてください。

Apache Flink ストリーミング Python コードを作成して調べる

この例の Python アプリケーションコードは、 から入手できます GitHub。アプリケーションコードをダウンロードするには、次の操作を行います。

1. Git クライアントをまだインストールしていない場合は、インストールします。詳細については、「Git のインストール」をご参照ください。
2. 次のコマンドを使用してリモートリポジトリのクローンを作成します。

```
git clone https://github.com/aws-samples/amazon-kinesis-data-analytics-examples.git
```

3. amazon-kinesis-data-analytics-java-examples/python/GettingStarted ディレクトリに移動します。

アプリケーションコードはgetting_started.pyファイルに含まれています。アプリケーションコードに関して、以下の点に注意してください。

- アプリケーションは Kinesis テーブルソースを使用して、ソースストリームから読み取りを行います。次のスニペットは、 create_table 関数を呼び出して Kinesis テーブルソースを作成します。

```
table_env.execute_sql(
        create_table(output_table_name, output_stream, output_region)
```

このcreate_table関数は SQL コマンドを使用して、ストリーミングソースに裏付けられたテーブルを作成します。

```
def create_table(table_name, stream_name, region, stream_initpos = None):
    init_pos = "\n'scan.stream.initpos' = '{0}',".format(stream_initpos) if
 stream_initpos is not None else ''

    return """ CREATE TABLE {0} (
                ticker VARCHAR(6),
```

```
                price DOUBLE,
                event_time TIMESTAMP(3),
                WATERMARK FOR event_time AS event_time - INTERVAL '5' SECOND
              )
              PARTITIONED BY (ticker)
              WITH (
                'connector' = 'kinesis',
                'stream' = '{1}',
                'aws.region' = '{2}',{3}
                'format' = 'json',
                'json.timestamp-format.standard' = 'ISO-8601'
              ) """.format(table_name, stream_name, region, init_pos)
}
```

- アプリケーションは 2 つのテーブルを作成し、一方のテーブルの内容をもう一方のテーブルに書き込みます。

```
    # 2. Creates a source table from a Kinesis Data Stream
    table_env.execute_sql(
        create_table(input_table_name, input_stream, input_region)
    )

    # 3. Creates a sink table writing to a Kinesis Data Stream
    table_env.execute_sql(
        create_table(output_table_name, output_stream, output_region, stream_initpos)
    )

    # 4. Inserts the source table data into the sink table
    table_result = table_env.execute_sql("INSERT INTO {0} SELECT * FROM {1}"
                         .format(output_table_name, input_table_name))
```

- アプリケーションは、flink- sql-connector-kinesis_2.12/1.15.2 ファイルから Flink コネクタを使用します。

Python アプリへのサードパーティの依存関係の追加

サードパーティの Python パッケージ (「boto3」など) を使用する場合は、その推移的な依存関係と、これらの依存関係をターゲットにするのに必要なプロパティを追加する必要があります。大まかに言うと、 PyPi 依存関係については、Python 環境フォルダ内にあるファイルとsite-packagesフォルダをコピーして、次のようなディレクトリ構造を作成できます。

```
PythonPackages
#   README.md
#   python-packages.py
#
####my_deps
       ####boto3
       #   #   session.py
       #   #   utils.py
       #   #   ...
       #
       ####botocore
       #   #   args.py
       #   #   auth.py
       #    ...
       ####mynonpypimodule
       #   #   mymodulefile1.py
       #   #   mymodulefile2.py
            ...
####lib
#    # flink-sql-connector-kinesis-4.2.0-1.18.jar
#    # ...
...
```

boto3 をサードパーティの依存関係として追加するには:

1. 必要な依存関係を持つスタンドアロンの Python 環境 (conda など) をローカルマシン上に作成します。
2. その環境の `site_packages` フォルダーにあるパッケージの最初のリストを書き留めておきます。
3. `pip-install` アプリに必要なすべての依存関係。
4. 上記のステップ 3 の後に `site_packages` フォルダーに追加されたパッケージを書き留めておきます。これらは、パッケージ (`my_deps` フォルダーの下) に含める必要のあるフォルダーで、上記のように整理されています。これにより、ステップ 2 と 3 の間のパッケージの「差分」をキャプチャして、アプリケーションに適したパッケージ依存関係を特定できます。
5. `jarfiles` プロパティについて後述するように、`kinesis.analytics.flink.run.options` プロパティグループの `pyFiles` プロパティの引数として `my_deps/` を与えます。Flink では「add_python_file」関数を使用して Python の依存関係を指定することもできますが、指定する必要があるのはどちらか一方のみで、両方を指定する必要はないことに注意してください。

> **Note**
>
> フォルダー `my_deps` に名前を付ける必要はありません。重要なのは、 `pyFiles` または `add_python_file` を使用して依存関係を登録することです。例としては、PyFlink 内での「boto3 の使用方法」をご覧ください。

Apache Flink ストリーミング Python コードをアップロードする

このセクションでは、Amazon S3 バケットを作成し、アプリケーションコードをアップロードします。

コンソールを使用してアプリケーションコードをアップロードするには:

1. 任意の圧縮アプリケーションを使用して、 `getting-started.py`および Flink SQL コネクタファイルを圧縮します。アーカイブ `myapp.zip` に名をつけます。アーカイブに外部フォルダーを含める場合は、設定ファイル `GettingStarted/getting-started.py` 内のコードパスにこれを含める必要があります。
2. Amazon S3 コンソール (https://console.aws.amazon.com/s3/) を開きます。
3. [バケットを作成] を選択します。
4. [Bucket name (バケット名)] フィールドに**`ka-app-code-<username>`**と入力します。バケット名にユーザー名などのサフィックスを追加して、グローバルに一意にします。[次へ] をクリックします。
5. 設定オプションのステップでは、設定をそのままにし、[次へ] を選択します。
6. アクセス許可の設定のステップでは、設定をそのままにし、[次へ] を選択します。
7. [バケットを作成] を選択します。
8. Amazon S3 コンソールで、ka-app-code-***`<username>`*** バケットを選択し、アップロード を選択します。
9. ファイルの選択のステップで、[ファイルを追加] を選択します。前のステップで作成した `myapp.zip` ファイルに移動します。[次へ] をクリックします。
10. オブジェクトの設定を変更する必要はないので、[アップロード] を選択してください。

アプリケーションコードをアップロードするには AWS CLI:

> **Note**
>
> Finder (macOS) または Windows エクスプローラー (Windows) の圧縮機能を使用して myapp.zip アーカイブを作成しないでください。この値は無効になることがあります。

1. 任意の圧縮アプリケーションを使用して、 streaming-file-sink.pyおよび Flink SQL コネクタファイルを圧縮します。

   > **Note**
   >
   > Finder (macOS) または Windows エクスプローラー (Windows) の圧縮機能を使用して「myapp.zip」アーカイブを作成しないでください。この値は無効になることがあります。

2. 任意の圧縮アプリケーションを使用して、 getting-started.pyおよび Flink SQL コネクタファイルを圧縮します。アーカイブ myapp.zip に名をつけます。アーカイブに外部フォルダーを含める場合は、設定ファイル GettingStarted/getting-started.py 内のコードパスにこれを含める必要があります。
3. 次のコマンドを実行します。

   ```
   $ aws s3 --region aws region cp myapp.zip s3://ka-app-code-<username>
   ```

アプリケーションコードが Amazon S3 バケットに保存され、アプリケーションからアクセスできるようになります。

Managed Service for Apache Flink アプリケーションを作成して実行する

以下の手順を実行し、コンソールを使用してアプリケーションを作成、設定、更新、および実行します。

アプリケーションの作成

1. https://console.aws.amazon.com/flink で Apache Flink 用 Managed Serviceコンソールを開く
2. Managed Service for Apache Flinkのダッシュボードで、「分析アプリケーションの作成」 を選択します。

3. 「Managed Service for Apache Flink-アプリケーションの作成」ページで、次のようにアプリケーションの詳細を入力します。

   - [アプリケーション名] には **MyApplication** と入力します。
   - [Description (説明)] に **My java test app** と入力します。
   - [ランタイム] には、[Apache Flink] を選択します。
   - バージョンは Apache Flink バージョン 1.18 のままにしておきます。

4. [アクセス許可] には、[IAM ロールの作成 / 更新**kinesis-analytics-MyApplication-us-west-2**] を選択します。

5. [Create application] を選択します。

> **Note**
>
> コンソールを使用して Apache Flink アプリケーション用 Managed Service を作成する場合は、IAM ロールとポリシーをアプリケーションが自動的に作成するオプションを選択できます。アプリケーションではこのロールとポリシーを使用して、依存リソースにアクセスします。これらの IAM リソースは、次のようにアプリケーション名とリージョンを使用して命名されます。
>
> - ポリシー: `kinesis-analytics-service-`*`MyApplication`*`-`*`us-west-2`*
> - ロール: `kinesisanalytics-`*`MyApplication`*`-`*`us-west-2`*

アプリケーションを設定する

アプリケーションを設定するには、次の手順に従います。

アプリケーションを構成するには

1. MyApplication ページで、 の設定 を選択します。
2. [Configure application] ページで、[Code location] を次のように指定します。

   - [Amazon S3 バケット] で、**ka-app-code-*<username>***と入力します。
   - [Amazon S3 オブジェクトへのパス] で、**myapp.zip**と入力します。

3. [Access to application resources] の [Access permissions] では、[Create / update IAM role**kinesis-analytics-MyApplication-us-west-2**] を選択します。

4. [プロパティ] で [グループの追加]を選択します。
5. 次のように入力します。

グループ ID	キー	値
consumer.config.0	**input.stream.name**	**ExampleInputStream**
consumer.config.0	**aws.region**	**us-west-2**
consumer.config.0	**scan.stream.initpos**	**LATEST**

[保存] を選択します。

6. プロパティ で、グループの追加をもう一度選択します。
7. 次のように入力します。

グループ ID	キー	値
producer.config.0	**output.stream.name**	**ExampleOutputStream**
producer.config.0	**aws.region**	**us-west-2**
producer.config.0	**shard.count**	**1**

8. プロパティ で、グループの追加をもう一度選択します。[グループ ID] に、「**flink.sql.connector.kinesis.options**」と入力します。この特別なプロパティグループは、アプリケーションにコードリソースの場所を指定します。詳細については、「コードファイルの指定」を参照してください。
9. 次のように入力します。

グループ ID	キー	値
kinesis.analytics.flink.run.options	**python**	**getting-started.py**
kinesis.analytics.flink.run.options	**jarfile**	**flink-sql-connector-kinesis-4.2.0-1.18.jar**

10. [Monitoring] の [Monitoring metrics level] が [Application] に設定されていることを確認します。
11. CloudWatch のログ記録では、有効化チェックボックスを選択します。
12. [更新] を選択します。

> **Note**
>
> Amazon CloudWatch ログ記録を有効にすると、Managed Service for Apache Flink によってロググループとログストリームが作成されます。これらのリソースの名前は次のとおりです。
>
> - ロググループ: `/aws/kinesis-analytics/MyApplication`
> - ログストリーム: `kinesis-analytics-log-stream`

IAM ポリシーを編集する

Amazon S3 バケットにアクセスする許可を追加するように IAM ポリシーを編集します。

IAM ポリシーを編集して S3 バケット権限を追加するには

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. [ポリシー] を選択します。前のセクションでコンソールによって作成された **kinesis-analytics-service-MyApplication-us-west-2** ポリシーを選択します。
3. [概要] ページで、[ポリシーの編集] を選択します。[JSON] タブを選択します。
4. 次のポリシー例で強調表示されているセクションをポリシーに追加します。サンプルのアカウント ID (*012345678901*) を自分のアカウント ID に置き換えます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ReadCode",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": [
```

```
                "arn:aws:s3:::ka-app-code-username/myapp.zip"
            ]
        },
        {
            "Sid": "DescribeLogGroups",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:*"
            ]
        },
        {
            "Sid": "DescribeLogStreams",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogStreams"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:*"
            ]
        },
        {
            "Sid": "PutLogEvents",
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:kinesis-analytics-log-stream"
            ]
        },
        {
            "Sid": "ReadInputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
        },
        {
            "Sid": "WriteOutputStream",
```

```
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleOutputStream"
        }
    ]
}
```

アプリケーションを実行する

Flink ジョブグラフは、アプリケーションを実行し、Apache Flink ダッシュボードを開き、目的の Flink ジョブを選択すると表示できます。

アプリケーションの停止

アプリケーションを停止するには、MyApplicationページで停止 を選択します。アクションを確認します。

次のステップ

AWS リソースをクリーンアップする

AWS リソースをクリーンアップする

このセクションでは、入門 (Python) チュートリアルで作成した AWS リソースをクリーンアップする手順について説明します。

このトピックには、次のセクションが含まれています。

- Managed Service for Apache Flink アプリケーションを削除する
- Kinesis データストリームを削除する
- Amazon S3 オブジェクトとバケットを削除する
- IAM リソースを削除する
- CloudWatch リソースを削除する

Managed Service for Apache Flink アプリケーションを削除する

アプリケーションを削除するには、次の手順に従います。

アプリケーションを削除するには

1. Kinesis コンソール (https://console.aws.amazon.com/kinesis) を開きます。
2. Managed Service for Apache Flink パネルで、 を選択しますMyApplication。
3. アプリケーションページで「Delete」を選択し、削除を確定します。

Kinesis データストリームを削除する

1. https://console.aws.amazon.com/flink で Apache Flink 用 Managed Serviceコンソールを開く
2. Kinesis Data Streams パネルで、 を選択しますExampleInputStream。
3. ExampleInputStream ページで、「Kinesis ストリームの削除」を選択し、削除を確定します。
4. Kinesis ストリームページで、 を選択しExampleOutputStream、アクション を選択し、削除 を選択し、削除を確定します。

Amazon S3 オブジェクトとバケットを削除する

S3 オブジェクトとバケットを削除するには、次の手順に従います。

S3 オブジェクトとバケットを削除するには

1. https://console.aws.amazon.com/s3/でAmazon S3 コンソールを開きます。
2. ka-app-code-***<username>*** バケットを選択します。
3. [削除] を選択し、バケット名を入力して削除を確認します。

IAM リソースを削除する

以下の手順を使用して IAM リソースを削除します。

IAM リソースを削除するには

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. ナビゲーションバーで、[ポリシー] を選択します。
3. フィルターコントロールに「kinesis」と入力します。
4. kinesis-analytics-service-MyApplication-us-west-2 ポリシーを選択します。
5. [ポリシーアクション]、[削除] の順に選択します。

6. ナビゲーションバーで [ロール] を選択します。
7. kinesis-analytics-MyApplication-us-west-2 ロールを選択します。
8. [ロールの削除] を選択し、削除を確定します。

CloudWatch リソースを削除する

CloudWatch リソースを削除するには、次の手順に従います。

CloudWatch リソースを削除するには

1. https://console.aws.amazon.com/cloudwatch/ で CloudWatch コンソールを開きます。
2. ナビゲーションバーで [ログ] を選択します。
3. /aws/kinesis-analytics/MyApplication ロググループを選択します。
4. [ロググループの削除]を選択し、削除を確認してください。

開始方法 (スカラー)

> **Note**
>
> バージョン 1.15 以降、Flink は Scala 無料です。アプリケーションが Scala の任意のバージョンから Java API を使用できるようになっています。Flink は引き続きいくつかの主要コンポーネントで Scala を内部的に使用しますが、Scala をユーザーコードクラスローダーに公開しません。そのため、Scala の依存関係を JAR アーカイブに追加する必要があります。Flink 1.15 での Scala の変更についての詳しい情報は、Scala Free in One Fifteenを参照してください。

このエクササイズでは、ソースとシンクとしてKinesisストリームを使用して、Scala向けのApache Flinkアプリケーション 用 Managed Service を作成します。

このトピックには、次のセクションが含まれています。

- 依存リソースを作成する
- サンプルレコードを入力ストリームに書き込む
- アプリケーションコードをダウンロードして調べる
- アプリケーション・コードをコンパイルしてアップロードするには
- アプリケーションの作成と実行 (コンソール)
- アプリケーションの作成と実行 (CLI)
- AWS リソースをクリーンアップする

依存リソースを作成する

この練習用の Managed Service for Apache Flink を作成する前に、以下の依存リソースを作成します。

- 入力用と出力用の 2 つの Kinesis ストリーム。
- アプリケーションのコードを保存するためのAmazon S3バケット (`ka-app-code-<username>`)

Kinesis ストリームと Amazon S3 バケットは、コンソールを使用して作成できます。これらのリソースの作成手順については、次の各トピックを参照してください。

- 「Amazon Kinesis Data Streamsデベロッパーガイド」の「データストリームの作成および更新」データストリーム**ExampleInputStream**と**ExampleOutputStream**に名前を付けます。

  データストリームを作成するには (AWS CLI)

  - 最初のストリーム (ExampleInputStream) を作成するには、次の Amazon Kinesis create-stream AWS CLI コマンドを使用します。

    ```
    aws kinesis create-stream \
        --stream-name ExampleInputStream \
        --shard-count 1 \
        --region us-west-2 \
        --profile adminuser
    ```

  - 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 ExampleOutputStream \
        --shard-count 1 \
        --region us-west-2 \
        --profile adminuser
    ```

- Amazon Simple Storage Service ユーザーガイドの「S3 バケットを作成する方法」を参照してください。ログイン名 (**ka-app-code-*<username>*** など) を追加して、Amazon S3 バケットにグローバルに一意の名前を付けます。

その他のリソース

アプリケーションを作成すると、Managed Service for Apache Flink は、以下の Amazon CloudWatch リソースがまだ存在しない場合、それらを作成します。

- /AWS/KinesisAnalytics-java/MyApplicationという名前のロググループ。
- kinesis-analytics-log-stream というログストリーム

サンプルレコードを入力ストリームに書き込む

このセクションでは、Python スクリプトを使用して、アプリケーションが処理するサンプルレコードをストリームに書き込みます。

> **Note**
>
> このセクションでは AWS SDK for Python (Boto) が必要です。

> **Note**
>
> このセクションの Python スクリプトでは、 AWS CLIを使用しています。アカウント認証情報とデフォルトのリージョンを使用する AWS CLI ように を設定する必要があります。を設定するには AWS CLI、次のように入力します。
>
> ```
> aws configure
> ```

1. 次の内容で、stock.py という名前のファイルを作成します。

```
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', region_name='us-west-2'))
```

2. stock.py スクリプトを実行します。

```
$ python stock.py
```

チュートリアルの残りの部分を完了する間、スクリプトを実行し続けてください。

アプリケーションコードをダウンロードして調べる

この例の Python アプリケーションコードは、 から入手できます GitHub。アプリケーションコードをダウンロードするには、次の操作を行います。

1. Git クライアントをまだインストールしていない場合は、インストールします。詳細については、「Git のインストール」をご参照ください。
2. 次のコマンドを使用してリモートリポジトリのクローンを作成します。

```
git clone https://github.com/aws-samples/amazon-kinesis-data-analytics-examples.git
```

3. amazon-kinesis-data-analytics-java-examples/scala/GettingStarted ディレクトリに移動します。

アプリケーションコードに関して、以下の点に注意してください。

- build.sbtファイルには、Managed Service for Apache Flink ライブラリなど、アプリケーションの設定と依存関係に関する情報が含まれています。
- この BasicStreamingJob.scala ファイルには、アプリケーションの機能を定義するメインメソッドが含まれています。
- アプリケーションは Kinesis ソースを使用して、ソースストリームから読み取りを行います。次のスニペットでは、Kinesis ソースが作成されます。

```
private def createSource: FlinkKinesisConsumer[String] = {
  val applicationProperties = KinesisAnalyticsRuntime.getApplicationProperties
  val inputProperties = applicationProperties.get("ConsumerConfigProperties")

  new FlinkKinesisConsumer[String](inputProperties.getProperty(streamNameKey,
 defaultInputStreamName),
```

```
    new SimpleStringSchema, inputProperties)
}
```

また、アプリケーションは Kinesis シンクを使用して結果ストリームに書き込みます。次のスニペットでは、Kinesis シンクが作成されます。

```
private def createSink: KinesisStreamsSink[String] = {
  val applicationProperties = KinesisAnalyticsRuntime.getApplicationProperties
  val outputProperties = applicationProperties.get("ProducerConfigProperties")

  KinesisStreamsSink.builder[String]
    .setKinesisClientProperties(outputProperties)
    .setSerializationSchema(new SimpleStringSchema)
    .setStreamName(outputProperties.getProperty(streamNameKey,
 defaultOutputStreamName))
    .setPartitionKeyGenerator((element: String) => String.valueOf(element.hashCode))
    .build
}
```

- アプリケーションは、 StreamExecutionEnvironment オブジェクトを使用して外部リソースにアクセスするためのソースコネクタとシンクコネクタを作成します。
- アプリケーションは、動的アプリケーションプロパティを使用してソースコネクタとシンクコネクタを作成します。アプリケーションのプロパティを読み取ってコネクタを設定します。ランタイムプロパティの詳細については、ランタイムプロパティを参照してください。

アプリケーション・コードをコンパイルしてアップロードするには

このセクションでは、アプリケーションコードをコンパイルし、依存リソースを作成する セクションで作成したAmazon S3バケットにアップロードします。

アプリケーションコードのコンパイル

このセクションでは、「SBT」ビルド・ツールを使用してアプリケーションの Scala コードをビルドします。SBTをインストールするには、Install sbt with cs setupを参照してください。また、Java 開発キット (JDK) をインストールする必要があります。演習を完了するための前提条件 を参照してください。

1. アプリケーションコードを使用するには、コードをコンパイルして JAR ファイルにパッケージ化します。SBT を使用してコードをコンパイルしてパッケージ化できます。

```
sbt assembly
```

2. アプリケーションのコンパイルに成功すると、次のファイルが作成されます。

```
target/scala-3.2.0/getting-started-scala-1.0.jar
```

Apache Flink Streaming Scala Code のアップロード

このセクションでは、Amazon S3 バケットを作成し、アプリケーションコードをアップロードします。

1. https://console.aws.amazon.com/s3/でAmazon S3 コンソールを開きます。
2. [バケットを作成] を選択します。
3. [Bucket name (バケット名)] フィールドにka-app-code-<username>と入力します。バケット名にユーザー名などのサフィックスを追加して、グローバルに一意にします。[次へ] をクリックします。
4. 設定オプションのステップでは、設定をそのままにし、[次へ] を選択します。
5. アクセス許可の設定のステップでは、設定をそのままにし、[次へ] を選択します。
6. [バケットを作成] を選択します。
7. ka-app-code-<username>バケットを選択し、アップロード を選択します。
8. ファイルの選択のステップで、[ファイルを追加] を選択します。前のステップで作成した getting-started-scala-1.0.jar ファイルに移動します。
9. オブジェクトの設定を変更する必要はないので、[アップロード] を選択してください。

アプリケーションコードが Amazon S3 バケットに保存され、アプリケーションからアクセスできるようになります。

アプリケーションの作成と実行 (コンソール)

以下の手順を実行し、コンソールを使用してアプリケーションを作成、設定、更新、および実行します。

アプリケーションの作成

1. https://console.aws.amazon.com/flink で Apache Flink 用 Managed Serviceコンソールを開く

2. Managed Service for Apache Flinkのダッシュボードで、「分析アプリケーションの作成」 を選択します。
3. 「Managed Service for Apache Flink-アプリケーションの作成」ページで、次のようにアプリケーションの詳細を入力します。
   - [アプリケーション名] には **MyApplication** と入力します。
   - [Description (説明)] に **My scala test app** と入力します。
   - [ランタイム] には、[Apache Flink] を選択します。
   - バージョンは Apache Flink バージョン 1.19.1 のままにします。
4. [アクセス許可] には、[IAM ロールの作成 / 更新**kinesis-analytics-MyApplication-us-west-2**] を選択します。
5. [Create application] を選択します。

> **Note**
>
> コンソールを使用して Apache Flink アプリケーション用 Managed Service を作成する場合は、IAM ロールとポリシーをアプリケーションが自動的に作成するオプションを選択できます。アプリケーションではこのロールとポリシーを使用して、依存リソースにアクセスします。これらの IAM リソースは、次のようにアプリケーション名とリージョンを使用して命名されます。
>
> - ポリシー: `kinesis-analytics-service-MyApplication-us-west-2`
> - ロール: `kinesisanalytics-MyApplication-us-west-2`

アプリケーションを設定する

アプリケーションを設定するには、次の手順に従います。

アプリケーションを構成するには

1. MyApplication ページで、 の設定 を選択します。
2. [Configure application] ページで、[Code location] を次のように指定します。
   - [Amazon S3 バケット] で、**ka-app-code-*<username>***と入力します。

- [Amazon S3 オブジェクトへのパス] で、**getting-started-scala-1.0.jar.**と入力します。

3. [Access to application resources] の [Access permissions] では、[Create / update IAM role**kinesis-analytics-MyApplication-us-west-2**] を選択します。
4. [プロパティ] で [グループの追加]を選択します。
5. 次のように入力します。

グループ ID	キー	値
ConsumerConfigProperties	**input.stream.name**	**ExampleInputStream**
ConsumerConfigProperties	**aws.region**	**us-west-2**
ConsumerConfigProperties	**flink.stream.initpos**	**LATEST**

[保存] を選択します。

6. プロパティ で、グループの追加をもう一度選択します。
7. 次のように入力します。

グループ ID	キー	値
ProducerConfigProperties	**output.stream.name**	**ExampleOutputStream**
ProducerConfigProperties	**aws.region**	**us-west-2**

8. [Monitoring] の [Monitoring metrics level] が [Application] に設定されていることを確認します。
9. CloudWatch のログ記録では、有効化チェックボックスを選択します。
10. [更新] を選択します。

> **Note**
>
> Amazon CloudWatch ログ記録を有効にすると、Managed Service for Apache Flink によってロググループとログストリームが作成されます。これらのリソースの名前は次のとおりです。
>
> - ロググループ: `/aws/kinesis-analytics/MyApplication`
> - ログストリーム: `kinesis-analytics-log-stream`

IAM ポリシーを編集する

Amazon S3 バケットにアクセスする許可を追加するように IAM ポリシーを編集します。

IAM ポリシーを編集して S3 バケット権限を追加するには

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. [ポリシー] を選択します。前のセクションでコンソールによって作成された **kinesis-analytics-service-MyApplication-us-west-2** ポリシーを選択します。
3. [概要] ページで、[ポリシーの編集] を選択します。[JSON] タブを選択します。
4. 次のポリシー例で強調表示されているセクションをポリシーに追加します。サンプルのアカウント ID (*012345678901*) を自分のアカウント ID に置き換えます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ReadCode",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": [
                "arn:aws:s3:::ka-app-code-username/getting-started-scala-1.0.jar"
            ]
        },
        {
            "Sid": "DescribeLogGroups",
            "Effect": "Allow",
```

```
            "Action": [
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:*"
            ]
        },
        {
            "Sid": "DescribeLogStreams",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogStreams"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:*"
            ]
        },
        {
            "Sid": "PutLogEvents",
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:kinesis-analytics-log-stream"
            ]
        },
        {
            "Sid": "ReadInputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
        },
        {
            "Sid": "WriteOutputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleOutputStream"
        }
    ]
```

```
}
```

アプリケーションを実行する

Flink ジョブグラフは、アプリケーションを実行し、Apache Flink ダッシュボードを開き、目的の Flink ジョブを選択すると表示できます。

アプリケーションの停止

アプリケーションを停止するには、MyApplicationページで停止 を選択します。アクションを確認します。

アプリケーションの作成と実行 (CLI)

このセクションでは、 を使用して Managed Service for Apache Flink アプリケーション AWS Command Line Interface を作成して実行します。kinesisanalyticsv2 AWS CLI コマンドを使用して、Managed Service for Apache Flink アプリケーションを作成して操作します。

許可ポリシーを作成する

> **Note**
>
> アプリケーションのアクセス権限ポリシーとロールを作成する必要があります。これらの IAM リソースを作成しない場合、アプリケーションはそのデータストリームやログストリームにアクセスできません。

まず、2 つのステートメントを含むアクセス許可ポリシーを作成します。1 つは、ソースストリームの読み取りアクションに対するアクセス許可を付与し、もう 1 つはシンクストリームの書き込みアクションに対するアクセス許可を付与します。次に、IAM ロール (次のセクションで作成) にポリシーをアタッチします。そのため、 Managed Service for Apache Flinkがこのロールを引き受けると、ソースストリームからの読み取りとシンクストリームへの書き込みを行うために必要なアクセス許可がサービスに付与されます。

次のコードを使用して `AKReadSourceStreamWriteSinkStream` アクセス許可ポリシーを作成します。**`username`** を Amazon S3 バケットの作成に使用したユーザー名に置き換え、アプリケーショ

ンコードを保存します。Amazon リソースネーム (ARN) **(012345678901)**のアカウント ID を自分のアカウント ID に置き換えます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ReadCode",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": [
                "arn:aws:s3:::ka-app-code-username/getting-started-scala-1.0.jar"
            ]
        },
        {
            "Sid": "DescribeLogGroups",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:*"
            ]
        },
        {
            "Sid": "DescribeLogStreams",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogStreams"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-analytics/
MyApplication:log-stream:*"
            ]
        },
        {
            "Sid": "PutLogEvents",
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents"
```

```
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-analytics/
MyApplication:log-stream:kinesis-analytics-log-stream"
            ]
        },
        {
            "Sid": "ReadInputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
        },
        {
            "Sid": "WriteOutputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleOutputStream"
        }
    ]
}
```

アクセス許可ポリシーを作成する step-by-step 手順については、IAM ユーザーガイドの「チュートリアル: 最初のカスタマー管理ポリシーの作成とアタッチ」を参照してください。

IAM ポリシーを作成する

このセクションでは、Managed Service for Apache Flink アプリケーションがソースストリームを読み取り、シンクストリームに書き込むために想定できる IAM ロールを作成します。

Apache Flink 用 Managed Service は、許可なしにはストリームにアクセスできません。IAM ロールを介してこれらの許可を付与します。各 IAM ロールには、2 つのポリシーがアタッチされます。信頼ポリシーは、ロールを引き受けるための許可を Managed Service for Apache Flink 付与し、許可ポリシーは、ロールを引き受けた後に Managed Service for Apache Flink が実行できる事柄を決定します。

前のセクションで作成したアクセス許可ポリシーをこのロールにアタッチします。

IAM ロールを作成するには

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。

2. ナビゲーションペインで [ロール] を選択し、続いて [ロールを作成] を選択します。
3. [信頼されるエンティティの種類を選択] で、[AWS のサービス] を選択します。
4. [このロールを使用するサービスを選択] で、[Kinesis Analytics] を選択します。
5. [ユースケースの選択]で、[Managed Service for Apache Flink]を選択します。
6. [Next: Permissions] (次のステップ: 許可) を選択します。
7. [アクセス権限ポリシーをアタッチする] ページで、[Next: Review] (次: 確認) を選択します。ロールを作成した後に、アクセス許可ポリシーをアタッチします。
8. [Create role (ロールの作成)] ページで、ロールの名前に **`MF-stream-rw-role`** を入力します。[ロールの作成] を選択します。

   これで、`MF-stream-rw-role` と呼ばれる新しい IAM ロールが作成されます。次に、ロールの信頼ポリシーとアクセス許可ポリシーを更新します。
9. アクセス許可ポリシーをロールにアタッチします。

   > **Note**
   >
   > この演習では、Managed Service for Apache Flink が、Kinesis データストリーム (ソース) からのデータの読み取りと、別の Kinesis データストリームへの出力の書き込みの両方を実行するためにこのロールを引き受けます。前のステップである「許可ポリシーの作成」で作成したロールを添付します。

   a. [概要] ページで、[アクセス許可] タブを選択します。
   b. [Attach Policies (ポリシーのアタッチ)] を選択します。
   c. 検索ボックスに**`AKReadSourceStreamWriteSinkStream`**(前のセクションで作成したポリシー) と入力します。
   d. `AKReadSourceStreamWriteSinkStream`ポリシーを選択し、[ポリシーを添付]を選択します。

これで、アプリケーションがリソースにアクセスするために使用するサービスの実行ロールが作成されました。新しいロールの ARN を書き留めておきます。

ロールを作成する step-by-step 手順については、IAM ユーザーガイドの「IAM ロールの作成 (コンソール)」を参照してください。

アプリケーションの作成

次の JSON コードを create_request.json という名前のファイルに保存します。サンプルロールの ARN を、前に作成したロールの ARN に置き換えます。バケット ARN のサフィックス (ユーザー名) を、前のセクションで選択したサフィックスに置き換えます。サービス実行ロールのサンプルのアカウント ID (012345678901) を、自分のアカウント ID に置き換えます。

```
{
    "ApplicationName": "getting_started",
    "ApplicationDescription": "Scala getting started application",
    "RuntimeEnvironment": "FLINK-1_19",
    "ServiceExecutionRole": "arn:aws:iam::012345678901:role/MF-stream-rw-role",
    "ApplicationConfiguration": {
        "ApplicationCodeConfiguration": {
            "CodeContent": {
                "S3ContentLocation": {
                    "BucketARN": "arn:aws:s3:::ka-app-code-username",
                    "FileKey": "getting-started-scala-1.0.jar"
                }
            },
            "CodeContentType": "ZIPFILE"
        },
        "EnvironmentProperties":  {
         "PropertyGroups": [
            {
               "PropertyGroupId": "ConsumerConfigProperties",
               "PropertyMap" : {
                    "aws.region" : "us-west-2",
                    "stream.name" : "ExampleInputStream",
                    "flink.stream.initpos" : "LATEST"
               }
            },
            {
               "PropertyGroupId": "ProducerConfigProperties",
               "PropertyMap" : {
                    "aws.region" : "us-west-2",
                    "stream.name" : "ExampleOutputStream"
               }
            }
         ]
      }
    },
    "CloudWatchLoggingOptions": [
```

```
      {
         "LogStreamARN": "arn:aws:logs:us-west-2:012345678901:log-
group:MyApplication:log-stream:kinesis-analytics-log-stream"
      }
   ]
}
```

次のリクエストCreateApplicationで を実行してアプリケーションを作成します。

```
aws kinesisanalyticsv2 create-application --cli-input-json file://create_request.json
```

これでアプリケーションが作成されました。次のステップでは、アプリケーションを起動します。

アプリケーションを起動する

このセクションでは、 StartApplicationアクションを使用してアプリケーションを起動します。

アプリケーションを起動するには

1. 次の JSON コードを start_request.json という名前のファイルに保存します。

```
{
    "ApplicationName": "getting_started",
    "RunConfiguration": {
        "ApplicationRestoreConfiguration": {
         "ApplicationRestoreType": "RESTORE_FROM_LATEST_SNAPSHOT"
         }
    }
}
```

2. 前述のリクエストを指定して StartApplication アクションを実行し、アプリケーションを起動します。

```
aws kinesisanalyticsv2 start-application --cli-input-json file://start_request.json
```

アプリケーションが実行されます。Amazon CloudWatch コンソールで Managed Service for Apache Flink メトリクスをチェックして、アプリケーションが動作していることを確認できます。

アプリケーションの停止

このセクションでは、 StopApplicationアクションを使用してアプリケーションを停止します。

アプリケーションを停止するには

1. 次の JSON コードを stop_request.json という名前のファイルに保存します。

```
{
   "ApplicationName": "s3_sink"
}
```

2. 前述のリクエストを指定して StopApplication アクションを実行し、アプリケーションを起動します。

```
aws kinesisanalyticsv2 stop-application --cli-input-json file://stop_request.json
```

アプリケーションが停止します。

CloudWatch ログ記録オプションを追加する

を使用して AWS CLI 、Amazon CloudWatch ログストリームをアプリケーションに追加できます。アプリケーションで CloudWatch ログを使用する方法については、「アプリケーションログ記録のセットアップ」を参照してください。

環境プロパティを更新する

このセクションでは、 UpdateApplicationアクションを使用して、アプリケーションコードを再コンパイルせずにアプリケーションの環境プロパティを変更します。この例では、ソースストリームおよびレプリケート先ストリームのリージョンを変更します。

アプリケーションの環境プロパティを更新します

1. 次の JSON コードを update_properties_request.json という名前のファイルに保存します。

```
{
      "ApplicationName": "getting_started",
       "CurrentApplicationVersionId": 1,
       "ApplicationConfigurationUpdate": {
        "EnvironmentPropertyUpdates": {
           "PropertyGroups": [
            {
               "PropertyGroupId": "ConsumerConfigProperties",
```

```
                "PropertyMap" : {
                     "aws.region" : "us-west-2",
                     "stream.name" : "ExampleInputStream",
                     "flink.stream.initpos" : "LATEST"
                }
             },
             {
                "PropertyGroupId": "ProducerConfigProperties",
                "PropertyMap" : {
                     "aws.region" : "us-west-2",
                     "stream.name" : "ExampleOutputStream"
                }
             }
            ]
         }
    }
```

2. 前述のリクエストでUpdateApplicationアクションを実行し、環境プロパティを更新します。

```
aws kinesisanalyticsv2 update-application --cli-input-json file://
update_properties_request.json
```

アプリケーションコードの更新

アプリケーションコードを新しいバージョンのコードパッケージで更新する必要がある場合は、 CLI UpdateApplication アクションを使用します。

> **Note**
>
> 同じファイル名のアプリケーションコードの新しいバージョンをロードするには、新しいオブジェクトバージョンを指定する必要があります。Amazon S3 オブジェクトバージョンを使用する方法の詳細については、「バージョニングの有効化または無効化」を参照してください。

を使用するには AWS CLI、Amazon S3 バケットから以前のコードパッケージを削除し、新しいバージョンをアップロードして、同じ Amazon S3 バケットとオブジェクト名、および新しいオブジェクトバージョンUpdateApplicationを指定して を呼び出します。アプリケーションは新しいコードパッケージで再起動します。

以下の UpdateApplication アクションのサンプル・リクエストは、アプリケーション・コードを再読み込 み、アプリケーションを再起動します。CurrentApplicationVersionId を現在のアプリケーションバージョンに更新します。ListApplications または DescribeApplication アクションを使用して、現在のアプリケーションバージョンを確認できます。バケット名のサフィックス (「<username>」) を、 依存リソースを作成する セクションで選択したサフィックスで更新します。

```
{{
    "ApplicationName": "getting_started",
    "CurrentApplicationVersionId": 1,
    "ApplicationConfigurationUpdate": {
        "ApplicationCodeConfigurationUpdate": {
            "CodeContentUpdate": {
                "S3ContentLocationUpdate": {
                    "BucketARNUpdate": "arn:aws:s3:::ka-app-code-<username>",
                    "FileKeyUpdate": "getting-started-scala-1.0.jar",
                    "ObjectVersionUpdate": "SAMPLEUehYngP87ex1nzYIGYgfhypvDU"
                }
            }
        }
    }
}
```

AWS リソースをクリーンアップする

このセクションでは、「タンブリングウィンドウ」チュートリアルで作成した AWS リソースをクリーンアップする手順について説明します。

このトピックには、次のセクションが含まれています。

- Managed Service for Apache Flink アプリケーションを削除する
- Kinesis データストリームを削除する
- Amazon S3 オブジェクトとバケットを削除する
- IAM リソースを削除する
- CloudWatch リソースを削除する

Managed Service for Apache Flink アプリケーションを削除する

1. https://console.aws.amazon.com/flink で Apache Flink 用 Managed Serviceコンソールを開く

2. Managed Service for Apache Flink パネルで、 を選択しますMyApplication。
3. アプリケーションのページで[削除]を選択し、削除を確認します。

Kinesis データストリームを削除する

1. 「https://console.aws.amazon.com/kinesis」で Kinesis コンソールを開きます。
2. Kinesis Data Streams パネルで、 を選択しますExampleInputStream。
3. ExampleInputStream ページで、Kinesis ストリームの削除を選択し、削除を確定します。
4. Kinesis ストリームページで、 を選択しExampleOutputStream、アクション を選択し、削除 を選択し、削除を確定します。

Amazon S3 オブジェクトとバケットを削除する

1. https://console.aws.amazon.com/s3/でAmazon S3 コンソールを開きます。
2. ka-app-code-***<username>*** バケットを選択します。
3. [削除] を選択し、バケット名を入力して削除を確認します。

IAM リソースを削除する

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. ナビゲーションバーで、[ポリシー] を選択します。
3. フィルターコントロールに「kinesis」と入力します。
4. kinesis-analytics-service-MyApplication-us-west-2 ポリシーを選択します。
5. [ポリシーアクション]、[削除] の順に選択します。
6. ナビゲーションバーで [ロール] を選択します。
7. kinesis-analytics-MyApplication-us-west-2 ロールを選択します。
8. [ロールの削除] を選択し、削除を確定します。

CloudWatch リソースを削除する

1. https://console.aws.amazon.com/cloudwatch/ で CloudWatch コンソールを開きます。
2. ナビゲーションバーで [ログ] を選択します。

3. /aws/kinesis-analytics/MyApplication ロググループを選択します。
4. [ロググループの削除]を選択し、削除を確認してください。

Apache Beam で Apache Flink アプリケーション用 Managed Serviceを作成する

> **Note**
>
> Apache Beam は Apache Flink バージョン 1.19 ではサポートされていません。2024 年 6 月 27 日現在、Flink 1.18 と互換性のある Apache Flink Runner はありません。詳細については、Apache Beam ドキュメントの「Flink バージョンの互換性」を参照してください。 >

「Apache Beam」フレームワークを Apache Flink アプリケーション用 Managed Serviceで使用して、ストリーミングデータを処理できます。Apache Beam を使用する Apache Flink アプリケーション用 Managed Serviceでは、「Apache Flink ランナー」を使用して Beam パイプラインを実行します。

Apache Flink アプリケーション用 Managed Serviceで Apache Beam を使用する方法に関するチュートリアルについては、 Apache Flink CloudFormation 用マネージドサービスとの併用 を参照してください。

このトピックには、次のセクションが含まれています。

- Managed Service for Apache Flink で Apache Beam を使用する
- ビーム機能
- Apache Beam を使用してアプリケーションを作成する

Managed Service for Apache Flink で Apache Beam を使用する

Apache Flink 用 Managed Serviceで Apache Flink ランナーを使用する際には、次の点に注意してください。

- Apache Beam メトリクスは Apache Flink 用 Managed Serviceコンソールでは表示できません。
- 「Apache Beam は Apache Flink バージョン 1.8 以降を使用する Apache Flink アプリケーション用 Managed Serviceでのみサポートされています。」「Apache Beam は、Apache Flink バージョン 1.6 を使用する Apache Flink アプリケーション用 Managed Serviceではサポートされていません。」

ビーム機能

Apache Flink のマネージドサービスは、Apache Flink ランナーと同じ Apache Beam 能力をサポートしています。Apache Flink ランナーでサポートされている機能については、「ビーム互換性マトリックス」を参照してください。

Apache Flink 用 Managed Serviceで Apache Flink アプリケーションをテストして、アプリケーションに必要なすべての機能がサポートされていることを確認することをお勧めします。

Apache Beam を使用してアプリケーションを作成する

この課題では、「Apache Beam」を使用してデータを変換する Apache Flink アプリケーション用 Managed Serviceを作成します。Apache Beam はストリーミングデータを処理するためのプログラミングモデルです。Apache Flink 用 Managed Service での Apache Beam の使用ついては、 Apache Beams の使用 を参照してください。

> **Note**
>
> この演習に必要な前提条件を設定するには、まず開始方法 (DataStream API)演習を完了してください。

このトピックには、次のセクションが含まれています。

- 依存リソースを作成する
- サンプルレコードを入力ストリームに書き込む
- アプリケーションコードをダウンロードして調べる
- アプリケーションコードをコンパイルする
- Apache Flink ストリーミング Java コードをアップロードする
- Managed Service for Apache Flink アプリケーションを作成して実行する
- AWS リソースをクリーンアップする
- 次のステップ

依存リソースを作成する

この練習用の Managed Service for Apache Flink を作成する前に、以下の依存リソースを作成します。

- 2 つの Kinesis Data Streams (`ExampleInputStream`と`ExampleOutputStream`)
- アプリケーションのコードを保存するためのAmazon S3バケット (`ka-app-code-<username>`)

Kinesis ストリームと Amazon S3 バケットは、コンソールを使用して作成できます。これらのリソースの作成手順については、次の各トピックを参照してください。

- 「Amazon Kinesis Data Streamsデベロッパーガイド」の「データストリームの作成および更新」データストリーム**`ExampleInputStream`**と**`ExampleOutputStream`**に名前を付けます。
- Amazon Simple Storage Service ユーザーガイドの「S3 バケットを作成する方法」を参照してください。ログイン名 (**`ka-app-code-<username>`** など) を追加して、Amazon S3 バケットにグローバルに一意の名前を付けます。

サンプルレコードを入力ストリームに書き込む

このセクションでは、Python スクリプトを使用して、アプリケーションが処理するランダムな文字列をストリームに書き込みます。

> **Note**
>
> このセクションでは AWS SDK for Python (Boto) が必要です。

1. 次の内容で、`ping.py` という名前のファイルを作成します。

```
import json
import boto3
import random

kinesis = boto3.client('kinesis')

while True:
        data = random.choice(['ping', 'telnet', 'ftp', 'tracert', 'netstat'])
        print(data)
        kinesis.put_record(
                StreamName="ExampleInputStream",
                Data=data,
                PartitionKey="partitionkey")
```

2. `ping.py` スクリプトを実行します。

```
$ python ping.py
```

チュートリアルの残りの部分を完了する間、スクリプトを実行し続けてください。

アプリケーションコードをダウンロードして調べる

この例の Java アプリケーションコードは、 から入手できます GitHub。アプリケーションコードをダウンロードするには、次の操作を行います。

1. Git クライアントをまだインストールしていない場合は、インストールします。詳細については、「Git のインストール」をご参照ください。
2. 次のコマンドを使用してリモートリポジトリのクローンを作成します。

```
git clone https://github.com/aws-samples/amazon-kinesis-data-analytics-examples.git
```

3. amazon-kinesis-data-analytics-java-examples/Beam ディレクトリに移動します。

アプリケーションコードはBasicBeamStreamingJob.javaファイルに含まれています。アプリケーションコードに関して、以下の点に注意してください。

- アプリケーションは Apache Beam を使用してParDo、 というカスタム変換関数を呼び出して受信レコードを処理しますPingPongFn。

  PingPongFn 関数を呼び出すコードは次のとおりです。

```
.apply("Pong transform",
    ParDo.of(new PingPongFn())
```

- Apache Beam を使用する Apache Flink アプリケーション用 Managed Serviceには、以下のコンポーネントが必要です。これらのコンポーネントとバージョンを pom.xml に含めないと、アプリケーションは環境の依存関係から誤ったバージョンをロードし、バージョンが一致しないため、実行時にアプリケーションがクラッシュします。

```
<jackson.version>2.10.2</jackson.version>
...
<dependency>
    <groupId>com.fasterxml.jackson.module</groupId>
    <artifactId>jackson-module-jaxb-annotations</artifactId>
```

```
    <version>2.10.2</version>
</dependency>
```

- PingPongFn 変換関数は、入力データが ping でない限り、入力データを出力ストリームに渡します。「ping」である場合は、文字列「pong\n」を出力ストリームに出力します。

  変換関数のコードは以下のとおりです。

```
    private static class PingPongFn extends DoFn<KinesisRecord, byte[]> {
    private static final Logger LOG = LoggerFactory.getLogger(PingPongFn.class);

    @ProcessElement
    public void processElement(ProcessContext c) {
        String content = new String(c.element().getDataAsBytes(),
 StandardCharsets.UTF_8);
        if (content.trim().equalsIgnoreCase("ping")) {
            LOG.info("Ponged!");
            c.output("pong\n".getBytes(StandardCharsets.UTF_8));
        } else {
            LOG.info("No action for: " + content);
            c.output(c.element().getDataAsBytes());
        }
    }
}
```

アプリケーションコードをコンパイルする

アプリケーションをコンパイルするには、次の操作を行います。

1. Java と Maven がまだインストールされていない場合は、インストールします。詳細については、開始方法 (DataStream API)チュートリアルの「前提条件」を参照してください。
2. 次のコマンドを使用して、アプリケーションをコンパイルします。

```
mvn package -Dflink.version=1.15.2 -Dflink.version.minor=1.8
```

> **Note**
>
> 提供されているソースコードは Java 11 のライブラリーに依存しています。

アプリケーションをコンパイルすると、アプリケーション JAR ファイル (`target/basic-beam-app-1.0.jar`) が作成されます。

Apache Flink ストリーミング Java コードをアップロードする

このセクションでは、依存リソースを作成する のセクションで作成した Amazon S3 バケットにアプリケーションコードをアップロードします。

1. Amazon S3 コンソールで、ka-app-code-***<username>*** バケットを選択し、アップロード を選択します。
2. ファイルの選択のステップで、[ファイルを追加] を選択します。前のステップで作成した `basic-beam-app-1.0.jar` ファイルに移動します。
3. オブジェクトの設定を変更する必要はないので、[アップロード] を選択してください。

アプリケーションコードが Amazon S3 バケットに保存され、アプリケーションからアクセスできるようになります。

Managed Service for Apache Flink アプリケーションを作成して実行する

以下の手順を実行し、コンソールを使用してアプリケーションを作成、設定、更新、および実行します。

アプリケーションの作成

1. https://console.aws.amazon.com/flink で Apache Flink 用 Managed Serviceコンソールを開く
2. Managed Service for Apache Flinkのダッシュボードで、「分析アプリケーションの作成」 を選択します。
3. 「Managed Service for Apache Flink-アプリケーションの作成」ページで、次のようにアプリケーションの詳細を入力します。
   - [アプリケーション名] には **`MyApplication`** と入力します。
   - [ランタイム] には、[Apache Flink] を選択します。

     > **Note**
     >
     > Apache Beam は現在、Apache Flink バージョン 1.19 以降と互換性がありません。

   - バージョンプルダウンから Apache Flink バージョン 1.15 を選択します。

4. [アクセス許可] には、[IAM ロールの作成 / 更新**kinesis-analytics-MyApplication-us-west-2**] を選択します。
5. [Create application] を選択します。

> **Note**
>
> コンソールを使用して Apache Flink アプリケーション用 Managed Service を作成する場合は、IAM ロールとポリシーをアプリケーションが自動的に作成するオプションを選択できます。アプリケーションではこのロールとポリシーを使用して、依存リソースにアクセスします。これらの IAM リソースは、次のようにアプリケーション名とリージョンを使用して命名されます。
>
> - ポリシー: `kinesis-analytics-service-MyApplication-us-west-2`
> - ロール: `kinesis-analytics-MyApplication-us-west-2`

IAM ポリシーを編集する

IAM ポリシーを編集し、Kinesis Data Streamsにアクセスするための許可を追加します。

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. [ポリシー] を選択します。前のセクションでコンソールによって作成された **kinesis-analytics-service-MyApplication-us-west-2** ポリシーを選択します。
3. [概要] ページで、[ポリシーの編集] を選択します。[JSON] タブを選択します。
4. 次のポリシー例で強調表示されているセクションをポリシーに追加します。サンプルのアカウント ID (*012345678901*) を自分のアカウント ID に置き換えます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ReadCode",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "logs:DescribeLogGroups",
                "s3:GetObjectVersion"
            ],
```

```
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:*",
                "arn:aws:s3:::ka-app-code-<username>/basic-beam-app-1.0.jar"
            ]
        },
        {
            "Sid": "DescribeLogStreams",
            "Effect": "Allow",
            "Action": "logs:DescribeLogStreams",
            "Resource": "arn:aws:logs:us-west-2:012345678901:log-group:/aws/
kinesis-analytics/MyApplication:log-stream:*"
        },
        {
            "Sid": "PutLogEvents",
            "Effect": "Allow",
            "Action": "logs:PutLogEvents",
            "Resource": "arn:aws:logs:us-west-2:012345678901:log-group:/aws/
kinesis-analytics/MyApplication:log-stream:kinesis-analytics-log-stream"
        },
        {
            "Sid": "ListCloudwatchLogGroups",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:*"
            ]
        },
        {
            "Sid": "ReadInputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
        },
        {
            "Sid": "WriteOutputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleOutputStream"
        }
    ]
```

```
}
```

アプリケーションを設定する

1. MyApplication ページで、 の設定 を選択します。
2. [Configure application] ページで、[Code location] を次のように指定します。
   - [Amazon S3 バケット] で、**ka-app-code-*<username>***と入力します。
   - [Amazon S3 オブジェクトへのパス] で、**basic-beam-app-1.0.jar**と入力します。
3. [Access to application resources] の [Access permissions] では、[Create / update IAM role**kinesis-analytics-MyApplication-us-west-2**] を選択します。
4. 次のように入力します。

グループ ID	キー	値
BeamApplicationProperties	**InputStreamName**	**ExampleInputStream**
BeamApplicationProperties	**OutputStreamName**	**ExampleOutputStream**
BeamApplicationProperties	**AwsRegion**	**us-west-2**

5. [Monitoring] の [Monitoring metrics level] が [Application] に設定されていることを確認します。
6. CloudWatch のログ記録では、有効化チェックボックスをオンにします。
7. [更新] を選択します。

> **Note**
>
> CloudWatch ログ記録を有効にすると、Managed Service for Apache Flink によってロググループとログストリームが作成されます。これらのリソースの名前は次のとおりです。
>
> - ロググループ: `/aws/kinesis-analytics/MyApplication`
> - ログストリーム: `kinesis-analytics-log-stream`

このログストリームはアプリケーションのモニターに使用されます。このログストリームは、アプリケーションの結果の送信に使用されたログストリームとは異なります。

アプリケーションを実行する

Flink ジョブグラフは、アプリケーションを実行し、Apache Flink ダッシュボードを開き、目的の Flink ジョブを選択すると表示できます。

CloudWatch コンソールで Managed Service for Apache Flink メトリクスをチェックして、アプリケーションが動作していることを確認できます。

AWS リソースをクリーンアップする

このセクションでは、「タンブリングウィンドウ」チュートリアルで作成した AWS リソースをクリーンアップする手順について説明します。

このトピックには、次のセクションが含まれています。

- Managed Service for Apache Flink アプリケーションを削除する
- Kinesis データストリームを削除する
- Amazon S3 オブジェクトとバケットを削除する
- IAM リソースを削除する
- CloudWatch リソースを削除する

Managed Service for Apache Flink アプリケーションを削除する

1. https://console.aws.amazon.com/flink で Apache Flink 用 Managed Serviceコンソールを開く
2. Managed Service for Apache Flink パネルで、 を選択しますMyApplication。
3. アプリケーションのページで[削除]を選択し、削除を確認します。

Kinesis データストリームを削除する

1. 「https://console.aws.amazon.com/kinesis」で Kinesis コンソールを開きます。
2. Kinesis Data Streams パネルで、 を選択しますExampleInputStream。

3. ExampleInputStream ページで、Kinesis ストリームの削除を選択し、削除を確定します。
4. Kinesis ストリームページで、 を選択しExampleOutputStream、アクション を選択し、削除 を選択し、削除を確定します。

Amazon S3 オブジェクトとバケットを削除する

1. https://console.aws.amazon.com/s3/でAmazon S3 コンソールを開きます。
2. ka-app-code-***<username>*** バケットを選択します。
3. [削除] を選択し、バケット名を入力して削除を確認します。

IAM リソースを削除する

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. ナビゲーションバーで、[ポリシー] を選択します。
3. フィルターコントロールに「kinesis」と入力します。
4. kinesis-analytics-service-MyApplication-us-west-2 ポリシーを選択します。
5. [ポリシーアクション]、[削除] の順に選択します。
6. ナビゲーションバーで [ロール] を選択します。
7. kinesis-analytics-MyApplication-us-west-2 ロールを選択します。
8. [ロールの削除] を選択し、削除を確定します。

CloudWatch リソースを削除する

1. https://console.aws.amazon.com/cloudwatch/ で CloudWatch コンソールを開きます。
2. ナビゲーションバーで [ログ] を選択します。
3. /aws/kinesis-analytics/MyApplication ロググループを選択します。
4. [ロググループの削除]を選択し、削除を確認してください。

次のステップ

Apache Beam を使用してデータを変換する基本的なApache Flink アプリケーション用 Managed Service を作成し、実行しました。次に、より高度なApache Flink ソリューション用 Managed Service の例として次のアプリケーションをご覧ください。

- 「Apache Flink Streaming Workshop 用 Managed Service上のビーム」:このワークショップでは、バッチとストリーミングを1つの Apache Beam パイプラインに統合したエンド・ツー・エンドの例について説明します。

トレーニングワークショップ、ラボ、ソリューション実装

次の end-to-end 例は、Apache Flink 用 Managed Service の高度なソリューションを示しています。

トピック

- Amazon Managed Service for Apache Flink を使用したアプリケーションのデプロイ、運用、スケーリング
- Managed Service for Apache Flink にデプロイする前に、Apache Flink アプリケーションをローカルで開発する
- Managed Service for Apache Flink Studio でイベント検出を使用する
- Amazon Kinesis の AWS ストリーミングデータソリューションを使用する
- Apache Flink と Apache Kafka で Clickstream ラボを使用する方法
- Application Auto Scaling を使用してカスタムスケーリングを設定する
- サンプル Amazon CloudWatch ダッシュボードを表示する
- Amazon MSK の AWS ストリーミングデータソリューションに テンプレートを使用する
- で Apache Flink ソリューション用 Managed Service の詳細をご覧ください。 GitHub

Amazon Managed Service for Apache Flink を使用したアプリケーションのデプロイ、運用、スケーリング

このワークショップでは、Java での Apache Flink アプリケーションの開発、IDE での実行とデバッグの方法、Amazon Managed Service for Apache Flink でパッケージ化、デプロイ、実行する方法について説明します。また、アプリケーションのスケーリング、モニタリング、トラブルシューティングの方法についても説明します。

このワークショップは、Amazon Managed Service for Apache Flink ワークショップ でご覧いただけます。

Managed Service for Apache Flink にデプロイする前に、Apache Flink アプリケーションをローカルで開発する

このワークショップでは、Apache Flink 用 Managed Serviceへのデプロイを長期的な目標として、Apache Flink アプリケーションのローカルでの開発を開始するための基本知識を紹介します。

ソリューションは以下にあります。「Apache Flink を使った口ーカル開発入門ガイド」

Managed Service for Apache Flink Studio でイベント検出を使用する

このワークショップでは、Apache Flink Studio 用 Managed Service によるイベント検出と、Apache Flink アプリケーション用 Managed Service としてのデプロイについて説明します。

ソリューションは以下にあります。「Apache Flink 用 Managed Service によるイベント検出」

Amazon Kinesis の AWS ストリーミングデータソリューションを使用する

Amazon Kinesis 用 AWS ストリーミングデータソリューションは、ストリーミングデータを簡単にキャプチャ、保存、処理、配信するために必要な AWS サービスを自動的に設定します。このソリューションには、ストリーミングデータのユースケースを解決するための複数のオプションが用意されています。Managed Service for Apache Flink オプションは、シミュレートされたニューヨークのタクシーデータに対して分析オペレーションを実行する実際のアプリケーションを示す end-to-end ストリーミング ETL の例を提供します。

各ソリューションには、以下のコンポーネントとが含まれています。

- 完全な例をデプロイする AWS CloudFormation パッケージ。
- アプリケーションメトリクスを表示するための CloudWatch ダッシュボード。
- CloudWatch 最も関連性の高いアプリケーションメトリクスに関する アラーム。
- すべての必要な IAM ロールとポリシー

ソリューションはこちらでご覧いただけます。Amazon Kinesis 向けストリーミングデータソリューション

Apache Flink と Apache Kafka で Clickstream ラボを使用する方法

ストリーミングストレージには Apache Kafka 用 Amazon Managed Streaming、ストリーム処理には Apache Flink アプリケーション向けの Apache Flink 用 Managed Service を使用した、クリックストリームのユースケースを対象とするエンドツーエンドラボです。

ソリューションは以下にあります。「Clickstream Lab」

Application Auto Scaling を使用してカスタムスケーリングを設定する

Application Auto Scaling を使用して、ユーザーが Managed Service for Apache Flink アプリケーションを自動的にスケーリングするのに役立つサンプル。これにより、ユーザーはカスタムスケーリングポリシーとカスタムスケーリング属性を設定できます。

ソリューションは以下にあります。

- Managed Service for Apache Flink アプリの自動スケーリング
- スケジュールに基づくスケーリング

カスタムスケーリングを実行できる方法の詳細については、「Amazon Managed Service for Apache Flink のメトリクスベースおよびスケジュールされたスケーリングを有効にする」を参照してください。

サンプル Amazon CloudWatch ダッシュボードを表示する

Managed Service for Apache Flink アプリケーションをモニタリングするためのサンプル CloudWatch ダッシュボード。サンプルダッシュボードには、ダッシュボードの機能のデモンストレーションに役立つ「デモアプリケーション」も含まれています。

このソリューションはこちらでご覧いただけます。「Apache Flink Metrics Dashboard 用 Managed Service」

Amazon MSK の AWS ストリーミングデータソリューションに テンプレートを使用する

Amazon MSK 用 AWS ストリーミングデータソリューションは、データがプロデューサー、ストリーミングストレージ、コンシューマー、および送信先を通過する AWS CloudFormation テンプレートを提供します。

ソリューションはこちらでご覧いただけます。AWS Amazon MSK 向けストリーミングデータソリューション

で Apache Flink ソリューション用 Managed Service の詳細をご覧ください。 GitHub

以下の end-to-end 例は、Apache Flink ソリューション用 Managed Service の高度な例であり、 で利用できます GitHub。

- 「Amazon Managed Service for Apache Flink — ベンチマークユーティリティ」
- 「スナップショットマネージャー — Apache Flink 向けのAmazon Managed Service for Apache Flink」
- 「Apache Flink を備えたStreaming ETLとAmazon Managed Service for Apache Flink」
- 「顧客からのフィードバックに基づくリアルタイムのセンチメント分析」

ユーティリティ

以下のユーティリティを使用すると、Apache Flink 用 Managed Service をより使いやすくすることができます。

トピック

- スナップショットマネージャ
- ベンチマーキング

スナップショットマネージャ

Flink Applicationsでは、よりシームレスな障害回復を可能にするために、定期的にセーブポイント/スナップショットをトリガーするのが最善の方法です。スナップショットマネージャーは、このタスクを自動化し、以下の利点を提供します。

- 実行中の Apache Flink アプリケーション用 Apache Flink 用 Managed Service の新しいスナップショットを取得
- アプリケーションスナップショットの数を取得
- その数が必要なスナップショット数を超えているかどうかをチェック
- 必要数以上の古いスナップショットを削除

例については、「スナップショットマネージャ」を参照してください。

ベンチマーキング

Apache Flink Flink Benchmarking Utility 用 Managed Service は、Apache Flink 用 Managed Service のキャパシティプランニング、統合テスト、Apache Flink アプリケーションのベンチマークに役立ちます。

例については、「ベンチマーキング」を参照してください。

Managed Service for Apache Flink：例

このセクションでは、 Managed Service for Apache Flink 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、Managed Service for Apache Flink アプリケーションの作成と結果のテストに役立つサンプルコードと step-by-step 手順が含まれています。

例に進む前に、以下に目を通しておくことをお勧めします。

- 仕組み
- 開始方法 (DataStream API)

> **Note**
>
> これらの例は、米国西部 (オレゴン) リージョン (us-west-2) を使用していると仮定しています。別のリージョンを使用している場合は、アプリケーションコード、コマンド、IAM ロールを適切に更新してください。

トピック

- Java の例
- Python の例
- Scala の例

Java の例

次の例は、Java で記述されたアプリケーションを作成する方法を示しています。

> **Note**
>
> ほとんどの例は、ローカル、開発マシン、選択した IDE、Amazon Managed Service for Apache Flink の両方で実行されるように設計されています。アプリケーションパラメータを渡すために使用できるメカニズムと、両方の環境でアプリケーションを変更せずに実行するために依存関係を正しく設定する方法を示しています。

DataStream API の開始方法

この例では、DataStreamAPI を使用して Kinesis データストリームから読み取り、別の Kinesis データストリームに書き込むシンプルなアプリケーションを示しています。この例では、正しい依存関係でファイルを設定し、uber-JAR を構築してから設定パラメータを解析する方法を示しています。これにより、アプリケーションをローカル、IDE、Amazon Managed Service for Apache Flink の両方で実行できます。

コード例: GettingStarted

Table API と SQL の開始方法

この例は、 Table API と SQL を使用するシンプルなアプリケーションを示しています。同じ Java アプリケーションで DataStream API を Table API または SQL と統合する方法を示します。また、DataGenコネクタを使用して Flink アプリケーション自体内からランダムなテストデータを生成する方法も示します。外部データジェネレーターは必要ありません。

完全な例: GettingStartedTable

S3 シンク (DataStream API) の使用

この例では、 DataStream API の を使用して S3 バケットに JSON ファイルをFileSink書き込む方法を示します。

コード例: S3Sink

Kinesis ソース、標準または EFO コンシューマー、シンク (DataStream API) の使用

この例では、標準コンシューマーまたは EFO を使用して Kinesis データストリームから消費するソースを設定する方法と、Kinesis データストリームへのシンクを設定する方法を示します。

コード例: KinesisConnectors

Amazon Data Firehose シンク (DataStream API) の使用

この例では、Amazon Data Firehose (以前は Kinesis Data Firehose と呼ばれていました) にデータを送信する方法を示します。

コード例: KinesisFirehoseSink

スライディングウィンドウとタンブリングウィンドウの使用 (DataStream API)

これら 2 つの例は、処理時間ウィンドウに集約を実装する方法、スライディングまたはタンブリングして DataStream API を使用する方法を示しています。

コード例：

- WindowingSliding
- WindowingTumbling

カスタムメトリクスの使用

これら 2 つの個別の例RecordCountと は、DataStreamAPI にカスタムメトリクスを実装し、メトリクスに送信する方法WordCountを示しています CloudWatch 。

コード例: CustomMetrics

Python の例

次の例は、Python で記述されたアプリケーションを作成する方法を示しています。

> **Note**
>
> ほとんどの例は、ローカル、開発マシン、選択した IDE、Amazon Managed Service for Apache Flink の両方で実行されるように設計されています。これらは、アプリケーションパラメータを渡すために使用できるシンプルなメカニズムと、両方の環境でアプリケーションを変更せずに実行するために依存関係を正しく設定する方法を示しています。

プロジェクトの依存関係

ほとんどの PyFlink 例では、Flink コネクタや Python サードパーティーライブラリなど、JAR ファイル形式の 1 つ以上の依存関係が必要です。これらの依存関係はサンプルリポジトリには含まれていません。アプリケーションをパッキングする前に、それらをマシンにダウンロードする必要があります。依存関係の詳細については、「README: Packaging」を参照してください。

必要なコネクタの正しいバージョンをダウンロードするためのリンクについては、Flink のドキュメントを参照してください。使用している Flink バージョンの依存関係を常に使用します。以下は、Flink 1.18 で頻繁に使用されるコネクタの依存関係です。

- Kinesis SQL コネクタ
- Kafka SQL コネクタ
- Amazon Data Firehose SQL コネクタ

次の例には、依存関係を設定するために必要なコードが既に含まれています。

例

の開始方法 PyFlink

この例では、 Table API と SQL を使用した PyFlink アプリケーションの基本構造を示します。また、 PyFlink アプリケーションに単一の JAR 依存関係を含める方法も示します。この場合、Kinesis SQL コネクタが含まれています。

コード例: GettingStarted

タンブリングウィンドウとスライディングウィンドウの使用

これら 2 つの例は、 Table API を使用したイベント時のタンブリングウィンドウとスライディングウィンドウの実装を示しています。この例は、イベントタイムウィンドウに使用されるソーステーブルでウォーターマークを定義する方法も示しています。どちらの例にも、Kinesis SQL コネクタの JAR 依存関係が含まれています。

コード例：

- TumblingWindows
- SlidingWindows

S3 シンクの使用

この例では、出力を JSON ファイルとして Amazon S3 に書き込む方法を示します。Amazon S3 にファイルを書き込んでローテーションするには、Amazon S3シンクのチェックポイントを有効にする必要があります。

コード例: StreamingFileSink

Scala の例

以下の例では、Scala を使用した Apache Flink を使用してアプリケーションを作成する方法を示しています。

マルチステップアプリケーション

この例では、Scala で Flink アプリケーションをセットアップする方法を示します。依存関係を含めて uber-JAR を構築するように SBT プロジェクトを設定する方法を示します。

コード例: GettingStarted

Amazon Managed Service for Apache Flink を使用する

のクラウドセキュリティが最優先事項 AWS です。 AWS のお客様は、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の恩恵を受けることができます。

セキュリティは、 AWS とユーザーの間で共有される責任です。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 — 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責任 AWS は AWS にあります。 AWS また、 は、安全に使用できるサービスも提供します。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。Managed Service for Apache Flink 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AWS のサービスを参照してください。
- クラウドのセキュリティ — 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。

このドキュメントでは、Managed Service for Apache Flink を使用する際に責任共有モデルを適用する方法が説明されています。以下のトピックで、セキュリティおよびコンプライアンスの目的を満たすようにManaged Service for Apache Flinkを設定する方法について説明します。また、Managed Service for Apache Flink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 Managed Service for Apache Flink でのデータ保護
- Amazon Managed Service for Apache Flink のIDとアクセスマネジメント
- Managed Service for Apache Flink
- Amazon Managed Service for Apache Flink のコンプライアンス検証
- Amazon Managed Service for Apache Flink
- Managed Service for Apache Flink のインフラストラクチャセキュリティ
- Managed Service for Apache Flink のセキュリティのベストプラクティス

Amazon Managed Service for Apache Flink 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。Managed Service for Apache Flink は、Firehose や Amazon S3 などの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。

Managed Service for Apache Flink でのデータ暗号化

保管中の暗号化

Managed Service for Apache Flinkを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。

- を使用して、受信 Kinesis データストリームのデータを暗号化できますStartStreamEncryption。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、Firehose を使用して保管時に暗号化し、暗号化された Amazon S3 バケットにデータを保存できます。 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。
- Managed Service for Apache Flinkは、あらゆるストリーミングソースから読み取り、あらゆるストリーミングまたはデータベース宛先に書き込むことができます。送信元と送信先が、すべての転送中のデータと保管中のデータを暗号化することを確保します。
- アプリケーションのコードは保管時に暗号化されます。
- 耐久性のあるアプリケーションストレージは、保管時に暗号化されます。
- 実行中のアプリケーションストレージは、保管時に暗号化されます。

転送中の暗号化

Managed Service for Apache Flink は、すべての転送中のデータを暗号化します。転送中の暗号化は、すべての Managed Service for Apache Flink 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。

Managed Service for Apache Flink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 Managed Service for Apache Flinkへの転送中のデータ。
- Managed Service for Apache Flink の内部コンポーネント間の転送中のデータ。
- Managed Service for Apache Flink と Firehose の間で転送中のデータ。

キー管理

Managed Service for Apache Flinkでのデータ暗号化は、サービスマネージドキーを使用します。カスタマー管理のキーはサポートされていません。

Amazon Managed Service for Apache Flink のIDとアクセスマネジメント

AWS Identity and Access Management (IAM) は、管理者が AWS リソースへのアクセスを安全に制御 AWS のサービス するのに役立つ です。IAM の管理者は、誰を認証(サインインを許可) し、誰に Managed Service for Apache Flink リソースの使用を承認する (アクセス許可を持たせる) かを制御します。IAM は、追加料金なしで AWS のサービス 使用できる です。

トピック

- 対象者
- アイデンティティを使用した認証
- ポリシーを使用したアクセスの管理
- Amazon Managed Service for Apache FlinkはどのようにIAMと協力しますか。
- Amazon Managed Service for Apache Flink のアイデンティティベースのポリシーの例
- Amazon Managed Service for Apache Flink のアイデンティティとアクセスのトラブルシューティング
- サービス間の混乱した代理の防止

対象者

AWS Identity and Access Management (IAM) の使用方法は、Apache Flink 用 Managed Service で行う作業によって異なります。

サービスユーザー – ジョブを実行するために Managed Service for Apache Flink サービスを使用するユーザーには、必要な認証情報とアクセス許可を管理者が付与します。作業を実行するためにさらに多くの Managed Service for Apache Flink の機能を使用する際は、追加の許可が必要になる場合があります。アクセスの管理方法を理解しておくと、管理者に適切な許可をリクエストするうえで役立ちます。Managed Service for Apache Flink の機能にアクセスできない場合は、「Amazon Managed Service for Apache Flink のアイデンティティとアクセスのトラブルシューティング」を参照してください。

サービス管理者 - 社内のManaged Service for Apache Flinkリソースを担当している場合は、通常、Managed Service for Apache Flinkへのフルアクセスがあります。サービスのユーザーがどのManaged Service for Apache Flink 機能やリソースにアクセスするかを決めるのは管理者の業務です。その後、IAM 管理者にリクエストを送信して、サービスユーザーの権限を変更する必要があります。このページの情報を点検して、IAM の基本概念を理解してください。Managed Service for Apache FlinkでManaged Service for Apache Flinkと IAM を併用するには、「Amazon Managed Service for Apache FlinkはどのようにIAMと協力しますか。」を参照してください。

IAM 管理者 – IAM 管理者である場合は、Managed Service for Apache Flinkへのアクセスを管理するポリシーの作成方法の詳細について理解しておくことをお勧めします。IAM で使用できるManaged Service for Apache Flink ID ベースのポリシーの例を表示するには、「Amazon Managed Service for Apache Flink のアイデンティティベースのポリシーの例」を参照してください。

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 の認証情報は、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、「 ユーザーガイド」の「 へのサインイン AWS アカウント方法AWS サインイン 」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 Software Development Kit (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に暗号で署名します。 AWS ツールを使用しない場合は、リクエストに自分で署名する必要があります。推奨される方法を使用してリクエストを自分で署名する方法の詳細については、IAM ユーザーガイドの API AWS リクエストの署名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。例えば、 AWS では、多要素認証 (MFA) を使用してアカウントのセキュリティを向上させる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「Multi-factor

authentication」(多要素認証) および「IAM ユーザーガイド」の「AWSでの多要素認証 (MFA) の使用」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、一時的な認証情報を使用して にアクセスするための ID プロバイダーとのフェデレーションの使用を要求 AWS のサービス 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のユーザー、または ID ソースを通じ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 にアクセスすると 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ことも、独自の ID ソース内のユーザーとグループのセットに接続して同期して、すべての AWS アカウント とアプリケーションで使用することもできます。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 内のアイデンティティです。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、ロールでは一時的な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 (ロールではなく) の作成が適している場合」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 内のアイデンティティです。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。ロール を切り替える AWS Management Console ことで、 で IAM ロールを一時的に引き受けることができます。ロールを引き受けるには、 または AWS API AWS CLI オペレーションを呼び出すか、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「IAM ロールの使用」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詳細については、「IAM ユーザーガイド」の「Creating a role for a third-party Identity Provider」(サードパーティーアイデンティティプロバイダー向けロールの作成) 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 ユーザーガイド」の「アクセス許可セット」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、 (ロールをプロキシとして使用する代わりに) ポリシーをリソースに直接アタッチできま

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 でアクションを実行する場合 AWS、ユーザーは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 へのリクエストのリクエスト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。
  - 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、サービスにリンクされた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。 AWS ロールを EC2 インスタンスに割り当て、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。

IAM ロールと IAM ユーザーのどちらを使用するかについては、IAM ユーザーガイドの(IAM ユーザーではなく) IAM ロールをいつ作成したら良いのか?を参照してください。

ポリシーを使用したアクセスの管理

でアクセスを制御する AWS には、ポリシーを作成し、 AWS ID またはリソースにアタッチします。ポリシーは AWS 、アイデンティティまたはリソースに関連付けられているときにアクセス許可を定義する のオブジェクトです。 は、プリンシパル (ユーザー、ルートユーザー、またはロールセッション) がリクエストを行うときに、これらのポリシー AWS を評価します。ポリシーでの権限により、リクエストが許可されるか拒否されるかが決まります。ほとんどのポリシーは JSON ドキュメント AWS として に保存されます。JSON ポリシードキュメントの構造と内容の詳細については、IAM ユーザーガイドのJSON ポリシー概要を参照してください。

管理者は AWS JSON ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどんなリソースにどんな条件でアクションを実行できるかということです。

デフォルトでは、ユーザーやロールに権限はありません。IAM 管理者は、リソースで必要なアクションを実行するための権限をユーザーに付与する IAM ポリシーを作成できます。その後、管理者はロールに IAM ポリシーを追加し、ユーザーはロールを引き継ぐことができます。

IAM ポリシーは、オペレーションの実行方法を問わず、アクションの許可を定義します。例えば、`iam:GetRole`アクションを許可するポリシーがあるとします。そのポリシーを持つユーザーは、 AWS Management Console、、 AWS CLIまたは AWS API からロール情報を取得できます。

アイデンティティベースのポリシー

アイデンティティベースポリシーは、IAM ユーザーグループ、ユーザーのグループ、ロールなど、アイデンティティにアタッチできる JSON 許可ポリシードキュメントです。これらのポリシーは、ユーザーとロールが実行できるアクション、リソース、および条件をコントロールします。アイデンティティベースのポリシーを作成する方法については、IAM ユーザーガイドのIAM ポリシーの作成を参照してください。

アイデンティティベースのポリシーは、さらにインラインポリシーまたはマネージドポリシーに分類できます。インラインポリシーは、単一のユーザー、グループ、またはロールに直接埋め込まれています。管理ポリシーは、 内の複数のユーザー、グループ、ロールにアタッチできるスタンドアロンポリシーです AWS アカウント。管理ポリシーには、 AWS 管理ポリシーとカスタマー管理ポリシーが含まれます。マネージドポリシーまたはインラインポリシーのいずれかを選択する方法については、IAM ユーザーガイドのマネージドポリシーとインラインポリシーの比較を参照してください。

リソースベースのポリシー

リソースベースのポリシーは、リソースに添付する JSON ポリシードキュメントです。リソースベースのポリシーには例として、IAM ロールの信頼ポリシー や Amazon S3 バケットポリシー があげられます。リソースベースのポリシーをサポートするサービスでは、サービス管理者はポリシーを使用して特定のリソースへのアクセスを制御できます。ポリシーがアタッチされているリソースの場合、指定されたプリンシパルがそのリソースに対して実行できるアクションと条件は、ポリシーによって定義されます。リソースベースのポリシーでは、プリンシパルを指定する必要があります。プリンシパルには、アカウント、ユーザー、ロール、フェデレーティッドユーザー、または を含めることができます AWS のサービス。

リソースベースのポリシーは、そのサービス内にあるインラインポリシーです。リソースベースのポリシーでは、IAM の AWS マネージドポリシーを使用できません。

アクセスコントロールリスト (ACL)

アクセスコントロールリスト (ACL) は、どのプリンシパル (アカウントメンバー、ユーザー、またはロール) がリソースにアクセスするための許可を持つかを制御します。ACL はリソースベースのポリシーに似ていますが、JSON ポリシードキュメント形式は使用しません。

Amazon S3、 AWS WAF、および Amazon VPC は、ACLs。ACL の詳細については、Amazon Simple Storage Service デベロッパーガイドのアクセスコントロールリスト (ACL) の概要を参照してください。

その他のポリシータイプ

AWS は、一般的ではない追加のポリシータイプをサポートします。これらのポリシータイプでは、より一般的なポリシータイプで付与された最大の権限を設定できます。

- アクセス許可の境界 - アクセス許可の境界は、アイデンティティベースのポリシーによって IAM エンティティ (IAM ユーザーまたはロール) に付与できる権限の上限を設定する高度な機能です。エンティティにアクセス許可の境界を設定できます。結果として得られる権限は、エンティティのアイデンティティベースポリシーとそのアクセス許可の境界の共通部分になります。`Principal` フィールドでユーザーまたはロールを指定するリソースベースのポリシーでは、アクセス許可の境界は制限されません。これらのポリシーのいずれかを明示的に拒否した場合、権限は無効になります。アクセス許可の境界の詳細については、IAM ユーザーガイドのIAM エンティティのアクセス許可の境界を参照してください。
- サービスコントロールポリシー (SCPs) – SCPs は、 の組織または組織単位 (OU) に対する最大アクセス許可を指定する JSON ポリシーです AWS Organizations。 AWS Organizations は、 AWS

アカウント ビジネスが所有する複数の をグループ化して一元管理するサービスです。組織内のすべての機能を有効にすると、サービスコントロールポリシー (SCP) を一部またはすべてのアカウントに適用できます。SCP は、各 を含むメンバーアカウントのエンティティのアクセス許可を制限します AWS アカウントのルートユーザー。Organizations と SCP の詳細については、AWS Organizations ユーザーガイドの「SCP の仕組み」を参照してください。

- セッションポリシー - セッションポリシーは、ロールまたはフェデレーションユーザーの一時的なセッションをプログラムで作成する際にパラメータとして渡す高度なポリシーです。結果としてセッションの権限は、ユーザーまたはロールのアイデンティティベースポリシーとセッションポリシーの共通部分になります。また、リソースベースのポリシーから権限が派生する場合もあります。これらのポリシーのいずれかを明示的に拒否した場合、権限は無効になります。詳細については、IAM ユーザーガイドのセッションポリシーを参照してください。

複数のポリシータイプ

1 つのリクエストに複数のタイプのポリシーが適用されると、結果として作成される権限を理解するのがさらに難しくなります。複数のポリシータイプが関与する場合にリクエストを許可するかどうか AWS を決定する方法については、IAM ユーザーガイドの「ポリシー評価ロジック」を参照してください。

Amazon Managed Service for Apache FlinkはどのようにIAMと協力しますか。

IAM を使用してManaged Service for Apache Flinkへのアクセスを管理する前に、Managed Service for Apache Flinkで利用できる IAM の機能について学習します。

Amazon Managed Service for Apache Flink で使用できる IAM の機能

IAM 機能	Managed Service for Apache Flink
アイデンティティベースのポリシー	あり
リソースベースのポリシー	なし
ポリシーアクション	あり
ポリシーリソース	Yes

IAM 機能	Managed Service for Apache Flink
ポリシー条件キー	いいえ
ACL	なし
ABAC (ポリシー内のタグ)	はい
一時的な認証情報	あり
プリンシパル権限	あり
サービスロール	いいえ
サービスリンクロール	なし

Managed Service for Apache Flink およびその他の AWS のサービスがほとんどの IAM 機能と連携する方法の概要を把握するには、「IAM ユーザーガイド」の「IAM AWS と連携する のサービス」を参照してください。

Managed Service for Apache Flinkのアイデンティティベースのポリシー

アイデンティティベースポリシーをサポートする	あり

アイデンティティベースポリシーは、IAM ユーザー、ユーザーグループ、ロールなど、アイデンティティにアタッチできる JSON 許可ポリシードキュメントです。これらのポリシーは、ユーザーとロールが実行できるアクション、リソース、および条件をコントロールします。アイデンティティベースのポリシーを作成する方法については、IAM ユーザーガイドのIAM ポリシーの作成を参照してください。

IAM アイデンティティベースのポリシーでは、許可または拒否するアクションとリソース、およびアクションを許可または拒否する条件を指定できます。プリンシパルは、それが添付されているユーザーまたはロールに適用されるため、アイデンティティベースのポリシーでは指定できません。JSON ポリシーで使用できるすべての要素について学ぶには、IAM ユーザーガイドのIAM JSON ポリシーの要素のリファレンスを参照してください。

Managed Service for Apache Flink のアイデンティティベースのポリシーの例

Managed Service for Apache Flink ID ベースのポリシーの例は、「Amazon Managed Service for Apache Flink のアイデンティティベースのポリシーの例」でご確認ください。

Managed Service for Apache Flink 内のリソースベースのポリシー

リソースベースのポリシーのサポート	あり

リソースベースのポリシーは、リソースに添付する JSON ポリシードキュメントです。リソースベースのポリシーには例として、IAM ロールの信頼ポリシー や Amazon S3 バケットポリシー があげられます。リソースベースのポリシーをサポートするサービスでは、サービス管理者はポリシーを使用して特定のリソースへのアクセスを制御できます。ポリシーがアタッチされているリソースの場合、指定されたプリンシパルがそのリソースに対して実行できるアクションと条件は、ポリシーによって定義されます。リソースベースのポリシーでは、プリンシパルを指定する必要があります。プリンシパルには、アカウント、ユーザー、ロール、フェデレーティッドユーザー、または を含めることができます AWS のサービス。

クロスアカウントアクセスを有効にするには、アカウント全体、または別のアカウントの IAM エンティティをリソースベースのポリシーのプリンシパルとして指定します。リソースベースのポリシーにクロスアカウントのプリンシパルを追加しても、信頼関係は半分しか確立されない点に注意してください。プリンシパルとリソースが異なる にある場合 AWS アカウント、信頼されたアカウントの IAM 管理者は、プリンシパルエンティティ (ユーザーまたはロール) にリソースへのアクセス許可も付与する必要があります。IAM 管理者は、アイデンティティベースのポリシーをエンティティにアタッチすることで権限を付与します。ただし、リソースベースのポリシーで、同じアカウントのプリンシパルへのアクセス権が付与されている場合は、アイデンティティベースのポリシーをさらに付与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」の「IAM でのクロスアカウントリソースアクセス」を参照してください。

Managed Service for Apache Flinkのポリシーアクション

ポリシーアクションに対するサポート	あり

管理者は AWS JSON ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどんなリソースにどんな条件でアクションを実行できるかということです。

JSON ポリシーのAction要素には、ポリシー内のアクセスを許可または拒否するために使用できるアクションが記述されます。ポリシーアクションの名前は通常、関連付けられた AWS API オペレーションと同じです。一致する API オペレーションのない許可のみのアクションなど、いくつかの例外があります。また、ポリシーに複数のアクションが必要なオペレーションもあります。これらの追加アクションは、依存アクションと呼ばれます。

このアクションは、関連付けられたオペレーションを実行するための権限を付与するポリシーで使用されます。

ElastiCache アクションの一覧については、「サービス認証リファレンス」の「Amazon Managed Service for Apache Flink によって定義されるアクション」を参照してください。

Managed Service for Apache Flink のポリシーアクションは、アクションの前に、プレフィックス を使用します。

```
Kinesis Analytics
```

単一のステートメントで複数のアクションを指定するには、アクションをカンマで区切ります。

```
"Action": [
      "Kinesis Analytics:action1",
      "Kinesis Analytics:action2"
         ]
```

ワイルドカード (*) を使用して複数アクションを指定できます。例えば、Describe という単語で始まるすべてのアクションを指定するには、次のアクションを含めます。

```
"Action": "Kinesis Analytics:Describe*"
```

Managed Service for Apache Flink ID ベースのポリシーの例は、「Amazon Managed Service for Apache Flink のアイデンティティベースのポリシーの例」でご確認ください。

Amazon Managed Service for Apache Flink サポートの新しいリソース

ポリシーリソースに対するサポート	あり

管理者は AWS JSON ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースにどのような条件でアクションを実行できるかということです。

Resource JSON ポリシー要素は、アクションが適用されるオブジェクトを指定します。ステートメントには、Resource または NotResource 要素を含める必要があります。ベストプラクティスとして、Amazon リソースネーム (ARN) を使用してリソースを指定します。これは、リソースレベルの許可と呼ばれる特定のリソースタイプをサポートするアクションに対して実行できます。

オペレーションのリスト化など、リソースレベルの権限をサポートしないアクションの場合は、ステートメントがすべてのリソースに適用されることを示すために、ワイルドカード (*) を使用します。

```
"Resource": "*"
```

ElastiCache リソースタイプとその ARN の一覧については、「サービス認証リファレンス」の「Amazon Managed Service for Apache Flink で定義されるリソース」を参照してください。各リソースの ARN を指定できるアクションについては、「Amazon Managed Service for Apache Flink によって定義されるアクション」を参照してください。

Managed Service for Apache Flink ID ベースのポリシーの例は、「Amazon Managed Service for Apache Flink のアイデンティティベースのポリシーの例」でご確認ください。

Managed Service for Apache Flinkのポリシー条件キー

サービス固有のポリシー条件キーのサポート	あり

管理者は AWS JSON ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどんなリソースにどんな条件でアクションを実行できるかということです。

Condition 要素 (または Condition ブロック) を使用すると、ステートメントが有効な条件を指定できます。Condition 要素はオプションです。イコールや未満などの 条件演算子 を使用して条件式を作成することで、ポリシーの条件とリクエスト内の値を一致させることができます。

1 つのステートメントに複数の Condition 要素を指定する場合、または 1 つの Condition 要素に複数のキーを指定する場合、 AWS では AND 論理演算子を使用してそれらを評価します。1 つの条

件キーに複数の値を指定すると、 は論理ORオペレーションを使用して条件 AWS を評価します。ステートメントの権限が付与される前にすべての条件が満たされる必要があります。

条件を指定する際にプレースホルダー変数も使用できます。例えば IAM ユーザーに、IAM ユーザー名がタグ付けされている場合のみリソースにアクセスできる権限を付与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IAM ポリシーの要素: 変数およびタグを参照してください。

AWS は、グローバル条件キーとサービス固有の条件キーをサポートします。すべての AWS グローバル条件キーを確認するには、「IAM ユーザーガイド」のAWS 「 グローバル条件コンテキストキー」を参照してください。

ElastiCache 条件キーの一覧については、「サービス認証リファレンス」の「Amazon Managed Service for Apache Flink の条件キー」を参照してください。条件キーを使用できるアクションとリソースについては、「Amazon Managed Service for Apache Flink によって定義されたアクション」を参照してください。

Managed Service for Apache Flink ID ベースのポリシーの例は、「Amazon Managed Service for Apache Flink のアイデンティティベースのポリシーの例」でご確認ください。

Managed Service for Apache Flinでのアクセスコントロールリスト (ACLs)

ACL のサポート	なし

アクセスコントロールリスト (ACL) は、どのプリンシパル (アカウントメンバー、ユーザー、またはロール) がリソースにアクセスするための許可を持つかをコントロールします。ACL はリソースベースのポリシーに似ていますが、JSON ポリシードキュメント形式は使用しません。

Managed Service for Apache Flink の属性ベースのアクセス制御 (ABAC)

ABAC のサポート (ポリシー内のタグ)	はい

属性ベースのアクセス制御 (ABAC) は、属性に基づいてアクセス許可を定義するアクセス許可戦略です。では AWS、これらの属性はタグ と呼ばれます。タグは、IAM エンティティ (ユーザーまたはロール) および多くの AWS リソースにアタッチできます。エンティティとリソースのタグ付けは、ABAC の最初の手順です。その後、プリンシパルのタグがアクセスしようとしているリソースのタグと一致した場合にオペレーションを許可するように ABAC ポリシーをします。

ABAC は、急成長する環境やポリシー管理が煩雑になる状況で役立ちます。

タグに基づいてアクセスを管理するには、aws:ResourceTag/*key-name*、aws:RequestTag/*key-name*、または aws:TagKeys の条件キーを使用して、ポリシーの条件要素でタグ情報を提供します。

サービスがすべてのリソースタイプに対して 3 つの条件キーすべてをサポートする場合、そのサービスの値はありです。サービスが一部のリソースタイプに対してのみ 3 つの条件キーのすべてをサポートする場合、値は「部分的」になります。

ABAC の詳細については、IAM ユーザーガイドのABAC とは?を参照してください。ABAC をセットアップするステップを説明するチュートリアルについては、IAM ユーザーガイドの属性に基づくアクセスコントロール (ABAC) を使用するを参照してください。

Managed Service for Apache Flink での一時的な認証情報の使用

一時的な認証情報のサポート	あり

一部の は、一時的な認証情報を使用してサインインすると機能 AWS のサービス しません。一時的な認証情報 AWS のサービス を使用する などの詳細については、IAM ユーザーガイドのAWS のサービス「IAM と連携する 」を参照してください。

ユーザー名とパスワード以外の AWS Management Console 方法で にサインインする場合、一時的な認証情報を使用します。例えば、会社の Single Sign-On (SSO) リンク AWS を使用して にアクセスすると、そのプロセスによって一時的な認証情報が自動的に作成されます。また、ユーザーとしてコンソールにサインインしてからロールを切り替える場合も、一時的な認証情報が自動的に作成されます。ロールの切り替えに関する詳細については、IAM ユーザーガイドのロールへの切り替え (コンソール)を参照してください。

一時的な認証情報は、 AWS CLI または AWS API を使用して手動で作成できます。その後、これらの一時的な認証情報を使用して . AWS recommends にアクセスできます AWS。これは、長期的なアクセスキーを使用する代わりに、一時的な認証情報を動的に生成することを推奨しています。詳細については、IAM の一時的セキュリティ認証情報を参照してください。

Managed Service for Apache Flink のクロスサービスプリンシパル許可

フォワードアクセスセッション (FAS) をサポート	あり

IAM ユーザーまたはロールを使用して で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 へのリクエストのリクエスト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。

Managed Service for Apache Flinkのサービスロール

サービスロールに対するサポート	あり

サービスロールとは、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。

> **Warning**
>
> サービスロールの許可を変更すると、 Managed Service for Apache Flinkの機能が破損する可能性があります。Managed Service for Apache Flink が指示する場合以外は、サービスロールを編集しないでください。

Managed Service for Apache Flinkのサービスにリンクされたロール

サービスリンクロールのサポート	あり

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、サービスにリンクされた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。

サービスリンクロールの作成または管理の詳細については、IAM と提携するAWS のサービスを参照してください。表の中から、[Service-linked role] (サービスにリンクされたロール) 列に Yes と記載されたサービスを見つけます。サービスリンクロールに関するドキュメントをサービスで表示するには、はい リンクを選択します。

Amazon Managed Service for Apache Flink のアイデンティティベースのポリシーの例

デフォルトでは、ユーザーおよびロールには Managed Service for Apache Flinkリソースを作成または変更する許可がありません。また、、 AWS Command Line Interface （ AWS CLI ） AWS Management Console、または AWS API を使用してタスクを実行することはできません。IAM 管理者は、リソースで必要なアクションを実行するための権限をユーザーに付与する IAM ポリシーを作成できます。その後、管理者はロールに IAM ポリシーを追加し、ユーザーはロールを引き受けることができます。

これらサンプルの JSON ポリシードキュメントを使用して、IAM アイデンティティベースのポリシーを作成する方法については、IAM ユーザーガイドのIAM ポリシーの作成を参照してください。

ElastiCache が定義するアクションとリソースタイプ (リソースタイプごとの ARN の形式を含む) の詳細については、「サービス認証リファレンス」の「Amazon Managed Service for Apache Flink のアクション、リソース、および条件キー」を参照してください。

トピック

- ポリシーのベストプラクティス
- Amazon Managed Service for Apache Flink を使用する
- 自分の権限の表示をユーザーに許可する

ポリシーのベストプラクティス

ID ベースのポリシーは、ユーザーのアカウントで誰かが Managed Service for Apache Flink リソースを作成、アクセス、または削除できるどうかを決定します。これらのアクションを実行すると、AWS アカウントに料金が発生する可能性があります。アイデンティティベースポリシーを作成したり編集したりする際には、以下のガイドラインと推奨事項に従ってください:

- AWS 管理ポリシーを開始し、最小特権のアクセス許可に移行する – ユーザーとワークロードにアクセス許可を付与するには、多くの一般的なユースケースにアクセス許可を付与する AWS 管理ポリシーを使用します。これらは で使用できます AWS アカウント。ユースケースに固有の AWS カ

スタマー管理ポリシーを定義して、アクセス許可をさらに減らす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「AWS マネージドポリシー」または「ジョブ機能のAWS マネージドポリシー」を参照してください。

- 最小特権を適用する – IAM ポリシーで許可を設定する場合は、タスクの実行に必要な許可のみを付与します。これを行うには、特定の条件下で特定のリソースに対して実行できるアクションを定義します。これは、最小特権アクセス許可とも呼ばれています。IAM を使用して許可を適用する方法の詳細については、IAM ユーザーガイドのIAM でのポリシーとアクセス許可を参照してください。
- IAM ポリシーで条件を使用してアクセスをさらに制限する - ポリシーに条件を追加して、アクションやリソースへのアクセスを制限できます。例えば、ポリシー条件を記述して、すべてのリクエストを SSL を使用して送信するように指定できます。条件を使用して、 などの特定の を介してサービスアクションが使用される場合に AWS のサービス、サービスアクションへのアクセスを許可することもできます AWS CloudFormation。詳細については、「IAM ユーザーガイド」の [IAM JSON policy elements: Condition] (IAM JSON ポリシー要素:条件) を参照してください。
- IAM Access Analyzer を使用して IAM ポリシーを検証し、安全で機能的な権限を確保する - IAM Access Analyzer は、新規および既存のポリシーを検証して、ポリシーが IAM ポリシー言語 (JSON) および IAM のベストプラクティスに準拠するようにします。IAM アクセスアナライザーは 100 を超えるポリシーチェックと実用的な推奨事項を提供し、安全で機能的なポリシーの作成をサポートします。詳細については、IAM ユーザーガイドのIAM Access Analyzer ポリシーの検証を参照してください。
- 多要素認証 (MFA) を要求する – で IAM ユーザーまたはルートユーザーを必要とするシナリオがある場合は AWS アカウント、セキュリティを強化するために MFA を有効にします。API オペレーションが呼び出されるときに MFA を必須にするには、ポリシーに MFA 条件を追加します。詳細については、IAM ユーザーガイドのMFA 保護 API アクセスの設定を参照してください。

IAM でのベストプラクティスの詳細については、IAM ユーザーガイドのIAM でのセキュリティのベストプラクティスを参照してください。

Amazon Managed Service for Apache Flink を使用する

Amazon Managed Service for Apache Flink コンソールにアクセスするには、許可の最小限のセットが必要です。これらの許可により、 AWS アカウントの Managed Service for Apache Flink リソースの詳細をリストおよび表示できます。最小限必要な許可よりも制限が厳しいアイデンティティベースのポリシーを作成すると、そのポリシーを持つエンティティ (ユーザーまたはロール) に対してコンソールが意図したとおりに機能しません。

AWS CLI または AWS API のみを呼び出すユーザーには、最小限のコンソールアクセス許可を付与する必要はありません。代わりに、実行しようとしている API オペレーションに一致するアクションのみへのアクセスが許可されます。

ユーザーとロールが引き続き Managed Service for Apache Flink コンソールを使用できるようにするには、エンティティに Managed Service for Apache Flink `ConsoleAccess`または `ReadOnly` AWS マネージドポリシーもアタッチします。詳細については、IAM ユーザーガイドのユーザーへの許可の追加を参照してください。

自分の権限の表示をユーザーに許可する

この例では、ユーザーアイデンティティにアタッチされたインラインおよびマネージドポリシーの表示を IAM ユーザーに許可するポリシーの作成方法を示します。このポリシーには、コンソールで、または AWS CLI または AWS API を使用してプログラムでこのアクションを実行するアクセス許可が含まれています。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
```

```
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

Amazon Managed Service for Apache Flink のアイデンティティとアクセスのトラブルシューティング

以下の情報は、Managed Service for Apache Flinkと IAM を併用した場合に発生しうる一般的な問題の診断と解決に役立ちます。

トピック

- Managed Service for Apache Flink でアクションを実行する権限がありません。
- iam を実行する権限がありません。PassRole
- AWS アカウント外のユーザーに Managed Service for Apache Flink リソースへのアクセスを許可したい

Managed Service for Apache Flink でアクションを実行する権限がありません。

がアクションを実行する権限がないと AWS Management Console 通知した場合は、管理者に連絡してサポートを依頼する必要があります。担当の管理者はお客様のユーザー名とパスワードを発行した人です。

以下のエラー例は、mateojackson ユーザーがコンソールを使用して架空の *my-example-widget* リソースに関する詳細情報を表示しようとしているが、架空の Kinesis Analytics:*GetWidget* 許可がないという場合に発生します。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform: Kinesis
 Analytics:GetWidget on resource: my-example-widget
```

この場合、Mateo は、Kinesis Analytics:*GetWidget* アクションを使用して *my-example-widget* リソースへのアクセスが許可されるように、管理者にポリシーの更新を依頼します。

iam を実行する権限がありません。PassRole

`iam:PassRole` アクションを実行する権限がないというエラーが表示された場合は、ポリシーを更新して Managed Service for Apache Flink にロールを渡すことができるようにする必要があります。

一部の AWS のサービス では、新しいサービスロールまたはサービスにリンクされたロールを作成する代わりに、そのサービスに既存のロールを渡すことができます。そのためには、サービスにロールを渡す権限が必要です。

以下の例のエラーは、`marymajor` という IAM ユーザーがコンソールを使用してManaged Service for Apache Flinkでアクションを実行しようする場合に発生します。ただし、このアクションをサービスが実行するには、サービスロールから付与された権限が必要です。メアリーには、ロールをサービスに渡す許可がありません。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

この場合、Mary のポリシーを更新してメアリーに `iam:PassRole` アクションの実行を許可する必要があります。

サポートが必要な場合は、 AWS 管理者にお問い合わせください。サインイン認証情報を提供した担当者が管理者です。

AWS アカウント外のユーザーに Managed Service for Apache Flink リソースへのアクセスを許可したい

他のアカウントのユーザーや組織外の人が、リソースにアクセスするために使用できるロールを作成できます。ロールの引き受けを委託するユーザーを指定できます。リソースベースのポリシーまたはアクセスコントロールリスト (ACL) をサポートするサービスの場合、それらのポリシーを使用して、リソースへのアクセスを付与できます。

詳細については、以下を参照してください:

- Managed Service for Apache Flink がこれらの機能をサポートしているかどうかを確認するには、「Amazon Managed Service for Apache Flinkはどのようにしますか。」を参照してください。
- 所有している のリソースへのアクセスを提供する方法については、IAM ユーザーガイドの AWS アカウント「所有している別の の IAM ユーザーへのアクセスを提供する」を参照してください。AWS アカウント

- リソースへのアクセスをサードパーティー に提供する方法については AWS アカウント、IAM ユーザーガイドの「サードパーティー AWS アカウント が所有する へのアクセスを提供する」を参照してください。
- ID フェデレーションを介してアクセスを提供する方法については、IAM ユーザーガイドの外部で認証されたユーザー (ID フェデレーション) へのアクセスの許可を参照してください。
- クロスアカウントアクセスでのロールとリソースベースのポリシーの使用の違いについては、IAM ユーザーガイドの「IAM でのクロスアカウントリソースアクセス」を参照してください。

サービス間の混乱した代理の防止

では AWS、あるサービス (呼び出し元サービス) が別のサービス (呼び出し先サービス) 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、Apache Flink のマネージドサービスに固有のサービス間の混乱した代理問題の防止に焦点を当てています。ただし、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した代理問題セクションを参照してください。

Managed Service for Apache Flink のコンテキストでは、ロール信頼ポリシーで aws:SourceArn および aws:SourceAccount グローバル条件コンテキストキーを使用して、ロールへのアクセスを、予想されるリソースによって生成されたリクエストのみに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、Managed Service for Apache Flinkが使用するリソースのARNです。この値は、次の形式 `arn:aws:kinesisanalytics:region:account:resource` 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:*`。

Managed Service for Apache Flink に提供するロールのポリシーと、ユーザー用に生成されたロールの信頼ポリシーをこれらのキーを使用することができます。

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] を選択します。

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

Managed Service for Apache Flink

Managed Service for Apache Flinkは、アプリケーションに監視機能を提供します。詳細については、「ロギングとモニタリング」を参照してください。

Amazon Managed Service for Apache Flink 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 Managed Service for Apache Flink 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、「」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、「 でのレポートのダウンロード AWS Artifact」を参照してください。

Managed Service for Apache Flink を使用する際のお客様のコンプライアンス責任は、組織のデータの機密性や組織のコンプライアンス目的、適用可能な法律、規制によって決定されます。Managed Service for Apache Flink の使用が HIPAA、PCI、または FedRAMP などの規格に準拠していることを前提としている場合、 AWS は以下を支援するリソースを提供します。

- 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 にデプロイする手順について説明します AWS。
- Architecting for HIPAA Security and Compliance on Amazon Web Services 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、リソース設定が社内プラクティス、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。

FedRAMP

AWS FedRAMP コンプライアンスプログラムには、FedRAMP 認定サービスとして Managed Service for Apache Flink が含まれています。連邦または商用のお客様は、このサービスを使用して、影響レベルの高いデータを含む AWS GovCloud （米国) リージョンの承認境界、および中程度レベルのデータを含む米国東部 (バージニア北部）、米国東部 (オハイオ）、米国西部 (北カリフォルニア）、米国西部 (オレゴン) リージョンで、機密性の高いワークロードを処理して保存できます。

AWS FedRAMP セキュリティパッケージへのアクセスは、FedRAMP PMO、 AWS セールスアカウントマネージャーを通じてリクエストすることも、 AWS Artifact の AWS Artifact からダウンロードすることもできます。

詳細については、「FedRAMP」を参照してください。

Amazon Managed Service for Apache Flink

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、および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。

AWS グローバルインフラストラクチャに加えて、Apache Flink 用 Managed Service 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いくつかの機能があります。

ディザスタリカバリ

Managed Service for Apache Flink 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。Managed Service for Apache Flink は、複数の冗長化のメカニズムによってこれを実現します。各Managed Service for Apache Flinkアプリケーションはシングルテナントの Apache Flinkクラスターで実行されます。Apache Flink クラスターは、 JobMananger 複数のアベイラビリティーゾーンにまたがる Zookeeper を使用して、高可用性モードで で実行されます。Managed Service for Apache Flink は、AmazonEKSを使用してApache Flinkをデプロ

イします。Amazon EKS では、アベイラビリティーゾーン全体で AWS リージョンごとに複数の Kubernetes ポッドが使用されます。障害が発生した場合、Managed Service for Apache Flink はまず、アプリケーションのチェックポイント (利用可能な場合) を使用して、実行中の Apache Flink クラスター内のアプリケーションの回復を試みます。

Managed Service for Apache Flink は、チェックポイントとスナップショットを使用してアプリケーションの状態をバックアップします。

- チェックポイントは、Managed Service for Apache Flink が自動的に定期的に作成し、障害からの復元に使用するアプリケーションの状態のバックアップです。
- スナップショットは、手動で作成して復元するアプリケーションの状態のバックアップです。

チェックポイントの詳細については、耐障害性を参照してください。

バージョニング

保存されているアプリケーションの状態は、次のようにバージョン管理されます。

- チェックポイントはサービスによって自動的にバージョン管理されます。サービスがチェックポイントを使用してアプリケーションを再起動する場合、最新のチェックポイントが使用されます。
- セーブポイントは、 CreateApplicationSnapshotアクションの SnapshotNameパラメータを使用してバージョニングされます。

Managed Service for Apache Flink は、チェックポイントとセーブポイントに保存されているデータを暗号化します。

Managed Service for Apache Flink 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 Managed Service for Apache Flink 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の手順で保護されています。

AWS が公開している API コールを使用して、ネットワーク経由で Managed Service for Apache Flink にアクセスします。Managed Service for Apache FlinkへのAPIコールはすべて、Transport Layer Security (TLS)で保護されて、IAMで認証されます。クライアントは TLS 1.2 以降をサポートしている必要があります。また、一時的ディフィー・ヘルマン Ephemeral Diffie-Hellman (DHE) や

Elliptic Curve Ephemeral Diffie-Hellman (ECDHE) などの Perfect Forward Secrecy (PFS) を使用した暗号スイートもクライアントでサポートされている必要があります。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。

Managed Service for Apache Flink のセキュリティのベストプラクティス

Amazon Managed Service for Apache Flink 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、これらは処方箋ではなく、有用な考慮事項と見なしてください。

最小特権アクセスの実装

アクセス許可を付与する場合、どのユーザーにどの Managed Service for Apache Flink リソースに対するアクセス許可を付与するかは、お客様が決定します。これらのリソースで許可したい特定のアクションを有効にするのも、お客様になります。このため、タスクの実行に必要なアクセス許可のみを付与する必要があります。最小特権アクセスの実装は、セキュリティリスクと、エラーや悪意によってもたらされる可能性のある影響の低減における基本になります。

IAM ロールを使用して他の Amazon のサービスにアクセスする

Managed Service for Apache Flink アプリケーションには、Kinesis データストリーム、Firehose ストリーム、Amazon S3 バケットなどの他の サービスのリソースにアクセスするための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

依存リソースにサーバー側の暗号化を実装する

保管中のデータと転送中のデータは Managed Service for Apache Flink 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 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

Managed Service for Apache Flink は AWS CloudTrail、Managed Service for Apache Flink 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。

で収集された情報を使用して CloudTrail、Managed Service for Apache Flink に対するリクエスト、リクエスト元の IP アドレス、リクエスト者、リクエスト日時などの詳細を確認できます。

詳細については、「the section called “の使用 AWS CloudTrail”」を参照してください。

Amazon Managed Service for Apache Flink でのログ記録とモニタリング

モニタリングは、Managed Service for Apache Flink 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。

Managed Service for Apache Flink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、通常の Managed Service for Apache Flink パフォーマンスのベースラインを環境に確立します。これを行うには、さまざまな時間帯とさまざまな負荷条件下でのパフォーマンスを測定します。Managed Service for Apache Flink 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。

トピック

- ログ記録
- モニタリング
- アプリケーションログ記録の設定
- CloudWatch Logs Insights を使用したログの分析
- Managed Service for Apache Flink でのメトリクスとディメンションの表示
- CloudWatch ログへのカスタムメッセージの書き込み
- を使用した Managed Service for Apache Flink API コールのログ記録 AWS CloudTrail

ログ記録

実稼働アプリケーションでは、エラーや障害を理解するためにロギングが重要である。ただし、ログ記録サブシステムはログエントリを収集して CloudWatch ログに転送する必要があります。一部のログ記録は適切で望ましいですが、広範なログ記録を行うとサービスが過負荷になり、Flink アプリケーションが遅れる可能性があります。例外や警告をログに記録するのは確かに良い考えです。しかし、Flink アプリケーションによって処理されるすべてのメッセージに対してログ メッセージを生成することはできません。Flink は、高いスループットと低いレイテンシを実現するために最適化されていますが、ロギング サブシステムは最適化されていません。処理されたすべてのメッセージのログ出力を生成することが本当に必要な場合は、Flink アプリケーション DataStream 内で追加のと適切なシンクを使用して、データを Amazon S3 または に送信します CloudWatch。この目的には Javaロギングシステムを使用しないでください。さらに、Managed Service for Apache FlinkDebug `Monitoring Log Level` の設定によって大量のトラフィックが生成され、バックプレッシャが発生する可能性があります。アプリケーションの問題を積極的に調査する場合にのみ使用してください。

CloudWatch Logs Insights を使用したログのクエリ

CloudWatch Logs Insights は、ログを大規模にクエリするための強力なサービスです。お客様はその機能を活用してログを迅速に検索して、運用イベント中のエラーを特定して軽減する必要があります。

次のクエリは、すべてのタスク マネージャー ログから例外を検索し、発生時刻に従って例外を並べ替えます。

```
fields @timestamp, @message
| filter isPresent(throwableInformation.0) or isPresent(throwableInformation) or
 @message like /(Error|Exception)/
| sort @timestamp desc
```

その他の便利なクエリについては、クエリの例を参照してください。

モニタリング

実稼働環境でストリーミング アプリケーションを実行する場合は、アプリケーションを継続的かつ無期限に実行するように設定します。Flink アプリケーションだけでなく、すべてのコンポーネントのモニターと適切なアラームを実装することが重要です。そうでなければ、新たな問題を早期に見逃

してしまい、運用上の事件が完全に解明され、軽減することがはるかに難しくなってしまうと気付くリスクがあります。一般的に監視すべき事項は次のとおりです。

- ソースはデータを取り込んでいますか。
- データはソース(ソースの観点から)から読み込まれていますか。
- Flinkアプリケーションはデータを受信していますか?
- Flinkアプリケーションは対応できるのか、それとも遅れていますか。
- Flinkアプリケーションは(アプリケーションの観点から)データをシンクに永続化していますか?
- シンクはデータを受信していますか?

その場合は、Flinkアプリケーションについてより具体的なメトリクスを検討する必要があります。このCloudWatch ダッシュボードは、適切な開始点を提供します。実稼働アプリケーションのモニタリング対象となるメトリクスの詳細については、 Amazon Managed Service for Apache Flink での CloudWatch アラームの使用 を参照してください。これらのメトリクスには次のものが含まれます。

- records_lag_maxとMillisBehindLatest — アプリケーションが Kinesis または Kafka からデータを消費している場合、これらのメトリックは、アプリケーションが遅れていて、現在の負荷に対応するためにスケーリングする必要があるかどうかを示します。これは、あらゆる種類のアプリケーションで追跡しやすい汎用的な指標です。しかし、これはリアクティブスケーリング、つまりアプリケーションがすでに遅れている場合にのみ使用できます。
- cpuUtilization および heapMemoryUtilization – これらのメトリクスは、アプリケーションの全体的なリソース使用率を適切に示し、アプリケーションが I/O バインドされていない限り、プロアクティブスケーリングに使用できます。
- ダウンタイム — ダウンタイムがゼロより大きい場合は、アプリケーションに障害が発生したことを示します。値が0より大きい場合、アプリケーションはデータを処理していません。
- lastCheckpointSize および lastCheckpointDuration- これらのメトリクスは、 状態で保存されるデータの量と、チェックポイントの取得にかかる時間をモニタリングします。チェックポイントが増えたり、時間がかかったりしても、アプリケーションはチェックポイント処理に時間を費やし続けて、実際の処理のサイクルは少なくなります。場合によっては、チェックポイントが増えすぎたり、時間がかかりすぎて失敗したりすることがあります。絶対値を監視することに加えて、顧客は`RATE(lastCheckpointSize)`と`RATE(lastCheckpointDuration)`による変化率の監視を検討する必要もあります。
- numberOfFailedチェックポイント — このメトリクスは、アプリケーションが起動してから失敗したチェックポイントの数をカウントします。アプリケーションによっては、チェックポイントが失

敗することがあっても許容できる場合があります。ただし、チェックポイントが定期的に失敗する場合は、アプリケーションが異常である可能性が高く、注意深く見ることが必要です。絶対値ではなく勾配でアラームを出すようにモニタリング`RATE(numberOfFailedCheckpoints)`をお勧めします。

アプリケーションログ記録の設定

Managed Service for Apache Flink アプリケーションに Amazon CloudWatch ログ記録オプションを追加することで、アプリケーションイベントや設定の問題を監視できます。

このトピックでは、アプリケーションイベントを CloudWatch Logs ストリームに書き込むようにアプリケーションを設定する方法について説明します。 CloudWatch ログ記録オプションは、アプリケーションがアプリケーションイベントを CloudWatch Logs に書き込む方法を設定するために使用するアプリケーション設定とアクセス許可のコレクションです。 AWS Management Console または AWS Command Line Interface () を使用して、 CloudWatch ログ記録オプションを追加および設定できますAWS CLI。

アプリケーションへの CloudWatch ログ記録オプションの追加については、次の点に注意してください。

- コンソールを使用して CloudWatch ログ記録オプションを追加すると、Managed Service for Apache Flink によって CloudWatch ロググループとログストリームが作成され、アプリケーションがログストリームに書き込むために必要なアクセス許可が追加されます。
- API を使用して CloudWatch ログ記録オプションを追加する場合は、アプリケーションのロググループとログストリームも作成し、アプリケーションがログストリームに書き込むために必要なアクセス許可を追加する必要があります。

このトピックには、次のセクションが含まれています。

- コンソールを使用した CloudWatch ログ記録の設定
- CLI を使用した CloudWatch ログ記録の設定
- アプリケーションモニタリングレベル
- ベストプラクティスのログ記録
- ログ記録のトラブルシューティング
- 次のステップ

コンソールを使用した CloudWatch ログ記録の設定

コンソールでアプリケーションの CloudWatch ログ記録を有効にすると、 CloudWatch ロググループとログストリームが自動的に作成されます。また、アプリケーションのアクセス許可ポリシーは、ストリームへの書き込みアクセス許可で更新されます。

Managed Service for Apache Flink は、次の規則を使用して という名前のロググループを作成します。ここで、 *ApplicationName*はアプリケーションの名前です。

```
/AWS/KinesisAnalytics/ApplicationName
```

。Managed Service for Apache Flinkは、新しいロググループに以下の名前でログストリームを作成します。

```
kinesis-analytics-log-stream
```

アプリケーションの設定ページの監視ログレベルセクションを使用して、アプリケーション監視メトリックレベルと監視ログレベルを設定します。アプリケーションログレベルの詳細については、 the section called “アプリケーションモニタリングレベル” を参照してください。

CLI を使用した CloudWatch ログ記録の設定

を使用して CloudWatch ログ記録オプションを追加するには AWS CLI、次の手順を実行します。

- CloudWatch ロググループとログストリームを作成します。
- CreateApplication アクションを使用してアプリケーションを作成するときにログ記録オプションを追加するか、 AddApplicationCloudWatchLoggingOption アクションを使用して既存のアプリケーションにログ記録オプションを追加します。
- ログに書き込むための権限をアプリケーションのポリシーに追加する

このセクションは、以下のトピックで構成されます。

- CloudWatch ロググループとログストリームの作成
- アプリケーションの CloudWatch ログ記録オプションの使用
- CloudWatch ログストリームに書き込むアクセス許可の追加

CloudWatch ロググループとログストリームの作成

Logs コンソールまたは API を使用して CloudWatch 、ログ CloudWatch グループとストリームを作成します。 CloudWatch ロググループとログストリームの作成については、「ロググループとログストリームの使用」を参照してください。

アプリケーションの CloudWatch ログ記録オプションの使用

次の API アクションを使用して、新規または既存のアプリケーションに CloudWatch ログオプションを追加したり、既存のアプリケーションのログオプションを変更したりできます。JSON ファイルを API アクションの入力に使用する方法の詳細については、Apache Flink API のマネージドサービスのサンプルコード を参照してください。

アプリケーション作成時の CloudWatch ログオプションの追加

次の例は、アプリケーションの作成時に CreateApplicationアクションを使用して CloudWatch ログオプションを追加する方法を示しています。この例では、 *CloudWatch ######## Amazon ####### (ARN) ###################*自分の情報を追加します。これらのアクションの詳細については、「CreateApplication」を参照してください。

```
{
    "ApplicationName": "test",
    "ApplicationDescription": "test-application-description",
    "RuntimeEnvironment": "FLINK-1_15",
    "ServiceExecutionRole": "arn:aws:iam::123456789123:role/myrole",
    "ApplicationConfiguration": {
        "ApplicationCodeConfiguration": {
            "CodeContent": {
                "S3ContentLocation":{
                            "BucketARN": "arn:aws:s3:::mybucket",
                            "FileKey": "myflink.jar"
                }
            },
            "CodeContentType": "ZIPFILE"
        }
    },
    "CloudWatchLoggingOptions": [{
      "LogStreamARN": "<Amazon Resource Name (ARN) of the CloudWatch log stream to add
 to the new application>"
 }]
}
```

既存のアプリケーションへの CloudWatch ログオプションの追加

次の例は、 AddApplicationCloudWatchLoggingOptionアクションを使用して既存のアプリケーションに CloudWatch ログオプションを追加する方法を示しています。例では、各 ############## を独自の情報に置き換えます。これらのアクションの詳細については、「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>"
   },
   "CurrentApplicationVersionId": <Version of the application to add the log to>
}
```

既存の CloudWatch ログオプションの更新

次の例は、 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を示しています。例では、各 ############## を独自の情報に置き換えます。これらのアクションの詳細については、「UpdateApplication」を参照してください。

```
{
   "ApplicationName": "<Name of the application to update the log option for>",
   "CloudWatchLoggingOptionUpdates": [
         {
            "CloudWatchLoggingOptionId": "<ID of the logging option to modify>",
            "LogStreamARNUpdate": "<ARN of the new log stream to use>"
         }
      ],
   "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

次の例は、 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を示しています。例では、各 ########## を独自の情報に置き換えます。これらのアクションの詳細については、「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>
}
```

アプリケーションのログ記録レベルの設定

アプリケーションロギングのレベルを設定するには、CreateApplicationアクションのMonitoringConfigurationパラメータまたはUpdateApplicationアクションのMonitoringConfigurationUpdateパラメータを使用します。

アプリケーションログレベルの詳細については、 the section called “アプリケーションモニタリングレベル” を参照してください。

アプリケーションの作成時にアプリケーションのログ記録レベルを設定する

以下のCreateApplicationアクションリクエスト例では、アプリケーションログレベルを INFO に設定しています。

```
{
   "ApplicationName": "MyApplication",
   "ApplicationDescription": "My Application Description",
   "ApplicationConfiguration": {
      "ApplicationCodeConfiguration":{
      "CodeContent":{
        "S3ContentLocation":{
          "BucketARN":"arn:aws:s3:::mybucket",
          "FileKey":"myflink.jar",
          "ObjectVersion":"AbCdEfGhIjKlMnOpQrStUvWxYz12345"
        }
      },
      "CodeContentType":"ZIPFILE"
      },
      "FlinkApplicationConfiguration":
         "MonitoringConfiguration": {
            "ConfigurationType": "CUSTOM",
            "LogLevel": "INFO"
         }
      },
```

```
   "RuntimeEnvironment": "FLINK-1_15",
   "ServiceExecutionRole": "arn:aws:iam::123456789123:role/myrole"
}
```

アプリケーションのログ記録レベルを更新する

以下のUpdateApplicationアクションリクエスト例では、アプリケーションログレベルを INFO に設定しています。

```
{
   "ApplicationConfigurationUpdate": {
      "FlinkApplicationConfigurationUpdate": {
         "MonitoringConfigurationUpdate": {
            "ConfigurationTypeUpdate": "CUSTOM",
            "LogLevelUpdate": "INFO"
         }
      }
   }
}
```

CloudWatch ログストリームに書き込むアクセス許可の追加

Managed Service for Apache Flink には、設定ミスエラーを に書き込むためのアクセス許可が必要です CloudWatch。これらのアクセス許可は、 Managed Service for Apache Flink が引き受ける AWS Identity and Access Management (IAM) ロールに追加できます。

AManaged Service for Apache Flinkに IAM ロールを使用する方法の詳細については、 Amazon Managed Service for Apache Flink のIDとアクセスマネジメント 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 Managed Service for Apache Flink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": "kinesisanlaytics.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
```

```
    }
  ]
}
```

アクセス権限ポリシー

Managed Service for Apache Flink リソース CloudWatch から にログイベントを書き込むアクセス許可をアプリケーションに付与するには、次の IAM アクセス許可ポリシーを使用できます。ロググループとストリームに正しい Amazon リソースネーム (ARN) を指定する

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Stmt0123456789000",
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents",
                "logs:DescribeLogGroups",
                "logs:DescribeLogStreams"
            ],
            "Resource": [
                "arn:aws:logs:us-east-1:123456789012:log-group:my-log-group:log-
stream:my-log-stream*",
                "arn:aws:logs:us-east-1:123456789012:log-group:my-log-group:*",
                "arn:aws:logs:us-east-1:123456789012:log-group:*",
            ]
        }
    ]
}
```

アプリケーションモニタリングレベル

アプリケーションのモニタリングメトリックスレベルとモニタリングログレベルを使用してアプリケーションログメッセージの生成を制御します。

アプリケーションのモニタリンググメトリックレベルは、ログメッセージの細分性を制御します。モニタリングメトリクスのレベルは次のように定義されます。

- アプリケーション：：メトリックの範囲はアプリケーション全体です。
- タスク：メトリックの範囲は各タスクに限定されます。タスクの詳細については、「the section called “スケーリング”」をご参照ください。

- オペレータ：メトリックの範囲は各オペレータに限定されます。演算子についての詳細は、「the section called “DataStream API 演算子”」を参照してください。
- 並列処理：メトリックの範囲はアプリケーションの並列処理に限定されます。このメトリクスレベルは、 UpdateApplication API の MonitoringConfigurationUpdateパラメータを使用してのみ設定できます。コンソールを使用してこのメトリクスレベルを設定することはできません。並列クエリについては、「the section called “スケーリング”」を参照してください。

アプリケーションのモニタリンググログレベルは、アプリケーションのログ詳細度を制御します。モニタリングログレベルは次のように定義されます。

- エラー:アプリケーションで発生可能の壊滅的なイベント
- 警告:アプリケーションの潜在的で有害な状況。
- 情報:アプリケーションの情報提供および一時的な障害イベント。。このログレベルを使用することをお勧めします。
- デバッグ:アプリケーションのデバッグに最も役立つ、きめ細かい情報イベント。注：このレベルは一時的なデバッグ目的でのみ使用してください。

ベストプラクティスのログ記録

アプリケーションには Info ロギングレベルを使用することをおすすめします。Apache Flink エラーを確実に表示するために、このレベルをお勧めします。エラーレベルは Error レベルではなく Info レベルで記録されます。

Debug レベルはアプリケーションの問題を調査する間は一時的にのみ使用することをおすすめします。問題が解決したら、Info レベルに戻してください。Debug ログレベルを使用すると、アプリケーションのパフォーマンスに大きく影響します。

ロギングが多すぎると、アプリケーションのパフォーマンスにも大きな影響を与える可能性があります。例のように処理されたレコードごとにログエントリを書き込まないことをお勧めします。ロギングが多すぎると、データ処理に重大なボトルネックが生じ、ソースからデータを読み取る際にバックプレッシャが発生する可能性があります。

ログ記録のトラブルシューティング

アプリケーションログがログストリームに書き込まれていない場合は、次のことを確認してください。

- アプリケーションのIAM ロールとポリシーが正しいことを確認してください。ログストリームにアクセスするには、アプリケーションのポリシーに以下の権限が必要です。
  - `logs:PutLogEvents`
  - `logs:DescribeLogGroups`
  - `logs:DescribeLogStreams`

  詳細については、「the section called “ CloudWatch ログストリームに書き込むアクセス許可の追加”」を参照してください。
- ステップ 5: アプリケーションが実行されていることを検証する アプリケーションのステータスを確認するには、コンソールでアプリケーションのページを表示するか、 DescribeApplication または ListApplicationsアクションをAPI使用します。
- などの CloudWatch メトリクスをモニタリングdowntimeして、他のアプリケーションの問題を診断します。 CloudWatch メトリクスの読み取りについては、「」を参照してくださいManaged Service for Apache Flink のメトリクスとディメンション。

次のステップ

アプリケーションでログ CloudWatch 記録を有効にしたら、Logs Insights CloudWatch を使用してアプリケーションログを分析できます。詳細については、「the section called “ログの分析”」を参照してください。

CloudWatch Logs Insights を使用したログの分析

前のセクションで説明したようにアプリケーションに CloudWatch ログ記録オプションを追加したら、 CloudWatch Logs Insights を使用してログストリームに特定のイベントやエラーをクエリできます。

CloudWatch Logs Insights を使用すると、Logs CloudWatch でログデータをインタラクティブに検索および分析できます。

CloudWatch Logs Insights の開始方法については、「Logs Insights を使用した CloudWatch ログデータの分析」を参照してください。

サンプルクエリを実行する

このセクションでは、サンプルの CloudWatch Logs Insights クエリを実行する方法について説明します。

前提条件

- Logs で設定された既存のロググループと CloudWatch ログストリーム。
- Logs に保存されている既存の CloudWatch ログ。

AWS CloudTrail、Amazon Route 53、Amazon VPC などのサービスを使用している場合は、これらのサービスからのログを CloudWatch ログに移動するように設定済みである可能性があります。ログを CloudWatch ログに送信する方法の詳細については、「CloudWatch ログの開始方法」を参照してください。

CloudWatch Logs Insights のクエリは、ログイベントから一連のフィールド、またはログイベントに対して実行された数学的集計やその他のオペレーションの結果を返します。このチュートリアルでは、ログイベントのリストを返すクエリを示します。

CloudWatch Logs Insights サンプルクエリを実行するには

1. https://console.aws.amazon.com/cloudwatch/ で CloudWatch コンソールを開きます。
2. ナビゲーションペインで、[Insights] を選択します。
3. [Logs Insights] (ログのインサイト) ページでは、クエリエディタにデフォルトクエリが表示されます。デフォルトでは、最新の 20 件のログイベントが返されます。クエリエディタの上で、クエリを実行する対象のロググループを選択します。

   ロググループを選択すると、 CloudWatch Logs Insights はロググループ内のデータ内のフィールドを自動的に検出し、右側のペインの検出済みフィールドに表示します。また、このロググループのログイベントを時間の経過に従って棒グラフで表示します。この棒グラフは、表に示されるイベントだけでなく、クエリと時間範囲に一致するロググループ内のイベントの分布を示します。
4. [Run query] (クエリの実行) を選択します。

   クエリの結果が表示されます。この例では、タイプを問わず、最新の 20 件のログイベントが結果として表示されます。
5. 返されたログイベントのいずれかについて、すべてのフィールドを表示するには、そのログイベントの左側にある矢印を選択します。

CloudWatch Logs Insights クエリを実行および変更する方法の詳細については、「サンプルクエリを実行および変更する」を参照してください。

クエリの例

このセクションでは、Managed Service for Apache Flink アプリケーションログを分析するための CloudWatch Logs Insights クエリの例を示します。これらのクエリは、いくつかのエラー状態の例を検索して、他のエラー状態を検索するクエリを作成するためのテンプレートとして機能します。

> **Note**
>
> 次のクエリ例のリージョン (*us-west-2*)、アカウント ID (*012345678901*)、アプリケーション名 (*YourApplication*) をアプリケーションのリージョンとアカウント ID に置き換えます。

このトピックには、次のセクションが含まれています。

- 分析オペレーション: タスクの分散
- 分析オペレーション: 並列処理の変更
- エラーの分析: アクセスが拒否されました
- エラーの分析: ソースまたはシンクが見つかりません
- エラーの分析: アプリケーションタスク関連の障害

分析オペレーション: タスクの分散

次の CloudWatch Logs Insights クエリは、Apache Flink Job Manager がタスクマネージャー間で分散するタスクの数を返します。クエリが以前のジョブのタスクを返さないように、クエリの時間枠を 1 回のジョブ実行と一致するように設定する必要があります。並列ロードの詳細については、「スケーリング」をご参照ください。

```
fields @timestamp, message
| filter message like /Deploying/
| parse message " to flink-taskmanager-*" as @tmid
| stats count(*) by @tmid
| sort @timestamp desc
| limit 2000
```

次の CloudWatch Logs Insights クエリは、各タスクマネージャーに割り当てられたサブタスクを返します。サブタスクの総数は、各タスクの並列処理の合計です。タスク並列処理は演算子の並列処理

から派生し、コード内でsetParallelismを指定して変更しない限り、デフォルトではアプリケーションの並列処理と同じです。演算子の並列処理の設定の詳細について、Apache Flink ドキュメントの並列処理の設定:演算子レベルを参照してください。

```
fields @timestamp, @tmid, @subtask
| filter message like /Deploying/
| parse message "Deploying * to flink-taskmanager-*" as @subtask, @tmid
| sort @timestamp desc
| limit 2000
```

タスクスケジューリングについて詳しくは、Apache Flink ドキュメントのジョブとスケジューリングを参照してください。

分析オペレーション: 並列処理の変更

次の CloudWatch Logs Insights クエリは、アプリケーションの並列処理に対する変更 (自動スケーリングなど) を返します。このクエリでは、アプリケーションの並列処理に対する手動による変更も返されます。 Auto Scaling の詳細については、「」を参照してくださいthe section called “Auto Scaling”。

```
fields @timestamp, @parallelism
| filter message like /property: parallelism.default, /
| parse message "default, *" as @parallelism
| sort @timestamp asc
```

エラーの分析: アクセスが拒否されました

次の CloudWatch Logs Insights クエリはAccess Deniedログを返します。

```
fields @timestamp, @message, @messageType
| filter applicationARN like /arn:aws:kinesisanalytics us-
west-2:012345678901:application\/YourApplication/
| filter @message like /AccessDenied/
| sort @timestamp desc
```

エラーの分析: ソースまたはシンクが見つかりません

次の CloudWatch Logs Insights クエリはResourceNotFoundログを返します。 は、Kinesis ソースまたはシンクが見つからない場合に結果をResourceNotFoundログに記録します。

```
fields @timestamp,@message
| filter applicationARN like /arn:aws:kinesisanalytics us-
west-2:012345678901:application\/YourApplication/
| filter @message like /ResourceNotFoundException/
| sort @timestamp desc
```

エラーの分析: アプリケーションタスク関連の障害

次の CloudWatch Logs Insights クエリは、アプリケーションのタスク関連の失敗ログを返します。これらのログは、アプリケーションのステータスがRUNNINGからRESTARTINGに切り替わった場合に生成されます。

```
fields @timestamp,@message
| filter applicationARN like /arn:aws:kinesisanalytics us-
west-2:012345678901:application\/YourApplication/
| filter @message like /switched from RUNNING to RESTARTING/
| sort @timestamp desc
```

Apache Flink バージョン 1.8.2 以前を使用するアプリケーションでは、タスク関連の障害が発生すると、アプリケーションのステータスが代わりにRUNNINGからFAILEDに切り替わります。Apache Flink 1.8.2 以前のバージョンを使用している場合は、次のクエリを使用してアプリケーションタスク関連の障害を検索してください。

```
fields @timestamp,@message
| filter applicationARN like /arn:aws:kinesisanalytics us-
west-2:012345678901:application\/YourApplication/
| filter @message like /switched from RUNNING to FAILED/
| sort @timestamp desc
```

Managed Service for Apache Flink でのメトリクスとディメンションの表示

このトピックには、次のセクションが含まれています。

- アプリケーションメトリクス
- Kinesis Data Streams コネクタメトリクス
- Amazon MSK コネクタメトリクス

- Apache Zeppelin メトリクス
- CloudWatch メトリクスの表示
- CloudWatch メトリクスレポートレベルの設定
- Amazon Managed Service for Apache Flink でのカスタムメトリクスの使用
- Amazon Managed Service for Apache Flink での CloudWatch アラームの使用

Managed Service for Apache Flink がデータソースを処理すると、Managed Service for Apache Flink は次のメトリクスとディメンションを Amazon に報告します CloudWatch。

アプリケーションメトリクス

メトリクス	単位	説明	レベル	使用に関する注意事項	
backPressuredTimeMsPerSecond*	ミリ秒	このタスクまたはオペレーターが1秒あたりにバックプレッシャーを受ける時間(ミリ秒単位)。	タスク、オペレータ、並列度	*Flink バージョン 1.13 を実行しているManaged Service for Apache Flinkアプリケーションでのみ使用できます。 これらのメトリックはアプリケーションのボトルネックを特定することに役立ちます。	
busyTimeMsPerSecond*	ミリ秒	このタスクまたはオペレーターがビジー状態(アイドル	タスク、オペレータ、並列度	*Flink バージョン 1.13 を実行しているManaged Service	

メトリクス	単位	説明	レベル	使用に関する注意事項	
		状態でもバックプレッシャーでもない)の1秒あたりの時間(ミリ秒単位)。値を計算できなかった場合はNaNでもかまいません。		for Apache Flinkアプリケーションでのみ使用できます。 これらのメトリックはアプリケーションのボトルネックを特定することに役立ちます。	
cpuUtilization	割合 (%)	タスクマネージャー全体の CPU 使用率。たとえば、タスクマネージャーが5つある場合、Apache Flink Managed Service for Apache Flink は、レポート間隔ごとにメトリックサンプルを5つ公開します。	アプリケーション	このメトリックスを使用して、アプリケーションの CPU 使用率の最小値、平均値、最大値を監視できます。CPUUtilization メトリクスは、コンテナ内で実行されている TaskManager JVM プロセスの CPU 使用率のみを考慮します。	

メトリクス	単位	説明	レベル	使用に関する注意事項	
containerCPUUtilization	割合 (%)	Flink アプリケーションクラスター内のタスクマネージャーコンテナー全体の CPU 使用率。例えば、5 つのタスクマネージャーがあり、それに応じて 5 つの TaskManager コンテナがあり、Managed Service for Apache Flink は 1 分間のレポート間隔ごとにこのメトリクスの 2 x 5 つのサンプルを発行します。	アプリケーション	コンテナごとに次のように計算されます。 コンテナが消費した合計 CPU 時間 (秒単位) * 100/コンテナの CPU 上限 (CPU/秒) containerCPUUtilization メトリクスは、コンテナ内で実行されている TaskManager JVM プロセスの CPU 使用率のみを考慮します。同じコンテナ内で JVM の外部で実行されているほかのコンポーネントもあります。このcontainerCPUUtilization メトリックにより、コンテナでの CPU の消	

メトリクス	単位	説明	レベル	使用に関する注意事項	
				耗とそれに起因する障害に関するすべてのプロセスを含めて、より包括的な全体像を把握できます。	

メトリクス	単位	説明	レベル	使用に関する注意事項	
containerMemoryUtilization	割合 (%)	Flink アプリケーションクラスター内のタスクマネージャーコンテナー全体のメモリー使用率。例えば、5 つのタスクマネージャーがあり、それに応じて 5 つの TaskManager コンテナがあり、Managed Service for Apache Flink は 1 分間のレポート間隔ごとにこのメトリクスの 2 x 5 つのサンプルを発行します。	アプリケーション	コンテナごとに次のように計算されます。 コンテナのメモリ使用量 (バイト) * 100/ポッドデプロイメント仕様に基づくコンテナのメモリ上限 (バイト単位) HeapMemoryUtilization および ManagedMemoryUtilzations メトリクスは、TaskManager JVM のヒープメモリ使用量やマネージドメモリ (RocksDB ステートバックエンド などのネイティブプロセスの JVM 外のメモリ使用量)	

メトリクス	単位	説明	レベル	使用に関する注意事項	
				などの特定のメモリメトリクスのみを考慮します。このcontainerMemoryUtilization メトリックには、ワーキングセットメモリを含めることで全体像を把握できるため、メモリの総消費量を追跡しやすくなります。使い果たすと、TaskManager ポッドOut of Memory Errorに対して になります。	

メトリクス	単位	説明	レベル	使用に関する注意事項	
containerDiskUtilization	割合 (%)	Flink アプリケーションクラスター内のタスクマネージャーコンテナー全体のディスク使用率。例えば、5 つのタスクマネージャーがあり、それに応じて 5 つの TaskManager コンテナがあり、Managed Service for Apache Flink は 1 分間のレポート間隔ごとにこのメトリクスの 2 x 5 つのサンプルを発行します。	アプリケーション	コンテナごとに次のように計算されます。 ディスク使用量 (バイト) * 100/コンテナのディスク上限 (バイト) コンテナの場合、コンテナのルートボリュームが設定されているファイルシステムの使用率を表します。	

メトリクス	単位	説明	レベル	使用に関する注意事項	
currentInputWatermark	ミリ秒	このアプリケーション、オペレータ、タスク、スレッドが受け取った最後のウォーターマーク	アプリケーション、オペレータ、タスク、並列処理	このレコードは、入力が 2 つあるディメンションでのみ出力されます。これは最後に受信したウォーターマークの最小値です。	
currentOutputWatermark	ミリ秒	このアプリケーション、オペレータ、タスク、スレッドが最後に出力したウォーターマーク	アプリケーション、オペレータ、タスク、並列処理		

メトリクス	単位	説明	レベル	使用に関する注意事項	
downtime	ミリ秒	現在障害または回復中のジョブの場合は、その停止中に経過した時間です。	アプリケーション	この指標は、ジョブが失敗または回復している間に経過した時間を測定します。このメトリックは、実行中のジョブの場合は 0 を返し、完了したジョブの場合は-1を返します。このメトリクスが0または-1でない場合は、アプリケーションのApache Flinkジョブが実行に失敗したことを示します。	

メトリクス	単位	説明	レベル	使用に関する注意事項	
fullRestarts	カウント	このジョブが送信されてから完全に再開された回数の合計です。この指標では、詳細な再起動は測定されません。	アプリケーション	このメトリクスを使用して、アプリケーションの全般的な状態を評価できます。再起動は、Managed Service for Apache Flink の内部メンテナンス中に発生する可能性があります。通常よりも高い速度で再起動される場合は、アプリケーションに問題があることを示している可能性があります。	

メトリクス	単位	説明	レベル	使用に関する注意事項	
heapMemoryUtilization	割合 (%)	タスクマネージャー全体のヒープメモリ使用率。たとえば、タスクマネージャーが5つある場合、Apache Flink Managed Service for Apache Flink は、レポート間隔ごとにメトリックサンプルを5つ公開します。	アプリケーション	このメトリクスを使用して、アプリケーションのヒープメモリ使用率の最小値、平均値、最大値を監視できます。は、TaskManager JVM のヒープメモリ使用量などの特定のメモリメトリクスHeapMemoryUtilization のみを考慮します。	

メトリクス	単位	説明	レベル	使用に関する注意事項	
idleTimeMsPerSecond*	ミリ秒	このタスクまたはオペレータが 1 秒あたりにアイドル状態 (処理するデータがない) の時間 (ミリ秒単位)。アイドル時間にはバックプレッシャーの時間は含まれないため、タスクにバックプレッシャーがかかっている場合、そのタスクはアイドルではありません。	タスク、オペレータ、並列度	*Flink バージョン 1.13 を実行しているManaged Service for Apache Flinkアプリケーションでのみ使用できます。 これらのメトリックはアプリケーションのボトルネックを特定することに役立ちます。	

メトリクス	単位	説明	レベル	使用に関する注意事項	
lastCheckpointSize	バイト	最後のチェックポイントの合計サイズ	アプリケーション	このメトリックを使用して、実行中のアプリケーションストレージの使用率を判断できます。 このメトリックの値が増加している場合は、メモリリークやボトルネックなど、アプリケーションに問題がある可能性があります。	

メトリクス	単位	説明	レベル	使用に関する注意事項	
lastCheckpointDuration	ミリ秒	最後のチェックポイントを完了するまでにかかった時間	アプリケーション	このメトリックは、最新のチェックポイントを完了するまでにかかった時間を測定します。このメトリックの値が増加している場合は、メモリリークやボトルネックなど、アプリケーションに問題がある可能性があります。場合によっては、チェックポイントを無効にすることでこの問題を解決できます。	

メトリクス	単位	説明	レベル	使用に関する注意事項	
managedMemoryUsed*	バイト	現在使用中のメモリの量。	アプリケーション、オペレータ、タスク、並列処理	*Flink バージョン 1.13 を実行しているManaged Service for Apache Flinkアプリケーションでのみ使用できます。 これは Flink が Java ヒープ外で管理するメモリに関するものです。RocksDB のステートバックエンドに使用され、アプリケーションでも利用できます。	

メトリクス	単位	説明	レベル	使用に関する注意事項	
managedMemoryTotal *	バイト	メモリの合計量。	アプリケーション、オペレータ、タスク、並列処理	*Flink バージョン 1.13 を実行しているManaged Service for Apache Flinkアプリケーションでのみ使用できます。 これは Flink が Java ヒープ外で管理するメモリに関するものです。RocksDB のステートバックエンドに使用され、アプリケーションでも利用できます。この ManagedMemoryUtilzations メトリックは、Managed Memory (RocksDB State Backendのようなネイティブプロセスの	

メトリクス	単位	説明	レベル	使用に関する注意事項	
				JVM 外のメモリ使用量) のような特定のメモリーメトリクスのみを考慮します。	
managedMemoryUtilization*	割合 (%)	managedMemoryUsed/ によって導出managedMemoryTotal	アプリケーション、オペレータ、タスク、並列処理	*Flink バージョン 1.13 を実行しているManaged Service for Apache Flinkアプリケーションでのみ使用できます。 これは Flink が Java ヒープ外で管理するメモリに関するものです。RocksDB のステートバックエンドに使用され、アプリケーションでも利用できます。	

メトリクス	単位	説明	レベル	使用に関する注意事項	
numberOfFailedCheckpoints	カウント	チェックポイントが失敗した回数。	アプリケーション	このメトリックを使用して、アプリケーションの状態と進行状況を監視できます。スループットや権限の問題など、アプリケーションの問題が原因でチェックポイントが失敗することがあります。	

メトリクス	単位	説明	レベル	使用に関する注意事項	
numRecordsIn*	カウント	このアプリケーション、オペレータ、またはタスクが受信したレコードの総数。	アプリケーション、オペレータ、タスク、並列処理	*一定期間 (秒/分) にわたって SUM 統計を適用するには: • 正しいレベルのメトリクスを選択してください。オペレーターのメトリクスを追跡している場合は、対応するオペレーターメトリクスを選択する必要があります。 • Managed Service for Apache Flink では 1 分あたり 4 つのメトリックスナップショットが作成されるため、m1/4 というメトリック計算を使用する必要があり	

メトリクス	単位	説明	レベル	使用に関する注意事項	
				ます。ここで m1 は、一定期間(秒/分)にわたるSUM統計です。 メトリックのレベルは、このメトリックがアプリケーション全体、特定のオペレータ、または特定のタスクが受信したレコードの総数を測定するかどうかを指定します。	

メトリクス	単位	説明	レベル	使用に関する注意事項	
numRecordsInPerSecond*	Count/Second	このアプリケーション、オペレータ、またはタスクが 1 秒あたりに受信したレコードの総数です。	アプリケーション、オペレータ、タスク、並列処理	*一定期間 (秒/分) にわたって SUM 統計を適用するには: • 正しいレベルのメトリクスを選択してください。オペレーターのメトリクスを追跡している場合は、対応するオペレーターメトリクスを選択する必要があります。 • Managed Service for Apache Flink では 1 分あたり 4 つのメトリックスナップショットが作成されるため、m1/4 というメトリック計算を使用する必要があり	

メトリクス	単位	説明	レベル	使用に関する注意事項	
				ます。ここで m1 は、一定期間 (秒/分) にわたる SUM 統計です。 メトリックのレベルは、このメトリックがアプリケーション全体、特定のオペレータ、または特定のタスクが 1 秒あたりに受信したレコードの総数を測定するかどうかを指定します。	

メトリクス	単位	説明	レベル	使用に関する注意事項	
numRecordsOut*	カウント	このアプリケーション、オペレータ、またはタスクが送信したレコードの総数。	アプリケーション、オペレータ、タスク、並列処理	*一定期間 (秒/分) にわたって SUM 統計を適用するには: • 正しいレベルのメトリクスを選択してください。オペレーターのメトリクスを追跡している場合は、対応するオペレーターメトリクスを選択する必要があります。 • Managed Service for Apache Flink では 1 分あたり 4 つのメトリックスナップショットが作成されるため、m1/4 というメトリック計算を使用する必要があり	

メトリクス	単位	説明	レベル	使用に関する注意事項	
				ます。ここで m1 は、一定期間 (秒/分) にわたる SUM 統計です。 メトリックのレベルは、このメトリックがアプリケーション全体、特定のオペレータ、または特定のタスクが発行したレコードの総数を測定するかどうかを指定します。	

メトリクス	単位	説明	レベル	使用に関する注意事項	
numLateRecordsDropped*	カウント	アプリケーション、オペレータ、タスク、並列処理		*一定期間 (秒/分) にわたって SUM 統計を適用するには: • 正しいレベルのメトリクスを選択してください。オペレーターのメトリクスを追跡している場合は、対応するオペレーターメトリクスを選択する必要があります。 • Managed Service for Apache Flink では 1 分あたり 4 つのメトリックスナップショットが作成されるため、m1/4 というメトリック計算を使用する必要があり	

メトリクス	単位	説明	レベル	使用に関する注意事項	
				ます。ここで m1 は、一定期間 (秒/分) にわたる SUM 統計です。 このオペレータまたはタスクが遅れたために減少したレコードの数。	

メトリクス	単位	説明	レベル	使用に関する注意事項	
numRecordsOutPerSecond*	Count/Second	このアプリケーション、オペレータ、またはタスクが 1 秒あたりに送信したレコードの総数。	アプリケーション、オペレータ、タスク、並列処理	*一定期間 (秒/分) にわたって SUM 統計を適用するには: • 正しいレベルのメトリクスを選択してください。オペレーターのメトリクスを追跡している場合は、対応するオペレーターメトリクスを選択する必要があります。 • Managed Service for Apache Flink では 1 分あたり 4 つのメトリックスナップショットが作成されるため、m1/4 というメトリック計算を使用する必要があり	

メトリクス	単位	説明	レベル	使用に関する注意事項	
				ます。ここで m1 は、一定期間 (秒/分) にわたる SUM 統計です。 メトリックのレベルは、このメトリックがアプリケーション全体、特定のオペレータ、または特定のタスクが 1 秒あたりに送信したレコードの総数を測定するかどうかを指定します。	
oldGenerationGCCount	カウント	すべてのタスクマネージャーで発生した古いガベージコレクション操作の総数。	アプリケーション		

メトリクス	単位	説明	レベル	使用に関する注意事項	
oldGenerationGCTime	ミリ秒	古いガベージコレクション操作の実行にかかった合計時間。	アプリケーション	このメトリックを使用して、ガベージコレクションの合計時間、平均時間、最大時間を監視できます。	
threadCount	カウント	アプリケーションが使用したライブスレッドの総数。	アプリケーション	このメトリックは、アプリケーションコードが使用するスレッド数を測定します。これはアプリケーションの並列処理とは異なります。	
uptime	ミリ秒	ジョブが中断されずに実行された時間。	アプリケーション	この指標を使用して、ジョブが正常に実行されているかどうかを判断できます。このメトリックは、完了したジョブに対して -1 を返します。	

メトリクス	単位	説明	レベル	使用に関する注意事項	
KPUs*	カウント	アプリケーションで使用される KPUsの合計数。	アプリケーション	*このメトリクスは、請求期間 (1 時間) ごとに 1 つのサンプルを受け取ります。時間の経過に伴う KPUs の数を視覚化するには、少なくとも 1 (1) 時間にわたって MAX または AVG を使用します。 KPU 数には KPU orchestration が含まれます。詳細については、「Managed Service for Apache Flink の料金」を参照してください。	

Kinesis Data Streams コネクタメトリクス

AWS は、以下に加えて、Kinesis Data Streams のすべてのレコードを発行します。

メトリクス	単位	説明	レベル	使用に関する注意事項
millisbehindLatest	ミリ秒	コンシューマーがストリームの先頭から遅れているミリ秒数は、コンシューマーが現在時刻からどれだけ遅れているかを示します。	アプリケーション (ストリーム用)、並列処理 (用 ShardId)	• 値ゼロはレコード処理が追いついて、現在処理する新しいレコードは存在しないことを示します。特定のシャードのメトリックは、ストリーム名とシャード ID で指定できます。 • 値が -1 の場合は、サービスがメトリックの値をまだ報告していないことを示します。
bytesRequestedPerFetch	バイト	getRecords への1回の呼び出しで要求されたバイト数。	アプリケーション (ストリーム用)、並列処理 (用 ShardId)	

Amazon MSK コネクタメトリクス

AWS は、以下に加えて、Amazon MSK のすべてのレコードを発行します。

メトリクス	単位	説明	レベル	使用に関する注意事項
currentoffsets	該当なし	各パーティションのコンシューマーの現在の読み取りオフセット。特定のパーティションのメトリックは、トピック名とパーティション ID で指定できます。	アプリケーション (トピック用)、並列処理 (用 PartitionId)	
commitsFailed	該当なし	オフセットのコミットとチェックポイントが有効になっている場合、Kafka へのオフセット コミットの失敗の合計数	アプリケーション、オペレータ、タスク、並列処理	オフセットを Kafka にコミットすることは、コンシューマの進行状況を公開するための手段にすぎないため、コミットの失敗は Flink のチェックポイントが設定されたパーティション オフセットの完全性に影響しません。
commitsSucceeded	該当なし	オフセットのコミットとチェックポイント設定が有効な場合、Kafka へのオフセット コミッ	アプリケーション、オペレータ、タスク、並列処理	

メトリクス	単位	説明	レベル	使用に関する注意事項
		トが成功した合計数。		
committed offsets	該当なし	最後に正常にコミットされたオフセットは、パーティションごとに Kafka に送信されます。特定のパーティションのメトリックは、トピック名とパーティション ID で指定できます。	アプリケーション (トピック用)、並列処理 (用 PartitionId)	
records_lag_max	カウント	このウィンドウ内の任意のパーティションのレコード数に関する最大ラグ	アプリケーション、オペレータ、タスク、並列処理	
bytes_consumed_rate	バイト	トピック用に消費された1秒あたりの平均バイト数	アプリケーション、オペレータ、タスク、並列処理	

Apache Zeppelin メトリクス

Studio ノートブックの場合、 はアプリケーションレベルで次のメトリクスを AWS 出力します: KPUs、cpuUtilizationheapMemoryUtilization、、oldGenerationGCTime、oldGenerationGC および threadCount。さらに、アプリケーションレベルで次の表に示すようなメトリクスを出力します。

メトリクス	単位	説明	Prometheus 名
zeppelinCpuUtilization	割合 (%)	Apache Zeppelin サーバーの CPU 使用率の全体的パーセンテージ。	process_cpu_usage
zeppelinHeapMemoryUtilization	割合 (%)	Apache Zeppelin サーバーのヒープメモリ使用率の全体的パーセンテージ。	jvm_memory_used_bytes
zeppelinThreadCount	カウント	Apache Zeppelin サーバーが使用しているライブスレッドの総数。	jvm_threads_live_threads
zeppelinWaitingJobs	カウント	キューに入っていて 1 つのスレッドを待っている Apache Zeppelin ジョブの数。	jetty_threads_jobs
zeppelinServerUptime	[秒]	サーバーが稼働していた合計時間。	process_uptime_seconds

CloudWatch メトリクスの表示

Amazon CloudWatch コンソールまたは を使用して、アプリケーションの CloudWatch メトリクスを表示できます AWS CLI。

CloudWatch コンソールを使用してメトリクスを表示するには

1. https://console.aws.amazon.com/cloudwatch/ で CloudWatch コンソールを開きます。
2. ナビゲーションペインで メトリクスを選択します。
3. Managed Service for Apache Flink のCloudWatch 「カテゴリ別のメトリクス」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
```

CloudWatch メトリクスレポートレベルの設定

アプリケーションが作成するアプリケーションメトリクスのレベルを制御できます。Managed Service for Apache Flink は、以下のメトリクスレベルをサポートしています。

- アプリケーション：アプリケーションは各アプリケーションの最高レベルのメトリックのみを報告します。Managed Service for Apache Flinkのメトリックは、デフォルトではアプリケーションレベルで公開されます。
- タスク:アプリケーションは、1秒あたりにアプリケーションに出入りするレコード数など、タスクメトリックレポートレベルで定義されたメトリックについて、タスク固有のメトリックディメンションをレポートします。
- オペレータ:アプリケーションは、オペレータメトリックレポートレベルで定義されたメトリック（各フィルタまたはマップ操作のメトリックスなど）について、オペレータ固有のメトリックディメンションをレポートします。
- 並列処理:アプリケーションは各実行スレッドのレポートについて `Task` レベルと `Operator` レベルメトリクスを報告します。このレポートレベルはコストがかかりすぎたため、並列処理が64を超えるアプリケーションには推薦されません。

> **Note**
>
> サービスが生成するメトリックデータの量が多いため、このメトリックレベルはトラブルシューティングにのみ使用する必要があります。このメトリック レベルはCLIでのみ設定できます。このメトリックレベルはコンソールでは利用できない。

デフォルトレベルはアプリケーションです。アプリケーションは現在のレベルとすべてのそれより高いレベルのメトリクスを報告します。たとえば、レポートレベルがオペレーターに設定されている場合、アプリケーションはアプリケーション、タスク、オペレーターのメトリックをレポートします。

CloudWatch メトリクスレポートレベルは、 CreateApplicationアクションの MonitoringConfigurationパラメータまたは UpdateApplicationアクションの MonitoringConfigurationUpdateパラメータを使用して設定します。次の UpdateApplicationアクションのリクエスト例では、 CloudWatch メトリクスレポートレベルをタスク に設定します。

```
{
   "ApplicationName": "MyApplication",
   "CurrentApplicationVersionId": 4,
   "ApplicationConfigurationUpdate": {
      "FlinkApplicationConfigurationUpdate": {
         "MonitoringConfigurationUpdate": {
            "ConfigurationTypeUpdate": "CUSTOM",
            "MetricsLevelUpdate": "TASK"
         }
      }
   }
}
```

CreateApplicationアクションのLogLevelパラメータまたはUpdateApplicationアクションのLogLevelUpdateパラメータでロギングレベルを設定することができます。次のログレベルから選択できます。

- ERROR: 回復可能性のあるエラーイベントをログに記録します。
- WARN: エラーの原因となる可能性のある警告イベントをログに記録します。
- INFO： 情報イベントをログに記録します。
- DEBUG： 一般的なデバッグイベントをログに記録します。

Log4j のロギングレベルの詳細については、Apache Log4j ドキュメントのカスタムログレベルを参照してください。

Amazon Managed Service for Apache Flink でのカスタムメトリクスの使用

Managed Service for Apache Flink は、リソースの使用状況とスループットのメトリクスを含め CloudWatch、19 のメトリクスを に公開します。さらに、イベントの処理や外部リソースへのアクセスなど、アプリケーション固有のデータを追跡するための独自のメトリクスを作成できます。

このトピックには、次のセクションが含まれています。

- 仕組み
- 例
- カスタムメトリクスの表示

仕組み

Managed Service for Apache Flinkのカスタムメトリクスは Apache Flink メトリックシステムを使用します。Managed Service Flink メトリクスには、以下の属性を持っています。

- タイプ:メトリックのタイプは、データをどのように測定することを説明して報告します。Apache Flink メトリックのタイプには、カウント、ゲージ、ヒストグラム、メーターなどがあります。Apache Flink メトリクスタイプの詳細について、メトリクスタイプを参照してください。

  > **Note**
  >
  > AWS CloudWatch メトリクスは、ヒストグラム Apache Flink メトリクスタイプをサポートしていません。 CloudWatch は、カウント、ゲージ、およびメータタイプの Apache Flink メトリクスのみを表示できます。

- スコープ： メトリクスのスコープは、識別子と、メトリクスが にどのように報告されるかを示すキーと値のペアのセットで構成されます CloudWatch。メトリクスの ID は次で構成されます。
  - メトリクスが報告されるレベルを示すシステムスコープ (例：オペレーター)。
  - ユーザー変数やメトリックグループ名などの属性を定義するユーザースコープ。これらの属性は`MetricGroup.addGroup(key, value)`または`MetricGroup.addGroup(name)`によって定義されます。

  スコープの詳細については、スコップを参照してください。

Apache Flink メトリクスの詳細については、Apache Flink ドキュメントのメトリックを参照してください。

Managed Service for Apache Flinkでカスタムメトリクスを作成するには、RichFunctionを拡張する任意のユーザー関数からGetMetricGroupを呼び出して、Apache Flink メトリックシステムにアクセスできます。このメソッドは、カスタムメトリクスの作成と登録に使用できるMetricGroupオブジェクトを返します。Managed Service for Apache Flink は、グループキーを使用して作成されたすべてのメトリクスを KinesisAnalyticsにレポートします CloudWatch。定義したカスタムメトリクスには、次の特徴があります。

- カスタムメトリックスにはメトリクス名とグループ名があります。これらの名前は英数字で構成されます。
- ユーザースコープで定義した属性 (KinesisAnalyticsメトリクスグループを除く) は CloudWatch ディメンションとして公開されます。
- カスタムメトリックスはデフォルトで Application レベルで公開されます。
- アプリケーションの監視レベルに基づいて、ディメンション (タスク/オペレーター/並列処理) がメトリックスに追加されます。アプリケーションのモニタリングレベルは、 CreateApplicationアクションの MonitoringConfigurationパラメータ、または UpdateApplicationアクションの または MonitoringConfigurationUpdateパラメータを使用して設定します。

例

以下のコード例は、カスタムメトリックを作成および増加させるマッピングクラスを作成する方法と、マッピングクラスをDataStreamオブジェクトに追加してアプリケーションに実装する方法を示しています。

レコード数カスタムメトリクス

以下のコード例は、データストリーム内のレコードをカウントするメトリクス (numRecordsInメトリクスと同じ機能)を作成するマッピングクラスの作成方法を示しています。

```
    private static class NoOpMapperFunction extends RichMapFunction<String, String> {
        private transient int valueToExpose = 0;
        private final String customMetricName;

        public NoOpMapperFunction(final String customMetricName) {
            this.customMetricName = customMetricName;
        }

        @Override
        public void open(Configuration config) {
            getRuntimeContext().getMetricGroup()
```

```
                    .addGroup("KinesisAnalytics")
                    .addGroup("Program", "RecordCountApplication")
                    .addGroup("NoOpMapperFunction")
                    .gauge(customMetricName, (Gauge<Integer>) () -> valueToExpose);
        }

        @Override
        public String map(String value) throws Exception {
            valueToExpose++;
            return value;
        }
    }
```

前の例で、アプリケーションが処理するレコードごとにvalueToExpose変数はインクリメントされます。

マッピングクラスを定義したら、マップを実装するアプリケーション内ストリームを作成します。

```
DataStream<String> noopMapperFunctionAfterFilter =
    kinesisProcessed.map(new NoOpMapperFunction("FilteredRecords"));
```

このアプリケーションの完全なコードについては、レコードカウント:カスタムメトリックアプリケーションを参照してください。

単語数カスタムメトリクス

次のコード例は、データストリーム内の単語数をカウントするメトリクスを作成するマッピングクラスを作成する方法を示しています。

```
private static final class Tokenizer extends RichFlatMapFunction<String, Tuple2<String,
 Integer>> {

            private transient Counter counter;

            @Override
            public void open(Configuration config) {
                this.counter = getRuntimeContext().getMetricGroup()
                        .addGroup("KinesisAnalytics")
                        .addGroup("Service", "WordCountApplication")
                        .addGroup("Tokenizer")
                        .counter("TotalWords");
            }
```

```
            @Override
            public void flatMap(String value, Collector<Tuple2<String, Integer>>out) {
                // normalize and split the line
                String[] tokens = value.toLowerCase().split("\\W+");

                // emit the pairs
                for (String token : tokens) {
                    if (token.length() > 0) {
                        counter.inc();
                        out.collect(new Tuple2<>(token, 1));
                    }
                }
            }
        }
```

前の例では、アプリケーションが処理する単語ごとにcounter変数はインクリメントされます。

マッピングクラスを定義したら、マップを実装するアプリケーション内ストリームを作成します。

```
// Split up the lines in pairs (2-tuples) containing: (word,1), and
// group by the tuple field "0" and sum up tuple field "1"
DataStream<Tuple2<String, Integer>> wordCountStream = input.flatMap(new
 Tokenizer()).keyBy(0).sum(1);

// Serialize the tuple to string format, and publish the output to kinesis sink
wordCountStream.map(tuple -> tuple.toString()).addSink(createSinkFromStaticConfig());
```

このアプリケーションの完全なコードについては、ワードカウント:カスタムメトリックを参照してください。

カスタムメトリクスの表示

アプリケーションのカスタムメトリクスは、AWS/KinesisAnalyticsダッシュボードの CloudWatch メトリクスコンソールのアプリケーションメトリクスグループの下に表示されます。

Amazon Managed Service for Apache Flink での CloudWatch アラームの使用

Amazon CloudWatch メトリクスアラームを使用すると、指定した期間にわたって CloudWatch メトリクスを監視できます。アラームは、複数の期間にわたる閾値に対するメトリックまたはメートルの値に基づいて、1つまたは複数のアクションを実行します。例えば、アラームは Amazon Simple Notification Service (Amazon SNS) トピックに通知を送信します。

CloudWatch アラームの詳細については、「Amazon CloudWatch アラームの使用」を参照してください。

推薦アラーム

このセクションには、Managed Service for Apache Flinkアプリケーションをモニタリングするための推薦アラームが含まれています。

この表には推奨されるアラームが説明されており、次のセクションがあります。

- メトリック表現：しきい値に対してテストするメトリックまたはメトリック式。
- 統計:メトリックのチェックに使用される統計。たとえば、平均です。
- しきい値：このアラームを使用するには、期待されるアプリケーションパフォーマンスの上限を定義するしきい値を決定する必要があります。このしきい値は、通常の状態でアプリケーションを監視して決定する必要があります。
- 説明：このアラームをトリガーする可能性のある原因と、この状態に対して考えられる解決方法。

メトリクス式	統計)	Threshold	説明
`downtime > 0`	[Average] (平均)	0	ダウンタイムが 0 より大きい場合は、アプリケーションが失敗したことを示します。値が0より大きい場合、アプリケーションはデータを処理していません。すべてのアプリケーションに推奨されます。`Downtime` メトリクスは、停止期間を測定します。ダウンタイムが 0 より大きい場合は、アプリケーションが失敗したことを示します。トラブルシューティングについては、「」を参照して

メトリクス式	統計)	Threshold	説明
			ください[アプリケーションを再起動中]。

メトリクス式	統計)	Threshold	説明
RATE（numberOfFailedCheckpoints） > 0	[Average] (平均)	0	このメトリクスは、アプリケーションが起動してから失敗したチェックポイントの数をカウントします。アプリケーションによっては、チェックポイントが失敗することがあっても許容できる場合があります。ただし、チェックポイントが定期的に失敗する場合は、アプリケーションが異常である可能性が高く、注意深く見ることが必要です。絶対値ではなく勾配でアラームを発するようにRATE(numberOfFailedチェックポイント)をモニタリングすることをお勧めします。すべてのアプリケーションに推奨されます。このメトリクスを使用して、アプリケーションのヘルスとチェックポイントの進行状況をモニタリングします。アプリケーションは、状態データが正常であればチェックポイントに保存します。アプリケーションが入力デー

メトリクス式	統計)	Threshold	説明
			タの処理を進めていない場合、タイムアウトによりチェックポイントが失敗する可能性があります。トラブルシューティングについては、「」を参照してくださいチェックポイントがタイムアウトしています。。
`Operator.numRecordsOutPerSecond` < しきい値	[Average] (平均)	通常の状態でアプリケーションから出力されるレコードの最小数。	すべてのアプリケーションに推奨されます。このしきい値を下回ると、アプリケーションが入力データに対して予想される進行を行っていない可能性があります。トラブルシューティングについては、「」を参照してくださいスループットが遅すぎる。

メトリクス式	統計)	Threshold	説明
`records_lag_max|millisbehindLatest` > しきい値	最大値	通常の状態で予想される最大レイテンシー。	アプリケーションが Kinesis または Kafka から消費している場合、これらのメトリクスは、アプリケーションが遅れているかどうかを示し、現在の負荷に対応するためにスケーリングする必要があります。これは、あらゆる種類のアプリケーションで追跡しやすい汎用的な指標です。しかし、これはリアクティブスケーリング、つまりアプリケーションがすでに遅れている場合にのみ使用できます。すべてのアプリケーションに推奨されます。Kafka ソースには `records_lag_max` メトリクスを使用し、Kinesis ストリームソースには `millisbehindLatest` を使用します。このしきい値を超えると、アプリケーションが入力データに対して予想される進行を行っていない可能性があります。トラブルシューティングに

メトリクス式	統計)	Threshold	説明
			ついては、「」を参照してくださいスループットが遅すぎる。

メトリクス式	統計)	Threshold	説明
lastCheckpointDuration > しきい値	最大値	通常の条件下で予想されるチェックポイントの最大期間。	状態に保存されるデータの量とチェックポイントにかかる時間を監視します。チェックポイントが増えたり、時間がかかったりしても、アプリケーションはチェックポイント処理に時間を費やし続けて、実際の処理のサイクルは少なくなります。場合によっては、チェックポイントが増えすぎたり、時間がかかりすぎて失敗したりすることがあります。絶対値を監視することに加えて、顧客はRATE(lastCheckpointSize) とRATE(lastCheckpointDuration) による変化率の監視を検討する必要もあります。がlastCheckpointDuration 継続的に増加する場合、このしきい値を超えると、アプリケーションが入力データに対して予期された進行を行っていないか、バックプレッシャーなどのアプ

メトリクス式	統計)	Threshold	説明
			リケーションの状態に問題がある可能性があります。トラブルシューティングについては、「」を参照してください州の無限の成長。

メトリクス式	統計)	Threshold	説明
lastCheckpointSize > しきい値	最大値	通常の条件下で予想されるチェックポイントの最大サイズ。	状態に保存されるデータの量とチェックポイントにかかる時間を監視します。チェックポイントが増えたり、時間がかかったりしても、アプリケーションはチェックポイント処理に時間を費やし続けて、実際の処理のサイクルは少なくなります。場合によっては、チェックポイントが増えすぎたり、時間がかかりすぎて失敗したりすることがあります。絶対値を監視することに加えて、顧客はRATE(lastCheckpointSize) とRATE(lastCheckpointDuration) による変化率の監視を検討する必要もありますが。がlastCheckpointSize 継続的に増加する場合、このしきい値を超えると、アプリケーションが状態データを蓄積している可能性があります。状態データが大きすぎると、チェックポイ

メトリクス式	統計)	Threshold	説明
			ントからの復旧時にアプリケーションがメモリ不足になったり、チェックポイントからの復旧に時間がかかりすぎる可能性があります。トラブルシューティングについては、「」を参照してください州の無限の成長。
heapMemoryUtilization > しきい値	最大値	これにより、アプリケーションの全体的なリソース使用率がわかり、アプリケーションが I/O バインドされていない限り、プロアクティブスケーリングに使用できます。通常の条件下で予想される最大heapMemoryUtilization サイズ。推奨値は 90% です。	このメトリクスを使用して、アプリケーション全体のタスクマネージャーの最大メモリ使用率をモニタリングできます。アプリケーションがこのしきい値に達した場合は、より多くのリソースをプロビジョニングする必要があります。これを行うには、自動スケーリングを有効にするか、アプリケーションの並列処理を増やします。リソースの増加の詳細については、「」を参照してくださいスケーリング。

メトリクス式	統計)	Threshold	説明
cpuUtilization > しきい値	最大値	これにより、アプリケーションの全体的なリソース使用率がわかり、アプリケーションが I/O バインドされていない限り、プロアクティブスケーリングに使用できます。通常の条件下で予想される最大cpuUtilization サイズで、推奨値は 80% です。	このメトリクスを使用して、アプリケーション全体のタスクマネージャーの最大 CPU 使用率をモニタリングできます。アプリケーションがこのしきい値に達した場合は、より多くのリソースをプロビジョニングする必要があります。これを行うには、自動スケーリングを有効にするか、アプリケーションの並列処理を増やします。リソースの増加の詳細については、「」を参照してくださいスケーリング。
threadsCount > しきい値	最大値	通常の条件下で予想される最大threadsCount サイズ。	このメトリクスを使用して、アプリケーション全体のタスクマネージャーでスレッドリークを監視できます。このメトリクスがこのしきい値に達した場合は、アプリケーションコードでスレッドが閉じられていないかを確認してください。

メトリクス式	統計)	Threshold	説明
(oldGarbageCollectionTime * 100)/60_000 over 1 min period') > しきい値	最大値	予想される最大oldGarbageCollectionTime 期間。一般的なガベージコレクション時間が指定されたしきい値の 60% になるようにしきい値を設定することをお勧めしますが、アプリケーションの正しいしきい値は異なります。	このメトリクスが継続的に増加している場合は、アプリケーション全体のタスクマネージャーにメモリリークがあることを示している可能性があります。
RATE(oldGarbageCollectionCount) > しきい値	最大値	oldGarbageCollectionCount 通常の条件下で予想される最大値。アプリケーションの正しいしきい値は異なります。	このメトリクスが継続的に増加している場合は、アプリケーション全体のタスクマネージャーにメモリリークがあることを示している可能性があります。
Operator.currentOutputWatermark - Operator.currentInputWatermark > しきい値	最小値	通常の条件下で予想される最小透かしインクリメント。アプリケーションの正しいしきい値は異なります。	このメトリクスが継続的に増加している場合、アプリケーションがますます古いイベントを処理しているか、アップストリームサブタスクがますます長い時間ウォーターマークを送信していないことを示している可能性があります。

CloudWatch ログへのカスタムメッセージの書き込み

Managed Service for Apache Flink アプリケーションの CloudWatch ログにカスタムメッセージを書き込むことができます。Apache log4j ライブラリまたは Simple Logging Facade for Java (SLF4J) ライブラリを使用します。

トピック

- Log4J を使用して CloudWatch ログに書き込む
- SLF4J を使用して CloudWatch ログに書き込む

Log4J を使用して CloudWatch ログに書き込む

1. 次の依存関係をアプリケーションの pom.xml ファイルに追加します。

```
<dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-api</artifactId>
    <version>2.6.1</version>
</dependency>
<dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-core</artifactId>
    <version>2.6.1</version>
</dependency>
```

2. ライブラリからオブジェクトを含めます。

```
import org.apache.logging.log4j.Logger;
```

3. Logger オブジェクトをインスタンス化し、次のアプリケーションクラスを渡します。

```
private static final Logger log =
 LogManager.getLogger.getLogger(YourApplicationClass.class);
```

4. log.info を使用してログに書き込みます。アプリケーションログには、多数のメッセージが書き込まれます。カスタムメッセージをフィルタリングしやすくするには、INFO アプリケーションログレベルを使用してください。

```
log.info("This message will be written to the application's CloudWatch log");
```

アプリケーションは、次のようなメッセージを含むレコードをログに書き込みます。

```
{
  "locationInformation": "com.amazonaws.services.managed-
flink.StreamingJob.main(StreamingJob.java:95)",
  "logger": "com.amazonaws.services.managed-flink.StreamingJob",
  "message": "This message will be written to the application's CloudWatch log",
  "threadName": "Flink-DispatcherRestEndpoint-thread-2",
  "applicationARN": "arn:aws:kinesisanalyticsus-east-1:123456789012:application/test",
  "applicationVersionId": "1", "messageSchemaVersion": "1",
  "messageType": "INFO"
}
```

SLF4J を使用して CloudWatch ログに書き込む

1. 次の依存関係をアプリケーション pom.xml ファイルに追加します。

   ```
   <dependency>
       <groupId>org.slf4j</groupId>
       <artifactId>slf4j-log4j12</artifactId>
       <version>1.7.7</version>
       <scope>runtime</scope>
   </dependency>
   ```

2. ライブラリからのオブジェクトを含めます。

   ```
   import org.slf4j.Logger;
   import org.slf4j.LoggerFactory;
   ```

3. Loggerオブジェクトをインスタンス化し、次のアプリケーションクラスを渡します。

   ```
   private static final Logger log =
    LoggerFactory.getLogger(YourApplicationClass.class);
   ```

4. log.infoを使用してログに書き込みます。アプリケーションログには、多数のメッセージが書き込まれます。カスタムメッセージをフィルタリングしやすくするには、INFOアプリケーションログレベルを使用してください。

   ```
   log.info("This message will be written to the application's CloudWatch log");
   ```

アプリケーションは、次のようなメッセージを含むレコードをログに書き込みます。

```
{
  "locationInformation": "com.amazonaws.services.managed-
flink.StreamingJob.main(StreamingJob.java:95)",
  "logger": "com.amazonaws.services.managed-flink.StreamingJob",
  "message": "This message will be written to the application's CloudWatch log",
  "threadName": "Flink-DispatcherRestEndpoint-thread-2",
  "applicationARN": "arn:aws:kinesisanalyticsus-east-1:123456789012:application/test",
  "applicationVersionId": "1", "messageSchemaVersion": "1",
  "messageType": "INFO"
}
```

を使用した Managed Service for Apache Flink API コールのログ記録 AWS CloudTrail

Managed Service for Apache Flink は AWS CloudTrail、Managed Service for Apache Flink のユーザー、ロール、または AWS 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。 は、Managed Service for Apache Flink のすべての API コールをイベントとして CloudTrail キャプチャします。キャプチャーされた呼び出しには、Managed Service for Apache Flink・コンソールの呼び出しと、Managed Service for Apache FlinkAPIオペレーションへのコード呼び出しが含まれます。証跡を作成する場合は、Managed Service for Apache Flink の CloudTrail イベントなど、Amazon S3 バケットへの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。で収集された情報を使用して CloudTrail、Managed Service for Apache Flink に対するリクエスト、リクエスト元の IP アドレス、リクエスト者、リクエスト日時などの詳細を確認できます。

の詳細については CloudTrail、「 AWS CloudTrail ユーザーガイド」を参照してください。

の Managed Service for Apache Flink 情報 CloudTrail

CloudTrail AWS アカウントを作成すると、 がアカウントで有効になります。Managed Service for Apache Flink でアクティビティが発生すると、そのアクティビティは CloudTrail イベント履歴 の他の AWS サービスイベントとともに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、「イベント履歴を使用した CloudTrail イベントの表示」を参照してください。

Managed Service for Apache Flink のイベントなど、 AWS アカウント内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、証跡はすべての AWS リージョンに適用されます。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口グに記録し、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 口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 サポートされているサービスと統合
- の Amazon SNS 通知の設定 CloudTrail
- 複数のリージョンからの CloudTrail 口グファイルの受信と複数のアカウントからの CloudTrail 口グファイルの受信

Apache Flink 用 Managed Service アクションはすべて によって口グに記録 CloudTrail され、Apache Flink 用 Managed Service API リファレンス に記載されています。例えば、 および `UpdateApplication`アクションを呼び出す`CreateApplication`と、 CloudTrail 口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、誰がリクエストを生成したかという情報が含まれます。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 root または AWS Identity and Access Management (IAM) ユーザーの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 Element」を参照してください。

Managed Service for Apache Flink 口グファイルエントリについて

証跡は、指定した Amazon S3 バケットにイベントを口グファイルとして配信できるようにする設定です。 CloudTrail 口グファイルには 1 つ以上の口グエントリが含まれます。イベントは任意のソースからの単一のリクエストを表し、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に関する情報が含まれます。 CloudTrail 口グファイルはパブリック API コール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および DescribeApplicationアクションを示す CloudTrail ログエントリを示しています。


```
{
    "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": "2019-03-07T01:19:47Z",
            "eventSource": "kinesisanlaytics.amazonaws.com",
            "eventName": "AddApplicationCloudWatchLoggingOption",
            "awsRegion": "us-east-1",
            "sourceIPAddress": "127.0.0.1",
            "userAgent": "aws-sdk-java/unknown-version Linux/x.xx",
            "requestParameters": {
                "applicationName": "cloudtrail-test",
                "currentApplicationVersionId": 1,
                "cloudWatchLoggingOption": {
                    "logStreamARN": "arn:aws:logs:us-east-1:012345678910:log-
group:cloudtrail-test:log-stream:flink-cloudwatch"
                }
            },
            "responseElements": {
                "cloudWatchLoggingOptionDescriptions": [
                    {
                        "cloudWatchLoggingOptionId": "2.1",
                        "logStreamARN": "arn:aws:logs:us-east-1:012345678910:log-
group:cloudtrail-test:log-stream:flink-cloudwatch"
                    }
                ],
                "applicationVersionId": 2,
                "applicationARN": "arn:aws:kinesisanalyticsus-
east-1:012345678910:application/cloudtrail-test"
            },
            "requestID": "18dfb315-4077-11e9-afd3-67f7af21e34f",
            "eventID": "d3c9e467-db1d-4cab-a628-c21258385124",
```

```
            "eventType": "AwsApiCall",
            "apiVersion": "2018-05-23",
            "recipientAccountId": "012345678910"
        },
        {
            "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": "2019-03-12T02:40:48Z",
            "eventSource": "kinesisanlaytics.amazonaws.com",
            "eventName": "DescribeApplication",
            "awsRegion": "us-east-1",
            "sourceIPAddress": "127.0.0.1",
            "userAgent": "aws-sdk-java/unknown-version Linux/x.xx",
            "requestParameters": {
                "applicationName": "sample-app"
            },
            "responseElements": null,
            "requestID": "3e82dc3e-4470-11e9-9d01-e789c4e9a3ca",
            "eventID": "90ffe8e4-9e47-48c9-84e1-4f2d427d98a5",
            "eventType": "AwsApiCall",
            "apiVersion": "2018-05-23",
            "recipientAccountId": "012345678910"
        }
    ]
}
```

Apache Flink 用Amazon マネージドサービスのパフォーマンスのチューニング

このトピックでは、Apache Flink アプリケーション用 Managed Service のパフォーマンスを監視および改善する手法について説明します。

トピック

- パフォーマンスのトラブルシューティング
- パフォーマンスに関するベストプラクティス
- パフォーマンスのモニタリング

パフォーマンスのトラブルシューティング

このセクションには、パフォーマンスの問題を診断して修正するために確認できる徴候のリストが含まれています。

データソースが Kinesis ストリームの場合、パフォーマンスの問題は通常、高いまたは増加する `millisbehindLatest` メトリクスとして現れます。他のソースについては、ソースからの読み取りの遅れを表す同様のメトリクスを確認できます。

データパス

アプリケーションのパフォーマンス上の問題を調査するときは、データがたどる経路全体を考慮してください。以下のアプリケーションコンポーネントは、適切に設計またはプロビジョニングされていないと、パフォーマンスのボトルネックとなり、バックプレッシャーとなる可能性があります。

- 「データソースとデスティネーション:」アプリケーションがやり取りする外部リソースが、アプリケーションのスループットに合わせて適切にプロビジョニングされていることを確認してください。
- 「状態データ:」アプリケーションがステート・ストアとあまり頻繁にやり取りしないようにしてください。

  アプリケーションが使用しているシリアライザーを最適化できます。デフォルトの Kryo シリアライザーはシリアライズ可能なすべての型を処理できますが、アプリケーションが POJO タイプにしかデータを保存しない場合には、より高性能なシリアライザーを使用できます。Apache Flink

シリアライザーについて詳しくは、Apache Flink ドキュメントの「データ型とシリアル化」を参照してください。

- 「オペレータ:」オペレータが実装するビジネスロジックが複雑すぎないことや、処理されるレコードごとにリソースを作成したり使用したりしていないことを確認してください。また、アプリケーションがスライディングウィンドウやタンブリングウィンドウをあまり頻繁に作成していないようにしてください。

パフォーマンストラブルシューティングソリューション

このセクションでは、パフォーマンスに関する問題のソリューションを紹介します。

トピック

- CloudWatch 監視レベル
- アプリケーション CPU メトリック
- アプリケーション並列処理
- アプリケーションログ
- オペレータ並列処理
- アプリケーションロジック
- アプリケーションメモリ

CloudWatch 監視レベル

CloudWatch 監視レベルがあまりにも冗長な設定になっていないか確認してください。

Debug モニタリングログレベル設定では大量のトラフィックが生成され、バックプレッシャーが発生する可能性があります。アプリケーションの問題を積極的に調査する場合にのみ使用してください。

アプリケーション Parallelism の設定が高い場合、 Parallelism モニタリングメトリクスレベルを使用すると同様に大量のトラフィックが生成され、バックプレッシャーにつながる可能性があります。このメトリクス· レベルは、アプリケーションの Parallelism が低い場合、またはアプリケーションの問題を調査している場合にのみ使用します。

詳細については、「アプリケーションモニタリングレベル」を参照してください。

アプリケーション CPU メトリック

アプリケーションの CPU メトリクスを確認してください。このメトリックスが 75% を超える場合は、自動スケーリングを有効にすることで、アプリケーションがアプリケーション自体により多くのリソースを割り当てられるようにすることができます。

自動スケーリングが有効になっている場合、CPU 使用率が 15 分間 75% を超えると、アプリケーションはより多くのリソースを割り当てます。スケーリングの詳細については、次の スケーリングを適切に管理する セクション、「スケーリング」を参照してください。

> **Note**
>
> アプリケーションは CPU 使用率に応じてのみ自動的にスケーリングされます。アプリケーションは、heapMemoryUtilization などの他のシステムメトリクスに応じて自動スケーリングすることはありません。アプリケーションで他のメトリクスの使用率が高い場合は、アプリケーションの並列度を手動で増やします。

アプリケーション並列処理

アプリケーションの並列度を増やす。アクションのパラメータを使用してアプリケーションの並列処理を更新します。ParallelismConfigurationUpdate UpdateApplication

アプリケーションの最大 KPU はデフォルトで 64 ですが、制限の引き上げをリクエストすることで増やすことができます。

アプリケーションの並列度だけを増やすのではなく、そのワークロードに基づいて各オペレータに並列度を割り当てることも重要です。以下の オペレータ並列処理 を参照してください。

アプリケーションログ

処理中のすべてのレコードについて、アプリケーションがエントリを記録しているかどうかを確認してください。アプリケーションのスループットが高いときに各レコードにログエントリを書き込むと、データ処理に重大なボトルネックが生じます。この状態を確認するには、アプリケーションが処理するレコードごとに書き込まれるログエントリをログに問い合わせてください。アプリケーションログの読み取りの詳細については、 the section called “ログの分析” を参照してください。

オペレータ並列処理

アプリケーションのワークロードがワーカープロセスに均等に分散されていることを確認します。

アプリケーションのオペレータのワークロードのチューニングについては、 オペレータースケーリング を参照してください。

アプリケーションロジック

外部依存関係 (データベースやウェブサービスなど) へのアクセス、アプリケーション・ステートへのアクセスなど、非効率的な操作や非実行的な操作がないか、アプリケーションロジックを調べます。外部依存関係は、パフォーマンスが低かったり、確実にアクセスできない場合にもパフォーマンスを低下させる可能性があり、外部依存関係から HTTP 500 エラーが返される可能性があります。

アプリケーションが外部依存関係を使用して受信データを強化したり処理したりする場合は、代わりに非同期 IO の使用を検討してください。詳細については、「Apache Flink ドキュメント」の「非同期 IO」を参照してください。

アプリケーションメモリ

アプリケーションにリソースリークがないかチェックします。アプリケーションがスレッドやメモリを適切に処理していないと、millisbehindLatest、CheckpointSize、CheckpointDuration メトリクスが急増したり徐々に増加したりしていることがあります。この状態は、タスクマネージャーやジョブマネージャーの障害の原因にもなります。

パフォーマンスに関するベストプラクティス

このセクションでは、パフォーマンスを考慮したアプリケーションの設計に関する特別な考慮事項について説明します。

スケーリングを適切に管理する

このセクションには、アプリケーションレベルとオペレーターレベルのスケーリングの管理に関する情報が含まれています。

このセクションは、以下のトピックで構成されます。

- アプリケーションのスケーリングの適切な管理
- オペレータースケーリングの適切な管理

アプリケーションのスケーリングの適切な管理

自動スケーリングを使用すると、アプリケーションアクティビティの予期しない急増に対処できます。アプリケーションのKPUは、以下の基準を満たしている場合、自動的に増加します。

- アプリケーションの自動スケーリングが有効になっている。
- CPU 使用率が 75 %以上の状態が 15 分間続く。

自動スケーリングが有効になっていても CPU 使用率がこのしきい値を維持しない場合、アプリケーションは KPU をスケールアップしません。このしきい値を満たさないCPU使用率の急増や、heapMemoryUtilization などの別の使用率指標の急増が発生した場合は、アプリケーションがアクティビティ急増を処理できるように、手動でスケーリングを増やします。

> **Note**
>
> アプリケーションが自動スケーリングによってリソースを自動的に追加した場合、アプリケーションはしばらくアクティブになっていないときに新しいリソースを解放します。リソースをダウンスケーリングすると、一時的にパフォーマンスに影響します。

(スケーリングの詳細については、 スケーリング を参照してください。)

オペレータースケーリングの適切な管理

アプリケーションのワークロードがワーカープロセスに均等に分散されていることと、アプリケーション内のオペレーターが安定してパフォーマンスを発揮するために必要なシステムリソースを持っていることを確認することで、アプリケーションのパフォーマンスを向上させることができます。

parallelism 設定を使用して、アプリケーションのコード内の各オペレータの並列度を設定できます。オペレータに並列度を設定しない場合、アプリケーションレベルの並列度設定が使用されます。アプリケーションレベルの並列度設定を使用するオペレータは、アプリケーションで使用可能なすべてのシステムリソースを消費し、アプリケーションが不安定になる可能性があります。

各オペレータの並列度を最も適切に判断するには、アプリケーション内の他のオペレータと比較したオペレータの相対的なリソース要件を考慮します。リソースを大量に消費するオペレータには、リソースをあまり消費しないオペレータよりも高いオペレータ並列度を設定します。

アプリケーションのオペレータ並列度の合計は、アプリケーション内のすべてのオペレータの並列度の合計です。アプリケーションで使用可能なタスクスロットの合計との最適な比率を決定して、アプ

リケーションのオペレータ並列度全体を調整します。オペレータの総並列度とタスクタイムスロットの一般的な安定比は 4:1 です。つまり、アプリケーションが使用可能なオペレータサブタスクの4つごとに1つのタスクタイムスロットがあります。リソースを大量に消費するオペレータを使用するアプリケーションには 3:1 または 2:1 の比率が必要ですが、リソース集約度の低いオペレータを使用するアプリケーションでは 10:1 の比率で安定している場合があります。

オペレータの比率は ランタイムプロパティ を使用して設定できるため、アプリケーションコードをコンパイルしてアップロードしなくてもオペレータの並列度を調整できます。

以下のコード例は、オペレータの並列度を現在のアプリケーションの並列度に対する調整可能な比率として設定する方法を示しています。

```
Map<String, Properties> applicationProperties =
 KinesisAnalyticsRuntime.getApplicationProperties();
operatorParallelism =
    StreamExecutionEnvironment.getParallelism() /
    Integer.getInteger(

 applicationProperties.get("OperatorProperties").getProperty("MyOperatorParallelismRatio")
    );
```

サブタスク、タスクスロット、その他のアプリケーションリソースについては、 アプリケーションリソース を参照してください。

アプリケーションのワーカープロセス全体にわたるワークロードの分散を制御するには、Parallelism 設定と KeyBy パーティションメソッドを使用します。詳細については、「Apache Flink ドキュメント」の以下のトピックを参照してください。

- 同時実行
- DataStream トランスフォーメーション

外部依存リソースの使用状況を監視します。

デスティネーション (Kinesis ストリーム、Firehose、DynamoDB、 OpenSearch サービスなど) にパフォーマンスのボトルネックがあると、アプリケーションにバックプレッシャーが発生します。外部依存関係がアプリケーションのスループットに合わせて適切にプロビジョニングされていることを確認します。

> **Note**
>
> 他のサービスに障害が発生すると、アプリケーションに障害が発生する可能性があります。アプリケーションに障害が発生した場合は、 CloudWatch 宛先サービスのログに障害がないか確認してください。

Apache Flink アプリケーションをローカルで実行します。

メモリ問題をトラブルシューティングするには、アプリケーションをローカルの Flink インストールで実行できます。これにより、Apache Flink 用 Managed Service でアプリケーションを実行しているときには使用できないスタックトレースやヒープダンプなどのデバッグツールにアクセスできるようになります。

ローカルの Flink インストールの作成について詳しくは、Apache Flink ドキュメントの「スタンドアロン」を参照してください。

パフォーマンスのモニタリング

このセクションでは、アプリケーションのパフォーマンスを監視するためのツールについて説明します。

メトリクスを用いたパフォーマンスモニタリング CloudWatch

メトリクスを使用して、アプリケーションのリソース使用量、スループット、チェックポイント、ダウンタイムを監視します。 CloudWatch Managed Service for Apache Flink CloudWatch アプリケーションでメトリックを使用する方法については、を参照してください。Managed Service for Apache Flink のメトリクスとディメンション

CloudWatch ログとアラームを使ったパフォーマンスモニタリング

パフォーマンス上の問題を引き起こす可能性のあるエラー状態は、 CloudWatch ログを使用して監視します。

Apache Flink のジョブのステータスが `RUNNING` ステータスから `FAILED` ステータスに変化すると、エラー状態がログエントリに表示されます。

CloudWatch アラームを使用して、リソースの使用状況やチェックポイントメトリクスが安全なしきい値を超えたり、アプリケーションのステータスが予期せず変化したりするなど、パフォーマンス上の問題に関する通知を作成します。

Managed Service for Apache Flink CloudWatch アプリケーションのアラームを作成する方法については、を参照してください。アラーム

Managed Service for Apache Flink と Studio ノートブックのクォータ

> **Note**
>
> Apache Flink バージョン 1.6、1.8、および 1.11 は、Apache Flink コミュニティで 3 年以上サポートされていません。Amazon Managed Service for Apache Flink では、2024 年 11 月 5 日にこれらのバージョンを廃止する予定です。この日以降、これらの Flink バージョン用の新しいアプリケーションを作成することはできません。現時点では、既存のアプリケーションの実行を継続できます。Amazon Managed Service for Apache Flink のインプレースバージョンアップグレード機能を使用して、アプリケーションをステートリーにアップグレードできます。詳細については、「」を参照してくださいApache Flink のインプレースバージョンアップグレード。

Amazon Managed Service for Apache Flink を使用する場合は、次のクォータに注意してください。

- アカウントのリージョンごとに、最大 50 個の Apache Flink アプリケーション用 Managed Service アプリケーションを作成できます。サービスクォータ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 Support センターを参照してください。

  Apache Flink 用 Managed Service をサポートするリージョンのリストについては、「Apache Flink 用 Managed Service リージョンとエンドポイント 」を参照してください。

- Kinesis 処理ユニット (KPU) の数は、デフォルトで 64 に制限されています。このクォータの拡大をリクエストする方法については、「Service Quotas」の「クオータの拡大をリクエストするには」を参照してください。新しい KPU 制限を適用する必要があるアプリケーションプレフィックスを必ず指定してください。

  Managed Service for Apache Flink では、アプリケーションが使用するリソースではなく、割り当てられたリソースに対してアカウントに課金 AWS されます。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 の最大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 GiB のメモリーが提供されます。このサービスは、KPU ごとに 50 GiB の実行中のアプリケーションストレージもプロビジョニングします。

- アプリケーションごとに最大 1,000 個の Apache Flink 用 Managed Service スナップショット を作成できます。

- 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。

- アプリケーション JAR ファイルの最大サイズは 512 MiB です。このクォータを超えると、アプリケーションは起動できなくなります。

Studioノートブックの場合、以下のクォータが適用されます。クォータの引き上げをリクエストするには、 サポートケースを作成します。

- `websocketMessageSize` = 5 MiB
- `noteSize` = 5 MiB
- `noteCount` = 1000
- `Max cumulative UDF size` = 100 MiB
- `Max cumulative dependency jar size` = 300 MiB

Managed Service for Apache Flink メンテナンス

Managed Service for Apache Flink は、コンプライアンスを維持し、 AWS セキュリティ目標を達成するために、オペレーティングシステムとコンテナイメージのセキュリティ更新プログラムでアプリケーションに定期的にパッチを適用します。次の表は、Apache Flink 用 Managed Service がこの種のメンテナンスを実行するデフォルトのタイムウィンドウを示しています。アプリケーションのメンテナンスは、ユーザーのリージョンに対応する時間帯にいつでも行われる可能性があります。このメンテナンスプロセス中、アプリケーションに 10 ~ 30 秒のダウンタイムが発生する可能性があります。ただし、実際のダウンタイム時間はアプリケーションの状態によって異なります。このダウンタイムの影響を最小限に抑える方法については、 the section called “フォールトトレランス：チェックポイントとセーブポイント” を参照してください。

Managed Service for Apache Flink がアプリケーションのメンテナンスを実行する時間枠を変更するには、 UpdateApplicationMaintenanceConfiguration API を使用します。

リージョン	メンテナンスタイムウィンドウ
AWS GovCloud （米国西部）	06:00 ~ 14:00 UTC
AWS GovCloud （米国東部）	03:00 ~ 11:00 UTC
米国東部 (バージニア北部)	03:00 ~ 11:00 UTC
米国東部 (オハイオ)	03:00 ~ 11:00 UTC
米国西部 (北カリフォルニア)	06:00 ~ 14:00 UTC
米国西部 (オレゴン)	06:00 ~ 14:00 UTC
アジアパシフィック (香港)	13:00 ~ 21:00 UTC
アジアパシフィック (ムンバイ)	16:30 ~ 00:30 UTC
アジアパシフィック (ハイデラバード)	16:30 ~ 00:30 UTC
アジアパシフィック (ソウル)	13:00 ~ 21:00 UTC
アジアパシフィック (シンガポール)	14:00 ~ 22:00 UTC

リージョン	メンテナンスタイムウィンドウ
アジアパシフィック (シドニー)	12:00 ~ 20:00 UTC
アジアパシフィック (ジャカルタ)	15:00 ~ 23:00 UTC
アジアパシフィック (東京)	13:00 ~ 21:00 UTC
カナダ (中部)	03:00 ~ 11:00 UTC
中国 (北京)	13:00 ~ 21:00 UTC
中国 (寧夏)	13:00 ~ 21:00 UTC
欧州 (フランクフルト)	06:00 ~ 14:00 UTC
欧州 (チューリッヒ)	20:00 ~ 04:00 UTC
欧州 (アイルランド)	22:00 ~ 06:00 UTC
欧州 (ロンドン)	22:00 ~ 06:00 UTC
欧州 (ストックホルム)	23:00 ~ 07:00 UTC
欧州 (ミラノ)	21:00 ~ 05:00 UTC
欧州 (スペイン)	21:00 ~ 05:00 UTC
アフリカ (ケープタウン)	20:00 ~ 04:00 UTC
欧州 (アイルランド)	22:00 ~ 06:00 UTC
欧州 (ロンドン)	23:00 ~ 07:00 UTC
欧州 (パリ)	23:00 ~ 07:00 UTC
欧州 (ストックホルム)	23:00 ~ 07:00 UTC
中東 (バーレーン)	13:00 ~ 21:00 UTC
中東 (アラブ首長国連邦)	18:00 ~ 02:00 UTC

リージョン	メンテナンスタイムウィンドウ
南米 (サンパウロ)	19:00 ~ 03:00 UTC
イスラエル (テルアビブ)	20:00 ~ 04:00 UTC

すべてのオペレータに UUID を設定

Apache Flink 用 Managed Service がスナップショットを持つアプリケーションの Flink ジョブを開始するとき、何らかの問題で Flink ジョブが起動できないことがあります。その1つはオペレータ ID の不一致です。Flink では、Flink のジョブグラフオペレータには明示的で一貫性のあるオペレータ ID が必要です。明示的に設定されていない場合、Flink はオペレータの ID を自動生成します。これは、Flink がこれらのオペレータ ID を使用してジョブグラフ内のオペレータを一意に識別し、それを使用して各オペレータの状態をセーブポイントに保存するためです。

「オペレータ ID の不一致」の問題は、Flink がジョブグラフのオペレータ ID と、セーブポイントで定義されたオペレータ ID との間で 1:1 のマッピングを見つけられない場合に発生します。これは、明示的な一貫性のあるオペレータ ID が設定されておらず、Flink がすべてのジョブグラフ作成と一致しないオペレータ ID を自動生成した場合に発生します。メンテナンスの実行中にアプリケーションがこの問題に遭遇する可能性が高くなります。これを回避するには、Flink コードのすべての演算子に UUID を設定することをお勧めします。詳細については、「プロダクションレディネス」段階にある「すべてのオペレータに UUID を設定する」トピックを参照してください。

アプリケーションでメンテナンスがいつ行われたかを特定する

Managed Service for Apache Flink がアプリケーションでメンテナンスアクションを実行しているかどうかは、 `ListApplicationOperations` API を使用して確認できます。

以下は、アプリケーションのメンテナンスのためにリストをフィルタリングする`ListApplicationOperations`のに役立つ のリクエストの例です。

```
{
   "ApplicationName": "MyApplication",
   "operation": "ApplicationMaintenance"
}
```

プロダクション・レディネス

これは、Apache Flink 用 Managed Service でプロダクションアプリケーションを実行する上で重要な点をまとめたものです。これはすべてを網羅しているわけではなく、アプリケーションを本番環境に投入する前に注意すべき最低限のことをまとめたものです。

「アプリケーションの負荷テスト」

アプリケーションの問題の中には、高負荷時にのみ顕在化するものがあります。アプリケーションが正常に動作しているように見えても、運用上のイベントによってアプリケーションの負荷が大幅に増幅されるケースが確認されています。このような状況は、アプリケーション自体とはまったく関係なく発生することがあります。データソースまたはデータシンクが数時間使用できなくなると、Flink アプリケーションは処理を進められません。この問題が修正されると、未処理のデータが大量に蓄積され、利用可能なリソースを完全に使い果たす可能性があります。その負荷によって、これまで発生していなかったバグやパフォーマンスの問題が増幅される可能性があります。

そのため、本番環境のアプリケーションでは適切な負荷テストを実施することが不可欠です。これらの負荷テストでは、回答すべき質問は次のとおりです。

- 継続的な高負荷の下でアプリケーションは安定していますか?
- 負荷がピークに達しても、アプリケーションはセーブポイントを取ることができますか？
- 1 時間のバックログ処理にはどれくらい時間がかかりますか? また、(ストリーム内のデータの最大保持期間にもよるが) 24 時間ではどれくらいかかりますか？
- アプリケーションが拡張されると、アプリケーションのスループットは向上しますか?

データストリームから消費する場合、これらのシナリオは、一定時間ストリームに生成することでシミュレートすることができます。次に、アプリケーションを起動して、最初からデータを消費するようにします。たとえば、Kinesis データストリームの場合は、開始位置をとして使用します。`TRIM_HORIZON`

最大並列度

最大並列度は、ステートフルアプリケーションが拡張できる最大並列度を定義します。これはステートが最初に作成されたときに定義され、ステートを破棄せずにこの最大値を超えてオペレータをスケールする方法はありません。

最大並列処理数は、ステートが最初に作成されたときに設定されます。

デフォルトでは、最大並列処理数は以下のように設定されています。

- 128 (並列度が 128 未満の場合)
- `MIN(nextPowerOfTwo(parallelism + (parallelism / 2)), 2^15)` : (並列度が 128 を超える場合)

アプリケーションを並列処理数を 128 以上に拡張する予定の場合は、最大並列処理数を明示的に定義する必要があります。

最大並列処理数は、アプリケーションレベルで、または演算子を 1 つだけ使用して定義できます。`env.setMaxParallelism(x)`特に指定がない限り、すべてのオペレータはアプリケーションの Max 並列処理を継承します。

詳細については、Apache Flink ドキュメントの「最大並列処理数の設定」を参照してください。

すべてのオペレータに UUID を設定

UUID は、Flink がセーブポイントを 1 つのオペレータにマッピングする操作に使用されます。各オペレータに特定の UUID を設定すると、リストアするセーブポイントプロセスのために安定したマッピングを与えることができます。

```
.map(...).uid("my-map-function")
```

詳細については、「プロダクション・レディネスのチェックリスト」を参照してください。

Managed Service for Apache Flink のベストプラクティス

このセクションでは、安定したパフォーマンスの Managed Service for Apache Flink アプリケーションを開発するための情報と推薦事項を説明します。

トピック

- フォールトトレランス：チェックポイントとセーブポイント
- サポートされていないコネクタのバージョン。
- パフォーマンスと並列処理
- オペレータごとの並列処理の設定
- ログ記録
- コーディング
- ルート認証情報の管理。
- シャード/パーティションが少ないソースからの読み取り
- Studio ノートブックの更新間隔
- Studio ノートブックの最適なパフォーマンス
- ウォーターマーク戦略とアイドルシャードがタイムウィンドウに与える影響
- すべてのオペレータに UUID を設定
- Maven シェードプラグイン ServiceResourceTransformer に を追加する

フォールトトレランス：チェックポイントとセーブポイント

チェックポイントとセーブポイントを使用して、Managed Service for Apache Flinkにフォールトトレランスを実装します。アプリケーションの開発およびメンテナンスを行うときは、以下のことを考える必要があります。

- アプリケーションではチェックポイントを有効にしておくことをお勧めします。チェックポインティングは、定期メンテナンス中のほか、サービスの問題、アプリケーションの依存関係の障害、その他の問題が原因で予期しない障害が発生した場合にも、アプリケーションの耐障害性を実現します。メンテナンスの詳細については、「メンテナンス」を参照してください。
- アプリケーションの開発時またはトラブルシューティング`false`時に ApplicationSnapshotConfiguration::SnapshotsEnabled を に設定します。アプリケーションが停止するたびにスナップショットが作成されるため、アプリケーションが異常な状態であったり、パ

フォーマンスが低下したりすると問題が発生する可能性があります。アプリケーションが実稼働環境で安定した状態にはいった後 `SnapshotsEnabled` を `true` に設定します。

> **Note**
>
> アプリケーションが正しい状態データで正しく再起動できるように、1 日に数回スナップショットを作成することをおすすめします。スナップショットの正しい頻度は、アプリケーションのビジネスロジックによって異なります。頻繁にスナップショットを作成すると、より多くの最近のデータを復元できますが、コストが増加し、より多くのシステムリソースが必要になります。

アプリケーションのダウンタイムのモニタリングについては、Managed Service for Apache Flink のメトリクスとディメンション を参照してください。

障害耐性の詳細については、「耐障害性」を参照してください。

サポートされていないコネクタのバージョン。

Apache Flink バージョン 1.15 以降では、Managed Service for Apache Flink は、アプリケーション JARs。Managed Service for Apache Flink バージョン 1.15 以降にアップグレードする場合は、最新の Kinesis コネクタを使用していることを確認してください。これはバージョン 1.15.2 と同じかそれより新しいバージョンです。他のすべてのバージョンは、Managed Service for Apache Flink ではサポートされません。これは、Stop with Savepoint 機能で整合性の問題や障害が発生し、クリーン停止/更新オペレーションが妨げられる可能性があるためです。Amazon Managed Service for Apache Flink バージョンのコネクタ互換性の詳細については、「Apache Flink コネクタ」を参照してください。

パフォーマンスと並列処理

アプリケーションの並列処理を調整し、パフォーマンスの落とし穴を避けることで、アプリケーションをあらゆるスループットレベルに合わせて拡張できます。アプリケーションの開発およびメンテナンスを行うときは、以下のことを考える必要があります。

- すべてのアプリケーションのソースとシンクが十分にプロビジョニングされており、スロットルされていないことを確認します。ソースとシンクが他の AWS サービスである場合は、 を使用してそれらのサービスをモニタリングしますCloudWatch。

- 並列処理が非常に高いアプリケーションの場合は、アプリケーション内のすべての演算子に高レベルの並列処理が適用されているかどうかを確認してください。デフォルトでは、Apache Flink はアプリケーショングラフ内のすべてのオペレータに同じアプリケーション並列を適用します。これにより、ソースまたはシンクにおけるプロビジョニングの問題、またはオペレーターのデータ処理のボトルネックが発生する可能性があります。SetParallelismを使用すると、コードの各オペレータの並列処理を変更できます。
- アプリケーションのオペレータの並列処理設定の意味を理解してください。オペレーターの並列処理を変更すると、オペレーターの並列処理が現在の設定と互換性がないときに作成されたスナップショットからアプリケーションを復元できない場合があります。オペレータの並列処理の設定の詳細について、オペレータの最大並列処理を明示的に設定するを参照してください。

簡易スケーリングについての詳細は、「スケーリング」を参照してください。

オペレータごとの並列処理の設定

デフォルトでは、すべてのオペレータにアプリケーションレベルで並列処理が設定されます。を使用して DataStream API を使用して、単一の演算子の並列処理を上書きできます`.setParallelism(x)`。オペレータの並列処理は、アプリケーションの並列処理と同じかそれ以下の任意の並列処理に設定できます。

可能であれば、オペレータの並列処理をアプリケーション並列処理の関数として定義してください。このようにすると、演算子の並列処理はアプリケーションの並列処理によって変化します。たとえば、オートスケーリングを使用している場合は、すべてのオペレータの並列処理が同じ比率で変化します。

```
int appParallelism = env.getParallelism();
...
...ops.setParalleism(appParallelism/2);
```

場合によっては、オペレータの並列処理を定数に設定することをお勧めします。たとえば、Kinesis Stream ソースの並列処理をシャードの数に設定します。このような場合、ソース ストリームを再シャーディングする必要がある場合など、コードを変更せずにオペレータの並列処理をアプリケーション設定パラメーターとして渡すことを検討する必要があります。

ログ記録

CloudWatch Logs を使用して、アプリケーションのパフォーマンスとエラーの状態をモニタリングできます。アプリケーションのロギングを設定するときは、以下のことを考える必要があります。

- ランタイムの問題がデバッグできるように、アプリケーションの CloudWatch ログ記録を有効にします。
- アプリケーションで処理されているすべてのレコードについてログエントリを作成しないでください。これにより、処理中に深刻なボトルネックが発生し、データ処理でバックプレッシャーがある可能性があります。
- アプリケーションが正しく実行されていないときに通知する CloudWatch アラームを作成します。詳細については、「アラーム」を参照してください。

SCIM を実装する方法の詳細については、「ロギングとモニタリング」を参照してください。

コーディング

推薦プログラミング手法で、アプリケーションのパフォーマンスと安定性を高めることができます。アプリケーションコードを作成する際は、以下の事を考える必要があります。

- アプリケーションコードのsystem.exit()、アプリケーションのmainメソッド、またはユーザー定義関数では使用しないでください。コードからアプリケーションをシャットダウンする場合は、アプリケーションで問題が発生したことに関するメッセージを含むExceptionまたはRuntimeExceptionから派生した例外をスローします。

  サービスがこの例外を処理する方法については、以下の点に注意してください。

  - 例外がアプリケーションの `main` メソッドからスローされた場合、アプリケーションが `RUNNING` ステータスに移行したときにサービスが `ProgramInvocationException` でラップし、ジョブマネージャーはジョブの送信に失敗します。
  - 例外がユーザー定義関数からスローされた場合、ジョブ・マネージャーはそのジョブを失敗させて再起動し、例外の詳細が例外ログに書き込まれます。
- アプリケーション JAR ファイルとそれに含まれる依存関係をシェーディングすることを検討してください。アプリケーションと Apache Flink ランタイムの間でパッケージ名が競合する可能性がある場合は、シェーディングをお勧めします。競合が発生すると、アプリケーションログにタイプ `java.util.concurrent.ExecutionException` の例外が含まれる可能性があります。ア

プリケーション JAR ファイルのシェーディングの詳細について、Apache Maven Shade プラグインを参照してください。

ルート認証情報の管理。

長期認証情報を実稼働環境 (またはその他の)アプリケーションに組み込むべきではありません。長期認証情報はバージョン管理システムにチェックインされる可能性が高くて、簡単に紛失する可能性があります。代わりに、Managed Service for Apache Flink アプリケーションにロールと関連して、そのロールに権限を付与することができます。実行中の Flink アプリケーションは、それぞれの権限を持つ一時的な認証情報を環境から取得できます。IAM とネイティブに統合されていないサービス(認証にユーザー名とパスワードが必要なデータベースなど)で認証が必要な場合は、AWS Secrets Managerにシークレットを保存することを検討する必要があります。

多くの AWS ネイティブサービスは認証をサポートしています。

- Kinesis Data Streams – ProcessTaxiStream.java
- Amazon MSK – https://github.com/aws/aws-msk-iam-auth/#using-the-amazon-msk-library-for-iam-authentication
- Amazon Elasticsearch Service – AmazonElasticsearchSink.java
- Amazon S3 – works out of the box on Managed Service for Apache Flink

シャード/パーティションが少ないソースからの読み取り

Apache Kafka または Kinesis Data Streamsから読み取る場合、ストリームの並列処理 (Kafka のパーティション数と Kinesis のシャード数) とアプリケーションの並列処理が一致しない場合があります。単純な設計では、アプリケーションの並列処理はストリームの並列処理を超えることはできません。ソースオペレータの各サブタスクは、1 つ以上のシャード/パーティションからしか読み取ることができません。つまり、シャードが 2つのストリームであり、並列処理が 8 のアプリケーションである場合、ストリームから実際に消費しているのは 2 つのサブタスクだけで、6 つのサブタスクはアイドル状態のままです。これにより、アプリケーションのスループットが大幅に制限される可能性があります。特に、逆シリアル化にコストがかかり、ソース側で実行される場合 (デフォルト)はなおさらです。

この影響を軽減するには、ストリームをスケーリングする方法があります。しかし、それが常に望ましいとは限らないし、可能とも限らない。あるいは、ソースを再構築して、シリアライズを一切行わずに渡すようにすることもできます。あるいは、シリアル化を行わずにbyte[]を渡すようにソース

を再構築することもできます。その後、データを再調整してすべてのタスクに均等に分散して、そこでデータを逆シリアル化できます。この方法では、すべてのサブタスクを逆シリアル化に利用できるようになり、この高価になる可能性のある操作がストリームのシャード/パーティションの数に制限されなくなります。

Studio ノートブックの更新間隔

段落結果の更新間隔を変更する場合は、1000 ミリ秒以上の値に設定してください。

Studio ノートブックの最適なパフォーマンス

次のステートメントでテストしたところ、events-per-second と number-of-keys の積が 25,000,000 未満のときに最高のパフォーマンスが得られました。events-per-second は 150,000 未満でした。

```
SELECT key, sum(value) FROM key-values GROUP BY key
```

ウォーターマーク戦略とアイドルシャードがタイムウィンドウに与える影響

Apache Kafka と Kinesis Data Streamsからイベントを読み取るとき、ソースはストリームの属性に基づいてイベント時間を設定できます。Kinesis の場合、イベント時間はイベントのおおよその到着時間と等しくなります。ただし、Flink アプリケーションがイベント時間を使用するには、イベントのソースでイベント時間を設定するだけでは十分ではありません。ソースは、イベント時間に関する情報をソースから他のすべてのオペレーターに伝達するウォーターマークを生成する必要があります。Flink のドキュメントには、そのプロセスがどのように実行するかについての概要が書かれています。

デフォルトでは、Kinesis から読み取られたイベントのタイムスタンプは、Kinesis によって決定されておおよその到着時刻に設定されます。アプリケーションでイベント時間が機能するための追加の前提条件は、ウォーターマーク戦略です。

```
WatermarkStrategy<String> s = WatermarkStrategy
    .<String>forMonotonousTimestamps()
    .withIdleness(Duration.ofSeconds(...));
```

次に、ウォーターマーク戦略を `assignTimestampsAndWatermarks` メソッドで `DataStream` に適用します。便利なビルトイン・ストラテジーがあります

- `forMonotonousTimestamps()` はイベント時間 (おおよその到着時間) だけを使用して、(特定のサブタスクごとに) 定期的に最大値をウォーターマークとして出力します。
- `forBoundedOutOfOrderness(Duration.ofSeconds(...))` は前のストラテジーと似ていますが、ウォーターマークの生成にはイベント時間、つまり継続時間を使用します。

これはうまくいきますが、注意すべき点がいくつかあります。ウォーターマークはサブタスクレベルで生成されて、オペレータグラフに流れます。

Flink ドキュメンテーションから:

ソース関数の各並列サブタスクは通常、ウォーターマークを個別に生成します。これらのウォーターマークは、その特定の並列ソースでのイベント時間を定義します。

ウォーターマークがストリーミングプログラムを通過するにつれて、ウォーターマークが到着したオペレーターのイベント時間を進めます。 オペレータがイベント時間を進めるたびに、後続のオペレーターのために下流に新しいウォーターマークを生成します。

一部のオペレーターは複数の入力ストリームを消費します。 たとえば、ユニオン、または keyBy(…) 関数や Partition(…) 関数に続くオペレータなどです。このようなオペレータの現在のイベント時間は、入力ストリームのイベント時間の最小値です。入力ストリームがイベント時間を更新すると、オペレータもイベント時間を更新します。

つまり、ソースサブタスクがアイドルシャードから消費している場合、ダウンストリームオペレータはそのサブタスクから新しいウォーターマークを受け取らないため、タイムウィンドウを使用するすべてのダウンストリームオペレータの処理が停止します。これを避けるために、顧客はウォーターマークストラテジーに `withIdleness` オプションを追加することができます。このオプションを使用すると、オペレーターはオペレーターのイベント時間を計算するときに、アイドル状態のアップストリームサブタスクからウォーターマークを除外します。そのため、アイドル状態のサブタスクがダウンストリームオペレータのイベント時間の進行をブロックすることがなくなりました。

ただし、サブタスクがイベントを読み取っていない場合、つまりストリームにイベントが存在しない場合、組み込みのウォーターマーク戦略を使用したアイドル状態オプションはイベント時間を進めません。有限セットのイベントがストリームから読み取られるテスト ケースは特に顕著になります。最後のイベントが読み込まれてからイベント時間が進まないため、最後のウィンドウ (最後のイベントを含む) が閉じることはありません。

[概要]

- この `withIdleness` 設定では、シャードがアイドル状態の場合に新しいウォーターマークは生成されません。アイドル状態のサブタスクによって送信された最後のウォーターマークが、ダウンストリーム オペレーターの最小ウォーターマーク計算から除外されるだけです。
- 組み込みのウォーターマーク戦略では、最後に開いたウィンドウは決して閉じません (ウォーターマークを進める新しいイベントが送信されても、新しいウィンドウが作成されて開いたままになる場合を除く)。
- Kinesis ストリームによって時間が設定されている場合でも、1 つのシャードが他のシャードよりも早く消費されると、遅延到着イベントが発生する可能性があります (たとえば、アプリの初期化中や、既存のすべてのシャードが親子関係を無視して並行して消費される場合など)。
- ウォーターマーク戦略のwithIdleness設定は、アイドル状態のシャード (`ConsumerConfigConstants.SHARD_IDLE_INTERVAL_MILLIS` に対するKinesisソース固有の設定を廃止するようです。

例

次のアプリケーションはストリームから読み取って、イベント時間に基づいてセッションウィンドウを作成しています。

```
Properties consumerConfig = new Properties();
consumerConfig.put(AWSConfigConstants.AWS_REGION, "eu-west-1");
consumerConfig.put(ConsumerConfigConstants.STREAM_INITIAL_POSITION, "TRIM_HORIZON");

FlinkKinesisConsumer<String> consumer = new FlinkKinesisConsumer<>("...", new
 SimpleStringSchema(), consumerConfig);

WatermarkStrategy<String> s = WatermarkStrategy
    .<String>forMonotonousTimestamps()
    .withIdleness(Duration.ofSeconds(15));

env.addSource(consumer)
    .assignTimestampsAndWatermarks(s)
    .map(new MapFunction<String, Long>() {
        @Override
        public Long map(String s) throws Exception {
            return Long.parseLong(s);
        }
    })
```

```
    .keyBy(l -> 0l)
    .window(EventTimeSessionWindows.withGap(Time.seconds(10)))
    .process(new ProcessWindowFunction<Long, Object, Long, TimeWindow>() {
        @Override
        public void process(Long aLong, ProcessWindowFunction<Long, Object, Long,
 TimeWindow>.Context context, Iterable<Long>iterable, Collector<Object> collector)
 throws Exception {
            long count = StreamSupport.stream(iterable.spliterator(), false).count();
            long timestamp = context.currentWatermark();

            System.out.print("XXXXXXXXXXXXXX Window with " + count + " events");
            System.out.println("; Watermark: " + timestamp + ", " +
 Instant.ofEpochMilli(timestamp));


            for (Long l : iterable) {
                System.out.println(l);
            }
        }
    });
```

次の例では、8つのイベントが16シャード ストリームに書き込まれます(最初の2つと最後のイベントは偶然に同じシャードに配置されます)。

```
$ aws kinesis put-record --stream-name hp-16 --partition-key 1 --data MQ==
$ aws kinesis put-record --stream-name hp-16 --partition-key 2 --data Mg==
$ aws kinesis put-record --stream-name hp-16 --partition-key 3 --data Mw==
$ date

{
    "ShardId": "shardId-000000000012",
    "SequenceNumber": "49627894338614655560500811028721934184977530127978070210"
}
{
    "ShardId": "shardId-000000000012",
    "SequenceNumber": "49627894338614655560500811028795678659974022576354623682"
}
{
    "ShardId": "shardId-000000000014",
    "SequenceNumber": "49627894338659257050897872275134360684221592378842022114"
}
Wed Mar 23 11:19:57 CET 2022
```

```
$ sleep 10
$ aws kinesis put-record --stream-name hp-16 --partition-key 4 --data NA==
$ aws kinesis put-record --stream-name hp-16 --partition-key 5 --data NQ==
$ date

{
    "ShardId": "shardId-000000000010",
    "SequenceNumber": "49627894338570054070103749783042116732419934393936642210"
}
{
    "ShardId": "shardId-000000000014",
    "SequenceNumber": "49627894338659257050897872275659034489934342334017700066"
}
Wed Mar 23 11:20:10 CET 2022

$ sleep 10
$ aws kinesis put-record --stream-name hp-16 --partition-key 6 --data Ng==
$ date

{
    "ShardId": "shardId-000000000001",
    "SequenceNumber": "49627894338369347363316974173886988345467035365375213586"
}
Wed Mar 23 11:20:22 CET 2022

$ sleep 10
$ aws kinesis put-record --stream-name hp-16 --partition-key 7 --data Nw==
$ date

{
    "ShardId": "shardId-000000000008",
    "SequenceNumber": "49627894338525452579706688535878947299195189349725503618"
}
Wed Mar 23 11:20:34 CET 2022

$ sleep 60
$ aws kinesis put-record --stream-name hp-16 --partition-key 8 --data OA==
$ date

{
    "ShardId": "shardId-000000000012",
    "SequenceNumber": "49627894338614655560500811029600823255837371928900796610"
}
```

```
Wed Mar 23 11:21:27 CET 2022
```

この入力により、イベント1、2、3、イベント4、5、イベント6、イベント7、イベント8の5つのセッションウィンドウが生成されるはずです。ただし、このプログラムでは最初の4つのウィンドウしか生成されません。

```
11:59:21,529 INFO  org.apache.flink.streaming.connectors.kinesis.FlinkKinesisConsumer
 [] - Subtask 5 will be seeded with initial shard StreamShardHandle{streamName='hp-16',
 shard='{ShardId: shardId-000000000006,HashKeyRange: {StartingHashKey:
 127605887595351923798765477786913079296,EndingHashKey:
 148873535527910577765226390751398592511},SequenceNumberRange: {StartingSequenceNumber:
 49627894338480851089309627289524549239292625588395704418,}}'}, starting state set as
 sequence number EARLIEST_SEQUENCE_NUM
11:59:21,530 INFO
 org.apache.flink.streaming.connectors.kinesis.internals.KinesisDataFetcher [] -
 Subtask 5 will start consuming seeded shard StreamShardHandle{streamName='hp-16',
 shard='{ShardId: shardId-000000000006,HashKeyRange: {StartingHashKey:
 127605887595351923798765477786913079296,EndingHashKey:
 148873535527910577765226390751398592511},SequenceNumberRange: {StartingSequenceNumber:
 49627894338480851089309627289524549239292625588395704418,}}'} from sequence number
 EARLIEST_SEQUENCE_NUM with ShardConsumer 0
11:59:21,530 INFO  org.apache.flink.streaming.connectors.kinesis.FlinkKinesisConsumer
 [] - Subtask 6 will be seeded with initial shard StreamShardHandle{streamName='hp-16',
 shard='{ShardId: shardId-000000000007,HashKeyRange: {StartingHashKey:
 148873535527910577765226390751398592512,EndingHashKey:
 170141183460469231731687303715884105727},SequenceNumberRange: {StartingSequenceNumber:
 49627894338503151834508157912666084957565273949901684850,}}'}, starting state set as
 sequence number EARLIEST_SEQUENCE_NUM
11:59:21,530 INFO  org.apache.flink.streaming.connectors.kinesis.FlinkKinesisConsumer
 [] - Subtask 6 will be seeded with initial shard StreamShardHandle{streamName='hp-16',
 shard='{ShardId: shardId-000000000010,HashKeyRange: {StartingHashKey:
 212676479325586539664609129644855132160,EndingHashKey:
 233944127258145193631070042609340645375},SequenceNumberRange: {StartingSequenceNumber:
 49627894338570054070103749782090692112383219034419626146,}}'}, starting state set as
 sequence number EARLIEST_SEQUENCE_NUM
11:59:21,530 INFO
 org.apache.flink.streaming.connectors.kinesis.internals.KinesisDataFetcher [] -
 Subtask 6 will start consuming seeded shard StreamShardHandle{streamName='hp-16',
 shard='{ShardId: shardId-000000000007,HashKeyRange: {StartingHashKey:
 148873535527910577765226390751398592512,EndingHashKey:
 170141183460469231731687303715884105727},SequenceNumberRange: {StartingSequenceNumber:
 49627894338503151834508157912666084957565273949901684850,}}'} from sequence number
 EARLIEST_SEQUENCE_NUM with ShardConsumer 0
```

```
11:59:21,531 INFO  org.apache.flink.streaming.connectors.kinesis.FlinkKinesisConsumer
 [] - Subtask 4 will be seeded with initial shard StreamShardHandle{streamName='hp-16',
 shard='{ShardId: shardId-000000000005,HashKeyRange: {StartingHashKey:
 106338239662793269832304564822427566080,EndingHashKey:
 127605887595351923798765477786913079295},SequenceNumberRange: {StartingSequenceNumber:
 49627894338458550344111096666383013521019977226889723986,}}'}, starting state set as
 sequence number EARLIEST_SEQUENCE_NUM
11:59:21,532 INFO
 org.apache.flink.streaming.connectors.kinesis.internals.KinesisDataFetcher [] -
 Subtask 4 will start consuming seeded shard StreamShardHandle{streamName='hp-16',
 shard='{ShardId: shardId-000000000005,HashKeyRange: {StartingHashKey:
 106338239662793269832304564822427566080,EndingHashKey:
 127605887595351923798765477786913079295},SequenceNumberRange: {StartingSequenceNumber:
 49627894338458550344111096666383013521019977226889723986,}}'} from sequence number
 EARLIEST_SEQUENCE_NUM with ShardConsumer 0
11:59:21,532 INFO  org.apache.flink.streaming.connectors.kinesis.FlinkKinesisConsumer
 [] - Subtask 3 will be seeded with initial shard StreamShardHandle{streamName='hp-16',
 shard='{ShardId: shardId-000000000004,HashKeyRange: {StartingHashKey:
 850705917302346158658436518579420528 64,EndingHashKey:
 106338239662793269832304564822427566079},SequenceNumberRange: {StartingSequenceNumber:
 49627894338436249598912566043241477802747328865383743554,}}'}, starting state set as
 sequence number EARLIEST_SEQUENCE_NUM
11:59:21,532 INFO  org.apache.flink.streaming.connectors.kinesis.FlinkKinesisConsumer
 [] - Subtask 2 will be seeded with initial shard StreamShardHandle{streamName='hp-16',
 shard='{ShardId: shardId-000000000003,HashKeyRange: {StartingHashKey:
 638029437976759618993827388934565396 48,EndingHashKey:
 850705917302346158658436518579420528 63},SequenceNumberRange: {StartingSequenceNumber:
 49627894338413948853714035420099942084474680503877763122,}}'}, starting state set as
 sequence number EARLIEST_SEQUENCE_NUM
11:59:21,532 INFO  org.apache.flink.streaming.connectors.kinesis.FlinkKinesisConsumer
 [] - Subtask 3 will be seeded with initial shard StreamShardHandle{streamName='hp-16',
 shard='{ShardId: shardId-000000000015,HashKeyRange: {StartingHashKey:
 319014718988379809496913694467282698240,EndingHashKey:
 340282366920938463463374607431768211455},SequenceNumberRange: {StartingSequenceNumber:
 49627894338681557796096402897798370703746460841949528306,}}'}, starting state set as
 sequence number EARLIEST_SEQUENCE_NUM
11:59:21,532 INFO  org.apache.flink.streaming.connectors.kinesis.FlinkKinesisConsumer
 [] - Subtask 2 will be seeded with initial shard StreamShardHandle{streamName='hp-16',
 shard='{ShardId: shardId-000000000014,HashKeyRange: {StartingHashKey:
 297747071055821155530452781502797185024,EndingHashKey:
 319014718988379809496913694467282698239},SequenceNumberRange: {StartingSequenceNumber:
 49627894338659257050897872274656834985473812480443547874,}}'}, starting state set as
 sequence number EARLIEST_SEQUENCE_NUM
```

```
11:59:21,532 INFO
 org.apache.flink.streaming.connectors.kinesis.internals.KinesisDataFetcher [] -
 Subtask 3 will start consuming seeded shard StreamShardHandle{streamName='hp-16',
 shard='{ShardId: shardId-000000000004,HashKeyRange: {StartingHashKey:
 850705917302346158658436518579420528 64,EndingHashKey:
 106338239662793269832304564822427566079},SequenceNumberRange: {StartingSequenceNumber:
 496278943384362495989125660432414778027473288653837435 54,}}'} from sequence number
 EARLIEST_SEQUENCE_NUM with ShardConsumer 0
11:59:21,532 INFO
 org.apache.flink.streaming.connectors.kinesis.internals.KinesisDataFetcher [] -
 Subtask 2 will start consuming seeded shard StreamShardHandle{streamName='hp-16',
 shard='{ShardId: shardId-000000000003,HashKeyRange: {StartingHashKey:
 638029437976759618993827388934565396 48,EndingHashKey:
 850705917302346158658436518579420528 63},SequenceNumberRange: {StartingSequenceNumber:
 496278943384139488537140354200999420844746805038777631 22,}}'} from sequence number
 EARLIEST_SEQUENCE_NUM with ShardConsumer 0
11:59:21,532 INFO  org.apache.flink.streaming.connectors.kinesis.FlinkKinesisConsumer
 [] - Subtask 0 will be seeded with initial shard StreamShardHandle{streamName='hp-16',
 shard='{ShardId: shardId-000000000001,HashKeyRange: {StartingHashKey:
 212676479325586539664609129644855132 16,EndingHashKey:
 425352958651173079329218259289710264 31},SequenceNumberRange: {StartingSequenceNumber:
 496278943383693473633169741738168706479293837808658022 58,}}'}, starting state set as
 sequence number EARLIEST_SEQUENCE_NUM
11:59:21,532 INFO  org.apache.flink.streaming.connectors.kinesis.FlinkKinesisConsumer
 [] - Subtask 0 will be seeded with initial shard StreamShardHandle{streamName='hp-16',
 shard='{ShardId: shardId-000000000009,HashKeyRange: {StartingHashKey:
 191408831393027885698148216680369618 944,EndingHashKey:
 212676479325586539664609129644855132 159},SequenceNumberRange: {StartingSequenceNumber:
 496278943385477533249052191589491563941105706729136457 14,}}'}, starting state set as
 sequence number EARLIEST_SEQUENCE_NUM
11:59:21,532 INFO  org.apache.flink.streaming.connectors.kinesis.FlinkKinesisConsumer
 [] - Subtask 7 will be seeded with initial shard StreamShardHandle{streamName='hp-16',
 shard='{ShardId: shardId-000000000000,HashKeyRange: {StartingHashKey: 0,EndingHashKey:
 212676479325586539664609129644855132 15},SequenceNumberRange: {StartingSequenceNumber:
 496278943383470466181184435506753349296567354193598218 26,}}'}, starting state set as
 sequence number EARLIEST_SEQUENCE_NUM
11:59:21,533 INFO  org.apache.flink.streaming.connectors.kinesis.FlinkKinesisConsumer
 [] - Subtask 0 will be seeded with initial shard StreamShardHandle{streamName='hp-16',
 shard='{ShardId: shardId-000000000012,HashKeyRange: {StartingHashKey:
 255211775190703847597530955573826158 592,EndingHashKey:
 276479423123262501563991868538311671 807},SequenceNumberRange: {StartingSequenceNumber:
 496278943386146555605008110283737635489285157574315870 10,}}'}, starting state set as
 sequence number EARLIEST_SEQUENCE_NUM
```

```
11:59:21,533 INFO  org.apache.flink.streaming.connectors.kinesis.FlinkKinesisConsumer
 [] - Subtask 7 will be seeded with initial shard StreamShardHandle{streamName='hp-16',
 shard='{ShardId: shardId-000000000008,HashKeyRange: {StartingHashKey:
 170141183460469231731687303715884105728,EndingHashKey:
 191408831393027885698148216680369618943},SequenceNumberRange: {StartingSequenceNumber:
 496278943385254525797066885358076206758379223114076652 82,}}'}, starting state set as
 sequence number EARLIEST_SEQUENCE_NUM
11:59:21,533 INFO
 org.apache.flink.streaming.connectors.kinesis.internals.KinesisDataFetcher [] -
 Subtask 0 will start consuming seeded shard StreamShardHandle{streamName='hp-16',
 shard='{ShardId: shardId-000000000001,HashKeyRange: {StartingHashKey:
 212676479325586539664609129644855132 16,EndingHashKey:
 425352958651173079329218259289710264 31},SequenceNumberRange: {StartingSequenceNumber:
 496278943383693473633169741738168706479293837808658022 58,}}'} from sequence number
 EARLIEST_SEQUENCE_NUM with ShardConsumer 0
11:59:21,533 INFO  org.apache.flink.streaming.connectors.kinesis.FlinkKinesisConsumer
 [] - Subtask 7 will be seeded with initial shard StreamShardHandle{streamName='hp-16',
 shard='{ShardId: shardId-000000000011,HashKeyRange: {StartingHashKey:
 233944127258145193631070042609340645376,EndingHashKey:
 255211775190703847597530955573826158591},SequenceNumberRange: {StartingSequenceNumber:
 496278943385923548153022804052322278306558673959256065 78,}}'}, starting state set as
 sequence number EARLIEST_SEQUENCE_NUM
11:59:21,533 INFO
 org.apache.flink.streaming.connectors.kinesis.internals.KinesisDataFetcher [] -
 Subtask 7 will start consuming seeded shard StreamShardHandle{streamName='hp-16',
 shard='{ShardId: shardId-000000000000,HashKeyRange: {StartingHashKey: 0,EndingHashKey:
 212676479325586539664609129644855132 15},SequenceNumberRange: {StartingSequenceNumber:
 496278943383470466181184435506753349296567354193598218 26,}}'} from sequence number
 EARLIEST_SEQUENCE_NUM with ShardConsumer 0
11:59:21,568 INFO  org.apache.flink.streaming.connectors.kinesis.FlinkKinesisConsumer
 [] - Subtask 1 will be seeded with initial shard StreamShardHandle{streamName='hp-16',
 shard='{ShardId: shardId-000000000002,HashKeyRange: {StartingHashKey:
 425352958651173079329218259289710264 32,EndingHashKey:
 638029437976759618993827388934565396 47},SequenceNumberRange: {StartingSequenceNumber:
 496278943383916481085155047969584063662020321423717826 90,}}'}, starting state set as
 sequence number EARLIEST_SEQUENCE_NUM
11:59:21,568 INFO  org.apache.flink.streaming.connectors.kinesis.FlinkKinesisConsumer
 [] - Subtask 1 will be seeded with initial shard StreamShardHandle{streamName='hp-16',
 shard='{ShardId: shardId-000000000013,HashKeyRange: {StartingHashKey:
 276479423123262501563991868538311671808,EndingHashKey:
 297747071055821155530452781502797185023},SequenceNumberRange: {StartingSequenceNumber:
 496278943386369563056993416515152992672011641189375674 42,}}'}, starting state set as
 sequence number EARLIEST_SEQUENCE_NUM
```

```
11:59:21,568 INFO
 org.apache.flink.streaming.connectors.kinesis.internals.KinesisDataFetcher [] -
 Subtask 1 will start consuming seeded shard StreamShardHandle{streamName='hp-16',
 shard='{ShardId: shardId-000000000002,HashKeyRange: {StartingHashKey:
 42535295865117307932921825928971026432,EndingHashKey:
 63802943797675961899382738893456539647},SequenceNumberRange: {StartingSequenceNumber:
 49627894338391648108515504796958406366202032142371782690,}}'} from sequence number
 EARLIEST_SEQUENCE_NUM with ShardConsumer 0
11:59:23,209 INFO
 org.apache.flink.streaming.connectors.kinesis.internals.KinesisDataFetcher [] -
 Subtask 0 will start consuming seeded shard StreamShardHandle{streamName='hp-16',
 shard='{ShardId: shardId-000000000009,HashKeyRange: {StartingHashKey:
 191408831393027885698148216680369618944,EndingHashKey:
 212676479325586539664609129644855132159},SequenceNumberRange: {StartingSequenceNumber:
 49627894338547753324905219158949156394110570672913645714,}}'} from sequence number
 EARLIEST_SEQUENCE_NUM with ShardConsumer 1
11:59:23,244 INFO
 org.apache.flink.streaming.connectors.kinesis.internals.KinesisDataFetcher [] -
 Subtask 6 will start consuming seeded shard StreamShardHandle{streamName='hp-16',
 shard='{ShardId: shardId-000000000010,HashKeyRange: {StartingHashKey:
 212676479325586539664609129644855132160,EndingHashKey:
 233944127258145193631070042609340645375},SequenceNumberRange: {StartingSequenceNumber:
 49627894338570054070103749782090692112383219034419626146,}}'} from sequence number
 EARLIEST_SEQUENCE_NUM with ShardConsumer 1
event: 6; timestamp: 1648030822428, 2022-03-23T10:20:22.428Z
11:59:23,377 INFO
 org.apache.flink.streaming.connectors.kinesis.internals.KinesisDataFetcher [] -
 Subtask 3 will start consuming seeded shard StreamShardHandle{streamName='hp-16',
 shard='{ShardId: shardId-000000000015,HashKeyRange: {StartingHashKey:
 319014718988379809496913694467282698240,EndingHashKey:
 340282366920938463463374607431768211455},SequenceNumberRange: {StartingSequenceNumber:
 49627894338681557796096402897798370703746460841949528306,}}'} from sequence number
 EARLIEST_SEQUENCE_NUM with ShardConsumer 1
11:59:23,405 INFO
 org.apache.flink.streaming.connectors.kinesis.internals.KinesisDataFetcher [] -
 Subtask 2 will start consuming seeded shard StreamShardHandle{streamName='hp-16',
 shard='{ShardId: shardId-000000000014,HashKeyRange: {StartingHashKey:
 297747071055821155530452781502797185024,EndingHashKey:
 319014718988379809496913694467282698239},SequenceNumberRange: {StartingSequenceNumber:
 49627894338659257050897872274656834985473812480443547874,}}'} from sequence number
 EARLIEST_SEQUENCE_NUM with ShardConsumer 1
11:59:23,581 INFO
 org.apache.flink.streaming.connectors.kinesis.internals.KinesisDataFetcher [] -
 Subtask 7 will start consuming seeded shard StreamShardHandle{streamName='hp-16',
```

```
 shard='{ShardId: shardId-000000000008,HashKeyRange: {StartingHashKey:
 170141183460469231731687303715884105728,EndingHashKey:
 191408831393027885698148216680369618943},SequenceNumberRange: {StartingSequenceNumber:
 496278943385254525797066885358076206758379223114076652 82,}}'} from sequence number
 EARLIEST_SEQUENCE_NUM with ShardConsumer 1
11:59:23,586 INFO
 org.apache.flink.streaming.connectors.kinesis.internals.KinesisDataFetcher [] -
 Subtask 1 will start consuming seeded shard StreamShardHandle{streamName='hp-16',
 shard='{ShardId: shardId-000000000013,HashKeyRange: {StartingHashKey:
 276479423123262501563991868538311671808,EndingHashKey:
 297747071055821155530452781502797185023},SequenceNumberRange: {StartingSequenceNumber:
 496278943386369563056993416515152992672011641189375674 42,}}'} from sequence number
 EARLIEST_SEQUENCE_NUM with ShardConsumer 1
11:59:24,790 INFO
 org.apache.flink.streaming.connectors.kinesis.internals.KinesisDataFetcher [] -
 Subtask 0 will start consuming seeded shard StreamShardHandle{streamName='hp-16',
 shard='{ShardId: shardId-000000000012,HashKeyRange: {StartingHashKey:
 255211775190703847597530955573826158592,EndingHashKey:
 276479423123262501563991868538311671807},SequenceNumberRange: {StartingSequenceNumber:
 496278943386146555605008110283737635489285157574315870 10,}}'} from sequence number
 EARLIEST_SEQUENCE_NUM with ShardConsumer 2
event: 4; timestamp: 1648030809282, 2022-03-23T10:20:09.282Z
event: 3; timestamp: 1648030797697, 2022-03-23T10:19:57.697Z
event: 5; timestamp: 1648030810871, 2022-03-23T10:20:10.871Z
11:59:24,907 INFO
 org.apache.flink.streaming.connectors.kinesis.internals.KinesisDataFetcher [] -
 Subtask 7 will start consuming seeded shard StreamShardHandle{streamName='hp-16',
 shard='{ShardId: shardId-000000000011,HashKeyRange: {StartingHashKey:
 233944127258145193631070042609340645376,EndingHashKey:
 255211775190703847597530955573826158591},SequenceNumberRange: {StartingSequenceNumber:
 496278943385923548153022804052322278306558673959256065 78,}}'} from sequence number
 EARLIEST_SEQUENCE_NUM with ShardConsumer 2
event: 7; timestamp: 1648030834105, 2022-03-23T10:20:34.105Z
event: 1; timestamp: 1648030794441, 2022-03-23T10:19:54.441Z
event: 2; timestamp: 1648030796122, 2022-03-23T10:19:56.122Z
event: 8; timestamp: 1648030887171, 2022-03-23T10:21:27.171Z
XXXXXXXXXXXXXX Window with 3 events; Watermark: 1648030809281, 2022-03-23T10:20:09.281Z
3
1
2
XXXXXXXXXXXXXX Window with 2 events; Watermark: 1648030834104, 2022-03-23T10:20:34.104Z
4
5
XXXXXXXXXXXXXX Window with 1 events; Watermark: 1648030834104, 2022-03-23T10:20:34.104Z
```

```
6
XXXXXXXXXXXXXX Window with 1 events; Watermark: 1648030887170, 2022-03-23T10:21:27.170Z
7
```

出力には4つのウィンドウしか表示されていません (イベント8を含む最後のウィンドウはありません)。これはイベント時間とウォーターマーク戦略によるものです。ビルドごとのウォーターマーク戦略では、ストリームから読み込まれた最後のイベントの時間を超えて時間が進むことはないため、最後のウィンドウは閉じることができません。。ただし、ウィンドウを閉じるには、最後のイベントから10秒以上経過する必要があります。この場合、最後のウォーターマークは2022-03-23T10:21:27.170Zですが、セッションウィンドウを閉じるには、10秒と1ミリ秒後のウォーターマークが必要です。

この `withIdleness` オプションがウォーターマーク戦略から削除される場合、ウィンドウ オペレーターの「グローバル ウォーターマーク」を進めることができないため、セッション ウィンドウは閉じられなくなります。

Flink アプリケーションの起動時 (またはデータスキューの場合)、一部のシャードは他のシャードよりも早く消費される可能性があることに注意してください。これにより、サブタスクから出力される一部のウォーターマークが早すぎることがあります (サブタスクは、サブスクライブしている他のシャードから消費せずに、1つのシャードの内容に基づいてウォーターマークを出力する場合があります)。これを軽減する方法としては、安全バッファー (`forBoundedOutOfOrderness(Duration.ofSeconds(30))`) を追加したり、到着が遅れたイベント (`allowedLateness(Time.minutes(5))`) を明示的に許可したりするさまざまなウォーターマーク戦略があります。

すべてのオペレータに UUID を設定

Apache Flink 用 Managed Service がスナップショットを持つアプリケーションの Flink ジョブを開始するとき、何らかの問題で Flink ジョブが起動できないことがあります。その1つはオペレータ ID の不一致です。Flink では、Flink のジョブグラフオペレータには明示的で一貫性のあるオペレータ ID が必要です。明示的に設定されていない場合、Flink はオペレータの ID を自動生成します。これは、Flink がこれらのオペレータ ID を使用してジョブグラフ内のオペレータを一意に識別し、それを使用して各オペレータの状態をセーブポイントに保存するためです。

「オペレータ ID の不一致」の問題は、Flink がジョブグラフのオペレータ ID と、セーブポイントで定義されたオペレータ ID との間で 1:1 のマッピングを見つけられない場合に発生します。これは、明示的な一貫性のあるオペレータ ID が設定されておらず、Flink がすべてのジョブグラフ作成と一致しないオペレータ ID を自動生成した場合に発生します。メンテナンスの実行中にアプリケーショ

ンがこの問題に遭遇する可能性が高くなります。これを避けるため、Flink コードではすべてのオペレータに UUID を設定することをお勧めします。詳細については、「プロダクションレディネス」段階にある「すべてのオペレータに UUID を設定する」トピックを参照してください。

Maven シェードプラグイン ServiceResourceTransformer に を追加する

FlinkはJavaのサービスプロバイダーインターフェース (SPI)を使用して、コネクターやフォーマットなどのコンポーネントをロードします。SPI を使用する複数の Flink 依存関係により、uber-jar での衝突やアプリケーションの予期しない動作が発生する可能性があります。pom.xml で定義されている ServiceResourceTransformer Maven シェードプラグインの を追加することをお勧めします。

```
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-shade-plugin</artifactId>
                <executions>
                    <execution>
                        <id>shade</id>
                        <phase>package</phase>
                        <goals>
                            <goal>shade</goal>
                        </goals>
                        <configuration>
                            <transformers combine.children="append">
                                <!-- The service transformer is needed to merge META-
INF/services files -->
                                <transformer
 implementation="org.apache.maven.plugins.shade.resource.ServicesResourceTransformer"/>
                                <!-- ... -->
                            </transformers>
                        </configuration>
                    </execution>
                </executions>
        </plugin>
```

Apache フリンクステートフル関数

「ステートフル関数」は、分散型ステートフルアプリケーションの構築を簡素化する API です。これは、強固な一貫性が保証された状態で動的に相互作用できる永続的な状態の関数に基づいています。

ステートフルファンクションアプリケーションは基本的には単なる Apache Flink アプリケーションなので、Apache Flink 用 Managed Service にデプロイできます。ただし、Kubernetes クラスタ用と Apache Flink 用 Managed Serviceのステートフルファンクションのパッケージ化には、いくつかの違いがあります。ステートフルファンクションアプリケーションの最も重要な点は、ステートフルファンクションランタイムの設定に必要なランタイム情報がすべて「モジュール設定」に含まれていることです。通常、この設定はステートフル関数固有のコンテナにパッケージ化され、Kubernetes にデプロイされます。しかし、Apache Flink 用 Managed Serviceでは不可能です。

以下は、Apache Flink のマネージドサービス用の StateFun Python サンプルを改変したものです。

Apache Flink アプリケーションテンプレート

Stateful Functions ランタイムにカスタマーコンテナを使用する代わりに、Stateful Functions ランタイムを呼び出すだけで、必要な依存関係を含む Flink アプリケーション jar をコンパイルできます。Flink 1.13 では、必要な依存関係は次のようになります。

```
<dependency>
 <groupId>org.apache.flink</groupId>
 <artifactId>statefun-flink-distribution</artifactId>
 <version>3.1.0</version>
 <exclusions>
  <exclusion>
   <groupId>org.slf4j</groupId>
   <artifactId>slf4j-log4j12</artifactId>
  </exclusion>
  <exclusion>
   <groupId>log4j</groupId>
   <artifactId>log4j</artifactId>
  </exclusion>
 </exclusions>
</dependency>
```

また、Flink アプリケーションの Stateful Function ランタイムを呼び出す主なメソッドは以下のようになります。

```
public static void main(String[] args) throws Exception {
 final StreamExecutionEnvironment env =
 StreamExecutionEnvironment.getExecutionEnvironment();

 StatefulFunctionsConfig stateFunConfig = StatefulFunctionsConfig.fromEnvironment(env);

 stateFunConfig.setProvider((StatefulFunctionsUniverseProvider) (classLoader,
 statefulFunctionsConfig) -> {
  Modules modules = Modules.loadFromClassPath();
  return modules.createStatefulFunctionsUniverse(stateFunConfig);
 });

 StatefulFunctionsJob.main(env, stateFunConfig);
}
```

これらのコンポーネントは汎用的で、ステートフル関数に実装されているロジックとは無関係であることに注意してください。

モジュール設定の場所

ステートフル関数モジュール構成は、ステートフル関数ランタイムで検出できるようにクラスパスに含める必要があります。Flink アプリケーションの resources フォルダーに含め、jar ファイルにパッケージ化するのが一番です。

一般的な Apache Flink アプリケーションと同様に、Maven を使用して uber jar ファイルを作成し、それを Apache Flink のマネージドサービスにデプロイできます。

Apache フリンク設定

Apache Flink 用 Managed Service は、Apache Flink フレームワークの実装です。Apache Flink 用 Managed Service は、このセクションで説明されているデフォルト値を使用します。これらの値には、Apache Flink アプリケーション用 Managed Service を使用してコードで設定できるものもあれば、変更できないものもあります。

このトピックには、次のセクションが含まれています。

- Apache フリンク設定
- ステートバックエンド
- Checkpointing
- セーブポインティング
- ヒープサイズ
- バッファデブローティング
- 変更可能な Flink 設定プロパティ
- 設定済み Flink プロパティの表示

Apache フリンク設定

Apache Flink 用 Managed Service には、ほとんどのプロパティに Apache Flink 推奨値と、一般的なアプリケーションプロファイルに基づくいくつかの値で構成されるデフォルトの Flink 設定が用意されています。Flink 設定の詳細については、「設定」を参照してください。サービス提供のデフォルト設定は、ほとんどのアプリケーションに適用されます。ただし、並列度が高く、メモリや状態の使用量が多い特定のアプリケーションのパフォーマンスを向上させるために Flink の設定プロパティを微調整したり、Apache Flink の新しいデバッグ機能を有効にしたりする必要がある場合は、サポートケースをリクエストすることで特定のプロパティを変更することができます。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。アプリケーションの現在の設定は、Apache Flink ダッシュボードを使用して確認できます。

ステートバックエンド

Apache Flink 用 Managed Service は、一時的なデータをステートバックエンドに保存します。Apache Flink のマネージドサービスは RocksDB を使用していま

す。StateBackend「setStateBackend」を呼び出して別のバックエンドを設定しても効果はありません。

ステートバックエンドでは、以下の機能が利用できます。

- インクリメンタルステートのバックエンドスナップショット
- 非同期ステートのバックエンドスナップショット
- チェックポイントのローカルリカバリ

Apache Flink 用 Managed Service では、state.backend.rocksdb.ttl.compaction.filter.enabled 設定はデフォルトで有効になっています。このフィルターを使用すると、アプリケーションコードを更新してコンパクション・クリーンアップ・ストラテジーを有効にすることができます。詳細については、「Apache Flink ドキュメント」の「Flink 1.8.0 のステート TTL」を参照してください。

ステートバックエンドについて詳しくは、「Apache Flink ドキュメント」の「ステートバックエンド」を参照してください。

Checkpointing

Apache Flink 用 Managed Service は、以下の値を持つデフォルトのチェックポイント設定を使用します。これらの値の一部は変更可能です。設定する必要があります。CheckpointConfiguration ConfigurationType変更したチェックポイント値を使用するには、Apache Flink CUSTOM 用マネージドサービスの設定を変更してください。

設定	変更することはできますか？	その方法は?	デフォルト値
CheckpointingEnabled	変更可能	アプリケーションの作成 アプリケーションの更新 AWS CloudFormation	True
CheckpointInterval	変更可能	アプリケーションの作成	60000

設定	変更することはできますか？	その方法は?	デフォルト値
		アプリケーションの更新 AWS CloudFormation	
MinPauseBetweenCheckpoints	変更可能	アプリケーションの作成 アプリケーションの更新 AWS CloudFormation	5000
アライメントされていないチェックポイント	変更可能	サポートケース	False
同時チェックポイントの数	変更不可	該当なし	1
チェックポイントモード	変更不可	該当なし	1 回だけ
チェックポイント保持ポリシー	変更不可	該当なし	故障した時
チェックポイントタイムアウト	変更不可	該当なし	60 分
最大チェックポイントの保持	変更不可	該当なし	1
リスタートストラテジー	変更不可	該当なし	10 秒ごとに無限にリトライされる固定ディレイ。

設定	変更することはできますか？	その方法は?	デフォルト値
チェックポイントとセーブポイントの口ケーション	変更不可	該当なし	永続的なチェックポイントとセーブポイントのデータは、サービスが所有する S3 バケットに保存されます。
ステートバックエンドのメモリしきい値	変更不可	該当なし	1048576

セーブポインティング

デフォルトでは、セーブポイントからリストアする場合、再開操作はセーブポイントのすべての状態をリストアするプログラムにマッピングしようとします。オペレータをドロップした場合、デフォルトでは、欠落したオペレータに対応するデータを含むセーブポイントからのリストアは失敗します。AllowNonRestoredStateアプリケーションのパラメータをに設定すると、操作を正常に実行できます。FlinkRunConfiguration`true`これにより、再開操作は、新しいプログラムにマッピングできない状態をスキップすることができます。

詳細については、「Apache Flink ドキュメント」の「復元されない状態を許可する」を参照してください。

ヒープサイズ

Apache Flink 用 Managed Service は、各 KPU に 3 GiB の JVM ヒープを割り当て、ネイティブコード割り当て用に1 GiB を確保します。アプリケーションの容量を増やす情報については、 the section called “スケーリング” を参照してください。

の詳細については、「Apache ActiveMQ ドキュメント」の「設定」を参照してください。

バッファデブローティング

バッファデブローティングは、バックプレッシャーが高いアプリケーションに役立ちます。アプリケーションでチェックポイントやセーブポイントの失敗が発生する場合、この機能を有効にすると役に立ちます。そのためには、「サポートケース」をリクエストしてください。

詳細については、「Apache Flink ドキュメント」の「バッファデブローティングのメカニズム」を参照してください。

変更可能な Flink 設定プロパティ

「サポートケース」を使用して変更できる Flink 設定は次のとおりです。アプリケーションプレフィックスを指定することで、1 以上のプロパティを一度に変更できます。また、複数のアプリケーションを同時に変更できます。このリスト以外にも変更したい Flink 設定プロパティがある場合は、状況に応じて適切なプロパティを指定してください。

耐障害性

```
restart-strategy:

restart-strategy.fixed-delay.delay:
```

チェックポイントとステートバックエンド

```
state.backend:

state.backend.fs.memory-threshold:

state.backend.incremental:
```

Checkpointing

```
execution.checkpointing.unaligned:
```

RocksDB ネイティブメトリクス

RocksDB ネイティブメトリクスはには出荷されません。 CloudWatch有効にすると、これらのメトリクスには Flink ダッシュボードから、またはカスタムツールを使用して Flink REST API からアクセスできます。

Apache Flink 用マネージドサービスにより、お客様は API を使用して最新の Flink REST API (または使用しているサポート対象バージョン) に読み取り専用モードでアクセスできます。CreateApplicationPresignedUrlこの API は Flink 独自のダッシュボードで使用されますが、カスタム監視ツールでも使用できます。

```
state.backend.rocksdb.compaction.style:

state.backend.rocksdb.memory.partitioned-index-filters:

state.backend.rocksdb.metrics.actual-delayed-write-rate:

state.backend.rocksdb.metrics.background-errors:

state.backend.rocksdb.metrics.block-cache-capacity:

state.backend.rocksdb.metrics.block-cache-pinned-usage:

state.backend.rocksdb.metrics.block-cache-usage:

state.backend.rocksdb.metrics.column-family-as-variable:

state.backend.rocksdb.metrics.compaction-pending:

state.backend.rocksdb.metrics.cur-size-active-mem-table:

state.backend.rocksdb.metrics.cur-size-all-mem-tables:

state.backend.rocksdb.metrics.estimate-live-data-size:

state.backend.rocksdb.metrics.estimate-num-keys:

state.backend.rocksdb.metrics.estimate-pending-compaction-bytes:

state.backend.rocksdb.metrics.estimate-table-readers-mem:

state.backend.rocksdb.metrics.is-write-stopped:

state.backend.rocksdb.metrics.mem-table-flush-pending:

state.backend.rocksdb.metrics.num-deletes-active-mem-table:

state.backend.rocksdb.metrics.num-deletes-imm-mem-tables:

state.backend.rocksdb.metrics.num-entries-active-mem-table:
```

state.backend.rocksdb.metrics.num-entries-imm-mem-tables:

state.backend.rocksdb.metrics.num-immutable-mem-table:

state.backend.rocksdb.metrics.num-live-versions:

state.backend.rocksdb.metrics.num-running-compactions:

state.backend.rocksdb.metrics.num-running-flushes:

state.backend.rocksdb.metrics.num-snapshots:

state.backend.rocksdb.metrics.size-all-mem-tables:

state.backend.rocksdb.thread.num:

高度な状態バックエンドオプション

state.storage.fs.memory-threshold:

フルオプション TaskManager

task.cancellation.timeout:

taskmanager.jvm-exit-on-oom:

taskmanager.numberOfTaskSlots:

taskmanager.slot.timeout:

taskmanager.network.memory.fraction:

taskmanager.network.memory.max:

taskmanager.network.request-backoff.initial:

taskmanager.network.request-backoff.max:

taskmanager.network.memory.buffer-debloat.enabled:

taskmanager.network.memory.buffer-debloat.period:

taskmanager.network.memory.buffer-debloat.samples:

taskmanager.network.memory.buffer-debloat.threshold-percentages:

メモリ設定

taskmanager.memory.jvm-metaspace.size:

taskmanager.memory.jvm-overhead.fraction:

taskmanager.memory.jvm-overhead.max:

taskmanager.memory.managed.consumer-weights:

taskmanager.memory.managed.fraction:

taskmanager.memory.network.fraction:

taskmanager.memory.network.max:

taskmanager.memory.segment-size:

taskmanager.memory.task.off-heap.size:

RPC/Akka

akka.ask.timeout:

akka.client.timeout:

akka.framesize:

akka.lookup.timeout:

akka.tcp.timeout:

クライアント

client.timeout:

高度なクラスターオプション

cluster.intercept-user-system-exit:

cluster.processes.halt-on-fatal-error:

ファイルシステムの設定

`fs.s3.connection.maximum:`

`fs.s3a.connection.maximum:`

`fs.s3a.threads.max:`

`s3.upload.max.concurrent.uploads:`

高度なフォールトトレランスオプション

`heartbeat.timeout:`

`jobmanager.execution.failover-strategy:`

メモリ設定

`jobmanager.memory.heap.size:`

メトリクス

`metrics.latency.interval:`

REST エンドポイントとクライアント向けの拡張オプション

`rest.flamegraph.enabled:`

`rest.server.numThreads:`

高度な SSL セキュリティオプション

`security.ssl.internal.handshake-timeout:`

高度なスケジュールオプション

`slot.request.timeout:`

Flink ウェブ UI のアドバンスオプション

`web.timeout:`

設定済み Flink プロパティの表示

自分で設定した Apache Flink プロパティや、「サポートケース」を通じて修正を依頼した Apache Flink プロパティは、Apache Flink ダッシュボードから以下の手順で確認できます。

1. Flink ダッシュボードへ移動
2. 左側のナビゲーションペインで、[Job Manager] を選択します。
3. [設定] を選択すると、Flink プロパティのリストが表示されます。

Amazon VPC 内のリソースにアクセスするための Apache Flink のマネージドサービスの設定

アカウントで、Apache Flink アプリケーション用 Managed Service が 仮想プライベートクラウド (VPC) のプライベートサブネットに接続するように構成することができます。Amazon Virtual Private Cloud (Amazon VPC) を使用して、データベース、キャッシュインスタンス、内部サービスなどのリソースのプライベートネットワークを作成します。実行中にプライベートリソースにアクセスするには、アプリケーションを VPC に接続します。

このトピックには、次のセクションが含まれています。

- Amazon VPC の概念
- VPC アプリケーション権限
- Apache Flink 用 VPC 接続マネージドサービスアプリケーションのインターネットおよびサービスアクセス
- Managed Service for Apache Flink VPC API
- 例:VPC を使用して Amazon MSK クラスター内のデータにアクセスする

Amazon VPC の概念

Amazon VPC は、Amazon EC2 のネットワークレイヤーです。Amazon EC2 を初めて使用する場合は、Linux インスタンス用 Amazon EC2 ユーザーガイドの「Amazon EC2 とは」を参照してください。

VPC の主な概念は次のとおりです。

- 仮想プライベートクラウド (VPC) は、 AWS アカウント専用の仮想ネットワークです。
- サブネットは、VPC の IP アドレスの範囲です。
- ルートテーブルは、ネットワークトラフィックの経路を決めるために使用される一連のルール (ルートと呼ばれます) で構成されます。
- インターネットゲートウェイは、VPC のインスタンスとインターネットとの間の通信を可能にする VPC コンポーネントであり、冗長性と高い可用性を備えており、水平スケーリングが可能です。そのため、ネットワークトラフィックに課される可用性のリスクや帯域幅の制約はありません。

- VPC エンドポイントを使用すると、インターネットゲートウェイ、NAT デバイス、VPN 接続、PrivateLink 接続を必要とせずに、VPC AWS をサポートされているサービスや VPC エンドポイントサービスにプライベートに接続できます。 AWS Direct Connect VPC のインスタンスは、サービスのリソースと通信するためにパブリック IP アドレスを必要としません。VPC と他のサービス間のトラフィックは、Amazon ネットワークを離れません。

Amazon VPC サービスの詳細については、「Amazon Virtual Private Cloud ユーザーガイド」を参照してください。

Apache Flink 用 Managed Service は、アプリケーションの VPC 構成で提供されるサブネットの 1 つに「エラスティックネットワークインターフェイス」を作成します。VPC サブネット内に作成される Elastic Network Interface の数は、アプリケーションの並列処理と KPU ごとの並列度によって異なる場合があります。Application Scaling に関する詳細は、スケーリング を参照してください。

> **Note**
>
> SQL アプリケーションでは VPC 設定はサポートされていません。

> **Note**
>
> Apache Flink 用 Managed Service は、VPC 設定を持つアプリケーションのチェックポイントとスナップショットの状態を管理します。

VPC アプリケーション権限

このセクションでは、アプリケーションが VPC と連携するために必要なアクセス権限ポリシーについて説明します。アクセス権限ポリシーの使用については、 Amazon Managed Service for Apache Flink のIDとアクセスマネジメント を参照してください。

次のアクセス権限ポリシーは、VPC を操作するために必要なアクセス権限をアプリケーションに付与します。このアクセス権限ポリシーを使用するには、このポリシーをアプリケーションの実行ロールに追加します。

Amazon VPC にアクセスするためのアクセス権限ポリシー

```
{
```

```
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VPCReadOnlyPermissions",
      "Effect": "Allow",
      "Action": [
        "ec2:DescribeVpcs",
        "ec2:DescribeSubnets",
        "ec2:DescribeSecurityGroups",
        "ec2:DescribeDhcpOptions"
      ],
      "Resource": "*"
    },
    {
      "Sid": "ENIReadWritePermissions",
      "Effect": "Allow",
      "Action": [
        "ec2:CreateNetworkInterface",
        "ec2:CreateNetworkInterfacePermission",
        "ec2:DescribeNetworkInterfaces",
        "ec2:DeleteNetworkInterface"
      ],
      "Resource": "*"
    }

    ]
}
```

> **Note**
>
> コンソールを使用してアプリケーションリソース (CloudWatch Logs や Amazon VPC など) を指定すると、コンソールはアプリケーション実行ロールを変更して、それらのリソースにアクセスする権限を付与します。コンソールを使用せずにアプリケーションを作成する場合のみ、アプリケーションの実行ロールを手動で変更する必要があります。

Apache Flink 用 VPC 接続マネージドサービスアプリケーションのインターネットおよびサービスアクセス

デフォルトでは、Apache Flink アプリケーション用 Managed Service をアカウントの VPC に接続すると、VPC でアクセスが付与されない限り、インターネットにアクセスすることはできません。アプリケーションがインターネットアクセスを必要とする場合、以下の条件を満たす必要があります。

- Apache Flink アプリケーション用 Managed Service には、プライベートサブネットのみを設定する必要があります。
- VPC にはパブリックサブネット内の NAT ゲートウェイまたはインスタンスが含まれている必要があります。
- プライベートサブネットからパブリックサブネットの NAT ゲートウェイへのアウトバウンドトラフィック用のルートが存在する必要があります。

> **Note**
>
> 複数のサービスが VPC エンドポイントを提供します。VPC エンドポイントを使用すると、インターネットアクセスなしで VPC 内から Amazon サービスに接続できます。

サブネットがパブリックかプライベートかは、そのルートテーブルによって決まります。すべてのルートテーブルには、パブリックデスティネーションを持つパケットのネクストホップを決定するデフォルトルートがあります。

- 「プライベートサブネットの場合:」デフォルトルートは NAT ゲートウェイ (nat-...) または NAT インスタンス (eni-...) を指します。
- 「パブリックサブネットの場合:」デフォルトルートはインターネットゲートウェイ (igw-...) を指します。

パブリックサブネット (NAT 付き) と 1 つまたは複数のプライベートサブネットで VPC を構成したら、次の手順を実行してプライベートサブネットとパブリックサブネットを識別します。

- VPC コンソールのナビゲーションペインから、[Subnets] (サブネット) を選択します。
- サブネットを選択後、[Route Table] タブを選択します。デフォルトルートの確認:
  - 「パブリックサブネット:」Destination: 0.0.0.0/0、ターゲット:igw-...

- 「プライベートサブネット:」Destination: 0.0.0.0/0、ターゲット:NAT-... または eni-...

Apache Flink 用 Managed Service をプライベートサブネットに関連付けるには:

- https://console.aws.amazon.com/flink で Apache Flink 用 Managed Serviceコンソールを開く
- 「Apache Flink アプリケーション用 Managed Service」ページで、アプリケーションを選択し、「アプリケーションの詳細」を選択します。
- アプリケーションのページで、構成 をクリックします。
- 「VPC Connectivity」セクションで、アプリケーションに関連付ける VPC を選択します。アプリケーションが VPC リソースにアクセスするために使用する VPC に関連付けられているサブネットとセキュリティグループを選択します。
- [更新] を選択します。

関連情報

パブリックサブネットとプライベートサブネットを持つ VPC を作成する

NAT ゲートウェイの基本

Managed Service for Apache Flink VPC API

次の Apache Flink API オペレーション用 Managed Service を使用して、アプリケーションの VPC を管理します。Apache Flink API 用 Managed Serviceの使用方法については、 API サンプルコード を参照してください。

アプリケーションの作成

CreateApplicationアクションを使用して、作成中にアプリケーションに VPC 設定を追加します。

以下の `CreateApplication` アクションのリクエストコード例には、アプリケーションの作成時に VPC 構成が含まれています。

```
{
  "ApplicationName":"MyApplication",
  "ApplicationDescription":"My-Application-Description",
  "RuntimeEnvironment":"FLINK-1_15",
  "ServiceExecutionRole":"arn:aws:iam::123456789123:role/myrole",
```

```
  "ApplicationConfiguration": {
    "ApplicationCodeConfiguration":{
      "CodeContent":{
        "S3ContentLocation":{
          "BucketARN":"arn:aws:s3:::mybucket",
          "FileKey":"myflink.jar",
          "ObjectVersion":"AbCdEfGhIjKlMnOpQrStUvWxYz12345"
        }
      },
      "CodeContentType":"ZIPFILE"
    },
      "FlinkApplicationConfiguration":{
      "ParallelismConfiguration":{
        "ConfigurationType":"CUSTOM",
        "Parallelism":2,
        "ParallelismPerKPU":1,
        "AutoScalingEnabled":true
      }
    },
  "VpcConfigurations": [
        {
           "SecurityGroupIds": [ "sg-0123456789abcdef0" ],
           "SubnetIds": [ "subnet-0123456789abcdef0" ]
        }
     ]
  }
}
```

AddApplicationVpcConfiguration

AddApplicationVpcConfigurationアクションを使用して、作成後に VPC 設定をアプリケーションに追加します。

以下の AddApplicationVpcConfiguration アクションのリクエストコード例は、既存のアプリケーションに VPC 設定を追加します。

```
{
   "ApplicationName": "MyApplication",
   "CurrentApplicationVersionId": 9,
   "VpcConfiguration": {
      "SecurityGroupIds": [ "sg-0123456789abcdef0" ],
      "SubnetIds": [ "subnet-0123456789abcdef0" ]
   }
```

```
}
```

DeleteApplicationVpcConfiguration

DeleteApplicationVpcConfigurationアクションを使用して、アプリケーションから VPC 設定を削除します。

AddApplicationVpcConfiguration アクションの以下のリクエストコード例では、アプリケーションから既存の VPC タグを削除します。

```
{
   "ApplicationName": "MyApplication",
   "CurrentApplicationVersionId": 9,
   "VpcConfigurationId": "1.1"
}
```

アプリケーションを更新します。

UpdateApplicationアクションを使用して、アプリケーションのすべての VPC 設定を一度に更新します。

以下の UpdateApplication アクションのリクエストコード例は、アプリケーションのすべての VPC 構成を更新します。

```
{
   "ApplicationConfigurationUpdate": {
      "VpcConfigurationUpdates": [
         {
            "SecurityGroupIdUpdates": [ "sg-0123456789abcdef0" ],
            "SubnetIdUpdates": [ "subnet-0123456789abcdef0" ],
            "VpcConfigurationId": "2.1"
         }
      ]
   },
   "ApplicationName": "MyApplication",
   "CurrentApplicationVersionId": 9
}
```

例:VPC を使用して Amazon MSK クラスター内のデータにアクセスする

VPC 内の Amazon MSK クラスターのデータにアクセスする方法に関する詳細なチュートリアルについては、 MSK レプリケーション を参照してください。

Managed Service for Apache Flinkのトラブルシューティング

以下は、Amazon Managed Service for Apache Flinkで発生する可能性のある問題のトラブルシューティングに役立ちます。

トピック

- 開発のトラブルシューティング
- ランタイムのトラブルシューティング

開発のトラブルシューティング

トピック

- FlinkRuntimeException：「許可されていない設定変更 (複数可) が検出されました」
- システムロールバックのベストプラクティス
- Hudi 設定のベストプラクティス
- Apache Flink Flame Graphs
- EFO コネクタ 1.15.2 の認証情報プロバイダーの問題
- サポートされていない Kinesis コネクタを使用するアプリケーション
- コンパイルエラー：「プロジェクトの依存関係を解決できませんでした」
- 無効な選択肢：「kinesisanalyticsv2」
- UpdateApplication アクションがアプリケーションコードをリロードしていない
- S3 StreamingFileSink FileNotFoundExceptions
- FlinkKafkaConsumer セーブポイントによる停止に関する問題
- Flink 1.15 非同期シンクデッドロック
- Amazon Kinesis Data Streams のリシャーディング中にソース処理が順序どおりに行われない

FlinkRuntimeException：「許可されていない設定変更 (複数可) が検出されました」

Flink バージョン 1.19 以降、Flink ジョブコードを使用した Flink ジョブ設定の変更を無効にしました。

このようなサポートされていない設定変更を実行すると、次の例外が発生します。

```
Caused by: org.apache.flink.client.program.ProgramInvocationException: The main method
 caused an error: Not allowed configuration change(s) were detected:
 - Configuration execution.checkpointing.unaligned.enabled:true not allowed.
 - Setter CheckpointConfig#setCheckpointStorage has been used
    at
 org.apache.flink.client.program.PackagedProgram.callMainMethod(PackagedProgram.java:372)
    at
 org.apache.flink.client.program.PackagedProgram.invokeInteractiveModeForExecution(PackagedProg
    at org.apache.flink.client.ClientUtils.executeProgram(ClientUtils.java:108)
    at
 org.apache.flink.client.deployment.application.DetachedApplicationRunner.tryExecuteJobs(Detach
    ... 9 more
```

例外メッセージは、使用を許可していない設定またはセッターを示します。前の例では、アプリケーションは `execution.checkpointing.unaligned.enabled`と の設定と使用を試みます`setCheckpointStorage`。

コードで引き続き変更できる設定は次のとおりです。

1. パイプライン。auto-watermark-interval
2. パイプライン。closure-cleaner-level
3. pipeline.max-parallelism
4. pipeline.name
5. pipeline.operator-chaining.chain-operators-with-different-max-parallelism
6. pipeline.operator-chaining.enabled
7. パイプライン。vertex-description-mode
8. pipeline.vertex-name-include-index-prefix
9. python.execution-mode
10. python.operator-chaining.enabled

サポートケースを使用して、一部のジョブまたはクラスター設定の変更をリクエストできます。詳細については、「変更可能な Flink 設定プロパティ」を参照してください。

システムロールバックのベストプラクティス

Amazon Managed Service for Apache Flink の自動システムロールバック機能とオペレーション可視性機能により、アプリケーションの問題を特定して解決できます。

システムロールバック

コードのバグやアクセス許可の問題などの顧客エラーが原因でアプリケーションの更新またはスケーリングオペレーションが失敗した場合、この機能にオプトインすると、Amazon Managed Service for Apache Flink は以前の実行中のバージョンへのロールバックを自動的に試行します。詳細については、「Managed Service for Apache Flink アプリケーションのシステムロールバックの有効化」を参照してください。この自動ロールバックが失敗した場合、またはオプトインまたはオプトアウトしていない場合、アプリケーションは READY状態になります。アプリケーションを更新するには、次のステップを実行します。

手動ロールバック

アプリケーションが進行しておらず、長期間一時的な状態である場合、またはアプリケーションがに正常に移行したがRunning、正常に更新された Flink アプリケーションで処理エラーなどのダウンストリームの問題が表示される場合は、 RollbackApplication API を使用して手動でロールバックできます。

1. 呼び出し RollbackApplication - これにより、以前の実行中のバージョンに戻り、以前の状態が復元されます。
2. DescribeApplicationOperation API を使用してロールバックオペレーションをモニタリングします。
3. ロールバックが失敗した場合は、前のシステムロールバックステップを使用します。

オペレーションの可視性

ListApplicationOperations API には、アプリケーション上のすべてのカスタマーオペレーションとシステムオペレーションの履歴が表示されます。

1. 失敗したoperationId をリストから取得します。
2. を呼び出しDescribeApplicationOperation、ステータスと statusDescriptionを確認します。

3. オペレーションが失敗した場合、説明は調査する潜在的なエラーを指します。

一般的なエラーコードのバグ： ロールバック機能を使用して、最後に動作しているバージョンに戻ります。バグを解決し、更新を再試行してください。

アクセス許可の問題： `DescribeApplicationOperation`を使用して、必要なアクセス許可を確認します。アプリケーションのアクセス許可を更新して再試行します。

Amazon Managed Service for Apache Flink サービスの問題： を確認する AWS Health Dashboard か、サポートケースを開きます。

Hudi 設定のベストプラクティス

Managed Service for Apache Flink で Hudi コネクタを実行するには、次の設定変更をお勧めします。

`hoodie.embed.timeline.server` の無効化

Flink の Hudi コネクタは、Flink ジョブマネージャー (JM) に埋め込みタイムライン (TM) サーバーをセットアップしてメタデータをキャッシュし、ジョブの並列処理が高い場合にパフォーマンスを向上させます。JM と TM 間の非 Flink 通信を無効にするため、Managed Service for Apache Flink でこの埋め込みサーバーを無効にすることをお勧めします。

このサーバーが有効になっている場合、Hudi 書き込みはまず JM 上の埋め込みサーバーに接続しようとし、次に Amazon S3 からのメタデータの読み取りにフォールバックします。つまり、Hudi は接続タイムアウトが発生し、Hudi の書き込みが遅延し、Managed Service for Apache Flink のパフォーマンスに影響します。

Apache Flink Flame Graphs

フレームグラフをサポートする Apache Flink バージョンのマネージドサービスのアプリケーションでは、フレームグラフがデフォルトで有効になっています。「Flink ドキュメント」に記載されているように、フレームグラフを開いたままにしておくと、アプリケーションのパフォーマンスに影響する可能性があります。

アプリケーションの Flame Graph を無効にする場合は、ケースを作成してアプリケーション ARN で無効化するようリクエストしてください。詳細については、この「AWS サポートセンター」を参照してください。

EFO コネクタ 1.15.2 の認証情報プロバイダーの問題

1.15.2 以前のバージョンの Kinesis Data Streams EFO コネクタには、 `FlinkKinesisConsumer` が `Credential Provider` 設定を考慮しないという「既知の問題」があります。この問題により有効な構成が無視され、「AUTO」認証情報プロバイダーが使用されることになります。これにより、EFO コネクタを使用した Kinesis へのクロスアカウントアクセスで問題が発生する可能性があります。

このエラーを解決するには、EFO コネクタバージョン 1.15.3 以降を使用してください。

サポートされていない Kinesis コネクタを使用するアプリケーション

Managed Service for Apache Flink for Apache Flink バージョン 1.15 以降では、サポートされていない Kinesis Connector バージョン (バージョン 1.15.2 以前) がアプリケーション JAR またはアーカイブ (ZIP) にバンドルされている場合、アプリケーションの開始または更新が自動的に拒否されます。JARs

拒否エラー

アプリケーションの作成/更新コールを送信すると、次のようなエラーが表示されます：

```
An error occurred (InvalidArgumentException) when calling the CreateApplication
 operation: An unsupported Kinesis connector version has been detected in the
 application. Please update flink-connector-kinesis to any version equal to or newer
 than 1.15.2.
For more information refer to connector fix: https://issues.apache.org/jira/browse/
FLINK-23528
```

修正手順

- アプリケーションの `flink-connector-kinesis` への依存関係を更新します。Maven をプロジェクトのビルド・ツールとして使用している場合は、 Maven の依存関係を更新してください。 に従ってください。Gradle を使用している場合は、 Gradle の依存関係を更新してください。 に従ってください。
- アプリケーションをリパッケージする。
- Amazon S3 バケットにアップロードします。
- Amazon S3 バケットにアップロードしたばかりの改訂アプリケーションを使用して、アプリケーションの作成/更新リクエストを再送信します。

- 同じエラーメッセージが引き続き表示される場合は、アプリケーションの依存関係を再確認してください。問題が解決しない場合は、サポートチケットを作成してください。

Maven の依存関係を更新してください。

1. プロジェクトの pom.xml を開きます。
2. プロジェクトの依存関係を検索します。それらは以下のようになります。

```
<project>

    ...

    <dependencies>

        ...

        <dependency>
            <groupId>org.apache.flink</groupId>
            <artifactId>flink-connector-kinesis</artifactId>
        </dependency>

        ...

    </dependencies>

    ...

</project>
```

3. flink-connector-kinesis を1.15.2と同じバージョンまたはそれより新しいバージョンに更新します。例:

```
<project>

    ...

    <dependencies>

        ...

        <dependency>
            <groupId>org.apache.flink</groupId>
```

```
            <artifactId>flink-connector-kinesis</artifactId>
            <version>1.15.2</version>
        </dependency>

        ...

    </dependencies>

    ...

</project>
```

Gradle の依存関係を更新してください。

1. プロジェクト `build.gradle` (または Kotlin アプリケーションの場合は `build.gradle.kts`) を開きます。
2. プロジェクトの依存関係を検索します。それらは以下のようになります。

```
...

dependencies {

    ...

    implementation("org.apache.flink:flink-connector-kinesis")

    ...

}

...
```

3. `flink-connector-kinesis` を1.15.2と同じバージョンまたはそれより新しいバージョンに更新します。例:

```
...

dependencies {

    ...
```

```
    implementation("org.apache.flink:flink-connector-kinesis:1.15.2")

    ...

}

...
```

コンパイルエラー：「プロジェクトの依存関係を解決できませんでした」

Apache Flink 用マネージドサービスのサンプルアプリケーションをコンパイルするには、まず Apache Flink Kinesis コネクタをダウンロードしてコンパイルし、ローカルの Maven リポジトリに追加する必要があります。コネクタがリポジトリに追加されていない場合、次のようなコンパイルエラーが表示されます。

```
Could not resolve dependencies for project your project name: Failure to
 find org.apache.flink:flink-connector-kinesis_2.11:jar:1.8.2 in https://
repo.maven.apache.org/maven2 was cached in the local repository, resolution will not be
 reattempted until the update interval of central has elapsed or updates are forced
```

このエラーを解決するには、コネクタの Apache Flink ソースコード (「https://flink.apache.org/downloads.html」からバージョン 1.8.2) をダウンロードする必要があります。Apache Flink ソースコードのダウンロード、コンパイル、インストールの方法については、 the section called “以前の Apache Flink バージョンでの Apache Flink Kinesis Streams コネクタの使用” を参照してください。

無効な選択肢：「kinesisanalyticsv2」

Apache Flink API 用 Managed Service の v2 を使用するには、「 AWS Command Line Interface 」(「AWS CLI」)の最新バージョンが必要です。

のアップグレードの詳細については AWS CLI、「 AWS Command Line Interface ユーザーガイド」の「 のインストール AWS Command Line Interface」を参照してください。

UpdateApplication アクションがアプリケーションコードをリロードしていない

S3 オブジェクトバージョンが指定されていない場合、 UpdateApplicationアクションは同じファイル名のアプリケーションコードをリロードしません。同じファイル名でアプリケーションコードをリロードするには、S3 バケットでバージョニングを有効にし、 ObjectVersionUpdate パラメータ

を使用して新しいオブジェクトバージョンを指定します。S3バケットでオブジェクトのバージョニングを有効にする方法の詳細については、「バージョニングの有効化または無効化」を参照してください。

S3 StreamingFileSink FileNotFoundExceptions

Apache Flink アプリケーション用 Managed Serviceでは、セーブポイントによって参照される進行中のパーツファイルが見つからない場合、スナップショットから開始すると、処理中のパーツファイル FileNotFoundException に遭遇する可能性があります。この障害モードが発生した場合、Apache Flink アプリケーション用 Managed Serviceのオペレータ状態は通常回復不能になり、SKIP_RESTORE_FROM_SNAPSHOT を使用してスナップショットなしで再起動する必要があります。以下のスタックトレースの例を参照してください。

```
java.io.FileNotFoundException: No such file or directory: s3://your-s3-bucket/pathj/
INSERT/2023/4/19/7/_part-2-1234_tmp_12345678-1234-1234-1234-123456789012
        at
 org.apache.hadoop.fs.s3a.S3AFileSystem.s3GetFileStatus(S3AFileSystem.java:2231)
        at
 org.apache.hadoop.fs.s3a.S3AFileSystem.innerGetFileStatus(S3AFileSystem.java:2149)
        at
 org.apache.hadoop.fs.s3a.S3AFileSystem.getFileStatus(S3AFileSystem.java:2088)
        at org.apache.hadoop.fs.s3a.S3AFileSystem.open(S3AFileSystem.java:699)
        at org.apache.hadoop.fs.FileSystem.open(FileSystem.java:950)
        at
 org.apache.flink.fs.s3hadoop.HadoopS3AccessHelper.getObject(HadoopS3AccessHelper.java:98)
        at
 org.apache.flink.fs.s3.common.writer.S3RecoverableMultipartUploadFactory.recoverInProgressPart
...
```

Flink は、ファイルシステム でサポートされているファイルシステムにレコードをStreamingFileSink書き込みます。受信ストリームは無制限であるため、データは有限サイズの部分ファイルに編成されて、データが書き込まれると新しいファイルが追加されます。パーツのライフサイクルとロールオーバーポリシーによって、パーツファイルのタイミング、サイズ、名前が決まります。

チェックポイントとセーブポイント (スナップショット) 中に、保留中のすべてのファイルの名前が変更され、コミットされます。ただし、処理中のパーツファイルはコミットされずに名前が変更され、その参照はチェックポイントまたはセーブポイントのメタデータ内に保持され、ジョブの復元時に使用されます。これらの処理中のパーツファイルは、最終的に Pending にロールオーバーされ、名前が変更され、後続のチェックポイントまたはセーブポイントによってコミットされます。

処理中のパーツファイルが見つからない場合の根本原因と緩和策は次のとおりです。

- Managed Service for Apache Flink アプリケーションの起動に使用される古いスナップショット – アプリケーションが停止または更新されたときに作成された最新のシステムスナップショットのみを使用して、Amazon S3 StreamingFileSink で Managed Service for Apache Flink アプリケーションを起動できます。このような障害を回避するには、最新のシステムスナップショットを使用してください。
  - たとえば、停止中または更新中に、システム・トリガーによるスナップショットではなく、`CreateSnapshot` を使用して作成されたスナップショットを選択した場合に発生します。古いスナップショットのセーブポイントは、後続のチェックポイントまたはセーブポイントによって名前が変更され、コミットされた進行中のパートファイルへの out-of-date 参照を保持します。
  - これは、システムがトリガーした最新の Stop/Update イベント以外のスナップショットが選択された場合にも発生する可能性があります。例えば、システム・スナップショットが無効になっているが、 `RESTORE_FROM_LATEST_SNAPSHOT` が設定されているアプリケーションです。通常、Amazon S3 を使用する Managed Service for Apache Flink StreamingFileSink アプリケーションでは、常にシステムスナップショットを有効にして`RESTORE_FROM_LATEST_SNAPSHOT`設定する必要があります。
- 進行中の部分ファイルの削除 — 処理中の部分ファイルは S3 バケットにあるため、バケットにアクセスできる他のコンポーネントやアクターによって削除される可能性があります。
  - これは、アプリケーションが長時間停止しすぎ、アプリケーションのセーブポイントで参照される進行中のパートファイルが S3 バケット MultiPartUploadライフサイクルポリシーによって削除された場合に発生する可能性があります。このような障害を回避するには、S3 Bucket MPU ライフサイクルポリシーがユースケースに十分に対応した期間を対象としていることを確認してください。
  - これは、処理中のパーツファイルが手動で削除された場合や、システムの別のコンポーネントによって削除された場合にも発生する可能性があります。このような不具合を回避するには、処理中のパーツファイルが他のアクターやコンポーネントによって削除されないようにしてください。
- セーブポイントの後に自動チェックポイントがトリガーされる競合状態 — これは 1.13 以前の Apache Flink 用 Managed Service バージョンに影響します。この問題は、Managed Service for Apache Flink バージョン 1.15 で修正されています。Managed Service for Apache Flink の最新バージョンにアプリケーションを移行して、繰り返しを防ぎます。また、 から への移行もお勧め StreamingFileSink しますFileSink。
  - アプリケーションが停止または更新されると、Apache Flink 用 Managed Service はセーブポイントをトリガーし、2 つのステップでアプリケーションを停止します。2 つのステップの間に自

動チェックポイントがトリガーされると、処理中のパーツファイルの名前が変更され、コミットされる可能性があるため、セーブポイントは使用できなくなります。

FlinkKafkaConsumer セーブポイントによる停止に関する問題

レガシー を使用すると、システムスナップショットが有効になっている場合、アプリケーション FlinkKafkaConsumer が UPDATING、STOPPING、または SCALING で停止する可能性があります。この問題には公開されている修正がないため、この問題を軽減するために新しい にアップグレードすることをお勧めします。 KafkaSource

スナップショットを有効にして `FlinkKafkaConsumer` を使用している場合、Flink ジョブが savepoint API リクエストで STOP を処理すると、 `ClosedException` がランタイムエラーで報告されて `FlinkKafkaConsumer` が失敗する可能性があります。このような状況では、Flink アプリケーションが停止し、チェックポイント失敗と表示されます。

Flink 1.15 非同期シンクデッドロック

Apache Flink 実装 AsyncSink インターフェイスの AWS コネクタには既知の問題があります。これは、以下のコネクターで Flink 1.15 を使用するアプリケーションに影響します。

- Java アプリケーションの場合:
  - KinesisStreamsSink – `org.apache.flink:flink-connector-kinesis`
  - KinesisStreamsSink – `org.apache.flink:flink-connector-aws-kinesis-streams`
  - KinesisFirehoseSink – `org.apache.flink:flink-connector-aws-kinesis-firehose`
  - DynamoDbSink – `org.apache.flink:flink-connector-dynamodb`

- Flink SQL/テーブルAPI/Python アプリケーション:
  - Kinesis – `org.apache.flink:flink-sql-connector-kinesis`
  - Kinesis – `org.apache.flink:flink-sql-connector-aws-kinesis-streams`
  - firehose - `org.apache.flink:flink-sql-connector-aws-kinesis-firehose`
  - DynamoDB - `org.apache.flink:flink-sql-connector-dynamodb`

影響を受けるアプリケーションには次の症状があります。

- Flink ジョブの `RUNNING` 状態は変わりませんが、データは処理されていません。

- ジョブは再起動されません。
- チェックポイントがタイムアウトしています。

この問題は、 AWS SDK のバグが原因で、非同期 HTTP クライアントの使用時に発信者に特定のエラーが発生しないことが原因です。その結果、シンクはチェックポイントフラッシュ操作中に処理中のリクエストが完了するまで無期限に待機することになります。

この問題は、バージョン 2.20.144 以降の AWS SDK で修正されました。

以下に、影響を受けるコネクタを更新してアプリケーションで新しいバージョンの AWS SDK を使用する手順を示します。

トピック

- Java アプリケーションを更新する
- Python アプリケーションを更新する

Java アプリケーションを更新する

Java アプリケーションを更新するには、以下の手順に従います。

flink-connector-kinesis

このアプリケーションは `flink-connector-kinesis` を使用します。

Kinesis コネクタはシェーディングを使用して、 AWS SDK を含む一部の依存関係をコネクタ jar にパッケージ化します。 AWS SDK バージョンを更新するには、次の手順を使用して、これらのシェーディングクラスを置き換えます。

Maven

1. プロジェクトの依存関係として Kinesis コネクタと必要な AWS SDK モジュールを追加します。
2. `maven-shade-plugin` を設定します。
    a. Kinesis Connector jar のコンテンツをコピーするときに、シェーディングされた AWS SDK クラスを除外するフィルターを追加します。
    b. 再配置ルールを追加して、更新された AWS SDK クラスを Kinesis コネクタで想定されるパッケージに移動します。

pom.xml

```
<project>
    ...
    <dependencies>
        ...
        <dependency>
            <groupId>org.apache.flink</groupId>
            <artifactId>flink-connector-kinesis</artifactId>
            <version>1.15.4</version>
        </dependency>

        <dependency>
            <groupId>software.amazon.awssdk</groupId>
            <artifactId>kinesis</artifactId>
            <version>2.20.144</version>
        </dependency>
        <dependency>
            <groupId>software.amazon.awssdk</groupId>
            <artifactId>netty-nio-client</artifactId>
            <version>2.20.144</version>
        </dependency>
        <dependency>
            <groupId>software.amazon.awssdk</groupId>
            <artifactId>sts</artifactId>
            <version>2.20.144</version>
        </dependency>
        ...
    </dependencies>
    ...
    <build>
        ...
        <plugins>
            ...
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-shade-plugin</artifactId>
                <version>3.1.1</version>
                <executions>
                    <execution>
                        <phase>package</phase>
                        <goals>
```

```
                            <goal>shade</goal>
                        </goals>
                        <configuration>
                            ...
                            <filters>
                                ...
                                <filter>
                                    <artifact>org.apache.flink:flink-connector-
kinesis</artifact>
                                    <excludes>
                                        <exclude>org/apache/flink/kinesis/
shaded/software/amazon/awssdk/**</exclude>
                                        <exclude>org/apache/flink/kinesis/
shaded/org/reactivestreams/**</exclude>
                                        <exclude>org/apache/flink/kinesis/
shaded/io/netty/**</exclude>
                                        <exclude>org/apache/flink/kinesis/
shaded/com/typesafe/netty/**</exclude>
                                    </excludes>
                                </filter>
                                ...
                            </filters>
                            <relocations>
                                ...
                                <relocation>
                                    <pattern>software.amazon.awssdk</pattern>

 <shadedPattern>org.apache.flink.kinesis.shaded.software.amazon.awssdk</
shadedPattern>
                                </relocation>
                                <relocation>
                                    <pattern>org.reactivestreams</pattern>

 <shadedPattern>org.apache.flink.kinesis.shaded.org.reactivestreams</
shadedPattern>
                                </relocation>
                                <relocation>
                                    <pattern>io.netty</pattern>

 <shadedPattern>org.apache.flink.kinesis.shaded.io.netty</shadedPattern>
                                </relocation>
                                <relocation>
                                    <pattern>com.typesafe.netty</pattern>
```

```
 <shadedPattern>org.apache.flink.kinesis.shaded.com.typesafe.netty</
shadedPattern>
                                </relocation>
                                ...
                            </relocations>
                           ...
                       </configuration>
                   </execution>
               </executions>
           </plugin>
           ...
       </plugins>
       ...
   </build>
</project>
```

Gradle

1. プロジェクトの依存関係として Kinesis コネクタと必要な AWS SDK モジュールを追加します。
2. ShadowJar の設定を調整します。

   a. Kinesis Connector jar のコンテンツをコピーするときは、シェーディングされた AWS SDK クラスを除外します。

   b. 更新された AWS SDK クラスを Kinesis コネクタで期待されるパッケージに再配置します。

「グラドルをビルドする」

```
...
dependencies {
    ...
    flinkShadowJar("org.apache.flink:flink-connector-kinesis:1.15.4")

    flinkShadowJar("software.amazon.awssdk:kinesis:2.20.144")
    flinkShadowJar("software.amazon.awssdk:sts:2.20.144")
    flinkShadowJar("software.amazon.awssdk:netty-nio-client:2.20.144")
    ...
}
```

```
...
shadowJar {
    configurations = [project.configurations.flinkShadowJar]

    exclude("software/amazon/kinesis/shaded/software/amazon/awssdk/**/*")
    exclude("org/apache/flink/kinesis/shaded/org/reactivestreams/**/*.class")
    exclude("org/apache/flink/kinesis/shaded/io/netty/**/*.class")
    exclude("org/apache/flink/kinesis/shaded/com/typesafe/netty/**/*.class")

    relocate("software.amazon.awssdk",
 "org.apache.flink.kinesis.shaded.software.amazon.awssdk")
    relocate("org.reactivestreams",
 "org.apache.flink.kinesis.shaded.org.reactivestreams")
    relocate("io.netty", "org.apache.flink.kinesis.shaded.io.netty")
    relocate("com.typesafe.netty",
 "org.apache.flink.kinesis.shaded.com.typesafe.netty")
}
...
```

影響を受けるその他のコネクター

影響を受ける別のコネクタをアプリケーションで使用する場合:

AWS SDK バージョンを更新するには、プロジェクトビルド設定で SDK バージョンを適用する必要があります。

Maven

pom.xml ファイルの依存関係管理セクションに AWS SDK 部品表 (BOM) を追加して、プロジェクトの SDK バージョンを適用します。

pom.xml

```
<project>
    ...
    <dependencyManagement>
        <dependencies>
            ...
            <dependency>
                <groupId>software.amazon.awssdk</groupId>
                <artifactId>bom</artifactId>
                <version>2.20.144</version>
```

```
                <scope>import</scope>
                <type>pom</type>
            </dependency>
            ...
        </dependencies>
    </dependencyManagement>
    ...
</project>
```

Gradle

AWS SDK 部品表 (BOM) にプラットフォームの依存関係を追加して、プロジェクトの SDK バージョンを適用します。これには Gradle 5.0 以降が必要です。

「グラドルをビルドする」

```
...
dependencies {
    ...
    flinkShadowJar(platform("software.amazon.awssdk:bom:2.20.144"))
    ...
}
...
```

Python アプリケーションを更新する

Python アプリケーションでは、コネクタと他の Java 依存関係を単一の uber-jar の一部としてパッケージ化する方法と、コネクタ jar を直接使用する方法の 2 つの方法でコネクタを使用できます。Async Sink デッドロックの影響を受けるアプリケーションを修正するには:

- アプリケーションが uber jar を使用している場合は、 Java アプリケーションを更新する の指示に従ってください。
- コネクタ JAR をソースから再構築するには、以下の手順に従います。

「ソースからコネクタを構築:」

「Flink のビルド要件と同様の前提条件:」

- Java 11
- Maven 3.2.5

flink-sql-connector-kinesis

1. Flink 1.15.4 のソースコードのダウンロード:

```
wget https://archive.apache.org/dist/flink/flink-1.15.4/flink-1.15.4-src.tgz
```

2. ソースコードの解凍:

```
tar -xvf flink-1.15.4-src.tgz
```

3. Kinesis コネクタディレクトリに移動します。

```
cd flink-1.15.4/flink-connectors/flink-connector-kinesis/
```

4. コネクタ jar をコンパイルしてインストールし、必要な AWS SDK バージョンを指定します。-DskipTests ビルド時間を短縮するには、 -Dfast テスト実行をスキップして追加のソースコードチェックをスキップします。

```
mvn clean install -DskipTests -Dfast -Daws.sdkv2.version=2.20.144
```

5. Kinesis コネクタディレクトリに移動します。

```
cd ../flink-sql-connector-kinesis
```

6. SQL コネクタ jar をコンパイルしてインストールします。

```
mvn clean install -DskipTests -Dfast
```

7. 作成された jar は次の場所で入手できます。

```
target/flink-sql-connector-kinesis-1.15.4.jar
```

flink-sql-connector-aws-kinesis-streams

1. Flink 1.15.4 のソースコードのダウンロード:

```
wget https://archive.apache.org/dist/flink/flink-1.15.4/flink-1.15.4-src.tgz
```

2. ソースコードの解凍:

```
tar -xvf flink-1.15.4-src.tgz
```

3. Kinesis コネクタディレクトリに移動します。

```
cd flink-1.15.4/flink-connectors/flink-connector-aws-kinesis-streams/
```

4. コネクタ jar をコンパイルしてインストールし、必要な AWS SDK バージョンを指定します。-DskipTests ビルド時間を短縮するには、 -Dfast テスト実行をスキップして追加のソースコードチェックをスキップします。

```
mvn clean install -DskipTests -Dfast -Daws.sdk.version=2.20.144
```

5. Kinesis コネクタディレクトリに移動します。

```
cd ../flink-sql-connector-aws-kinesis-streams
```

6. SQL コネクタ jar をコンパイルしてインストールします。

```
mvn clean install -DskipTests -Dfast
```

7. 作成された jar は次の場所で入手できます。

```
target/flink-sql-connector-aws-kinesis-streams-1.15.4.jar
```

flink-sql-connector-aws-kinesis-firehose

1. Flink 1.15.4 のソースコードのダウンロード:

```
wget https://archive.apache.org/dist/flink/flink-1.15.4/flink-1.15.4-src.tgz
```

2. ソースコードの解凍:

```
tar -xvf flink-1.15.4-src.tgz
```

3. コネクタディレクトリに移動

```
cd flink-1.15.4/flink-connectors/flink-connector-aws-kinesis-firehose/
```

4. コネクタ jar をコンパイルしてインストールし、必要な AWS SDK バージョンを指定します。-DskipTests ビルド時間を短縮するには、 -Dfast テスト実行をスキップして追加のソースコードチェックをスキップします。

```
mvn clean install -DskipTests -Dfast -Daws.sdk.version=2.20.144
```

5. SQL コネクタディレクトリに移動します。

```
cd ../flink-sql-connector-aws-kinesis-firehose
```

6. SQL コネクタ jar をコンパイルしてインストールします。

```
mvn clean install -DskipTests -Dfast
```

7. 作成された jar は次の場所で入手できます。

```
target/flink-sql-connector-aws-kinesis-firehose-1.15.4.jar
```

flink-sql-connector-dynamodb

1. Flink 1.15.4 のソースコードのダウンロード:

```
wget https://archive.apache.org/dist/flink/flink-connector-aws-3.0.0/flink-
connector-aws-3.0.0-src.tgz
```

2. ソースコードの解凍:

```
tar -xvf flink-connector-aws-3.0.0-src.tgz
```

3. コネクタディレクトリに移動

```
cd flink-connector-aws-3.0.0
```

4. コネクタ jar をコンパイルしてインストールし、必要な AWS SDK バージョンを指定します。-DskipTests ビルド時間を短縮するには、 -Dfast テスト実行をスキップして追加のソースコードチェックをスキップします。

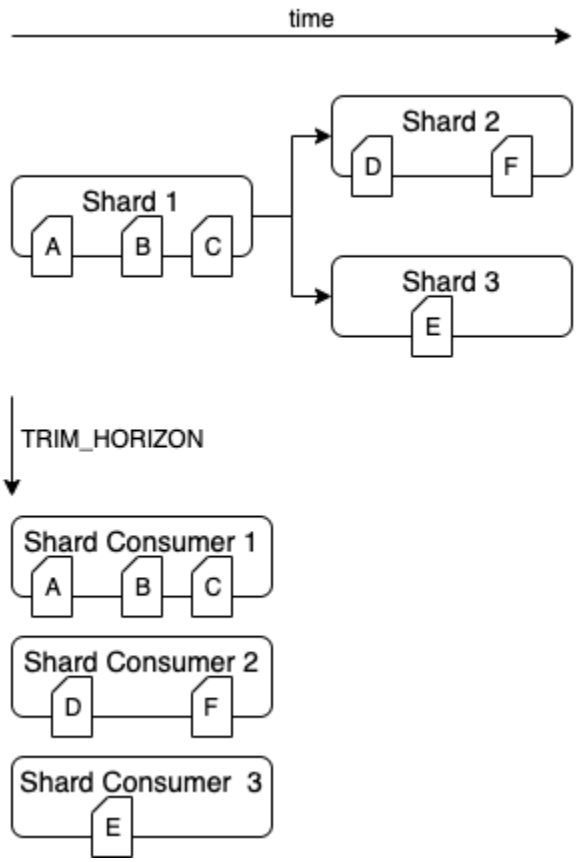
```
mvn clean install -DskipTests -Dfast -Dflink.version=1.15.4 -
Daws.sdk.version=2.20.144
```

5. 作成された jar は次の場所で入手できます。

```
flink-sql-connector-dynamodb/target/flink-sql-connector-dynamodb-3.0.0.jar
```

Amazon Kinesis Data Streams のリシャーディング中にソース処理が順序どおりに行われない

現在の FlinkKinesisConsumer 実装では、Kinesis シャード間の強力な順序保証は提供されていません。これにより、Kinesis Streams の再シャーディング中に、特に out-of-order 処理ラグが発生する Flink アプリケーションに対して処理が行われる可能性があります。たとえば、イベント時間に基づくウィンドウオペレーターのような状況下では、遅延が原因でイベントが破棄される可能性があります。



これはオープンソースの Flink の既知の問題です。コネクタの修正が利用可能になるまで、Flink アプリケーションが Kinesis データストリームのリパーティショニング中に遅れないようにすることが重要です。Flink アプリで処理遅延が許容されるようにすることで、 out-of-order 処理の影響とデータ損失のリスクを最小限に抑えることができます。

ランタイムのトラブルシューティング

このセクションには、Apache Flink アプリケーション用 Managed Serviceの実行時問題の診断と修正に関する情報が含まれています。

トピック

- トラブルシューティングツール
- アプリケーションの問題
- アプリケーションを再起動中
- スループットが遅すぎる
- 州の無限の成長
- I/O バウンドオペレーター
- Kinesis データストリームからのアップストリームまたはソーススロットリング
- チェックポイント
- チェックポイントがタイムアウトしています。
- Apache Beam アプリケーションのチェックポイント障害
- バックプレッシャー
- データスキュー機能
- ステートスキュー機能
- 異なるリージョンのリソースとの統合

トラブルシューティングツール

アプリケーションの問題を検出するための主なツールは CloudWatch アラームです。アラームを使用すると CloudWatch 、アプリケーションのエラーまたはボトルネック状態を示す CloudWatch メトリクスのしきい値を設定できます。推奨される CloudWatch アラームについては、「」を参照してくださいAmazon Managed Service for Apache Flink での CloudWatch アラームの使用。

アプリケーションの問題

このセクションには、Apache Flink アプリケーション用 Managed Serviceで発生する可能性のあるエラー状態に対する解決策が含まれています。

トピック

- アプリケーションが一時的なステータスでスタックしている
- スナップショットの作成が失敗する
- VPC 内のリソースにアクセスできない
- Amazon S3 バケットへの書き込み時にデータが失われる
- アプリケーションは RUNNING ステータスですが、データを処理していません
- スナップショット、アプリケーション更新、またはアプリケーション停止エラー: InvalidApplicationConfigurationException
- java.nio.file.NoSuchFileException: /usr/local/openjdk-8/lib/security/cacerts

アプリケーションが一時的なステータスでスタックしている

アプリケーションが一時的なステータス (STARTING、UPDATING、、または AUTOSCALING) のままである場合はSTOPPING、 Forceパラメータを に設定して StopApplicationアクションを使用してアプリケーションを停止できますtrue。この DELETING ステータスのアプリケーションを強制停止することはできません。または、アプリケーションが UPDATING または AUTOSCALING ステータスの場合は、実行中の以前のバージョンにロールバックできます。アプリケーションをロールバックすると、前回成功したスナップショットの状態データがロードされます。アプリケーションにスナップショットがない場合、Apache Flink 用 Managed Serviceはロールバックリクエストを拒否します。アプリケーションのロールバックの詳細については、「 RollbackApplicationアクション」を参照してください。

> **Note**
>
> アプリケーションを強制停止すると、データが失われたり重複したりする可能性があります。アプリケーションの再起動時にデータが失われたり、データが重複して処理されたりするのを防ぐため、アプリケーションのスナップショットを頻繁に撮ることをお勧めします。

アプリケーションが停止する原因には次のようなものがあります。

- 「アプリケーションの状態が大きすぎる:」アプリケーションの状態が大きすぎたり永続的すぎたりすると、チェックポイントまたはスナップショット操作中にアプリケーションが停止する可能性があります。アプリケーションの lastCheckpointDuration と lastCheckpointSize メトリックをチェックして、値が着実に増加しているか、または異常に高い値がないかを確認してください。

- 「アプリケーションコードが大きすぎる:」アプリケーションの JAR ファイルが 512 MB 未満であることを確認してください。512 MB を超える JAR ファイルはサポートされていません。
- 「アプリケーションスナップショットの作成に失敗する:」Managed Service for Apache Flinkが UpdateApplication または StopApplication リクエスト中にアプリケーションのスナップショットを取得します。その後、サービスはこのスナップショット状態を使用し、更新されたアプリケーション設定を使用してアプリケーションを復元し、「1 回のみ」の処理セマンティクスを実現します。自動スナップショット作成が失敗した場合は、 スナップショットの作成が失敗する 以下を参照してください。
- 「スナップショットからの復元が失敗する:」アプリケーションの更新でオペレータを削除または変更した後に、スナップショットから復元しようとした場合、スナップショットに欠落しているオペレータの状態データが含まれていると、復元はデフォルトで失敗します。さらに、アプリケーションは STOPPED または UPDATING ステータスのままになります。この動作を変更して復元を成功させるには、アプリケーションの の AllowNonRestoredStateパラメータを FlinkRunConfiguration に変更しますtrue。これにより、新しいプログラムにマッピングできない状態データを再開操作がスキップできます。
- 「アプリケーションの初期化に時間がかかる:」Apache Flink 用 Managed Service fは、Flink ジョブの開始を待つ間、5 分間の内部タイムアウト (ソフト設定) を使用します。ジョブがこのタイムアウト内に開始されない場合は、次のように CloudWatch ログが表示されます。

  ```
  Flink job did not start within a total timeout of 5 minutes for application: %s under
   account: %s
  ```

  上記のエラーが発生した場合、Flink ジョブの main メソッドで定義されている操作に 5 分以上かかっているため、Apache Flink のマネージドサービス側で Flink ジョブの作成がタイムアウトになります。Flink JobManagerログとアプリケーションコードをチェックして、 mainメソッドでこの遅延が予想されるかどうかを確認することをお勧めします。そうでない場合は、5 分以内に完了するように問題に対処する必要があります。

「ListApplications」または「DescribeApplication」のアクションを使用して申請状況を確認できます。

スナップショットの作成が失敗する

Apache Flink 用 Managed Serviceは、以下の状況ではスナップショットを作成できません。

- アプリケーションがスナップショットの制限を超えました。スナップショットの上限は 1,000 件です。詳細については、「スナップショット」を参照してください。

- アプリケーションには、ソースまたはシンクにアクセスするアクセス許可がありません。
- アプリケーションコードが正しく機能していない。
- アプリケーションには他の構成上の問題も発生しています。

アプリケーションの更新中にスナップショットを作成しているとき、またはアプリケーションを停止しているときに例外が発生した場合は、アプリケーションの「ApplicationSnapshotConfiguration」の SnapshotsEnabled プロパティを false に設定して、リクエストを再試行してください。

アプリケーションのオペレータが適切にプロビジョニングされていないと、スナップショットが失敗する可能性があります。オペレータのパフォーマンスのチューニングについては、 オペレータースケーリング を参照してください。

アプリケーションが正常な状態に戻ったら、アプリケーションの SnapshotsEnabled プロパティを true に設定することをお勧めします。

VPC 内のリソースにアクセスできない

アプリケーションが Amazon VPC 上で実行されている VPC を使用している場合は、以下を実行して、アプリケーションがそのリソースにアクセスできることを確認します。

- CloudWatch ログで次のエラーを確認します。このエラーは、アプリケーションが VPC 内のリソースにアクセスできないことを示しています。

  ```
  org.apache.kafka.common.errors.TimeoutException: Failed to update metadata after
   60000 ms.
  ```

  このエラーが表示された場合は、ルートテーブルが正しく設定されていることと、コネクタの接続設定が正しいことを確認してください。

  CloudWatch ログの設定と分析の詳細については、「」を参照してくださいロギングとモニタリング。

Amazon S3 バケットへの書き込み時にデータが失われる

Apache Flink バージョン 1.6.2 を使用して Amazon S3 バケットに出力を書き込むと、一部のデータが失われる可能性があります。Amazon S3 を直接出力に使用する場合は、サポートされている最新バージョンの Apache Flink を使用することをお勧めします。Apache Flink 1.6.2 を使用して Amazon

S3 バケットに書き込むには、Firehose を使用することをお勧めします。Apache Flink 用 Managed Service で Firehose を使用する方法の詳細については、「」を参照してくださいFirehose シンク。

アプリケーションは RUNNING ステータスですが、データを処理していません

アプリケーションのステータスは、「ListApplications」または「DescribeApplication」のアクションを使用して確認できます。アプリケーションが RUNNINGステータスになってもシンクにデータを書き込んでいない場合は、Amazon CloudWatch ログストリームをアプリケーションに追加することで問題をトラブルシューティングできます。詳細については、「アプリケーションのCloudWatch ログ記録オプションの使用」を参照してください。ログストリームには、アプリケーションの問題のトラブルシューティングに使用できるメッセージが含まれています。

スナップショット、アプリケーション更新、またはアプリケーション停止エラー: InvalidApplicationConfigurationException

スナップショット操作中、またはアプリケーションの更新や停止など、スナップショットを作成する操作中に、次のようなエラーが発生することがあります。

```
An error occurred (InvalidApplicationConfigurationException) when calling the
 UpdateApplication operation:

Failed to take snapshot for the application xxxx at this moment. The application is
 currently experiencing downtime.
Please check the application's CloudWatch metrics or CloudWatch logs for any possible
 errors and retry the request.
You can also retry the request after disabling the snapshots in the Managed Service for
 Apache Flink console or by updating
the ApplicationSnapshotConfiguration through the AWS SDK
```

このエラーは、アプリケーションがスナップショットを作成できない場合に発生します。

スナップショット操作またはスナップショットを作成する操作中にこのエラーが発生した場合は、次の操作を実行してください。

- アプリケーションのスナップショットを無効にします。これは、Managed Service for Apache Flink コンソールで、または UpdateApplicationアクションの SnapshotsEnabledUpdateパラメータを使用して実行できます。
- スナップショットを作成できない理由を調べてください。詳細については、「アプリケーションが一時的なステータスでスタックしている」を参照してください。
- アプリケーションが正常な状態に戻ったら、スナップショットを再度有効にします。

java.nio.file.NoSuchFileException: /usr/local/openjdk-8/lib/security/cacerts

SSL トラストストアの場所は以前のデプロイで更新されました。代わりに、以下の値を `ssl.truststore.location` パラメータに使用します

```
/usr/lib/jvm/java-11-amazon-corretto/lib/security/cacerts
```

アプリケーションを再起動中

アプリケーションが正常でない場合、その Apache Flink ジョブは継続的に失敗し、再起動します。このセクションでは、この状態の症状とトラブルシューティングの手順について説明します。

症状

この状態では、次の症状が発生する可能性があります。

- `FullRestarts` 指標はゼロではない。このメトリックは、アプリケーションを起動してからアプリケーションのジョブが再開された回数を表します。
- `Downtime` 指標はゼロではない。このメトリクスは、アプリケーションが `FAILING` または `RESTARTING` のステータスにあるミリ秒数を表す。 ステータスにある時間 (ミリ秒) を表します。
- アプリケーションログには、`RESTARTING` または `FAILED` へのステータス変更が含まれます。次の CloudWatch Logs Insights クエリを使用して、これらのステータス変更についてアプリケーションログをクエリできます:. エラーの分析: アプリケーションタスク関連の障害

原因と解決策

次のような状況になると、アプリケーションが不安定になり、再起動を繰り返す可能性があります。

- オペレータが例外を投げている:アプリケーション内のオペレータの例外が処理されない場合、アプリケーションは (オペレータは障害を処理できないと解釈して) フェールオーバーします。「一度だけ」の処理セマンティクスを維持するため、アプリケーションは最新のチェックポイントから再起動します。そのため、この再起動期間中は `Downtime` は 0 ではありません。これを防ぐには、アプリケーションコード内の再試行可能な例外をすべて処理することをお勧めします。

  この状態の原因は、アプリケーションのログをクエリしてアプリケーションの状態が `RUNNING` から `FAILED` に変更されていないかを調べることで調べることができます。詳細については、「the section called “エラーの分析: アプリケーションタスク関連の障害”」を参照してください。

- Kinesis データストリームが適切にプロビジョニングされていない:アプリケーションのソースまたはシンクが Kinesis データストリームの場合は、ストリームのメトリクスにエラーがないか確認してください。`ReadProvisionedThroughputExceeded` `WriteProvisionedThroughputExceeded`

  これらのエラーが表示される場合は、ストリームのシャード数を増やすことで Kinesis ストリームの利用可能なスループットを増やすことができます。詳細については、「Kinesis データストリームで開いているシャードの数を変更するにはどうすればよいですか」を参照してください。

- 「他のソースまたはシンクが適切にプロビジョニングされていないか、使用できない:」アプリケーションがソースとシンクを正しくプロビジョニングしていることを確認してください。アプリケーションで使用されるすべてのソースまたはシンク (AWS 他のサービス、外部のソースや宛先など) が適切にプロビジョニングされているか、読み取り/書き込みスロットリングが行われていないか、定期的に使用できないことを確認します。

  依存するサービスでスループット関連の問題が発生している場合は、それらのサービスが利用できるリソースを増やすか、エラーや利用不能の原因を調査してください。

- 「オペレータが適切にプロビジョニングされていない:」アプリケーション内のいずれかのオペレータのスレッドのワークロードが正しく分散されていないと、オペレータが過負荷になり、アプリケーションがクラッシュする可能性があります。オペレータの並列処理のチューニングについては、オペレータースケーリングの適切な管理 を参照してください。

- アプリケーションの失敗 DaemonException:1.11 以前のバージョンの Apache Flink を使用している場合、このエラーはアプリケーションログに表示されます。0.14 以降の KPL バージョンを使用するには、Apache Flink の新しいバージョンへのアップグレードが必要な場合があります。

- アプリケーションが TimeoutException、 FlinkException、またはで失敗する RemoteTransportException:タスクマネージャーがクラッシュすると、アプリケーションログにこれらのエラーが表示されることがあります。アプリケーションが過負荷になると、タスクマネージャーに CPU やメモリのリソースが圧迫され、タスクマネージャーが機能しなくなる可能性があります。

  これらのエラーは次のようになります。

  - `java.util.concurrent.TimeoutException: The heartbeat of JobManager with id xxx timed out`
  - `org.apache.flink.util.FlinkException: The assigned slot xxx was removed`
  - `org.apache.flink.runtime.io.network.netty.exception.RemoteTransportException: Connection unexpectedly closed by remote task manager`

この問題のトラブルシューティングを行うには、以下を確認します。

- CPU やメモリの使用量が異常に急増していないか、 CloudWatch メトリクスを確認してください。
- アプリケーションにスループットの問題がないかチェックしてください。詳細については、「パフォーマンスのトラブルシューティング」を参照してください。
- アプリケーションログを調べて、アプリケーションコードで発生させている未処理の例外がないか調べてください。

- アプリケーションが JaxbAnnotationModule Not Found エラーで失敗する:このエラーは、アプリケーションが Apache Beam を使用しているが、依存関係や依存バージョンが正しくない場合に発生します。Apache Beam を使用する Apache Flink 用 Managed Serviceアプリケーションでは、以下のバージョンの依存関係を使用する必要があります。

```
<jackson.version>2.10.2</jackson.version>
...
<dependency>
    <groupId>com.fasterxml.jackson.module</groupId>
    <artifactId>jackson-module-jaxb-annotations</artifactId>
    <version>2.10.2</version>
</dependency>
```

jackson-module-jaxb-annotations の正しいバージョンを明示的な依存関係として指定しないと、アプリケーションはそれを環境の依存関係から読み込み、バージョンが一致しないため、実行時にアプリケーションがクラッシュします。

Apache Beam 向けの Apache Flink 用 Managed Service の使用に関する詳細については、Apache Flink CloudFormation 用マネージドサービスとの併用 を参照してください。

- 「アプリケーションが java.io.IOException: ネットワークバッファの数が不十分で失敗する」

アプリケーションに十分なメモリがネットワークバッファに割り当てられていない場合に発生します。ネットワークバッファはサブタスク間の通信を容易にします。ネットワーク経由で送信する前にレコードを保存したり、受信データをレコードに分解してサブタスクに渡す前に保存したりするために使用されます。必要なネットワークバッファの数は、ジョブグラフの並列処理と複雑さに直接影響します。この問題を軽減する方法はいくつかあります。

- parallelismPerKpu を低く設定することで、サブタスクやネットワーク· バッファごとに割り当てられるメモリーを増やすことができます。parallelismPerKpuを下げると KPU が増加

し、したがってコストも増加することに注意してください。これを避けるには、並列処理を同じ係数だけ下げることで、同じ量の KPU を維持できます。

- オペレータの数を減らすか、オペレータをチェーン化して必要なバッファの数を減らすことで、ジョブグラフを簡略化できます。
- それ以外の場合は、https://aws.amazon.com/premiumsupport/ に連絡して、カスタムネットワークバッファー構成を依頼することもできます。

スループットが遅すぎる

アプリケーションが受信ストリーミングデータを十分な速さで処理しないと、パフォーマンスが低下し、不安定になります。このセクションでは、この状態の症状とトラブルシューティングの手順について説明します。

症状

この状態では、次の症状が発生する可能性があります。

- アプリケーションのデータソースが Kinesis ストリームの場合、ストリームの `millisbehindLatest` メトリクスは継続的に増加します。
- アプリケーションのデータソースが Amazon MSK クラスターの場合、クラスターのコンシューマーラグメトリクスは増え続けます。詳細については、「Amazon MSK 開発者ガイド」の「コンシューマーラグモニタリング」を参照してください。
- アプリケーションのデータソースが別のサービスまたはソースである場合は、入手可能なコンシューマーラグのメトリクスまたはデータを確認してください。

原因と解決策

アプリケーションのスループットが遅くなる原因は多数あります。アプリケーションが入力に追いついていない場合は、次の点を確認してください。

- スループットラグが急上昇し、その後次第に減少する場合は、アプリケーションが再起動しているかどうかを確認してください。アプリケーションは再起動中に入力の処理を停止するため、遅延が急増します。アプリケーションの障害については、「アプリケーションを再起動中」を参照してください。
- スループットの遅延がずっと続く場合は、アプリケーションのパフォーマンスが最適化されているかどうかを確認してください。アプリケーションのパフォーマンスを最適化する方法については、パフォーマンスのトラブルシューティング を参照してください。

- スループットラグが急上昇しているのではなく継続的に増加していて、アプリケーションのパフォーマンスが最適化されている場合は、アプリケーションリソースを増やす必要があります。アプリケーションリソースを増やす方法については、 スケーリング を参照してください。
- アプリケーションが別のリージョンの Kafka クラスターから読み取りを行ったり、コンシューマーラグが大きいにもかかわらず `FlinkKafkaConsumer` または `KafkaSource` がほとんどアイドル状態 (高い `idleTimeMsPerSecond` または低い `CPUUtilization`) になっている場合は、2097152 など `receive.buffer.byte` の値を増やすことができます。詳細については、「カスタム MSK 設定」の高レイテンシー環境セクションを参照してください。

アプリケーションソースのスループットが低下したり、コンシューマーラグが増加したりする場合のトラブルシューティング手順については、 パフォーマンスのトラブルシューティング を参照してください。

州の無限の成長

アプリケーションが古い状態情報を適切に処理しないと、情報が継続的に蓄積され、アプリケーションのパフォーマンスや安定性の問題が発生します。このセクションでは、この状態の症状とトラブルシューティングの手順について説明します。

症状

この状態では、次の症状が発生する可能性があります。

- `lastCheckpointDuration` 指標は徐々に増加しているか、急上昇しています。
- `lastCheckpointSize` 指標は徐々に増加しているか、急上昇しています。

原因と解決策

次のような状況では、アプリケーションに状態データが蓄積される可能性があります。

- アプリケーションが必要以上に長く状態データを保持している。
- アプリケーションがウィンドウクエリを使用していて、時間が長すぎる。
- ステートデータに TTL を設定していません。詳細については、Apache Flink ドキュメントの「ステート存続可能時間 (TTL)」を参照してください。
- Apache Beam バージョン 2.25.0 以降に依存するアプリケーションを実行している。主要な実験と価値を拡張することで、 BeamApplicationProperties新しいバージョンの読み取りトランス

フォームをオプトアウトできます。use_deprecated_read詳細については、Apache Beam Documentation を参照してください。

アプリケーションがステートサイズの拡大に直面することがありますが、これは長期的には持続不可能です (結局、Flink アプリケーションは無期限に実行されます) 。場合によっては、アプリケーションがデータをそのままの状態で保存していて、古い情報を適切にエージングアウトしていないことが原因であることもあります。しかし、Flink が提供できることに対して、単に理不尽な期待が寄せられることもあります。アプリケーションでは、数日から数週間に及ぶ長い時間枠にわたってアグリゲーションを使用することがあります。インクリメンタル· アグリゲーションが可能な場合を除き、Flink AggregateFunctionsはウィンドウ全体のイベントをそのままの状態に保つ必要があります。

さらに、プロセス関数を使用してカスタムオペレータを実装する場合、アプリケーションはビジネスロジックで不要になったデータを状態から削除する必要があります。その場合、state を使うと、time-to-live処理時間に基づいてデータを自動的にエージングアウトさせることができます。Apache Flink のマネージドサービスはインクリメンタルチェックポイントを使用しているため、ステート ttl は 「RocksDB コンパクション」に基づいています。ステートサイズ (チェックポイントサイズで示される) が実際に減少するのを確認できるのは、コンパクション操作が行われた後だけです。特に 200 MB 未満のチェックポイントサイズでは、ステートの有効期限が切れることによってチェックポイントのサイズが減少することはほとんどありません。ただし、セーブポイントは古いデータを含まない状態のクリーンコピーに基づいているため、Apache Flink 用 Managed Serviceでスナップショットをトリガーして、古い状態を強制的に削除できます。

デバッグ目的では、チェックポイントのサイズが実際に減少または安定することをより迅速に検証する (そして ROCKSBS でのコンパクションの影響を避ける) ために、インクリメンタルチェックポイントを無効にするのが理にかなっています。ただし、これにはサービスチームへのチケットが必要です。

I/O バウンドオペレーター

データパス上の外部システムへの依存を避けた方がいいです。個々のイベントを充実させるために外部システムに問い合わせるよりも、参照データセットを状態にしておく方がはるかにパフォーマンスが高くなることが多いです。ただし、Amazon Sagemaker でホストされている機械学習モデルでイベントを充実させたい場合など、状態に簡単に移行できない依存関係がある場合もあります。

ネットワークを介して外部システムとやり取りするオペレーターはボトルネックになり、バックプレッシャーの原因となる可能性があります。機能の実装には「AsyncIO」を使用することを強くお勧

めします。これにより、個々の呼び出しの待ち時間を短縮し、アプリケーション全体の処理速度が低下するのを防ぐことができます。

さらに、I/O バウンド演算子を使用するアプリケーションでは、Apache Flink 用マネージドサービスアプリケーションの ParallelismPerKPU 設定を増やすことも理にかなっています。このコンフィギュレーションは、アプリケーションがKPU (Kinesis Processing Unit) ごとに実行できる並列サブタスクの数を記述します。この値をデフォルトの 1 からたとえば 4 に増やすと、アプリケーションは同じリソース (同じコスト) を利用しますが、並列度を 4 倍に拡張できます。これは I/O バインドアプリケーションには有効ですが、I/O バインドでないアプリケーションにはさらなるオーバーヘッドを引き起こします。

Kinesis データストリームからのアップストリームまたはソーススロットリング

症状:アプリケーションがアップストリームのソース Kinesis `LimitExceededExceptions` データストリームから受信しています。

「考えられる原因」:Apache Flink ライブラリ Kinesis コネクタのデフォルト設定は、Kinesis データ・ストリーム・ソースから読み込むように設定されており、`GetRecords` 呼び出しごとにフェッチされるレコードの最大数は非常にアグレッシブなデフォルト設定になっています。Apache Flink はデフォルトで 1 `GetRecords` 回の呼び出しで 10,000 レコードを取得するように設定されています (この呼び出しはデフォルトで 200 ミリ秒ごとに行われます)。ただし、シャードあたりの制限は 1,000 レコードのみです。

このデフォルトの動作により、Kinesis データストリームからデータを使用しようとするとスロットリングが発生し、アプリケーションのパフォーマンスと安定性に影響する可能性があります。

メトリクスを確認して、 CloudWatch `ReadProvisionedThroughputExceeded`このメトリクスがゼロより大きい期間が長時間または持続していることを確認することで、これを確認できます。

また、Apache Flink 用の Amazon CloudWatch マネージドサービスのログでも、エラーが続いていることを確認できます。`LimitExceededException`

解決策:このシナリオを解決するには、次の 2 つの方法のいずれかを実行できます。

- 1 `GetRecords` 回の呼び出しで取得されるレコード数のデフォルト制限を下げてください。
- Apache Flink 用 Amazon マネージドサービスアプリケーションでアダプティブリードを有効にします。アダプティブリード機能の詳細については、「SHARD_USE_ADAPTIVE_READS」を参照してください。

チェックポイント

チェックポイントは、アプリケーションの状態をフォールトトレラントに保つための Flink のメカニズムです。このメカニズムにより、ジョブが失敗した場合に Flink はオペレーターの状態を回復でき、アプリケーションには障害のない実行と同じセマンティクスが与えられます。Apache Flink 用 Managed Serviceでは、アプリケーションの状態は RocksDB に保存されます。RocksDB は組み込みのキー/バリューストアで、動作状態をディスク上に保持します。チェックポイントを取得すると、その状態は Amazon S3 にもアップロードされるため、ディスクが失われた場合でも、チェックポイントを使用してアプリケーションの状態を復元できます。

詳細については、「状態スナップショットの仕組み」を参照してください。

チェックポイントステージ

Flink のチェックポインティングオペレーターサブタスクには、主に 5 つのステージがあります。

- Waiting 「Start Delay」— Flink はストリームに挿入されたチェックポイントバリアを使用するため、このステージの時間はオペレータがチェックポイントバリアに到達するのを待つ時間です。
- アライメント「Alignment Duration」 — この段階では、サブタスクは1つのバリアに達しましたが、他の入力ストリームからのバリアを待っています。
- 同期チェックポイント 「同期時間」 — この段階は、サブタスクが実際にオペレータの状態のスナップショットを撮り、サブタスク上の他のすべてのアクティビティをブロックする段階です。
- 非同期チェックポイント 「非同期時間」 — この段階の大部分は、Amazon S3 に状態をアップロードするサブタスクです。この段階では、サブタスクはブロックされなくなり、レコードを処理できるようになります。
- 承認 — 通常、これは短いステージであり、単にサブタスクが に確認を送信 JobManager し、コミットメッセージ (Kafka シンクなど) を実行するだけです。

これらの各段階 (確認は除く) は、Flink WebUIから入手できるチェックポイントの期間メトリックに対応しており、チェックポイントが長くなる原因の特定に役立ちます。

チェックポイントで利用できる各メトリックの正確な定義を確認するには、「履歴タブ」を参照してください。

調査中

長いチェックポイント期間を調査する場合、決定すべき最も重要なのはチェックポイントのボトルネック、つまりどのオペレーターとサブタスクがチェックポイントに最も時間がかかっているのか、

そのサブタスクのどの段階に長時間かかっているのかを判断することです。これは、ジョブチェックポイントタスクの Flink WebUI を使用して確認できます。Flink の Web インターフェースには、チェックポイントの問題の調査に役立つデータや情報が表示されます。詳細については、「チェックポイントの監視」を参照してください。

まず、Job グラフ内の各オペレータの「エンドツーエンド期間」を確認して、どのオペレータがチェックポイントに時間がかかっているかを判断し、さらに調査する必要があります。Flink のドキュメントによると、所要時間の定義は次のとおりです。

「トリガーのタイムスタンプから最新の確認応答までの期間 (確認応答をまだ受け取っていない場合はn/a) 。」 チェックポイントが完了するまでのこの終了までの期間は、チェックポイントを確認した最後のサブタスクによって決まります。「通常、この時間は 1 つのサブタスクが実際に状態をチェックポイントするのに必要な時間よりも長くなります。」

チェックポイントの他の時間でも、その時間がどこに費やされているかについて、より詳細な情報が得られます。

「Sync Duration」の値が大きい場合は、スナップショット作成中に何かが発生していることを示しています。この段階では `snapshotState()` がSnapshotState インターフェースを実装するクラスが呼び出されます。これはユーザーコードである可能性があるため、スレッドダンプはこれを調査するのに役立ちます。

「非同期時間」が長い場合は、Amazon S3 への状態のアップロードに多くの時間が費やされていると考えられます。これは、状態が大きい場合や、アップロードされる状態ファイルが多数ある場合に発生する可能性があります。このような場合は、アプリケーションがどのように状態を使用しているかを調べ、可能な限り Flink のネイティブデータ構造が使用されていることを確認する必要があります (「Using Keyed State」)。Apache Flink 用 Managed Service では、Amazon S3 の呼び出し回数を最小限に抑え、時間がかかりすぎないように Flink を設定します。以下は、オペレーターのチェックポイント統計の例です。前述のオペレータチェックポイント統計に比べて、「非同期時間」が比較的長いことがわかります。

SubTasks:

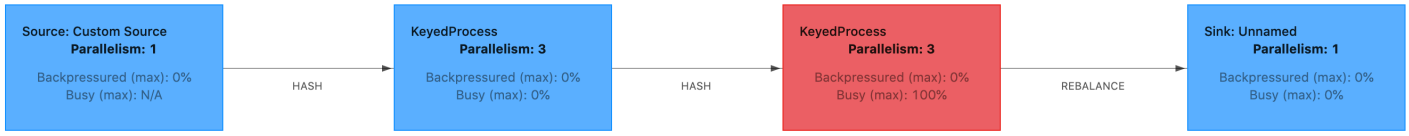
	End to End Duration	Checkpointed Data Size	Sync Duration	Async Duration	Processed (persisted) Data	Alignment Duration	Start Delay
Minimum	495ms	11.1 KB	8ms	357ms	0 B (0 B)	0ms	126ms
Average	813ms	586 KB	28ms	653ms	0 B (0 B)	0ms	126ms
Maximum	1s	1.70 MB	69ms	1s	0 B (0 B)	1ms	128ms

ID	Acknowledged	End to End Duration	Checkpointed Data Size	Sync Duration	Async Duration	Processed (persisted) Data	Alignment Duration	Start Delay	Unaligned Checkpoint
0	2022-03-02 14:16:49	566ms	11.1 KB	8ms	429ms	0 B (0 B)	0ms	126ms	false
1	2022-03-02 14:16:50	1s	1.70 MB	69ms	1s	0 B (0 B)	0ms	128ms	false
2	2022-03-02 14:16:49	495ms	11.1 KB	8ms	357ms	0 B (0 B)	1ms	126ms	false

- Sink: Unnamed 1/1 (100%) 2022-03-02 14:16:49 131ms 0 B 0 B (0 B)

SubTasks:

「Start Delay」の値が大きい場合は、チェックポイントの障壁がオペレータに到達するのを待つ時間の大半が費やされていることがわかります。これは、アプリケーションがレコードを処理するのに時間がかかっていることを示しています。つまり、バリアがジョブグラフ内をゆっくりと流れているということです。これは通常、Job にバックプレッシャーがかかっている場合や、オペレーターが常に忙しい場合に発生します。2 番目の KeyedProcess オペレーター JobGraph がビジー状態の の例を次に示します。



Flink フレームグラフまたは TaskManager スレッドダンプを使用して、何に時間がかかるかを調べることができます。ボトルネックが特定されたら、フレームグラフまたはスレッドダンプを使用してさらに調査できます。

スレッドダンプ

スレッドダンプは、フレームグラフよりもレベルが少し低いもう 1 つのデバッグツールです。スレッドダンプは、ある時点でのすべてのスレッドの実行状態を出力します。Flink は JVM スレッドダンプを受け取ります。これは Flink プロセス内のすべてのスレッドの実行状態です。スレッドの状態は、スレッドのスタックトレースといくつかの追加情報によって示されます。フレームグラフは、実際には複数のスタックトレースを短時間で連続して取得して構築されます。グラフはこれらのトレースを視覚化したもので、一般的なコードパスを簡単に識別できます。

```
"KeyedProcess (1/3)#0" prio=5 Id=1423 RUNNABLE
    at app//scala.collection.immutable.Range.foreach$mVc$sp(Range.scala:154)
    at $line33.$read$$iw$$iw$ExpensiveFunction.processElement(<console>:19)
    at $line33.$read$$iw$$iw$ExpensiveFunction.processElement(<console>:14)
    at app//
org.apache.flink.streaming.api.operators.KeyedProcessOperator.processElement(KeyedProcessOperat
    at app//org.apache.flink.streaming.runtime.tasks.OneInputStreamTask
$StreamTaskNetworkOutput.emitRecord(OneInputStreamTask.java:205)
    at app//
org.apache.flink.streaming.runtime.io.AbstractStreamTaskNetworkInput.processElement(AbstractStr
    at app//
org.apache.flink.streaming.runtime.io.AbstractStreamTaskNetworkInput.emitNext(AbstractStreamTas
    at app//
org.apache.flink.streaming.runtime.io.StreamOneInputProcessor.processInput(StreamOneInputProces
    ...
```

上の図は、Flink UI から取得した単一スレッドのスレッドダンプの抜粋です。1 行目には、このスレッドに関する次のような一般的な情報が含まれています。

- スレッド名 KeyedProcess (1/3)#0
- スレッドの優先度「prio=5」
- 一意のスレッド「Id=1423」
- スレッド状態:「実行可能」

通常、スレッドの名前からそのスレッドの一般的な目的に関する情報が得られます。演算子スレッドは、演算子と同じ名前を持つだけでなく、(KeyedProcess 1/3)#0 スレッドはKeyedProcess演算子からのもので、(3 つのうち) 1 つめのサブタスクからのものであるなど、関連するサブタスクの表示も持つため、演算子スレッドは名前で識別できます。

スレッドは、次に示す状態のいずれかになります。

- NEW — スレッドは作成されましたが、まだ処理されていません。
- RUNNABLE — スレッドは CPU 上で実行されています
- BLOCKED — スレッドは別のスレッドがロックを解放するのを待っている
- WAITING — スレッドは`wait()`、`join()`、または `park()` メソッドを使用して待機している
- TIMED_WAITING — スレッドはスリープ、ウェイト、ジョイン、パークの各メソッドを使用して待機していますが、待機時間は最大です。

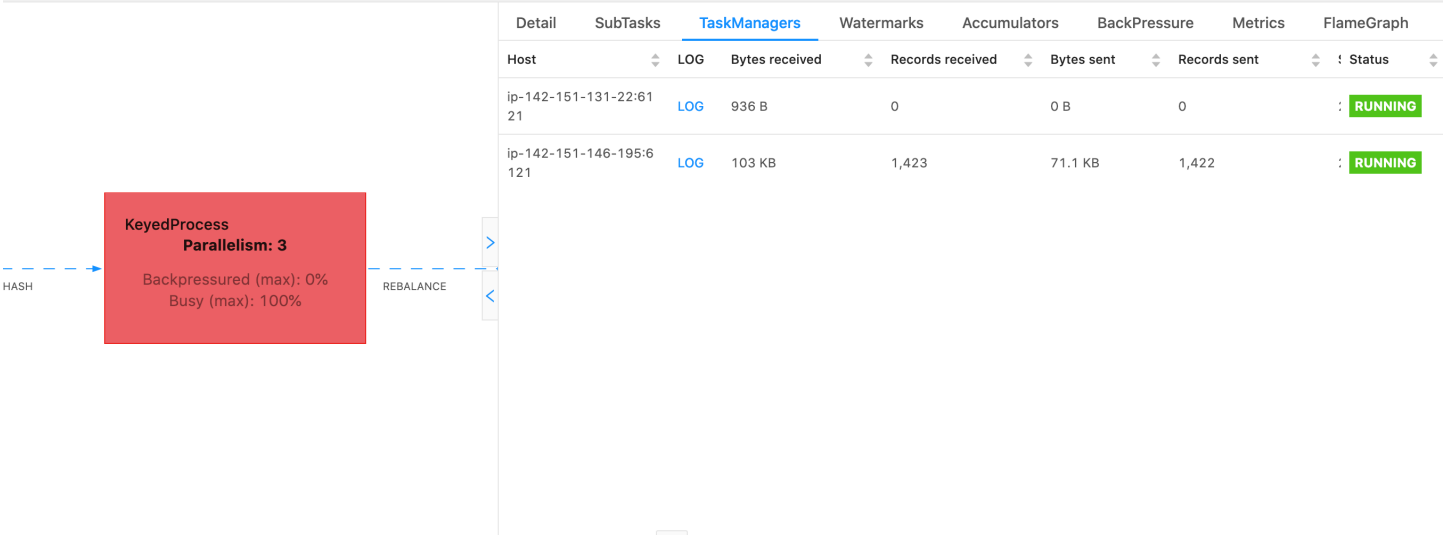
> **Note**
>
> Flink 1.13 では、スレッドダンプ内の 1 つのスタックトレースの最大深度は 8 に制限されています。

> **Note**
>
> スレッドダンプは読み取りが難しく、複数のサンプルを採取して手動で分析する必要があるため、Flink アプリケーションのパフォーマンス問題をデバッグする最後の手段はスレッドダンプです。できる限り、フレームグラフを使用するのが望ましいです。

Flink のスレッドダンプです。

Flink では、Flink UI の左側のナビゲーションバーで「タスクマネージャ」 オプションを選択し、特定のタスクマネージャーを選択して [スレッドダンプ] タブに移動すると、「スレッドダンプ」を実行できます。スレッドダンプは、ダウンロードしたり、お気に入りのテキストエディター (またはスレッドダンプアナライザー) にコピーしたり、Flink Web UIのテキストビュー内で直接分析したりできます (ただし、この最後のオプションは少し扱いにくい場合があります) 。

特定の演算子を選択したときにTaskManagers、タブのスレッドダンプを取るタスクマネージャーを決定するために使用できます。これは、オペレータがオペレータのさまざまなサブタスクで実行されており、異なるタスクマネージャーでも実行できることを示しています。



ダンプは複数のスタックトレースで構成されます。ただし、ダンプを調べるときには、オペレータに関連するものが最も重要です。オペレータースレッドにはオペレータと同じ名前があり、どのサブタスクに関連しているかがわかるので、これらは簡単に見つかります。例えば、次のスタックトレースは KeyedProcess演算子からのもので、最初のサブタスクです。

```
"KeyedProcess (1/3)#0" prio=5 Id=595 RUNNABLE
    at app//scala.collection.immutable.Range.foreach$mVc$sp(Range.scala:155)
    at $line360.$read$$iw$$iw$ExpensiveFunction.processElement(<console>:19)
    at $line360.$read$$iw$$iw$ExpensiveFunction.processElement(<console>:14)
    at app//
org.apache.flink.streaming.api.operators.KeyedProcessOperator.processElement(KeyedProcessOperat
    at app//org.apache.flink.streaming.runtime.tasks.OneInputStreamTask
$StreamTaskNetworkOutput.emitRecord(OneInputStreamTask.java:205)
    at app//
org.apache.flink.streaming.runtime.io.AbstractStreamTaskNetworkInput.processElement(AbstractStr
    at app//
org.apache.flink.streaming.runtime.io.AbstractStreamTaskNetworkInput.emitNext(AbstractStreamTas
    at app//
org.apache.flink.streaming.runtime.io.StreamOneInputProcessor.processInput(StreamOneInputProces
    ...
```

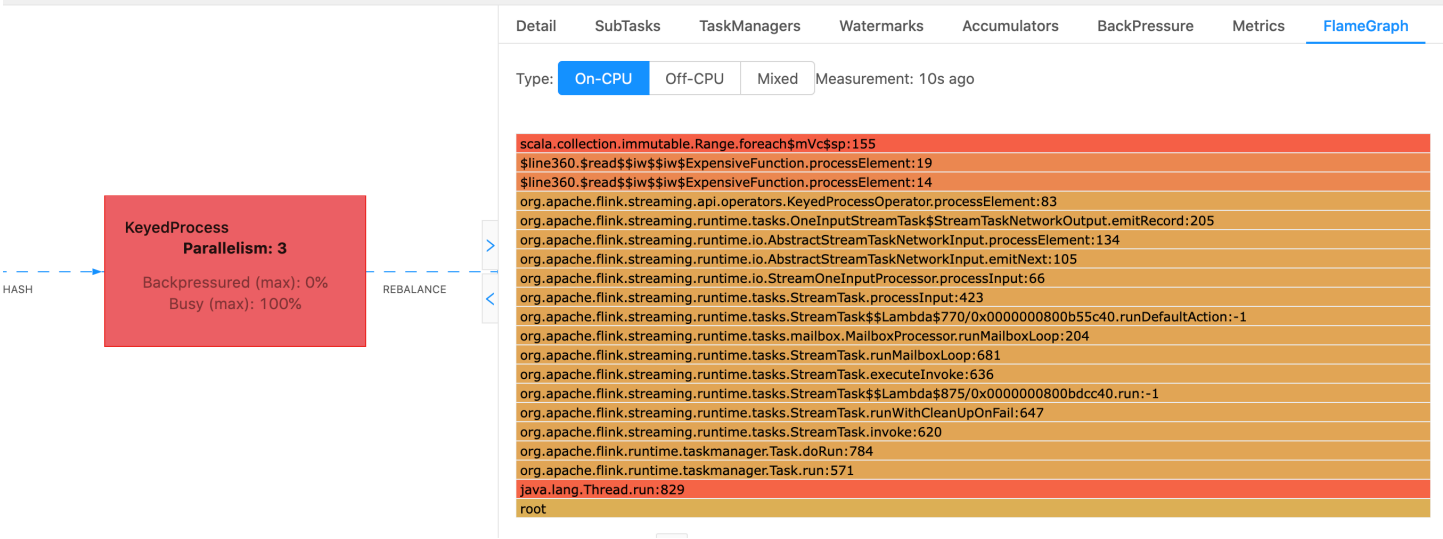
同じ名前のオペレータが複数あると混乱するかもしれませんが、オペレータに名前を付けることでこの問題を回避できます。例:

```
....
.process(new ExpensiveFunction).name("Expensive function")
```

「フレームグラフ」

フレームグラフは、ターゲットコードのスタックトレースを視覚化する便利なデバッグツールです。これにより、最も頻繁に使用されるコードパスを特定できます。スタックトレースを何度もサンプリングして作成されます。フレームグラフの X 軸にはさまざまなスタックプロファイルが表示され、Y 軸にはスタックの深さとスタックトレースの呼び出しが表示されます。フレームグラフの 1 つの長方形はスタックフレームを表し、フレームの幅はスタック内での出現頻度を示します。フレームグラフとその使用方法の詳細については、「フレームグラフ」を参照してください。

Flink では、オペレータを選択してタブを選択することで、オペレータのフレームグラフに Web UI からアクセスできますFlameGraph。十分な数のサンプルが収集されると、フレームグラフが表示されます。以下は、 FlameGraph チェックポイントに多くの時間がかかっ ProcessFunction ていた のです。



これは非常に単純なフレームグラフであり、すべての CPU 時間が ExpensiveFunction オペレータprocessElementの 内の各ルックで費やされていることを示しています。また、コード内のどこで実行されているかを判断するのに役立つ行番号も表示されます。

チェックポイントがタイムアウトしています。

アプリケーションが最適化されていなかったり、適切にプロビジョニングされていなかったりすると、チェックポイントが失敗する可能性があります。このセクションでは、この状態の症状とトラブルシューティングの手順について説明します。

症状

アプリケーションのチェックポイントに障害が発生すると、numberOfFailedCheckpoints が 0 より大きくなります。

チェックポイントが失敗するのは、アプリケーションエラーなどの直接的な障害でも、アプリケーションリソース不足などの一時的な障害でもかまいません。アプリケーションログとメトリクスをチェックして、次の症状がないか調べてください。

- コード内のエラー。
- アプリケーションの依存サービスへのアクセス中にエラーが発生しました。
- データのシリアル化中にエラーが発生しました。デフォルトのシリアライザーがアプリケーションデータをシリアル化できない場合、アプリケーションは失敗します。アプリケーションでカスタムシリアライザーを使用する方法については、Apache Flink ドキュメントの「データ型とシリアル化」を参照してください。
- メモリ不足のエラー
- 以下の指標が急上昇または着実に増加しています。
  - heapMemoryUtilization
  - oldGenerationGCTime
  - oldGenerationGCCount
  - lastCheckpointSize
  - lastCheckpointDuration

チェックポイントのモニタリングの詳細については、Apache Flink ドキュメントの「チェックポイントのモニタリング」を参照してください。

原因と解決策

アプリケーションログのエラーメッセージには、直接的な障害の原因が示されます。一時的な障害には以下の原因が考えられます。

- アプリケーションの KPU プロビジョニングが不十分。アプリケーションのプロビジョニングを引き上げる方法については、 スケーリング を参照してください。
- アプリケーションの状態サイズが大きすぎる。lastCheckpointSize メトリクスを使用してアプリケーションの状態サイズを監視できます。

- アプリケーションの状態データはキー間で不均等に分散されます。アプリケーションで KeyBy オペレータを使用する場合は、受信データがキー間で均等に分割されていることを確認してください。ほとんどのデータが 1 つのキーに割り当てられていると、障害の原因となるボトルネックになります。
- アプリケーションにメモリやガベージコレクションのバックプレッシャが発生しています。アプリケーションのheapMemoryUtilization、oldGenerationGCTime、oldGenerationGCCountの値が急上昇していないか、または着実に増加していないかを監視します。

Apache Beam アプリケーションのチェックポイント障害

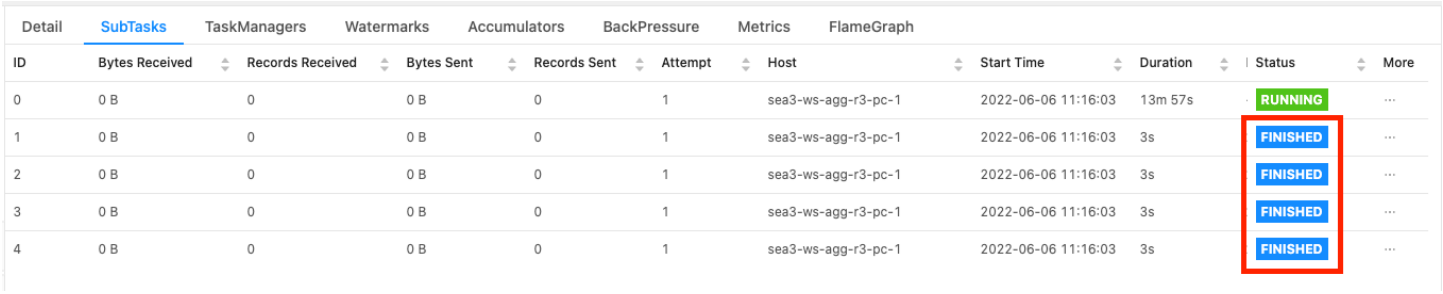
Beam アプリケーションが 0 ミリ秒にshutdownSourcesAfterIdleMs設定されている場合、タスクが「FINISHED」状態であるため、チェックポイントのトリガーに失敗することがあります。このセクションでは、この状態の症状と解決策について説明します。

症状

Managed Service for Apache Flink アプリケーション CloudWatch ログに移動し、次のログメッセージがログに記録されているかどうかを確認します。次のログメッセージは、一部のタスクが完了したためにチェックポイントがトリガーされなかったことを示しています。

```
                {
                "locationInformation":
"org.apache.flink.runtime.checkpoint.CheckpointCoordinator.onTriggerFailure(CheckpointCoordina
                "logger": "org.apache.flink.runtime.checkpoint.CheckpointCoordinator",
                "message": "Failed to trigger checkpoint for job your job ID since some
tasks of job your job ID has been finished, abort the checkpoint Failure reason: Not
all required tasks are currently running.",
                "threadName": "Checkpoint Timer",
                "applicationARN": your application ARN,
                "applicationVersionId": "5",
                "messageSchemaVersion": "1",
                "messageType": "INFO"
                }
```

一部のタスクが「FINISHED」状態になり、チェックポイントを設定できなくなった Flink ダッシュボードでも確認できます。



原因

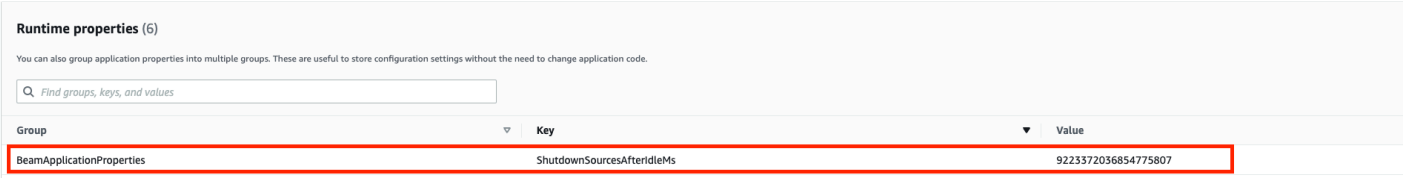
shutdownSourcesAfterIdleMs は、設定されたミリ秒の時間アイドル状態であったソースをシャットダウンする Beam 設定変数です。ソースがシャットダウンされると、チェックポイント設定はできなくなります。これにより、「チェックポイント障害」が発生する可能性があります。

タスクが「FINISHED」状態になる原因の 1 つは、 shutdownSourcesAfterIdleMs が 0 ミリ秒に設定されている場合です。つまり、アイドル状態のタスクはすぐにシャットダウンされます。

ソリューション

タスクがすぐに「FINISHED」状態にならないようにするには、 shutdownSourcesAfterIdleMs を Long.MAX_VALUE に設定します。これには 2 つの方法で実行できます。

- オプション 1: Managed Service for Apache Flink アプリケーション設定ページでビーム設定が設定されている場合は、次のように新しいキーと値のペアを追加して shutdpwnSourcesAfteridleMs を設定できます。



- オプション 2: ビーム設定が JAR ファイルに設定されている場合は、shutdownSourcesAfterIdleMs 次のように を設定できます。

```
                        FlinkPipelineOptions options =
PipelineOptionsFactory.create().as(FlinkPipelineOptions.class); // Initialize Beam
Options object

                        options.setShutdownSourcesAfterIdleMs(Long.MAX_VALUE); // set
shutdownSourcesAfterIdleMs to Long.MAX_VALUE
```

```
                        options.setRunner(FlinkRunner.class);

                        Pipeline p = Pipeline.create(options); // attach specified
options to Beam pipeline
```

バックプレッシャー

Flink はバックプレッシャーを使用して個々のオペレーターの処理速度を調整します。

オペレーターは、さまざまな理由から、受信したメッセージ量の処理を続けるのに苦労することがあります。操作には、オペレータが使用できる量よりも多くの CPU リソースが必要となる場合があります。オペレータは I/O 操作が完了するまで待つ場合があります。オペレータがイベントを十分な速さで処理できない場合、処理速度が遅いオペレータに供給する上流のオペレータにバックプレッシャーがかかります。これにより、アップストリームのオペレーターの速度が低下し、バックプレッシャーがソースにさらに伝わり、ソースも速度を落とすことでアプリケーション全体のスループットに適応するようになります。バックプレッシャーとその仕組みについて詳しくは、「Apache Flink™ がバックプレッシャーを処理する方法」を参照してください。

アプリケーション内のどのオペレータが遅いのかを知ることで、アプリケーションのパフォーマンス問題の根本原因を理解するための重要な情報を得ることができます。バックプレッシャ情報は「Flink ダッシュボードを通じて公開されます」。処理速度が遅いオペレータを特定するには、シンクに最も近い背圧値が高いオペレータ (次の例ではオペレータ B) を探します。その場合、速度低下の原因となっているオペレータは、ダウンストリームのオペレータの 1 つ (この例ではオペレータ C) です。B はイベントをより速く処理できますが、実際の処理速度が遅いオペレータ C に出力を転送できないため、バックプレッシャーがかかっています。

```
A (backpressured 93%) -> B (backpressured 85%) -> C (backpressured 11%) -> D
 (backpressured 0%)
```

処理が遅いオペレータを特定したら、そのオペレータが遅い理由を理解するように努めてください。理由は無数にあるかもしれませんが、何が問題なのかが明らかではなく、解決までに何日ものデバッグとプロファイリングが必要な場合もあります。以下に、明らかで一般的な理由をいくつか挙げます。その一部を以下で詳しく説明します。

- オペレータが、ネットワークコールなどの低速な I/O を実行している (代わりに AsyncIO の使用を検討してください)。

- データに偏りがあり、1 人のオペレーターが他のオペレーターよりも多くのイベントを受信しています (Flink ダッシュボードの個々のサブタスク (つまり、同じオペレーターのインスタンス) の送受信メッセージ数を確認して確認してください。
- リソースを大量に消費する (データ· スキューがない場合、CPU/メモリ· バウンドの作業ではスケールアウトを、I/Oバウンドの作業では ParallelismPerKPU を増やすことを検討する) 。
- オペレータへの広範囲なロギング (実稼働アプリケーションではロギングを最小限に抑えるか、代わりにデバッグ出力をデータストリームに送信することを検討してください)。

廃棄シンクを使用したスループットのテスト

「Discarding Sink」は、アプリケーションの実行中に受信したすべてのイベントを単に無視します (シンクがないアプリケーションは実行に失敗します)。これは、スループットのテスト、プロファイリング、およびアプリケーションが適切にスケーリングされているかどうかの検証に非常に役立ちます。また、シンクがバックプレッシャーの原因になっているのか、それともアプリケーションなのかを検証するための、非常に実用的なサニティチェックでもあります (ただし、バックプレッシャのメトリクスをチェックするだけでも簡単でわかりやすい場合が多いです)。

アプリケーションのすべてのシンクを廃棄シンクに置き換え、本番データに似たデータを生成するモックソースを作成することで、特定の並列度設定におけるアプリケーションの最大スループットを測定できます。さらに、並列度を増やして、アプリケーションが適切にスケーリングされ、スループットが高くなると (データスキューなどにより) 発生するボトルネックがないことを確認できます。

データスキュー機能

Flink アプリケーションはクラスター上で分散的に実行されます。Flink は複数のノードにスケールアウトするために、キー付きストリームの概念を採用しています。つまり、ストリームのイベントは、顧客 ID などの特定のキーに従って分割され、Flink はノードごとに異なるパーティションを処理できるということです。その後、「キー付きウィンドウ」、「プロセス関数」、「非同期 I/O」など、多くの Flink オペレータがこれらのパーティションに基づいて評価されます。

パーティションキーの選択は、ビジネスロジックによって決まることがよくあります。同時に、「DynamoDB」や Spark などのベストプラクティスの多くが Flink にも同様に適用されます。たとえば、次のようなものがあります。

- パーティションキーのカーディナリティを高く保つこと
- パーティション間のイベントボリュームの偏りを回避

Flink ダッシュボードでサブタスク (つまり、同じオペレータのインスタンス) の送受信レコードを比較することで、パーティション内のスキューを特定できます。さらに、Apache Flink 用 Managed Service モニタリングでは、numRecordsIn/Out と numRecordsInPerSecond/OutPerSecond のメトリクスをサブタスク・レベルでも公開するように設定できます。

ステートスキュー機能

ステートフルオペレータ、つまりウィンドウなどのビジネスロジックの状態を維持するオペレータの場合、データスキューは常にステートスキューにつながります。サブタスクの中には、データに偏りがあるために他のサブタスクよりも多くのイベントを受け取り、そのため状態を維持するデータも多くなるものがあります。ただし、パーティションのバランスが均等なアプリケーションでも、その状態で保持されるデータの量には偏りがある可能性があります。たとえば、セッションウィンドウでは、一部のユーザーとセッションがそれぞれ他のユーザーよりもずっと長くなることがあります。長いセッションが同じパーティションに属していると、同じオペレーターの異なるサブタスクが保持するステートサイズのバランスが崩れてしまう可能性があります。

ステートスキューは、個々のサブタスクに必要なメモリとディスクリソースを増やすだけでなく、アプリケーション全体のパフォーマンスを低下させる可能性もあります。アプリケーションがチェックポイントまたはセーブポイントを取得しているとき、オペレータの状態は Amazon S3 に保持され、ノードまたはクラスターの障害から状態を保護します。このプロセスの間 (特に Apache Flink 用 Managed Serviceでデフォルトで有効になっている 1 回限りのセマンティクスの場合)、チェックポイント/セーブポイントが完了するまで、外部から処理が停止します。データに偏りがある場合、操作を完了するまでの時間は、特に大量の状態を蓄積した 1 つのサブタスクによって制限される可能性があります。極端なケースでは、1 つのサブタスクが状態を維持できないことが原因で、チェックポイントやセーブポイントの取得に失敗することがあります。

データスキューと同様に、ステートスキューはアプリケーションの処理速度を大幅に低下させる可能性があります。

ステートスキューを特定するには、Flink ダッシュボードを活用できます。最近のチェックポイントまたはセーブポイントを見つけて、詳細内の個々のサブタスクに保存されているデータ量を比較します。

異なるリージョンのリソースとの統合

Flink 設定のクロスリージョンレプリケーションに必要な設定により、Apache Flink アプリケーション用 Managed Serviceとは異なるリージョンの Amazon S3 バケットへの書き込みに StreamingFileSink を使用できるようになります。そのためには、「AWS Support Center」にサポートチケットを提出してください。

Apache Flink 用 Amazon マネージドサービスのドキュメント履歴

次の表は、 Apache Flink 用 Managed Service の前回のリリースからの重要な変更を示しています。

- API version: 2018-05-23
- ドキュメント最終更新日: 2023 年 8 月 30 日

変更	説明	日付
Kinesis Data Analytics は Apache Flink 用 Managed Service として知られています。	サービスエンドポイント、API、コマンドラインインターフェイス、IAM アクセスポリシー、 CloudWatch メトリックス、請求ダッシュボードに変更はありません。 AWS 既存のアプリケーションは、以前と同じように動作します。詳細については、「Apache Flink 用 Managed Service とは?」を参照してください。	2023 年 8 月 30 日
Apache Flink バージョン 1.15.2 をサポート	Apache Flink 用 Managed Service が Apache Flink バージョン 1.15.2 を使用するアプリケーションをサポートするようになりました。Apache Flink テーブル API を使用して Kinesis Data Analytics アプリケーションを作成します。詳細については、 アプリケーションの作成 を参照してください。	2022 年 11 月 22 日

変更	説明	日付
Apache Flink バージョン 1.13.2 をサポート	Managed Service for Apache Flink が Apache Flink バージョン 1.13.2 を使用するアプリケーションをサポートするようになりました。Apache Flink テーブル API を使用して Kinesis Data Analytics アプリケーションを作成します。詳細については、「入門:Flink 1.13.2」を参照してください。	2021 年 10 月 13 日
Python のサポート	Apache Flink 用 Managed Service が Apache Flink テーブル API と SQL で Python を使用するアプリケーションをサポートするようになりました。詳細については、「Pythonの使用」を参照してください。	2021 年 3 月 25 日
Apache Flink 1.11.1 のサポート	Apache Flink アプリケーション用 Managed Service が Apache Flink 1.11.1 を使用するアプリケーションをサポートするようになりました。Apache Flink テーブル API を使用して Kinesis Data Analytics アプリケーションを作成します。詳細については、「アプリケーションの作成」を参照してください。	2020 年 11 月 19 日

変更	説明	日付
Apache Flink Dashboard	Apache Flink Dashboardを使用して、アプリケーションの状態とパフォーマンスを監視します。詳細については、「Apache Flink Dashboard」を参照してください。	2020 年 11 月 19 日
EFO Consumer	拡張ファンアウト (EFO) コンシューマーを使用して Kinesis データストリームから読み取るアプリケーションを作成します。詳細については、「EFO Consumer」を参照してください。	2020 年 10 月 6 日
Apache Beam	Apache Beam を使用してストリーミングデータを処理するアプリケーションを作成します。詳細については、「Apache Flink CloudFormation 用マネージドサービスとの併用」を参照してください。	2020 年 9 月 15 日
パフォーマンス	アプリケーションのパフォーマンスに関する問題のトラブルシューティング方法と、パフォーマンスの高いアプリケーションの作成方法。詳細については、「パフォーマンス」を参照してください。	2020 年 7 月 21 日

変更	説明	日付
Custom Keystore	転送中の暗号化にカスタムキーストアを使用する Amazon MSK クラスターにアクセスする方法。詳細については、「カスタムトラストストア」を参照してください。	2020 年 6 月 10 日
CloudWatch アラーム	Apache Flink CloudWatch 用マネージドサービスでアラームを作成する際の推奨事項。詳細については、「アラーム」を参照してください。	2020 年 6 月 5 日
CloudWatch 新しいメトリクス	Apache Flink のマネージドサービスが Amazon メトリックスに 22 個のメトリクスを送信するようになりました。CloudWatch 詳細については、「Managed Service for Apache Flink のメトリクスとディメンション」を参照してください。	2020 年 5 月 12 日
CloudWatch カスタムメトリックス	アプリケーション固有のメトリックスを定義し、Amazon Metrics に送信します。CloudWatch 詳細については、「カスタムメトリクス」を参照してください。	2020 年 5 月 12 日

変更	説明	日付
「例: 異なるアカウントで Kinesis Stream から読み取る」	Apache Flink AWS 用マネージドサービスアプリケーションの別のアカウントで Kinesis ストリームにアクセスする方法について説明します。詳細については、「クロスアカウント」を参照してください。	2020 年 3 月 30 日
Apache Flink 1.8.2 のサポート	Apache Flink アプリケーション用 Managed Service が Apache Flink 1.8.2 を使用するアプリケーションをサポートするようになりました。Flink StreamingFileSink コネクタを使用して出力を S3 に直接書き込みます。詳細については、「アプリケーションの作成」を参照してください。	2019 年 12 月 17 日
Managed Service for Apache Flink VPC	Apache Flink アプリケーション用 Managed Service を仮想プライベートクラウドに接続するように構成します。詳細については、「Amazon VPC の使用」を参照してください。	2019 年 11 月 25 日
Managed Service for Apache Flink ベストプラクティス	Apache Flink アプリケーション用 Managed Service アプリケーションを作成および管理するためのベストプラクティス 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。	2019 年 10 月 14 日

変更	説明	日付
Apache Flink アプリケーション用 Managed Service のアプリケーションログの分析	CloudWatch ログインサイトを使用して Apache Flink 用マネージドサービスアプリケーションを監視します。詳細については、「ログの分析」を参照してください。	2019 年 6 月 26 日
Apache Flink アプリケーションランタイムプロパティ用 Managed Service	Apache Flink 用 Managed Service でランタイムプロパティを操作します。詳細については、「ランタイムプロパティ」を参照してください。	2019 年 6 月 24 日
Apache Flink アプリケーション用マネージドサービスのタグ付け	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、「タグ付けを使用する」を参照してください。	2019 年 5 月 8 日
での Apache Flink API 呼び出しのロギングマネージドサービス AWS CloudTrail	Apache Flink 用マネージドサービスは AWS CloudTrail、Apache Flink 用マネージドサービスのユーザ、ロール、AWS またはサービスが実行したアクションの記録を提供するサービス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

変更	説明	日付
アプリケーションの作成 (Firehose シンク)	Amazon Kinesis データストリームをソースとして、Amazon Data Firehose ストリームをシンクとして使用して、Apache Flink 用のマネージドサービスを作成するための演習を行います。詳細については、「Firehose シンク」を参照してください。	2018 年 12 月 13 日
パブリックリリース	これは「Java アプリケーション向け Apache Flink 用 Managed Service 開発者ガイド」のイニシャルリリースです。	2018 年 11 月 27 日

Apache Flink API のマネージドサービスのサンプルコード

このトピックには、Apache Flink 用 Managed Service アクションのリクエストブロックの例が含まれています。

AWS Command Line Interface (AWS CLI) のアクションの入力として JSON を使用するには、リクエストを JSON ファイルに保存します。次に、 `--cli-input-json` パラメータを使用してファイル名をアクションに渡します。

次の例は、アクションを備えた JSON ファイルを使用する方法を示しています。

```
$ aws kinesisanalyticsv2 start-application --cli-input-json file://start.json
```

で JSON を使用する方法の詳細については AWS CLI、『AWS Command Line Interface ユーザガイド』の「CLI スケルトンの生成」と「CLI 入力 JSON パラメータの生成」を参照してください。

トピック

- AddApplicationCloudWatchLoggingOption
- AddApplicationInput
- AddApplicationInputProcessingConfiguration
- AddApplicationOutput
- AddApplicationReferenceDataSource
- AddApplicationVpcConfiguration
- CreateApplication
- CreateApplicationSnapshot
- DeleteApplication
- DeleteApplicationCloudWatchLoggingOption
- DeleteApplicationInputProcessingConfiguration
- DeleteApplicationOutput
- DeleteApplicationReferenceDataSource
- DeleteApplicationSnapshot
- DeleteApplicationVpcConfiguration
- DescribeApplication

- DescribeApplicationSnapshot
- DiscoverInputSchema
- ListApplications
- ListApplicationSnapshots
- StartApplication
- StopApplication
- UpdateApplication

AddApplicationCloudWatchLoggingOption

AddApplicationCloudWatchLoggingOption以下のアクションのリクエストコード例は、Apache Flink 用マネージドサービスアプリケーションに Amazon CloudWatch ロギングオプションを追加します。

```
{
    "ApplicationName": "MyApplication",
    "CloudWatchLoggingOption": {
        "LogStreamARN": "arn:aws:logs:us-east-1:123456789123:log-group:my-log-
group:log-stream:My-LogStream"
    },
    "CurrentApplicationVersionId": 2
}
```

AddApplicationInput

AddApplicationInput以下のアクションのリクエストコード例は、Managed Service for Apache Flink アプリケーションにアプリケーション入力を追加します。

```
{
   "ApplicationName": "MyApplication",
   "CurrentApplicationVersionId": 2,
   "Input": {
      "InputParallelism": {
         "Count": 2
      },
      "InputSchema": {
         "RecordColumns": [
            {
               "Mapping": "$.TICKER",
```

```
                "Name": "TICKER_SYMBOL",
                "SqlType": "VARCHAR(50)"
            },
            {
                "SqlType": "REAL",
                "Name": "PRICE",
                "Mapping": "$.PRICE"
            }
         ],
         "RecordEncoding": "UTF-8",
         "RecordFormat": {
            "MappingParameters": {
               "JSONMappingParameters": {
                  "RecordRowPath": "$"
               }
            },
            "RecordFormatType": "JSON"
         }
      },
      "KinesisStreamsInput": {
         "ResourceARN": "arn:aws:kinesis:us-east-1:012345678901:stream/
ExampleInputStream"
      }
   }
}
```

AddApplicationInputProcessingConfiguration

AddApplicationInputProcessingConfiguration以下のアクションのリクエストコード例は、Managed Service for Apache Flink アプリケーションにアプリケーション入力処理設定を追加します。

```
{
   "ApplicationName": "MyApplication",
   "CurrentApplicationVersionId": 2,
   "InputId": "2.1",
   "InputProcessingConfiguration": {
      "InputLambdaProcessor": {
         "ResourceARN": "arn:aws:lambda:us-
east-1:012345678901:function:MyLambdaFunction"
      }
   }
}
```

AddApplicationOutput

AddApplicationOutput以下のアクションのリクエストコード例では、Kinesis データストリームをアプリケーション出力として Apache Flink 用マネージドサービスアプリケーションに追加します。

```
{
   "ApplicationName": "MyApplication",
   "CurrentApplicationVersionId": 2,
   "Output": {
      "DestinationSchema": {
         "RecordFormatType": "JSON"
      },
      "KinesisStreamsOutput": {
         "ResourceARN": "arn:aws:kinesis:us-east-1:012345678901:stream/
ExampleOutputStream"
      },
      "Name": "DESTINATION_SQL_STREAM"
   }
}
```

AddApplicationReferenceDataSource

AddApplicationReferenceDataSource以下のアクションのリクエストコード例では、CSV アプリケーション参照データソースを Managed Service for Apache Flink アプリケーションに追加します。

```
{
   "ApplicationName": "MyApplication",
   "CurrentApplicationVersionId": 5,
   "ReferenceDataSource": {
      "ReferenceSchema": {
         "RecordColumns": [
            {
               "Mapping": "$.TICKER",
               "Name": "TICKER",
               "SqlType": "VARCHAR(4)"
            },
            {
               "Mapping": "$.COMPANYNAME",
               "Name": "COMPANY_NAME",
               "SqlType": "VARCHAR(40)"
            },
         ],
```

```
         "RecordEncoding": "UTF-8",
         "RecordFormat": {
            "MappingParameters": {
               "CSVMappingParameters": {
                  "RecordColumnDelimiter": " ",
                  "RecordRowDelimiter": "\r\n"
               }
            },
            "RecordFormatType": "CSV"
         }
      },
      "S3ReferenceDataSource": {
         "BucketARN": "arn:aws:s3:::MyS3Bucket",
         "FileKey": "TickerReference.csv"
      },
      "TableName": "string"
   }
}
```

AddApplicationVpcConfiguration

AddApplicationVpcConfiguration以下のアクションのリクエストコード例は、既存のアプリケーションに VPC 設定を追加します。

```
{
   "ApplicationName": "MyApplication",
   "CurrentApplicationVersionId": 9,
   "VpcConfiguration": {
      "SecurityGroupIds": [ "sg-0123456789abcdef0" ],
      "SubnetIds": [ "subnet-0123456789abcdef0" ]
   }
}
```

CreateApplication

CreateApplication以下のアクションのリクエストコード例では、Apache Flink 用マネージドサービスアプリケーションを作成します。

```
{
  "ApplicationName":"MyApplication",
```

```
  "ApplicationDescription":"My-Application-Description",
  "RuntimeEnvironment":"FLINK-1_15",
  "ServiceExecutionRole":"arn:aws:iam::123456789123:role/myrole",
  "CloudWatchLoggingOptions":[
    {
      "LogStreamARN":"arn:aws:logs:us-east-1:123456789123:log-group:my-log-group:log-
stream:My-LogStream"
    }
  ],
  "ApplicationConfiguration": {
    "EnvironmentProperties":
      {"PropertyGroups":
        [
          {"PropertyGroupId": "ConsumerConfigProperties",
            "PropertyMap":
              {"aws.region": "us-east-1",
              "flink.stream.initpos": "LATEST"}
          },
          {"PropertyGroupId": "ProducerConfigProperties",
            "PropertyMap":
              {"aws.region": "us-east-1"}
          },
        ]
      },
    "ApplicationCodeConfiguration":{
      "CodeContent":{
        "S3ContentLocation":{
          "BucketARN":"arn:aws:s3:::mybucket",
          "FileKey":"myflink.jar",
          "ObjectVersion":"AbCdEfGhIjKlMnOpQrStUvWxYz12345"
        }
      },
      "CodeContentType":"ZIPFILE"
    },
      "FlinkApplicationConfiguration":{
      "ParallelismConfiguration":{
        "ConfigurationType":"CUSTOM",
        "Parallelism":2,
        "ParallelismPerKPU":1,
        "AutoScalingEnabled":true
      }
    }
  }
```

```
}
```

CreateApplicationSnapshot

CreateApplicationSnapshot以下のアクションのリクエストコード例は、アプリケーションの状態のスナップショットを作成します。

```
{
   "ApplicationName": "MyApplication",
   "SnapshotName": "MySnapshot"
}
```

DeleteApplication

DeleteApplication以下のアクションのリクエストコード例は、Apache Flink 用マネージドサービスアプリケーションを削除します。

```
{"ApplicationName": "MyApplication",
"CreateTimestamp": 12345678912}
```

DeleteApplicationCloudWatchLoggingOption

DeleteApplicationCloudWatchLoggingOption以下のアクションのリクエストコード例は、Apache Flink 用マネージドサービスアプリケーションから Amazon CloudWatch ロギングオプションを削除します。

```
{
    "ApplicationName": "MyApplication",
    "CloudWatchLoggingOptionId": "3.1"
    "CurrentApplicationVersionId": 3
}
```

DeleteApplicationInputProcessingConfiguration

DeleteApplicationInputProcessingConfiguration以下のアクションのリクエストコード例では、Apache Flink 用マネージドサービスアプリケーションから入力処理設定を削除します。

```
{
   "ApplicationName": "MyApplication",
   "CurrentApplicationVersionId": 4,
   "InputId": "2.1"
}
```

DeleteApplicationOutput

DeleteApplicationOutput以下のアクションのリクエストコード例は、Managed Service for Apache Flink アプリケーションからアプリケーション出力を削除します。

```
{
   "ApplicationName": "MyApplication",
   "CurrentApplicationVersionId": 4,
   "OutputId": "4.1"
}
```

DeleteApplicationReferenceDataSource

DeleteApplicationReferenceDataSource以下のアクションのリクエストコード例では、Managed Service for Apache Flink アプリケーションからアプリケーション参照データソースを削除します。

```
{
   "ApplicationName": "MyApplication",
   "CurrentApplicationVersionId": 5,
   "ReferenceId": "5.1"
}
```

DeleteApplicationSnapshot

DeleteApplicationSnapshot以下のアクションのリクエストコード例は、アプリケーションの状態のスナップショットを削除します。

```
{
   "ApplicationName": "MyApplication",
   "SnapshotCreationTimestamp": 12345678912,
   "SnapshotName": "MySnapshot"
}
```

DeleteApplicationVpcConfiguration

DeleteApplicationVpcConfiguration以下のアクションのリクエストコード例は、アプリケーションから既存の VPC 設定を削除します。

```
{
   "ApplicationName": "MyApplication",
   "CurrentApplicationVersionId": 9,
   "VpcConfigurationId": "1.1"
}
```

DescribeApplication

DescribeApplication以下のアクションのリクエストコード例では、Apache Flink のマネージドサービスアプリケーションに関する詳細が返されます。

```
{"ApplicationName": "MyApplication"}
```

DescribeApplicationSnapshot

DescribeApplicationSnapshot以下のアクションのリクエストコード例は、アプリケーション状態のスナップショットに関する詳細を返します。

```
{
   "ApplicationName": "MyApplication",
   "SnapshotName": "MySnapshot"
}
```

DiscoverInputSchema

DiscoverInputSchema以下のアクションのリクエストコード例は、ストリーミングソースからスキーマを生成します。

```
{
   "InputProcessingConfiguration": {
      "InputLambdaProcessor": {
```

```
         "ResourceARN": "arn:aws:lambda:us-
east-1:012345678901:function:MyLambdaFunction"
      }
   },
   "InputStartingPositionConfiguration": {
      "InputStartingPosition": "NOW"
   },
   "ResourceARN": "arn:aws:kinesis:us-east-1:012345678901:stream/ExampleInputStream",
   "S3Configuration": {
      "BucketARN": "string",
      "FileKey": "string"
   },
   "ServiceExecutionRole": "string"
}
```

DiscoverInputSchema以下のアクションのリクエストコード例では、参照ソースからスキーマを生成します。

```
{
   "S3Configuration": {
      "BucketARN": "arn:aws:s3:::mybucket",
      "FileKey": "TickerReference.csv"
   },
   "ServiceExecutionRole": "arn:aws:iam::123456789123:role/myrole"
}
```

ListApplications

ListApplications以下のアクションのリクエストコード例では、アカウント内の Managed Service for Apache Flink アプリケーションのリストが返されます。

```
{
   "ExclusiveStartApplicationName": "MyApplication",
   "Limit": 50
}
```

ListApplicationSnapshots

ListApplicationSnapshots以下のアクションのリクエストコード例は、アプリケーションの状態のスナップショットのリストを返します。

```
{"ApplicationName": "MyApplication",
   "Limit": 50,
   "NextToken": "aBcDeFgHiJkLmNoPqRsTuVwXyZ0123"
}
```

StartApplication

StartApplication以下のアクションのリクエストコード例は、Managed Service for Apache Flink アプリケーションを起動し、最新のスナップショット (存在する場合) からアプリケーションの状態をロードします。

```
{
    "ApplicationName": "MyApplication",
    "RunConfiguration": {
        "ApplicationRestoreConfiguration": {
         "ApplicationRestoreType": "RESTORE_FROM_LATEST_SNAPSHOT"
         }
    }
}
```

StopApplication

次の API_ StopApplication アクションのリクエストコード例は、Apache Flink 用マネージドサービスアプリケーションを停止します。

```
{"ApplicationName": "MyApplication"}
```

UpdateApplication

UpdateApplication以下のアクションのリクエストコード例は、Apache Flink 用マネージドサービスアプリケーションを更新して、アプリケーションコードの場所を変更します。

```
{"ApplicationName": "MyApplication",
"CurrentApplicationVersionId": 1,
"ApplicationConfigurationUpdate": {
      "ApplicationCodeConfigurationUpdate": {
         "CodeContentTypeUpdate": "ZIPFILE",
         "CodeContentUpdate": {
```

```
            "S3ContentLocationUpdate": {
               "BucketARNUpdate": "arn:aws:s3:::my_new_bucket",
               "FileKeyUpdate": "my_new_code.zip",
               "ObjectVersionUpdate": "2"
            }
         }
      }
}
```

Managed Service for Apache Flink API リファレンス

Apache Flink 用 Managed Service が提供する API の詳細については、「Apache Flink API Reference 用 Managed Service」を参照してください。

このコンテンツはリリースバージョンに移動されました。「リリースバージョン」を参照してください。