# Red Hat AMQ Streams 2.3

## OpenShift 上の AMQ Streams の概要

OpenShift Container Platform での AMQ Streams 2.3 の特徴と機能の確認

Last Updated: 2023-04-10

# Red Hat AMQ Streams 2.3 OpenShift 上の AMQ Streams の概要

OpenShift Container Platform での AMQ Streams 2.3 の特徴と機能の確認

## 法律上の通知



## 概要

Kafka コンポーネントの機能と、AMQ Streams を使用して OpenShift に Kafka をデプロイして管理する方法について説明します。

# 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 主な特長

AMQ Streams は、OpenShift クラスターで Apache Kafka を実行するプロセスを簡素化します。

本書は、AMQ Streams を理解するためのスタート地点となるように作成されました。また、本書では AMQ Streams の中心となる Kafka の主要な概念をいくつか紹介し、Kafka コンポーネントの目的を簡単に説明します。Kafka のセキュリティーや監視オプションなど、設定ポイントを概説します。AMQ Streams のディストリビューションでは、Kafka クラスターのデプロイおよび管理ファイルと、デプロイメントの設定およびモニタリングのサンプルファイル を提供します。

一般的な Kafka デプロイメントと、Kafka のデプロイおよび管理に使用するツールについて説明します。

## 1.1. KAFKA の機能

Kafka の基盤のデータストリーム処理機能とコンポーネントアーキテクチャーによって以下が提供されます。

- スループットが非常に高く、レイテンシーが低い状態でデータを共有するマイクロサービスおよびその他のアプリケーション
- メッセージの順序の保証
- アプリケーションの状態を再構築するためにデータストレージからメッセージを巻き戻し/再生
- キーバリューログの使用時に古いレコードを削除するメッセージ圧縮
- クラスター設定での水平スケーラビリティー
- 耐障害性を制御するデータのレプリケーション
- 即時アクセス用の大量データの保持

## 1.2. KAFKA のユースケース

Kafka の機能は、以下に適しています。

- イベント駆動型のアーキテクチャー
- アプリケーションの状態に加えられた変更をイベントのログとしてキャプチャーするイベントソーシング
- メッセージのブローカー
- Web サイトアクティビティーの追跡
- メトリクスによる運用上のモニタリング
- ログの収集および集計
- 分散システムのコミットログ
- アプリケーションがリアルタイムでデータに対応できるようにするストリーム処理

## 1.3. AMQ STREAMS による KAFKA のサポート

AMQ Streams は、Kafka を OpenShift で実行するためのコンテナーイメージおよび Operator を提供します。AMQ Streams Operator は、AMQ Streams の実行に必要です。AMQ Streams では、専門的な運用知識をもとに、専用の Operator を提供し、Kafka を効果的に管理します。

Operator は以下のプロセスを単純化します。

- Kafka クラスターのデプロイおよび実行
- Kafka コンポーネントのデプロイおよび実行
- Kafka へアクセスするための設定
- Kafka へのアクセスのセキュア化
- Kafka のアップグレード
- ブローカーの管理
- トピックの作成および管理
- ユーザーの作成および管理

# 第2章 AMQ STREAMS での KAFKA のデプロイメント

Apache Kafka コンポーネントは、AMQ Streams ディストリビューションを使用して OpenShift にデプロイする場合に提供されます。Kafka コンポーネントは通常、クラスターとして実行され、可用性を確保します。

Kafka コンポーネントが組み込まれた通常のデプロイメントには以下が含まれます。

- ブローカーノードの **Kafka** クラスター
- レプリケートされた ZooKeeper インスタンスの **zookeeper** クラスター
- 外部データ接続用の **Kafka Connect** クラスター
- セカンダリークラスターで Kafka クラスターをミラーリングする **Kafka MirrorMaker** クラスター
- 監視用に追加の Kafka メトリクスデータを抽出する **Kafka Exporter**
- Kafka クラスターに対して HTTP ベースの要求を行う **Kafka Bridge**

少なくとも Kafka および ZooKeeper は必要ですが、上記のコンポーネントがすべて必須なわけではありません。MirrorMaker や Kafka Connect など、一部のコンポーネントでは Kafka なしでデプロイできます。

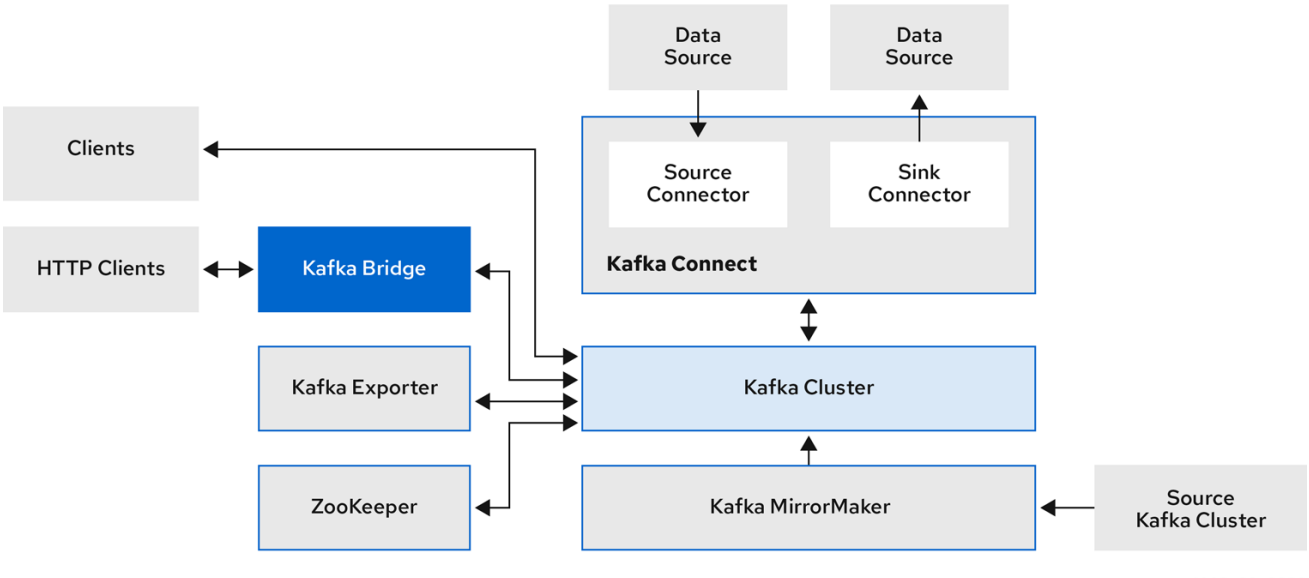
## 2.1. KAFKA コンポーネントのアーキテクチャー

Kafka ブローカーのクラスターがメッセージの配信を処理します。

ブローカーは、設定データの保存やクラスターの調整に Apache ZooKeeper を使用します。Apache Kafka の実行前に、Apache ZooKeeper クラスターを用意する必要があります。

他の Kafka コンポーネントはそれぞれ Kafka クラスターと対話し、特定のロールを実行します。

### Kafka コンポーネントの操作




### Apache ZooKeeper

Apache ZooKeeper はクラスター調整サービスを提供し、ブローカーおよびコンシューマーのステータスを保存して追跡するので、Kafka のコアとなる依存関係です。ZooKeeper は、コントローラーの選出にも使用されます。

**Kafka Connect**

Kafka Connect は、**Connector** プラグインを使用して Kafka ブローカーと他のシステムの間でデータをストリーミングする統合ツールです。Kafka Connect は、Kafka と、データベースなどの外部データソースまたはターゲットを統合するためのフレームワークを提供し、コネクターを使用してデータをインポートまたはエクスポートします。コネクターは、必要な接続設定を提供するプラグインです。

- **ソース** コネクターは、外部データを Kafka にプッシュします。
- **sink** コネクターは Kafka からデータを抽出します。
  外部データは適切な形式に変換されます。

  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。

**Kafka MirrorMaker**

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 台の Kafka クラスター間でデータをレプリケーションします。
MirrorMaker はソースの Kafka クラスターからメッセージを取得して、ターゲットの Kafka クラスターに書き込みます。

**Kafka Bridge**

Kafka Bridge には、HTTP ベースのクライアントと Kafka クラスターを統合する API が含まれています。

**Kafka Exporter**

Kafka Exporter は、Prometheus メトリクス (主にオフセット、コンシューマーグループ、コンシューマーラグおよびトピックに関連するデータ) として分析用にデータを抽出します。コンシューマーラグとは、パーティションに最後に書き込まれたメッセージと、そのパーティションからコンシューマーが現在取得中のメッセージとの間の遅延を指します。

# 第3章 KAFKA の概要

Apache Kafka は、耐障害性のリアルタイムデータフィードを実現する、オープンソースの分散型 publish/subscribe メッセージングシステムです。

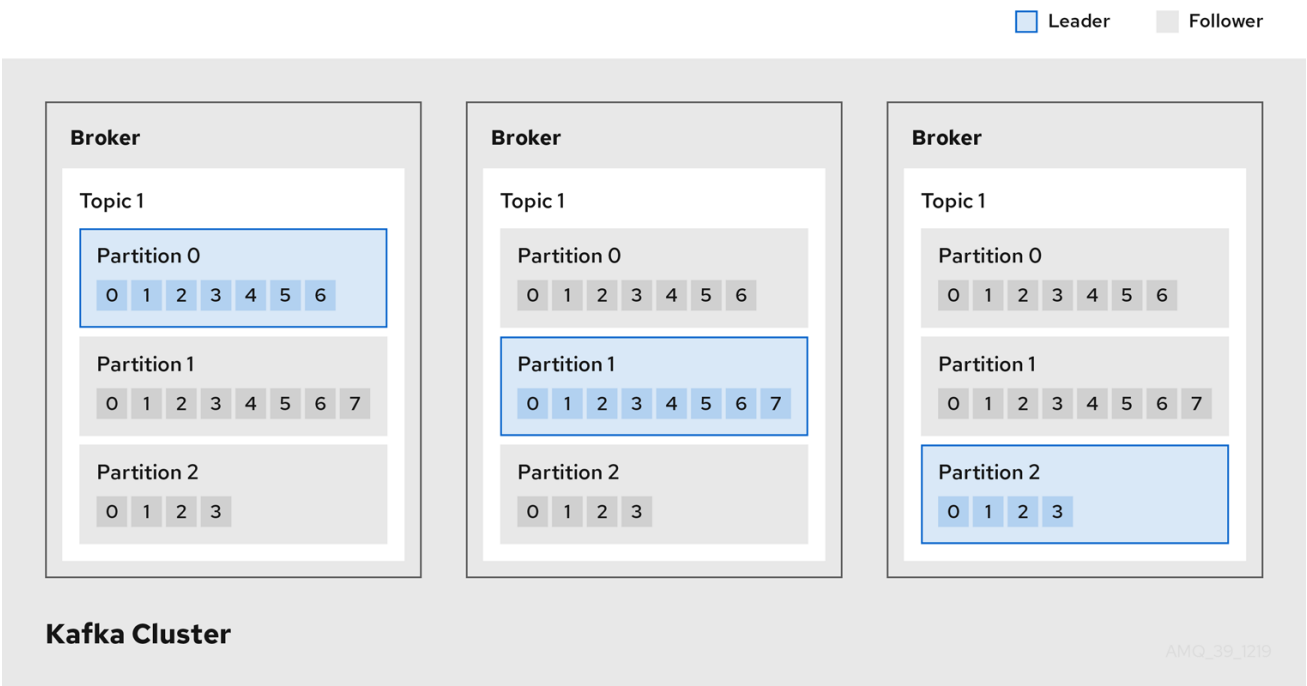
Apache Kafka の詳細については、Apache Kafka のドキュメント を参照してください。

## 3.1. KAFKA のメッセージブローカーとしての動作

AMQ Streams の使用経験を最大化するには、Kafka がメッセージブローカーとしてどのように動作するかを理解する必要があります。

Kafka クラスターは、複数のブローカーで設定されます。ブローカーには、データの受信およびストアに関するトピックが含まれます。トピックはパーティションに分割され、そこにデータが書き込まれます。パーティションは、耐障害性を確保するため、複数のトピックでレプリケーションされます。

### Kafka ブローカーおよびトピック




**ブローカー**

サーバーまたはノードと呼ばれるブローカーは、ストレージとメッセージの受け渡しをオーケストレーションします。

**トピック**

トピックは、データの保存先を提供します。各トピックは、複数のパーティションに分割されます。

**クラスター**

Broker インスタンスのグループ

**パーティション**

トピックパーティションの数は、トピックの **PartitionCount** (パーティション数) で定義されます。

**パーティションリーダー**

パーティションリーダーは、トピックの全プロデューサー要求を処理します。

**パーティションフォロワー**

パーティションフォロワーは、パーティションリーダーのパーティションデータをレプリケーションし、オプションでコンシューマー要求を処理します。
トピックでは、**ReplicationFactor** (レプリケーション係数) を使用して、クラスター内のパーティションごとのレプリカ数を設定します。トピックは、最低でもパーティション 1 つで設定されます。

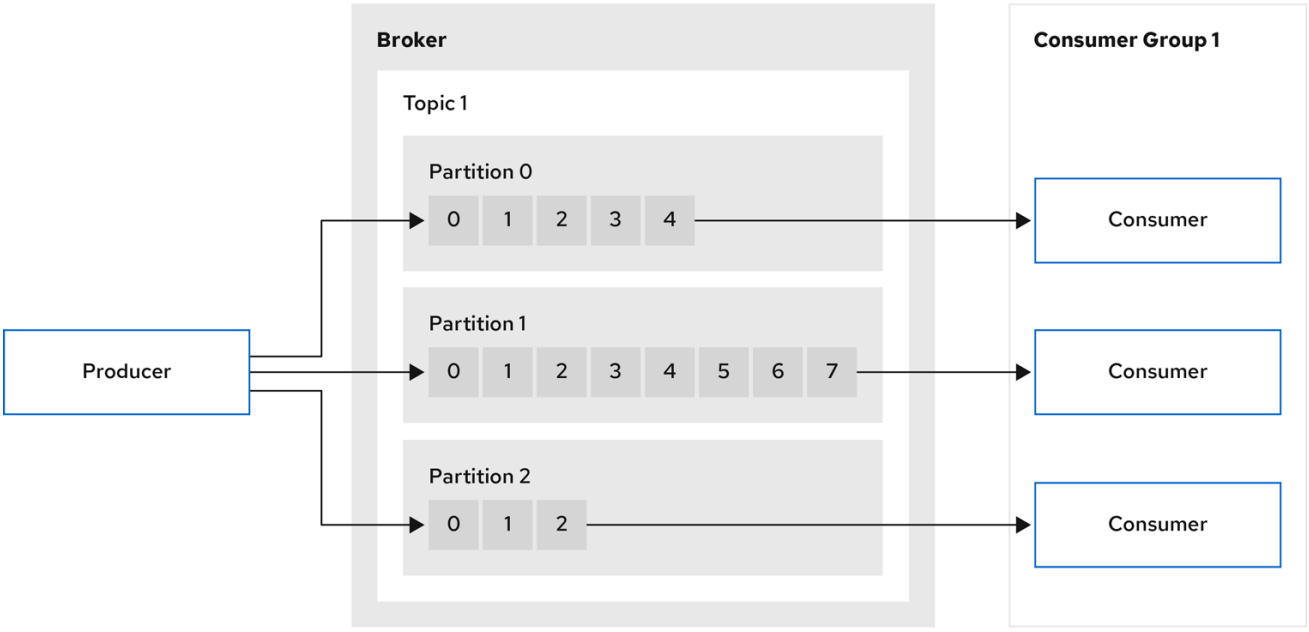
**in-sync** レプリカ (ISR) はリーダーと同数のメッセージを保持しています。設定では、メッセージを生成できるように **同期する** 必要のあるレプリカ数を定義し、メッセージがレプリカパーティションに正常にコピーされない限りに、コミットされないようにします。こうすることで、リーダーに障害が発生しても、メッセージは失われません。

**Kafka ブローカーおよびトピック** の図では、レプリケーションされたトピック内に、各番号付きのパーティションにリーダー 1 つとフォロワーが 2 つあることが分かります。

## 3.2. プロデューサーおよびコンシューマー

プロデューサーおよびコンシューマーは、ブローカー経由でメッセージ (パブリッシュしてサブスクライブ) を送受信します。メッセージは、**キー** (オプション) と、メッセージデータ、ヘッダー、および関連するメタデータが含まれる **値** で設定されます。キーは、メッセージの件名またはメッセージのプロパティーの特定に使用されます。メッセージはバッチで配信されます。バッチやレコードには、レコードのタイムスタンプやオフセットの位置など、クライアントのフィルタリングやルーティングに役立つ情報を提供するヘッダーとメタデータが含まれます。

**プロデューサーおよびコンシューマー**




**プロデューサー**

プロデューサーは、メッセージをブローカートピックに送信し、パーティションの終端オフセットに書き込みます。メッセージは、ラウンドロビンベースでプロデューサーにより複数のパーティションに書き込まれるか、メッセージキーベースで特定のパーティションに書き込まれます。

**コンシューマー**

コンシューマーはトピックをサブスクライブし、トピック、パーティション、およびオフセットをもとにメッセージを読み取ります。

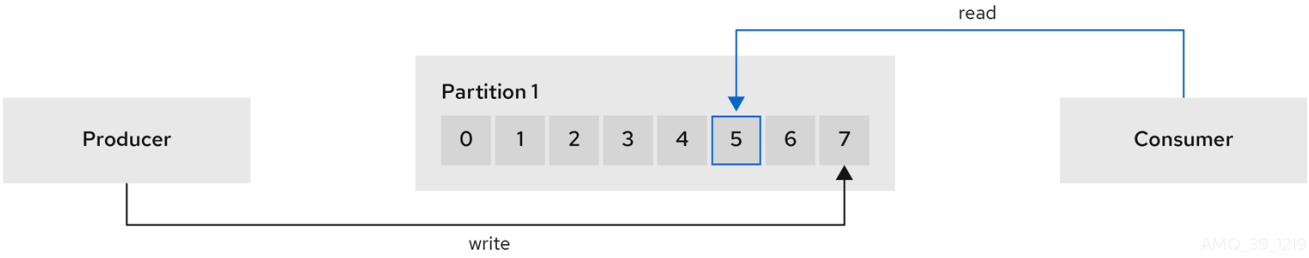
**コンシューマーグループ**

コンシューマーグループは通常、特定のトピックから複数のプロデューサーで生成される大規模なデータストリームを共有するために使用されます。コンシューマーは **group.id** でグループ化され、メッセージをメンバー全体に分散できます。グループ内のコンシューマーは、同じパーティションからのデータは読み取りませんが、1 つ以上のパーティションからデータを受信できます。

**オフセット**

オフセットは、パーティション内のメッセージの位置を表します。特定のパーティションの各メッセージには一意のオフセットがあり、パーティション内のコンシューマーの位置を特定して、消費したレコード数を追跡するのに役立ちます。
コミットされたオフセットは、オフセットコミットログに書き込まれます。**_consumer_offsets** トピックには、コンシューマーグループをもとに、コミットされたオフセット、最後のオフセットと次のオフセットの位置に関する情報が保存されます。

### データの生成および使用



# 第4章 KAFKA CONNECT について

Kafka Connect は、Kafka ブローカーと他のシステムの間でデータをストリーミングする統合ツールです。もう 1 つのシステムは通常、データベースなどの外部データソースまたはターゲットです。

Kafka Connect はプラグインアーキテクチャーを使用しています。プラグインは他のシステムへの接続を可能にし、データを操作するための追加の設定を提供します。プラグインには、**コネクター** や、データコンバーターや変換などの他のコンポーネントが含まれます。コネクターは、特定のタイプの外部システムで動作します。各コネクターは、その設定のスキーマを定義します。設定を Kafka Connect に指定して、Kafka Connect 内に **コネクターインスタンス** を作成します。次に、コネクターインスタンスは、システム間でデータを移動するための一連のタスクを定義します。

AMQ Streams は、**分散モード** で Kafka Connect を操作し、データストリーミングタスクを 1 つ以上のワーカー Pod に分散します。Kafka Connect クラスターは、ワーカー Pod のグループで設定されます。各コネクターは、1 つのワーカーでインスタンス化されます。各コネクターは、ワーカーのグループ全体に分散される 1 つ以上のタスクで設定されます。ワーカー間での分散により、拡張性の高いパイプラインが可能になります。

ワーカーは、データをある形式からソースシステムまたはターゲットシステムに適した別の形式に変換します。コネクターインスタンスの設定によっては、ワーカーが変換 (単一メッセージ変換 (SMT) とも呼ばれます) を適用する場合もあります。変換は、コンバージョンされる前に、特定のデータのフィルタリングなど、メッセージを調整します。Kafka Connect にはいくつかの組み込みの変換がありますが、必要に応じてプラグインによって他の変換を提供できます。

## 4.1. KAFKA CONNECT でのデータのストリーミング方法

Kafka Connect は、コネクターインスタンスを使用して他のシステムと統合し、データをストリーミングします。

Kafka Connect は、起動時に既存のコネクターインスタンスをロードし、データストリーミングタスクとコネクター設定をワーカー Pod 全体に分散します。ワーカーは、コネクターインスタンスのタスクを実行します。各ワーカーは個別の Pod として実行され、Kafka Connect クラスターの耐障害性を高めます。ワーカーよりも多くのタスクがある場合は、ワーカーには複数のタスクが割り当てられます。ワーカーに障害が発生した場合、そのタスクは Kafka Connect クラスター内のアクティブなワーカーに自動的に割り当てられます。

ストリーミングデータで使用される主な Kafka Connect コンポーネントは次のとおりです。

- タスクを作成するためのコネクター
- データを移動するタスク
- タスクを実行するワーカー
- データを操作するための変換
- データを変換するコンバーター

### 4.1.1. コネクター

コネクターは、次のいずれかのタイプにすることができます。

- データを Kafka にプッシュするソースコネクター
- Kafka からデータを抽出するシンクコネクター

プラグインは、Kafka Connect がコネクターインスタンスを実行するための実装を提供します。コネクターインスタンスは、Kafka との間でデータを転送するために必要なタスクを作成します。Kafka Connect ランタイムは、必要な作業をワーカー Pod 間で分割するタスクを調整します。

MirrorMaker 2.0 は、Kafka Connect フレームワークも使用します。この場合、外部データシステムは別の Kafka クラスターです。MirrorMaker 2.0 専用のコネクターは、ソースとターゲットの Kafka クラスター間のデータレプリケーションを管理します。

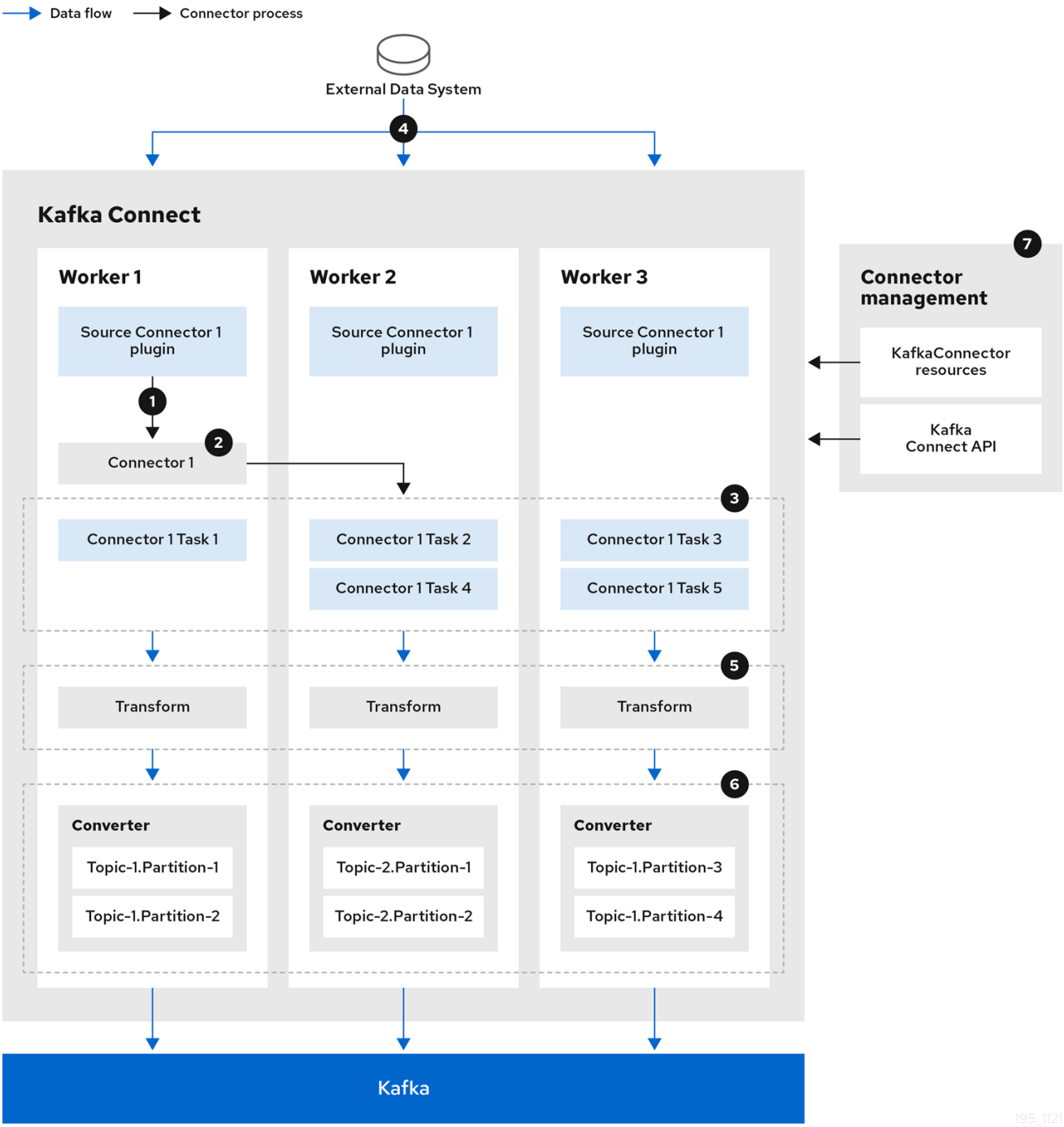
**注記**

MirrorMaker 2.0 コネクターに加えて、Kafka は例として 2 つのコネクターを提供します。

- **File Stream Source Connector** は、ワーカーのファイルシステム上のファイルから Kafka にデータをストリーミングし、入力ファイルを読み取り、各行を特定の Kafka トピックに送信します。
- **FileStreamSinkConnector** は、Kafka からワーカーのファイルシステムにデータをストリーミングし、Kafka トピックからメッセージを読み取り、出力ファイルにそれぞれの行を書き込みます。

次のソースコネクターの図は、外部データシステムからレコードをストリーミングするソースコネクターのプロセスフローを示しています。Kafka Connect クラスターは、ソースコネクターとシンクコネクターを同時に操作する場合があります。ワーカーはクラスター内で分散モードで実行されています。ワーカーは、複数のコネクターインスタンスに対して 1 つ以上のタスクを実行できます。

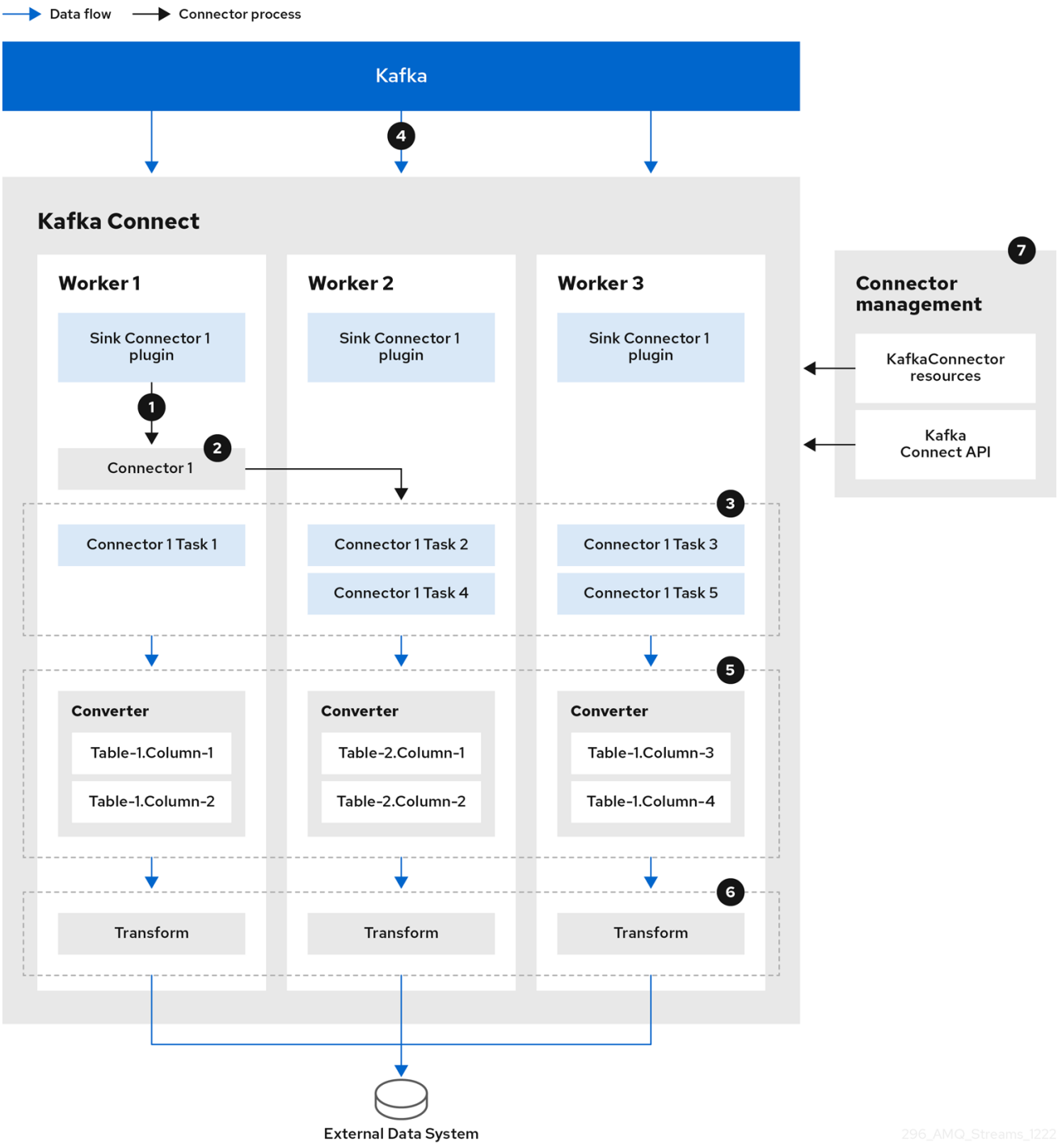
**Kafka へのソースコネクターストリーミングデータ**




1. プラグインがソースコネクターの実装アーティファクトを提供する
2. 1 つのワーカーがソースコネクターインスタンスを開始する
3. ソースコネクターがデータをストリーミングするタスクを作成する
4. タスクが並行して実行され、外部データシステムをポーリングしてレコードを返す
5. 変換がレコードのフィルタリングや再ラベル付けなど、レコードを調整する
6. コンバーターがレコードを Kafka に適した形式に変換する
7. ソースコネクターが KafkaConnectors または Kafka Connect API を使用して管理される

次のシンクコネクターの図は、Kafka から外部データシステムにデータをストリーミングするときのプロセスフローを示しています。

### Kafka からのシンクコネクターストリーミングデータ




1. プラグインがシンクコネクターの実装アーティファクトを提供する
2. 1 つのワーカーがシンクコネクターインスタンスを開始する
3. シンクコネクターがデータをストリーミングするタスクを作成する
4. タスクが並行して実行され、Kafka をポーリングしてレコードを返す
5. コンバーターがレコードを外部データシステムに適した形式に変換する

6. 変換がレコードのフィルタリングや再ラベル付けなど、レコードを調整する

7. シンクコネクターが KafkaConnectors または Kafka Connect API を使用して管理される

### 4.1.2. タスク

Kafka Connect ランタイムによって調整されたデータ転送は、並行して実行されるタスクに分割されます。タスクは、コネクターインスタンスによって提供される設定を使用して開始されます。Kafka Connect は、タスク設定をワーカーに配布します。ワーカーは、タスクをインスタンス化して実行します。

- ソースコネクタータスクは外部データシステムをポーリングし、ワーカーが Kafka ブローカーに送信するレコードのリストを返します。
- シンクコネクタータスクは、外部データシステムに書き込むためにワーカーから Kafka レコードを受信します。

シンクコネクターの場合、作成されるタスクの数は、消費されるパーティションの数に関連します。ソースコネクターの場合、ソースデータの分割方法はコネクターによって定義されます。コネクター設定で **tasksMax** を設定することにより、並行して実行できるタスクの最大数を制御できます。コネクターが作成するタスク数は、最大設定よりも少ないなる可能性があります。たとえば、ソースデータをその数のパーティションに分割できない場合、コネクターは作成するタスクが少なくなる可能性があります。

**注記**

Kafka Connect のコンテキストでは、**partition** (パーティション) は、外部システムのトピックパーティションまたは **shard of data** (データのシャード) を意味する場合があります。

### 4.1.3. ワーカー

ワーカーは、Kafka Connect クラスターにデプロイされたコネクター設定を採用します。設定は、Kafka Connect によって使用される内部 Kafka トピックに保存されます。ワーカーは、コネクターとそのタスクも実行します。

Kafka Connect クラスターには、同じ **group.id** を持つワーカーのグループが含まれています。ID は、Kafka 内のクラスターを識別します。ID は、**KafkaConnect** リソースを介してワーカー設定で割り当てられます。ワーカー設定では、Kafka Connect の内部トピックの名前も指定されます。トピックには、コネクター設定、オフセット、およびステータス情報が格納されます。これらのトピックのグループ ID と名前も、Kafka Connect クラスターに固有である必要があります。

ワーカーには、1 つ以上のコネクターインスタンスとタスクが割り当てられます。Kafka Connect をデプロイするための分散型アプローチは、フォールトトレラントでスケーラブルです。ワーカー Pod に障害が発生した場合、実行中のタスクはアクティブなワーカーに再割り当てされます。**Kafka Connect** リソースの **replicas** プロパティーを設定することで、ワーカー Pod のグループに追加できます。

### 4.1.4. 変換

Kafka Connect は、外部データを変換します。シングルメッセージは、変更メッセージをターゲットの宛先に適した形式に変換します。たとえば、変換によってフィールドが挿入または名前変更される場合があります。変換では、データをフィルタリングしてルーティングすることもできます。プラグインには、ワーカーが 1 つ以上の変換を実行するために必要な実装が含まれています。

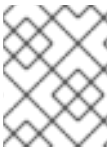
- ソースコネクターは、データを Kafka でサポートされている形式に変換する前に変換を適用します。
- シンクコネクターは、データを外部データシステムに適した形式に変換した後に変換を適用します。

変換は、コネクタープラグインに含めるために JAR ファイルにパッケージ化された Java クラスファイルのセットで設定されます。Kafka Connect は一連の標準変換を提供しますが、独自の変換を作成することもできます。

### 4.1.5. コンバーター

ワーカーはデータを受信すると、コンバーターを使用してデータを適切な形式に変換します。**KafkaConnect** リソースのワーカー **config** でワーカーのコンバーターを指定します。

Kafka Connect は、JSON や Avro などの Kafka でサポートされている形式との間でデータを変換できます。また、データを構造化するためのスキーマもサポートしています。データを構造化形式に変換しない場合は、スキーマを有効にする必要はありません。



**注記**

特定のコネクターのコンバーターを指定して、全ワーカーに該当する一般的な Kafka Connect ワーカー設定をオーバーライドすることもできます。

**関連情報**

- Apache Kafka のドキュメント
- ワーカーの Kafka Connect 設定
- MirrorMaker 2.0 を使用した Kafka クラスター間でのデータの同期

# 第5章 KAFKA BRIDGE インターフェイス

Kafka Bridge では、HTTP ベースのクライアントと Kafka クラスターとの対話を可能にする RESTful インターフェイスが提供されます。 また、クライアントアプリケーションが Kafka プロトコルを変換する必要なく、AMQ Streams で Web API コネクションの利点を活用できます。

API には **consumers** と **topics** の 2 つの主なリソースがあります。これらのリソースは、Kafka クラスターでコンシューマーおよびプロデューサーと対話するためにエンドポイント経由で公開され、アクセスが可能になります。リソースと関係があるのは Kafka ブリッジのみで、Kafka に直接接続されたコンシューマーやプロデューサーとは関係はありません。

## 5.1. HTTP 要求

Kafka Bridge は、以下の方法で Kafka クラスターへの HTTP 要求をサポートします。

- トピックにメッセージを送信する。
- トピックからメッセージを取得する。
- トピックのパーティションリストを取得する。
- コンシューマーを作成および削除する。
- コンシューマーをトピックにサブスクライブし、このようなトピックからメッセージを受信できるようにする。
- コンシューマーがサブスクライブしているトピックの一覧を取得する。
- トピックからコンシューマーのサブスクライブを解除する。
- パーティションをコンシューマーに割り当てる。
- コンシューマーオフセットの一覧をコミットする。
- パーティションで検索して、コンシューマーが最初または最後のオフセットの位置、または指定のオフセットの位置からメッセージを受信できるようにする。

上記の方法で、JSON 応答と HTTP 応答コードのエラー処理を行います。メッセージは JSON またはバイナリー形式で送信できます。

クライアントは、ネイティブの Kafka プロトコルを使用する必要なくメッセージを生成して使用できます。

**関連情報**

- 要求および応答の例など、API ドキュメントを確認するには、AMQ Streams Kafka Bridge の使用を参照してください。

## 5.2. KAFKA BRIDGE でサポートされるクライアント

Kafka Bridge を使用して、**内部**および**外部**の HTTP クライアントアプリケーションの両方を Kafka クラスターに統合できます。

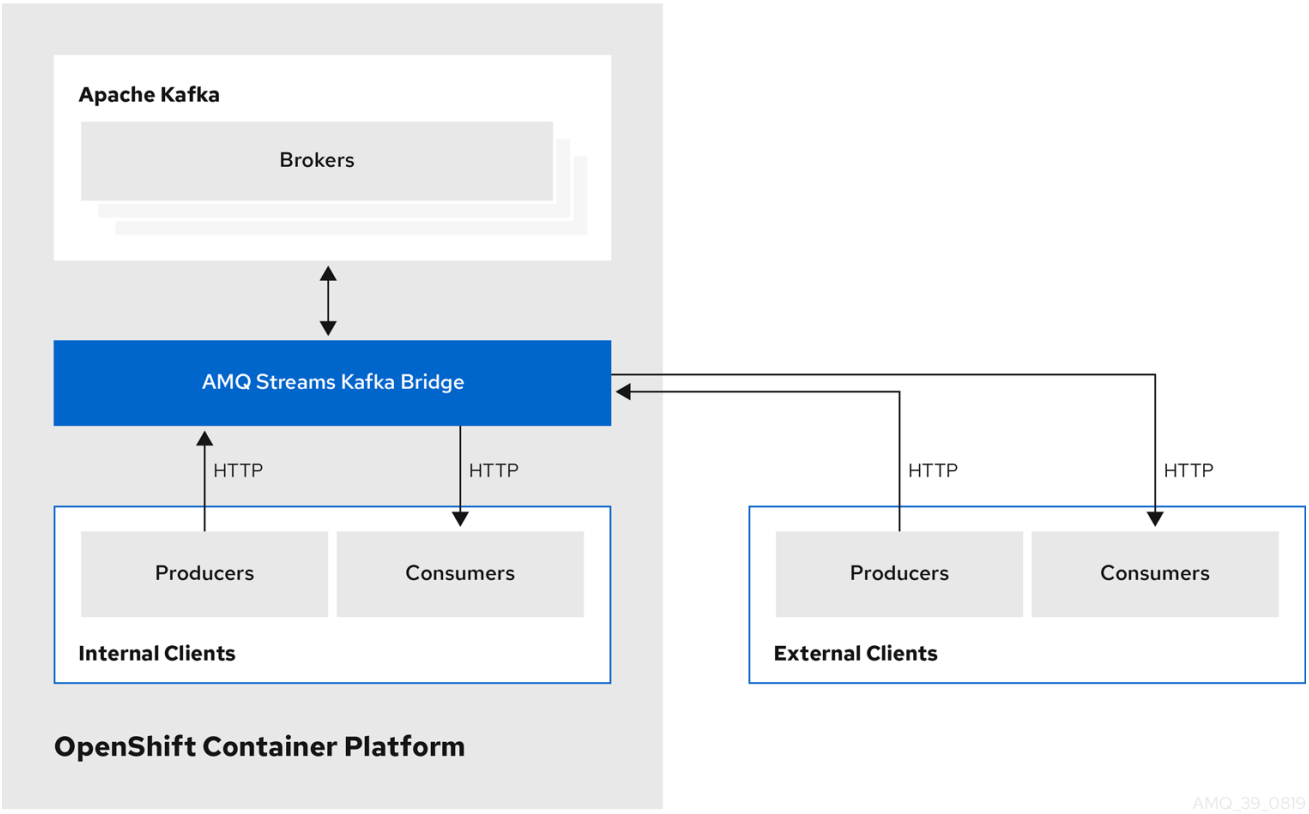
**内部クライアント**

内部クライアントとは、Kafka Bridge 自体と**同じ** OpenShift クラスターで実行されるコンテナーベースの HTTP クライアントのことです。内部クライアントは、ホストの Kafka Bridge および **KafkaBridge** のカスタムリソースで定義されたポートにアクセスできます。

**外部クライアント**

外部クライアントとは、Kafka Bridge がデプロイおよび実行される OpenShift クラスター**外部**で実行される HTTP クライアントのことです。外部クライアントは、OpenShift Route、ロードバランサーサービス、または Ingress を使用して Kafka Bridge にアクセスできます。

### HTTP 内部および外部クライアントの統合



# 第6章 AMQ STREAMS の OPERATOR

AMQ Streams では **Operator** を使用して Kafka をサポートし、Kafka のコンポーネントおよび依存関係を OpenShift にデプロイして管理します。

Operator は、OpenShift アプリケーションのパッケージ化、デプロイメント、および管理を行う方法です。AMQ Streams Operator は OpenShift の機能を拡張し、Kafka デプロイメントに関連する共通タスクや複雑なタスクを自動化します。Kafka 操作の情報をコードに実装することで、Kafka の管理タスクは簡素化され、人の介入が少なくなります。

## Operator

AMQ Streams は、OpenShift クラスター内で実行中の Kafka クラスターを管理するための Operator を提供します。

**Cluster Operator**

Apache Kafka クラスター、Kafka Connect、Kafka MirrorMaker、Kafka Bridge、Kafka Exporter、Cruise Control、および Entity Operator をデプロイおよび管理します。

**Entitiy Operator**

Topic Operator および User Operator を設定します。

**Topic Operator**

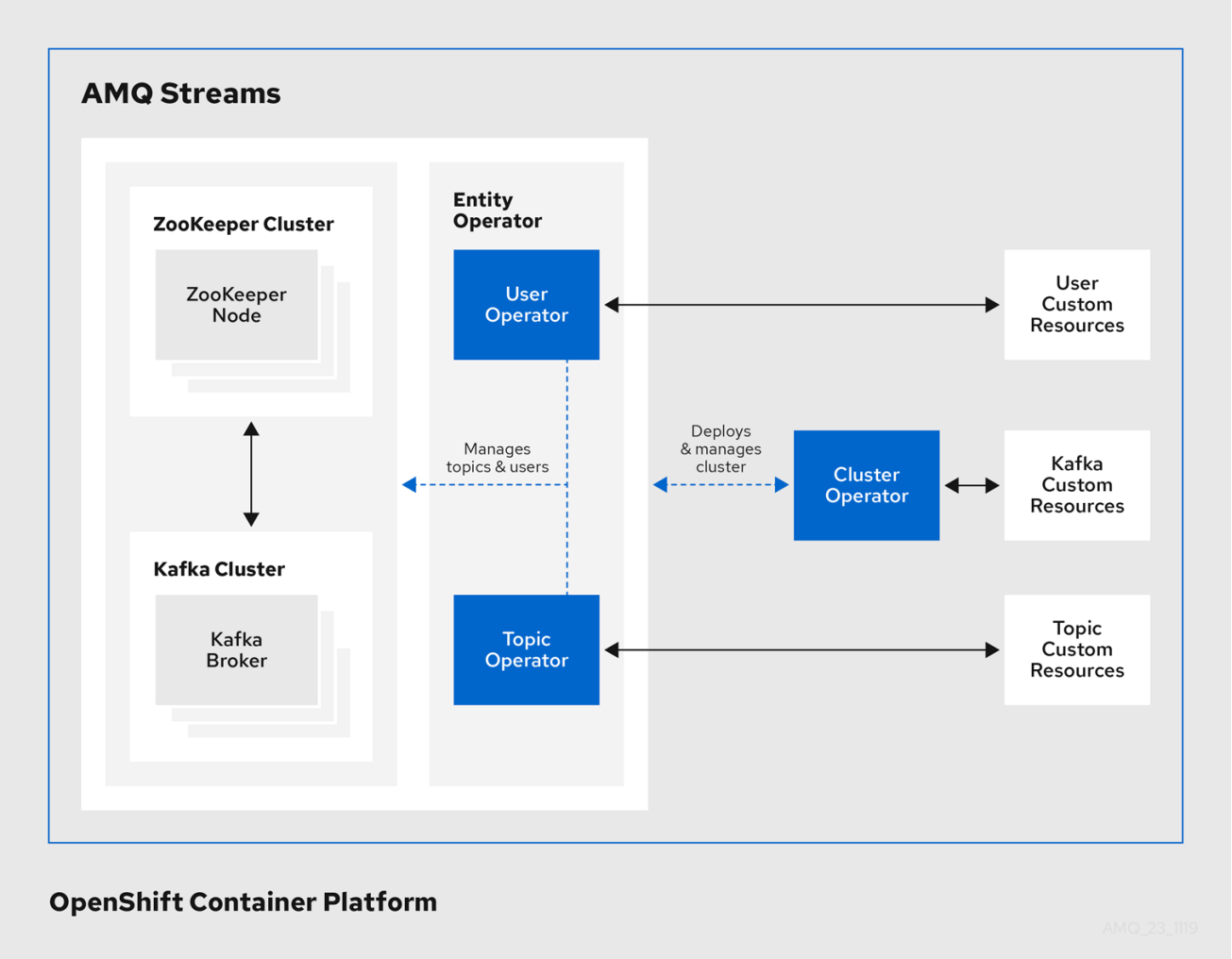
Kafka トピックを管理します。

**User Operator**

Kafka ユーザーを管理します。

Cluster Operator は、Kafka クラスターと同時に、Topic Operator および User Operator を **Entity Operator** 設定の一部としてデプロイできます。

### AMQ Streams アーキテクチャー内の Operator




## 6.1. CLUSTER OPERATOR

AMQ Streams では、Cluster Operator を使用してクラスターをデプロイおよび管理します。デフォルトでは、AMQ Streams をデプロイすると、単一の Cluster Operator レプリカがデプロイされます。リーダーの選択でレプリカを追加し、中断が発生した場合に追加の Cluster Operator がスタンバイ状態になるようにすることができます。

Cluster Operator は、以下の Kafka コンポーネントのクラスターを管理します。

- Kafka (ZooKeeper、Entity Operator、Kafka Exporter、Cruise Control を含む)
- Kafka Connect
- Kafka MirrorMaker
- Kafka Bridge

クラスターは、カスタムリソースを使用してデプロイされます。

たとえば、以下のように Kafka クラスターをデプロイします。

- クラスター設定のある **Kafka** リソースが OpenShift クラスター内で作成されます。
- **Kafka** リソースに宣言された内容を基にして、該当する Kafka クラスターが Cluster Operator によってデプロイされます。

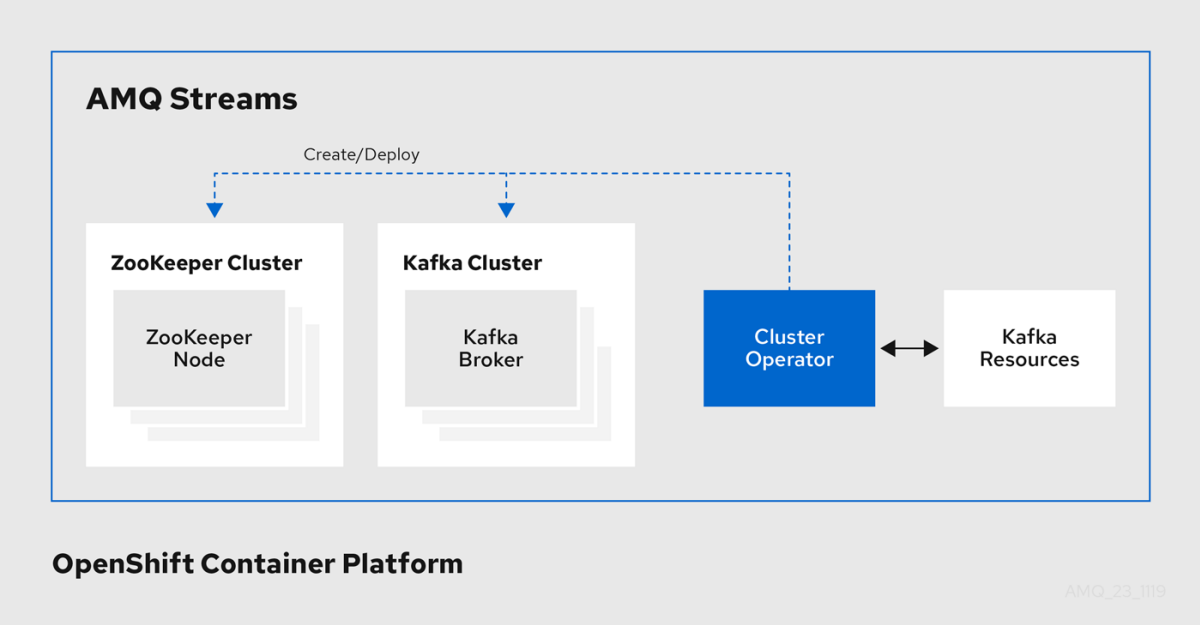
Cluster Operator は、**Kafka** リソースの設定を通じて、以下の AMQ StreamsOperator をデプロイすることもできます。

- **KafkaTopic** カスタムリソースを介して Operator スタイルのトピック管理を提供する Topic Operator
- **KafkaUser** カスタムリソースを介して Operator スタイルのユーザー管理を提供する User Operator

Topic Operator および User Operator は、デプロイメントの Entity Operator 内で機能します。

AMQ Streams Drain Cleaner のデプロイメントで Cluster Operator を使用すると、Pod のエビクションに役立ちます。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 は、エビクトされる Pod に、ローリング更新のアノテーションを付けます。このアノテーションは、Cluster Operator にローリング更新を実行するように通知します。

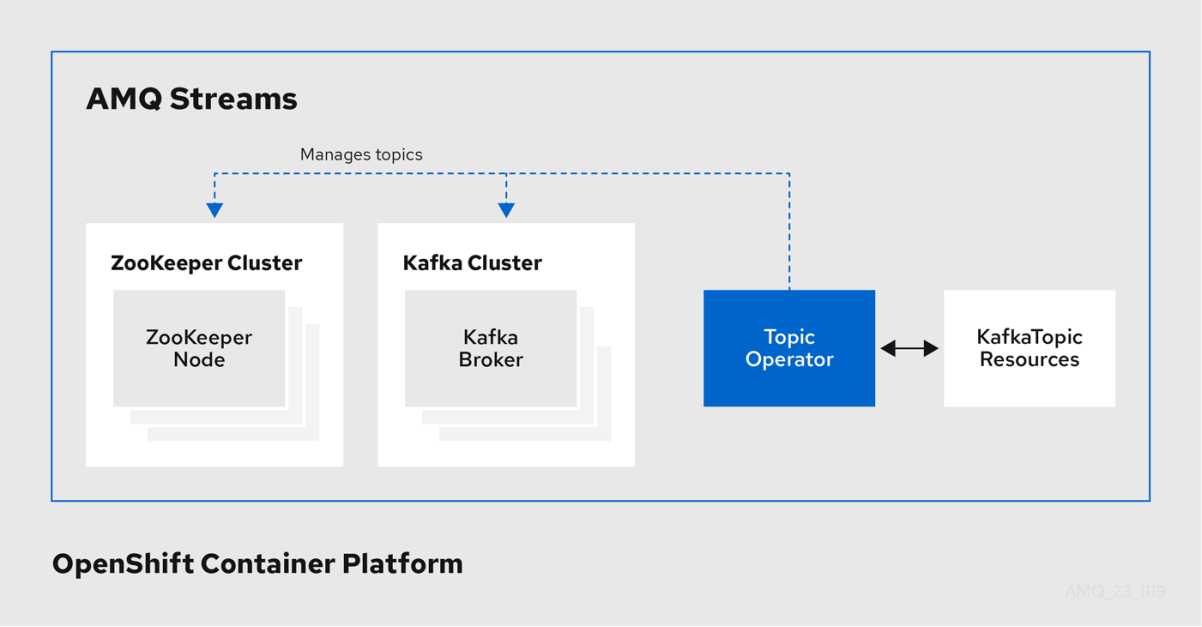
### Cluster Operator のアーキテクチャー例




## 6.2. TOPIC OPERATOR

Topic Operator は、OpenShift リソースより Kafka クラスターのトピックを管理する方法を提供します。

### Topic Operator のアーキテクチャー例




Topic Operator のロールは、対応する Kafka トピックと同期して Kafka トピックを記述する **KafkaTopic** OpenShift リソースのセットを保持することです。

特に、**KafkaTopic** が

- 作成されると、Topic Operator によってトピックが作成されます。
- 削除されると、Topic Operator によってトピックが削除されます。
- 変更されると、Topick Operator によってトピックが更新されます。

上記と逆の方向で、トピックが

- Kafka クラスター内で作成されると、Operator によって **KafkaTopic** が作成されます。
- Kafka クラスターから削除されると、Operator によって **KafkaTopic** が削除されます。
- Kafka クラスターで変更されると、Operator によって **KafkaTopic** が更新されます。

このため、**KafkaTopic** をアプリケーションのデプロイメントの一部として宣言でき、トピックの作成は Topic Operator によって行われます。アプリケーションは、必要なトピックからの作成または消費のみに対処する必要があります。

Topic Operator は、各トピックの情報を **トピックストア** で維持します。トピックストアは、Kafka トピックまたは OpenShift **KafkaTopic** カスタムリソースからの更新と継続的に同期されます。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。トピックが再設定されたり、別のブローカーに再割り当てされた場合、**KafkaTopic** は常に最新の状態になります。

## 6.3. USER OPERATOR

User Operator は、Kafka ユーザーが記述される **KafkaUser** リソースを監視して Kafka クラスターの Kafka ユーザーを管理し、Kafka ユーザーが Kafka クラスターで適切に設定されるようにします。

たとえば、**KafkaUser** が

- 作成されると、User Operator によって記述されるユーザーが作成されます。

- 削除されると、User Operator によって記述されるユーザーが削除されます。
- 変更されると、User Operator によって記述されるユーザーが更新されます。

User Operator は Topic Operator とは異なり、Kafka クラスターからの変更は OpenShift リソースと同期されません。アプリケーションで直接 Kafka トピックを Kafka で作成することは可能ですが、ユーザーが User Operator と同時に直接 Kafka クラスターで管理されることは想定されません。

User Operator では、アプリケーションのデプロイメントの一部として **KafkaUser** リソースを宣言できます。ユーザーの認証および承認メカニズムを指定できます。たとえば、ユーザーがブローカーへのアクセスを独占しないようにするため、Kafka リソースの使用を制御する **ユーザークォータ** を設定することもできます。

ユーザーが作成されると、ユーザークレデンシャルが **Secret** に作成されます。アプリケーションはユーザーとそのクレデンシャルを使用して、認証やメッセージの生成または消費を行う必要があります。

User Operator は 認証のクレデンシャルを管理する他に、**KafkaUser** 宣言にユーザーのアクセス権限の記述を含めることで承認も管理します。

## 6.4. AMQ STREAMS OPERATOR のフィーチャーゲート

**フィーチャーゲート**を使用して、operator の一部の機能を有効または無効にすることができます。

フィーチャーゲートは Operator の設定で指定され、alpha、beta、または General Availability (GA) の 3 段階の成熟度があります。

詳細はフィーチャーゲートを参照してください。

# 第7章 KAFKA の設定

AMQ Streams を使用した Kafka コンポーネントの OpenShift クラスターへのデプロイメントは、カスタムリソースの適用により高度な設定が可能です。カスタムリソースは、OpenShift リソースを拡張するために CRD (カスタムリソース定義、Custom Resource Definition) によって追加される API のインスタンスとして作成されます。

CRD は、OpenShift クラスターでカスタムリソースを記述するための設定手順として機能し、デプロイメントで使用する Kafka コンポーネントごとに AMQ Streams で提供されます。CRD およびカスタムリソースは YAML ファイルとして定義されます。YAML ファイルのサンプルは AMQ Streams ディストリビューションに同梱されています。

また、CRD を使用すると、CLI へのアクセスや設定検証などのネイティブ OpenShift 機能を AMQ Streams リソースで活用することもできます。

本項では、カスタムリソースを使用して Kafka のコンポーネントを設定する方法を見ていきます。まず、一般的な設定のポイント、次にコンポーネント固有の重要な設定に関する考慮事項について説明します。

AMQ Streams には、設定ファイルの例 が含まれており、デプロイメント用の独自の Kafka コンポーネント設定を構築する際の開始点として役立ちます。

## 7.1. カスタムリソース

CRD をインストールして新規カスタムリソースタイプをクラスターに追加した後に、その仕様に基づいてリソースのインスタンスを作成できます。

AMQ Streams コンポーネントのカスタムリソースには、**spec** で定義される共通の設定プロパティーがあります。

Kafka トピックカスタムリソースからのこの抜粋では、**apiVersion** および **kind** プロパティーを使用して、関連付けられた CRD を識別します。**Spec** プロパティーは、トピックのパーティションおよびレプリカの数を定義する設定を示しています。

### Kafka トピックカスタムリソース

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaTopic
metadata:
  name: my-topic
  labels:
    strimzi.io/cluster: my-cluster
spec:
  partitions: 1
  replicas: 1
  # ...
```

共通の設定、特定のコンポーネントに特有の設定など、他にも YAML 定義に組み込むことができる設定オプションが多数あります。

#### 関連情報

- Extend the Kubernetes API with CustomResourceDefinitions

## 7.2. 共通の設定

複数のリソースに共通する設定オプションの一部が以下に記載されています。セキュリティー および メトリクスコレクション も採用できます (該当する場合)。

**ブートストラップサーバー**

ブートストラップサーバーは、以下の Kafka クラスターに対するホスト/ポート接続に使用されます。

- Kafka Connect
- Kafka Bridge
- Kafka MirrorMaker プロデューサーおよびコンシューマー

**CPU およびメモリーリソース**

コンポーネントの CPU およびメモリーリソースを要求します。制限は、特定のコンテナーが消費できる最大リソースを指定します。
Topic Operator および User Operator のリソース要求および制限は **Kafka** リソースに設定されます。

**ロギング**

コンポーネントのロギングレベルを定義します。ロギングは直接 (インライン) または外部で Config Map を使用して定義できます。

**ヘルスチェック**

ヘルスチェックの設定では、**liveness** および **readiness** プローブが導入され、コンテナーを再起動するタイミング (liveliness) と、コンテナーがトラフィック (readiness) を受け入れるタイミングが分かります。

**JVM オプション**

JVM オプションでは、メモリー割り当ての最大と最小を指定し、実行するプラットフォームに応じてコンポーネントのパフォーマンスを最適化します。

**Pod のスケジューリング**

Pod スケジュールは **アフィニティー/非アフィニティールール** を使用して、どのような状況で Pod がノードにスケジューリングされるかを決定します。

### 共通設定の YAML 例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
  name: my-cluster
spec:
  # ...
  bootstrapServers: my-cluster-kafka-bootstrap:9092
  resources:
    requests:
      cpu: 12
      memory: 64Gi
    limits:
      cpu: 12
      memory: 64Gi
  logging:
```

```
    type: inline
    loggers:
      connect.root.logger.level: "INFO"
  readinessProbe:
    initialDelaySeconds: 15
    timeoutSeconds: 5
  livenessProbe:
    initialDelaySeconds: 15
    timeoutSeconds: 5
  jvmOptions:
    "-Xmx": "2g"
    "-Xms": "2g"
  template:
    pod:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
              - matchExpressions:
                  - key: node-type
                    operator: In
                    values:
                      - fast-network
  # ...
```

## 7.3. KAFKA クラスターの設定

Kafka クラスターは、1 つまたは複数のブローカーで設定されます。プロデューサーおよびコンシューマーがブローカー内のトピックにアクセスできるようにするには、Kafka 設定でクラスターへのデータの保存方法、およびデータへのアクセス方法を定義する必要があります。ラック全体で複数のブローカーノードを使用して Kafka クラスターを実行するように設定できます。

**ストレージ**

Kafka および ZooKeeper は、ディスクにデータを格納します。
AMQ Streams は、**StorageClass** でプロビジョニングされるブロックストレージが必要です。ストレージ用のファイルシステム形式は **XFS** または **EXT4** である必要があります。3 種類のデータストレージがサポートされます。

**一時データストレージ (開発用のみで推奨されます)**

一時ストレージは、インスタンスの有効期間についてのデータを格納します。インスタンスを再起動すると、データは失われます。

**永続ストレージ**

永続ストレージは、インスタンスのライフサイクルとは関係なく長期のデータストレージに関連付けられます。

**JBOD (Just a Bunch of Disks、Kafka のみに適しています)**

JBOD では、複数のディスクを使用して各ブローカーにコミットログを保存できます。

既存の Kafka クラスターが使用するディスク容量は、増やすことができます (インフラストラクチャーでサポートされる場合)。

**リスナー**

リスナーは、クライアントが Kafka クラスターに接続する方法を設定します。

Kafka クラスター内の各リスナーに一意の名前とポートを指定することで、複数のリスナーを設定できます。

以下のタイプのリスナーがサポートされます。

- OpenShift 内でのアクセスに使用する **内部リスナー**
- OpenShift 外からアクセスするときに使用する **外部リスナー**

リスナーの TLS 暗号化を有効にし、認証 を設定できます。

内部リスナーは、**internal** 型を指定して Kafka を公開します。

- 同じ OpenShift クラスター内で接続する **internal**
- ブローカーごとの **ClusterIP** サービスを使用して Kafka を公開する **cluster-ip**

外部リスナーは、外部用 **type** を指定して Kafka を公開します。

- OpenShift ルートおよびデフォルトの HAProxy ルーターを使用する **route**
- ロードバランサーサービスを使用する **loadbalancer**
- OpenShift ノードのポートを使用する **nodeport**
- OpenShift **Ingress** および Ingress NGINX Controller for Kubernetes を使用する **ingress**

**注記**

**cluster-ip** タイプを使用すると、独自のアクセスメカニズムを追加できます。たとえば、カスタム Ingress コントローラーまたは OpenShift Gateway API でリスナーを使用できます。

トークンベースの認証に OAuth 2.0 を使用している場合は、リスナーが承認サーバーを使用するように設定できます。

**ラックアウェアネス**

ラックは、データセンター、データセンター内のラック、または可用性ゾーンを表します。Kafka ブローカー Pod とトピックレプリカをラック全体に分散するようにラック認識を設定します。**rack** プロパティーを使用してラック認識を有効にし、**topologyKey** を指定します。**topologyKey** は、ラックを識別する OpenShift ワーカーノードに割り当てられたラベルの名前です。AMQ Streams は、各 Kafka ブローカーにラック ID を割り当てます。Kafka ブローカーは ID を使用して、パーティションのレプリカをラック全体に分散させます。ラック認識で使用する **RackAwareReplicaSelector** セレクタープラグインを指定することもできます。プラグインはブローカーとコンシューマーのラック ID を照合するため、メッセージは最も近いレプリカから消費されます。プラグインを使用するには、コンシューマーもラック認識を有効にする必要があります。Kafka Connect、MirrorMaker 2.0、および Kafka Bridge でラック認識を有効にできます。

### Kafka 設定の YAML 例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
```

```
kafka:
  # ...
  listeners:
    - name: tls
      port: 9093
      type: internal
      tls: true
      authentication:
        type: tls
    - name: external1
      port: 9094
      type: route
      tls: true
      authentication:
        type: tls
  # ...
  storage:
    type: persistent-claim
    size: 10000Gi
  # ...
  rack:
    topologyKey: topology.kubernetes.io/zone
  config:
    replica.selector.class: org.apache.kafka.common.replica.RackAwareReplicaSelector
  # ...
```

## 7.4. KAFKA MIRRORMAKER の設定

MirrorMaker を設定するには、ソースおよびターゲット (宛先) の Kafka クラスターが実行中である必要があります。

従来のバージョンの MirrorMaker のサポートも継続されますが、AMQ Streams で MirrorMaker 2.0 を使用することもできます。

### 7.4.1. MirrorMaker 2.0 の設定

MirrorMaker 2.0 はソースの Kafka クラスターからメッセージを消費して、ターゲットの Kafka クラスターに書き込みます。

MirrorMaker 2.0 は以下を使用します。

- ソースクラスターからデータを消費するソースクラスターの設定
- データをターゲットクラスターに出力するターゲットクラスターの設定

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。

MirrorMaker 2.0 を設定して、ソースクラスターとターゲットクラスターの接続の詳細を含む Kafka Connect のデプロイメントを定義し、MirrorMaker 2.0 コネクターのセットを実行して接続を確立します。

MirrorMaker 2.0 は、以下のコネクターで設定されます。

**MirrorSourceConnector**

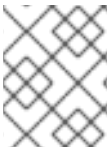
ソースコネクターは、トピックをソースクラスターからターゲットクラスターにレプリケートします。また、ACL をレプリケートし、**MirrorCheckpointConnector** を実行する必要があります。

**MirrorCheckpointConnector**

チェックポイントコネクターは定期的にオフセットを追跡します。有効にすると、ソースクラスターとターゲットクラスター間のコンシューマーグループオフセットも同期されます。

**MirrorHeartbeatConnector**

ハートビートコネクターは、ソースクラスターとターゲットクラスター間の接続を定期的にチェックします。

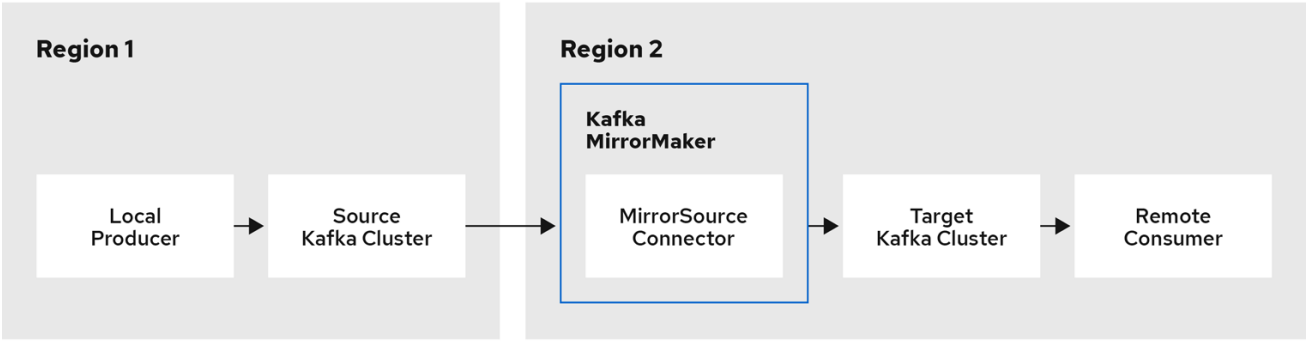


**注記**

User Operator を使用して ACL を管理する場合、コネクターを介した ACL レプリケーションはできません。

ソースクラスターからターゲットクラスターへのデータの **ミラーリング** プロセスは非同期です。各 MirrorMaker 2.0 インスタンスは、1 つのソースクラスターから 1 つのターゲットクラスターにデータをミラーリングします。複数の MirrorMaker 2.0 インスタンスを使用して、任意の数のクラスター間でデータをミラーリングできます。

**図7.1 2 つのクラスターにおけるレプリケーション**




デフォルトでは、ソースクラスターの新規トピックのチェックは 10 分ごとに行われます。頻度は、**refresh.topics.interval.seconds** をソースコネクター設定に追加することで変更できます。

#### 7.4.1.1. クラスター設定

**active/passive** または **active/active** クラスター設定で MirrorMaker 2.0 を使用できます。

**アクティブ/アクティブのクラスター設定**

アクティブ/アクティブ設定には、双方向でデータをレプリケートするアクティブなクラスターが 2 つあります。アプリケーションはいずれかのクラスターを使用できます。各クラスターは同じデータを提供できます。これにより、地理的に異なる場所で同じデータを利用できるようにします。コンシューマーグループは両方のクラスターでアクティブであるため、レプリケートされたトピックのコンシューマーオフセットはソースクラスターに同期されません。

**アクティブ/パッシブクラスター設定**

アクティブ/パッシブ設定には、パッシブクラスターにデータをレプリケートするアクティブクラスターがあります。パッシブクラスターはスタンバイのままになります。システムに障害が発生した場合に、データ復旧にパッシブクラスターを使用できます。

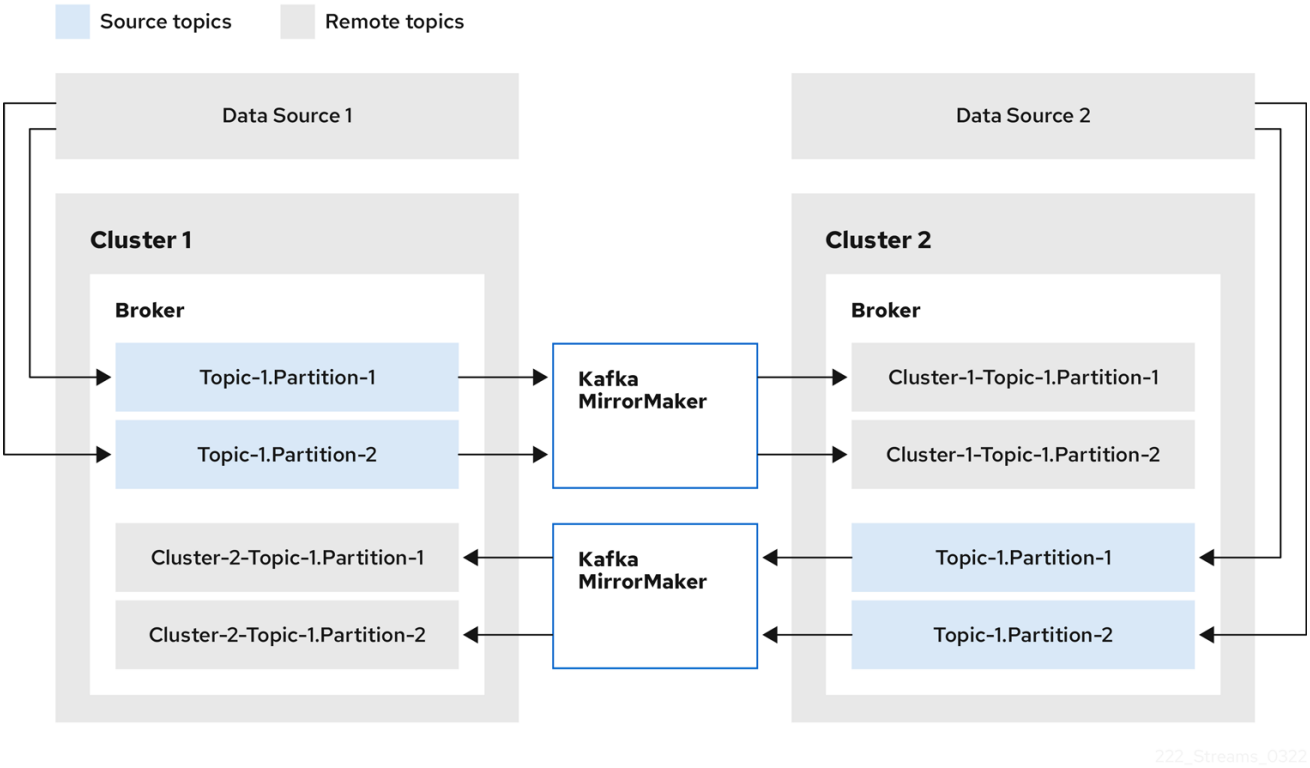
フロデューサーとコンシューマーがアクティフなクラスターのみに接続することを前提とします。MirrorMaker 2.0 クラスターは、ターゲットの宛先ごとに必要です。

### 7.4.1.2. 双方向レプリケーション (active/active)

MirrorMaker 2.0 アーキテクチャーでは、**active/active** クラスター設定で双方向レプリケーションがサポートされます。

各クラスターは、**source** および **remote** トピックの概念を使用して、別のクラスターのデータをレプリケートします。同じトピックが各クラスターに保存されるため、リモートトピックの名前がソースクラスターを表すように自動的に MirrorMaker 2.0 によって変更されます。元のクラスターの名前の先頭には、トピックの名前が追加されます。

**図7.2 トピック名の変更**




ソースクラスターにフラグを付けると、トピックはそのクラスターにレプリケートされません。

**remote** トピックを介したレプリケーションの概念は、データの集約が必要なアーキテクチャーの設定に役立ちます。コンシューマーは、同じクラスター内でソースおよびリモートトピックにサブスクライブできます。これに個別の集約クラスターは必要ありません。

### 7.4.1.3. 一方向レプリケーション (active/passive)

MirrorMaker 2.0 アーキテクチャーでは、**active/passive** クラスター設定で一方向レプリケーションがサポートされます。

**active/passive** のクラスター設定を使用してバックアップを作成したり、データを別のクラスターに移行したりできます。このような場合には、リモートトピックの名前を自動的に変更させないように指定することがあります。

**IdentityReplicationPolicy** をソースコネクター設定に追加することで、名前の自動変更をオーバーライドできます。この設定が適用されると、トピックには元の名前が保持されます。

**MirrorMaker 2.0 設定の YAML の例**

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaMirrorMaker2
metadata:
  name: my-mirror-maker2
  spec:
    version: 3.3.1
    connectCluster: "my-cluster-target"
    clusters:
    - alias: "my-cluster-source"
      bootstrapServers: my-cluster-source-kafka-bootstrap:9092
    - alias: "my-cluster-target"
      bootstrapServers: my-cluster-target-kafka-bootstrap:9092
    mirrors:
    - sourceCluster: "my-cluster-source"
      targetCluster: "my-cluster-target"
      sourceConnector: {}
    topicsPattern: ".*"
    groupsPattern: "group1|group2|group3"
```

### 7.4.2. MirrorMaker の設定

従来のバージョンの MirrorMaker では、プロデューサーとコンシューマーを使用して、クラスターにまたがってデータをレプリケートします。

MirrorMaker は以下を使用します。

- ソースクラスターからデータを使用するコンシューマーの設定。
- データをターゲットクラスターに出力するプロデューサーの設定。

コンシューマーおよびプロデューサー設定には、認証および暗号化設定が含まれます。

**include** フィールドは、ソースからターゲットクラスターにミラーリングするトピックを定義します。

**主なコンシューマー設定**

**コンシューマーグループ ID**

使用するメッセージがコンシューマーグループに割り当てられるようにするための MirrorMaker コンシューマーのコンシューマーグループ ID。

**コンシューマーストリームの数**

メッセージを並行して使用するコンシューマーグループ内のコンシューマー数を決定する値。

**オフセットコミットの間隔**

メッセージの使用とメッセージのコミットの期間を設定するオフセットコミットの間隔。

**キープロデューサーの設定**

**送信失敗のキャンセルオプション**

メッセージ送信の失敗を無視するか、または MirrorMaker を終了して再作成するかを定義できます。

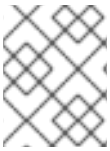
**MirrorMaker 設定の YAML 例**

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaMirrorMaker
metadata:
  name: my-mirror-maker
spec:
  # ...
  consumer:
    bootstrapServers: my-source-cluster-kafka-bootstrap:9092
    groupId: "my-group"
    numStreams: 2
    offsetCommitInterval: 120000
    # ...
  producer:
    # ...
    abortOnSendFailure: false
    # ...
  include: "my-topic|other-topic"
  # ...
```

## 7.5. KAFKA CONNECT の設定

AMQ Streams の **KafkaConnect** リソースを使用して、新しい Kafka Connect クラスターをすばやく簡単に作成します。

**KafkaConnect** リソースを使用して Kafka Connect をデプロイする場合は、Kafka クラスターに接続するためのブートストラップサーバーアドレスを (**spec.bootstrapServers** で) 指定します。サーバーがダウンした場合に備えて、複数のアドレスを指定できます。また、認証情報と TLS 暗号化証明書を指定して、安全な接続を確立します。



**注記**

Kafka クラスターは、AMQ Streams で管理したり、OpenShift クラスターにデプロイしたりする必要はありません。

**KafkaConnect** リソースを使用して、以下を指定することもできます。

- 接続を確立するためのプラグインを含むコンテナーイメージを構築するためのプラグイン設定
- Kafka Connect クラスターに属するワーカー Pod の設定
- **KafkaConnector** リソースを使用してプラグインを管理できるようにするアノテーション

Cluster Operator は、**KafkaConnect** リソースを使用してデプロイされた Kafka Connect クラスターと、**KafkaConnector** リソースを使用して作成されたコネクターを管理します。

**プラグイン設定**
プラグインは、コネクターインスタンスを作成するための実装を提供します。プラグインがインスタンス化されると、特定のタイプの外部データシステムに接続するための設定が提供されます。プラグインは、特定の種類のデータソースに接続するためのコネクターとタスクの実装を定義する 1 つ以上の JAR ファイルのセットを提供します。多くの外部システム用のプラグインは、Kafka Connect で使用できます。独自のプラグインを作成することもできます。

この設定では、Kafka Connect にフィードするソース入力データおよびターゲット出力データを記述します。ソースコネクターの場合、外部ソースデータは、メッセージを格納する特定のトピックを参照する必要があります。プラグインには、データの変換に必要なライブラリーとファイルが含まれている場

合もあります。

Kafka Connect デプロイメントには、1 つ以上のプラグインを含めることができますが、各プラグインのバージョンは 1 つだけです。

選択したプラグインを含むカスタム Kafka Connect イメージを作成できます。イメージは次の 2 つの方法で作成できます。

- Kafka Connect 設定を自動的に使用
- Dockerfile と Kafka コンテナーイメージをベースイメージとして手動で使用

コンテナーイメージを自動的に作成するには、**KafkaConnect** リソースの **build** プロパティーを使用して、Kafka Connect クラスターに追加するプラグインを指定します。AMQ Streams は、プラグインアーティファクトを自動的にダウンロードして新しいコンテナーイメージに追加します。

**プラグイン設定の例**

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
  name: my-connect-cluster
  annotations:
    strimzi.io/use-connector-resources: "true"
spec:
  # ...
  build: 1
    output: 2
      type: docker
      image: my-registry.io/my-org/my-connect-cluster:latest
      pushSecret: my-registry-credentials
    plugins: 3
      - name: debezium-postgres-connector
        artifacts:
          - type: tgz
            url: https://ARTIFACT-ADDRESS.tgz
            sha512sum: HASH-NUMBER-TO-VERIFY-ARTIFACT
      # ...
  # ...
```

1 コネクタープラグインで自動的にコンテナーイメージをビルドするための ビルド設定プロパティー。

2 新しいイメージがプッシュされるコンテナーレジストリーの設定。**output** 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。

3 新しいコンテナーイメージに追加するプラグインとそのアーティファクトのリスト。**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。各プラグインは、1 つ以上のアーティファクトで設定する必要があります。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。

Dockerfile を使用してイメージをビルドしている場合は、AMQ Streams の最新のコンテナーイメージをベースイメージとして使用して、プラグイン設定ファイルを追加できます。

**プラグイン設定の手動追加を示す例**

```
FROM registry.redhat.io/amq7/amq-streams-kafka-33-rhel8:2.3.0
USER root:root
COPY ./my-plugins/ /opt/kafka/plugins/
USER 1001
```

### ワーカー用の Kafka Connect クラスター設定

ワーカーの設定は、**KafkaConnect** リソースの **config** プロパティーで指定します。

分散型 Kafka Connect クラスターには、グループ ID と一連の内部設定トピックがあります。

- **group.id**
- **offset.storage.topic**
- **config.storage.topic**
- **status.storage.topic**

Kafka Connect クラスターは、デフォルトでこれらのプロパティーに同じ値で設定されています。Kafka Connect クラスターは、エラーを作成するため、グループ ID またはトピック名を共有できません。複数の異なる Kafka Connect クラスターを使用する場合、これらの設定は、作成された各 Kafka Connect クラスターのワーカーに対して一意である必要があります。

各 Kafka Connect クラスターで使用されるコネクターの名前も一意である必要があります。

次のワーカー設定の例では、JSON コンバーターが指定されています。レプリケーション係数は、Kafka Connect によって使用される内部 Kafka トピックに設定されます。これは、実稼働環境では少なくとも 3 つ必要です。トピックの作成後にレプリケーション係数を変更しても効果はありません。

**ワーカー設定の例**

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
# ...
spec:
  config:
    # ...
    group.id: my-connect-cluster 1
    offset.storage.topic: my-connect-cluster-offsets 2
    config.storage.topic: my-connect-cluster-configs 3
    status.storage.topic: my-connect-cluster-status 4
    key.converter: org.apache.kafka.connect.json.JsonConverter 5
    value.converter: org.apache.kafka.connect.json.JsonConverter 6
    key.converter.schemas.enable: true 7
    value.converter.schemas.enable: true 8
    config.storage.replication.factor: 3 9
    offset.storage.replication.factor: 3 10
    status.storage.replication.factor: 3 11
  # ...
```

1 Kafka 内の Kafka Connect クラスター ID。Kafka Connect クラスターごとに一意である必要があります。

2 コネクターオフセットを保存する Kafka トピック。Kafka Connect クラスターごとに一意である必要があります。

3 コネクターおよびタスクステータスの設定を保存する Kafka トピック。Kafka Connect クラスターごとに一意である必要があります。

4 コネクターおよびタスクステータスの更新を保存する Kafka トピック。Kafka Connect クラスターごとに一意である必要があります。

5 Kafka に保存するためにメッセージキーを JSON 形式に変換するコンバーター。

6 Kafka に保存するためにメッセージ値を JSON 形式に変換するコンバーター。

7 メッセージキーを構造化された JSON 形式に変換できるスキーマ。

8 メッセージ値を構造化された JSON 形式に変換できるスキーマ。

9 コネクターオフセットを保存する Kafka トピックのレプリケーション係数。

10 コネクターとタスクのステータス設定を保存する Kafka トピックのレプリケーション係数。

11 コネクターとタスクのステータスの更新を保存する Kafka トピックのレプリケーション係数。

### コネクターの Kafka Connector 管理

デプロイメントでワーカー Pod に使用されるコンテナーイメージにプラグインが追加されたら、AMQ Streams の**KafkaConnector** カスタムリソースまたは Kafka Connect API を使用してコネクターインスタンスを管理できます。これらのオプションを使用して、新しいコネクターインスタンスを作成することもできます。

**KafkaConnector** リソースは、Cluster Operator によるコネクターの管理に OpenShift ネイティブのアプローチを提供します。**KafkaConnector** リソースを使用してコネクターを管理するには、**KafkaConnect** カスタムリソースでアノテーションを指定する必要があります。

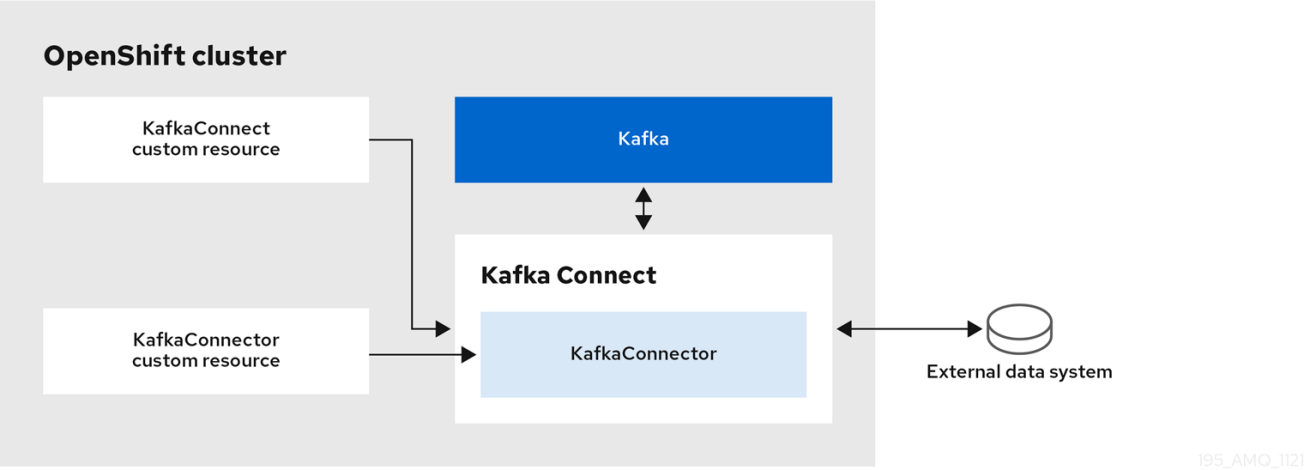
### KafkaConnectors を有効にするためのアノテーション

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
  name: my-connect-cluster
  annotations:
    strimzi.io/use-connector-resources: "true"
  # ...
```

**use-connector-resources** を **true** に設定すると、KafkaConnectors はコネクターを作成、削除、および再設定できます。

**KafkaConnect** 設定で **use-connector-resources** が有効になっている場合は、**KafkaConnector** リソースを使用してコネクターを定義および管理する必要があります。**KafkaConnector** リソースは、外部システムに接続するように設定されています。これらは、外部データシステムと相互作用する Kafka Connect クラスターおよび Kafka クラスターと同じ OpenShift クラスターにデプロイされます。

### 同じ OpenShift クラスター内に含まれる Kafka コンポーネント




設定は、認証を含め、コネクターインスタンスが外部データシステムに接続する方法を指定します。また、監視するデータを指定する必要があります。ソースコネクターの場合は、設定でデータベース名を指定できます。ターゲットトピック名を指定することで、Kafka のどこにデータを配置するかを指定することもできます。

タスクの最大数を指定するには、**tasksMax** を使用します。たとえば、**tasksMax: 2** のソースコネクターは、ソースデータのインポートを 2 つのタスクに分割する場合があります。

### KafkaConnector ソースコネクター設定の例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnector
metadata:
  name: my-source-connector (1)
  labels:
    strimzi.io/cluster: my-connect-cluster (2)
spec:
  class: org.apache.kafka.connect.file.FileStreamSourceConnector (3)
  tasksMax: 2 (4)
  config: (5)
    file: "/opt/kafka/LICENSE" (6)
    topic: my-topic (7)
    # ...
```

(1) コネクターの名前として使用される **KafkaConnector** リソースの名前。OpenShift リソースで有効な名前を使用します。

(2) コネクターインスタンスを作成する Kafka Connect クラスターの名前。コネクターは、リンク先の Kafka Connect クラスターと同じ namespace にデプロイする必要があります。

(3) コネクタークラスのフルネーム。これは、Kafka Connect クラスターによって使用されているイメージに存在するはずです。

(4) コネクターが作成できる Kafka Connect タスクの最大数。

(5) キーと値のペアとしての コネクター設定。

(6) 外部データファイルの場所。この例では、**/opt/kafka/LICENSE** ファイルから読み取るように **FileStreamSourceConnector** を設定しています。

7 ソースデータのパブリッシュ先となる Kafka トピック。

**注記**

OpenShift Secrets または ConfigMaps から コネクターの機密設定値をロード できます。

### Kafka Connect API

**KafkaConnector** リソースを使用してコネクターを管理する代わりに、Kafka Connect REST API を使用します。Kafka Connect REST API は、**<connect_cluster_name>-connect-api:8083** で実行しているサービスとして利用できます。ここで、**<connect_cluster_name>** は、お使いの Kafka Connect クラスターの名前になります。

コネクター設定を JSON オブジェクトとして追加します。

**コネクター設定を追加するための curl 要求の例**

```
curl -X POST \
  http://my-connect-cluster-connect-api:8083/connectors \
  -H 'Content-Type: application/json' \
  -d '{ "name": "my-source-connector",
    "config":
    {
      "connector.class":"org.apache.kafka.connect.file.FileStreamSourceConnector",
      "file": "/opt/kafka/LICENSE",
      "topic":"my-topic",
      "tasksMax": "4",
      "type": "source"
    }
}'
```

KafkaConnectors が有効になっている場合、Kafka Connect REST API に直接手作業で追加された変更は Cluster Operator によって元に戻されます。

REST API でサポートされる操作は、Apache Kafka のドキュメント を参照してください。

**注記**

Kafka Connect API サービスを OpenShift の外部に公開できます。これを行うには、入力やルートなどのアクセスを提供する接続メカニズムを使用するサービスを作成します。接続のセキュリティーが低いので慎重に使用してください。

**関連情報**

- Kafka Connect 設定オプション
- 複数インスタンスの Kafka Connect 設定
- プラグインを使用した Kafka Connect の拡張
- AMQ Streams を使用した新しいコンテナーイメージの自動作成
- Kafka Connect ベースイメージからの Docker イメージの作成

- Build スキーマ参照
- ソースおよびシンクコネクターの設定オプション
- 外部ソースからの設定値の読み込み

## 7.6. KAFKA BRIDGE の設定

Kafka Bridge 設定には、接続先の Kafka クラスターのブートストラップサーバー仕様と、必須の暗号化および認証オプションが必要になります。

コンシューマーの Apache Kafka 設定ドキュメント および プロデューサーの Apache Kafka 設定ドキュメント で説明されているように、Kafka Bridge コンシューマーおよびプロデューサー設定は標準です。

HTTP 関連の設定オプションでは、サーバーがリッスンするポート接続を設定します。

### CORS

Kafka Bridge では、CORS (Cross-Origin Resource Sharing) の使用がサポートされます。CORS は、複数のオリジンから指定のリソースにブラウザーでアクセスできるようにする HTTP メカニズムです (たとえば、異なるドメイン上のリソースへのアクセス)。CORS を使用する場合、Kafka Bridge を通じた Kafka クラスターとの対話用に、許可されるリソースオリジンおよび HTTP メソッドのリストを定義できます。リストは、Kafka Bridge 設定の **http** 仕様で定義されます。

CORS では、異なるドメイン上のオリジンソース間での **シンプルな** 要求および **プリフライト** 要求が可能です。

- シンプルな要求は、使用可能なオリジンをヘッダーに定義する必要のある HTTP 要求です。
- プリフライト要求では、オリジンとメソッドが使用可能であることを確認する実際の要求の前に、最初の OPTIONS HTTP 要求が送信されます。

### Kafka ブリッジ設定の YAML 例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaBridge
metadata:
  name: my-bridge
spec:
  # ...
  bootstrapServers: my-cluster-kafka:9092
  http:
    port: 8080
    cors:
      allowedOrigins: "https://strimzi.io"
      allowedMethods: "GET,POST,PUT,DELETE,OPTIONS,PATCH"
  consumer:
    config:
      auto.offset.reset: earliest
  producer:
    config:
      delivery.timeout.ms: 300000
  # ...
```

### 関連情報

- Fetch CORS 仕様

# 第8章 KAFKA のセキュリティー

AMQ Streams のセキュアなデプロイメントには、以下が含めることができます。

- データ交換の暗号化
- アイデンティティー証明に使用する認証
- ユーザーが実行するアクションを許可または拒否する認可

## 8.1. 暗号化

AMQ Streams は、暗号化通信用のプロトコルである Transport Layer Security (TLS) をサポートします。

通信は、以下の間で常に暗号化されます。

- Kafka ブローカー
- ZooKeeper ノード
- Operator および Kafka ブローカー
- Operator および ZooKeeper ノード
- Kafka Exporter

Kafka ブローカーとクライアントの間で TLS 暗号化を設定することもできます。Kafka ブローカーの外部リスナーを設定するときに、外部クライアントに対して TLS が指定されます。

AMQ Streams コンポーネントおよび Kafka クライアントは、暗号化にデジタル署名を使用します。Cluster Operator は、証明書を設定し、Kafka クラスター内で暗号化を有効にします。Kafka クライアントと Kafka ブローカーの通信やクラスター間の通信に、**Kafka リスナー証明書**と呼ばれる独自のサーバー証明書を指定できます。

AMQ Streams は **シークレット** を使用して、mTLS に必要な証明書および秘密鍵を PEM および PKCS #12 形式で保存します。

TLS CA (認証局) は、コンポーネントの ID を認証するために証明書を発行します。AMQ Streams は、CA 証明書に対してコンポーネントの証明書を検証します。

- AMQ Streams コンポーネントは、**クラスター CA** 証明局に対して検証されます。
- Kafka クライアントは、**クライアント CA** 証明局に対して検証されます。

## 8.2. 認証

Kafka リスナーは認証を使用して、Kafka クラスターへのクライアント接続のセキュリティーを確保します。

サポート対象の認証メカニズム:

- mTLS 認証 (TLS が有効な暗号化を使用するリスナーの場合)
- SASL SCRAM-SHA-512

- OAuth 2.0 のトークンベースの認証
- カスタム認証

User Operator では mTLS および SCRAM 認証のユーザー認証情報は管理対象ですが、OAuth 2.0 は管理対象ではありません。たとえば、User Operator を使用して、Kafka クラスターにアクセスする必要があるクライアントに対応するユーザーを作成し、認証タイプとして **tls** を指定できます。

OAuth 2.0 トークンベースの認証を使用すると、アプリケーションクライアントは、アカウント認証情報を公開せずに Kafka ブローカーにアクセスできます。承認サーバーは、アクセスの付与とアクセスに関する問い合わせを処理します。

カスタム認証では、kafka でサポートされているあらゆるタイプの認証が可能です。柔軟性を高めることができますが、複雑さも増します。

## 8.3. 承認

Kafka クラスターは、承認メカニズムを使用して特定のクライアントまたはユーザーによって Kafka ブローカーで許可される操作を制御します。承認は、Kafka クラスターに適用されると、クライアント接続に使用する全リスナーに対して有効になります。

ユーザーを Kafka ブローカー設定の **スーパーユーザー** のリストに追加すると、承認メカニズムにより適用される承認制約に関係なく、そのユーザーにはクラスターへのアクセスが無制限に許可されます。

サポート対象の承認メカニズム:

- 簡易承認
- OAuth 2.0 での承認 (OAuth 2.0 トークンベースの認証を使用している場合)
- Open Policy Agent (OPA) での承認
- カスタム承認

簡易承認では、デフォルトの Kafka 承認プラグインである **AclAuthorizer** が使用されます。**AclAuthorizer** は、アクセス制御リスト (ACL) を使用して、どのユーザーがどのリソースにアクセスできるかを定義します。カスタム認証の場合は、ACL ルールを適用するように独自の **Authorizer** プラグインを設定します。

OAuth 2.0 および OPA は、承認サーバーからのポリシーベースの制御を提供します。Kafka ブローカーのリソースへのアクセス権限を付与するのに使用されるセキュリティーポリシーおよびパーミッションは、承認サーバーで定義されます。

承認サーバーに接続し、クライアントまたはユーザーが要求した操作が許可または拒否されることを検証するのに、URL が使用されます。ユーザーとクライアントは、承認サーバーで作成されるポリシーと照合され、Kafka ブローカーで特定のアクションを実行するためのアクセスが許可されます。

# 第9章 モニタリング

モニタリングデータでは、AMQ Streams のパフォーマンスおよびヘルスを監視できます。分析および通知のメトリクスデータを取得するようにデプロイメントを設定できます。

メトリクスデータは、接続性およびデータ配信の問題を調査するときに役立ちます。たとえば、メトリクスデータを使用すると、更新されていないパーティションや、メッセージの消費速度を特定できます。アラートルールでは、指定した通信チャネルを使用して、このようなメトリクスに関する緊急通知を送信できます。モニタリングの視覚化では、デプロイメントの設定更新のタイミングと方法を判別できるように、リアルタイムのメトリクスデータを表示します。メトリクス設定ファイルのサンプルは AMQ Streams に同梱されています。

分散トレースは、AMQ Streams でメッセージのエンドツーエンドのトレース機能を提供することで、メトリクスデータの収集を補完します。

Cruise Control は、ワークロードのデータに基づく Kafka クラスターのリバランスをサポートします。

### メトリクスおよびモニタリングツール

AMQ Streams では、メトリクスおよびモニタリングに以下のツールを使用できます。

**Prometheus**

Prometheus は、Kafka、ZooKeeper、および Kafka Connect クラスターからメトリクスをプルします。Prometheus の **Alertmanager** プラグインはアラートを処理して、そのアラートを通知サービスにルーティングします。

**Kafka Exporter**

Kafka Exporter は、さらに Prometheus メトリクスを追加します。

**Grafana**

Grafana Labs は、ダッシュボードで Prometheus メトリクスを視覚化できます。

**Jaeger**

Jaeger のドキュメント は、アプリケーション間のトランザクションを追跡するための分散トレースサポートを提供します。

**Cruise Control**

Cruise Control は、データ分散を監視し、Kafka クラスター全体でデータのリバランスを実行します。

## 9.1. PROMETHEUS

Prometheus は、Kafka コンポーネントおよび AMQ Streams Operator からメトリクスデータを抽出できます。

Prometheus を使用してメトリクスデータを取得し、アラートを発行するには、Prometheus および Prometheus Alertmanager プラグインをデプロイする必要があります。メトリクスデータを公開するには、Kafka リソースもメトリクス設定でデプロイまたは再デプロイする必要があります。

Prometheus は、公開されたメトリクスデータをモニタリング用に収集します。Alertmanager は、事前に定義されたアラートルールをもとに、条件が問題発生の可能性を示した場合に、アラートを発行します。

メトリクスおよびアラートルール設定ファイルのサンプルは AMQ Streams に同梱されています。AMQ Streams に含まれるアラートメカニズムのサンプルは、通知を Slack チャネルに送信するように設定されています。

## 9.2. GRAFANA

Grafana は Prometheus によって公開されるメトリクスデータを使用して、モニタリングできるように、ダッシュボードを視覚化して表示します。

データソースとして Prometheus を追加している場合には、Grafana のデプロイメントが必要です。ダッシュボードの例 (AMQ Streams JSON ファイルとして提供) は、モニタリングデータを表示するために、Grafana インターフェイスを使用してインポートされます。

## 9.3. KAFKA EXPORTER

Kafka Exporter は、Apache Kafka ブローカーおよびクライアントのモニタリングを強化するオープンソースプロジェクトです。Kafka Exporter は、Kafka クラスターとともにデプロイされ、オフセット、コンシューマーグループ、コンシューマーラグ、およびトピックに関連する Kafka ブローカーからの Prometheus メトリクスデータを追加で抽出します。提供される Grafana ダッシュボードを使用して、Prometheus が Kafka Exporter から収集したデータを可視化することができます。

サンプル設定ファイル、アラートルール、および Kafka Exporter の Grafana ダッシュボードは AMQ Streams で提供されます。

## 9.4. 分散トレース

分散トレースは、分散システム内のアプリケーション間のトランザクションの進行状況を追跡します。マイクロサービスのアーキテクチャーでは、トレースはサービス間のトランザクションの進捗を追跡します。トレースデータは、アプリケーションのパフォーマンスを監視し、ターゲットシステムおよびエンドユーザーアプリケーションの問題を調べるのに役立ちます。

AMQ Streams では、トレースによってメッセージのエンドツーエンドの追跡が容易になります。これは、ソースシステムから Kafka、さらに Kafka からターゲットシステムおよびアプリケーションへのメッセージの追跡です。分散トレースは、Grafana ダッシュボードおよびコンポーネントロガーでのメトリックの監視を補完します。

トレースのサポートは、以下の Kafka コンポーネントに組み込まれています。

- ソースクラスターからターゲットクラスターへのメッセージをトレースする MirrorMaker
- Kafka Connect が使用して生成したメッセージをトレースする Kafka Connect
- Kafka と HTTP クライアントアプリケーション間のメッセージをトレースする Kafka Bridge

トレースは Kafka ブローカーではサポートされません。

カスタムリソースを使用して、これらのコンポーネントのトレースを有効にして設定します。**spec.template** プロパティーを使用してトレース設定を追加します。

**spec.tracing.type** プロパティーを使用してトレースタイプを指定することにより、トレースを有効にします。

**opentelemetry**

**type: opentelemetry** を指定して、OpenTelemetry を使用します。デフォルトでは、OpenTelemetry は OTLP (OpenTelemetry Protocol) エクスポーターとエンドポイントを使用してトレースデータを取得します。Jaeger トレースなど、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。これを行うには、トレース設定で OpenTelemetry エクスポーターとエンドポイントを変更します。

**jaeger**

OpenTracing と Jaeger クライアントを使用してトレースデータを取得するには、**type:jaeger** を指定します。

**注記**

**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、今後の Kafka バージョンのサポートを保証できません。可能であれば、**type: jaeger** トレースのサポートを 2023 年 6 月まで維持し、その後削除します。できるだけ早く OpenTelemetry に移行してください。

**Kafka クライアントのトレース**
Kafka プロデューサーやコンシューマーなどのクライアントアプリケーションも、トランザクションをモニタリングするように設定できます。クライアントはトレースプロファイルで設定され、トレーサーはクライアントアプリケーションが使用するように初期化されます。

## 9.5. CRUISE CONTROL

Cruise Control は、次の Kafka 操作をサポートするオープンソースシステムです。

- クラスターワークロードのモニタリング
- 定義済みの制約に基づくクラスターのリバランス

この操作は、ブローカー Pod をより効率的に使用する、よりバランスの取れた Kafka クラスターを実行するのに役立ちます。

通常、クラスターの負荷は時間とともに不均等になります。大量のメッセージトラフィックを処理するパーティションは、使用可能なブローカー全体で均等に分散されない可能性があります。クラスターを再分散するには、管理者はブローカーの負荷を監視し、トラフィックの多いパーティションを容量に余裕のあるブローカーに手作業で再割り当てします。

Cruise Control はクラスターのリバランス処理を自動化します。CPU、ディスク、およびネットワーク負荷を基にして、クラスターにおけるリソース使用の**ワークロードモデル**を構築し、パーティションの割り当てをより均等にする、最適化プロポーザル (承認または拒否可能) を生成します。これらのプロポーザルの算出には、設定可能な最適化ゴールが複数使用されます。

特定のモードで最適化の提案を生成できます。デフォルトの **full** モードでは、すべてのブローカー間でパーティションがリバランスされます。**add-brokers** および **remove-brokers** モードを使用して、クラスターをスケールアップまたはスケールダウンするときの変更に対応することもできます。

最適化プロポーザルを承認すると、Cruise Control はそのプロポーザルを Kafka クラスターに適用します。**KafkaRebalance** リソースを使用して、最適化の提案を設定および生成します。最適化の提案が自動または手動で承認されるように、アノテーションを使用してリソースを設定できます。

**注記**

Prometheus は、Cruise Control のメトリクスデータを抽出できます。これには、最適化プロポーザルおよびリバランス操作に関連するデータが含まれます。サンプル設定ファイルおよび Cruise Control の Grafana ダッシュボードは、AMQ Streams で提供されます。

# 付録A サブスクリプションの使用

AMQ Streams は、ソフトウェアサブスクリプションから提供されます。サブスクリプションを管理するには、Red Hat カスタマーポータルでアカウントにアクセスします。

## アカウントへのアクセス

1. access.redhat.com に移動します。
2. アカウントがない場合は、作成します。
3. アカウントにログインします。

## サブスクリプションのアクティベート

1. access.redhat.com に移動します。
2. **My Subscriptions** に移動します。
3. **Activate a subscription** に移動し、16 桁のアクティベーション番号を入力します。

## Zip および Tar ファイルのダウンロード

zip または tar ファイルにアクセスするには、カスタマーポータルを使用して、ダウンロードする関連ファイルを検索します。RPM パッケージを使用している場合は、この手順は必要ありません。

1. ブラウザーを開き、access.redhat.com/downloads で Red Hat カスタマーポータルの **Product Downloads** ページにログインします。
2. **INTEGRATION AND AUTOMATION** カテゴリーで、**AMQ Streams for Apache Kafka**エントリーを見つけます。
3. 必要な AMQ Streams 製品を選択します。**Software Downloads** ページが開きます。
4. コンポーネントの **Download** リンクをクリックします。

## DNF を使用したパッケージのインストール

パッケージとすべてのパッケージ依存関係をインストールするには、以下を使用します。

```
dnf install <package_name>
```

ローカルディレクトリーからダウンロード済みのパッケージをインストールするには、以下を使用します。

```
dnf install <path_to_download_package>
```

**改訂日時:** 2023-04-06