Red Hat AMQ 2021.Q2

AMQ Streams 1.7 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.Q2 AMQ Streams 1.7 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 1.7 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 4

第1章 機能 5
1.1. OPENSHIFT CONTAINER PLATFORM のサポート 5
1.2. KAFKA 2.7.0 のサポート 5
1.3. V1BETA2 API バージョンの導入 5
1.3.1. カスタムリソースの v1beta2 へのアップグレード 6
1.3.2. kubectl apply commands with v1beta2 6
1.4. マルチバージョン製品のアップグレード 6
1.5. KAFKA CONNECT ビルドの設定 7
1.6. メトリクスの設定 7
1.7. 変更データキャプチャー統合の DEBEZIUM 9
1.8. SERVICE REGISTRY 9

第2章 改良された機能 11
2.1. KAFKA 2.7.0 で改良された機能 11
2.2. DEPLOYMENT ストラテジーの設定 11
2.3. CA SECRET での所有者参照の無効化 11
2.4. KAFKA ユーザーの SECRET 名のプレフィックス 12
2.5. CLUSTER OPERATOR による個別の KAFKA および ZOOKEEPER POD のローリングアップデート 12
2.6. TOPIC OPERATOR のトピックストア 12
2.7. JAAS 設定 13
2.8. KAFKA ステータスのクラスター識別 13
2.9. KAFKA CONNECT のステータス 14
2.10. 読み取り専用のルートファイルシステムでの AMQ STREAMS の実行 14
2.11. サンプル YAML ファイルによるブローカー間プロトコルバージョンの指定 14
2.12. ネットワークポリシーによる CLUSTER OPERATOR アクセスの制限 15
2.13. ラベルおよびアノテーションの SECRET への追加 15
2.14. カスタムリソースの調整の一時停止 16
2.15. コネクターおよびタスクの再起動 16
2.16. OAUTH 2.0 認証および承認 17
JWT アクセストークンのチェック 17
SASL PLAIN 認証での OAuth 2.0 のサポート 17
2.17. KAFKA CONNECT の ラック プロパティー 19
2.18. POD トポロジー分散制約 19

第3章 テクノロジープレビュー 21
3.1. CRUISE CONTROL によるクラスターのリバランス 21
3.1.1. テクノロジープレビューの改良 21

第4章 非推奨の機能 23
4.1. S2I (SOURCE-TO-IMAGE) 対応の KAFKA CONNECT 23
4.2. メトリクスの設定 23
4.3. API バージョン 23
4.4. アノテーション 24

第5章 修正された問題 25

第6章 既知の問題 28
6.1. CLUSTER OPERATOR の IPV6 クラスターへのデプロイに関する問題 28
6.2. KAFKA BRIDGE サービスの 3SCALE 検出の問題 29

第7章 サポート対象のインテグレーション製品 31

第8章 重要なリンク 32

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 1.7 は Strimzi 0.22.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 1.7 は OpenShift Container Platform 4.6 および 4.7 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 2.7.0 のサポート

AMQ Streams は Apache Kafka バージョン 2.7.0 に対応するようになりました。

AMQ Streams は Kafka 2.7.0 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 2.7.0 にアップグレードする前に、Cluster Operator を AMQ Streams バージョン 1.7 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は、「AMQ Streams のアップグレード」を参照してください。

詳細は、Kafka 2.6.0 および Kafka 2.7.0 のリリースノートを参照してください。

注記

Kafka 2.6.x は、AMQ Streams 1.7 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 2.7.0 には、Kafka 2.6.x と同じ ZooKeeper バージョン (ZooKeeper バージョン 3.5.8) が必要です。そのため、AMQ Streams 1.6 から AMQ Streams 1.7 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**しません**。

1.3. V1BETA2 API バージョンの導入

AMQ Streams 1.7 では、AMQ Streams カスタムリソースのスキーマを更新する **v1beta2** API バージョンが導入されました。古い API バージョンは非推奨になりました。

AMQ Streams 1.7 にアップグレードした後、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。これは、アップグレード後にいつでも実行できますが、**AMQ Stremas の次回のマイナーバージョン更新 (AMQ Streams 1.8) までに完了する必要があります。**

カスタムリソースのアップグレードをサポートするため、**Red Hat AMQ Streams 1.7.0 API 変換ツール**が提供されます。AMQ Streams のダウンロードサイト から API 変換ツールをダウンロードできます。ツールの使用方法は、ドキュメントおよび提供される readme に記載されています。

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** が AMQ Streams によって準備されます。

1.3.1. 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。

すべての手順は「AMQ Streams カスタムリソースのアップグレード」を参照してください。

1.3.2. kubectl apply commands with v1beta2

カスタムリソースを **v1beta2** にアップグレードした後に、**kubectl apply** コマンドは一部のタスクの実行時に機能しなくなりました。**v1beta2** カスタムリソースの大きなファイルサイズに対応するには、別のコマンドを使用する必要があります。

AMQ Streams Operator のデプロイ時に、**kubectl apply -f** の代わりに **kubectl create -f** を使用します。

以下の場合は **kubectl apply -f** の代わりに **kubectl replace -f** を使用します。

- Cluster Operator のアップグレード
- Cluster Operator の以前のバージョンへのダウングレード

作成しているカスタムリソースがすでに存在する場合（別の namespace ですでにインストールされている場合など）は、**kubectl replace** を使用します。カスタムリソースが存在しない場合は、**kubectl create** を使用します。

「AMQ Streams のデプロイおよびアップグレード」を参照してください。

1.4. マルチバージョン製品のアップグレード

1 回のアップグレードで、以前の AMQ Streams バージョンから直接最新の AMQ Streams バージョンにアップグレードできるようになりました。たとえば、中間のバージョンを省略して、AMQ Streams 1.5 から直接 AMQ Streams 1.7 にアップグレードできます。

「AMQ Streams のアップグレード」を参照してください。

1.5. KAFKA CONNECT ビルドの設定

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、ビルド設定を使用できるようになりました。

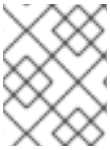
AMQ Streams で新しいイメージを自動的に作成するには、**ビルド** 設定にはコンテナーイメージを保存するコンテナーレジストリーを参照する **出力** プロパティーと、イメージに追加するコネクター **プラグイン** とそれらのアーティファクトをリストするプラグインプロパティーが必要です。

ビルド 設定はイメージストリームを参照することもできます。イメージストリームは、OpenShift Container Platform の統合レジストリーに保存されているコンテナーイメージを参照します。

出力 プロパティーは、イメージのタイプおよび名前を記述し、任意でコンテナーレジストリーへのアクセスに必要なクレデンシャルが含まれる Secret の名前を記述します。**plugins** プロパティーは、アーティファクトのタイプとアーティファクトのダウンロード元となる URL を記述します。さらに、SHA-512 チェックサムを指定して、アーティファクトを展開する前に検証することもできます。

新しいイメージを自動的に作成する Kafka Connect の設定例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
  name: my-connect-cluster
spec:
  # ...
  build:
    output:
      type: docker
      image: my-registry.io/my-org/my-connect-cluster:latest
      pushSecret: my-registry-credentials
    plugins:
      - name: debezium-postgres-connector
        artifacts:
          - type: tgz
            url: https://ARTIFACT-ADDRESS.tgz
            sha512sum: HASH-NUMBER-TO-VERIFY-ARTIFACT
      # ...
  #...
```



注記

4章 非推奨の機能 の説明にあるように、S2I (Source-to-Image) での Kafka Connect のサポートは非推奨になりました。

以下を参照してください。

- AMQ Streams を使用した新しいコンテナーイメージの自動作成
- Build スキーマ参照

1.6. メトリクスの設定

メトリクスは、カスタムリソースのデプロイ時に自動作成される ConfigMap を使用して設定されるようになりました。

metricsConfig プロパティーを使用して、Kafka コンポーネントの Prometheus メトリクスを有効化および設定します。**metricsConfig** プロパティーには、Prometheus JMX exporter の追加設定が含まれる ConfigMap への参照が含まれます。

Kafka のメトリクス設定例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
    metricsConfig:
      type: jmxPrometheusExporter
      valueFrom:
        configMapKeyRef:
          name: my-config-map
          key: my-key
    # ...
  zookeeper:
    # ...
```

ConfigMap は、JMX Prometheus エクスポーターの YAML 設定をキーの下に保存します。

Kafka のメトリクス設定が含まれる ConfigMap の例

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: my-configmap
data:
  my-key: |
    lowercaseOutputName: true
    rules:
    # Special cases and very specific rules
    - pattern: kafka.server<type=(.+), name=(.+), clientId=(.+), topic=(.+), partition=(.*)><>Value
      name: kafka_server_$1_$2
      type: GAUGE
      labels:
       clientId: "$3"
       topic: "$4"
       partition: "$5"
    # further configuration
```

追加設定なしで Prometheus メトリクスのエクスポートを有効にするには、**metricsConfig.valueFrom.configMapKeyRef.key** 配下に空のファイルが含まれる ConfigMap を参照します。空のファイルを参照する場合、名前が変更されていない限り、すべてのメトリクスが公開されます。

4章 非推奨の機能 の説明にあるように、**spec.metrics** プロパティーは非推奨になりました。

「共通の設定プロパティー」を参照してください。

1.7. 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 トピックに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 での Debezium のデプロイについて、詳しくは 製品ドキュメント を参照してください。

1.8. 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。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、「製品ドキュメント」を参照してください。

第2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。

2.1. KAFKA 2.7.0 で改良された機能

Kafka 2.7.0 に導入された改良機能の概要は『Kafka 2.7.0 Release Notes』を参照してください。

2.2. DEPLOYMENT ストラテジーの設定

Kafka Connect、MirrorMaker、および Kafka Bridge の Deployment ストラテジーを設定できるようになりました。

RollingUpdate ストラテジーはデフォルトですべてのリソースに使用されます。Kafka クラスターのローリングアップデート中に、**Deployment** の古い Pod と新しい Pod が並行して実行されます。これは、ほとんどのユースケースに最適なストラテジーです。

リソースの消費を減らすには、**再作成** ストラテジーを選択します。このストラテジーでは、ローリングアップデート中に、**Deployment** の古い Pod は新規 Pod が作成される前に終了します。

KafkaConnect、**KafkaMirrorMaker**、**KafkaMirrorMaker2**、および **KafkaBridge** リソースの **spec.template.deployment** で Deployment ストラテジーを設定します。

Kafka Connect の 再作成 デプロイメントストラテジーの例

```
apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaConnect
metadata:
  name: my-connect-cluster
spec:
  #...
  template:
    deployment:
      deploymentStrategy: Recreate
  #...
```

spec.template.deployment が設定されていない場合は、**RollingUpdate** ストラテジーが使用されます。

「**DeploymentTemplate** スキーマ参照」を参照してください。

2.3. CA SECRET での所有者参照の無効化

クラスターおよびクライアント CA Secret は、**Kafka** カスタムリソースに設定される **ownerReference** フィールドで作成されます。

generateSecretOwnerReference: false プロパティーを Kafka クラスター設定に追加することで、CA Secret **ownerReference** を無効にすることができます。CA Secret の **ownerReference** が無効になっている場合、対応する **Kafka** カスタムリソースが削除されると Secret は OpenShift によって削除されません。その後、CA Secret は新しい Kafka クラスターで再利用できます。

クラスターおよびクライアント CA Secret で ownerReference を無効にする設定例

```
apiVersion: kafka.strimzi.io/v1beta1
kind: Kafka
# ...
spec:
# ...
  clusterCa:
    generateCertificateAuthority: true
    generateSecretOwnerReference: false
  clientsCa:
    generateCertificateAuthority: true
    generateSecretOwnerReference: false
# ...
```

「CA Secret での **ownerReference** の無効化」を参照してください。

2.4. KAFKA ユーザーの SECRET 名のプレフィックス

secretPrefix プロパティーを使用して、**KafkaUser** リソース用に作成されたすべてのシークレット名にプレフィックスを追加する User Operator を設定できるようになりました。

例として、以下の設定を見てみましょう。

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
  zookeeper:
    # ...
  entityOperator:
    # ...
    userOperator:
      secretPrefix: kafka-
    # ...
```

my-user という名前のユーザーに、**kafka-my-user** という名前のシークレットを作成します。

「**EntityUserOperatorSpec** スキーマ参照」を参照してください。

2.5. CLUSTER OPERATOR による個別の KAFKA および ZOOKEEPER POD のローリングアップデート

アノテーションを使用すると、Kafka クラスターまたは ZooKeeper クラスター StatefulSets の一部である既存 Pod のローリングアップデートを手動でトリガーできます。同じ StatefulSet の複数の Pod に同時にアノテーションが付けられると、連続したローリングアップデートは同じ調整実行内で実行されます。

「Pod アノテーションを使用したローリングアップデートの実行」を参照してください。

2.6. TOPIC OPERATOR のトピックストア

AMQ Streams では、ZooKeeper を使用してトピックメタデータを保存しなくなりました。Karaf メタデータは Kafka クラスターに格納され、Topic Operator の制御下に置かれるようになりました。

この変更は、将来的に ZooKeeper が Kafka の依存関係でなくなることを想定して、AMQ Streams を準備するために必要です。

Topic Operator は永続ストレージを使用して、トピック設定をキーと値のペアとして記述するトピックメタデータを保存するようになりました。トピックメタデータは、ローカルのインメモリーでアクセスされます。ローカルのインメモリートピックストアに適用される操作からの更新は、ディスク上のバックアップトピックストアに永続化されます。トピックストアは、Kafka トピックからの更新と継続的に同期されます。

AMQ Streams 1.7 にアップグレードする場合、Topic Operator によってトピックストアが制御されるようにシームレスに移行されます。メタデータは ZooKeeper から検出および移行され、古いストアから削除されます。

新しい内部トピック

トピックストアでのトピックメタデータの処理をサポートするため、AMQ Streams 1.7 へのアップグレード時に 2 つの内部トピックが Kafka クラスターに作成されます。

内部トピック名	説明
__strimzi_store_topic	トピックメタデータを保存するためにトピックを入力します。
__strimzi-topic-operator-kstreams-topic-store-changelog	圧縮されたトピックストア値のログを維持します。

警告

これらのトピックは、Topic Operator の実行に不可欠であるため、削除しないでください。

「Topic Operator のトピックストア」を参照してください。

2.7. JAAS 設定

sasl.jaas.config プロパティーの JAAS 設定文字列が、SCRAM-SHA-512 認証のある **KafkaUser** の生成されたシークレットに追加されました。

「SCRAM-SHA-512 認証」を参照してください。

2.8. KAFKA ステータスのクラスター識別

KafkaStatus スキーマは、Kafka クラスターの識別のために **clusterId** が含まれるように更新されます。Kafka リソースの **status** プロパティーによって、Kafka クラスターのステータス情報を提供します。

Kafka status プロパティー

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
  # ...
status:
  conditions:
    lastTransitionTime: "YEAR-MONTH-20T11:37:00.706Z"
    status: "True"
    type: Ready
  observedGeneration: 1
  clusterId: CLUSTER-ID
  # ...
```

Kafka リソースのステータスを取得すると、Kafka クラスターの ID も返されます。

```
oc get kafka MY-KAFKA-CLUSTER -o jsonpath='{.status}'
```

Kafka リソースのクラスター ID のみを取得することもできます。

```
oc get kafka MY-KAFKA-CLUSTER -o jsonpath='{.status.clusterId}'
```

「KafkaStatus スキーマ参照」 および「 カスタムリソースのステータスの検索」を参照してください。

2.9. KAFKA CONNECT のステータス

KafkaConnector リソースのステータスを取得すると、コネクターによって使用されるトピックのリストが **topics** プロパティーで返されるようになりました。

「 **KafkaConnectorStatus** スキーマ参照 」および「 カスタムリソースのステータスの検索」を参照してください。

2.10. 読み取り専用のルートファイルシステムでの AMQ STREAMS の実行

読み取り専用のルートファイルシステムで AMQ Streams を実行できるようになりました。一時ファイルがマウントされた **/tmp** ファイルに書き込まれるように、追加のボリュームが追加されました。以前は、**/tmp** ディレクトリーがコンテナーから直接使用されていました。

このやり方では、コンテナーファイルシステムを変更する必要はなく、AMQ Streams を読み取り専用のルートファイルシステムからスムーズに実行することができます。

2.11. サンプル YAML ファイルによるブローカー間プロトコルバージョンの指定

AMQ Streams で提供される Kafka 設定ファイルのサンプルで **inter.broker.protocol.version** が指定されるようになりました。Kafka **設定** の **inter.broker.protocol.version** および **log.message.format.version** プロパティーは、指定された Kafka バージョン(**spec.kafka.version**)によってサポートされるバージョンです。プロパティーは、メッセージに追加されるログ形式のバージョンと、Kafka クラスターで使用されるプロトコルのバージョンを表します。Kafka バージョンのアップグレード時に、これらのプロパティーの更新が必要になります。

指定の Kafka バージョン

-

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    version: 2.7.0
    #...
    config:
      #...
      log.message.format.version: 2.7
      inter.broker.protocol.version: 2.7
```

「Kafka のアップグレード」を参照してください。

2.12. ネットワークポリシーによる CLUSTER OPERATOR アクセスの制限

Cluster Operator は、管理するリソースと同じ namespace または別の namespace で実行できます。2 つの新しい環境変数によって、Cluster Operator にアクセスできる namespace が制御されるようになりました。

デフォルトでは、**STRIMZI_OPERATOR_NAMESPACE** 環境変数が Kubernetes Downward API を使用して Cluster Operator が稼働している namespace を検索するように設定されます。Cluster Operator がリソースと同じ namespace で実行されている場合は、ローカルアクセスのみが必要で、Strimzi によって許可されます。

Cluster Operator が管理するリソースとは別の namespace で実行されている場合、ネットワークポリシーが設定されている場合を除き、Kubernetes クラスターのすべての namespace は Cluster Operator へのアクセスが許可されます。オプションの **STRIMZI_OPERATOR_NAMESPACE_LABELS** 環境変数を使用して、namespace ラベルを使用して Cluster Operator のネットワークポリシーを確立します。namespace ラベルを追加すると、Cluster Operator へのアクセスは指定された namespace に限定されます。

Cluster Operator デプロイメントに設定されたネットワークポリシー

```
#...
env:
  - name: STRIMZI_OPERATOR_NAMESPACE_LABELS
    value: label1=value1,label2=value2
  #...
```

「Cluster Operator の設定」を参照してください。

2.13. ラベルおよびアノテーションの SECRET への追加

Kafkaカスタムリソースで**clusterCaCert**テンプレートプロパティを構成することで、クラスタオペレータが作成したクラスタCAシークレットにカスタムラベルやアノテーションを追加することができます。ラベルとアノテーションは、オブジェクトを特定し、コンテキスト情報を追加するのに便利です。Strimzi カスタムリソースでテンプレートプロパティーを設定します。

ラベルおよびアノテーションを Secret に追加するテンプレートのカスタマイズ例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
```

```
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
    template:
      clusterCaCert:
        metadata:
          labels:
            label1: value1
            label2: value2
          annotations:
            annotation1: value1
            annotation2: value2
    # ...
```

「OpenShift リソースのカスタマイズ」を参照してください。

2.14. カスタムリソースの調整の一時停止

設定で **strimzi.io/pause-reconliation アノテーションを true に設定すると、カスタムリソースの調整**を一時停止できます。たとえば、Cluster Operator による調整が一時停止されるように、アノテーションを **KafkaConnect** リソースに適用できます。

一時停止された調整条件タイプを持つカスタムリソースの例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
  annotations:
    strimzi.io/pause-reconciliation: "true"
    strimzi.io/use-connector-resources: "true"
  creationTimestamp: 2021-03-12T10:47:11Z
  #...
spec:
  # ...
status:
  conditions:
  - lastTransitionTime: 2021-03-12T10:47:41.689249Z
    status: "True"
    type: ReconciliationPaused
```

重要

現在、**KafkaTopic** リソースの調整を一時停止することはできません。

「カスタムリソースの調整の一時停止」を参照してください。

2.15. コネクターおよびタスクの再起動

関連するカスタムリソースで Kubernetes アノテーションを使用して、コネクターインスタンスとそのタスクを再起動できるようになりました。

Kafka Connect と Mirror Maker 2.0 の両方のコネクターを再起動できます。MirrorMaker 2.0 は、Kafka Connect フレームワークを使用して、ソースとターゲットの Kafka クラスター間でデータをレプリケートします。

- Kafka Connect コネクターを再起動するには、該当する **KafkaConnector** カスタムリソースにアノテーションを付けます。
- MirrorMaker 2.0 コネクターを再起動するには、対応する **KafkaMirrorMaker2** カスタムリソースにアノテーションを付けます。

アノテーションを使用して、コネクターの指定タスクを再起動することもできます。

Kafka Connect の場合は、「Kafka コネクターの再起動の実行」と「Kafka コネクタータスクの再起動の実行」を参照してください。

MirrorMaker 2.0 の場合は、「Kafka MirrorMaker 2.0 コネクターの再起動の実行」 と「Kafka MirrorMaker 2.0 コネクタータスクの再起動の実行」を参照してください。

2.16. OAUTH 2.0 認証および承認

本リリースには、AMQ Streams で OAuth 2.0 トークンベースの認証および承認に対して改良された以下の機能が含まれています。

JWT アクセストークンのチェック

JWT アクセストークンに、2 つの追加チェックを設定できるようになりました。これらのチェックはいずれも Kafka ブローカーリスナーの OAuth 2.0 設定で設定されます。

カスタムクレームチェック

カスタムクレームチェックでは、Kafka ブローカーによる JWT アクセストークンの検証にカスタムルールが適用されます。これらは JsonPath フィルタークエリーを使用して定義されます。

アクセストークンに必要なデータが含まれていないと拒否されます。**イントロスペクションエンドポイント**トークン検証を使用する場合は、カスタムチェックがイントロスペクションエンドポイントの応答 JSON に適用されます。

カスタムクレームチェックを設定するには、**customClaimCheck** オプションを追加して JsonPath フィルタークエリーを定義します。カスタムクレームチェックはデフォルトで無効になっています。

「Kafka ブローカーの OAuth 2.0 サポートの設定」を参照してください。

オーディエンスチェック

承認サーバーは、JWT アクセストークンに **aud** (オーディエンス) 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、**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。

「Kafka ブローカーの OAuth 2.0 サポートの設定」を参照してください。

SASL PLAIN 認証での OAuth 2.0 のサポート

Kafka クライアントと Kafka ブローカー間の OAuth 2.0 認証に PLAIN メカニズムを設定できるようになりました。これまでは、認証されるメカニズムは OAUTHBEARER のみでした。

PLAIN は、すべての Kafka クライアントツール (kafkacat などの開発者ツールを含む) によって使用される簡易認証メカニズムです。AMQ Streams には、PLAIN を OAuth 2.0 認証と使用できるようにするサーバー側のコールバックが含まれています。これらの機能は **OAuth 2.0 over PLAIN**と呼ばれます。

注記

Red Hat は、可能な限りクライアントに OAUTHBEARER 認証を使用することを推奨します。OAUTHBEARER では、クライアントクレデンシャルは Kafka ブローカーと**共有されることがない**ため、PLAIN よりも高レベルのセキュリティーが提供されます。OAUTHBEARER をサポートしない Kafka クライアントの場合のみ、PLAIN の使用を検討してください。

提供される **OAuth 2.0 over PLAIN**コールバックと併用すると、Kafka クライアントは以下のいずれかの方法を使用して Kafka ブローカーで認証することができます。

- クライアント ID およびシークレット (OAuth 2.0 クライアントクレデンシャルメカニズムを使用)
- 設定時に手動で取得された有効期限の長いアクセストークン

PLAIN を使用するには、Kafka ブローカーの **oauth** リスナー設定で有効にする必要があります。新しい 3 つの設定オプションがサポートされるようになりました。

- **enableOauthBearer**
- **enablePlain**
- **tokenEndpointUri**

oauth リスナーの設定例

```
# ...
name: external
port: 9094
type: loadbalancer
tls: true
authentication:
  type: oauth
  # ...
  checkIssuer: false
  fallbackUserNameClaim: client_id
  fallbackUserNamePrefix: client-account-
  validTokenType: bearer
  userInfoEndpointUri: https://OAUTH-SERVER-ADDRESS/auth/realms/external/protocol/openid-connect/userinfo
  enableOauthBearer: false (1)
  enablePlain: true (2)
  tokenEndpointUri: https://OAUTH-SERVER-ADDRESS/auth/realms/external/protocol/openid-connect/token (3)
  #...
```

1. リスナーでの OAUTHBEARER 認証を無効にします。**true** またはオプションが指定されていない場合は、OAUTHBEARER 認証が有効になります。
2. リスナーで PLAIN 認証を有効にします。デフォルトは **false** です。

3 承認サーバーへの OAuth 2.0 トークンエンドポイント URL。**enablePlain** が **true** で、クライアント ID とシークレットが認証に使用される場合に設定する必要があります。

「OAuth 2.0 認証メカニズム」および「Kafka ブローカーの OAuth 2.0 サポートの設定」を参照してください。

2.17. KAFKA CONNECT のラック プロパティー

Kafka Connect で新しい **rack** プロパティーが利用できるようになりました。ラックアウェアネス (Rack awareness) は、異なるラック全体でレプリカを分散するために設定されます。Kafka Connect クラスターの **ラック** を設定すると、コンシューマーは最も近いレプリカからデータを取得できます。これは、Kafka クラスターが複数のデータセンターにまたがる場合に便利です。

topology キーはクラスターノードのラベルと一致する必要があります。

ラック 設定の例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
#...
spec:
  #...
  rack:
    topologyKey: topology.kubernetes.io/zone
```

「**KafkaConnectSpec** スキーマ参照」および「**KafkaConnectS2ISpec** スキーマ参照」を参照してください。

2.18. POD トポロジー分散制約

Pod トポロジー分散制約 は、以下の AMQ Streams カスタムリソースでサポートされるようになりました。

- **Kafka** には以下のものが含まれます。
  - ZooKeeper
  - Entitiy Operator
- **KafkaConnect**
- **KafkaConnectS2I**
- **KafkaBridge**
- **KafkaMirrorMaker2** および **KafkaMirrorMaker**

Pod トポロジー分散制約により、Kafka 関連の Pod をノード、ゾーン、リージョン、またはその他のユーザー定義のドメインに分散できます。これらは、Pod スケジューリングの既存の **アフィニティー** および **容認** プロパティーと共に使用できます。

制約は、関連するカスタムリソースの **template.pod.topologySpreadConstraints** プロパティーに指定されます。

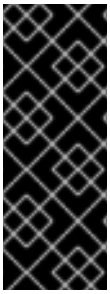
Kafka Connect の Pod トポロジー分散制約の例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
#...
spec:
  # ...
  template:
    pod:
      topologySpreadConstraints:
      - maxSkew: "1"
        whenUnsatisfiable: DoNotSchedule
        labelSelector:
          matchLabels:
            label1: value1
#...
```

参照:

- 『**AMQ Streams の使用**』の「アフィニティー、容認 (Toleration)、およびトポロジー分散制約の指定」。
- OpenShift Container Platform ドキュメントの「Pod トポロジー分散制約を使用した Pod 配置の制御」。

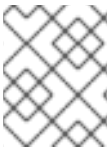
第3章 テクノロジープレビュー



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機能は、最新の技術をいち早く提供し、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。

3.1. 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、最適化プロポーザルを生成します。
- 最適化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 によるクラスターのリバランス」を参照してください。

3.1.1. テクノロジープレビューの改良

テクノロジープレビューである Cruise Control のクラスターのリバランスに、以下の改良が追加されました。

新しいゴール: ブローカーごとの最小トピックリーダー数

MinTopicLeadersPerBrokerGoal という名前の新しいゴールを使用できます。

定義されたトピックグループの各トピックに対し、アクティブなブローカーごとに最低でも一定数のリーダーレプリカがあるようにします。

MinTopicLeadersPerBrokerGoal はデフォルトのゴールで、ハードゴールとして事前設定されています。

「最適化ゴールの概要」を参照してください。

ロギングの改良: 動的ロギング設定および Log4j 2

Cruise Control が動的ロギング設定をサポートするようになりました。そのため、Cruise Control のロギングレベルを変更しても、Kafka クラスターまたは Cruise Control Pod へのローリングアップデートがトリガーされなくなりました。

Log4j 2 が Cruise Control のロギングに使用されるようになりました。

Cruise Control ロギングの既存の設定を Log4j から **Log4j 2** と互換性のある構文に更新する **必要** があります。ロギングは **Kafka** カスタムリソースで設定されます。

- **inline ロギング** の場合は、**cruisecontrol.root.logger** プロパティーを **rootLogger.level** プロパティーに置き換えます。
- **外部ロギング** の場合は、既存の設定を **log4j2.properties** という名前の新規設定ファイルに置き換えます。設定は、**Log4j 2** 互換構文を使用する **必要があり** ます。

「外部ロギング」および「Cruise Control の設定」を参照してください。

第4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。

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

AMQ Streams 1.7 では、1章*機能* の説明に従って、**KafkaConnect** リソースに **ビルド** 設定が導入されています。**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。

「Kafka Connect S2I の Kafka Connect への移行 」を参照してください。

4.2. メトリクスの設定

メトリクス設定は、Kafka コンポーネントの ConfigMap として指定されるようになりました。以前のバージョンでは、**spec.metrics** プロパティーが使用されていました。

設定を更新し、Prometheus メトリクスのエクスポートを有効にするには、**.spec.metrics** プロパティーの設定に一致する新しい ConfigMap を作成する必要があります。**.spec.metricsConfig** プロパティーは、1章*機能* の説明に従って ConfigMap を指定するために使用されます。

「AMQ Streams のアップグレード」を参照してください。

4.3. API バージョン

v1beta2 が導入され、カスタムリソースのスキーマが更新されるようになりました。古い API バージョンは非推奨になりました。

以下の AMQ Streams カスタムリソースでは、**v1alpha1** API バージョンは非推奨になりました。

- **Kafka**
- **KafkaConnect**
- **KafkaConnectS2I**
- **KafkaConnector**
- **KafkaMirrorMaker**
- **KafkaMirrorMaker2**
- **KafkaTopic**
- **KafkaUser**
- **KafkaBridge**
- **KafkaRebalance**

v1beta1 API バージョンは、以下の AMQ Streams カスタムリソースで非推奨となりました。

- **Kafka**
- **KafkaConnect**
- **KafkaConnectS2I**
- **KafkaMirrorMaker**
- **KafkaTopic**
- **KafkaUser**

重要

v1alpha1 および **v1beta1** バージョンは、次のマイナーリリースで削除されます。

「AMQ Streams カスタムリソースのアップグレード」を参照してください。

4.4. アノテーション

以下のアノテーションは非推奨となり、AMQ Streams 1.8.0 で削除されます。

表4.1 非推奨となったアノテーションとその代替

非推奨のアノテーション	代替のアノテーション
cluster.operator.strimzi.io/delete-claim	**strimzi.io/delete-claim** (Internal)
operator.strimzi.io/generation	**strimzi.io/generation**（内部）
operator.strimzi.io/delete-pod-and-pvc	**strimzi.io/delete-pod-and-pvc**
operator.strimzi.io/manual-rolling-update	**strimzi.io/manual-rolling-update**

第5章 修正された問題

AMQ Streams 1.7 で修正された問題を、以下の表に示します。Kafka 2.7.0 で修正された問題の詳細は、『Kafka 2.7.0 Release Notes』を参照してください。

課題番号	説明
ENTMQST-1561	OpenSSL タスクは、メインスレッドではなく、別のワーカーエグゼキューターで実行される必要がある。
ENTMQST-1607	アップグレード中にブローカーから **log.message.format.version** および **inter.broker.protocol.version** を確認します。
ENTMQST-1631	Topic Operator が KafkaTopic の名前を変更することがある。
ENTMQST-1676	Kafka のアップグレードおよびダウングレードを簡素化。
ENTMQST-1914	Java 11 言語レベルへ移行。
ENTMQST-2030	ACL の追加または削除に **bin/kafka-acls.sh** ユーティリティーを使用すると、操作は成功しますが、警告が生成されます。
ENTMQST-2085	Kafka および Zookeeper を同時に再起動すると、すべての KafkaTopic カスタムリソースが削除され、再作成される。
ENTMQST-2184	Kafka **min.insync.replicas** が 1 を超える場合、メトリクスレポーターはメトリクスを生成できない
ENTMQST-2188	MirrorMaker: ターゲットクラスターのコンシューマーグループへのオフセットの同期を有効にする。
ENTMQST-2269	kafka-configs.sh は非推奨である **--zookeeper** オプションですが、ユーザーの設定を一覧表示する代替機能を提供しません。
ENTMQST-2295	Topic Operator でのメトリクスの誤った調整を監視する。
ENTMQST-2311	ネットワークポリシーの設定を向上。
ENTMQST-2335	**tini** init で ConnectS2I デプロイメントを実行
ENTMQST-2386	JBOD ボリュームの追加または削除が機能しない。

課題番号	説明
ENTMQST-2472	Prometheus Operator バンドルファイルの namespace を置き換えると無効な YAML が生成される。
ENTMQST-2480	Grafana ダッシュボードで CPU メトリクスの使用に一貫性がない。
ENTMQST-2483	KafkaConnect Build: Kafka Connect カスタムリソースのコネクタープラグインの宣言的な管理。
ENTMQST-2525	コネクター/タスクの再起動操作を実行するアノテーションを追加。
ENTMQST-2529	Kafka Exporter ダッシュボードは namespace およびクラスター名を自動選択しない。
ENTMQST-2547	メトリクス ConfigMap が使用されていると、ネットワークポリシーが適切に設定されない。
ENTMQST-2548	ユーザー証明書が CA で個別に期限切れになる場合に更新が全くトリガーされない。
ENTMQST-2550	Cruise Control Pod がロールまたは削除されると、Cruise Control Grafana ダッシュボードの表示に影響する。
ENTMQST-2595	レプリカまたはパーティションの数が減少すると、Topic Operator がトピックの作成に失敗する。
ENTMQST-2625	JMX 設定の問題によって **port already in use** エラーが発生する。
ENTMQST-2636	'resource' パーミッションで 'keycloak' 承認を使用する場合に OAuth NullPointerException が発生。
ENTMQST-2643	Kafka リソースのステータスに ISO-8601 タイムスタンプ標準を使用する。

表5.1 CVE (Common Vulnerabilities and Exposures) の修正

課題番号	タイトル	説明

課題番号	タイトル	説明
ENTMQST-2334	CVE-2020-25649 jackson-databind: FasterXML DOMDeserializer insecure entity expansion is vulnerable to XML external entity (XXE) [amq-st-1]	FasterXML Jackson Databind で、エンティティー拡張のセキュリティーが適切に保護されていないという不具合が発見されました。この不具合により、XML 外部エンティティー (XXE) 攻撃に対して脆弱になります。この脆弱性では、データの整合性が最も懸念されます。

第6章 既知の問題

ここでは、AMQ Streams 1.7 の既知の問題について説明します。

6.1. CLUSTER OPERATOR の IPV6 クラスターへのデプロイに関する問題

説明および回避策

ENTMQST-2754

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

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。以下はその例です。

   ```
   apiVersion: operators.coreos.com/v1alpha1
   kind: Subscription
   metadata:
     name: amq-streams
     namespace: OPERATOR-NAMESPACE
   spec:
     channel: amq-streams-1.7.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。以下はその例です。

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: amq-streams
  namespace: OPERATOR-NAMESPACE
spec:
  channel: amq-streams-1.7.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
```

6.2. KAFKA BRIDGE サービスの 3SCALE 検出の問題

説明および回避策

ENTMQST-2777

「Kafka Bridge 向けの 3scale のデプロイ」で説明されているように、Red Hat 3scale は Kafka Bridge サービスを 検出できません。

回避策

AMQ Streams クラスターで以下の手順を実行して、サービス検出を有効にします。

1. **KafkaBridge** カスタムリソースの **spec** プロパティーを編集します。**discovery.3scale.net: true** テンプレートプロパティーを既存の設定に追加します。

   Kafka Bridge のテンプレート設定例

   ```
   apiVersion: kafka.strimzi.io/v1beta1
   kind: KafkaBridge
   metadata:
     name: KAFKA-BRIDGE-NAME
   spec:
     replicas: 1
     bootstrapServers: my-cluster-kafka-bootstrap:9092
     http:
       port: 8080
     template:
       apiService:
         metadata:
           labels:
             discovery.3scale.net: true (1)
   # ...
   ```

   (1) 3scale が Kafka Bridge サービスを検出できるようにします。

2. カスタムリソースを作成または更新します。

   ```
   kubectl apply -f KAFKA-BRIDGE-CONFIG-FILE
   ```

3. 「Kafka Bridge 向けの 3scale のデプロイ」の手順 6 で説明されているように、3scale サービス検出を 続行します。

第7章 サポート対象のインテグレーション製品

AMQ Streams 1.7 は、以下の Red Hat 製品との統合をサポートします。

- OAuth 2.0 認証および OAuth 2.0 承認用の **Red Hat Single Sign-On 7.4 以上**。
- Kafka Bridge をセキュアにし、追加の API 管理機能を提供する **Red Hat 3scale API Management 2.6 以上**。
- データベースを監視し、イベントストリームを作成する **Red Hat Debezium 1.4 以上**。
- 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の **Service Registry 2020-Q4 以上**。

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 1.7 のドキュメントを参照してください。

第8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： 2022-09-08 16:16:46 +1000