Red Hat AMQ Streams 2.0

AMQ Streams 2.0 on OpenShift のリリースノート

AMQ Streams on OpenShift Container Platform の本リリースにおける新機能と変更内容のハイライト

Last Updated: 2022-06-14

Red Hat AMQ Streams 2.0 AMQ Streams 2.0 on OpenShift のリリースノート

AMQ Streams on OpenShift Container Platform の本リリースにおける新機能と変更内容のハイライト

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

本リリースノートでは、AMQ Streams 2.0 リリースで導入された新機能、改良された機能、および修正についてまとめています。

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 特長

AMQ Streams バージョン 2.0 は Strimzi 0.26.x をベースとしています。

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。

AMQ Streams 2.0.1 パッチリリースが利用できるようになりました。

AMQ Streams の製品イメージがバージョン 2.0.1 にアップグレードされました。

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 2.0 は OpenShift Container Platform 4.6 〜 4.9 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 Streams for Apache Kafka Supported Configurations」を参照してください。

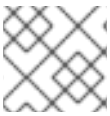
1.2. KAFKA 3.0.0 のサポート

AMQ Streams は Apache Kafka バージョン 3.0.0 に対応するようになりました。

AMQ Streams は Kafka 3.0.0 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 3.0.0 にアップグレードする前に、Cluster Operator を AMQ Streams バージョン 2.0 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 2.8.0 および Kafka 3.0.0 のリリースノートを参照してください。



注記

Kafka 2.8.x は、AMQ Streams 2.0 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、Red Hat Streams for Apache Kafka Component Details を参照してください。

Kafka 3.0.0 には ZooKeeper バージョン 3.6.3 が必要です。そのため、AMQ Streams 1.8 から AMQ Streams 2.0 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. 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 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

重要

AMQ Streams バージョン 2.0 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。

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

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 **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。

完全な手順については、Upgrading AMQ Streams を参照してください。

1.4. SCALA のアップグレード

AMQ Streams では、Scala2.12 の代わりに Scala2.13 が使用されるようになりました。

Maven ビルドで Kafka Streams の依存関係を使用している場合は、Scala2.13 および Kafka バージョン 3.0.0 を指定するように更新する必要があります。依存関係が更新されない場合は、ビルドされません。

Kafka Streams の依存関係

```
<dependency>
  <groupId>org.apache.kafka</groupId>
  <artifactId>kafka-streams-scala_2.13</artifactId>
```

```
    <version>3.0.0</version>
</dependency>
```

Kafka Streams Scala 依存関係 を参照してください。

1.5. POWERPC アーキテクチャーのサポート

AMQ Streams 2.0 は、PowerPC **ppc64le** アーキテクチャーでの実行が有効になっています。

IBM Power システムのサポートは、OpenShift Container Platform 4.9 の Kafka 3.0.0 で実行されている AMQ Streams に適用されます。AMQ Streams 1.8 以前のバージョンの Kafka バージョンには ppc64 バイナリーが **含まれていません**。

1.5.1. IBM Power システムの要件

- OpenShift Container Platform 4.9
- Kafka 3.0.0

1.5.2. IBM Power Systems でサポート対象外

- Kafka 2.8.0 以前
- これは ppc64le の最初のリリースであるため、AMQ Streams のアップグレードおよびダウングレード
- 非接続 OpenShift Container Platform 環境の AMQ Streams
- AMQ Streams OPA の統合

1.6. POD エビクションの AMQ STREAMS DRAIN CLEANER

AMQ Streams 2.0 では、AMQ Streams Drain Cleaner が導入されました。AMQ Streams Drain Cleaner を Cluster Operator と組み合わせて使用し、ドレイン (解放) されているノードから Kafka ブローカーまたは ZooKeeper Pod を移動します。

Cluster Operator および Kafka クラスターが実行中の OpenShift クラスターに、AMQ Streams Drain Cleaner をデプロイします。AMQ Streams Drain Cleaner をデプロイすることで、Cluster Operator を使用して OpenShift ではなく Kafka Pod を移動できます。

AMQ Streams Drain Cleaner を実行すると、Pod にローリングアップデート Pod アノテーションが付けられます。Cluster Operator はアノテーションに基づいてローリングアップデートを実行します。Cluster Operator は、トピックの複製の数が最低数未満にならないようにします。

Evicting pods with AMQ Streams Drain Cleaner 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。

1.8. 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。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 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 3.0.0 で改良された機能

Kafka 3.0.0 に導入された改良機能の概要はKafka 3.0.0 Release Notes を参照してください。

2.2. リスナーの自動ネットワークポリシーの無効化

これで、リスナーの **NetworkPolicy** リソースの自動作成を無効にし、代わりに独自のカスタムネットワークポリシーを使用できるようになりました。

デフォルトでは、AMQ Streams では、Kafka ブローカーで有効になっているリスナーごとに **NetworkPolicy** リソースが自動的に作成されます。特定のリスナーの場合、ネットワークポリシーはすべての namespace にまたがるアプリケーションが接続できるようにします。この動作は、**networkPolicyPeers** を使用して制限できます。

自動作成された **NetworkPolicies** を無効にするには、Cluster Operator 設定で **STRIMZI_NETWORK_POLICY_GENERATION** 環境変数を **false** に設定します。

Cluster Operator configuration および Network policies を参照してください。

2.3. SCRAM ユーザーが KAFKA ADMIN API を使用して管理

User Operator は、ZooKeeper の代わりに Kafka Admin API を使用して SCRAM-SHA-512 ユーザーの認証情報を管理するようになりました。Operator は SCRAM-SHA-512 認証情報管理の Kafka に直接接続します。

この変更は、Kafka の ZooKeeper への依存関係を削除するために Apache Kafka プロジェクトが進めている作業の一部です。

ZooKeeper 関連の一部の設定は非推奨となり、削除されました。

Deprecated features および Using the User Operator を参照してください。

2.4. コネクタープラグインでの MAVEN コーディネートの使用

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 **build** 設定で、Kafka Connect をデプロイできます。コネクタープラグインのアーティファクトを Maven コーディネートとして指定できるようになりました。

AMQ Streams では、以下のタイプのアーティファクトがサポートされます。

- 直接ダウンロードして使用する JAR ファイル
- ダウンロードおよび解凍された TGZ アーカイブ
- Maven コーディネートを使用する Maven アーティファクト
- 直接ダウンロードおよび使用されるその他のアーティファクト

Maven コーディネートは、プラグインアーティファクトおよび依存関係を特定し、Maven リポジトリーから検索および取得できるようにします。

Maven アーティファクトの例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
  name: my-connect-cluster
spec:
  #...
  build:
    output:
      #...
    plugins:
      - name: my-plugin
        artifacts:
          - type: maven (1)
          - repository: https://mvnrepository.com (2)
          - group: org.apache.camel.kafkaconnector (3)
          - artifact: camel-kafka-connector (4)
          - version: 0.11.0 (5)
  #...
```

(1) (必須) アーティファクトのタイプ。

(2) (任意) アーティファクトのダウンロード元となる Maven リポジトリー。リポジトリーを指定しないと、デフォルトで Maven Central リポジトリー が使用されます。

(3) (必須) Maven グループ ID。

(4) (必須) Maven アーティファクトタイプ。

(5) (必須) Maven バージョン番号。

Build schema reference を参照してください。

2.5. 外部設定データの環境変数設定プロバイダー

環境変数の設定プロバイダープラグインを使用して、環境変数から設定データを読み込みます。

プロバイダーを使用して、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、プロバイダーを使用して、Kafka Connect コネクター設定のクレデンシャルを提供します。

環境変数の値は、シークレットまたは設定マップからマッピングできます。環境変数設定プロバイダーを使用して、たとえば、OpenShift シークレットからマップされた環境変数から証明書または JAAS 設定を読み込むことができます。

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

2.6. 安全でないサーバーからコネクタープラグインアーティファクトをダウンロードできるようにする

安全でないサーバーから、コネクタープラグインアーティファクトをダウンロードできるようになりました。これは JAR、TGZ、およびコンテナーイメージにダウンロードされ、追加されたその他のファイ

ルに適用されます。これは、コネクター **build** 設定の **insecure** プロパティーを使用して設定できます。ダウンロードするプラグインアーティファクトを指定する場合は、プロパティーを **true** に設定します。これにより、すべての TLS 検証が無効になります。サーバーが安全ではない場合、アーティファクトは引き続きダウンロードされます。

安全ではないダウンロードを許可する TGZ プラグインアーティファクトの例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
  name: my-connect-cluster
spec:
  #...
  build:
    output:
      #...
    plugins:
      - name: my-plugin
        artifacts:
          - type: tgz
            url: https://my-domain.tld/my-connector-archive.tgz
            sha512sum: 158...jg10
            insecure: true
  #...
```

Build schema reference を参照してください。

2.7. KAFKA CONNECT ベースイメージをプルするためのシークレットの指定

OpenShift 上の Kafka Connect ビルドのベースイメージをプルするために使用される、コンテナーレジストリーシークレットを指定できるようになりました。シークレットは **template** のカスタマイズとして設定されています。Kafka Connect **spec** の **buildConfig** テンプレートを使用して、**pullSecret** プロパティーを使用したシークレットの指定でができます。シークレットには、ベースイメージをプルするための認証情報が含まれます。

プルシークレットのテンプレートのカスタマイズ例

```
# ...
template:
  buildConfig:
    metadata:
      labels:
        label1: value1
        label2: value2
      annotations:
        annotation1: value1
        annotation2: value2
    pullSecret: "<secret_credentials>"
# ...
```

2.8. /TMP ディレクトリーのボリュームを指定します。

/tmp (一時的な **emptyDir** ボリューム) の合計ローカルストレージサイズを設定できるようになりました。この仕様は、現在 Pod で実行されている単一コンテナーで機能します。デフォルトのストレージサイズは **1Mi** です。サイズは **template** のカスタマイズとして設定されます。リソースの **spec** で **pod** テンプレートを使用し、**tmpDirSizeLimit** プロパティーを使用してボリュームサイズを指定できます。

/tmp のローカルストレージサイズを指定するテンプレートのカスタマイズ例

```
# ...
template:
  pod:
    tmpDirSizeLimit: "2Mi"
# ...
```

2.9. メータリング

クラスター管理者は、OpenShift で利用可能なメータリングツールを使用して、クラスターで実行されている内容を分析できます。このツールを使用して、レポートを生成し、インストールされた AMQ Streams コンポーネントを分析できるようになりました。Prometheus をデフォルトのデータソースとして使用すると、Pod、namespace、およびその他ほとんどの Kubernetes リソースのレポートを生成できます。OpenShift の Metering Operator を使用すると、インストールされた AMQ Streams コンポーネントを分析し、Red Hat サブスクリプションに準拠しているかどうかを判断することができます。

AMQ Streams でメータリングを使用するには、まず OpenShift Container Platform に Metering operator をインストールし、設定する必要があります。

Using Metering on AMQ Streams を参照してください。

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。

3.1. 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
```

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。

3.2. 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。

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、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。

3.2.1. テクノロジープレビューの改良

Cruise Control REST API のセキュリティー

Cruise Control REST API は HTTP Basic 認証および SSL でセキュリティー保護され、Kafka ブローカーの停止などの破壊的な Cruise Control 操作からクラスターを保護します。

以下の設定はデフォルトで有効になっています。

- **webserver.security.enable: true**
- **webserver.ssl.enable: true**

組み込みの HTTP Basic 認証または SSL 設定を無効にすることはできますが、**これは推奨されません。** セキュリティーを無効にする設定オプションが **Kafka** リソースの **spec.cruiseControl.config** に追加されました。

注記

Cruise Control が AMQ Streams クラスターにデプロイされると、REST API ではなくカスタムリソースを使用して操作を実行します。

Cruise Control configuration を参照してください。

Cruise Control の異常検出

Cruise Control の異常検出が AMQ Streams で使用できるようになりました。異常検出を使用して Cruise Control のメトリクスを監視することができます。Cruise Control の異常通知機能は、ブローカーの障害や最適化ゴールの生成をブロックするその他の条件に関するアラートと、実行されたすべてのアクションを提供します。

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。

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. JAVA 8

Java 8 のサポートは Kafka 3.0.0 および AMQ Streams 2.0 で非推奨となりました。Java 8 は、今後、クライアントを含むすべての AMQ Streams コンポーネントではサポートされません。

AMQ Streams は Java 11 をサポートします。新規アプリケーションの開発時に Java 11 を使用します。現在 Java 8 を使用しているアプリケーションを Java 11 に移行する計画です。

4.2. USER OPERATOR の ZOOKEEPER 設定

スタンドアロンの User Operator 設定から 2 つの環境変数が **削除** されました。

- **STRIMZI_ZOOKEEPER_CONNECT**
- **STRIMZI_ZOOKEEPER_SESSION_TIMEOUT_MS**

以前のバージョンでは、これらの環境変数は、スタンドアロンの User Operator のみをデプロイするのに使用される **install/user-operator/05-Deployment-strimzi-user-operator.yaml** で設定されていました。

User Operator の **zookeeperSessionTimeoutSeconds** プロパティーは不要になり、**非推奨** になりました。このプロパティーは **Kafka** リソースの **userOperator** プロパティーに設定されました。

4.3. 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 で削除されます。MirrorMaker 2.0 のみが利用可能なバージョンになります。MirrorMaker 2.0 は Kafka Connect フレームワークをベースとし、コネクターによってクラスター間のデータ転送が管理されます。

そのため、Kafka MirrorMaker 1 のデプロイに使用される AMQ Streams **KafkaMirrorMaker** カスタムリソースが非推奨になりました。Kafka 4.0.0 が導入されると、**KafkaMirrorMaker** リソースは AMQ Streams から削除されます。

MirrorMaker 1 (AMQ ストリームドキュメントでちょうど **MirrorMaker** と呼ばれる) を使用している場合は、**IdentityReplicationPolicy** と **KafkaMirrorMaker2** のカスタムリソースを使用します。MirrorMaker 2.0 では、ターゲットクラスターにレプリケートされたトピックの名前が変更されます。**IdentityReplicationPolicy** 設定は、名前の自動変更を上書きします。これを使用して、MirrorMaker 1 と同じアクティブ/パッシブの一方向レプリケーションを作成します。

Kafka MirrorMaker 2.0 クラスターの設定 を参照してください。

4.4. サンプルから OPENSHIFT テンプレートが削除される

AMQ Streams サンプルファイルで提供される OpenShift テンプレートが削除され、サポート対象外になりました。テンプレート設定ファイルは、OpenShift コンソールから Kafka コンポーネントをデプロイするために提供されていました。たとえば、一時ストレージまたは永続ストレージで Kafka をデプロ

イするためにテンプレートが含まれていました。テンプレートは、OpenShift 3.11 を使用する場合にいくつかの利点を提供していました。OpenShift 4.x 以降、Cluster Operator を使用して OpenShift コンソールで Kafka コンポーネントをインストールし、管理することがより容易になります。

4.5. SOURCE-TO-IMAGE (S2I) 対応の KAFKA CONNECT のサポートの削除

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 のサポートが削除されました。

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。

第5章 修正された問題

以下のセクションには、AMQ Streams 2.0.x で修正された問題が記載されています。Red Hat は、最新のパッチリリースにアップグレードすることを推奨します。

Kafka 3.0.0 で修正された問題の詳細は、Kafka 3.0.0 Release Notes を参照してください。

5.1. AMQ STREAMS 2.0.1 で修正された問題

AMQ Streams 2.0.1 パッチリリースが利用できるようになりました。

AMQ Streams の製品イメージがバージョン 2.0.1 にアップグレードされました。

AMQ Streams 2.0.1 で解決された問題の詳細は、AMQ Streams 2. 0.x Resolved Issues を参照してください。

Log4j の脆弱性

AMQ Streams には log4j 1.2.17 が含まれています。このリリースでは、log4j の脆弱性が数多く修正されています。

本リリースで対応する脆弱性の詳細は、以下の CVE の説明を参照してください。

- CVE-2022-23307
- CVE-2022-23305
- CVE-2022-23302
- CVE-2021-4104
- CVE-2020-9488
- CVE-2019-17571
- CVE-2017-5645

5.2. FIXED ISSUES FOR AMQ STREAMS 2.0.0

Log4j2 の脆弱性

AMQ Streams には log4j2 2.17.1 が含まれています。本リリースでは、log4j2 の脆弱性が多数修正されています。

本リリースで対応する脆弱性の詳細は、以下の CVE の説明を参照してください。

- CVE-2021-45046
- CVE-2021-45105
- CVE-2021-44832
- CVE-2021-44228

表5.1 修正された問題

課題番号	説明
ENTMQST-3022	明示的に指定された監視 namespace に対してメトリクスが適切に機能しません。
ENTMQST-3053	セレクターラベルが変更されたために Operator に Kafka リソースが表示されなくなった場合、**strimzi_resource_state** は 0 に更新されません。
ENTMQST-3207	内部リスナーの場合でも、アドバタイズされたホスト名を証明書 SAN に追加します。
ENTMQST-3250	ログレベルの変更が Kafka Exporter で動作しません。
ENTMQST-3274	CRD の正規表現検証の修正
ENTMQST-3296	Kafka Exporter のすべての証明書を読み込みます。
ENTMQST-3297	**ZookeeperScaler** および **DefaultAdminClientProvider**で、クラスター CA からのすべての公開鍵を使用します。
ENTMQST-3318	TLS ユーザーとクォータの調整を修正します。
ENTMQST-3601	force-renew の実行後に内部コンポーネントの証明書が更新されない

表5.2 CVE (Common Vulnerabilities and Exposures) の修正

課題番号	説明
ENTMQST-3146	**CVE-2021-34429**: jetty-server: jetty: で特別に作成された URI により、セキュリティーの制約を回避できます。
ENTMQST-3307	**CVE-2021-38153** Kafka: Apache Kafka Connect およびクライアントに対するタイミング攻撃の脆弱性
ENTMQST-3308	**CVE-2021-38153** kafka-clients: Kafka: Apache Kafka Connect およびクライアントの攻撃脆弱性の調整
ENTMQST-3316	**CVE-2021-37136** netty-codec: Bzip2Decoder が圧縮したデータのサイズ制限を設定できない
ENTMQST-3317	**CVE-2021-37137** netty-codec: SnappyFrameDecoder がチャンクの長さを制限せず、スキップ可能なチャンクを不要な方法でバッファリングする可能性があります
ENTMQST-3428	**CVE-2021-37136** netty-codec: Bzip2Decoder が圧縮したデータのサイズ制限を設定できない - Drain Cleaner

課題番号	説明
ENTMQST-3532	**CVE-2021-44228** log4j-core: 攻撃者が制御する文字列の値がログに含まれる場合の Log4j 2.x でのリモートコード実行
ENTMQST-3555	**CVE-2021-45046** log4j-core: スレッドコンテキストメッセージパターンとコンテキストルックアップパターンを使用した log4j2.x の DoS
ENTMQST-3587	**CVE-2021-45105** log4j-core: Thread Context Map (MDC) 入力データを使用した log4j 2.x の DoS には、再帰ルックアップとコンテキストルックアップパターンが含まれています
ENTMQST-3602	**CVE-2021-44832** log4j-core: JDBC Appender を介したリモートコードの実行

第6章 既知の問題

ここでは、AMQ Streams 2.0 の既知の問題について説明します。

6.1. 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6.1 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。

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。

6.2. 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>
```

6.3. CRUISE CONTROL の CPU 使用率の見積もり

AMQ Streams の Cruise Control には、CPU 使用率予測の計算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、ノードの論理プロセッサーの数が、そのノード上の Kafka ブローカー Pod の CPU 制限と等しくない場合に発生します。この問題は、Pod の負荷が大きい場合に、クラスターのリバランスを阻止することができます。

回避策

この問題は、Cruise Control 設定に指定されたハードゴールおよびデフォルトゴールから CPU ゴールを除外することで、回避することができます。

CPU ゴールのない 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、**OptimizationFailureException** due to insufficient CPU capacity を参照してください。

第7章 サポートされる統合製品

AMQ Streams 2.0 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 2.0 のドキュメントを参照してください。

関連情報

- 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8章 重要なリンク

- Red Hat Streams for Apache Kafka でサポートされる構成
- Red Hat Streams for Apache Kafka コンポーネントの詳細

改訂日時： 2022-06-14 00:30:33 +1000