Red Hat AMQ Streams 2.3

AMQ Streams 2.3 on OpenShift リリースノート

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

Last Updated: 2023-04-10

Red Hat AMQ Streams 2.3 AMQ Streams 2.3 on OpenShift リリースノート

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

法律上の通知

概要

本リリースノートでは、AMQ Streams 2.3 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 2.3 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 2.3 on OpenShift は、Kafka 3.3.1 および Strimzi 0.32.x に基づいています。

注記

本リリースで解決された機能拡張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 2.3 は OpenShift Container Platform 4.8 から 4.12 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は、AMQ Streams Supported Configurations を参照してください。

1.2. KAFKA 3.3.1 のサポート

AMQ Streams は Apache Kafka バージョン 3.3.1 に対応するようになりました。

AMQ Streams は Kafka 3.3.1 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 3.3.1 にアップグレードする前に、Cluster Operator を AMQ Streams バージョン 2.3 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 3.3.0 および Kafka 3.3.1 のリリースノートを参照してください。

注記

Kafka 3.2.x は、AMQ Streams 2.3 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は、AMQ Streams Component Details を参照してください。

注記

Kafka 3.3.1 は、Kafka が Raft プロトコルを利用して ZooKeeper なしで実行される KRaft モードにアクセスできるようにします。KRaft モードは 開発者プレビュー として利用できます。

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 2.3 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。

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に、AMQ Streams では **Red Hat AMQ Streams API 変換ツール** と AMQ Streams 1.8 が提供されています。 AMQ Streams 1.8 ソフトウェアダウンロードページ からツールをダウンロードします。

カスタムリソースは、手順 2 つでアップグレードします。

ステップ 1: カスタムリソースの形式への変換

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

- 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。

詳しくは、1.7 より前の AMQ Streams バージョンからのアップグレード を参照してください。

1.4. CRUISE CONTROL 最適化提案の自動承認

AMQ Streams を Cruise Control とともに使用する場合、生成された最適化提案を承認するプロセスを自動化できるようになりました。**KafkaRebalance** カスタムリソースを使用して最適化の提案を生成します。自動承認を有効にするには、提案を生成する前に、**KafkaRebalance** カスタムリソースにアノテーションとして **strimzi.io/rebalance-auto-approval: "true"** を追加します。

手動承認では、生成された提案のステータスが **ProposalReady** のときに別のリクエストを行います。新しいリクエストの **KafkaRebalance** リソースに **strimzi.io/rebalance: approve** アノテーションを追加して、提案を承認します。

自動承認では、提案が生成されて承認され、1 回のリクエストでリバランスが完了します。

最適化提案の生成 を参照してください。

1.5. ACL ルール設定での複数操作のサポート

KafkaUser カスタムリソースが更新され、ACL リストの設定を管理しやすくなりました。

以前のリリースでは、**operation** プロパティーを使用して、リソースごとに ACL ルールの操作を個別に設定していました。

以前の ACL ルールの設定形式

```
authorization:
  type: simple
  acls:
    - resource:
        type: topic
        name: my-topic
      operation: Read
    - resource:
        type: topic
        name: my-topic
      operation: Describe
    - resource:
        type: topic
        name: my-topic
      operation: Write
    - resource:
        type: topic
        name: my-topic
      operation: Create
```

新しい **operation** プロパティーを使用すると、複数の ACL 操作を同じリソースの 1 つのルールとして一覧表示できます。

新しい ACL ルールの設定形式

```
authorization:
  type: simple
  acls:
    - resource:
        type: topic
        name: my-topic
      operations:
        - Read
        - Describe
        - Create
        - Write
```

古い設定形式の **operation** プロパティーは非推奨ですが、引き続きサポートされています。

ACLRule スキーマリファレンス を参照してください。

1.6. 新しい CLUSTER-IP 内部リスナータイプ

リスナーは、Kafka ブローカーへのクライアント接続に使用されます。これらは、**.spec.kafka.listeners** プロパティーを使用して **Kafka** リソースで設定されます。

内部リスナーの新しい **cluster-ip** タイプは、ブローカーごとの **ClusterIP** サービスに基づいて Kafka クラスターを公開します。

cluster-ip リスナーの設定例

```
#...
spec:
  kafka:
    #...
    listeners:
      - name: external-cluster-ip
        type: cluster-ip
        tls: false
        port: 9096
    #...
```

これは、ヘッドレスサービスを介してルーティングできない場合や、カスタムアクセスメカニズムを組み込む場合に便利なオプションです。たとえば、特定の Ingress コントローラーまたは Kubernetes Gateway API 用に独自のタイプの外部リスナーを構築するときに、このリスナーを使用できます。

GenericKafkaListener スキーマ参照 を参照してください。

1.7. 複数のレプリカを実行するための CLUSTER OPERATOR リーダーの選出

リーダーの選出を使用して、Cluster Operator の並列レプリカを複数実行します。1 つのレプリカがアクティブなリーダーとして選択され、デプロイされたリソースを操作します。他のレプリカはスタンバイモードで実行されます。レプリカは高可用性に役立ちます。レプリカを追加することで、重大な障害が原因で中断が発生しないようにします。これは、AMQ Streams が Kafka クラスターの Pod の作成と管理を処理する StrimziPodSets の導入以来、特に重要です。Cluster Operator は Pod の再起動を行います。

リーダーの選出を有効にするには、Cluster Operator の **STRIMZI_LEADER_ELECTION_ENABLED** 環境変数を true (デフォルト) に設定する必要があります。環境変数は、関連する環境変数とともに、Cluster Operator のデプロイに使用される **Deployment** カスタムリソースに設定されます。デフォルトでは、AMQ Streams は、常にリーダーレプリカである単一の Cluster Operator レプリカで実行されます。レプリカをさらに追加するには、**Deployment** カスタムリソースの **spec.replicas** 値を更新します。

Cluster Operator レプリカとリーダー選出の Deployment 設定

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: strimzi-cluster-operator
  labels:
    app: strimzi
spec:
  replicas: 1
  # ...
    spec:
      # ...
      containers:
        - name: strimzi-cluster-operator
          image: registry.redhat.io/amq7/amq-streams-rhel8-operator:2.3.0
          # ...
          env:
```

```
        # ...
        - name: STRIMZI_LEADER_ELECTION_ENABLED
          value: "true"
        - name: STRIMZI_LEADER_ELECTION_LEASE_NAME
          value: "strimzi-cluster-operator"
        - name: STRIMZI_LEADER_ELECTION_LEASE_NAMESPACE
          valueFrom:
            fieldRef:
              fieldPath: metadata.namespace
        - name: STRIMZI_LEADER_ELECTION_IDENTITY
          valueFrom:
            fieldRef:
# ...
```

リーダー選出による複数の Cluster Operator レプリカの実行 および リーダー選出環境変数 を参照してください。

1.8. IBM Z および LINUXONE アーキテクチャーのサポート

AMQ Streams 2.3 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 と LinuxONE のサポートは、OpenShift Container Platform 4.10 以降で Kafka で実行している AMQ Streams が対象となります。

1.8.1. IBM Z および LinuxONE の要件

- OpenShift Container Platform 4.10 以降

1.8.2. IBM Z および LinuxONE でのサポート対象外

- 非接続 OpenShift Container Platform 環境の AMQ Streams
- AMQ Streams OPA の統合

1.9. IBM POWER アーキテクチャーのサポート

AMQ Streams 2.3 は、BM Power ppc64le アーキテクチャーでの実行が可能になりました。

IBM Power のサポートは、OpenShift Container Platform 4.9 以降において Kafka で実行している AMQ Streams が対象です。

1.9.1. IBM Power の要件

- OpenShift Container Platform 4.9 以降

1.9.2. IBM Power でサポート対象外

- 非接続 OpenShift Container Platform 環境の AMQ Streams

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

Red Hat build of Debezium は、分散型の変更データキャプチャープラットフォームです。データベース

の行レベルの変更をキャプチャーして、変更イベントレコードを作成し、Kafka トピックにレコードをストリーミングします。Debezium は Apache Kafka に構築されます。AMQ Streams で Red Hat build of 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 AMQ Streams on OpenShift 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 をデプロイする方法の詳細は、Red Hat build of Debezium に関するドキュメント 参照してください。

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

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

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

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

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

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

AMQ Streams で Red Hat build of Apicurio Registry を使用する方法の詳細は、Red Hat build of Apicurio Registry に関するドキュメント を参照してください。

第2章 機能拡張

AMQ Streams 2.3 では、多くの機能拡張が追加されました。

2.1. KAFKA 3.3.1 で改良された機能

Kafka 3.3.0 および 3.3.1 で導入された拡張機能の概要については、Kafka 3.3.0 および Kafka 3.3.1 のリリースノートを参照してください。

2.2. KAFKA コネクターのステータス

コネクターまたは関連するタスクが **FAILED** として報告されている場合、**KafkaConnector** カスタムリソースのステータスが **NotReady** と表示されるようになりました。以前は、コネクターまたはタスクが失敗した場合でも、カスタムリソースに **READY** が表示されていました。

2.3. CONTROLPLANELISTENER フィーチャーゲートが GA に移行

ControlPlaneListener フィーチャーゲートは一般公開され (GA に移行)、永続的に有効になり、無効にすることはできません。

ControlPlaneListener が有効にされている場合、Kafka コントローラーとブローカー間の接続はポート 9090 の内部コントロールプレーンリスナーを使用します。

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

重要

ControlPlaneListener フィーチャーゲートを永続的に有効にすると、AMQ Streams 1.7 以前と AMQ Streams 2.3 以降の間で直接的にアップグレードまたはダウングレードができなくなります。以下の AMQ Streams バージョンのいずれかをアップグレードまたはダウングレードする必要があります。詳しくは、1.7 より前の AMQ Streams バージョンからのアップグレード を参照してください。

2.4. SERVICEACCOUNTPATCHING フィーチャーゲートが GA に移行

ServiceAccountPatching フィーチャーゲートは一般公開され (GA に移行)、永続的に有効になり、無効にすることはできません。

ServiceAccountPatching を有効にすると、Cluster Operator は常にサービスアカウントを調整し、必要に応じて更新します。たとえば、カスタムリソースの template プロパティーを使用してサービスアカウントのラベルまたはアノテーションを変更すると、Operator はそれらを既存のサービスアカウントリソースで自動的に更新します。

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

2.5. USESRIMZIPODSETS フィーチャーゲートがベータ版に移行

UseSrimziPodSets フィーチャーゲートは成熟度のベータレベルに移行し、デフォルトで有効になりました。これは、StatefulSets の代わりに StrimziPodSet がデフォルトで使用されることを意味します。

フィーチャーゲートは、**StrimziPodSet** と呼ばれる Pod を管理するためのリソースを制御します。AMQ Streams は、OpenShift ではなく Pod の作成および管理を処理します。StatefulSets の代わりに StrimziPodSets を使用すると、機能の制御が強化されます。

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

2.6. KAFKA BRIDGE のラック認識設定

Kafka Bridge Pod のラック認識がサポートされるようになりました。**KafkaBridge** カスタムリソースを使用して、ラック認識を設定します。Kafka Bridge Pod が実行されているラックを認識するように設定できます。ラックとして、アベイラビリティーゾーン、データセンター、またはデータセンターの実際のラックを指定できます。

Kafka Bridge の rack 設定例

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

ラック スキーマリファレンス を参照してください。

2.7. プラグイン可能な POD セキュリティープロファイル

セキュリティーコンテキストは、Pod とコンテナーの制約を定義します。OpenShift はビルトイン SCC (Security Context Constraints) を使用してパーミッションを制御します。SCC は、Pod がアクセスできるセキュリティー機能を制御する設定およびストラテジーです。独自の SCC を作成して管理することもできます。

Cluster Operator のオプションの **STRIMZI_POD_SECURITY_PROVIDER_CLASS** 環境変数は、Pod とコンテナーのセキュリティーコンテキスト設定を提供します。

AMQ Streams Pod およびコンテナーへのセキュリティーコンテキストの適用 を参照してください。

2.8. KAFKA ブローカーの再起動イベント

Cluster Operator が OpenShift クラスターで Kafka Pod を再起動した後に、Pod が再起動した理由を説明する OpenShift イベントを Pod の namespace 内で発行します。クラスターの動作を理解するために、コマンドラインから再起動イベントの理由を確認できます。コマンドラインから再起動 **events** をチェックする場合は、**reason** またはその他の **field-selector** オプションを指定して、返されるイベントをフィルタリングすることもできます。

次の例では、エラーが原因でトリガーされた再起動イベントを返します。

指定された理由でトリガーされた再起動イベントを返す

```
oc -n kafka get events --field-selector reportingController=strimzi.io/cluster-
operator,reason=PodForceRestartOnError
```

Kafka の再起動に関する情報の検索 を参照してください。

2.9. 設定可能な KAFKA 管理クライアント

STRIMZI_KAFKA_ADMIN_CLIENT_CONFIGURATION と呼ばれる新しい User Operator 環境変数は、追加の設定を Kafka 管理クライアントに渡すことができます。Kafka Admin クライアントは、ブローカーとトピックの管理に役立ちます。Kafka Admin クライアントは、User Operator の再構築なしに調整できるようになりました。たとえば、このクライアントを使用して、SASL 設定を渡したり、タイムアウトを調整したりできます。

Kafka 管理クライアントのタイムアウト設定

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: strimzi-user-operator
  labels:
    app: strimzi
spec:
  # ...
  template:
    # ...
    spec:
      # ...
      containers:
        - name: strimzi-user-operator
          # ...
          env:
            - name: STRIMZI_KAFKA_ADMIN_CLIENT_CONFIGURATION
              value: |
                default.api.timeout.ms=120000
                request.timeout.ms=60000
```

注記

このタイムアウト設定は、高度な設定オプションであり、検証なしで提供されます。

スタンドアロン User Operator のデプロイ を参照してください。

2.10. CRUISE CONTROL 容量のオーバーライド

新しい Cruise Control 設定オプションを使用すると、各 Kafka ブローカーのネットワーク容量と CPU 制限を設定するオーバーライドを指定できます。これらのオプションは、ブローカーが異種ネットワークまたは CPU リソースを持つノードで実行している場合に使用できます。

次のブローカリソースに対してオーバーライド容量制限を設定できます。

- **cpu** - ミリコアまたは CPU コアの CPU リソース (デフォルト: 1)
- **inboundNetwork**: バイト毎秒単位のインバウンドネットワークスループット (デフォルトは 10000 KiB/s)
- **outboundNetwork**: バイト毎秒単位のアウトバウンドネットワークスループット (デフォルトは 10000 KiB/s)

ビバイト (bibyte) 単位を使用した Cruise Control 容量オーバーライド設定の例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  # ...
  cruiseControl:
    # ...
    brokerCapacity:
      cpu: "1"
      inboundNetwork: 10000KiB/s
      outboundNetwork: 10000KiB/s
      overrides:
      - brokers: [0]
        cpu: "2.755"
        inboundNetwork: 20000KiB/s
        outboundNetwork: 20000KiB/s
      - brokers: [1, 2]
        cpu: 3000m
        inboundNetwork: 30000KiB/s
        outboundNetwork: 30000KiB/s
```

容量のオーバーライド を参照してください。

2.11. KAFKA クライアントの OAUTH 2.0 パスワード付与

Kafka ブローカーとの対話に OAuth パスワード付与メカニズムを使用するように Kafka クライアントを設定できるようになりました。

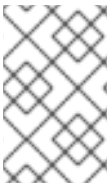
パスワード付与メカニズムのプロパティー

```
security.protocol=SASL_SSL
sasl.mechanism=OAUTHBEARER
sasl.jaas.config=org.apache.kafka.common.security.oauthbearer.OAuthBearerLoginModule required \
  oauth.token.endpoint.uri="<token_endpoint_url>" \
  oauth.client.id="<client_id>" \ 1
  oauth.client.secret="<client_secret>" \ 2
  oauth.password.grant.username="<username>" \ 3
  oauth.password.grant.password="<password>" \ 4
  oauth.scope="<scope>" \
  oauth.audience="<audience>" ;
  # ...
```

1. クライアント ID。承認サーバーで **クライアント** を作成するときに使用される名前です。
2. (オプション) 承認サーバーで **クライアント** を作成するときに作成されるクライアントシークレット。
3. パスワード付与認証のユーザー名。OAuth パスワード付与設定 (ユーザー名とパスワード) は、OAuth 2.0 パスワード付与メソッドを使用します。パスワード付与を使用するには、権限が制限された認可サーバーにクライアント用のユーザーアカウントを作成します。アカウントは、サービスアカウントのように機能する必要があります。認証にユーザーアカウントが必要な環境で使用しますが、最初に更新トークンの使用を検討してください。

4 パスワード付与認証のパスワード。

OAuth 2.0 を使用するための Kafka Java クライアントの設定 を参照してください。



注記

リリース時に、AMQ Streams on OpenShift 用の Kafka Bridge でパスワード付与が現時点で機能しないという小さな問題が発見されました。詳細は、OAuth パスワード付与の設定に関する既知の問題 を参照してください。

2.12. 認証と承認のメトリクス

oauth 認証と **opa** または **keycloak** 承認に固有のメトリクスを収集できるようになりました。これを行うには、Kafka リソースのリスナー設定で **enableMetrics** プロパティーを true に設定します。たとえば、**spec.kafka.listeners.authentication** および **spec.kafka.authorization** で **enableMetrics** を **true** に設定します。同様に、**KafkaBridge**、**KafkaConnect**、**KafkaMirrorMaker**、および **KafkaMirrorMaker2** カスタムリソースで oauth 認証のメトリクスを有効にすることができます。

メトリクスの導入 を参照してください。

第3章 テクノロジープレビュー

AMQ Streams 2.3 には、テクノロジープレビュー機能が含まれます。

重要

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

3.1. 分散トレース用の OPENTELEMETRY

このリリースでは、テクノロジープレビューとして、分散トレーシング用の OpenTelemetry が導入されています。指定されたトレースシステムと合わせて OpenTelemetry を使用できます。OpenTelemetry は、分散トレーシングの OpenTracing に取って代わります。OpenTracing のサポートは非推奨となりました。

デフォルトでは、OpenTelemetry はトレースに OTLP (OpenTelemetry Protocol) エクスポーターを使用します。OpenTelemetry 付きの AMQ Streams は、Jaeger エクスポーターで使用するために配布されますが、OpenTelemetry でサポートされている他のトレースシステムを指定できます。AMQ Streams は、デフォルトで OTLP エクスポーターで OpenTelemetry を使用するように移行する予定であり、Jaeger エクスポーターのサポートを段階的に廃止しています。

分散トレースの紹介 を参照してください。

3.2. 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。

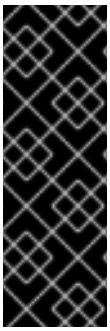
Kafka Static Quota プラグインの設定例

```
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。

第4章 開発者プレビュー

AMQ Streams 2.3 には、開発者プレビュー機能が含まれます。



重要

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

4.1. USEKRAFT フィーチャーゲート

Kafka クラスター管理者は、Cluster Operator デプロイメント設定でフィーチャーゲートを使用して、機能のサブセットのオンとオフを切り替えることができます。

Apache Kafka は、ZooKeeper を必要性をなくすため、段階的にこの機能を廃止しています。新しい **UseKRaft** フィーチャーゲートを有効にすると、ZooKeeper なしで KRaft (Kafka Raft メタデータ) モードで Kafka クラスターのデプロイを試すことができます。

このフィーチャーゲートは成熟度のアルファレベルにあり、開発者プレビューとして扱う必要があります。

注意

このフィーチャーゲートは実験的なものであり、開発とテスト **のみ** を目的としており、実稼働環境では有効にしないでください。

UseKRaft フィーチャーゲートを有効にするには、Cluster Operator 設定で **STRIMZI_FEATURE_GATES** 環境変数の値として **+UseKRaft** および **+USeSrimziPodSets** を指定します。**UseKRaft** フィーチャーゲートは、**UseSrimziPodSets** フィーチャーゲートに依存します。

UseKRaft フィーチャーゲートの有効化

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

現在、AMQ Streams の KRaft モードには、次の主要な制限があります。

- ZooKeeper を使用する Kafka クラスターから KRaft クラスターへの移動、またはその逆の移動はサポートされていません。
- Apache Kafka バージョンまたは AMQ Streams Operator のアップグレードとダウングレードはサポートされていません。ユーザーは、クラスターを削除し、Operator をアップグレードして、新しい Kafka クラスターをデプロイする必要がある場合があります。
- Topic Operator はサポートされていません。**spec.entityOperator.topicOperator** プロパティーを **Kafka** カスタムリソースから **削除する必要があります**。
- SCRAM-SHA-512 認証はサポートされていません。

- JBOD ストレージはサポートされていません。**type: jbod** ストレージを使用できますが、JBOD アレイに含めることができるディスクは 1 つだけです。
- Liveness および Readiness プローブは無効になっています。
- すべての Kafka ノードには、**controller** と **broker** の両方の KRaft ロールがあります。個別の **controller** と **broker** ノードを持つ Kafka クラスターはサポートされていません。

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

第5章 KAFKA の重大な変更

このセクションでは、AMQ Streams が継続して機能させるために、加える必要のある変更内容について説明します。

5.1. KAFKA のサンプルファイルコネクターの使用

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

コネクター設定ファイルの 2 つのサンプルが提供されています。

- **examples/connect/kafka-connect-build.yaml** は、Kafka Connect **build** 設定を提供します。これをデプロイして、ファイルコネクターを使用して新しい KafkaConnect イメージをビルドすることができます。
- **examples/connect/source-connector.yaml** は、ファイルコネクターを **KafkaConnector** リソースとしてデプロイするために必要な設定を提供します。

KafkaConnector リソース例のデプロイ および コネクタープラグインを使用した Kafka Connect の拡張 を参照してください。

第6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。

6.1. AMQ STREAMS 2.4.0 で削除された JAVA 8 サポート

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 のサポートは AMQ Streams 2.4.0 で削除されます。これは、クライアントを含むすべての AMQ Streams コンポーネントが対象です。

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。

当面の間 Java 8 を使い続ける方向けに、AMQ Streams 2.2 は Long Term Support (LTS) を提供します。LTS の期間および日付については、AMQ Streams LTS サポートポリシー を参照してください。

6.2. OPENTRACING

type: jaeger トレースのサポートは非推奨です。

Jaeger クライアントは廃止され、OpenTracing プロジェクトはアーカイブされました。そのため、将来の Kafka バージョンのサポートを保証することはできません。OpenTelemetry プロジェクトに基づく新しいトレース実装を導入しています。

6.3. ACL ルールの設定

ACL ルールの操作を設定するための **operation** プロパティーは廃止されました。**operations** プロパティーを使用した、より合理化された新しい設定形式が利用できるようになりました。詳細は、「ACL ルール設定での複数操作のサポート」 を参照してください。

6.4. 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。

6.5. CRUISE CONTROL TLS サイドカーのプロパティー

Cruise Control TLS サイドカーが削除されました。その結果、**.spec.cruiseControl.tlsSidecar** および **.spec.cruiseControl.template.tlsSidecar** プロパティーは非推奨になりました。プロパティーは無視され、今後削除されます。

6.6. 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、今後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。

6.7. 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、今後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。

6.8. 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、今後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目標が破られているかどうかを判断するための最適化提案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。

Kafka を使用した Cruise Control の設定およびデプロイ を参照してください。

第7章 修正された問題

AMQ Streams 2.3 on OpenShift で、以下の問題が修正されました。

Kafka 3.3.0 および 3.3.1 で修正された問題の詳細は、Kafka 3.3.0 および Kafka 3.3.1 のリリースノートを参照してください。

表7.1 修正された問題

課題番号	説明
ENTMQST-4273	inter.broker.protocol.version のみが設定されている場合に、アップグレード中に不要なローリング更新を避けるようにしてください。
ENTMQST-4095	Connect からのレスポンス形式が不正であるため、オペレーターが 1 つの調整で停止する可能性があります。
ENTMQST-1366	未使用の **@DefaultValue** アノテーションを削除します。
ENTMQST-4065	ユーザーが間違った TLS シークレットまたはそのシークレット内のキーを指定した場合の NPE を修正します。
ENTMQST-4178	スタンドアロンの UO および TO インストールファイルのさまざまな問題を修正します。
ENTMQST-4177	/tmp ディレクトリーを消去しようとしたときのエラーを無視します。
ENTMQST-4483	KafkaTopic **config: {}** が自動的に追加または削除されます。
ENTMQST-4016	**KafkaRebalance** 関連のアノテーション値が無効または不明な場合にログに記録されます。
ENTMQST-3840	ZooKeeper のローリング更新で準備のできていない Pod を処理してしまいます。
ENTMQST-4093	KAFKA **log.cleaner.io.max.bytes.per.second** は変更できません。

表7.2 CVE (Common Vulnerabilities and Exposures) の修正

課題番号	説明
ENTMQST-4312	CVE-2022-42004 jackson-databind: 深くネストされた配列の使用
ENTMQST-4311	CVE-2022-42003 jackson-databind: UNWRAP_SINGLE_VALUE_ARRAYS に関する深いラッパー配列のネスト
ENTMQST-4302	CVE-2022-38752 snakeyaml: java.base/java.util.ArrayList.hashCode の捕捉されなかった例外
ENTMQST-4188	CVE-2022-2047 jetty-http: インプルーバーのホスト名入力処理

第8章 既知の問題

このセクションでは、AMQ Streams 2.3 on OpenShift の既知の問題について説明します。

8.1. CORS を有効にしてメッセージを送信する KAFKA BRIDGE

Kafka Bridge で Cross-Origin Resource Sharing (CORS) が有効になっている場合、HTTP リクエストを送信してメッセージを生成すると、**400 bad request エラー**が返されます。

回避策

このエラーを回避するには、Kafka Bridge 設定で CORS を無効にします。メッセージを生成する HTTP リクエストでは、Kafka ブリッジで CORS を無効にする必要があります。この問題は、AMQ Streams の今後のリリースで修正される予定です。

CORS を使用するには、Kafka Bridge に Red Hat 3scale をデプロイしてください。

- 3scale のデプロイについては、3scale API Management と AMQ Streams Kafka Bridge の使用 を参照してください。
- 3scale による CORS リクエスト処理の詳細は、API ゲートウェイの管理 を参照してください。

8.2. 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。以下に例を示します。

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
```

```
metadata:
  name: amq-streams
  namespace: <operator_namespace>
spec:
  channel: amq-streams-1.8.x
  installPlanApproval: Automatic
  name: amq-streams
  source: mirror-amq-streams
  sourceNamespace: openshift-marketplace
  config:
    env:
    - name: KUBERNETES_MASTER
      value: MASTER-ADDRESS
```

5. エディターを保存し、終了します。
6. **Subscription** が更新されていることを確認します。

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

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

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

回避方法 2: ホスト名検証の無効化

1. すべての Operator サブスクリプションを一覧表示します。

```
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>
```

8.3. 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 リクエストと制限を同等レベルにする

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

回避策 2: CPU の目標を除外する

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

CPU の目標がない Cruise Control の設定例

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

```
    com.linkedin.kafka.cruisecontrol.analyzer.goals.ReplicaCapacityGoal,
    com.linkedin.kafka.cruisecontrol.analyzer.goals.DiskCapacityGoal,
    com.linkedin.kafka.cruisecontrol.analyzer.goals.NetworkInboundCapacityGoal,
    com.linkedin.kafka.cruisecontrol.analyzer.goals.NetworkOutboundCapacityGoal
  default.goals: >
    com.linkedin.kafka.cruisecontrol.analyzer.goals.RackAwareGoal,
    com.linkedin.kafka.cruisecontrol.analyzer.goals.MinTopicLeadersPerBrokerGoal,
    com.linkedin.kafka.cruisecontrol.analyzer.goals.ReplicaCapacityGoal,
    com.linkedin.kafka.cruisecontrol.analyzer.goals.DiskCapacityGoal,
    com.linkedin.kafka.cruisecontrol.analyzer.goals.NetworkInboundCapacityGoal,
    com.linkedin.kafka.cruisecontrol.analyzer.goals.NetworkOutboundCapacityGoal,
    com.linkedin.kafka.cruisecontrol.analyzer.goals.ReplicaDistributionGoal,
    com.linkedin.kafka.cruisecontrol.analyzer.goals.PotentialNwOutGoal,
    com.linkedin.kafka.cruisecontrol.analyzer.goals.DiskUsageDistributionGoal,
    com.linkedin.kafka.cruisecontrol.analyzer.goals.NetworkInboundUsageDistributionGoal,
    com.linkedin.kafka.cruisecontrol.analyzer.goals.NetworkOutboundUsageDistributionGoal,
    com.linkedin.kafka.cruisecontrol.analyzer.goals.TopicReplicaDistributionGoal,
    com.linkedin.kafka.cruisecontrol.analyzer.goals.LeaderReplicaDistributionGoal,
    com.linkedin.kafka.cruisecontrol.analyzer.goals.LeaderBytesInDistributionGoal
```

詳細は、CPU 容量の不足 を参照してください。

8.4. ユーザー OPERATOR のスケーラビリティー

複数のユーザーを同時に作成すると、User Operator がタイムアウトになることがあります。調整には時間がかかりすぎる場合があります。

回避策

この問題が発生した場合は、同時に作成するユーザーの数を減らしてください。また、準備が整ってから追加のユーザーを作成するようにしてください。

8.5. OAUTH パスワード付与の設定

現在 Kafka Bridge では、OAuth パスワード付与が正しく処理されません。OAuth 認証が正しく設定されていません。

これは、次のリリースで修正される予定です。

課題番号	説明
ENTMQST-4479	新しく追加された OAuth パスワード付与機能が Kafka Bridge で機能しない

8.6. OPENTELEMETRY: TLS が有効な JAEGER の実行

OpenTelemetry を使用したトレースのサポートは、次の Kafka コンポーネントに組み込まれています。

- Kafka Connect
- MirrorMaker

- MirrorMaker 2
- AMQ Streams Kafka Bridge

Jaeger エクスポーターを使用する場合、トレースデータは Jaeger gPRC エンドポイントを介して取得されます。デフォルトでは、このエンドポイントでは TLS が有効になっていません。ただし、Jaeger Operator を使用して Jaeger インスタンスをデプロイするときに、TLS を使用するように設定することもできます。たとえば、Jaeger Operator である OpenShift で Red Hat OpenShift 分散トレース Operator を実行すると、Operator は自動的に TLS を有効にします。gRPC エンドポイントで TLS を使用する Jaeger インスタンスは、AMQ ストリームではサポートされていません。

この問題を回避する方法は 2 つあります。

回避策 1: gRPC エンドポイントで TLS を無効にする

Jaeger カスタムリソースを作成し、次のプロパティーを指定して gRPC ポートで TLS を無効にします。

- **collector.grpc.tls.enabled: false**
- **reporter.grpc.tls.enabled: false**

TLS を無効にする Jaeger カスタムリソースの例

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: my-jaeger
spec:
  allInOne:
    options:
      agent.grpc.tls.enabled: false
      collector.grpc.tls.enabled: false
```

注記

この設定は、すべての Jaeger コンポーネントを単一の Pod にデプロイする **allInOne** ストラテジーを使用します。実稼働環境の **production** 戦略などの他のデプロイメント戦略では、スケーラビリティーと信頼性を高めるために、Jaeger コンポーネントを個別の Pod に分離します。

回避策 2: OpenTelemetry コレクターを使用してトレースをエクスポートする

コレクターを使用して、OpenTelemetry トレースデータを受信、処理、およびエクスポートします。OpenTelemetry コレクターを使用してトレースデータをエクスポートして問題を解決するには、次の手順に従います。

1. Red Hat OpenShift 分散トレース収集オペレーターをデプロイします。
2. **OpenTelemetryCollector** カスタムリソースを設定してコレクターをデプロイし、TLS が有効でないエンドポイントを介してトレースデータを受信し、それを TLS が有効なエンドポイントに渡します。
3. カスタムリソースで、**receivers** プロパティーを指定して、ポート 14250 で TLS が有効でない Jaeger gRPC エンドポイントを作成します。他のトレースシステムを使用している場合は、OTLP エンドポイントなどの他のエンドポイントを作成することもできます。

4. **exporters** プロパティーを指定して、TLS が有効な Jaeger gRPC エンドポイントを指すようにします。

5. カスタムリソースの **pipelines** プロパティーでパイプライン設定を宣言します。

この例では、パイプラインは Jaeger および OTLP レシーバーから Jaeger gRPC エンドポイントまでです。

OpenTelemetry コレクターの設定例

```
apiVersion: opentelemetry.io/v1alpha1
kind: OpenTelemetryCollector
metadata:
  name: cluster-collector
  namespace: <namespace>
spec:
  mode: deployment
  config: |
   receivers:
     otlp:
       protocols:
         grpc:
         http:
     jaeger:
       protocols:
         grpc:
   exporters:
     jaeger:
     endpoint: jaeger-all-in-one-inmemory-collector-headless.openshift-distributed-
tracing.svc.cluster.local:14250
       tls:
         ca_file: "/var/run/secrets/kubernetes.io/serviceaccount/service-ca.crt"
   service:
     pipelines:
       traces:
         receivers: [otlp,jaeger]
         exporters: [jaeger]
```

コレクターを使用するには、トレース設定でコレクターエンドポイントをエクスポーターエンドポイントとして指定する必要があります。

OpenTelemetry を使用した Kafka Connect のトレース設定の例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
  name: my-connect-cluster
spec:
  #...
  template:
    connectContainer:
      env:
        - name: OTEL_SERVICE_NAME
          value: my-otel-service
        - name: OTEL_EXPORTER_JAEGER_ENDPOINT
```

```
        value: "http:// jaeger-all-in-one-inmemory-collector-headless.openshift-distributed-
tracing.svc.cluster.local:14250"
  tracing:
    type: opentelemetry
  #...
```

第9章 RED HAT 製品へのサポートされる統合

AMQ Streams 2.3 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 build of Apicurio Registry

データストリーミングのサービススキーマの集中型ストアを提供します。

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10章 重要なリンク

- AMQ Streams Supported Configurations
- AMQ Streams Component Details

改訂日時: 2023-04-06