Red Hat Streams for Apache Kafka 2.7

OpenShift 上の Streams for Apache Kafka 2.7 のリリースノート

OpenShift Container Platform 上の Streams for Apache Kafka のこのリリースにおける新機能と変更点のハイライト

Last Updated: 2024-11-21

Red Hat Streams for Apache Kafka 2.7 OpenShift 上の Streams for Apache Kafka 2.7 のリリースノート

OpenShift Container Platform 上の Streams for Apache Kafka のこのリリースにおける新機能と変更点のハイライト

法律上の通知

概要

リリースノートでは、Streams for Apache Kafka 2.7 リリースで導入された新機能、機能拡張、および修正についてまとめています。

目次

第1章 STREAMS FOR APACHE KAFKA の名称変更のお知らせ

ブランディングの取り組みの一環として、AMQ Streams の名称が **Streams for Apache Kafka** に変更されます。この変更は、Red Hat の Apache Kafka 向け製品に対するお客様の認知度を高めることを目的としています。現在は移行期間中であり、AMQ Streams という古い呼称が使用されている可能性があります。当社では、新しい名称を反映するために、ドキュメント、リソース、メディアの更新に積極的に取り組んでいます。

第2章 FEATURES

Streams for Apache Kafka 2.7 では、このセクションで説明する機能が導入されています。

OpenShift 上の Streams for Apache Kafka 2.7 は、Apache Kafka 3.7.0 と Strimzi 0.40.x に基づいています。

注記

このリリースの機能拡張と解決されたバグをすべて確認するには、Streams for Apache Kafka の Jira プロジェクト を参照してください。

2.1. OPENSHIFT CONTAINER PLATFORM のサポート

Streams for Apache Kafka 2.7 は OpenShift Container Platform 4.12 から 4.16 でサポートされています。

詳細は、10章 *サポートされる構成* を参照してください。

2.2. KAFKA 3.7.0 のサポート

Streams for Apache Kafka は、Apache Kafka バージョン 3.7.0 をサポートおよび使用するようになりました。サポート対象は、Red Hat によってビルドされた Kafka ディストリビューションのみです。

ブローカーとクライアントアプリケーションを Kafka 3.7.0 にアップグレードする前に、Cluster Operator を Streams for Apache Kafka バージョン 2.7 にアップグレードする必要があります。アップグレードの手順については、Streams for Apache Kafka のアップグレード を参照してください。

詳細は、Kafka 3.7.0 リリースノートを参照してください。

Kafka 3.6.x は、Streams for Apache Kafka 2.7 にアップグレードする目的でのみサポートされます。

注記

Kafka 3.7.0 は KRaft モードへのアクセスを提供します。KRaft モードでは、Raft プロトコルを利用することで ZooKeeper なしで Kafka が実行されます。

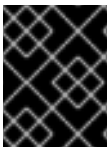
2.3. V1BETA2 API バージョンのサポート

すべてのカスタムリソースの **v1beta2** API バージョンが、Streams for Apache Kafka 1.7 で導入されました。Streams for Apache Kafka 1.8 で、**KafkaTopic** と **KafkaUser** を除くすべての Streams for Apache Kafka カスタムリソースから **v1alpha1** および **v1beta1** API バージョンが削除されました。

カスタムリソースを **v1beta2** にアップグレードすると、Kubernetes 1.22 に必要な Kubernetes CRD **v1** へ移行する準備ができます。

バージョン 1.7 より前の Streams for Apache Kafka バージョンからアップグレードする場合は、以下を行います。

1. Streams for Apache Kafka 1.7 へのアップグレード
2. カスタムリソースの **v1beta2** への変換
3. Streams for Apache Kafka 1.8 へのアップグレード



重要

Streams for Apache Kafka バージョン 2.7 にアップグレードする前に、API バージョン **v1beta2** を使用するようにカスタムリソースをアップグレードする必要があります。

2.3.1. カスタムリソースの v1beta2 へのアップグレード

カスタムリソースの **v1beta2** へのアップグレードをサポートするために、Streams for Apache Kafka には **API 変換ツール** が用意されています。このツールは、Streams for Apache Kafka 1.8 ソフトウェアダウンロードページ からダウンロードできます。

カスタムリソースは、手順 2 つでアップグレードします。

ステップ 1: カスタムリソースの形式への変換

API 変換ツールを使用して、以下のいずれかの方法でカスタムリソースの形式を **v1beta2** に適用可能な形式に変換できます。

- Streams for Apache Kafka カスタムリソースの設定を記述した YAML ファイルの変換
- クラスターでの Streams for Apache Kafka カスタムリソースの直接変換

各カスタムリソースを、**v1beta2** に適用可能な形式に手動で変換することもできます。カスタムリソースを手動で変換する手順は、ドキュメントを参照してください。

ステップ 2: CRD の v1beta2 へのアップグレード

次に、**crd-upgrade** コマンドで API 変換ツールを使用して、CRD の **ストレージ** API バージョンとして **v1beta2** を設定する必要があります。この手順は手動で行うことはできません。

詳細は、1.7 より前のバージョンの Streams for Apache Kafka からアップグレードする を参照してください。

2.4. STABLECONNECTIDENTITIES フィーチャーゲートの永続的な有効化

StableConnectIdentities フィーチャーゲートが GA (一般提供) に移行し、永続的に有効になりました。

この機能は、**Deployment** リソースを使用する代わりに、**StrimziPodSet** リソースを使用して Kafka Connect および Kafka MirrorMaker 2 Pod を管理します。これは、コネクタータスクの再バランスの数を最小限に抑えるのに役立ちます。

重要

StableConnectIdentities フィーチャーゲートを永続的に有効にすると、Streams for Apache Kafka 2.7 以降から Streams for Apache Kafka 2.3 以前に直接ダウングレードすることができなくなります。まず、Streams for Apache Kafka のいずれかのバージョンを介してダウングレードし、**StableConnectIdentities** フィーチャーゲートを無効にしてから、Streams for Apache Kafka 2.3 以前にダウングレードする必要があります。

StableConnectIdentities フィーチャーゲート を参照してください。

2.5. KAFKANODEPOOLS フィーチャーゲートのデフォルトでの有効化

この **KafkaNodePools** フィーチャーゲートはベータレベルの成熟度に達し、デフォルトで有効になりました。このフィーチャーゲートにより、**KafkaNodePool** カスタムリソースを通じて Apache Kafka ノードのさまざまなプールを設定できるようになります。

ノードプールは、Kafka クラスター内の Kafka ノードの個別のグループを指します。**KafkaNodePool** カスタムリソースは、ノードプール内のノードのみの設定を表します。各プールには独自の固有の設定があり、これにはレプリカの数、ストレージ設定、割り当てられたロールのリストなどの必須設定が含まれます。ノードプール内のノードにロールを割り当てることができるため、クラスター管理または KRaft モードに ZooKeeper を使用する Kafka クラスターでこの機能を試せます。割り当てることができるのは、controller ロール、broker ロール、または両方のロールです。ZooKeeper ベースの Apache Kafka クラスターで使用する場合、ロールを broker に設定する必要があります。

KafkaNodePools フィーチャーゲートを無効にするには、Cluster Operator 設定の **STRIMZI_FEATURE_GATES** 環境変数に **-KafkaNodePools** を指定します。

KafkaNodePools フィーチャーゲートの無効化

```
env:
  - name: STRIMZI_FEATURE_GATES
    value: -KafkaNodePools
```

ノードプールの設定 を参照してください。

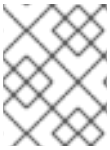
2.6. UNIDIRECTIONALTOPICOPERATOR フィーチャーゲートのデフォルトでの有効化

UnidirectionalTopicOperator フィーチャーゲートがベータレベルの成熟度に達し、デフォルトで有効になりました。このフィーチャーゲートにより、一方向トピック管理モードが導入されます。一方向モードでは、**KafkaTopic** リソースを使用して Kafka トピックを作成します。このトピックは Topic Operator によって管理されます。

UnidirectionalTopicOperator フィーチャーゲートを無効にするには、Cluster Operator 設定の **STRIMZI_FEATURE_GATES** 環境変数に **-UnidirectionalTopicOperator** を指定します。

UnidirectionalTopicOperator フィーチャーゲートの無効化

```
env:
  - name: STRIMZI_FEATURE_GATES
    value: -UnidirectionalTopicOperator
```



注記

双方向 の Topic Operator は KRaft モードでサポートされておらず、非推奨になりました。

Topic Operator の使用 を参照してください。

2.7. KRAFT: USEKRAFT フィーチャーゲートのデフォルトでの有効化

Streams for Apache Kafka の KRaft モードは、いくつかの制限がある テクノロジープレビュー ですが、このリリースでは、KRaft をサポートするいくつかの新機能が導入されています。

UseKRaft フィーチャーゲートがベータレベルの成熟度に達し、デフォルトで有効になりまし

た。**UseKRaft** フィーチャーゲートを有効にすると、Kafka クラスターが ZooKeeper なしで KRaft (Kafka Raft メタデータ) モードでデプロイされます。また、Kafka を KRaft モードで使用するには、Kafka カスタムリソースに **strimzi.io/kraft="enabled"** アノテーションが必要です。

KRaft モードを使用するには、**KafkaNodePool** リソースを使用してノードのグループ設定を管理する必要もあります。

UseKRaft フィーチャーゲートを無効にするには、Cluster Operator 設定の **STRIMZI_FEATURE_GATES** 環境変数の値として **-UseKRaft,-KafkaNodePools** を指定します。

UseKRaft フィーチャーゲートの無効化

```
env:
  - name: STRIMZI_FEATURE_GATES
    value: +UseKRaft,+KafkaNodePools
```

UseKRaft フィーチャーゲート および フィーチャーゲートリリース を参照してください。

2.8. KRAFT: ZOOKEEPER ベースから KRAFT ベースの KAFKA クラスターへの移行のサポート

Kafka クラスターでメタデータ管理に ZooKeeper を使用している場合は、KRaft モードでの Kafka の使用に移行できるようになりました。

移行中に次の操作を行います。

1. クラスター管理用の ZooKeeper に代わるコントローラーノードのクォーラムをノードプールとしてインストールします。
2. アノテーション **strimzi.io/kraft="migration"** を適用することで、クラスター設定で KRaft 移行を有効にします。
3. アノテーション **strimzi.io/kraft="enabled"** を使用して、ブローカーを KRaft の使用に切り替え、コントローラーを移行モードから切り替えます。

KRaft モードへの移行 を参照してください。

2.9. KRAFT: KRAFT ロール移行のサポート

Streams for Apache Kafka は、さまざまな KRaft ロールへのノード移行をサポートしています。ノードプールの設定により、次の移行を実行できるようになりました。

KRaft ロールの組み合わせ

broker 専用ロールと controller 専用ロールのみを持つ別々のノードプールから、二重のロールを持つ 1 つのノードプールの使用に移行します。

KRaft ロールの分割

controller ロールと broker ロールを組み合わせたノードプールの使用から、別々のロールを持つ 2 つのノードプールの使用に移行します。

ノードプール設定で **broker** ロールを削除するときにパーティションがまだ割り当てられている場合、変更が防止されます。

二重のロールを持つノードへの移行 および 個別の broker ロールと controller ロールへの移行 を参照してください。

注記

現在、スケーリング操作が可能なのは、専用ブローカーとして実行されるノードを含むブローカー専用のノードプールのみです。

2.10. KRAFT: KRAFT ベースのクラスターで KAFKA をアップグレードする

KRaft から KRaft へのアップグレードがサポートされるようになりました。KRaft ベースの Kafka クラスターを、サポートされている新しい Kafka バージョンおよび KRaft メタデータバージョンにアップグレードします。

Kafka バージョンは、以前と同様に指定します。KRaft メタデータバージョンは、**Kafka** リソースの新しい **metadataVersion** プロパティーを使用して指定します。

KRaft メタデータバージョンの設定

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    replicas: 3
    metadataVersion: 3.7-IV2
    version: 3.7.0
    # ...
```

metadataVersion が設定されていない場合、Kafka バージョンを更新した後、Streams for Apache Kafka はそれを現在のデフォルトに自動的に更新します。ローリング更新により、各 Pod が新バージョンの Kafka のブローカーバイナリーを使用するようになります。

KRaft ベースの Kafka クラスターとクライアントアプリケーションのアップグレード を参照してください。

2.11. KAFKA ブローカーの階層化ストレージ

階層化ストレージは Kafka の早期アクセス機能であり、Streams for Apache Kafka でも 開発者プレビュー として利用できます。

2.12. RHEL 7 がサポート対象外に

RHEL 7 はサポート対象外になりました。この決定は、既知の非互換性の問題 が原因で行われました。

第3章 機能拡張

Streams for Apache Kafka 2.7 では、いくつかの機能拡張が追加されています。

3.1. KAFKA 3.7.0 の機能拡張

Kafka 3.7.0 で導入された機能強化の概要については、Kafka 3.7.0 リリースノートを参照してください。

3.2. ブローカーのスケールダウン処理の改善

デフォルトでは、Streams for Apache Kafka は、Kafka クラスターでスケールダウン操作を開始する前に、ブローカー上にパーティションレプリカが存在しないことをチェックします。以前は、ブローカーがまだ使用中であることが判明した場合、スケールダウン操作がブロックされ、調整が失敗していました。現在は、スケールダウンによる変更は元に戻されますが、調整は通常どおり続行できます。

ただし、このブロックメカニズムを回避するほうがよい場合もあります。たとえば、ビジーなクラスターのチェックを無効にすると便利な場合があります。これを行うには、**strimzi.io/skip-broker-scaledown-check** を **true** に設定して、**Kafka** リソースにアノテーションを付けます。

スケールダウン操作でのチェックのスキップ を参照してください。

3.3. KAFKA EXPORTER: 切断されたコンシューマーでのメトリクス収集の防止

切断されたコンシューマーのメトリクスの収集を停止できるようになりました。切断されたコンシューマーでのメトリクスの収集を停止するには、Kafka Exporter 設定で **showAllOffsets** プロパティーを **false** に設定します。

KafkaExporterSpec スキーマリファレンス を参照してください。

3.4. CRD のラベルとアノテーションの一貫性向上

CRD 内のラベルとアノテーションの定義 (**template** セクションなど) で、整数値が使用できなくなりました。常に文字列値を使用する必要があります。

例として、以下の設定を見てみましょう。

アノテーションを含むテンプレート設定例

```
template:
  apiService:
    metadata:
      annotations:
        discovery.myapigateway.io/port: 8080
```

次のようにする必要があります。

アノテーションを含むテンプレート設定例

```
template:
  apiService:
    metadata:
```

```
    annotations:
      discovery.myapigateway.io/port: "8080"
```

これは、無効な値エラーが発生する可能性を回避するためです。

無効な値エラーの例

```
spec.template.apiService.metadata.annotations.discovery.myapigateway.io/port:
Invalid value: "integer":
spec.template.apiService.metadata.annotations.discovery.myapigateway.io/port in body must be of
type string: "integer"
```

3.5. DRAIN CLEANER: エビクション要求の処理の変更

Streams for Apache Kafka Drain Cleaner が Kubernetes Pod のエビクション要求を処理する方法が変更されました。Drain Cleaner は、デフォルトで Kubernetes のエビクション要求を拒否 (ブロック) して Kubernetes による Pod のエビクトを防ぎ、代わりに Cluster Operator を使用して Pod を移動するようになりました。

以前の方法を引き続き使用することもできます。以前は、Drain Cleaner が Kubernetes のエビクション要求を許可し、同時に Cluster Operator に Pod の移動を指示していました。このレガシーモードを機能させるには、次の手順を実行する必要があります。

1. Drain Cleaner 環境変数 **STRIMZI_DENY_EVICTION** を **false** に設定します。
2. **PodDisruptionBudget** の **maxUnavailable** オプションを **0** に設定して、Pod エビクションを一切許可しないように設定します。

Strimzi Drain Cleaner を使用した Pod のエビクト を参照してください。

3.6. KAFKA CONNECT メトリクスとダッシュボードのサンプルの強化

Kafka Connect で収集されるメトリクスが強化され、次のものが追加されました。

- コーディネーターメトリクスの収集
- 簡略化されたメトリクス正規表現

Kafka Connect のサンプル Grafana ダッシュボードファイルが更新され、より多くのコネクター情報とリバランスメトリクスを表示するパネルが追加されました。

3.7. MIRRORMAKER 2 ダッシュボードのサンプルの強化

MirrorMaker 2 のサンプル Grafana ダッシュボードファイルが、次のように更新されました。

- パネルを整理するための新しい行を追加
- より多くのコネクター情報と再バランスメトリクスを表示するようパネルを更新

3.8. KAFKA BRIDGE のテキスト形式

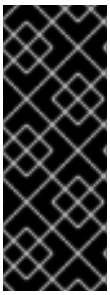
プロデューサー操作を行う場合に、**POST** リクエストには、生成されるメッセージの **埋め込みデータ形式** を示す **Content-Type** ヘッダーを指定する必要があります。以前は、レコードとキーの値でサポートされている形式が JSON とバイナリーでした。**text** 形式も使用できるようになりました。

表3.1 サポートされているコンテンツタイプの形式

埋め込みデータ形式	Content-Type ヘッダー
JSON	**Content-Type: application/vnd.kafka.json.v2+json**
バイナリー	**Content-Type: application/vnd.kafka.binary.v2+json**
テキスト	**Content-Type: application/vnd.kafka.text.v2+json**

第4章 テクノロジープレビュー

Streams for Apache Kafka 2.7 に含まれるテクノロジープレビュー機能です。



重要

テクノロジープレビュー機能は、Red Hat の実稼働環境のサービスレベルアグリーメント (SLA) ではサポートされません。また、機能的に完全ではない可能性があるため、Red Hat はテクノロジープレビュー機能を実稼働環境に実装することは推奨しません。テクノロジープレビューの機能は、最新の技術をいち早く提供して、開発段階で機能のテストやフィードバックの収集を可能にするために提供されます。サポート範囲の詳細は、テクノロジープレビュー機能のサポート範囲 を参照してください。

4.1. KRAFT モード

KRaft モードはテクノロジープレビューとして利用できます。

現在、Streams for Apache Kafka の KRaft モードには次の制限があります。

- KRaft モードでは **Unidirectional** Topic Operator のみがサポートされます。 **Bidirectional** Topic Operator はサポートされていないため、**UnidirectionTopicOperator** フィーチャーゲートが無効になっている場合は、**spec.entityOperator.topicOperator** プロパティーを **Kafka** カスタムリソースから **削除する必要があります**。
- JBOD ストレージはサポートされていません。**type: jbod** ストレージを使用できますが、JBOD アレイに含めることができるディスクは 1 つだけです。
- KRaft コントローラー専用ノードのスケールアップまたはスケールダウンはサポートされていません。
- Kafka クラスターから削除された Kafka ノードの登録解除。

4.2. STREAMS FOR APACHE KAFKA CONSOLE

Streams for Apache Kafka のコンソール (ユーザーインターフェイス) が、テクノロジープレビューとして利用可能になりました。Streams for Apache Kafka Console は、Streams for Apache Kafka デプロイメントとシームレスに統合するように設計されており、Kafka クラスターの監視と管理のための一元化されたハブを提供します。コンソールをデプロイし、Streams for Apache Kafka が管理する Kafka クラスターに接続します。

ブローカー、トピック、コンシューマーグループをカバーする専用ページから、接続された各クラスターの洞察を得ることができます。ブローカー、トピック、または接続されたコンシューマーグループに関する特定の情報を調べる前に、Kafka クラスターのステータスなどの重要な情報を確認できます。

Streams for Apache Kafka Console ガイド を参照してください。

4.3. STREAMS FOR APACHE KAFKA PROXY

Streams for Apache Kafka Proxy は、Kafka ベースのシステムを強化するために設計された Apache Kafka プロトコル対応プロキシーです。そのフィルターメカニズムにより、アプリケーションや Kafka クラスター自体を変更することなく、Kafka ベースのシステムに追加の動作を導入できます。

テクノロジープレビューの一部として、Streams for Apache Kafka Proxy の Record Encryption フィルターを試すことができます。このフィルターは、業界標準の暗号化技術を使用して Kafka メッセージに暗号化を適用し、Kafka クラスターに保存されているデータの機密性を確保します。

Streams for Apache Kafka Proxy ガイド を参照してください。

4.4. KAFKA STATIC QUOTA プラグインの設定

Kafka Static Quota プラグインのテクノロジープレビューを使用して、Kafka クラスターのブローカーにスループットおよびストレージの制限を設定します。**Kafka** リソースを設定して、プラグインを有効にし、制限を設定します。バイトレートのしきい値およびストレージクォータを設定して、ブローカーと対話するクライアントに制限を設けることができます。

Kafka Static Quota プラグインの設定例

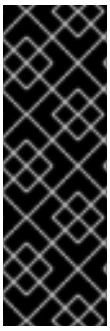
```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
    config:
      client.quota.callback.class: io.strimzi.kafka.quotas.StaticQuotaCallback
      client.quota.callback.static.produce: 1000000
      client.quota.callback.static.fetch: 1000000
      client.quota.callback.static.storage.soft: 400000000000
      client.quota.callback.static.storage.hard: 500000000000
      client.quota.callback.static.storage.check-interval: 5
```

Kafka Static Quota プラグインを使用したブローカーへの制限の設定 を参照してください。

第5章 開発者プレビュー

Streams for Apache Kafka 2.7 に含まれる開発者プレビュー機能です。

Kafka クラスター管理者は、Cluster Operator デプロイメント設定でフィーチャーゲートを使用して、機能のサブセットのオンとオフを切り替えることができます。開発者プレビューとして利用できるフィーチャーゲートは、アルファレベルの成熟度であり、デフォルトで無効になっています。



重要

開発者プレビューの機能は、Red Hat の実稼働環境のサービスレベルアグリーメント (SLA) ではサポートされず、機能的に完全ではないことがあるため、Red Hat はテクノロジープレビュー機能を実稼働環境に実装することは推奨しません。この開発者向けプレビュー機能は、近々発表予定の製品イノベーションをリリースに先駆けてご提供することにより、お客様は機能性をテストし、開発プロセス中にフィードバックをお寄せいただくことができます。サポート範囲の詳細は、開発者プレビューのサポート範囲 を参照してください。

5.1. KAFKA ブローカーの階層化ストレージ

Streams for Apache Kafka は、開発者プレビューとして Kafka ブローカーの階層化ストレージをサポートするようになりました。これにより、ローカルストレージだけでなくカスタムリモートストレージソリューションも導入できるようになりました。現在の制限 により、実稼働環境では推奨されません。

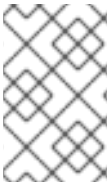
リモートストレージ設定は、**Kafka** リソースの **kafka.tieredStorage** プロパティーを使用して指定します。カスタムリモートストレージマネージャーを指定して、階層化ストレージを管理します。

カスタム階層化ストレージ設定の例

```
kafka:
  tieredStorage:
    type: custom
    remoteStorageManager:
      className: com.example.kafka.tiered.storage.s3.S3RemoteStorageManager
      classPath: /opt/kafka/plugins/tiered-storage-s3/*
      config:
        # remote storage manager configuration (1)
        storage.bucket.name: my-bucket
  config:
    ...
    rlmm.config.remote.log.metadata.topic.replication.factor: 1 (2)
```

(1) 必要な設定を使用してカスタムリモートストレージマネージャーを設定します。キーには自動的に **rsm.config** という接頭辞が付けられ、Kafka ブローカー設定に追加されます。

(2) Streams for Apache Kafka は、Remote Log Metadata Management (RLMM) に **TopicBasedRemoteLogMetadataManager** を使用します。**rlmm.config** 接頭辞を使用して RLMM 設定を追加します。



注記

カスタム階層化ストレージを使用する場合は、まずカスタムコンテナーイメージをビルドして、階層化ストレージプラグインを Streams for Apache Kafka イメージに追加する必要があります。

階層化ストレージ (早期アクセス) を参照してください。

第6章 KAFKA の重大な変更

このセクションでは、Kafka への変更を説明します。この変更に伴い、Streams for Apache Kafka を引き続き機能させるために、対応する変更が加えられました。

6.1. KAFKA のサンプルファイルコネクターの使用

Kafka では、デフォルトで、サンプルファイルコネクター **FileStreamSourceConnector** と **FileStreamSinkConnector** が **CLASSPATH** と **plugin.path** に含まれなくなりました。これらのサンプルコネクターを引き続き使用できるように、Streams for Apache Kafka が更新されました。サンプルは、他のコネクターと同様にプラグインパスに追加する必要があります。

Streams for Apache Kafka には、ファイルコネクターを **KafkaConnector** リソースとしてデプロイするために必要な設定を含むコネクター設定ファイルのサンプルが用意されています。

- **examples/connect/source-connector.yaml**

サンプル KafkaConnector リソースのデプロイ および コネクタープラグインでの Kafka Connect の拡張 を参照してください。

第7章 非推奨の機能

Streams for Apache Kafka の以前のリリースでサポートされていた非推奨の機能です。

7.1. スキーマプロパティーの非推奨化

スキーマ	非推奨のプロパティー	代替プロパティー
AclRule	**operation**	**operation**
CruiseControlSpec	**tlsSidecar**	-
CruiseControlTemplate	**tlsSidecarContainer**	-
CruiseControlSpec.BrokerCapacity	**disk**	-
CruiseControlSpec.BrokerCapacity	**cpuUtilization**	-
EntityUserOperator	**zookeeperSessionTimeoutSeconds**	-
JaegerTracing	**type**	-
KafkaConnectorSpec	**pause**	**state**
KafkaConnectTemplate	**deployment**	**StrimziPodSet** リソースに置き換えられました。
KafkaClusterTemplate	**statefulset**	**StrimziPodSet** リソースに置き換えられました。
KafkaExporterTemplate	**service**	-
KafkaMirrorMaker	すべてのプロパティー	-
KafkaMirrorMaker2ConnectorSpec	**pause**	**state**
KafkaMirrorMaker2MirrorSpec	**topicsBlacklistPattern**	**topicsExcludePattern**
KafkaMirrorMaker2MirrorSpec	**groupsBlacklistPattern**	**groupsExcludePattern**
ListenerStatus	**type**	**name**

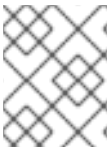
スキーマ	非推奨のプロパティー	代替プロパティー
ZookeeperClusterTemplate	**statefulset**	**StrimziPodSet** リソースに置き換えられました。

Streams for Apache Kafka カスタムリソース API リファレンス を参照してください。

7.2. STREAMS FOR APACHE KAFKA 2.7.0 での JAVA 11 の非推奨化

Java 11 のサポートは、Kafka 3.7.0 および Streams for Apache Kafka 2.7.0 で非推奨になりました。Java 11 は、今後、クライアントを含む Streams for Apache Kafka コンポーネントでサポートされなくなります。

Streams for Apache Kafka は Java 17 をサポートしています。新しいアプリケーションを開発する場合は、Java 17 を使用してください。また、現在 Java 11 を使用しているアプリケーションの Java 17 への移行も計画してください。



注記

Streams for Apache Kafka 2.4.0 で Java 8 のサポートが削除されました。現在 Java 8 を使用している場合は、同様に Java 17 への移行を計画してください。

7.3. 双方向の TOPIC OPERATOR

双方向の Topic Operator は非推奨となり、2.8 リリースで削除されます。双方向の Topic Operator は KRaft モードではサポートされていません。KRaft モードでサポートされている一方向 Topic Operator は、2.8 リリースで永続的に有効になります。

7.4. 環境変数設定プロバイダーは非推奨に

設定プロバイダーを使用すると、プロデューサーやコンシューマーを含むすべての Kafka コンポーネントの外部ソースから設定データをロードできます。

以前は、コンポーネントの **spec** 設定で **config.providers** プロパティーを使用して、**secrets.io.strimzi.kafka.EnvVarConfigProvider** 環境変数設定プロバイダーを有効にすることができました。ただし、このプロバイダーは現在非推奨となっており、将来削除される予定です。したがって、Kafka 独自の環境変数設定プロバイダー (**org.apache.kafka.common.config.provider.EnvVarConfigProvider**) を使用して設定プロパティーを環境変数として提供するように実装を更新することが推奨されます。

環境変数設定プロバイダーを有効にする設定例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
  name: my-connect
  annotations:
    strimzi.io/use-connector-resources: "true"
spec:
  # ...
  config:
```

```
    # ...
    config.providers: env
    config.providers.env.class: org.apache.kafka.common.config.provider.EnvVarConfigProvider
  # ...
```

7.5. KAFKA MIRRORMAKER 2 のアイデンティティーレプリケーションポリシー

ID レプリケーションポリシーは、リモートトピックの自動名前変更機能の後継として MirrorMaker 2 で使用される機能です。名前の前にソースクラスターの名前を付ける代わりに、トピックは元の名前を保持します。この設定は、アクティブ/パッシブバックアップおよびデータ移行シナリオに特に役立ちます。

ID レプリケーションポリシーを実装するには、MirrorMaker 2 設定でレプリケーションポリシークラス (**replication.policy.class**) を指定する必要があります。以前は、Streams for Apache Kafka の **mirror-maker-2-extensions** コンポーネントに含まれる **io.strimzi.kafka.connect.mirror.IdentityReplicationPolicy** クラスを指定できました。ただし、このコンポーネントは非推奨となり、今後削除される予定です。したがって、Kafka 独自のレプリケーションポリシークラス (**org.apache.kafka.connect.mirror.IdentityReplicationPolicy**) を使用するように実装を更新することを推奨します。

Kafka MirrorMaker 2 の設定 を参照してください。

7.6. KAFKA MIRRORMAKER 1

Kafka MirrorMaker は、データセンター内またはデータセンター全体の 2 台以上の Kafka クラスター間でデータをレプリケーションします。Kafka MirrorMaker 1 は Kafka 3.0.0 で非推奨となり、Kafka 4.0.0 で削除される予定です。MirrorMaker 2 が利用可能な唯一のバージョンになります。MirrorMaker 2 は、クラスター間のデータ転送を管理するコネクターである Kafka Connect フレームワークに基づいています。

その結果、Kafka MirrorMaker 1 のデプロイに使用される Streams for Apache Kafka の **KafkaMirrorMaker** カスタムリソースが非推奨になりました。**KafkaMirrorMaker** リソースは、Kafka 4.0.0 の導入時に Streams for Apache Kafka から削除されます。

MirrorMaker 1 (Streams for Apache Kafka ドキュメントでは **MirrorMaker** と表記) を使用している場合は、**KafkaMirrorMaker2** カスタムリソースと **IdentityReplicationPolicy** クラスを使用してください。MirrorMaker 2 は、ターゲットクラスターに複製されたトピックの名前を変更します。**IdentityReplicationPolicy** 設定は、名前の自動変更をオーバーライドします。これを使用して、MirrorMaker 1 と同じ active/passive の一方向レプリケーションを作成します。

Kafka MirrorMaker 2 の設定 を参照してください。

7.7. KAFKA BRIDGE の SPAN 属性

Kafka Bridge の次の span 属性が非推奨になりました。該当する場合は代替属性を記載しています。

- **http.method** が **http.request.method** に置き換えられました。
- **http.url** が **url.scheme**、**url.path**、および **url.query** に置き換えられました。
- **messaging.destination** が **messaging.destination.name** に置き換えられました。
- **http.status_code** が **http.response.status_code** に置き換えられました。

- **messaging.destination.kind=topic** が置き換えられました。代替属性はありません。

Kafka Bridge は分散トレーシングに OpenTelemetry を使用します。これらの変更は、OpenTelemetry のセマンティック規則の変更に沿ったものです。これらの属性は、Kafka Bridge の今後のリリースで削除される予定です。

第8章 修正された問題

OpenShift 上の Streams for Apache Kafka 2.7 で修正された問題です。

Kafka 3.7.0 で修正された問題の詳細は、Kafka 3.7.0 リリースノートを参照してください。

表8.1 修正された問題

課題番号	説明
ENTMQST-5839	OAuth の問題 (**fallbackUsernamePrefix** が反映されない) の修正
ENTMQST-5820	結果の **sasl.jaas.config** 値に不要な引用符が含まれているため、Oauth が有効なクラスターで MM2 コネクタータスクが失敗する
ENTMQST-5754	リソース削除時に不要なパッチ適用を避ける
ENTMQST-5753	複数の HTTP リクエストで別々の埋め込み形式を使用して生成することが認められない
ENTMQST-5656	API シークレットが変更されても Cruise Control が再起動しない
ENTMQST-5603	**UseKRaft** フィーチャーゲートのベータ版への昇格
ENTMQST-5583	一方向 Topic Operator が、BTO からのアップグレード時に停止を引き起こす
ENTMQST-5582	スムーズなローリング更新を可能にし、可用性を確保するために、Cruise Control トピックで適切な設定を使用する必要がある
ENTMQST-5581	一方向 Topic Operator でのログレベルの使用方法を改善する必要がある
ENTMQST-5546	**KafkaRoller** でコントローラー専用のノードを混合ノードに移行するのが困難である
ENTMQST-5540	MirrorMaker 2 コネクターのコネクター状態の処理を修正する
ENTMQST-5511	KRaft ノードのローリング、liveness、readiness
ENTMQST-5504	KRaft が有効な場合の Kafka および Strimzi のアップグレードのサポートを追加する
ENTMQST-5492	コントローラーノードのアドバタイズされたリスナーの処理を修正する
ENTMQST-5387	**StableConnectIdentities** フィーチャーゲートを GA に昇格する
ENTMQST-5383	カスタムの "bring-your-own" プラグインによる階層化ストレージのサポート
ENTMQST-5360	一方向 Topic Operator の追加タスク (2.7.0 エディション)
ENTMQST-5292	既存の PV/PVC から回復する際のクラスター ID 検証に関する問題に対処する

課題番号	説明
ENTMQST-4194	Topic Operator がユーザーに禁止設定を許可する
ENTMQST-4164	内部トピックの定期的な調整に関する永続的なエラー
ENTMQST-4087	Topic Operator がトピックの一括削除に失敗する
ENTMQST-3970	内部の Kafka Connect トピックが無効な設定で再作成される
ENTMQST-3994	ZooKeeper から KRaft への移行
ENTMQST-3974	Topic Operator の修正
ENTMQST-3886	状態ストア、トピックストアが別のインスタンスに移行した可能性がある

表8.2 CVE (Common Vulnerabilities and Exposures) の修正

課題番号	説明
ENTMQST-5886	CVE-2023-43642 の不具合が snappy-java の SnappyInputStream で発見される
ENTMQST-5885	CVE-2023-52428 Nimbus JOSE+JWT (9.37.2 より前)
ENTMQST-5884	CVE-2022-4899 の脆弱性が zstd v1.4.10 で発見される
ENTMQST-5883	CVE-2021-24032 の不具合が zstd で発見される
ENTMQST-5882	CVE-2024-23944 Apache ZooKeeper: 永続ウォッチャー処理における情報漏洩
ENTMQST-5881	lz4 の CVE-2021-3520 の不具合
ENTMQST-5835	CVE-2024-29025 netty-codec-http: 制限やスロットリングのないリソースの割り当て
ENTMQST-5646	CVE-2024-1023 vert.x: io.vertx/vertx-core: Vertx での Netty FastThreadLocal データ構造の使用によるメモリーリーク
ENTMQST-5667	CVE-2024-1300 vertx-core: io.vertx:vertx-core: TCP サーバーが TLS および SNI サポートで設定されている場合のメモリーリーク

第9章 既知の問題

このセクションでは、OpenShift 上の Streams for Apache Kafka 2.7 に関する既知の問題を説明します。

9.1. OPENSHIFT 4.16: シークレットの過剰な生成

OpenShift Container Platform (OCP) バージョン 4.16 に Kafka インスタンスをデプロイすると、Kafka namespace で **dockercfg** シークレットが継続的に作成されます。

この問題は、サービスアカウントに **openshift.io/internal-registry-pull-secret-ref** アノテーションが追加されると発生し、Streams for Apache Kafka と OpenShift がアノテーションを繰り返し書き換えるため、調整ループが発生します。時間が経つにつれて、何千もの不要なシークレットが蓄積される可能性があります。

回避策

この問題を軽減するには、問題が解決されている OCP バージョン 4.16.4 以降にアップグレードしてください。

アップグレードがすぐにできない場合は、一時的な回避策として、各サービスアカウントの **openshift.io/internal-registry-pull-secret-ref** アノテーションを手動で設定し、調整ループを防止します。

アノテーションの設定

```
template:
  serviceAccount:
    metadata:
      annotations:
        openshift.io/internal-registry-pull-secret-ref: my-cluster-entity-operator-dockercfg-qxwxd
```

9.2. RHEL 7 との非互換性

RHEL 7 を Kafka 3.7 と併用する場合、互換性の問題が発生することが確認されています。その結果、RHEL 7 はサポート対象外になりました。この問題は、RHEL 7 上の GCC (GNU Compiler Collection) のバージョンが古く、Kafka 3.7 に必要な RocksDB JNI ライブラリーのバージョン (**org.rocksdb:rocksdbjni:7.9.2**) と互換性がないために発生します。

RocksDB JNI バージョン 7.9.2 には、RHEL 7 で使用可能なものよりも新しいバージョンの GCC および関連する **libstdc++** ライブラリーが必要です。RocksDB に依存する Snappy 圧縮と Kafka Streams は、これらの古いライブラリーが原因で、RHEL 7 では正しく機能しません。

推奨設定

- Kafka 3.7 および最新の Streams for Apache Kafka 機能との互換性を確保するために、RHEL 7 で実行されているクライアントを RHEL 8 にアップグレードしてください。
- RHEL 7 を引き続き使用する場合は、Streams for Apache Kafka 2.5 LTS または 2.6 の使用を検討してください。

9.3. MIRRORMAKER 2 コネクターの自動再起動

状態がコネクター Operator に渡されないため、MirrorMaker 2 コネクターの自動再起動が機能しません。これは、Streams for Apache Kafka の今後のリリースで修正される予定です。

9.4. IPV6 クラスターの STREAMS FOR APACHE KAFKA CLUSTER OPERATOR

Streams for Apache Kafka Cluster Operator は、Internet Protocol version 6 (IPv6) クラスターでは起動しません。

回避策

この問題を回避する方法は 2 つあります。

回避方法 1:KUBERNETES_MASTER 環境変数の設定

1. OpenShift Container Platform クラスターの Kubernetes マスターノードのアドレスを表示します。

   ```
   oc cluster-info
   Kubernetes master is running at <master_address>
   # ...
   ```

   マスターノードのアドレスをコピーします。

2. すべての Operator サブスクリプションをリスト表示します。

   ```
   oc get subs -n <operator_namespace>
   ```

3. Streams for Apache Kafka の **Subscription** リソースを編集します。

   ```
   oc edit sub amq-streams -n <operator_namespace>
   ```

4. **spec.config.env** で、**KUBERNETES_MASTER** 環境変数を追加し、Kubernetes マスターノードのアドレスに設定します。以下に例を示します。

   ```
   apiVersion: operators.coreos.com/v1alpha1
   kind: Subscription
   metadata:
     name: amq-streams
     namespace: <operator_namespace>
   spec:
     channel: amq-streams-1.8.x
     installPlanApproval: Automatic
     name: amq-streams
     source: mirror-amq-streams
     sourceNamespace: openshift-marketplace
     config:
       env:
       - name: KUBERNETES_MASTER
         value: MASTER-ADDRESS
   ```

5. エディターを保存し、終了します。

6. **Subscription** が更新されていることを確認します。

```
oc get sub amq-streams -n <operator_namespace>
```

7. Cluster Operator の **Deployment** が、新しい環境変数を使用するように更新されていることを確認します。

```
oc get deployment <cluster_operator_deployment_name>
```

回避方法 2: ホスト名検証の無効化

1. すべての Operator サブスクリプションをリスト表示します。

```
oc get subs -n <operator_namespace>
```

2. Streams for Apache Kafka の **Subscription** リソースを編集します。

```
oc edit sub amq-streams -n <operator_namespace>
```

3. **spec.config.env** で、**true** に設定された **KUBERNETES_DISABLE_HOSTNAME_VERIFICATION** 環境変数を追加します。以下に例を示します。

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: amq-streams
  namespace: <operator_namespace>
spec:
  channel: amq-streams-1.8.x
  installPlanApproval: Automatic
  name: amq-streams
  source: mirror-amq-streams
  sourceNamespace: openshift-marketplace
  config:
    env:
    - name: KUBERNETES_DISABLE_HOSTNAME_VERIFICATION
      value: "true"
```

4. エディターを保存し、終了します。

5. **Subscription** が更新されていることを確認します。

```
oc get sub amq-streams -n <operator_namespace>
```

6. Cluster Operator の **Deployment** が、新しい環境変数を使用するように更新されていることを確認します。

```
oc get deployment <cluster_operator_deployment_name>
```

9.5. CRUISE CONTROL の CPU 使用率の推計

Streams for Apache Kafka の Cruise Control には、CPU 使用率の推計に関連する既知の問題があります。CPU 使用率は、ブローカー Pod の定義容量の割合として計算されます。この問題は、CPU コアが

異なるノードで Kafka ブローカーを実行する場合に発生します。たとえば、node1 には 2 つの CPU コアがあり、node2 には 4 つの CPU コアが含まれるとします。この場合、Cruise Control はブローカーの CPU 負荷を過大または過少に予測する可能性があります。この問題が原因で、Pod の負荷が大きいときにクラスターのリバランスができない場合があります。

この問題を回避する方法は 2 つあります。

回避策 1: CPU リクエストと制限を同等レベルにする

Kafka.spec.kafka.resources の CPU 制限と同等の CPU リクエストを設定できます。これにより、すべての CPU リソースが事前に予約され、常に利用できます。この設定を使用すると、CPU 目標に基づいてリバランス提案を準備するときに Cruise Control が CPU 使用率を適切に評価できます。

回避策 2: CPU の目標を除外する

Cruise Control 設定に指定されたハードおよびデフォルトの目標から CPU の目標を除外できます。

CPU の目標がない Cruise Control の設定例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
  zookeeper:
    # ...
  entityOperator:
    topicOperator: {}
    userOperator: {}
  cruiseControl:
    brokerCapacity:
      inboundNetwork: 10000KB/s
      outboundNetwork: 10000KB/s
    config:
      hard.goals: >
        com.linkedin.kafka.cruisecontrol.analyzer.goals.RackAwareGoal,
        com.linkedin.kafka.cruisecontrol.analyzer.goals.MinTopicLeadersPerBrokerGoal,
        com.linkedin.kafka.cruisecontrol.analyzer.goals.ReplicaCapacityGoal,
        com.linkedin.kafka.cruisecontrol.analyzer.goals.DiskCapacityGoal,
        com.linkedin.kafka.cruisecontrol.analyzer.goals.NetworkInboundCapacityGoal,
        com.linkedin.kafka.cruisecontrol.analyzer.goals.NetworkOutboundCapacityGoal
      default.goals: >
        com.linkedin.kafka.cruisecontrol.analyzer.goals.RackAwareGoal,
        com.linkedin.kafka.cruisecontrol.analyzer.goals.MinTopicLeadersPerBrokerGoal,
        com.linkedin.kafka.cruisecontrol.analyzer.goals.ReplicaCapacityGoal,
        com.linkedin.kafka.cruisecontrol.analyzer.goals.DiskCapacityGoal,
        com.linkedin.kafka.cruisecontrol.analyzer.goals.NetworkInboundCapacityGoal,
        com.linkedin.kafka.cruisecontrol.analyzer.goals.NetworkOutboundCapacityGoal,
        com.linkedin.kafka.cruisecontrol.analyzer.goals.ReplicaDistributionGoal,
        com.linkedin.kafka.cruisecontrol.analyzer.goals.PotentialNwOutGoal,
        com.linkedin.kafka.cruisecontrol.analyzer.goals.DiskUsageDistributionGoal,
        com.linkedin.kafka.cruisecontrol.analyzer.goals.NetworkInboundUsageDistributionGoal,
        com.linkedin.kafka.cruisecontrol.analyzer.goals.NetworkOutboundUsageDistributionGoal,
```

```
com.linkedin.kafka.cruisecontrol.analyzer.goals.TopicReplicaDistributionGoal,
com.linkedin.kafka.cruisecontrol.analyzer.goals.LeaderReplicaDistributionGoal,
com.linkedin.kafka.cruisecontrol.analyzer.goals.LeaderBytesInDistributionGoal
```

詳細は、Insufficient CPU capacity を参照してください。

9.6. FIPS モードで実行する場合の JMX 認証

JMX 認証を有効にして FIPS モードで Streams for Apache Kafka を実行すると、クライアントが認証に失敗する可能性があります。この問題を回避するには、FIPS モードでの実行中に JMX 認証を有効にしないでください。私たちはこの問題を調査しており、将来のリリースで解決できるよう取り組んでいます。

第10章 サポートされる構成

Streams for Apache Kafka 2.7 リリースでサポートされる構成です。

10.1. サポート対象のプラットフォーム

以下のプラットフォームは、記載されている OpenShift のバージョンで Kafka とともに実行される Streams for Apache Kafka 2.7 についてテストされています。

プラットフォーム	バージョン	アーキテクチャー
Red Hat OpenShift Container Platform	4.12 から 4.16	x86_64、ppc64le (IBM Power)、s390x (IBM Z および IBM® LinuxONE)、aarch64 (64 ビット ARM)
Red Hat OpenShift Container Platform の非接続環境	最新バージョン	x86_64、ppc64le (IBM Power)、s390x (IBM Z および IBM® LinuxONE)、aarch64 (64 ビット ARM)
Red Hat OpenShift Dedicated	最新バージョン	x86_64
Microsoft Azure Red Hat OpenShift (ARO)	最新バージョン	x86_64
Red Hat OpenShift Service on AWS (ROSA) ROSA with Hosted Control Plane (HCP) を含む	最新バージョン	x86_64
Red Hat MicroShift	最新バージョン	x86_64
Red Hat OpenShift Local	2.13-2.19 (OCP 4.12)、2.20-2.28 (OCP 4.13)、2.29-2.33 (OCP 4.14)、2.34-2.38 (OCP 4.15)、2.39 以降 (OCP 4.16)	x86_64

OpenShift Local は、Red Hat OpenShift Container Platform (OCP) の限定バージョンです。一部の機能が使用できない可能性があることを理解した上で、開発および評価のみに使用してください。

サポート対象外の機能

- Red Hat MicroShift は、コネクターを使用してコンテナーイメージを構築するための Kafka Connect のビルド設定をサポートしていません。
- IBM Z および IBM® LinuxONE s390x アーキテクチャーは、Streams for Apache Kafka OPA 統合をサポートしていません。

FIPS コンプライアンス

Streams for Apache Kafka 2.7.0 は FIPS 用に設計されています。Streams for Apache Kafka のコンテナーイメージは、承認を受けるために NIST に提出された RHEL 9.2 をベースとしています。

National Institute of Standards and Technology (NIST) によって承認されている RHEL のバージョンを確認するには、NIST Web サイトの Cryptographic Module Validation Program を参照してください。

Red Hat OpenShift Container Platform は FIPS 用に設計されています。OpenShift Container Platform のコアコンポーネントは、FIPS モードで起動された RHEL または RHEL CoreOS で実行される場合、x86_64、ppc64le (IBM Power)、s390x (IBM Z)、および aarch64 (64 ビット ARM) アーキテクチャー上でのみ、FIPS 検証のために NIST に提出された RHEL 暗号化ライブラリーを使用します。NIST の検証プログラムの詳細は、Cryptographic Module Validation Program を参照してください。検証のために提出された RHEL 暗号化ライブラリーの各バージョンの NIST における最新ステータスについては、Compliance Activities and Government Standards を参照してください。

OpenShift Container Platform 4.12 は、FIPS 140-2 をサポートする最後のバージョンです。NIST による今後の OpenShift バージョンの検証タイムラインが不確実であることから、Streams for Apache Kafka は、告知があるまで OpenShift 4.12 でサポートされます。

10.2. サポートされるクライアント

Streams for Apache Kafka では、Red Hat によって構築されたクライアントライブラリーのみがサポートされます。現在、Streams for Apache Kafka は Java クライアントライブラリーのみを提供しています。クライアントは、次のオペレーティングシステムおよびアーキテクチャー上の Streams for Apache Kafka 2.7 での使用がサポートされています。

オペレーティングシステム	アーキテクチャー	JVM
RHEL および UBI 8 および 9	x86、amd64、ppc64le (IBM Power)、s390x (IBM Z および IBM® LinuxONE)、aarch64 (64 ビット ARM)	Java 11 (非推奨) および Java 17

クライアントは Open JDK 11 および 17 でテストされていますが、Java 11 は Streams for Apache Kafka 2.7.0 で非推奨になりました。IBM JDK はサポートされていますが、リリースごとに定期的にテストされていません。Oracle JDK 11 はサポート対象外です。

Red Hat Universal Base Image (UBI) バージョンのサポートは、同じ RHEL バージョンに対応します。

10.3. サポートされる APACHE KAFKA エコシステム

Streams for Apache Kafka では、Apache Software Foundation から直接リリースされた次のコンポーネントのみがサポートされます。

- Apache Kafka Broker
- Apache Kafka Connect
- Apache MirrorMaker
- Apache MirrorMaker 2
- Apache Kafka Java Producer、Consumer、および Management クライアント、Kafka Streams
- Apache ZooKeeper

注記

Apache ZooKeeper は、Apache Kafka の実装の詳細としてのみサポートされており、他の目的のために変更しないでください。さらに、ZooKeeper ノードに割り当てられたコアまたは vCPU は、サブスクリプションコンプライアンスの計算には含まれません。つまり、ZooKeeper ノードは顧客のサブスクリプションにはカウントされません。

10.4. その他のサポートされる機能

- Kafka Bridge
- Drain Cleaner
- Cruise Control
- 分散トレーシング
- Streams for Apache Kafka Console (テクノロジープレビュー)
- Streams for Apache Kafka Proxy (テクノロジープレビュー)

注記

Streams for Apache Kafka Console および Streams for Apache Kafka Proxy は、実稼働環境に対応していません。テクノロジープレビュー機能については、x86 と amd64 でのみテストされています。

12章 *サポート対象となる Red Hat 製品との統合* も併せて参照してください。

10.5. ストレージ要件

Streams for Apache Kafka はブロックストレージでテストされており、Streams for Apache Kafka は、Kafka で一般的に使用される XFS および ext4 ファイルシステムと互換性があります。NFS などのファイルストレージオプションとは互換性がありません。

関連情報

Streams for Apache Kafka 2.5 LTS リリースでサポートされる構成の詳細は、Streams for Apache Kafka 2.5 リリースノート を参照してください。

第11章 コンポーネントの詳細

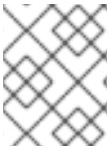
次の表に、Streams for Apache Kafka の各リリースのコンポーネントバージョンを示します。

注記

Operator、コンソール、プロキシーなどのコンポーネントは、OpenShift 上の Streams for Apache Kafka を使用する場合にのみ該当します。

Streams for Apache Kafka	Apache Kafka	Strimzi Operator	Kafka Bridge	Oauth	Cruise Control	Console	Proxy
2.7.0	3.7.0	0.40.0	0.28	0.15.0	2.5.128	0.1	0.5.1
2.6.0	3.6.0	0.38.0	0.27	0.14.0	2.5.128	-	-
2.5.1	3.5.0	0.36.0	0.26	0.13.0	2.5.123	-	-
2.5.0	3.5.0	0.36.0	0.26	0.13.0	2.5.123	-	-
2.4.0	3.4.0	0.34.0	0.25.0	0.12.0	2.5.112	-	-
2.3.0	3.3.1	0.32.0	0.22.3	0.11.0	2.5.103	-	-
2.2.2	3.2.3	0.29.0	0.21.5	0.10.0	2.5.103	-	-
2.2.1	3.2.3	0.29.0	0.21.5	0.10.0	2.5.103	-	-
2.2.0	3.2.3	0.29.0	0.21.5	0.10.0	2.5.89	-	-
2.1.0	3.1.0	0.28.0	0.21.4	0.10.0	2.5.82	-	-
2.0.1	3.0.0	0.26.0	0.20.3	0.9.0	2.5.73	-	-
2.0.0	3.0.0	0.26.0	0.20.3	0.9.0	2.5.73	-	-
1.8.4	2.8.0	0.24.0	0.20.1	0.8.1	2.5.59	-	-
1.8.0	2.8.0	0.24.0	0.20.1	0.8.1	2.5.59	-	-
1.7.0	2.7.0	0.22.1	0.19.0	0.7.1	2.5.37	-	-
1.6.7	2.6.3	0.20.1	0.19.0	0.6.1	2.5.11	-	-
1.6.6	2.6.3	0.20.1	0.19.0	0.6.1	2.5.11	-	-

Streams for Apache Kafka	Apache Kafka	Strimzi Operator	Kafka Bridge	Oauth	Cruise Control	Console	Proxy
1.6.5	2.6.2	0.20.1	0.19.0	0.6.1	2.5.11	-	-
1.6.4	2.6.2	0.20.1	0.19.0	0.6.1	2.5.11	-	-
1.6.0	2.6.0	0.20.0	0.19.0	0.6.1	2.5.11	-	-
1.5.0	2.5.0	0.18.0	0.16.0	0.5.0	-	-	-
1.4.1	2.4.0	0.17.0	0.15.2	0.3.0	-	-	-
1.4.0	2.4.0	0.17.0	0.15.2	0.3.0	-	-	-
1.3.0	2.3.0	0.14.0	0.14.0	0.1.0	-	-	-
1.2.0	2.2.1	0.12.1	0.12.2	-	-	-	-
1.1.1	2.1.1	0.11.4	-	-	-	-	-
1.1.0	2.1.1	0.11.1	-	-	-	-	-
1.0	2.0.0	0.8.1	-	-	-	-	-



注記

Strimzi 0.26.0 には Log4j の脆弱性が含まれています。製品に含まれるバージョンは、脆弱性が含まれていないバージョンに依存するように更新されました。

第12章 サポート対象となる RED HAT 製品との統合

Streams for Apache Kafka 2.7 は、次の Red Hat 製品との統合をサポートしています。

Red Hat build of Keycloak

OAuth 2.0 認証と OAuth 2.0 認可を提供します。

Red Hat 3scale API Management

Kafka Bridge のセキュリティーを保護し、追加の API 管理機能を提供します。

Red Hat build of Debezium

データベースを監視し、イベントストリームを作成します。

Red Hat build of Apicurio Registry

データストリーミングのサービススキーマの集中型ストアを提供します。

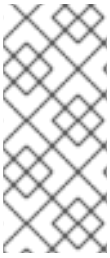
Red Hat build of Apache Camel K

軽量の統合フレームワークを提供します。

これらの製品によって Streams for Apache Kafka デプロイメントに導入できる機能の詳細は、製品ドキュメントを参照してください。

12.1. RED HAT BUILD OF KEYCLOAK (旧称 RED HAT SINGLE SIGN-ON)

Streams for Apache Kafka は、Red Hat build of Keycloak の Authorization Services による OAuth 2.0 トークンベースの認可をサポートしています。これにより、セキュリティーポリシーと権限を一元的に管理できます。



注記

Red Hat build of Keycloak は、現在メンテナンスサポート中の Red Hat Single Sign-On に代わるものです。Red Hat は、この移行を反映するために、ドキュメント、リソース、メディアの更新に取り組んでいます。当面の間、Streams for Apache Kafka ドキュメントの Single Sign-On の使用を説明している内容は、Red Hat build of Keycloak の使用にも適用されます。

12.2. RED HAT 3SCALE API MANAGEMENT

Kafka Bridge を OpenShift Container Platform にデプロイした場合は、3scale で使用できます。3scale API Management は、TLS を使用して Kafka Bridge を保護し、認証および認可を提供できます。また、3scale との統合により、メトリック、流量制御、請求などの追加機能も利用できるようになります。

3scale のデプロイについては、3scale API Management と Streams for Apache Kafka Bridge の使用 参照してください。

12.3. 変更データキャプチャーのための RED HAT BUILD OF DEBEZIUM

Red Hat build of Debezium は、分散型の変更データキャプチャープラットフォームです。データベースの行レベルの変更をキャプチャーして、変更イベントレコードを作成し、Kafka トピックにレコードをストリーミングします。Debezium は Apache Kafka 上に構築されています。Red Hat build of Debezium は、Streams for Apache Kafka にデプロイして統合できます。Streams for Apache Kafka のデプロイ後、Kafka Connect で Debezium をコネクター設定としてデプロイします。Debezium は、変更イベントレコードを OpenShift 上の Streams for Apache Kafka に渡します。アプリケーションはこれらの **変更イベントストリーム** を読み取りでき、変更イベントが発生した順にアクセスできます。

Streams for Apache Kafka を使用した Debezium のデプロイの詳細は、Red Hat build of Debezium の製品ドキュメントを参照してください。

12.4. スキーマ検証のための RED HAT BUILD OF APICURIO REGISTRY

Red Hat build of Apicurio Registry を、データストリーミングのサービススキーマの集中型ストアとして使用できます。Kafka では、Red Hat build of Apicurio Registry を使用して **Apache Avro** または JSON スキーマを格納できます。

Apicurio Registry は、REST API および Java REST クライアントを提供し、サーバー側のエンドポイントを介してクライアントアプリケーションからスキーマを登録およびクエリーします。

Apicurio Registry を使用すると、クライアントアプリケーションの設定からスキーマ管理のプロセスが分離されます。クライアントコードに URL を指定して、アプリケーションがレジストリーからスキーマを使用できるようにします。

たとえば、メッセージをシリアライズおよびデシリアライズするスキーマをレジストリーに保存できます。アプリケーションは保存されたスキーマを参照し、それらを使用して送受信するメッセージとスキーマとの互換性を維持します。

Kafka クライアントアプリケーションは、実行時に Apicurio Registry からスキーマをプッシュまたはプルできます。

Streams for Apache Kafka で Red Hat build of Apicurio Registry を使用する方法の詳細は、Red Hat build of Apicurio Registry の製品ドキュメントを参照してください。

12.5. RED HAT BUILD OF APACHE CAMEL K

Red Hat build of Apache Camel K は、Apache Camel K から構築された軽量の統合フレームワークで、OpenShift 上のクラウドでネイティブに実行されます。Camel K はサーバーレス統合をサポートしているため、基盤となるインフラストラクチャーを管理する必要なく、統合タスクの開発とデプロイメントが可能になります。Camel K を使用すると、イベント駆動型アプリケーションを構築し、Streams for Apache Kafka 環境と統合できます。異なるシステムまたはデータベース間でのリアルタイムのデータ同期が必要な状況では、Camel K を使用してイベントの変更をキャプチャーおよび変換し、それらを Streams for Apache Kafka に送信して他のシステムに配信できます。

Streams for Apache Kafka で Camel K を使用する方法の詳細は、Red Hat build of Apache Camel K の製品ドキュメントを参照してください。

関連情報

- Red Hat build of Keycloak でサポートされる構成
- Red Hat Single Sign-On でサポートされる構成 (メンテナンス)
- Red Hat 3scale API Management でサポートされる構成
- Red Hat build of Debezium でサポートされる構成
- Red Hat build of Apicurio Registry でサポートされる構成
- Red Hat build of Apache Camel K でサポートされる構成

改訂日時: 2024-11-20