Red Hat AMQ 2021.q2

AMQ Streams on OpenShift の概要

OpenShift Container Platform 上で AMQ Streams 1.7 を使用

Last Updated: 2021-06-08

Red Hat AMQ 2021.q2 AMQ Streams on OpenShift の概要

OpenShift Container Platform 上で AMQ Streams 1.7 を使用

法律上の通知

概要

本ガイドでは、AMQ Streams の特長および機能を概説します。

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。これは大規模な取り組みであるため、これらの変更は今後の複数のリリースで段階的に実施されます。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。
- 即座にアクセスするために大容量のデータを保持。

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。
- メトリクスによるオペレーションの監視。
- ログの収集および集計。
- 分散システムのログのコミット。
- 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 で提供される Operator は、Kafka を効果的に管理するために、専門的なオペレーション情報で目的に合うよう構築されています。

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 へのアクセスをセキュア化。
- 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 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。

その他のリソース

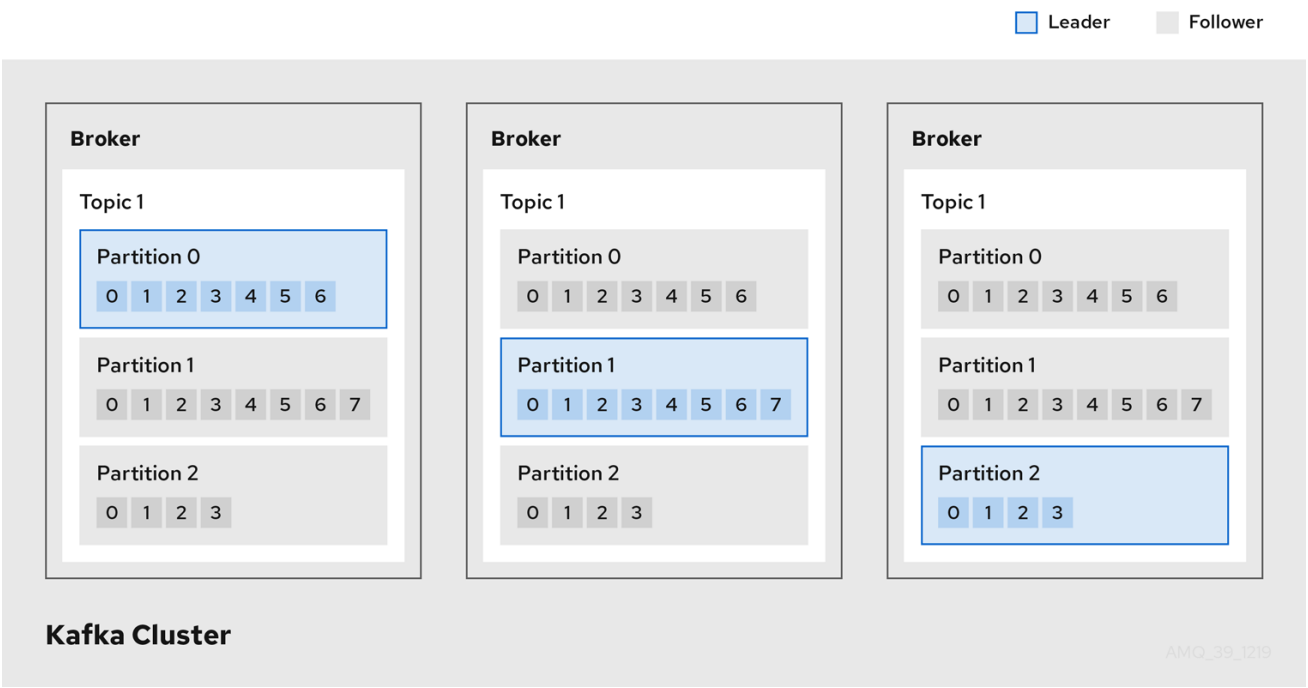
- Apache Kafka の詳細は、Apache Kafka の Web サイト を参照してください。

2.1. KAFKA の概念

Kafka の主な概念を知ることは、AMQ Streams の仕組みを理解するうえで重要です。

Kafka クラスターは、複数のブローカーで構成されます。トピックは Kafka クラスターでのデータ受信および保存に使用されます。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 つで構成されます。

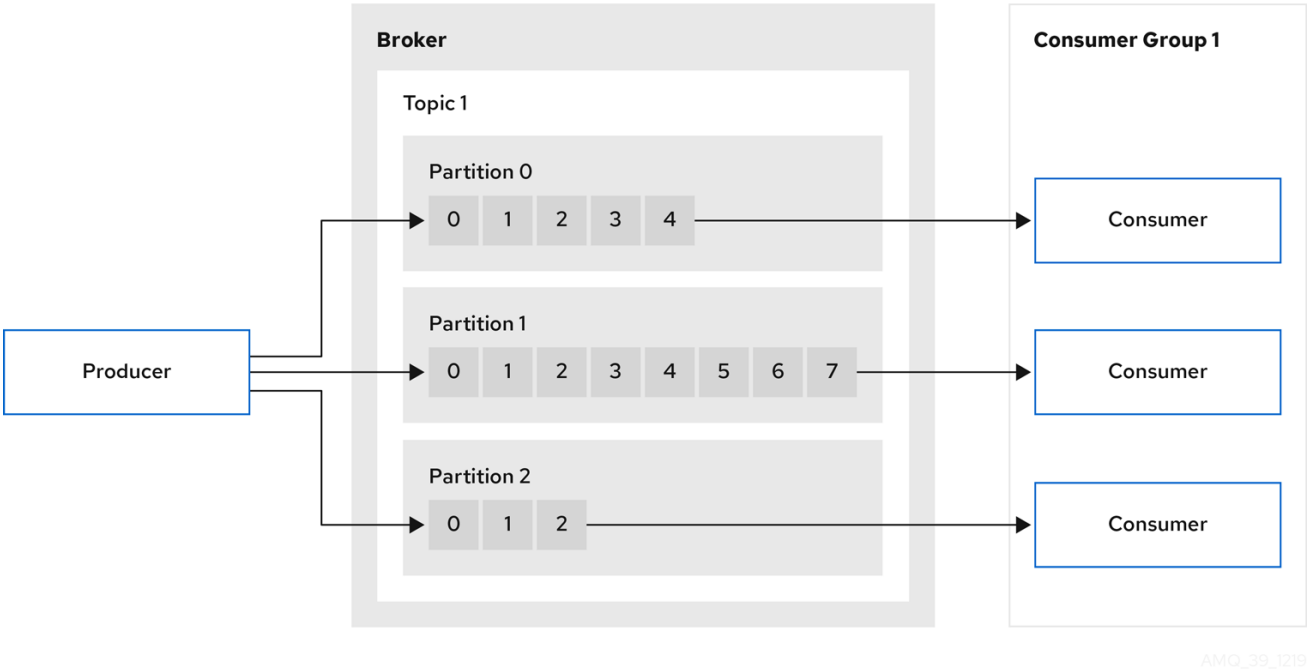
in-sync レプリカ (ISR) はリーダーと同数のメッセージを保持しています。設定では、メッセージを生成できるように **同期する** 必要のあるレプリカ数を定義し、メッセージがレプリカパーティションに正常にコピーされない限りに、コミットされないようにします。こうすることで、リーダーに障害が発生しても、メッセージは失われません。

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

2.2. 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** で構成されます。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

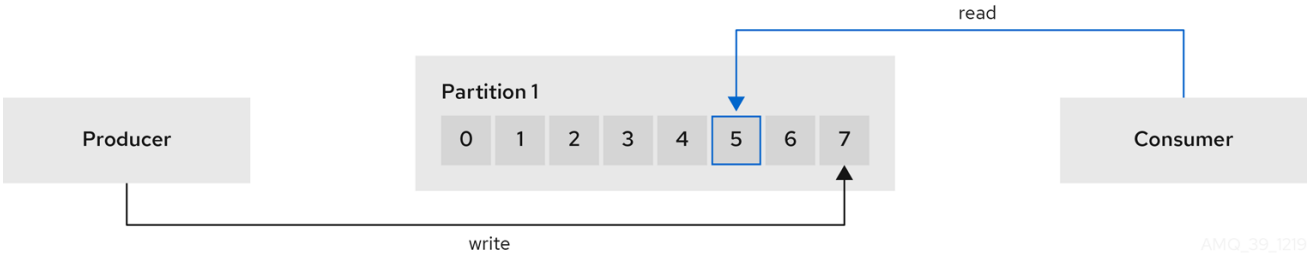
コンシューマーグループは、特定のトピックから複数のプロデューサーによって生成される、典型的に大量のデータストリームを共有するのに使用します。コンシューマーは **group.id** でグループ化され、メッセージをメンバー全体に分散できます。グループ内のコンシューマーは、同じパーティションからのデータは読み取りませんが、1 つ以上のパーティションからデータを受信できます。

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3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 にデプロイするために提供されます。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。

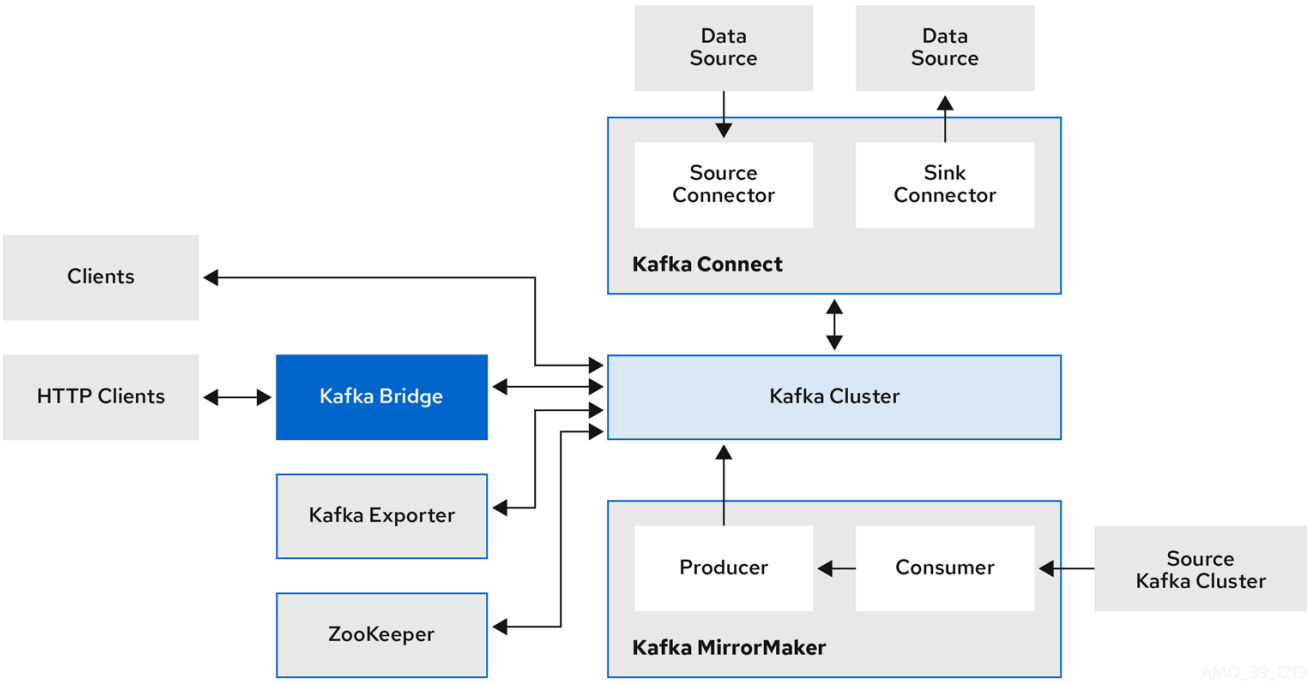
3.1. KAFKA コンポーネントのアーキテクチャー

Kafka ブローカーのクラスターは、Apache Kafka プロジェクトの主要な部分で、メッセージの配信を行います。

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、パーティションのリーダー選択にも使用されます。

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と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。

  Source2Image サポートを含めて Kafka Connect をデプロイすることで、コネクターを便利な方法で追加できます。

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。

3.2. KAFKA BRIDGE インターフェース

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 インターフェースが提供されます。 また、クライアントアプリケーションによる Kafka プロトコルの変換は必要なく、Web API コネクションの利点が AMQ Streams に提供されます。

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。

3.2.1. HTTP リクエスト

Kafka Bridge は、以下の方法で Kafka クラスターへの HTTP リクエストをサポートします。

- 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。

その他のリソース

- リクエストおよび応答の例など、API ドキュメントを確認するには、『Strimzi Kafka Bridge Documentation 』を参照してください。

3.2.2. 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。

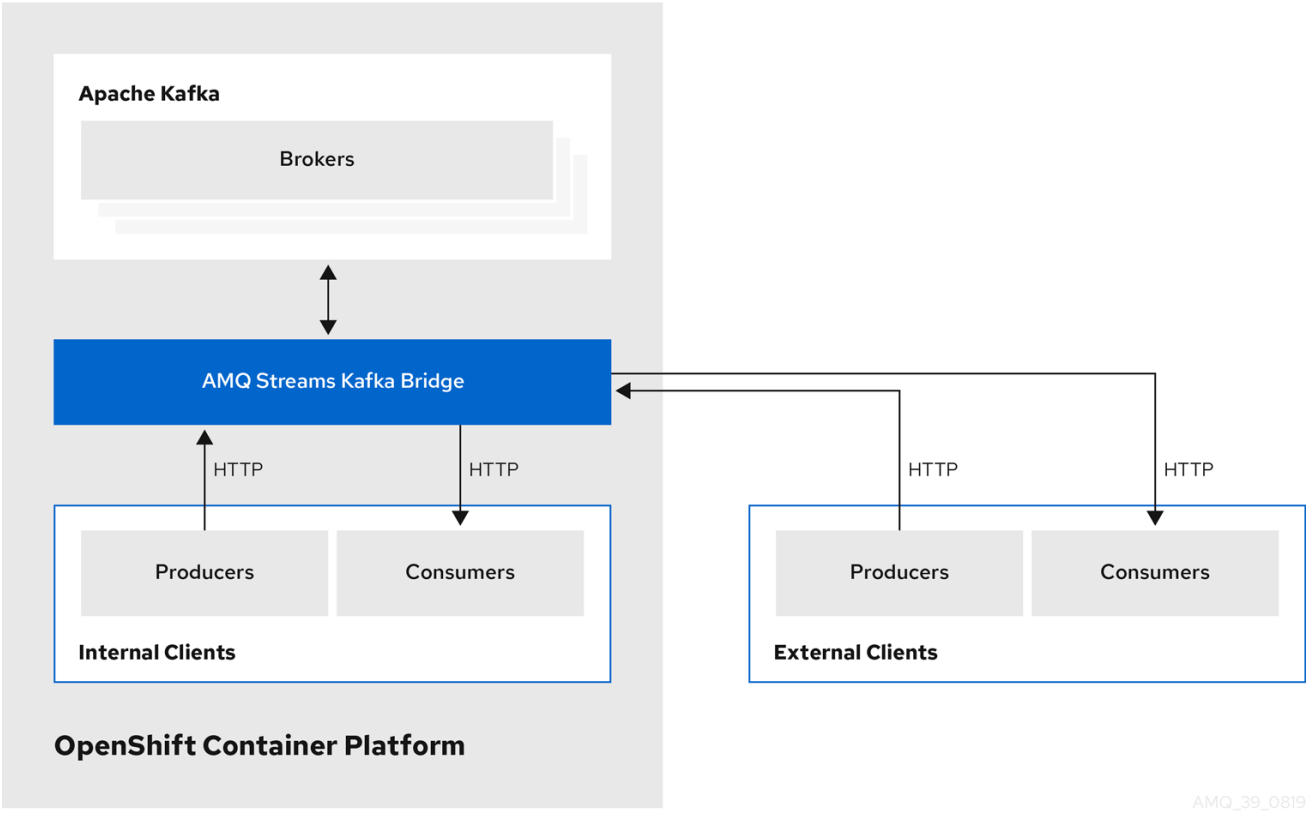
内部クライアント

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4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、必要な手動の作業が少なくなります。

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、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 を構成します。

Topic Operator

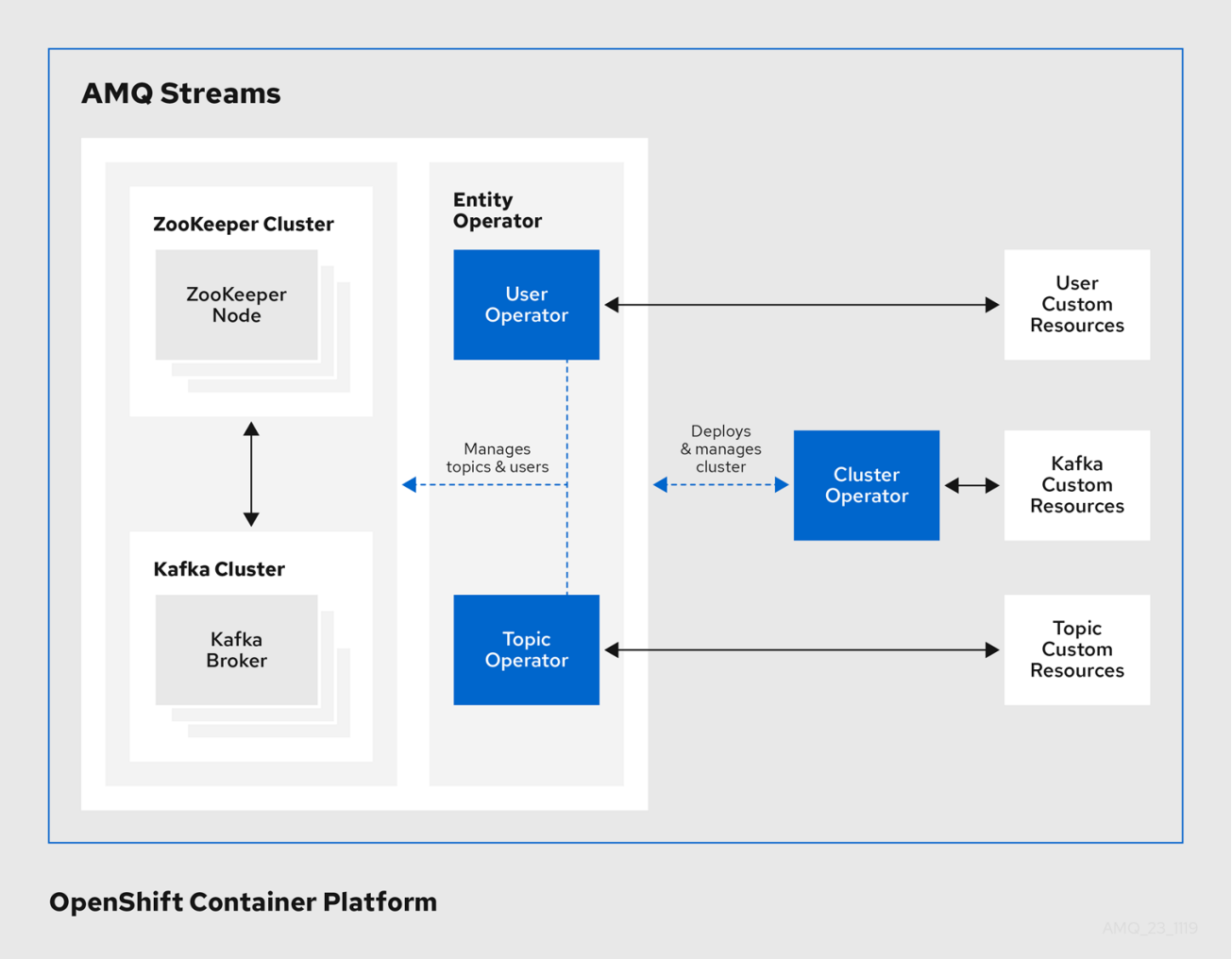
Kafka トピックを管理します。

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



4.1. CLUSTER OPERATOR

AMQ Streams では、Cluster Operator を使用して以下のクラスターをデプロイおよび管理します。

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

クラスターのデプロイメントにはカスタムリソースが使用されます。

たとえば、以下のように Kafka クラスターをデプロイします。

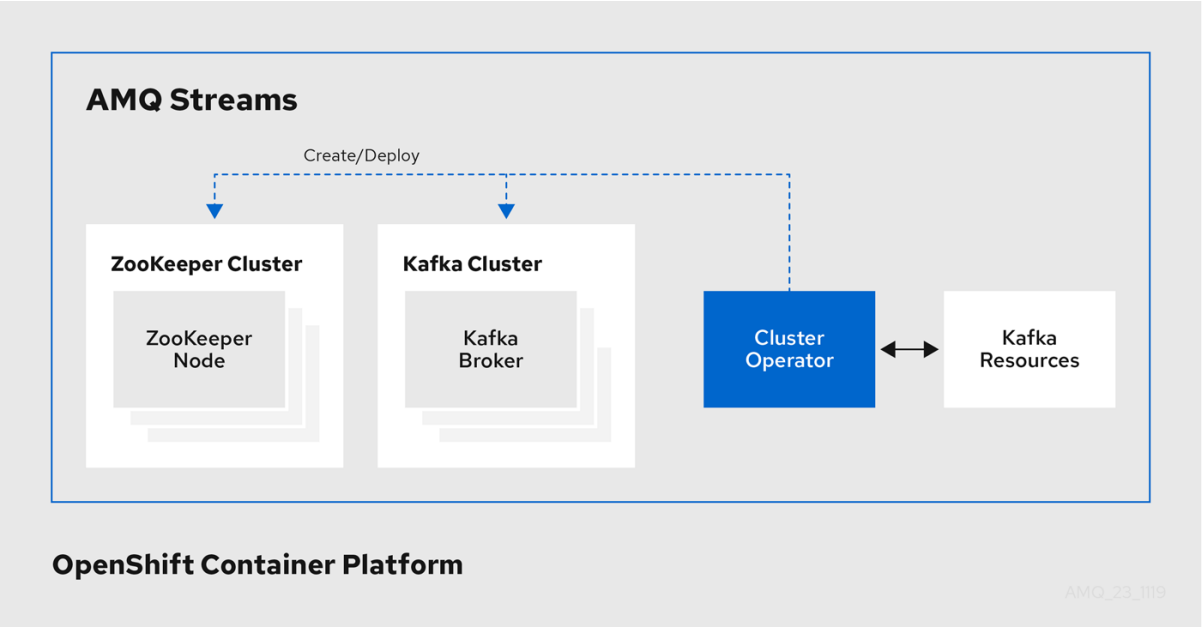
- クラスター設定のある **Kafka** リソースが OpenShift クラスター内で作成されます。
- **Kafka** リソースに宣言された内容を基にして、該当する Kafka クラスターが Cluster Operator によってデプロイされます。

Cluster Operator で以下もデプロイできます (**Kafka** リソースの設定より)。

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

デプロイメントの Entity Operator 内の Topic Operator および User Operator 関数。

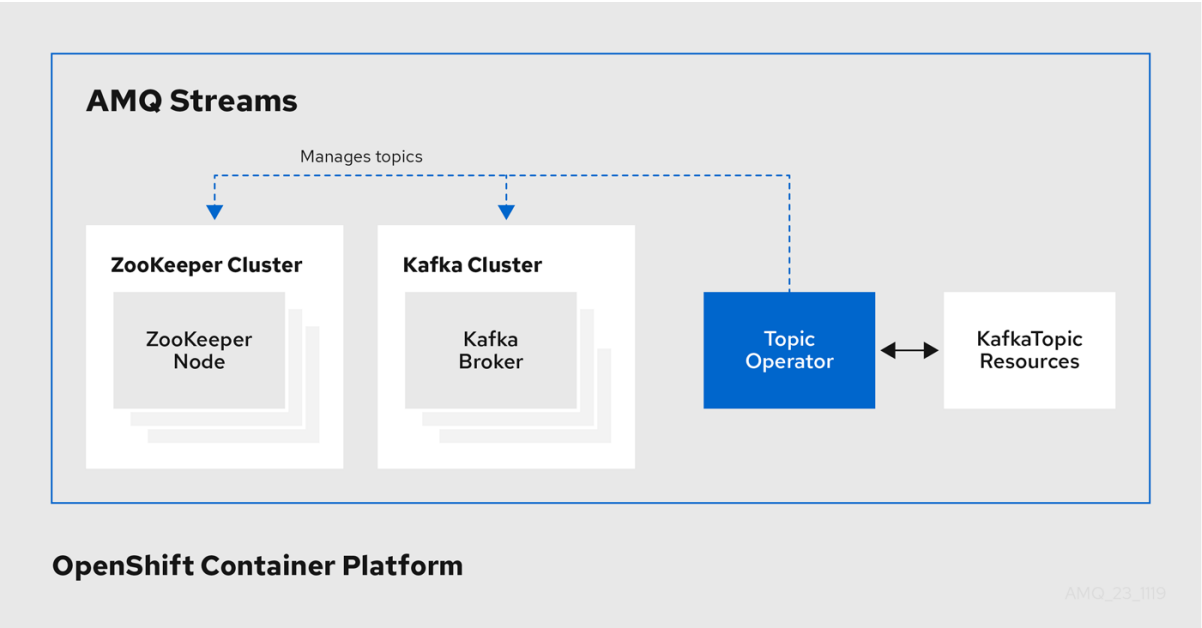
Cluster Operator のアーキテクチャー例



4.2. 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 の役割は、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。

KafkaTopic とトピックの関係は次のとおりです。

- **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。
- **KafkaTopic** が削除されると、Topic Operator によってトピックが削除されます。

- **KafkaTopic** が変更されると、Topick Operator によってトピックが更新されます。

上記と逆になるトピックと **KafkaTopic** の関係は次のとおりです。

- 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。
- トピックが 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。
- トピックが 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。

4.3. 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** とユーザーの関係は次のようになります。

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

第5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。

本章では、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。

5.1. 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。

5.2. 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。制限によって、指定のコンテナーが消費可能な最大リソースが指定されます。
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
# ...
```

5.3. 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構成されます。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** タイプを使用して指定されます。

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** と NGINX Ingress Controller for Kubernetes を使用する **ingress**

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

ラックアウェアネス (rack awareness) は、Kafka ブローカーの Pod とトピックレプリカを **racks** 全体に分散する設定機能です。ラックとは、データセンターまたは、データセンター内のラック、アベイラビリティーゾーンを表します。

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
    # ...
```

5.4. 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。

MirrorMaker 2.0

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 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。

クラスターの設定

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。

- **active/active** 設定では、両方のクラスターがアクティブで、同じデータを同時に提供します。これは、地理的に異なる場所で同じデータをローカルで利用可能にする場合に便利です。
- **active/passive** 設定では、アクティブなクラスターからのデータはパッシブなクラスターで複製され、たとえば、システム障害時のデータ復旧などでスタンバイ状態を維持します。

KafkaMirrorMaker2 カスタムリソースを設定し、ソースおよびターゲットクラスターの接続詳細を含む Kafka Connect デプロイメントを定義します。次に、複数の MirrorMaker 2.0 コネクターを実行し、接続を確立します。

トピックの設定は、**KafkaMirrorMaker2** カスタムリソースに定義されたトピックに従って、ソースクラスターとターゲットクラスターの間で自動的に同期化されます。設定の変更はリモートトピックに伝播されるため、新しいトピックおよびパーティションは削除および作成されます。トピックのレプリケーションは、トピックを許可または拒否するために、正規表現パターンを使用して定義されます。

以下の MirrorMaker 2.0 コネクターおよび関連する内部トピックは、クラスター間でのデータの転送および同期を管理するのに役立ちます。

MirrorSourceConnector

MirrorSourceConnector は、ソースクラスターからリモートトピックを作成します。

MirrorCheckpointConnector

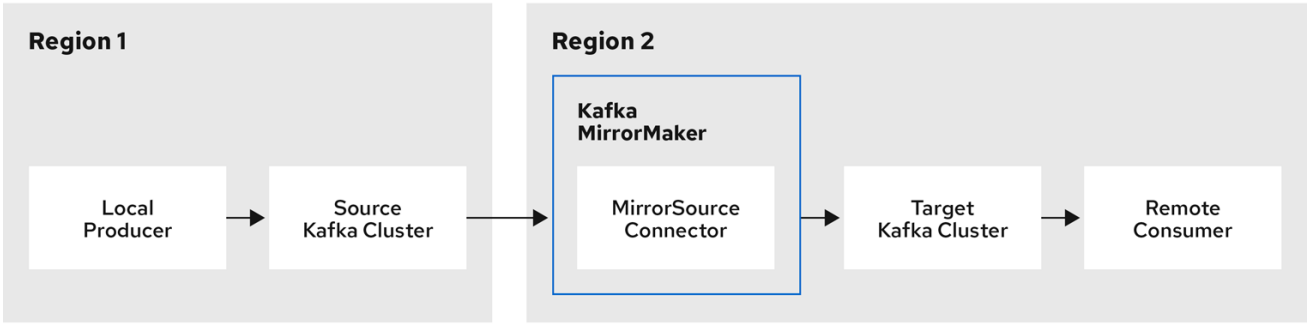
MirrorCheckpointConnector は、**オフセット同期** (offset sync) トピックと **チェックポイント** (checkpoint) トピックを使用して、指定のコンシューマーグループのオフセットを追跡し、マッピングします。オフセット同期トピックは、複製されたトピックパーティションのソースおよびターゲットオフセットをレコードメタデータからマッピングします。チェックポイントは、各ソースクラスターから生成され、チェックポイントトピックを介してターゲットクラスターでレプリケートされます。チェックポイントトピックは、各コンシューマーグループのレプリケートされたトピックパーティションのソースおよびターゲットクラスターで最後にコミットされたオフセットをマッピングします。

MirrorHeartbeatConnector

MirrorHeartbeatConnector は、クラスター間の接続を定期的に確認します。ハートビートは、ローカルクラスターで作成される **ハートビート** (heartbeat) トピックで、MirrorHeartbeatConnector によって毎秒作成されます。MirrorMaker 2.0 がリモートとローカルの両方にある場合、リモートで MirrorHeartbeatConnector によって生成されるハートビートはリモートトピックと同様に処理され、ローカルクラスターで MirrorSourceConnector によってミラーリング

されます。ハートビートトピックによって、リモートクラスターが利用可能で、クラスターが接続されていることを簡単にチェックできます。障害が発生した場合、ハートビートトピックのオフセットポジションとタイムスタンプは復旧と診断に役立ちます。

図5.1 2 つのクラスターにおけるレプリケーション



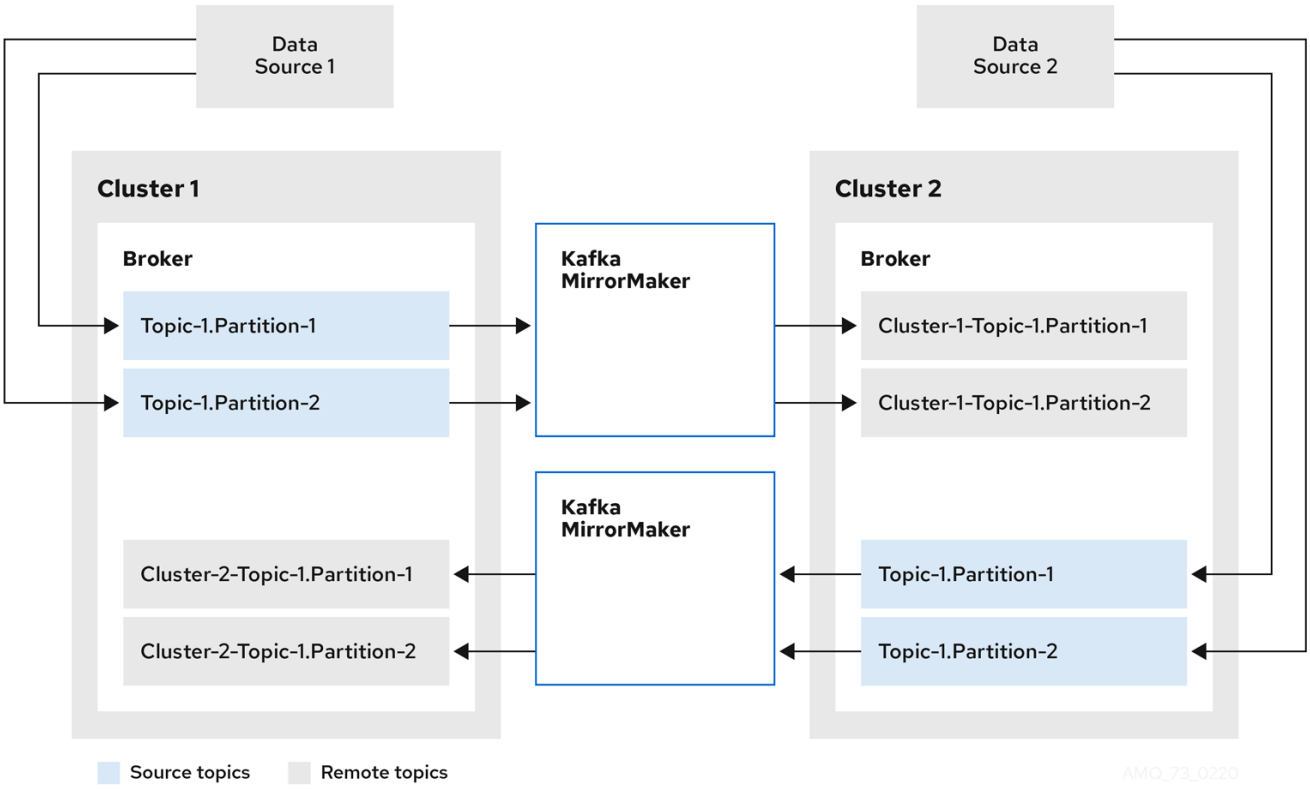
2 つのクラスターにおける双方向レプリケーション

MirrorMaker 2.0 アーキテクチャーは、**アクティブ/アクティブ** クラスター設定で双方向レプリケーションをサポートするため、両方のクラスターがアクティブになり、同じデータを同時に提供します。MirrorMaker 2.0 クラスターは、ターゲット宛先ごとに必要です。

リモートトピックは、クラスター名をトピック名に追加する、自動名前変更によって区別されます。これは、同じデータを地理的に異なる場所でローカルで使用できるようにする場合に便利です。

ただし、active/passive クラスター設定でデータをバックアップまたは移行する場合は、トピックの元の名前を維持することが望ましい場合があります。その場合は、自動名前変更を無効にするように MirrorMaker 2.0 を設定できます。

図5.2 双方向レプリケーション



MirrorMaker 2.0 設定の YAML の例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaMirrorMaker2
metadata:
  name: my-mirror-maker2
  spec:
    version: 2.7.0
    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"
```

MirrorMaker

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。

許可リスト (allowlist) では、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
    # ...
  whitelist: "my-topic|other-topic"
  # ...
```

5.5. KAFKA CONNECT の設定

Kafka Connect の基本設定には、Kafka クラスターに接続するブートストラップアドレスと、暗号化および認証の詳細が必要です。

Kafka Connect インスタンスはデフォルトでは、以下が同じ値で設定されます。

- Kafka Connect クラスターのグループ ID
- コネクターオフセットを保存する Kafka トピック
- コネクターおよびタスクステータス設定を保存する Kafka トピック
- コネクターおよびタスクステータスの更新情報を保存する Kafka トピック

複数の異なる Kafka Connect インスタンスが使用されている場合には、上記の設定はインスタンスごとに反映する必要があります。

Kafka Connect 設定の YAML 例

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
  name: my-connect
spec:
  # ...
  config:
    group.id: my-connect-cluster
    offset.storage.topic: my-connect-cluster-offsets
    config.storage.topic: my-connect-cluster-configs
    status.storage.topic: my-connect-cluster-status
  # ...
```

コネクター

コネクターは Kafka Connect とは別に設定されます。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。外部ソースデータは、対象の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。

Kafka には、以下のようにビルトインコネクターが 2 つあります。

- **FileStreamSourceConnector** は、外部システムから Kafka にデータをストリーミングし、入力ソースから行を読み取り、各行を Kafka トピックに送信します。
- **FileStreamSinkConnector** は、Kafka から外部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、出力ファイルにメッセージごとに 1 行を作成します。

コネクタープラグインを使用して他のコネクターを追加できます。コネクタープラグインは、JAR ファイルまたは TGZ アーカイブのセットで、特定タイプの外部システムへの接続に必要な実装を定義します。

新しいコネクタープラグインを使用するカスタム Kafka Connect イメージを作成します。

イメージを作成するには、以下を使用します。

- AMQ Streams が新しいイメージを自動的に作成するための Kafka Connect の設定。
- ベースイメージとしての Red Hat Ecosystem Catalog の Kafka コンテナーイメージ
- 新規コンテナーイメージを作成する OpenShift ビルド と S2I (Source-to-Image) フレームワーク。

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

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。**plugins** プロパティーは、アーティファクトのタイプとアーティファクトのダウンロード元となる URL を記述します。さらに、SHA-512 チェックサムを指定して、アーティファクトを展開する前に検証することもできます。

新しいイメージを自動的に作成する Kafka Connect の設定例

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

–

コネクターの管理
KafkaConnector リソースまたは Kafka Connect REST API を使用して、Kafka Connect クラスターでコネクターインスタンスを作成および管理できます。KafkaConnector リソースでは OpenShift ネイティブな方法が提供され、Cluster Operator によって管理されます。

KafkaConnector リソースの **spec** では、コネクタークラスと設定、およびデータを処理するコネクター **タスク** の最大数を指定します。

KafkaConnector 設定の YAML 例

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

アノテーションを **KafkaConnect** リソースに追加して、KafkaConnector を有効にします。
KafkaConnector リソースは、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。

KafkaConnector を有効にするアノテーションの YAML 例

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

5.6. 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** リクエストおよび **プリフライト** リクエストが可能です。

- シンプルなリクエストは、そのヘッダーで許可されるオリジンを定義する必要のある HTTP リクエストです。
- プリフライトリクエストでは、オリジンとメソッドが許可されることを確認するために、実際のリクエストの前に初期 OPTIONS HTTP リクエストが送信されます。

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
  # ...
```

その他のリソース

- Fetch CORS 仕様

第6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

6.1. 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

また、Kafka ブローカーのリスナーに TLS 暗号化を適用して、Kafka ブローカーとクライアント間で TLS を設定することもできます。外部リスナーの設定時に外部クライアントに TLS を指定します。

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、TLS 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 認証局 (CA) は、証明書を発行してコンポーネントのアイデンティティーを認証します。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 (CA) 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 (CA) に対して検証されます。

6.2. 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:

- 相互 TLS クライアント認証 (TLS 暗号化が有効なリスナーの場合)
- SASL SCRAM-SHA-512

- OAuth 2.0 のトークンベースの認証

User Operator では TLS 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。

6.3. 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

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。

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リクエスト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7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タイムクリティカルな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を基づく 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** 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** は、さらに Prometheus メトリクスを追加します。
- **Grafana** は、ダッシュボードで Prometheus メトリクスを視覚化できます。
- **Jaeger** は、分散トレースをサポートし、アプリケーション間のトランザクションをトレースします。
- **Cruise Control** は、Kafka クラスター全体に渡るデータを分散します。

その他のリソース

- Prometheus
- Kafka Exporter
- Grafana Labs
- Jaeger
- Cruise Control の Wiki

7.1. 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。

7.2. 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 インターフェースを使用してインポートされます。

7.3. 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。

7.4. 分散トレース

Kafka デプロイメントでは、以下に対して、Jaeger を使用した分散トレースがサポートされます。

- 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 Bridge が使用して生成したメッセージと、クライアントアプリケーションからの HTTP 要求をトレースする Kafka Bridge

テンプレート設定プロパティーは、Kafka リソース用に設定され、環境変数のトレースを記述します。

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。

7.5. CRUISE CONTROL

Cruise Control は、Kafka クラスター全体に渡るデータの監視および分散を単純化するオープンソースプロジェクトです。Cruise Control は Kafka クラスターと共にデプロイされ、そのトラフィックを監視し、よりバランスの取れたパーティション割り当てを提案し、その提案をもとにしたパーティション再割り当てのトリガーになります。

Control はリソースの使用状況に関する情報を収集し、Kafka クラスターのワークロードをモデル化および分析します。定義済みの **最適化ゴール** を基にして、Cruise Control はクラスターを効果的にリバラ

ンスする方法に関する **最適化プロポーザル** を生成します。**最適化プロポーザル** が承認されると、Cruise Control はプロポーザルに示されるリバランスを適用します。

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. **サブスクリプション** 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 **Red Hat AMQ Streams**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。

改訂日時： 2021-06-07 00:19:52 UTC