Red Hat AMQ 2021.Q3

AMQ Streams 1.8 on OpenShift リリースノート

AMQ Streams on OpenShift Container Platform の使用

Last Updated: 2023-02-23

Red Hat AMQ 2021.Q3 AMQ Streams 1.8 on OpenShift リリースノート

AMQ Streams on OpenShift Container Platform の使用

Enter your first name here. Enter your surname here.
Enter your organisation's name here. Enter your organisational division here.
Enter your email address here.

法律上の通知

概要

本リリースノートには、AMQ Streams 1.8 リリースに含まれる新機能、改良された機能、修正、および問題に関する最新情報が含まれています。

目次

多様性を受け入れるオープンソースの強化

Red Hat では、コード、ドキュメント、Web プロパティーにおける配慮に欠ける用語の置き換えに取り組んでいます。まずは、マスター (master)、スレーブ (slave)、ブラックリスト (blacklist)、ホワイトリスト (whitelist) の 4 つの用語の置き換えから始めます。この取り組みは膨大な作業を要するため、今後の複数のリリースで段階的に用語の置き換えを実施して参ります。詳細は、Red Hat CTO である Chris Wright のメッセージをご覧ください。

第1章 機能

AMQ Streams バージョン 1.8 は Strimzi 0.24.x をベースにしています。

本リリースで追加され、これまでの AMQ Streams リリースにはなかった機能は次のとおりです。

注記

本リリースで解決された改良機能とバグをすべて確認するには、AMQ Streams の Jira プロジェクト を参照してください。

1.1. OPENSHIFT CONTAINER PLATFORM のサポート

AMQ Streams 1.8 は OpenShift Container Platform 4.6 および 4.8 でサポートされます。

サポートされるプラットフォームバージョンの詳細については、Red Hat ナレッジベースの記事「Red Hat AMQ 7 でサポートされる構成」を参照してください。

1.2. KAFKA 2.8.0 のサポート

AMQ Streams は Apache Kafka バージョン 2.8.0 に対応するようになりました。

AMQ Streams は Kafka 2.8.0 を使用します。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がサポートされます。

ブローカーおよびクライアントアプリケーションを Kafka 2.8.0 にアップグレードする前に、Cluster Operator を AMQ Streams バージョン 1.8 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は、「Upgrading AMQ Streams 」を参照してください。

詳細は、Kafka 2.7.0 および Kafka 2.8.0 のリリースノートを参照してください。

注記

Kafka 2.7.x は、AMQ Streams 1.8. にアップグレードする目的でのみサポートされます。

サポートされるバージョンの詳細は、カスタマーポータルの「Red Hat AMQ 7 コンポーネントの詳細」を参照してください。

Kafka バージョン 2.8.0 には ZooKeeper バージョン 3.5.9 が必要です。そのため、AMQ Streams 1.7 から AMQ Streams 1.8. にアップグレードするときに、Cluster Operator は ZooKeeper のアップグレードを実行します。

1.3. フィーチャーゲートによる機能のオンまたはオフへの切り替え

Kafka クラスター管理者は、Operator のデプロイメント設定で **フィーチャーゲート**を使用して、機能のサブセットをオンまたはオフに切り替えできるようになりました。フィーチャーゲートは現在 Cluster Operator でのみ利用可能であり、今後のリリースではフィーチャーゲートは他の operator に追加される可能性があります。

AMQ Streams 1.8 では、以下のフィーチャーゲートと関連する新機能が導入されています。

- コントロールプレーンのリスナーを切り替える **ControlPlaneListener**
- サービスアカウントのパッチを切り替える**ServiceAccountPatching**

フィーチャーゲートのデフォルトの状態は **enabled** または **disabled** になります。フィーチャーゲートを有効にすると、operator の動作が変更され、AMQStreams デプロイメントの機能が有効になります。

フィーチャーゲートには **Alpha**、**Beta**、または **Generally Available** (GA) の成熟度レベルがあります。

表1.1 フィーチャーゲートの成熟度レベル

フィーチャーゲート成熟度レベル	説明	デフォルトの状態
Alpha	実験的で不安定な機能や、実稼働向けに十分にテストされていない可能性がある機能を制御します。これらの機能は、今後のリリースで変更される可能性があります。	Disabled
Beta	十分にテストされた機能を制御します。これらの機能が今後のリリースで変更される可能性はほぼありません。	Enabled
General Availability (GA)	安定性があり、十分にテストされ、実稼働での使用に適している機能を制御します。GA 機能は、今後のリリースでは変更されません。	Enabled

フィーチャーゲートの設定

Cluster Operator のデプロイメント設定では、**STRIMZI_FEATURE_GATES** 環境変数で、フィーチャーゲート名およびプレフィックスのコンマ区切りリストを指定します。**+** プレフィックスはフィーチャーゲートを有効にし、**-** プレフィックスを無効にします。

例: コントロールプレーンリスナーフィーチャーゲートの有効化

1. Cluster Operator の Deployment を編集します。

```
oc edit deployment strimzi-cluster-operator
```

2. **STRIMZI_FEATURE_GATES** 環境変数を **+ControlPlaneListener**の値とともに追加します。

```
# ...
env:
 #...
- name: STRIMZI_FEATURE_GATES
  value: +ControlPlaneListener
#...
```

「Feature gates」および「Cluster Operator configuration」を参照してください。

1.4. コントロールプレーンリスナー

注記

この機能は、Alpha 段階の **ControlPlaneListener** フィーチャーゲートを使用して制御され、デフォルトで無効になっています。詳細は、「フィーチャーゲート」を参照してください。

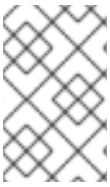
標準の AMQ Streams クラスターでは、コントロールプレーントラフィックおよびデータプレーントラフィックはいずれも、ポート 9091 で同じブローカー間リスナーを使用します。

今回のリリースでは、コントロールプレーントラフィックがポート 9090 で専用の**コントロールプレーンリスナー**を使用するようにクラスターを設定できるようになりました。データプレーントラフィックは、引き続きポート 9091 でリスナーを使用します。

コントロールプレーンリスナーを使用すると、パーティションリーダーシップの変更などの重要なコントローラーコネクションが、ブローカー全体のデータレプリケーションによって遅延されないため、パフォーマンスが向上する可能性があります。大半のデータプレーントラフィックは、このデータレプリケーションで構成されます。

「Control plane listener feature gate」を参照してください。

1.5. サービスアカウントのパッチの適用



注記

この機能は、Alpha 段階の **ServiceAccountPatching** フィーチャーゲートを使用して制御され、デフォルトで無効になっています。詳細は、「フィーチャーゲート」を参照してください。

デフォルトで、Cluster Operator はサービスアカウントを更新しません。

今回のリリースにより、調整ごとのサービスアカウントの更新適用を有効にできます。たとえば、Cluster Operator はカスタムラベルまたはアノテーションをサービスアカウントに適用できます。カスタムラベルとアノテーションは、**template.serviceAccount** プロパティーを使用してカスタムリソースに対して設定されます。

カスタムラベルおよびアノテーションの例

```
# ...
template:
  serviceAccount:
    metadata:
      labels:
        label1: value1
        label2: value2
      annotations:
        annotation1: value1
        annotation2: value2
# ...
```

「Service Account patching feature gate」を参照してください。

1.6. 変更データキャプチャー統合の DEBEZIUM

Red Hat 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 トピックへレコードをストリーミングします。Debezium は Apache Kafka に構築されます。AMQ Streams で Debezium をデプロイおよび統合できます。AMQ Streams のデプロイ後に、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 AMQ Streams に渡します。アプリケーションは **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。

Debezium には、以下を含む複数の用途があります。

- データレプリケーション。
- キャッシュの更新およびインデックスの検索。
- モノリシックアプリケーションの簡素化。
- データ統合。
- ストリーミングクエリーの有効化。

Debezium は、以下の共通データベースのコネクター (Kafka Connect をベースとする) を提供します。

- Db2
- MongoDB
- MySQL
- PostgreSQL
- SQL Server

AMQ Streams で Debezium をデプロイするための詳細は、「製品ドキュメント」を参照してください。

1.7. SERVICE REGISTRY

Service Registry は、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、Service Registry を使用して **Apache Avro** または JSON スキーマを格納できます。

Service 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。

Service 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。保存後、スキーマを使用するアプリケーションから参照され、アプリケーションが送受信するメッセージがこれらのスキーマと互換性を維持するようにします。

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

AMQ Streams で Service Registry を使用するための詳細は、Service Registry のドキュメントを参照してください。

第2章 アップグレードの要件

AMQ Streams バージョン 1.8. 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。

すべてのカスタムリソースの **v1beta2** API バージョンが AMQ Streams 1.7 で導入されました。AMQ Streams 1.8 では、**KafkaTopic** および **KafkaUser** を除くすべての AMQ Streams 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 v1.22 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 AMQ Streams 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. AMQ Streams 1.7 へのアップグレード
2. カスタムリソースを **v1beta2** に変換します。
3. AMQ Streams 1.8 へのアップグレード

「Deploying and upgrading AMQ Streams」を参照してください。

2.1. カスタムリソースの V1BETA2 バージョン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、AMQ Streams では **API 変換ツール** が提供されます。これはAMQ Streams のダウンロードサイト からダウンロードできます。

カスタムリソースのアップグレードは、2 つのステップで実行します。

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。

- AMQ Streams カスタムリソースの設定を記述する YAML ファイルの変換
- クラスターでの AMQ Streams 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。

完全な手順は、「Upgrading AMQ Streams」を参照してください。

第3章 改良された機能

このリリースで改良された機能は次のとおりです。

3.1. KAFKA 2.8.0 で改良された機能

Kafka 2.8.0 に導入された改良機能の概要は『Kafka 2.8.0 Release Notes』を参照してください。

3.2. KAFKA CONNECT ビルド設定の更新

AMQ Streams がデータコネクションに必要なコネクタープラグインでコンテナーイメージを自動的にビルドするために **build** 設定を使用できます。

専用のサービスアカウントが Kafka Connect ビルド Pod で作成されるようになりました。サービスアカウントは Kafka Connect 自体とは異なります。これまでは、ビルドはデフォルトのサービスアカウントで実行されました。認証とアクセスを指定する場合は、独自の ID があると便利です。

標準の HTTP プロキシー (**HTTP_PROXY**、**HTTPS_PROXY**、および **NO_PROXY**) が AMQ Streams デプロイメントの環境変数として設定されている場合、Kafka Connect ビルドはプロキシーの背後でも機能するようになりました。

以下を参照してください。

- Creating a new container image automatically using AMQ Streams
- **Build** スキーマ参照

3.3. 外部設定データの KUBERNETES 設定プロバイダー

Kubernetes Configuration Provider プラグインを使用して、外部ソースから設定データを読み込みます。OpenShift のシークレットまたは ConfigMap からデータをロードできます。

プロバイダーは AMQ Streams とは独立して動作します。新しい Secret または ConfigMap を使用している場合でも、Kafka コンポーネントを再起動することなくデータをロードします。

これを使用して、プロデューサーやコンシューマーを含む、すべての Kafka コンポーネントの設定データを読み込むことができます。たとえば、中断なしで複数のコネクターをホストする Kafka Connect インスタンスのクレデンシャルを提供する場合に使用します。

「Loading configuration values from external sources」を参照してください。

3.4. マーカーのあるログフィルター

ConfigMap を使用して AMQ Streams Operator のロギングレベル (log4j2) ロギングレベルを設定する場合、ロギングフィルターを定義して、ログに返される内容を制限できるようになりました。フィルタープロパティーを ConfigMap に追加します。

フィルターは**マーカー**を使用して、ログに含まれる内容を指定します。マーカーの種類、namespace、および名前を指定します。たとえば、Kafka クラスターで障害が発生した場合、種類を **Kafka** に指定してログを分離し、障害が発生しているクラスターの namespace および名前を使用します。

以下の例は、**my-kafka-cluster** という名前の Kafka クラスターのマーカーフィルターを示しています。

基本的なロギングフィルターの設定

```
appender.console.filter.filter1.type=MarkerFilter (1)
appender.console.filter.filter1.onMatch=ACCEPT (2)
appender.console.filter.filter1.onMismatch=DENY (3)
appender.console.filter.filter1.marker=Kafka(my-namespace/my-kafka-cluster) (4)
```

1. **MarkerFilter** 型は、フィルターを行うために指定されたマーカーを比較します。
2. **onMatch** プロパティーは、マーカーが一致するとログを受け入れます。
3. **onMismatch** プロパティーは、マーカーが一致しない場合にログを拒否します。
4. フィルター処理に使用されるマーカーの形式は KIND(NAMESPACE/NAME-OF-RESOURCE) です。

「Adding logging filters to Operators」を参照してください。

3.5. OAUTH 2.0 認証の改良

Audience および Scope の設定

承認サーバーからトークンを取得する際に、**clientAudience** および **clientScope** プロパティーを設定できるようになりました。プロパティーの値は、**audience** および **scope** パラメーターとしてトークンエンドポイントに渡されます。どちらのプロパティーも、**Kafka** カスタムリソースの OAuth 2.0 認証リスナー設定で設定されます。

以下のシナリオで、このプロパティーを使用します。

- ブローカー間認証用のアクセストークンを取得する場合
- **clientId およびシークレット**を使用した OAuth 2.0 over PLAIN クライアント認証のクライアントの名前
  具体的には、アクセストークンを取得するために、PLAIN コールバックが承認サーバーで最初に **clientID** (ユーザー名として) と **secret** (パスワードとして) を変換した場合に、**audience** および **scope** をリクエストに含めることができるようになりました。

これらのプロパティーは、クライアントがトークンとトークンのコンテンツを取得できるかどうかに影響します。リスナーによって課されるトークン検証ルールには影響を与えません。

client Audience および client Scope プロパティーの設定例

```
# ...
  authentication:
    type: oauth
    # ...
    clientAudience: AUDIENCE
    clientScope: SCOPE
```

承認サーバーは、JWT アクセストークンで **aud** (audience) クレームを提供することがあります。オーディエンスチェックが有効な場合、Kafka ブローカーは **aud** クレームにブローカーの **clientId** が含まれていないトークンを拒否します。オーディエンスチェックを有効にするには、**oauth** リスナー設定で **checkAudience** オプションを **true** に設定します。オーディエンスチェックはデフォルトで無効になっています。

「Configuring OAuth 2.0 support for Kafka brokers」および「KafkaListenerAuthenticationOAuth schema reference」を参照してください。

Kafka Connect および Kafka Bridge への audienceの指定

それぞれのカスタムソースで、Kafka Connect または Kafka Bridge の OAuth クライアント認証を設定するときに、**audience** オプションを指定できるようになりました。これまでのリリースでは、これらのリソースに対して **scope** オプションのみがサポートされていました。

「KafkaClientAuthenticationOAuth schema reference」を参照してください。

OAuth 2.0 over PLAIN でトークンエンドポイントを必要としない

OAuth 2.0 の PLAIN 認証に「クライアント ID および secret」メソッドを使用する場合に、The **tokenEndpointUri** オプションが不要になりました。

トークンエンドポイント URI が指定された OAuth 2.0 over PLAIN の 設定例

```
# ...
  authentication:
    type: oauth
    # ...
    enablePlain: true
    tokenEndpointUri: https://OAUTH-SERVER-ADDRESS/auth/realms/external/protocol/openid-connect/token
```

tokenEndpointUri が指定されていない場合、リスナーは以下を処理します。

- **username** パラメーターをアカウント名として。
- **password** パラメーターを検証のために承認サーバーに渡される raw アクセストークンとして (OAUTHBEARER 認証の場合と同じ動作)。

OAuth 2.0 over PLAIN 認証の「long-lived access token」メソッドの動作は変更されません。このメソッドを使用する場合は The **tokenEndpointUri** は必要ありません。

「OAuth 2.0 authentication mechanisms」を参照してください。

3.6. ユーザークォータ

User Operator を使用したユーザークォータの処理は ZooKeeper によって管理されなくなりました。その代わりに、ユーザークォータは API 経由で処理されます。

また、Kafka の mutation rate クォータにサポートが追加されました。このクォータは、1 秒あたりに許容されるパーティション変更の数を制限します。このクォータにより、Kafka クラスターが同時にトピック操作に圧倒されないようにします。

パーティションの変更数には、以下のユーザー要求の種類が含まれます。

- 新しいトピック用のパーティションの作成
- 既存のトピックへのパーティションの追加
- トピックからのパーティションの削除

mutation rate クォータを設定して、ユーザー要求に対して変更が許可されるレートを制御できます。レートは、作成または削除されたパーティション数から累積されます。

controllerMutationRate オプションを使用して、クォータを Kafka ユーザーに適用します。この例では、1 秒あたり 10 個のパーティションの作成および削除操作が許可されています。

ユーザークォータを使用した KafkaUser の設定例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaUser
metadata:
  name: my-user
  labels:
    strimzi.io/cluster: my-cluster
spec:
  # ...
  quotas:
    #...
    controllerMutationRate: 10 (1)
```

「User quotas」を参照してください。

3.7. カスタムリソースの調整を一時停止

AMQ Streams Operator によって管理されるカスタムリソースの調整を一時停止して、修正の実行や更新を行うことができます。作成するカスタムリソースの調整を一時停止することもできます。カスタムリソースは作成されますが、無視されます。

アノテーションをカスタムリソースに追加して一時停止します。

調整を一時停止するためのアノテーションの例

```
oc annotate KIND-OF-CUSTOM-RESOURCE NAME-OF-CUSTOM-RESOURCE strimzi.io/pause-reconciliation="true"
```

KafkaTopic カスタムリソースの調整を一時停止できるようになりました。

「Pausing reconciliation of custom resources」を参照してください。

3.8. KAFKA EXPORTER の更新

AMQ Streams で提供される Kafka Exporter のカスタムバージョンがバージョン 1.3.1 に更新されました。AMQ Streamsには、提供されている例（**examples/metrics/grafana-dashboards/strimzi-kafka-exporter.json**）にKafka ExporterのGrafanaダッシュボードの例が含まれています。

「Add Kafka Exporter」を参照してください。

3.9. KAFKA CONNECT ビルドによるハッシュを使用したダウンロードファイルの命名

KafkaConnect リソースを設定すると、カスタム Kafka Connect コンテナーイメージを作成できます。**spec.build** 設定を使用すると、プロセスが自動化されます。**plugins** を設定して実装アーティファクトを指定し、**output** を設定してイメージを格納するコンテナーレジストリーを参照します。AMQ Streams は、コネクタープラグインをダウンロードして、新しいコンテナーイメージに追加します。

ビルドプロセスでは URL ハッシュを使用して、ダウンロードしたアーティファクトファイルを命名す

るようになりました。以前のバージョンでは、これはダウンロード URL の最後のセグメントを使用していました。プラグインアーティファクトに特定の名前が必要な場合は、新しい **other** アーティファクトタイプとその **fileName** フィールドを使用できます。

プラグインアーティファクトの命名例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
  name: my-connect-cluster
spec:
  #...
  build:
    output:
      #...
    plugins:
      - name: my-plugin
        artifacts:
          - type: other
            url: https://my-domain.tld/my-other-file.ext
            sha512sum: 589...ab4
            fileName: name-of-file.ext
  #...
```

「**Build** スキーマ参照」を参照してください。

第4章 テクノロジープレビュー

重要

テクノロジー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。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. KAFKA STATIC QUOTA プラグインの設定

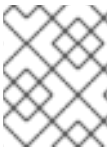
Kafka Static Quota プラグインを使用して、Kafka クラスターのブローカーにスループットおよびストレージの制限を設定します。**Kafka** リソースを設定して、プラグインを有効にし、制限を設定します。バイトレートのしきい値およびストレージクォータを設定して、ブローカーと対話するクライアントに制限を設けることができます。

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
```

Setting limits on brokers using the Kafka Static Quota plugin を参照してください。

4.2. CRUISE CONTROL によるクラスターのリバランス



注記

Cruise Control は本リリースでもテクノロジープレビューですが、新たな改良が加えられました。

Cruise Control をデプロイし、これを使用して **最適化ゴール** (CPU、ディスク、ネットワーク負荷などに定義された制約) を使用し、Kafka をリバランスできます。バランス調整された Kafka クラスターでは、ワークロードがブローカー Pod 全体に均等に分散されます。

Cruise Control は **Kafka** リソースの一部として設定され、デプロイされます。デフォルトの最適化ゴールを使用するか、要件に合わせて変更できます。Cruise Control の YAML 設定ファイルのサンプルは、**examples/cruise-control/** にあります。

Cruise Control がデプロイされると、**KafkaRebalance** カスタムリソースを作成して以下を行うことができます。

- 複数の最適化のゴールから、最適化のプロポーザルを生成します。
- 最適化のプロポーザルを基にして Kafka クラスターを再分散します。

異常検出、通知、独自ゴールの作成、トピックレプリケーション係数の変更などの、その他の Cruise Control の機能は現在サポートされていません。

「Cruise Control for cluster rebalancing」を参照してください。

4.2.1. テクノロジープレビューの改良

最適化プロポーザルの更新

既存の **KafkaRebalance** リソースを **Ready** のステータスで再利用できるようになりました。これは、クラスターのリバランスが正常に完了したことを示します。**KafkaRebalance** リソースに定義された最適化ゴールを再利用するか、ゴールを変更することができます。

最適化プロポーザルを更新するには、以下を実行します。

1. **KafkaRebalance** リソースの状態をチェックします。

```
oc describe kafkarebalance REBALANCE-NAME
```

2. **strimzi.io/rebalance=refresh** アノテーションを適用します。

```
oc annotate kafkarebalance REBALANCE-NAME strimzi.io/rebalance=refresh
```

Cruise Control は最適化プロポーザルを更新し、Kafka クラスターの最新状態を反映します。

「Approving an optimization proposal 」を参照してください。

最適化プロポーザルでのブローカー負荷の表示

最適化プロポーザルは要約ステータスに加えて**ブローカー負荷**で構成されるようになりました。ブローカーの負荷が ConfigMap 内で返され、CPU 使用率、ディスク使用量、ネットワーク出力レートなどの各 Kafka ブローカーの負荷のメトリクスが表示されます。メトリクスは 3 つのカテゴリーに分類されます。

before
: 最適化プロポーザルを適用する前の現在の値

after
: 最適化プロポーザルの適用後に想定される値

difference
: after 値と before 値の差

ブローカー負荷 ConfigMap の名前は **KafkaRebalance** リソースと同じになります。メトリクスは JSON 文字列としてエンコードされます。人が判読できる形式で表示するには、**jq** または同様の JSON パーサーを使用します。以下は例になります。

```
oc get configmap MY-REBALANCE -o json | jq '.["data"]["brokerLoad.json"]|fromjson|.'
```

「Optimization proposals overview」を参照してください。

第5章 非推奨の機能

このリリースで非推奨となり、これまでの AMQ Streams リリースではサポートされていた機能は次のとおりです。

5.1. S2I (SOURCE-TO-IMAGE) 対応の KAFKA CONNECT

build 設定が KafkaConnect リソースに導入されたため、AMQ Streams はデータコネクションに必要なコネクタープラグインでコンテナーイメージを自動的にビルドできるようになりました。

そのため、S2I (Source-to-Image) 対応の Kafka Connect のサポートが非推奨になりました。

この変更に備えるため、Kafka Connect S2I インスタンスを Kafka Connect インスタンスに移行できます。

「Migrating from Kafka Connect with S2I to Kafka Connect 」を参照してください。

5.2. ENABLEECDSA プロパティー

enableECDSA プロパティーは、Kafka ブローカーの **oauth** リスナー設定で非推奨になりました。このプロパティーの値は無視されるようになりました。

ECDSA 暗号化は、JCE (Java Cryptography Extension) で利用できます。Bouncy Castle Crypto ライブラリーは AMQ Streams でパッケージ化されなくなりました。

5.3. 多様性を考慮する用語

KafkaMirrorMaker2 リソースの**topicsBlacklistPattern**および**groupsBlacklistPattern**プロパティは非推奨になりました。プロパティは削除され、**topicsExcludePattern**と**groupsExcludePattern**に置き換えられます。

KafkaMirrorMakerリソースの**whitelist**プロパティは非推奨になりました。このプロパティーは **include** に置き換えられます。

5.4. プラグインアーティファクトファイルの KAFKA CONNECT の命名

Kafka Connect のビルドプロセスでは URL ハッシュを使用して、ダウンロードしたアーティファクトファイルを命名するようになりました。以前のバージョンでは、これはダウンロード URL の最後のセグメントを使用していました。

「Kafka Connect build uses hash to name download files」 を参照してください。

5.5. 非推奨となり削除された KAFKA 機能

本セクションでは、Apache Kafka プロジェクトで非推奨となり、削除される重要な機能を事前に報告します。

5.5.1. Kafka バージョン 3.0 で削除される予定の機能

Kafka バージョン 3.0 は、AMQ Streams の次回のメジャーリリースに同梱されます。

以下の表は、Kafka 2.x 以前で非推奨となり、Kafka 3.0 で**削除**される予定のメソッドやコンポーネントを示しています。このリストは包括的ではありません。

表5.1 Kafka 3.0 で削除される予定の非推奨の API メソッドおよびコンポーネント

API またはコンポーネント	課題へのリンク	説明
Admin API	KAFKA-12581	非推奨の Admin.electPreferredLeaders の削除
Admin API	KAFKA-6987	KafkaFuture を CompletableFuture で再実装 (KafkaFuture.Function は非推奨)
Admin client	KAFKA-12577	非推奨の **ConfigEntry** コンストラクターの削除
すべてのクライアント	KAFKA-12579	3.0 のクライアントからさまざまな非推奨メソッドを削除
すべてのクライアント	KAFKA-12600	クライアント設定 client.**dns.lookup**の非推奨の設定値の**デフォルト** を削除します。
すべてのクライアント	KAFKA-12578	非推奨のセキュリティークラス/メソッドの削除
Broker	KAFKA-12591	非推奨の **quota.producer.default** および **quota.consumer.default** の設定を削除します。
ブローカー	KAFKA-12592	非推奨の LogConfig.Compact の削除
Broker	KAFKA-12590	非推奨の SimpleAclAuthorizer の削除
Broker	KAFKA-5905	PrincipalBuilder および DefaultPrincipalBuilder の削除
Common	KAFKA-12573	非推奨の **Metric#value** を削除
Consumer API	KAFKA-12637	非推奨の PartitionAssignor インターフェースの削除
Connect API	KAFKA-12482	非推奨の rest.host.name および rest.port Connect ワーカー設定の削除
Connect API	KAFKA-12945	3.0 で port、host.name、および関連設定を削除

API またはコンポーネント	課題へのリンク	説明
Connect API	KAFKA-12717	内部コンバーター設定プロパティーの削除
Streams API	KAFKA-12574	eos-alpha の非推奨
Streams API	KAFKA-12808	StreamsMetrics で非推奨となったメソッドの削除
Streams API	KAFKA-7606	StreamsResetter から非推奨のオプションを削除
Streams API	KAFKA-12796	**streams-scala**の非推奨のクラスの削除
Streams API	KAFKA-12419	3.0 での Kafka Streams の非推奨 API の削除
Streams API	KAFKA-10434	WindowStore での非推奨メソッドの削除
Streams API	KAFKA-12449	非推奨の WindowStore#put の削除
Streams API	KAFKA-12813	ProcessorContext の非推奨のスケジュールメソッドの削除
Streams API	KAFKA-12809	Stores の非推奨のメソッドの削除
Streams API	KAFKA-12814	非推奨メソッド StreamsConfig#getConsumerConfig の削除
Streams API	KAFKA-12313	default.windowed.serde.inner.class 設定を非推奨
Streams API	KAFKA-8372	非推奨の RocksDB#compactRange API の削除
Streams API	KAFKA-12584	非推奨の **Sum** および **Total** クラスの削除
Streams API	KAFKA-12683	非推奨の "UsePreviousTimeOnInvalidTimeStamp" の削除

API またはコンポーネント	課題へのリンク	説明
Streams API	KAFKA-12810	非推奨の TopologyDescription.Source#topics の削除
Streams API	KAFKA-12630	非推奨の KafkaClientSupplier#getAdminClient の削除
Streams API	KAFKA-10046	非推奨の PartitionGrouper 設定は無視される
Streams API	KAFKA-12633	Remove deprecated "TopologyTestDriver#pipeInput / readOutput"
Streams API	KAFKA-12441	非推奨メソッド StreamsBuilder#addGlobalStore の削除
Streams API	KAFKA-12452	ProcessorContext#forward の非推奨オーバーロードの削除
Streams API	KAFKA-12450	ReadOnlyWindowStore から非推奨のメソッドを削除
Streams API	KAFKA-12880	3.0 で非推奨の Count と SampledTotal の削除
Streams API	KAFKA-12451	WindowStore の長期ベースの読み取り操作の非推奨アノテーションを削除
Streams API	KAFKA-12568	非推奨の「KStream#groupBy/join」、「Joined#named」オーバーロードの削除
Streams API	KAFKA-12849	TaskMetadata を内部実装のインターフェースに移行
Streams API	KAFKA-7785	PartitionGrouper インターフェースと設定を削除し、DefaultPartitionGrouper を内部パッケージに移動。

API またはコンポーネント	課題へのリンク	説明
Streams API	KAFKA-7106	Window 定義から segment/segmentInterval を削除
Streams API	KAFKA-8897	RocksDB のバージョンの増加
Streams API	KAFKA-12909	ユーザーに誤った left/outer stream-stream join の改善の選択を許可
Tools	KAFKA-8405	非推奨の **kafka-preferred-replica-election** コマンドの削除
Tools	KAFKA-12588	shell コマンドで非推奨の --zookeeper を削除

5.5.2. Kafka バージョン 4.0 で削除予定の Mirror Maker 1.0

Kafka バージョン 4.0 は、今後の AMQ Streams メジャーリリースに同梱される予定です。

以下の表は、Kafka 3.0 で非推奨となり、Kafka 4.0 で**削除**される予定の機能を示しています。

表5.2 Kafka 3.0 で非推奨となり Kafka 4.0 で削除される予定のコンポーネント

コンポーネント	課題へのリンク	概要
Mirror Maker 1.0	KAFKA-12436	MirrorMaker v1 の非推奨

第6章 修正された問題

AMQ Streams 1.8 で修正された問題を以下の表に示します。Kafka 2.8.0 で修正された問題の詳細は、『Kafka 2.8.0 Release Notes』を参照してください。

課題番号	説明
ENTMQST-1529	FileStreamSourceConnector が大きなファイルを使用する際に停止します。
ENTMQST-2359	Kafka Bridge が割り当ておよびサブスクリプションを処理しない。
ENTMQST-2453	理由がないため、**kafka-exporter** Pod を再起動します。
ENTMQST-2459	Kafka Exporter の実行により、CPU の使用率が高くなります。
ENTMQST-2511	ローリングアップデート中に Kafka Exporter の再起動を停止するようにヘルスチェックを微調整します。
ENTMQST-2777	ENTMQST-2777 Custom Bridge ラベルは、サービステンプレートが指定されていない場合に設定されません。
ENTMQST-2974	Kafka Connect コネクターのログレベルの変更は一時的のみ機能します。

表6.1 CVE (Common Vulnerabilities and Exposures) の修正

課題番号	説明
ENTMQST-1934	CVE-2020-9488 log4j: SMTP アペンダーのホスト不一致による証明書の不適切な検証 [amq-st-1]
ENTMQST-2613	CVE-2020-13949 libthrift: 信頼できないペイロードを処理する場合に DoS の可能性 [amq-st-1]
ENTMQST-2617	CVE-2021-21290 netty: ローカルシステムの一時ディレクトリーを介した情報の公開 [amq-st-1]
ENTMQST-2647	CVE-2021-21295 netty: 検証の欠落により HTTP/2 でのリクエストスマグリングの可能性 [amq-st-1]
ENTMQST-2663	CVE-2021-27568 json-smart: キャッチされない例外によりクラッシュまたは情報公開が発生 [amq-st-1]

課題番号	説明
ENTMQST-2711	ENTMQST-2711 CVE-2021-21409 netty: コンテンツ長ヘッダー経由のリクエストスマグリング [amq-st-1]
ENTMQST-2821	CVE-2021-28168 jersey-common: jersey: システムの一時ディレクトリーによるローカル情報の公開 [amq-st-1]
ENTMQST-2867	CVE-2021-29425 commons-io: apache-commons-io: Apache Commons IO 2.2 から 2.6 への制限されたパストラバーサル [amq-st-1]
ENTMQST-2908	ENTMQST-2908 CVE-2021-28165 jetty-server: jetty: 無効な大きな TLS フレームの受信によるリソースの枯渇 [amq-st-1]
ENTMQST-2909	CVE-2021-28164 jetty-server: jetty: あいまいなパスが WEB-INF にアクセス可能 [amq-st-1]
ENTMQST-2910	CVE-2021-28163 jetty-server: jetty: Symlink による webapp ディレクトリーの内容公開 [amq-st-1].
ENTMQST-2980	CVE-2021-28169 jetty-server: jetty: ConcatServlet および WelcomeFilter への要求が WEB-INF ディレクトリー内の保護されているリソースにアクセスできる [amq-st-1]
ENTMQST-3023	CVE-2021-34428 jetty-server: jetty: SessionListener により、セッションがログアウトの破損を非検証しなくなる [amq-st-1]。

第7章 既知の問題

ここでは、AMQ Streams 1.8. の既知の問題について説明します。

7.1. LOG4J の SMTP アペンダー

AMQ Streams には、潜在的に脆弱なバージョンの log4j (**log4j-1.2.17.redhat-3**) が同梱されています。脆弱性は、デフォルト設定で AMQ Streams によって使用されない SMTP アペンダー機能にあります。

表7.1 CVE の問題

課題番号	説明
ENTMQST-1934	CVE-2020-9488 log4j: SMTP アペンダーのホスト不一致による証明書の不適切な検証 [amq-st-1]

回避策

SMTP アペンダーを使用している場合は、**mail.smtp.ssl.checkserveridentity** が **true** に設定されていることを確認します。

7.2. IPV6 クラスターの AMQ STREAMS CLUSTER OPERATOR

AMQ Streams Cluster Operator は、IPv6 (Internet Protocol version 6) 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一覧表示します。

   ```
   oc get subs -n OPERATOR-NAMESPACE
   ```

3. AMQ Streams 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。以下は例になります。

   ```
   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一覧表示します。

```
oc get subs -n OPERATOR-NAMESPACE
```

2. AMQ Streams 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。以下は例になります。

```
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
```

第8章 サポートされる統合製品

AMQ Streams 1.8 は、以下の Red Hat 製品との統合をサポートします。

Red Hat Single Sign-On 7.4 以降

OAuth 2.0 認証および OAuth 2.0 承認を提供します。

Red Hat 3scale API Management 2.6 以降

Kafka Bridge をセキュアにし、追加の API 管理機能を提供します。

Red Hat Debezium 1.5

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 Service Registry 2.0

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。

これらの製品によって AMQ Streams デプロイメントに導入可能な機能の詳細は、AMQ Streams 1.8 のドキュメントを参照してください。

その他のリソース

- Red Hat Single Sign-On でサポートされる構成
- Red Hat 3scale API Management のサポート対象構成
- Red Hat Debezium でサポートされる構成
- Red Hat Service Registry Supported Configurations

第9章 重要なリンク

- Red Hat AMQ 7 でサポートされる構成
- Red Hat AMQ 7 コンポーネントの詳細

改訂日時： 2021-12-18 13:39:53 +1000