# Red Hat AMQ Streams 2.1

## AMQ Streams 2.1 on OpenShift のリリースノート

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

Last Updated: 2023-02-27

# Red Hat AMQ Streams 2.1 AMQ Streams 2.1 on OpenShift のリリースノート

OpenShift Container Platform 上の本 AMQ Streams 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 2.1 リリースで導入された新機能、強化された機能、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 機能

AMQ Streams 2.1 には、本セクションで説明する機能が導入されています。

AMQ Streams バージョン 2.1 は Strimzi 0.28.x をベースとしています。

**注記**

本リリースで解決された機能強化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 2.1 は OpenShift Container Platform 4.6 〜 4.10 でサポートされます。

サポートされているプラットフォームバージョンの詳細については、AMQ Streams Supported Configurations を参照してください。

## 1.2. KAFKA 3.1.0 のサポート

AMQ Streams は Apache Kafka バージョン 3.1.0 に対応するようになりました。

AMQ Streams は Kafka 3.1.0 を使用します。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。

ブローカーおよびクライアントアプリケーションを Kafka 3.1.0 にアップグレードする前に、Cluster Operator を AMQ Streams バージョン 2.1 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 3.0.0 および Kafka 3.1.0 のリリースノートを参照してください。

**注記**

Kafka 3.0.x は、AMQ Streams 2.1 にアップグレードする場合に限りサポートされます。

サポート対象バージョンの詳細については、AMQ Streams Component Details を参照してください。

Kafka 3.1.0 には、Kafka 3.0.0 と同じバージョンの ZooKeeper バージョン 3.6.3 が必要です。そのため、AMQ Streams 2.0 から AMQ Streams 2.1 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

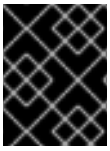
すべてのカスタムリソースの **v1beta2** API バージョンが AMQ Streams 1.7 で導入されました。AMQ Streams 1.8 では、**KafkaTopic** および **KafkaUser** を除くすべての AMQ Streams カスタムリソースから **v1alpha1** および **v1beta1** API バージョンが削除されました。

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. カスタムリソースの **v1beta2** への変換
3. AMQ Streams 1.8 へのアップグレード



**重要**

AMQ Streams バージョン 2.1 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 ソフトウェアの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. IBM Z および LINUXONE アーキテクチャーのサポート

AMQ Streams 2.1 は IBM Z および LinuxONE s390x アーキテクチャーで稼働するように有効になっています。

IBM Z および LinuxONE のサポートは、OpenShift Container Platform 4.10 の Kafka 3.1.0 で実行されている AMQ Streams に適用されます。AMQ Streams 2.0 以前のバージョンの Kafka バージョンには s390x バイナリーが含まれていません。

### 1.4.1. IBM Z および LinuxONE の要件

- OpenShift Container Platform 4.10
- Kafka 3.1.0

### 1.4.2. IBM Z および LinuxONE でのサポート対象外

- Kafka 3.0.0 以前
- AMQ Streams のアップグレードおよびダウングレード (これが s390x の最初のリリースであるため)
- 非接続 OpenShift Container Platform 環境の AMQ Streams
- AMQ Streams OPA の統合

## 1.5. IBM POWER アーキテクチャーのサポート

AMQ Streams 2.1 は、BM Power **ppc64le** アーキテクチャーで実行できます。

IBM Power のサポートは、OpenShift Container Platform 4.9 以降において Kafka 3.0.0 以降で実行している AMQ ストリームに適用されます。AMQ Streams 1.8 以前のバージョンの Kafka バージョンに ppc64le バイナリーは **含まれていません**。

### 1.5.1. IBM Power の要件

- OpenShift Container Platform 4.9 以降
- Kafka 3.0.0 以降

### 1.5.2. IBM Power でサポート対象外

- Kafka 2.8.0 以前
- 非接続 OpenShift Container Platform 環境の AMQ Streams

## 1.6. カスタム CA 証明書の更新

Cluster Operator は、ユーザーが提供したカスタム CA 証明書を検出できるようになりました。カスタム証明書を更新すると、Cluster Operator は、Zoo Keeper、Kafka、およびその他のコンポーネントのローリング更新を実行して、新しい CA 証明書を信頼します。

独自の証明書を使用している場合、Cluster Operator は自動的に更新されません。代わりに、既存の **Secret** を編集して、新しい CA 証明書を追加し、証明書生成アノテーション値を更新する必要があります。アノテーションは、Cluster Operator が更新プロセスで最新の証明書を使用するように、より高い増分値に設定されます。

**新しい CA 証明書で更新されるシークレット設定の例**

```
apiVersion: v1
kind: Secret
data:
  ca.crt: GCa6LS3RTHeKFiFDGBOUDYFAZ0F... (1)
metadata:
  annotations:
    strimzi.io/ca-cert-generation: "1" (2)
  labels:
    strimzi.io/cluster: my-cluster
    strimzi.io/kind: Kafka
```

```
  name: my-cluster-cluster-ca-cert
  #...
type: Opaque
```

1. Base64 でエンコードされた CA 証明書
2. CA 証明書生成アノテーションの値

独自の CA 証明書の更新 を参照してください。

## 1.7. DEBEZIUM FOR CHANGE DATA CAPTURE の統合

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 での Service Registry の使用について、詳しくは Service Registry documentation を参照してください。

# 第2章 機能拡張

AMQ Streams 2.1 では、多くの機能拡張が追加されました。

## 2.1. KAFKA 3.1.0 での機能拡張

Kafka 3.1.0 に導入された機能拡張の概要は、Kafka 3.1.0 Release Notes を参照してください。

## 2.2. FIPS 対応クラスターでの AMQ STREAMS の実行

現在、FIPS 準拠の設定ではありませんが、FIPS 対応のクラスターで AMQ ストリームを実行できるようになりました。

AMQ Streams コンテナーイメージで使用される OpenJDK は、FIPS 対応クラスターで自動的に FIPS モードに切り替わります。これにより、AMQ Streams がクラスターで実行されなくなります。

FIPS 対応クラスターで AMQ Streams を実行するには、Cluster Operator のデプロイメント設定で **FIPS_MODE** 環境変数を無効に設定し、OpenJDK FIPS モードを無効にします。AMQ Streams のデプロイは FIPS に準拠していませんが、AMQ Streams Operator とそのすべてのオペランドは、FIPS 対応の Kubernetes クラスターで実行できます。

### Cluster Operator の FIPS 設定例

```
apiVersion: apps/v1
kind: Deployment
spec:
  # ...
  template:
    spec:
      serviceAccountName: strimzi-cluster-operator
      containers:
        # ...
        env:
        # ...
        - name: "FIPS_MODE"
          value: "disabled" (1)
  # ...
```

(1) FIPS モードを無効にします。

Cluster Operator での FIPS モードの設定 を参照してください。

## 2.3. CRUISE CONTROL のブローカー内ディスクバランシング

**注記**

Cruise Control は引き続きテクノロジープレビューとなります。

Kafka デプロイメントで、同じブローカーにディスクが複数割り当てられた JBOD ストレージを使用している場合には、Cruise Control はディスク間でパーティションを分散できます。

**rebalanceDisk** 設定オプションを使用します。ブローカー内のディスク分散を実行するには、**KafkaRebalance.spec** で **rebalanceDisk** を **true** に設定します。

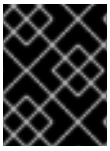
リバランスパフォーマンスチューニング を参照してください。

## 2.4. フィーチャーゲートは BETA 成熟段階に

フィーチャーゲートの **ControlPlaneListener** および **ServiceAccountPatching** は beta 成熟段階に移行します。つまり、どちらもデフォルトで有効になっています。

成熟段階が beta レベルのフィーチャーゲートは十分にテストされており、それらの機能は変更されない可能性が高くなります。

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



**重要**

AMQ Streams 1.7 以前のバージョンのアップグレードまたはダウングレード時に、**ControlPlaneListener** フィーチャーゲートを無効にする必要があります。

## 2.5. ロードバランサーリスナーのブートストラップサービス

新しいリスナー設定プロパティーを使用すると、**loadBalancer** タイプのリスナーのブートストラップサービスを作成するかどうかを制御できます。**<cluster_name>-kafka-external-bootstrap** ブートストラップサービスは、Kafka クラスターに対してデフォルトで作成されます。リスナー設定で **createBootstrapService** プロパティーを false に設定することにより、ロードバランサーのサービスを作成しないことを選択できます。

**ブートストラップサービスを作成しない loadbalancer 外部リスナーの設定例**

```
listeners:
  #...
  - name: external
    port: 9094
    type: loadbalancer
    tls: true
    authentication:
      type: tls
    configuration:
      createBootstrapService: false
      # ...
# ...
```

GenericKafkaListenerConfiguration スキーマプロパティー を参照してください。

## 2.6. OAUTH 設定オプション

新しい OAuth 設定プロパティーが OAuth 認証設定に導入されました。

タイムアウトとグループ情報の抽出に関連するプロパティー。

**Timout プロパティー**

- **connectTimeoutSeconds** は、タイムアウト前に認証サーバーに接続する最大時間を秒単位で指定します。
- **readTimeoutSeconds** は、タイムアウトの前に認証サーバーから読み取る最大時間を秒単位で指定します。

デフォルトは両方とも 60 秒です。

#### Groups プロパティー

- **groupsClaim** は、JWT トークンまたはイントロスペクションエンドポイント応答からグループ情報を抽出するための Json Path クエリーを指定します。デフォルトでは設定されません。
- **groupsClaimDelimiter** は、単一の区切り文字列として返されるときにグループ情報を解析するための区切り文字を指定します。デフォルト値は ,(コンマ) です。

### Kafka ブローカーリスナーの OAuth 設定例

```
#...
- name: external
  port: 9094
  type: loadbalancer
  tls: true
  authentication:
    type: oauth
    # ...
    connectTimeoutSeconds: 60
    readTimeoutSeconds: 60
    groupsClaim: "$.groups"
    groupsClaimDelimiter: ","
```

KafkaListenerAuthenticationOAuth スキーマリファレンス および KafkaClientAuthenticationOAuth スキーマプロパティー 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。

## 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

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。

# 第4章 非推奨の機能

以下の機能は、これまでの AMQ Streams リリースではサポート対象でしたが、このリリースで非推奨となりました。

## 4.1. JAVA 8

Java 8 のサポートは、Kafka 3.0.0 および AMQ Streams 2.0 で非推奨になりました。Java 8 は、将来、クライアントを含むすべての AMQ Streams コンポーネントでサポート対象外となります。

AMQ Streams は Java 11 をサポートします。新しいアプリケーションを開発する場合は、Java 11 を使用してください。また、現在 Java 8 を使用しているアプリケーションの Java 11 への移行も計画してください。

## 4.2. 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 Streams ドキュメントで **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。

## 4.3. ID 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 MirrorMaker 2.0 で使用され、リモートトピックの自動名前変更をオーバーライドします。その名前の前にソースクラスターの名前を追加する代わりに、トピックが元の名前を保持します。このオプションの設定は、active/passive バックアップおよびデータ移行に役立ちます。

現在、AMQ Streams Identity Replication Policy class (**io.strimzi.kafka.connect.mirror.IdentityReplicationPolicy**) は非推奨であり、将来削除される予定です。Kafka 独自の ID レプリケーションポリシー (**class org.apache.kafka.connect.mirror.IdentityReplicationPolicy**) に更新できます。

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

## 4.4. LISTENERSTATUS タイプのプロパティー

**ListenerStatus** の **type** プロパティーは非推奨になり、将来削除される予定です。**ListenerStatus** は、内部および外部リスナーのアドレスを指定するために使用されます。**type** を使用する代わりに、アドレスが **name** で指定されるようになりました。

ListenerStatus スキーマ参照 を参照してください。

## 4.5. CRUISE CONTROL の容量設定

**disk** および **cpuUtilization** 容量の設定プロパティーは非推奨になり、無視され、将来削除される予定です。プロパティーは、リソースベースの最適化ゴールの破損を判断するための最適化プロポーザルの容量制限の設定に使用されました。ディスクと CPU の容量制限は、AMQ Streams によって自動的に生成されるようになりました。

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

# 第5章 修正された問題

AMQ Streams 2.1 で修正された問題を、以下の表に示します。Kafka 3.1.0 で修正された問題の詳細は、Kafka 3.1.0 Release Notes を参照してください。

| 課題番号 | 説明 |
|---|---|
| ENTMQST-3595 | Cluster Operator に Kafka ブリッジに渡す Java オプションがない |
| ENTMQST-3835 | **tasks.max** が設定されている場合、調整のたびにコネクターが再起動される |
| ENTMQST-3763 | ZooKeeper ノードのスケールダウン時にエラーが発生する |
| ENTMQST-3422 | FIPS が有効なクラスターで実行できない |
| ENTMQST-3417 | ZooKeeperScaler でのキーストアのリーク/トラストストアのリークを修正 |
| ENTMQST-3583 | Cluster Operator で提供される JVM オプションが無視される |
| ENTMQST-3345 | 内部ブローカープロトコルとログメッセージ形式が M.m.p として含まれる Kafka のアップグレードで誤解を招くエラーが発生して失敗する |
| ENTMQST-3411 | KafkaExporter、CruiseControl、および EntityOperator Pod がクライアント CA の更新でロールされる |
| ENTMQST-3325 | KafkaMirrorMaker2 の条件が MM2 コネクターの状態を反映しない |
| ENTMQST-3504 | CPU 使用率が無効なために OptimizationFailureException が発生する |
| ENTMQST-3585 | Java システムプロパティーを Cruise Control に渡す |
| ENTMQST-3856 | ラックアウェアネスがコネクターで動作しない |
| ENTMQST-3354 | カスタムリソースでベースイメージが指定されると、そのベースイメージが Kafka Connect Build で適切に設定される |
| ENTMQST-3826 | /tmp ボリュームは圧縮ライブラリーに十分な大きさではない |
| ENTMQST-3584 | Kafka 関連の namespace を削除しても **strimzi_resources{kind="Kafka"}** メトリクスが削除されない |
| ENTMQST-3839 | ZooKeeper の接続解除後にブローカーが一貫性のない状態のままになる |
| ENTMQST-2331 | ZooKeeper、Kafka、および EntityOperator 証明書は、独自のクラスター CA 証明書を使用してが更新されていない |

**表5.1 CVE (Common Vulnerabilities and Exposures) の修正**

| 課題番号 | 説明 |
| --- | --- |
| ENTMQST-2851 | **CVE-2021-3520** lz4: memmove 引数が原因で、整数オーバーフローのバグによるメモリー破損が発生する |
| ENTMQST-3631 | **CVE-2021-43797** netty: ヘッダー名の制御文字が原因で HTTP 要求スマグリングが発生する可能性がある |

# 第6章 既知の問題

このセクションでは、AMQ Streams 2.1 の既知の問題について説明します。

## 6.1. 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。

#### 回避策 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>
```

**回避策 2: ホスト名の検証の無効化**

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>
```

## 6.2. CRUISE CONTROL の CPU 使用率予測

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、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。

### 回避策

この問題を回避する方法は 2 つあります。

### 回避策 1: CPU 要求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 要求を設定できます。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 ゴールに基づきリバランスプロポーザルを準備する際に、Cruise Control は CPU 使用率を適切に評価できます。

### 回避策 2: CPU ゴールを除外する

Cruise Control 設定に指定されたハードおよびデフォルトのゴールから CPU ゴールを除外できます。

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

# 第7章 サポート対象のインテグレーション製品

AMQ Streams 2.1 は、以下の Red Hat 製品とのインテグレーションをサポートします。

**Red Hat Single Sign-On**

OAuth 2.0 認証と OAuth 2.0 認証を提供します。

**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 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 Service 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。

これらの製品を使用することで AMQ Streams 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。

**関連情報**

- Red Hat Single Sign-On Supported Configurations
- Red Hat 3scale API Management Supported Configurations
- Red Hat Debezium Supported Configurations
- Red Hat Service Registry Supported Configurations

# 第8章 重要なリンク

- AMQ Streams Supported Configurations
- AMQ Streams Component Details

**改訂日時:** 2022-11-27 09:22:21 +1000