



Red Hat AMQ Streams 2.3

AMQ Streams Kafka Bridge の使用

AMQ Streams Kafka Bridge を使用した Kafka クラスターへの接続

Red Hat AMQ Streams 2.3 AMQ Streams Kafka Bridge の使用

AMQ Streams Kafka Bridge を使用した Kafka クラスターへの接続

法律上の通知

Copyright © 2023 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux[®] is the registered trademark of Linus Torvalds in the United States and other countries.

Java[®] is a registered trademark of Oracle and/or its affiliates.

XFS[®] is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL[®] is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js[®] is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack[®] Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

概要

AMQ Streams 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。

目次

多様性を受け入れるオープンソースの強化	3
第1章 KAFKA BRIDGE の概要	4
1.1. KAFKA BRIDGE の実行	4
1.2. KAFKA BRIDGE インターフェイス	5
1.3. KAFKA BRIDGE OPENAPI 仕様	6
1.4. KAFKA クラスターへの接続の保護	6
1.5. KAFKA BRIDGE HTTP インターフェイスの保護	6
1.6. KAFKA BRIDGE へのリクエスト	7
1.7. CORS	9
1.8. KAFKA BRIDGE のロガーの設定	11
第2章 KAFKA BRIDGE クイックスタート	13
2.1. KAFKA BRIDGE アーカイブのダウンロード	13
2.2. KAFKA BRIDGE のインストール	13
2.3. トピックおよびパーティションへのメッセージの作成	14
2.4. KAFKA BRIDGE コンシューマーの作成	20
2.5. KAFKA BRIDGE コンシューマーのトピックへのサブスクライブ	21
2.6. KAFKA BRIDGE コンシューマーからの最新メッセージの取得	22
2.7. ログへのオフセットのコミット	23
2.8. パーティションのオフセットのシーク	23
2.9. KAFKA BRIDGE コンシューマーの削除	25
第3章 KAFKA BRIDGE の設定	26
3.1. KAFKA BRIDGE プロパティの設定	26
3.2. 分散トレースの設定	27
第4章 AMQ STREAMS KAFKA BRIDGE API リファレンス	30
4.1. 概要	30
4.2. 定義	30
4.3. パス	36
付録A サブスクリプションの使用	64
アカウントへのアクセス	64
サブスクリプションのアクティベート	64
Zip および Tar ファイルのダウンロード	64
DNF を使用したパッケージのインストール	64

多様性を受け入れるオープンソースの強化

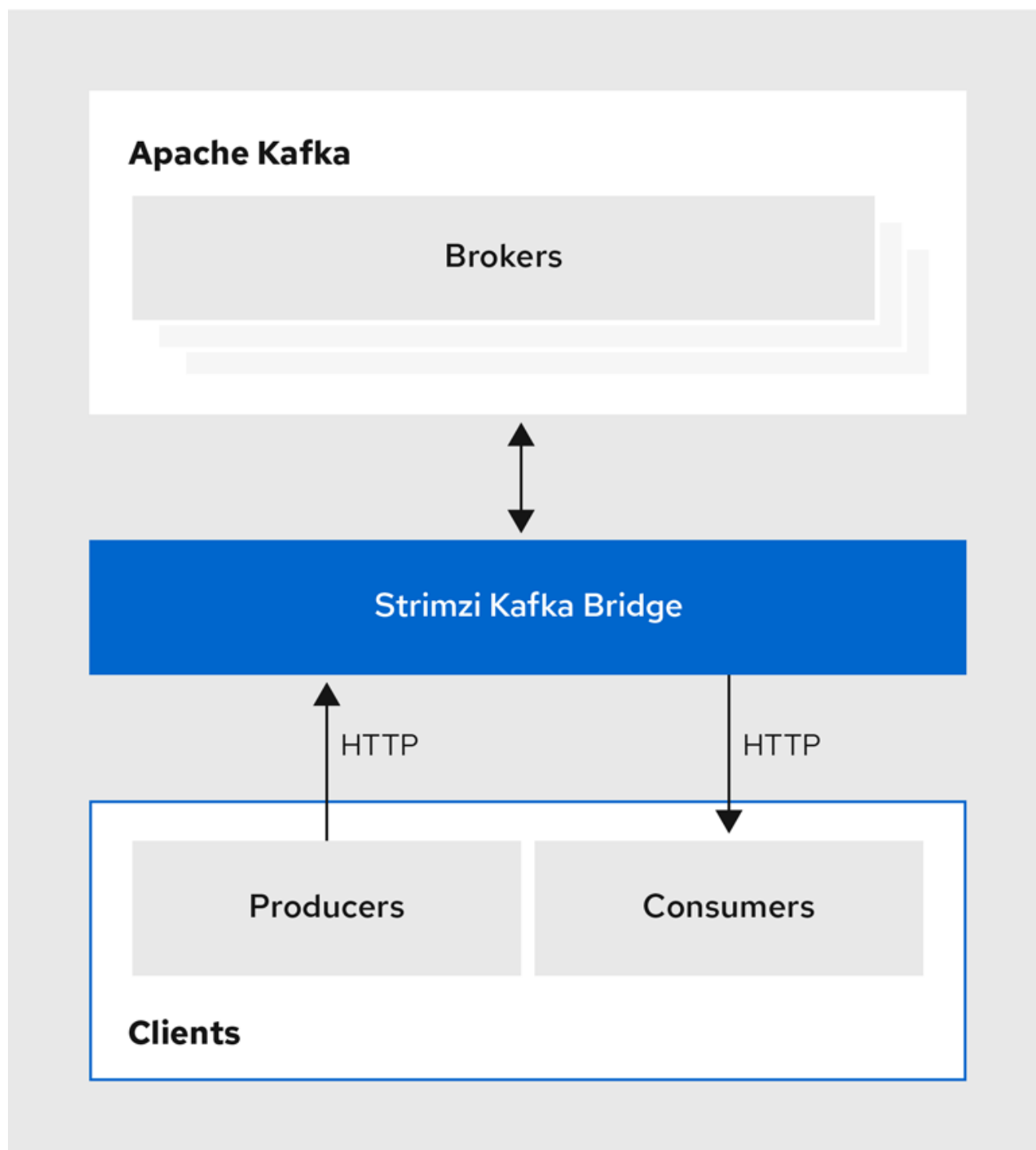
Red Hat では、コード、ドキュメント、Web プロパティにおける配慮に欠ける用語の置き換えに取り組んでいます。まずは、マスター (master)、スレーブ (slave)、ブラックリスト (blacklist)、ホワイトリスト (whitelist) の 4 つの用語の置き換えから始めます。この取り組みは膨大な作業を要するため、今後の複数のリリースで段階的に用語の置き換えを実施して参ります。詳細は、[Red Hat CTO である Chris Wright のメッセージ](#) をご覧ください。

第1章 KAFKA BRIDGE の概要

AMQ Streams Kafka Bridge を使用して、Kafka クラスターに HTTP リクエストを送信します。

Kafka Bridge を使用して、HTTP クライアントアプリケーションを Kafka クラスターと統合できます。

HTTP クライアント統合



1.1. KAFKA BRIDGE の実行

AMQ Streams Kafka Bridge をインストールして、Kafka クラスターと同じ環境で実行します。

Kafka Bridge インストールアーティファクトをダウンロードしてホストマシンに追加できます。ローカル環境で Kafka Bridge を試すには、[Kafka Bridge クイックスタート](#) を参照してください。

1.1.1. OpenShift での Kafka Bridge の実行

OpenShift に AMQ Streams をデプロイした場合は、AMQ Streams Cluster Operator を使用して、Kafka Bridge を OpenShift クラスターにデプロイできます。Kafka Bridge を **KafkaBridge** リソースとして設定およびデプロイします。Cluster Operator によって OpenShift namespace にデプロイされた実行中の Kafka クラスターが必要です。OpenShift クラスターの外部で Kafka Bridge にアクセスするようにデプロイメントを設定できます。

Kafka Bridge を **KafkaBridge** リソースとしてデプロイおよび設定する方法は、[AMQ Streams のドキュメント](#) を参照してください。

1.2. 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。

1.2.1. 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 レスポンスと HTTP レスポンス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

- [AMQ Streams Kafka Bridge API リファレンス](#)

1.3. KAFKA BRIDGE OPENAPI 仕様

Kafka Bridge API は、OpenAPI 仕様 (OAS) を使用します。OAS は、HTTP API を記述および実装するための標準フレームワークを提供します。

Kafka Bridge OpenAPI 仕様は JSON 形式です。OpenAPI JSON ファイルは、Kafka Bridge ソースダウンロードファイルの **src/main/resources/** フォルダにあります。ダウンロードファイルは、[カスタマーポータル](#) から入手できます。

[GET/openapi](#) メソッドを使用して、OpenAPI v2 仕様を JSON 形式で取得することもできます。

関連情報

- [OpenAPI イニシアチブ](#)

1.4. KAFKA クラスターへの接続の保護

Kafka Bridge と Kafka クラスターの間で以下を設定できます。

- TLS または SASL ベースの認証
- TLS 暗号化接続

[プロパティファイル](#) を使用して、認証用に Kafka Bridge を設定します。

また、Kafka ブローカーで ACL を使用して、Kafka Bridge での消費および生成が可能なトピックを制限できます。



注記

[OpenShift で Kafka Bridge を実行している](#) 場合は、**KafkaBridge** リソースを使用して認証を設定します。

1.5. KAFKA BRIDGE HTTP インターフェイスの保護

Kafka Bridge では、HTTP クライアントと Kafka Bridge の間の認証と暗号化は直接サポートされていません。クライアントから Kafka Bridge に送信されるリクエストは、認証または暗号化なしで送信されます。リクエストには、HTTPS ではなく HTTP を使用する必要があります。

Kafka Bridge は、次のツールと組み合わせて、保護できます。

- Kafka Bridge にアクセスできる Pod を定義するネットワークポリシーとファイアウォール
- リバースプロキシ (OAuth 2.0 など)
- API ゲートウェイ

1.6. KAFKA BRIDGE へのリクエスト

データ形式と HTTP ヘッダーを指定し、有効なリクエストが Kafka Bridge に送信されるようにします。

1.6.1. Content-Type ヘッダー

API リクエストおよびレスポンス本文は、常に JSON としてエンコードされます。

- コンシューマー操作の実行時に、**POST** リクエストの本文が空でない場合は、以下の **Content-Type** ヘッダーが含まれている必要があります。

Content-Type: application/vnd.kafka.v2+json

- プロデューサー操作を行う場合に、**POST** リクエストには、生成されるメッセージの **埋め込みデータ形式** を示す **Content-Type** ヘッダーを指定する必要があります。これは **json** または **binary** のいずれかになります。

埋め込みデータ形式	Content-Type ヘッダー
JSON	Content-Type: application/vnd.kafka.json.v2+json
バイナリー	Content-Type: application/vnd.kafka.binary.v2+json

次のセクションで説明されているように、埋め込みデータ形式はコンシューマーごとに設定されます。

POST リクエストの本文が空の場合は、**Content-Type** を設定しないでください。空の本文を使用して、デフォルト値のコンシューマーを作成できます。

1.6.2. 埋め込みデータ形式

埋め込みデータ形式は、Kafka メッセージが Kafka Bridge によりプロデューサーからコンシューマーに HTTP で送信される際の形式です。サポート対象の埋め込みデータ形式には、JSON とバイナリーの 2 種類があります。

`/consumers/groupid` エンドポイントを使用してコンシューマーを作成する場合、**POST** リクエスト本文で JSON またはバイナリーいずれかの埋め込みデータ形式を指定する必要があります。これは、以下の例のように **format** フィールドで指定します。

```
{
  "name": "my-consumer",
  "format": "binary", ①
  # ...
}
```

- ① バイナリー埋め込みデータ形式。

コンシューマーの作成時に指定する埋め込みデータ形式は、コンシューマーが消費する Kafka メッセージのデータ形式と一致する必要があります。

バイナリー埋め込みデータ形式を指定する場合は、以降のプロデューサーリクエストで、リクエスト本文にバイナリーデータが Base64 でエンコードされた文字列として含まれる必要があります。たとえば、`/topics/topicname` エンドポイントを使用してメッセージを送信する場合は、**records.value** を

Base64 でエンコードする必要があります。

```
{
  "records": [
    {
      "key": "my-key",
      "value": "ZWR3YXJkdGhldGhyZWVsZWdnZWRjYXQ="
    },
  ]
}
```

プロデューサーリクエストには、埋め込みデータ形式に対応する **Content-Type** ヘッダーも含まれる必要があります (例: **Content-Type: application/vnd.kafka.binary.v2+json**)。

1.6.3. メッセージの形式

/topics エンドポイントを使用してメッセージを送信する場合は、**records** パラメーターのリクエスト本文にメッセージペイロードを入力します。

records パラメーターには、以下のオプションフィールドを含めることができます。

- Message **headers**
- Message **key**
- Message **value**
- Destination **partition**

/topics への **POST** リクエストの例

```
curl -X POST \
  http://localhost:8080/topics/my-topic \
  -H 'content-type: application/vnd.kafka.json.v2+json' \
  -d '{
    "records": [
      {
        "key": "my-key",
        "value": "sales-lead-0001"
        "partition": 2
        "headers": [
          {
            "key": "key1",
            "value": "QXBhY2hIEthZmthIGlzIHRoZSBib21iIQ==" ❶
          }
        ]
      }
    ]
  }'
```

❶ バイナリー形式のヘッダー値。Base64 としてエンコードされます。

1.6.4. Accept ヘッダー

コンシューマーを作成したら、以降のすべての GET リクエストには **Accept** ヘッダーが以下のような形式で含まれる必要があります。

```
Accept: application/vnd.kafka.EMBEDDED-DATA-FORMAT.v2+json
```

EMBEDDED-DATA-FORMAT は **json** または **binary** です。

たとえば、サブスクリブされたコンシューマーのレコードを JSON 埋め込みデータ形式で取得する場合、この Accept ヘッダーが含まれるようにします。

```
Accept: application/vnd.kafka.json.v2+json
```

1.7. CORS

通常、HTTP クライアントは、異なるドメイン間でリクエストを発行することはできません。

たとえば、Kafka クラスターとともにデプロイした Kafka Bridge が、**http://my-bridge.io** ドメインを使用してアクセスできるとします。HTTP クライアントは URL を使用して Kafka Bridge と対話し、Kafka クラスターを介してメッセージを交換できます。ただし、クライアントは **http://my-web-application.io** ドメインで Web アプリケーションとして実行されています。クライアント (ソース) ドメインは Kafka Bridge (ターゲット) ドメインとは異なります。クライアントからのリクエストは、same-origin ポリシーの制限が原因で、失敗します。Cross-Origin Resource Sharing (CORS) を使用すると、この状況を回避できます。

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、**GET**、**HEAD**、**POST** のメソッドを使った標準的なリクエストに適しています。

プリフライトリクエストは、実際のリクエストが安全に送信できることを確認する最初のチェックとして **HTTP OPTIONS** リクエストを送信します。確認時に、実際のリクエストが送信されます。プリフライトリクエストは、**PUT** や **DELETE** など、より高い安全性が求められるメソッドや、非標準のヘッダーを使用するメソッドに適しています。

すべてのリクエストには、HTTP リクエストのソースであるヘッダーの **origins** 値が必要です。

CORS を使用すると、Kafka Bridge HTTP の設定で Kafka クラスターへのアクセスに使用可能なメソッドおよび元の URL を指定できます。

Kafka Bridge の CORS 設定例

```
# ...
http.cors.enabled=true
http.cors.allowedOrigins=http://my-web-application.io
http.cors.allowedMethods=GET,POST,PUT,DELETE,OPTIONS,PATCH
```

1.7.1. シンプルなリクエスト

たとえば、この単純なリクエストヘッダーは、元の URL を **http://my-web-application.io** と指定します。

```
Origin: http://my-web-application.io
```

ヘッダー情報は、レコード消費のリクエストに追加されます。

```
curl -v -X GET HTTP-BRIDGE-ADDRESS/consumers/my-group/instances/my-consumer/records \
-H 'Origin: http://my-web-application.io' \
-H 'content-type: application/vnd.kafka.v2+json'
```

Kafka Bridge からのレスポンスでは、**Access-Control-Allow-Origin** ヘッダーが返されます。これには、HTTP リクエストをブリッジに発行できるドメインのリストが含まれています。

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: * 1
```

- 1** アスタリスク (*) が返されると、どのドメインからでもリソースにアクセスできるという意味です。

1.7.2. プリフライトリクエスト

最初のプリフライトリクエストは、**OPTIONS** メソッドを使って Kafka Bridge に送信されます。**HTTP OPTIONS** リクエストはヘッダー情報を送信し、Kafka Bridge が実際のリクエストを許可することを確認します。

ここで、プリフライトリクエストは、**http://my-web-application.io** からの **POST** リクエストが有効であることを確認します。

```
OPTIONS /my-group/instances/my-consumer/subscription HTTP/1.1
Origin: http://my-web-application.io
Access-Control-Request-Method: POST 1
Access-Control-Request-Headers: Content-Type 2
```

- 1** Kafka Bridge では、実際のリクエストが **POST** リクエストであると警告が送信されます。
- 2** 実際のリクエストは **Content-Type** ヘッダーと共に送信されます。

OPTIONS は、プリフライトリクエストのヘッダー情報に追加されます。

```
curl -v -X OPTIONS -H 'Origin: http://my-web-application.io' \
-H 'Access-Control-Request-Method: POST' \
-H 'content-type: application/vnd.kafka.v2+json'
```

Kafka Bridge は最初のリクエストにレスポンスし、リクエストが受け入れられることを確認します。レスポンスヘッダーは、許可されるオリジン、メソッド、およびヘッダーを返します。

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: http://my-web-application.io
Access-Control-Allow-Methods: GET,POST,PUT,DELETE,OPTIONS,PATCH
Access-Control-Allow-Headers: content-type
```

オリジンまたはメソッドが拒否されると、エラーメッセージが返されます。

プリフライトリクエストで確認されたため、実際のリクエストには **Access-Control-Request-Method** ヘッダーは必要ありませんが、元のヘッダーが必要です。

```
curl -v -X POST HTTP-BRIDGE-ADDRESS/topics/bridge-topic \  
-H 'Origin: http://my-web-application.io' \  
-H 'content-type: application/vnd.kafka.v2+json'
```

このレスポンスは、送信元 URL が許可されることを示します。

```
HTTP/1.1 200 OK  
Access-Control-Allow-Origin: http://my-web-application.io
```

関連情報

- [Fetch CORS 仕様](#)

1.8. KAFKA BRIDGE のロガーの設定

Kafka Bridge OpenAPI 仕様で定義されている操作ごとに異なるログレベルを設定できます。

操作にはそれぞれ、対応の API エンドポイントがあり、このエンドポイントを通して、ブリッジが HTTP クライアントからリクエストを受信します。各エンドポイントのログレベルを変更すると、受信および送信 HTTP リクエストに関する詳細なログ情報を作成できます。

ロガーは **log4j.properties** ファイルで定義されます。このファイルには **healthy** および **ready** エンドポイントの以下のデフォルト設定が含まれています。

```
log4j.logger.http.openapi.operation.healthy=WARN, out  
log4j.additivity.http.openapi.operation.healthy=false  
log4j.logger.http.openapi.operation.ready=WARN, out  
log4j.additivity.http.openapi.operation.ready=false
```

その他すべての操作のログレベルは、デフォルトで **INFO** に設定されます。ロガーは以下のようにフォーマットされます。

```
log4j.logger.http.openapi.operation.<operation_id>
```

<operation_id> は、特定の操作の識別子です。

Open API 仕様で定義されている操作のリスト

- **createConsumer**
- **deleteConsumer**
- **subscribe**
- **unsubscribe**
- **poll**
- **assign**
- **commit**
- **send**

- **sendToPartition**
- **seekToBeginning**
- **seekToEnd**
- **seek**
- **healthy**
- **ready**
- **openapi**

第2章 KAFKA BRIDGE クイックスタート

このクイックスタートを使用して、ローカルの開発環境で AMQ Streams の Kafka Bridge を試すことができます。

次の方法を学習します。

- Kafka クラスターのトピックおよびパーティションへのメッセージを生成する。
- Kafka Bridge コンシューマーを作成する。
- 基本的なコンシューマー操作を実行する (たとえば、コンシューマーをトピックにサブスクライブする、生成したメッセージを取得するなど)。

このクイックスタートでは、HTTP リクエストはターミナルにコピーアンドペーストできる curl コマンドを使用します。

前提条件を確認し、本章に指定されている順序でタスクを行うようにしてください。

このクイックスタートでは、JSON 形式でメッセージを生成および消費します。

クイックスタートの前提条件

- Kafka クラスターがホストマシンで実行している。

2.1. KAFKA BRIDGE アーカイブのダウンロード

AMQ Streams Kafka Bridge の zip 形式のディストリビューションをダウンロードできます。

手順

- [カスタマーポータル](#) から、最新バージョンの AMQ Streams Kafka Bridge アーカイブをダウンロードします。

2.2. KAFKA BRIDGE のインストール

Kafka Bridge アーカイブで提供されるスクリプトを使用して、Kafka Bridge をインストールします。インストールアーカイブで提供される **application.properties** ファイルは、デフォルト設定を提供します。

次のデフォルトプロパティの値は、Kafka Bridge がポート 8080 でリクエストをリッスンするように設定します。

デフォルトの設定プロパティ

```
http.enabled=true  
http.host=0.0.0.0  
http.port=8080
```

前提条件

- Kafka Bridge インストールアーカイブがダウンロードされている。

手順

1. まだ行っていない場合は、Kafka Bridge インストールアーカイブを任意のディレクトリーに展開します。
2. 設定プロパティをパラメーターとして使用して、Kafka Bridge スクリプトを実行します。以下に例を示します。

```
./bin/kafka_bridge_run.sh --config-file=<path>/application.properties
```

3. インストールが成功したことをログで確認します。

```
HTTP-Kafka Bridge started and listening on port 8080
HTTP-Kafka Bridge bootstrap servers localhost:9092
```

次のステップ

- [トピックおよびパーティションへのメッセージを生成します。](#)

2.3. トピックおよびパーティションへのメッセージの作成

Kafka Bridge で、トピックエンドポイントを使用して Kafka トピックへのメッセージを JSON 形式で生成します。

`topics` エンドポイントを使用して、トピックへのメッセージを JSON 形式で生成できます。リクエスト本文でメッセージの宛先パーティションを指定できます。`partitions` エンドポイントは、すべてのメッセージに対して単一の宛先パーティションをパスパラメーターとして指定するための代替方法を提供します。

この手順では、メッセージは **bridge-quickstart-topic** と呼ばれるトピックに生成されます。

前提条件

- Kafka クラスタにはトピックが1つあり、そのトピックは3つのパーティションに分割されている。

kafka-topics.sh ユーティリティーを使用してトピックを作成できます。

3つのパーティションを使用したトピック作成の例

```
bin/kafka-topics.sh --bootstrap-server localhost:9092 --create --topic bridge-quickstart-topic -
-partitions 3 --replication-factor 1
```

トピックが作成されたことの確認

```
bin/kafka-topics.sh --bootstrap-server localhost:9092 --describe --topic bridge-quickstart-
topic
```



注記

OpenShift に AMQ Streams をデプロイした場合は、**KafkaTopic** カスタムリソースを使用してトピックを作成できます。

手順

1. Kafka Bridge を使用して、作成したトピックに3つのメッセージを生成します。

```
curl -X POST \
  http://localhost:8080/topics/bridge-quickstart-topic \
  -H 'content-type: application/vnd.kafka.json.v2+json' \
  -d '{
    "records": [
      {
        "key": "my-key",
        "value": "sales-lead-0001"
      },
      {
        "value": "sales-lead-0002",
        "partition": 2
      },
      {
        "value": "sales-lead-0003"
      }
    ]
  }'
```

- **sales-lead-0001** は、キーのハッシュに基づいてパーティションに送信されます。
 - **sales-lead-0002** は、パーティション2に直接送信されます。
 - **sales-lead-0003** は、ラウンドロビン方式を使用して **bridge-quickstart-topic** トピックのパーティションに送信されます。
2. リクエストが正常に行われると、Kafka Bridge は **offsets** 配列を **200** コードと **application/vnd.kafka.v2+json** の **content-type** ヘッダーとともに返します。各メッセージで、**offsets** 配列は以下を記述します。
- メッセージが送信されたパーティション。
 - パーティションの現在のメッセージオフセット。

レスポンス例

```
#...
{
  "offsets":[
    {
      "partition":0,
      "offset":0
    },
    {
      "partition":2,
      "offset":0
    },
    {
      "partition":0,
      "offset":1
    }
  ]
}
```

```

    }
  ]
}

```

追加のトピックリクエスト

他の curl リクエストを実行して、トピックおよびパーティションに関する情報を見つけます。

トピックの一覧表示

```

curl -X GET \
  http://localhost:8080/topics

```

レスポンス例

```

[
  "__strimzi_store_topic",
  "__strimzi-topic-operator-kstreams-topic-store-changelog",
  "bridge-quickstart-topic",
  "my-topic"
]

```

トピック設定およびパーティションの詳細の取得

```

curl -X GET \
  http://localhost:8080/topics/bridge-quickstart-topic

```

レスポンス例

```

{
  "name": "bridge-quickstart-topic",
  "configs": {
    "compression.type": "producer",
    "leader.replication.throttled.replicas": "",
    "min.insync.replicas": "1",
    "message.downconversion.enable": "true",
    "segment.jitter.ms": "0",
    "cleanup.policy": "delete",
    "flush.ms": "9223372036854775807",
    "follower.replication.throttled.replicas": "",
    "segment.bytes": "1073741824",
    "retention.ms": "604800000",
    "flush.messages": "9223372036854775807",
    "message.format.version": "2.8-IV1",
    "max.compaction.lag.ms": "9223372036854775807",
    "file.delete.delay.ms": "60000",
    "max.message.bytes": "1048588",
    "min.compaction.lag.ms": "0",
    "message.timestamp.type": "CreateTime",
    "preallocate": "false",
    "index.interval.bytes": "4096",
    "min.cleanable.dirty.ratio": "0.5",
    "unclean.leader.election.enable": "false",
    "retention.bytes": "-1",

```

```
"delete.retention.ms": "86400000",
"segment.ms": "604800000",
"message.timestamp.difference.max.ms": "9223372036854775807",
"segment.index.bytes": "10485760"
},
"partitions": [
  {
    "partition": 0,
    "leader": 0,
    "replicas": [
      {
        "broker": 0,
        "leader": true,
        "in_sync": true
      },
      {
        "broker": 1,
        "leader": false,
        "in_sync": true
      },
      {
        "broker": 2,
        "leader": false,
        "in_sync": true
      }
    ]
  },
  {
    "partition": 1,
    "leader": 2,
    "replicas": [
      {
        "broker": 2,
        "leader": true,
        "in_sync": true
      },
      {
        "broker": 0,
        "leader": false,
        "in_sync": true
      },
      {
        "broker": 1,
        "leader": false,
        "in_sync": true
      }
    ]
  },
  {
    "partition": 2,
    "leader": 1,
    "replicas": [
      {
        "broker": 1,
        "leader": true,
        "in_sync": true
      }
    ]
  }
]
```

```
    },
    {
      "broker": 2,
      "leader": false,
      "in_sync": true
    },
    {
      "broker": 0,
      "leader": false,
      "in_sync": true
    }
  ]
}
]
```

特定のトピックのパーティションの一覧表示

```
curl -X GET \
  http://localhost:8080/topics/bridge-quickstart-topic/partitions
```

レスポンス例

```
[
  {
    "partition": 0,
    "leader": 0,
    "replicas": [
      {
        "broker": 0,
        "leader": true,
        "in_sync": true
      },
      {
        "broker": 1,
        "leader": false,
        "in_sync": true
      },
      {
        "broker": 2,
        "leader": false,
        "in_sync": true
      }
    ]
  },
  {
    "partition": 1,
    "leader": 2,
    "replicas": [
      {
        "broker": 2,
        "leader": true,
        "in_sync": true
      },
      {
```

```
    "broker": 0,  
    "leader": false,  
    "in_sync": true  
  },  
  {  
    "broker": 1,  
    "leader": false,  
    "in_sync": true  
  }  
]  
},  
{  
  "partition": 2,  
  "leader": 1,  
  "replicas": [  
    {  
      "broker": 1,  
      "leader": true,  
      "in_sync": true  
    },  
    {  
      "broker": 2,  
      "leader": false,  
      "in_sync": true  
    },  
    {  
      "broker": 0,  
      "leader": false,  
      "in_sync": true  
    }  
  ]  
}  
]
```

特定のトピックパーティションの詳細の一覧表示

```
curl -X GET \  
http://localhost:8080/topics/bridge-quickstart-topic/partitions/0
```

レスポンス例

```
{  
  "partition": 0,  
  "leader": 0,  
  "replicas": [  
    {  
      "broker": 0,  
      "leader": true,  
      "in_sync": true  
    },  
    {  
      "broker": 1,  
      "leader": false,  
      "in_sync": true  
    }  
  ],  
}
```

```
{
  "broker": 2,
  "leader": false,
  "in_sync": true
}
]
```

特定のトピックパーティションのオフセットの一覧表示

```
curl -X GET \
  http://localhost:8080/topics/bridge-quickstart-topic/partitions/0/offsets
```

レスポンス例

```
{
  "beginning_offset": 0,
  "end_offset": 1
}
```

次のステップ

トピックおよびパーティションへのメッセージを作成したら、[Kafka Bridge コンシューマーを作成します](#)。

関連情報

- [POST /topics/{topicname}](#)
- [POST /topics/{topicname}/partitions/{partitionid}](#)

2.4. KAFKA BRIDGE コンシューマーの作成

Kafka クラスターでコンシューマー操作を実行するには、まず [consumers](#) エンドポイントを使用してコンシューマーを作成する必要があります。コンシューマーは [Kafka Bridge コンシューマー](#) と呼ばれます。

手順

1. **bridge-quickstart-consumer-group** という名前の新しいコンシューマーグループに Kafka Bridge コンシューマーを作成します。

```
curl -X POST http://localhost:8080/consumers/bridge-quickstart-consumer-group \
  -H 'content-type: application/vnd.kafka.v2+json' \
  -d '{
    "name": "bridge-quickstart-consumer",
    "auto.offset.reset": "earliest",
    "format": "json",
    "enable.auto.commit": false,
    "fetch.min.bytes": 512,
    "consumer.request.timeout.ms": 30000
  }'
```


- コンシューマーには **bridge-quickstart-consumer** という名前を付け、埋め込みデータ形式は **json** として設定します。
- 一部の基本的な設定が定義されます。
- コンシューマーはログへのオフセットに自動でコミットしません。これは、**enable.auto.commit** が **false** に設定されているからです。このクイックスタートでは、オフセットを後で手作業でコミットします。
リクエストが正常に行われると、Kafka Bridge はレスポンス本文でコンシューマー ID (**instance_id**) とベース URL (**base_uri**) を **200** コードとともに返します。

レスポンス例

```
#...
{
  "instance_id": "bridge-quickstart-consumer",
  "base_uri": "http://<bridge_id>-bridge-service:8080/consumers/bridge-quickstart-consumer-group/instances/bridge-quickstart-consumer"
}
```

2. ベース URL (**base_uri**) をコピーし、このクイックスタートの他のコンシューマー操作で使用します。

次のステップ

Kafka Bridge コンシューマーが作成されたので、[このコンシューマーをトピックにサブスクライブできます](#)。

関連情報

- [POST /consumers/{groupid}](#)

2.5. KAFKA BRIDGE コンシューマーのトピックへのサブスクライブ

Kafka Bridge コンシューマーを作成したら、[subscription](#) エンドポイントを使用して1つ以上のトピックにサブスクライブします。サブスクライブすると、コンシューマーはトピックに生成されたすべてのメッセージの受信を開始します。

手順

- 前述の [トピックおよびパーティションへのメッセージの作成](#) の手順ですすでに作成した **bridge-quickstart-topic** トピックに、コンシューマーをサブスクライブします。

```
curl -X POST http://localhost:8080/consumers/bridge-quickstart-consumer-group/instances/bridge-quickstart-consumer/subscription \
-H 'content-type: application/vnd.kafka.v2+json' \
-d '{
  "topics": [
    "bridge-quickstart-topic"
  ]
}'
```

topics 配列には、例のような単一のトピック、または複数のトピックを含めることができます。正規表現に一致する複数のトピックにコンシューマーをサブスクライブする場合は、**topics** 配列の代わりに **topic_pattern** 文字列を使用できます。

リクエストが正常に行われると、Kafka Bridge によって **204** (No Content) コードのみが返されます。

次のステップ

Kafka Bridge コンシューマーをトピックにサブスクライブしたら、[コンシューマーからメッセージを取得](#) できます。

関連情報

- [POST /consumers/{groupid}/instances/{name}/subscription](#)

2.6. KAFKA BRIDGE コンシューマーからの最新メッセージの取得

`records` エンドポイントからデータを要求して、Kafka Bridge コンシューマーから最新のメッセージを取得します。実稼働環境では、HTTP クライアントはこのエンドポイントを繰り返し (ループで) 呼び出すことができます。

手順

1. [トピックおよびパーティションへのメッセージの作成](#) の説明に従い、Kafka Bridge コンシューマーに新たなメッセージを生成します。
2. **GET** リクエストを `records` エンドポイントに送信します。

```
curl -X GET http://localhost:8080/consumers/bridge-quickstart-consumer-
group/instances/bridge-quickstart-consumer/records \
-H 'accept: application/vnd.kafka.json.v2+json'
```

Kafka Bridge コンシューマーを作成し、サブスクライブすると、最初の GET リクエストによって空のレスポンスが返されます。これは、ポーリング操作がリバランスプロセスを開始してパーティションを割り当てるからです。

3. 手順 2 を繰り返し、Kafka Bridge コンシューマーからメッセージを取得します。Kafka Bridge は、レスポンス本文でメッセージの配列 (トピック名、キー、値、パーティション、オフセットの記述) を **200** コードとともに返します。メッセージはデフォルトで最新のオフセットから取得されます。

```
HTTP/1.1 200 OK
content-type: application/vnd.kafka.json.v2+json
#...
[
  {
    "topic": "bridge-quickstart-topic",
    "key": "my-key",
    "value": "sales-lead-0001",
    "partition": 0,
    "offset": 0
  },
  {
    "topic": "bridge-quickstart-topic",
    "key": null,
    "value": "sales-lead-0003",
    "partition": 0,
```

```
"offset":1
},
#...
```



注記

空のレスポンスが返される場合は、[トピックおよびパーティションへのメッセージの作成](#)の説明に従い、コンシューマーに対して追加のレコードを生成し、メッセージの取得を再試行します。

次のステップ

Kafka Bridge コンシューマーからメッセージを取得したら、[ログへのオフセットのコミット](#)を行います。

関連情報

- [GET /consumers/{groupid}/instances/{name}/records](#)

2.7. ログへのオフセットのコミット

`offsets` エンドポイントを使用して、Kafka Bridge コンシューマーによって受信されるすべてのメッセージに対して、手動でオフセットをログにコミットします。この操作が必要なのは、前述の [Kafka Bridge コンシューマーの作成](#) で作成した Kafka Bridge コンシューマーが `enable.auto.commit` の設定で `false` に指定されているからです。

手順

- `bridge-quickstart-consumer` のオフセットをログにコミットします。

```
curl -X POST http://localhost:8080/consumers/bridge-quickstart-consumer-
group/instances/bridge-quickstart-consumer/offsets
```

リクエスト本文は送信されないため、オフセットはコンシューマーによって受信されたすべてのレコードに対してコミットされます。あるいは、リクエスト本文に、オフセットをコミットするトピックとパーティションを指定する配列 (`OffsetCommitSeekList`) を含めることもできます。

リクエストが正常に行われると、Kafka Bridge は **204** コードのみを返します。

次のステップ

オフセットをログにコミットしたら、[オフセットのシーク](#)用のエンドポイントを試してみてください。

関連情報

- [POST /consumers/{groupid}/instances/{name}/offsets](#)

2.8. パーティションのオフセットのシーク

`positions` エンドポイントを使用して、Kafka Bridge コンシューマーを設定し、パーティションのメッセージを特定のオフセットから取得し、さらに最新のオフセットから取得します。これは Apache Kafka では、シーク操作と呼ばれます。

手順

1. **quickstart-bridge-topic** トピックで、パーティション 0 の特定のオフセットをシークします。

```
curl -X POST http://localhost:8080/consumers/bridge-quickstart-consumer-
group/instances/bridge-quickstart-consumer/positions \
-H 'content-type: application/vnd.kafka.v2+json' \
-d '{
  "offsets": [
    {
      "topic": "bridge-quickstart-topic",
      "partition": 0,
      "offset": 2
    }
  ]
}'
```

リクエストが正常に行われると、Kafka Bridge は **204** コードのみを返します。

2. **GET** リクエストを **records** エンドポイントに送信します。

```
curl -X GET http://localhost:8080/consumers/bridge-quickstart-consumer-
group/instances/bridge-quickstart-consumer/records \
-H 'accept: application/vnd.kafka.json.v2+json'
```

Kafka Bridge は、シークしたオフセットからのメッセージを返します。

3. 同じパーティションの最後のオフセットをシークし、デフォルトのメッセージ取得動作を復元します。今回は、[positions/end](#) エンドポイントを使用します。

```
curl -X POST http://localhost:8080/consumers/bridge-quickstart-consumer-
group/instances/bridge-quickstart-consumer/positions/end \
-H 'content-type: application/vnd.kafka.v2+json' \
-d '{
  "partitions": [
    {
      "topic": "bridge-quickstart-topic",
      "partition": 0
    }
  ]
}'
```

リクエストが正常に行われると、Kafka Bridge は別の **204** コードを返します。



注記

[positions/beginning](#) エンドポイントを使用して、1つ以上のパーティションの最初のオフセットをシークすることもできます。

次のステップ

このクイックスタートでは、AMQ Streams Kafka Bridge を使用して Kafka クラスターの一般的な操作をいくつか実行しました。これで、先ほど作成した [Kafka Bridge コンシューマーの削除](#) が可能になります。

関連情報

- [POST /consumers/{groupid}/instances/{name}/positions](#)
- [POST /consumers/{groupid}/instances/{name}/positions/beginning](#)
- [POST /consumers/{groupid}/instances/{name}/positions/end](#)

2.9. KAFKA BRIDGE コンシューマーの削除

このクイックスタートで使用した Kafa Bridge コンシューマーを削除します。

手順

- **DELETE** リクエストを `instances` エンドポイントに送信して、Kafka Bridge コンシューマーを削除します。

```
curl -X DELETE http://localhost:8080/consumers/bridge-quickstart-consumer-group/instances/bridge-quickstart-consumer
```

リクエストが正常に行われると、Kafka Bridge は **204** コードを返します。

関連情報

- [DELETE /consumers/{groupid}/instances/{name}](#)

第3章 KAFKA BRIDGE の設定

設定プロパティを使用して Kafka Bridge のデプロイメントを設定します。Kafka を設定し、Kafka との対話に必要な HTTP 接続の詳細を指定します。設定プロパティを使用して、Kafka Bridge で分散トレースを有効にして使用することもできます。分散トレースを使用すると、分散システムのアプリケーション間におけるトランザクションの進捗を追跡できます。



注記

OpenShift で Kafka Bridge を実行しているときに **KafkaBridge** リソースを使用してプロパティを設定します。

3.1. KAFKA BRIDGE プロパティの設定

この手順では、Kafka Bridge によって使用される Kafka および HTTP 接続プロパティの設定方法を説明します。

Kafka 関連のプロパティに適切な接頭辞を使用して、他の Kafka クライアントと同様に Kafka Bridge を設定します。

- **kafka.** は、サーバー接続やセキュリティなど、プロデューサーとコンシューマーに適用される一般的な設定用です。
- **kafka.consumer.** は、コンシューマーにのみ渡されるコンシューマー固有の設定用です。
- **kafka.producer.** は、プロデューサーにのみ渡されるプロデューサー固有の設定用です。

HTTP プロパティは、Kafka クラスターへの HTTP アクセスを有効にする他に、CPRS (Cross-Origin Resource Sharing) により Kafka Bridge のアクセス制御を有効化または定義する機能を提供します。CORS は、複数のオリジンから指定のリソースにブラウザでアクセスできるようにする HTTP メカニズムです。CORS を設定するには、許可されるリソースオリジンのリストと、それらにアクセスする HTTP メソッドを定義します。リクエストの追加の HTTP ヘッダーには Kafka クラスターへのアクセスが許可される CORS オリジンが記述されています。

前提条件

- [Kafka Bridge インストールアーカイブ](#)がダウンロードされている。

手順

1. Kafka Bridge インストールアーカイブで提供される **application.properties** ファイルを編集します。
プロパティファイルを使用して、Kafka および HTTP 関連のプロパティを指定します。
 - a. Kafka コンシューマーおよびプロデューサーに固有のプロパティなど、標準の Kafka 関連のプロパティを設定します。
以下を使用します。
 - **kafka.bootstrap.servers:** Kafka クラスターへのホスト/ポート接続を定義する。
 - **kafka.producer.acks:** HTTP クライアントに確認を提供する。
 - **kafka.consumer.auto.offset.reset:** Kafka のオフセットのリセットを管理する方法を決定する。

Kafka プロパティの設定に関する詳細は、[Apache Kafka の Web サイト](#) を参照してください。

- b. Kafka クラスターへの HTTP アクセスを有効にするために HTTP 関連のプロパティを設定します。
以下に例を示します。

```
bridge.id=my-bridge
http.enabled=true
http.host=0.0.0.0
http.port=8080 ①
http.cors.enabled=true ②
http.cors.allowedOrigins=https://strimzi.io ③
http.cors.allowedMethods=GET,POST,PUT,DELETE,OPTIONS,PATCH ④
```

- ① 8080 番ポートで Kafka Bridge をリッスンするデフォルトの HTTP 設定。
- ② CORS を有効にするには **true** に設定します。
- ③ 許可される CORS オリジンのコンマ区切りリスト。URL または Java 正規表現を使用できます。
- ④ CORS で許可される HTTP メソッドのコンマ区切りリスト。

2. 設定ファイルを作成します。

3.2. 分散トレースの設定

分散トレースを有効にして、Kafka Bridge で使用され、生成されたメッセージと、クライアントアプリケーションからの HTTP リクエストをトレースします。

トレースを有効にするプロパティは **application.properties** ファイルにあります。分散トレースを有効にするには、以下を実行します。

- 使用するトレースを有効にするには、**bridge.tracing** プロパティの値を設定します。使用できる値は **jaeger** および **opentelemetry** です。
- トレースの環境変数を設定します。

デフォルト設定では、OpenTelemetry または OpenTracing に基づいて Jaeger トレースシステムを実装できます。

OpenTelemetry および OpenTracing は、トレースデータをメトリクスデータの **スパン** として収集するための API 仕様です。スパンは特定の操作を表します。トレースは、1つ以上のスパンのコレクションです。

トレースは、Kafka Bridge で以下の場合に生成されます。

- Kafka からコンシューマー HTTP クライアントにメッセージを送信します。
- プロデューサー HTTP クライアントからメッセージを受信して Kafka に送信します。

Jaeger は必要な API を実装し、分析用にユーザーインターフェイスでトレースデータを視覚化します。

エンドツーエンドのトレーシングを設定するために、HTTP クライアントでトレーシングを設定する必要があります。

注意

OpenTracing プロジェクトがアーカイブされ、AMQ Streams では OpenTracing のサポートが廃止されました。

前提条件

- [Kafka Bridge インストールアーカイブ](#)がダウンロードされている。

手順

1. Kafka Bridge インストールアーカイブで提供される **application.properties** ファイルを編集します。
使用するトレースを有効にするには、**bridge.tracing** プロパティを使用します。

OpenTelemetry を有効にする設定例

```
#bridge.tracing=jaeger 1  
bridge.tracing=opentelemetry 2
```

- 1 OpenTracing を有効にするためのプロパティ (非推奨)。ここではコメントはそのまま残します。
- 2 OpenTelemetry を有効にするためのプロパティはコメント解除されます。

トレースを有効にすると、Kafka Bridge スクリプトの実行時にトレースを初期化します。

2. 設定ファイルを作成します。
3. トレーシングの環境変数を設定します。

OpenTelemetry の環境変数

```
OTEL_SERVICE_NAME=my-tracing-service 1  
OTEL_EXPORTER_JAEGER_ENDPOINT=http://localhost:14250 2
```

- 1 OpenTelemetry トレーサーサービスの名前。
- 2 ポート 14250 でスパンをリスンする Jaeger コレクターエンドポイント。

OpenTracing の環境変数

```
JAEGER_SERVICE_NAME=my-jaeger-service 1  
JAEGER_AGENT_HOST=localhost 2  
JAEGER_AGENT_PORT=6831 3
```

- 1 OpenTracing Jaeger トレーサーサービスの名前。

- 2 スパンをリッスンする Jaeger エージェントと通信するためのホスト名。
 - 3 Jaeger エージェントと通信するためのポート。ポート 6831 は Jaeger エージェントによって公開されます。
4. プロパティを有効にしてトレース用に Kafka Bridge スクリプトを実行します。

OpenTelemetry を有効にした状態での Kafka Bridge の実行

```
./bin/kafka_bridge_run.sh --config-file=<path>/application.properties
```

Kafka Bridge の内部コンシューマーとプロデューサーが有効になり、トレースできるようになりました。

3.2.1. OpenTelemetry でのトレースシステムの指定

デフォルトの Jaeger システムの代わりに、OpenTelemetry でサポートされる他のトレースシステムを指定できます。

OpenTelemetry で別のトレースシステムを使用する場合は、以下の手順を実施します。

1. トレースシステムのライブラリーを Kafka クラスパスに追加します。
2. トレースシステムの名前を追加のエクスポーター環境変数として追加します。

Jaeger を使用しない場合の追加の環境変数

```
OTEL_SERVICE_NAME=my-tracing-service  
OTEL_TRACES_EXPORTER=zipkin 1  
OTEL_EXPORTER_ZIPKIN_ENDPOINT=http://localhost:9411/api/v2/spans 2
```

- 1 トレースシステムの名前。この例では、Zipkin を指定します。
- 2 スパンをリッスンする特定の選択されたエクスポーターのエンドポイント。この例では、Zipkin エンドポイントが指定されています。

関連情報

- [OpenTelemetry エクスポーターの値](#)

第4章 AMQ STREAMS KAFKA BRIDGE API リファレンス

4.1. 概要

AMQ Streams Kafka Bridge は、HTTP ベースのクライアントアプリケーションを Kafka クラスタと統合するための REST API を提供します。API を使用して、ネイティブの Kafka プロトコルではなく、HTTP を介してコンシューマーを作成および管理し、レコードを送受信できます。

4.1.1. バージョン情報

バージョン : 0.1.0

4.1.2. タグ

- **コンシューマー:** Kafka クラスタにコンシューマーを作成し、トピックのサブスクライブ、処理されたレコードの取得、オフセットのコミットなどの一般的なアクションを実行するためのコンシューマー操作。
- **プロデューサー:** 指定されたトピックまたはトピックパーティションにレコードを送信するプロデューサー操作。
- **シーク:** コンシューマーが特定のオフセット位置からメッセージの受信を開始できるようにするシーク操作。
- **トピック:** 指定されたトピックまたはトピックパーティションにメッセージを送信するトピック操作。任意で、リクエストにメッセージキーを含めます。トピックとトピックメタデータを取得することもできます。

4.1.3. 消費されるアイテム

- **application/json**

4.1.4. 生成されるアイテム

- **application/json**

4.2. 定義

4.2.1. AssignedTopicPartitions

型 : < string, < integer (int32) > array > マップ

4.2.2. BridgeInfo

Kafka Bridge インスタンスに関する情報。

名前	スキーマ
bridge_version optional	string

4.2.3. Consumer

名前	説明	スキーマ
auto.offset.reset optional	コンシューマーのオフセットの位置をリセットします。 latest (デフォルト) に設定すると、メッセージは最新のオフセットから読み取られます。 earliest に設定すると、メッセージは最初のオフセットから読み込まれます。	string
consumer.request.timeout.ms optional	コンシューマーがリクエストのメッセージを待機する最大時間をミリ秒単位で設定します。レスポンスなしでタイムアウト期間に達すると、エラーが返されます。デフォルトは 30000 (30 秒) です。	整数
enable.auto.commit optional	true (デフォルト) に設定すると、メッセージオフセットはコンシューマーに対して自動的にコミットされます。 false に設定すると、メッセージオフセットは手動でコミットする必要があります。	boolean
fetch.min.bytes optional	コンシューマーが受信するデータの最小量をバイト単位で設定します。ブローカーは、送信するデータがこの量を超えるまで待機します。デフォルトは 1 バイトです。	整数
format オプション	コンシューマーに許可されるメッセージ形式。 binary (デフォルト) または json にすることができます。メッセージは JSON 形式に変換されます。	string
isolation.level optional	read_uncommitted (デフォルト) に設定すると、トランザクションの結果に関係なく、すべてのトランザクションレコードが取得されます。 read_committed に設定すると、コミットされたトランザクションからのレコードが取得されます。	string
name 任意	コンシューマーインスタンスの一意の名前。この名前は、コンシューマーグループの範囲内で一意です。名前は URL で使用されます。名前が指定されていない場合は、ランダムに生成された名前が割り当てられます。	string

4.2.4. ConsumerRecord

名前	スキーマ
headers オプション	KafkaHeaderList
key 任意	string
offset オプション	整数 (int64)

名前	スキーマ
partition オプション	整数 (int32)
topic オプション	string
value 任意	string

4.2.5. ConsumerRecordList

型: < [ConsumerRecord](#) > 配列

4.2.6. CreatedConsumer

名前	説明	スキーマ
base_uri optional	このコンシューマーインスタンスに対する後続リクエストの URI 構築に使用されるベース URI。	string
instance_id optional	グループ内のコンシューマーインスタンスの一意の ID。	string

4.2.7. エラー

名前	スキーマ
error_code optional	整数 (int32)
message 任意	string

4.2.8. KafkaHeader

名前	説明	スキーマ
key 必須		string
value 必須	Base64 でエンコードされたバイナリー形式のヘッダー値 パターン: <code>"^(?:[A-Za-z0-9+/]{4})*(?:[A-Za-z0-9+/]{2}== [A-Za-z0-9+/]{3}=)?\$"</code>	string (byte)

4.2.9. KafkaHeaderList

型: < [KafkaHeader](#) > 配列

4.2.10. OffsetCommitSeek

名前	スキーマ
offset 必須	整数 (int64)
partition 必須	整数 (int32)
topic 必須	string

4.2.11. OffsetCommitSeekList

名前	スキーマ
offsets オプション	< OffsetCommitSeek > 配列

4.2.12. OffsetRecordSent

名前	スキーマ
offset オプション	整数 (int64)
partition オプション	整数 (int32)

4.2.13. OffsetRecordSentList

名前	スキーマ
offsets オプション	< OffsetRecordSent > 配列

4.2.14. OffsetsSummary

名前	スキーマ
beginning_offset optional	整数 (int64)
end_offset optional	整数 (int64)

4.2.15. パーティション

名前	スキーマ
partition オプション	整数 (int32)
topic オプション	string

4.2.16. PartitionMetadata

名前	スキーマ
leader オプション	整数 (int32)
partition オプション	整数 (int32)
replicas 任意	< レプリカ > 配列

4.2.17. Partitions

名前	スキーマ
partitions オプション	< パーティション > 配列

4.2.18. ProducerRecord

名前	スキーマ
headers オプション	KafkaHeaderList

名前	スキーマ
partition オプション	整数 (int32)

4.2.19. ProducerRecordList

名前	スキーマ
records オプション	< ProducerRecord > 配列

4.2.20. ProducerRecordToPartition

型: オブジェクト

4.2.21. ProducerRecordToPartitionList

名前	スキーマ
records オプション	< ProducerRecordToPartition > 配列

4.2.22. Replica

名前	スキーマ
broker オプション	整数 (int32)
in_sync optional	boolean
leader オプション	boolean

4.2.23. SubscribedTopicList

名前	スキーマ
partitions オプション	< AssignedTopicPartitions > 配列
topics オプション	Topics

名前	スキーマ
----	------

4.2.24. TopicMetadata

名前	説明	スキーマ
configs オプション	トピックごとの設定のオーバーライド	< string, string > マップ
name 任意	トピックの名前	string
partitions オプション		< PartitionMetadata > 配列

4.2.25. Topics

名前	説明	スキーマ
topic_pattern optional	複数のトピックを照合するための正規表現トピックパターン	string
topics オプション		< string > 配列

4.3. パス

4.3.1. GET /

4.3.1.1. 説明

Kafka Bridge インスタンスに関する情報を JSON 形式で取得します。

4.3.1.2. レスポンス

HTTP コード	説明	スキーマ
200	Kafka Bridge インスタンスに関する情報。	BridgeInfo

4.3.1.3. 生成されるアイテム

- `application/json`

4.3.1.4. HTTP レスポンスの例

4.3.1.4.1. レスポンス 200

```
{
  "bridge_version" : "0.16.0"
}
```

4.3.2. POST /consumers/{groupid}

4.3.2.1. 説明

指定されたコンシューマーグループにコンシューマーインスタンスを作成します。任意で、コンシューマー名とサポートされている設定オプションを指定できます。これは、このコンシューマーインスタンスに対する後続のリクエストの URL 構築に使用する必要があるベース URI を返します。

4.3.2.2. パラメーター

型	名前	説明	スキーマ
パス	<code>groupid</code> 必須	コンシューマーを作成するコンシューマーグループの ID。	string
ボディ	<code>body</code> 必須	コンシューマーの名前と設定。この名前は、コンシューマーグループの範囲内で一意です。名前が指定されていない場合は、ランダムに生成された名前が割り当てられます。すべてのパラメーターはオプションです。サポートされている設定オプションを次の例に示します。	Consumer

4.3.2.3. レスポンス

HTTP コード	説明	スキーマ
200	コンシューマーは正常に作成されました。	CreatedConsumer
409	指定された名前のコンシューマーインスタンスは、Kafka Bridge にすでに存在します。	エラー
422	1つ以上のコンシューマー設定オプションに無効な値があります。	エラー

4.3.2.4. 消費されるアイテム

- `application/vnd.kafka.v2+json`

4.3.2.5. 生成されるアイテム

- `application/vnd.kafka.v2+json`

4.3.2.6. タグ

- Consumers

4.3.2.7. HTTP リクエストの例

4.3.2.7.1. リクエストの本文

```
{
  "name": "consumer1",
  "format": "binary",
  "auto.offset.reset": "earliest",
  "enable.auto.commit": false,
  "fetch.min.bytes": 512,
  "consumer.request.timeout.ms": 30000,
  "isolation.level": "read_committed"
}
```

4.3.2.8. HTTP レスポンスの例

4.3.2.8.1. レスポンス 200

```
{
  "instance_id": "consumer1",
  "base_uri": "http://localhost:8080/consumers/my-group/instances/consumer1"
}
```

4.3.2.8.2. レスポンス 409

```
{
  "error_code": 409,
  "message": "A consumer instance with the specified name already exists in the Kafka Bridge."
}
```

4.3.2.8.3. レスポンス 422

```
{
  "error_code": 422,
  "message": "One or more consumer configuration options have invalid values."
}
```

4.3.3. DELETE /consumers/{groupid}/instances/{name}

4.3.3.1. 説明

指定されたコンシューマーインスタンスを削除します。この操作のリクエストは、このコンシューマーの作成に使用された `/consumers/{groupid}` への **POST** リクエストからのレスポンスで返されたベース URL (ホストおよびポートを含む) を使用する必要があります。

4.3.3.2. パラメーター

型	名前	説明	スキーマ
パス	groupid 必須	コンシューマーが属するコンシューマーグループの ID。	string
パス	name 必須	削除するコンシューマーの名前。	string

4.3.3.3. レスポンス

HTTP コード	説明	スキーマ
204	コンシューマーは正常に削除されました。	コンテンツなし
404	指定されたコンシューマーインスタンスが見つかりませんでした。	エラー

4.3.3.4. 消費されるアイテム

- `application/vnd.kafka.v2+json`

4.3.3.5. 生成されるアイテム

- `application/vnd.kafka.v2+json`

4.3.3.6. タグ

- Consumers

4.3.3.7. HTTP レスポンスの例

4.3.3.7.1. レスポンス 404

```
{
  "error_code" : 404,
  "message" : "The specified consumer instance was not found."
}
```

4.3.4. POST `/consumers/{groupid}/instances/{name}/assignments`

4.3.4.1. 説明

1つ以上のトピックパーティションをコンシューマーに割り当てます。

4.3.4.2. パラメーター

型	名前	説明	スキーマ
パス	groupid 必須	コンシューマーが属するコンシューマーグループのID。	string
パス	name 必須	トピックパーティションを割り当てるコンシューマーの名前。	string
ボディ	body 必須	コンシューマーに割り当てるトピックパーティションのリスト。	Partitions

4.3.4.3. レスポンス

HTTP コード	説明	スキーマ
204	パーティションは正常に割り当てられました。	コンテンツなし
404	指定されたコンシューマーインスタスが見つかりませんでした。	エラー
409	トピック、パーティション、およびパターンへのサブスクリプションは相互に排他的です。	エラー

4.3.4.4. 消費されるアイテム

- `application/vnd.kafka.v2+json`

4.3.4.5. 生成されるアイテム

- `application/vnd.kafka.v2+json`

4.3.4.6. タグ

- Consumers

4.3.4.7. HTTP リクエストの例

4.3.4.7.1. リクエストの本文

```
{
  "partitions": [ {
    "topic": "topic",
    "partition": 0
  }, {
    "topic": "topic",
```

```

    "partition" : 1
  }
}

```

4.3.4.8. HTTP レスポンスの例

4.3.4.8.1. レスポンス 404

```

{
  "error_code" : 404,
  "message" : "The specified consumer instance was not found."
}

```

4.3.4.8.2. レスポンス 409

```

{
  "error_code" : 409,
  "message" : "Subscriptions to topics, partitions, and patterns are mutually exclusive."
}

```

4.3.5. POST /consumers/{groupid}/instances/{name}/offsets

4.3.5.1. 説明

コンシューマーオフセットのリストをコミットします。コンシューマーによってフェッチされたすべてのレコードのオフセットをコミットするには、リクエストの本文を空のままにします。

4.3.5.2. パラメーター

型	名前	説明	スキーマ
パス	groupid 必須	コンシューマーが属するコンシューマーグループの ID。	string
パス	name 必須	コンシューマーの名前。	string
ボディ	body 任意	コンシューマーオフセットコミットログにコミットするコンシューマーオフセットのリスト。オフセットをコミットする1つ以上のトピックパーティションを指定できます。	OffsetCommitSeekList

4.3.5.3. レスポンス

HTTP コード	説明	スキーマ
204	コミットは正常に行われました。	コンテンツなし

HTTP コード	説明	スキーマ
404	指定されたコンシューマーインスタンスが見つかりませんでした。	エラー

4.3.5.4. 消費されるアイテム

- `application/vnd.kafka.v2+json`

4.3.5.5. 生成されるアイテム

- `application/vnd.kafka.v2+json`

4.3.5.6. タグ

- Consumers

4.3.5.7. HTTP リクエストの例

4.3.5.7.1. リクエストの本文

```
{
  "offsets": [{
    "topic": "topic",
    "partition": 0,
    "offset": 15
  }, {
    "topic": "topic",
    "partition": 1,
    "offset": 42
  }]
}
```

4.3.5.8. HTTP レスポンスの例

4.3.5.8.1. レスポンス 404

```
{
  "error_code": 404,
  "message": "The specified consumer instance was not found."
}
```

4.3.6. POST /consumers/{groupid}/instances/{name}/positions

4.3.6.1. 説明

サブスクライブされたコンシューマーが、次に特定のトピックパーティションからレコードのセットをフェッチするときに、特定のオフセットから複数のオフセットをフェッチするように設定します。これは、コンシューマーのデフォルトのフェッチ動作をオーバーライドします。1つ以上のトピックパー

パーティションを指定できます。

4.3.6.2. パラメーター

型	名前	説明	スキーマ
パス	groupid 必須	コンシューマーが属するコンシューマーグループのID。	string
パス	name 必須	サブスクライブされたコンシューマーの名前。	string
ボディ	body 必須	サブスクライブされたコンシューマーが次にレコードをフェッチするパーティションオフセットのリスト。	OffsetCommitSeekList

4.3.6.3. レスポンス

HTTP コード	説明	スキーマ
204	シークは正常に実行されました。	コンテンツなし
404	指定されたコンシューマーインスタンスが見つからなかったか、指定されたコンシューマーインスタンスに指定されたパーティションの1つが割り当てられていませんでした。	エラー

4.3.6.4. 消費されるアイテム

- **application/vnd.kafka.v2+json**

4.3.6.5. 生成されるアイテム

- **application/vnd.kafka.v2+json**

4.3.6.6. タグ

- Consumers
- Seek

4.3.6.7. HTTP リクエストの例

4.3.6.7.1. リクエストの本文

```
{
  "offsets": [{
    "topic": "topic",
    "partition": 0,
```

```

    "offset" : 15
  }, {
    "topic" : "topic",
    "partition" : 1,
    "offset" : 42
  }
}

```

4.3.6.8. HTTP レスポンスの例

4.3.6.8.1. レスポンス 404

```

{
  "error_code" : 404,
  "message" : "The specified consumer instance was not found."
}

```

4.3.7. POST /consumers/{groupid}/instances/{name}/positions/beginning

4.3.7.1. 説明

1つ以上の指定されたトピックパーティションの最初のオフセットをシークする (そしてその後読み取る) ようにサブスクライブされたコンシューマーを設定します。

4.3.7.2. パラメーター

型	名前	説明	スキーマ
パス	groupid 必須	サブスクライブされたコンシューマーが属するコンシューマーグループの ID。	string
パス	name 必須	サブスクライブされたコンシューマーの名前。	string
ボディ	body 必須	コンシューマーがサブスクライブしているトピックパーティションのリスト。コンシューマーは、指定されたパーティションの最初のオフセットを探します。	Partitions

4.3.7.3. レスポンス

HTTP コード	説明	スキーマ
204	正常に実行されたものを最初にシークします。	コンテンツなし

HTTP コード	説明	スキーマ
404	指定されたコンシューマーインスタンスが見つからなかったか、指定されたコンシューマーインスタンスに指定されたパーティションの1つが割り当てられていませんでした。	エラー

4.3.7.4. 消費されるアイテム

- `application/vnd.kafka.v2+json`

4.3.7.5. 生成されるアイテム

- `application/vnd.kafka.v2+json`

4.3.7.6. タグ

- Consumers
- Seek

4.3.7.7. HTTP リクエストの例

4.3.7.7.1. リクエストの本文

```
{
  "partitions": [ {
    "topic": "topic",
    "partition": 0
  }, {
    "topic": "topic",
    "partition": 1
  } ]
}
```

4.3.7.8. HTTP レスポンスの例

4.3.7.8.1. レスポンス 404

```
{
  "error_code": 404,
  "message": "The specified consumer instance was not found."
}
```

4.3.8. POST /consumers/{groupid}/instances/{name}/positions/end

4.3.8.1. 説明

1つ以上の指定されたトピックパーティションの終わりでオフセットをシークする (そしてその後読み取る) ようにサブスクライブされたコンシューマーを設定します。

4.3.8.2. パラメーター

型	名前	説明	スキーマ
パス	groupid 必須	サブスクライブされたコンシューマーが属するコンシューマーグループの ID。	string
パス	name 必須	サブスクライブされたコンシューマーの名前。	string
ボディ	body 任意	コンシューマーがサブスクライブしているトピックパーティションのリスト。コンシューマーは、指定されたパーティションの最後のオフセットをシークします。	Partitions

4.3.8.3. レスポンス

HTTP コード	説明	スキーマ
204	最後に正常に実行されたものをシークします。	コンテンツなし
404	指定されたコンシューマーインスタスが見つからなかったか、指定されたコンシューマーインスタスに指定されたパーティションの1つが割り当てられていませんでした。	エラー

4.3.8.4. 消費されるアイテム

- `application/vnd.kafka.v2+json`

4.3.8.5. 生成されるアイテム

- `application/vnd.kafka.v2+json`

4.3.8.6. タグ

- Consumers
- Seek

4.3.8.7. HTTP リクエストの例

4.3.8.7.1. リクエストの本文

```
{
  "partitions": [{
    "topic": "topic",
    "partition": 0
  }, {
```

```

    "topic" : "topic",
    "partition" : 1
  ]]
}

```

4.3.8.8. HTTP レスポンスの例

4.3.8.8.1. レスポンス 404

```

{
  "error_code" : 404,
  "message" : "The specified consumer instance was not found."
}

```

4.3.9. GET /consumers/{groupid}/instances/{name}/records

4.3.9.1. 説明

メッセージ値、トピック、パーティションなど、サブスクライブされたコンシューマーのレコードを取得します。この操作のリクエストは、このコンシューマーの作成に使用された `/consumers/{groupid}` への **POST** リクエストからのレスポンスで返されたベース URL (ホストおよびポートを含む) を使用する必要があります。

4.3.9.2. パラメーター

型	名前	説明	スキーマ
パス	groupid 必須	サブスクライブされたコンシューマーが属するコンシューマーグループの ID。	string
パス	name 必須	レコードを取得するサブスクライブされたコンシューマーの名前。	string
クエリー	max_bytes optional	レスポンスに含めることができるエンコードされていないキーと値の最大サイズ (バイト単位)。それ以外の場合は、コード 422 のエラーレスポンスが返されます。	整数
クエリー	timeout オプション	HTTP Bridge がリクエストをタイムアウトする前にレコードの取得に費やす最大時間 (ミリ秒単位)。	整数

4.3.9.3. レスポンス

HTTP コード	説明	スキーマ
200	ポーリングリクエストは正常に実行されました。	ConsumerRecordList

HTTP コード	説明	スキーマ
404	指定されたコンシューマーインスタスが見つかりませんでした。	エラー
406	コンシューマー作成リクエストで使用された format が、このリクエストの Accept ヘッダーに埋め込まれたフォーマットと一致しないか、ブリッジが JSON 形式でエンコードされていないトピックからメッセージを受け取りました。	エラー
422	レスポンスにおいて、コンシューマーが受信できる最大バイト数を超えています	エラー

4.3.9.4. 生成されるアイテム

- `application/vnd.kafka.json.v2+json`
- `application/vnd.kafka.binary.v2+json`
- `application/vnd.kafka.v2+json`

4.3.9.5. タグ

- Consumers

4.3.9.6. HTTP レスポンスの例

4.3.9.6.1. レスポンス 200

```
[ {
  "topic" : "topic",
  "key" : "key1",
  "value" : {
    "foo" : "bar"
  },
  "partition" : 0,
  "offset" : 2
}, {
  "topic" : "topic",
  "key" : "key2",
  "value" : [ "foo2", "bar2" ],
  "partition" : 1,
  "offset" : 3
} ]
```

```
[
  {
    "topic": "test",
    "key": "a2V5",
    "value": "Y29uZmx1ZW50",
    "partition": 1,
    "offset": 100,
```

```

    },
    {
      "topic": "test",
      "key": "a2V5",
      "value": "a2Fma2E=",
      "partition": 2,
      "offset": 101,
    }
  ]

```

4.3.9.6.2. レスポンス 404

```

{
  "error_code" : 404,
  "message" : "The specified consumer instance was not found."
}

```

4.3.9.6.3. レスポンス 406

```

{
  "error_code" : 406,
  "message" : "The `format` used in the consumer creation request does not match the embedded format in the Accept header of this request."
}

```

4.3.9.6.4. レスポンス 422

```

{
  "error_code" : 422,
  "message" : "Response exceeds the maximum number of bytes the consumer can receive"
}

```

4.3.10. POST /consumers/{groupid}/instances/{name}/subscription

4.3.10.1. 説明

コンシューマーを1つ以上のトピックにサブスクライブします。コンシューマーがサブスクライブするトピックを(トピックタイプの)リスト、または **topic_pattern** フィールドとして記述できます。各呼び出しは、サブスクライバーのサブスクリプションを置き換えます。

4.3.10.2. パラメーター

型	名前	説明	スキーマ
パス	groupid 必須	サブスクライブされたコンシューマーが属するコンシューマーグループの ID。	string
パス	name 必須	トピックをサブスクライブするコンシューマーの名前。	string

型	名前	説明	スキーマ
ボディ	body 必須	コンシューマーがサブスクライブするトピックのリスト。	Topics

4.3.10.3. レスポンス

HTTP コード	説明	スキーマ
204	コンシューマーは正常にサブスクライブしました。	コンテンツなし
404	指定されたコンシューマーインスタスが見つかりませんでした。	エラー
409	トピック、パーティション、およびパターンへのサブスクリプションは相互に排他的です。	エラー
422	(Topics タイプの) リストまたは topic_pattern を指定する必要があります。	エラー

4.3.10.4. 消費されるアイテム

- **application/vnd.kafka.v2+json**

4.3.10.5. 生成されるアイテム

- **application/vnd.kafka.v2+json**

4.3.10.6. タグ

- Consumers

4.3.10.7. HTTP リクエストの例

4.3.10.7.1. リクエストの本文

```
{
  "topics": [ "topic1", "topic2" ]
}
```

4.3.10.8. HTTP レスポンスの例

4.3.10.8.1. レスポンス 404

```
{
  "error_code": 404,
}
```

```
"message" : "The specified consumer instance was not found."
}
```

4.3.10.8.2. レスポンス 409

```
{
  "error_code" : 409,
  "message" : "Subscriptions to topics, partitions, and patterns are mutually exclusive."
}
```

4.3.10.8.3. レスポンス 422

```
{
  "error_code" : 422,
  "message" : "A list (of Topics type) or a topic_pattern must be specified."
}
```

4.3.11. GET /consumers/{groupid}/instances/{name}/subscription

4.3.11.1. 説明

コンシューマーがサブスクライブしているトピックのリストを取得します。

4.3.11.2. パラメーター

型	名前	説明	スキーマ
パス	groupid 必須	サブスクライブされたコンシューマーが属するコンシューマーグループの ID。	string
パス	name 必須	サブスクライブされたコンシューマーの名前。	string

4.3.11.3. レスポンス

HTTP コード	説明	スキーマ
200	サブスクライブされたトピックとパーティションのリスト。	SubscribedTopicList
404	指定されたコンシューマーインスタンスが見つかりませんでした。	エラー

4.3.11.4. 生成されるアイテム

- `application/vnd.kafka.v2+json`

4.3.11.5. タグ

- Consumers

4.3.11.6. HTTP レスポンスの例

4.3.11.6.1. レスポンス 200

```
{
  "topics" : [ "my-topic1", "my-topic2" ],
  "partitions" : [ {
    "my-topic1" : [ 1, 2, 3 ]
  }, {
    "my-topic2" : [ 1 ]
  } ]
}
```

4.3.11.6.2. レスポンス 404

```
{
  "error_code" : 404,
  "message" : "The specified consumer instance was not found."
}
```

4.3.12. DELETE /consumers/{groupid}/instances/{name}/subscription

4.3.12.1. 説明

すべてのトピックからコンシューマーの登録を解除します。

4.3.12.2. パラメーター

型	名前	説明	スキーマ
パス	groupid 必須	サブスクライブされたコンシューマーが属するコンシューマーグループの ID。	string
パス	name 必須	トピックからサブスクライブを解除するコンシューマーの名前。	string

4.3.12.3. レスポンス

HTTP コード	説明	スキーマ
204	コンシューマーは正常にサブスクライブを解除しました。	コンテンツなし
404	指定されたコンシューマーインスタスが見つかりませんでした。	エラー

4.3.12.4. タグ

- Consumers

4.3.12.5. HTTP レスポンスの例

4.3.12.5.1. レスポンス 404

```
{
  "error_code" : 404,
  "message" : "The specified consumer instance was not found."
}
```

4.3.13. GET /healthy

4.3.13.1. 説明

ブリッジが実行しているかどうかを確認します。これは、必ずしもリクエストを受け入れる準備ができていることを意味するわけではありません。

4.3.13.2. レスポンス

HTTP コード	説明	スキーマ
200	ブリッジは正常です。	コンテンツなし

4.3.14. GET /openapi

4.3.14.1. 説明

OpenAPI v2 仕様を JSON 形式で取得します。

4.3.14.2. レスポンス

HTTP コード	説明	スキーマ
200	JSON 形式の OpenAPI v2 仕様が正常に取得されました。	string

4.3.14.3. 生成されるアイテム

- `application/json`

4.3.15. GET /ready

4.3.15.1. 説明

ブリッジの準備ができており、リクエストを受け入れることができるかどうかを確認してください。

4.3.15.2. レスポンス

HTTP コード	説明	スキーマ
200	ブリッジの準備が完了しました。	コンテンツなし

4.3.16. GET /topics

4.3.16.1. 説明

すべてのトピックのリストを取得します。

4.3.16.2. レスポンス

HTTP コード	説明	スキーマ
200	トピックのリスト。	< string > 配列

4.3.16.3. 生成されるアイテム

- `application/vnd.kafka.v2+json`

4.3.16.4. タグ

- Topics

4.3.16.5. HTTP レスポンスの例

4.3.16.5.1. レスポンス 200

```
[ "topic1", "topic2" ]
```

4.3.17. POST /topics/{topicname}

4.3.17.1. 説明

1つ以上のレコードを特定のトピックに送信し、任意でパーティション、キー、またはその両方を指定します。

4.3.17.2. パラメーター

型	名前	説明	スキーマ
パス	topicname 必須	レコードの送信先またはメタデータの取得元のトピックの名前。	string
クエリー	async オプション	メタデータを待機する代わりに、レコードの送信直後に返すかどうか。指定されている場合はオフセットは返されません。デフォルトは false です。	boolean
ボディ	body 必須		ProducerRecordList

4.3.17.3. レスポンス

HTTP コード	説明	スキーマ
200	レコードは正常に送信されました。	OffsetRecordSent List
404	指定されたトピックが見つかりませんでした。	エラー
422	レコードリストが無効です。	エラー

4.3.17.4. 消費されるアイテム

- application/vnd.kafka.json.v2+json
- application/vnd.kafka.binary.v2+json

4.3.17.5. 生成されるアイテム

- application/vnd.kafka.v2+json

4.3.17.6. タグ

- producer
- Topics

4.3.17.7. HTTP リクエストの例

4.3.17.7.1. リクエストの本文

```
{
  "records": [{
    "key": "key1",
    "value": "value1"
  }]
```

```

    }, {
      "value" : "value2",
      "partition" : 1
    }, {
      "value" : "value3"
    }
  ]
}

```

4.3.17.8. HTTP レスポンスの例

4.3.17.8.1. レスポンス 200

```

{
  "offsets" : [ {
    "partition" : 2,
    "offset" : 0
  }, {
    "partition" : 1,
    "offset" : 1
  }, {
    "partition" : 2,
    "offset" : 2
  } ]
}

```

4.3.17.8.2. レスポンス 404

```

{
  "error_code" : 404,
  "message" : "The specified topic was not found."
}

```

4.3.17.8.3. レスポンス 422

```

{
  "error_code" : 422,
  "message" : "The record list contains invalid records."
}

```

4.3.18. GET /topics/{topicname}

4.3.18.1. 説明

特定のトピックに関するメタデータを取得します。

4.3.18.2. パラメーター

型	名前	説明	スキーマ
---	----	----	------

型	名前	説明	スキーマ
パス	topicname 必須	レコードの送信先またはメタデータの取得元のトピックの名前。	string

4.3.18.3. レスポンス

HTTP コード	説明	スキーマ
200	トピックのメタデータ	TopicMetadata

4.3.18.4. 生成されるアイテム

- `application/vnd.kafka.v2+json`

4.3.18.5. タグ

- Topics

4.3.18.6. HTTP レスポンスの例

4.3.18.6.1. レスポンス 200

```
{
  "name" : "topic",
  "offset" : 2,
  "configs" : {
    "cleanup.policy" : "compact"
  },
  "partitions" : [ {
    "partition" : 1,
    "leader" : 1,
    "replicas" : [ {
      "broker" : 1,
      "leader" : true,
      "in_sync" : true
    }, {
      "broker" : 2,
      "leader" : false,
      "in_sync" : true
    } ]
  }, {
    "partition" : 2,
    "leader" : 2,
    "replicas" : [ {
      "broker" : 1,
      "leader" : false,
      "in_sync" : true
    }, {
      "broker" : 2,
```

```

    "leader" : true,
    "in_sync" : true
  }}
}}
}

```

4.3.19. GET /topics/{topicname}/partitions

4.3.19.1. 説明

トピックのパーティションのリストを取得します。

4.3.19.2. パラメーター

型	名前	説明	スキーマ
パス	topicname 必須	レコードの送信先またはメタデータの取得元のトピックの名前。	string

4.3.19.3. レスポンス

HTTP コード	説明	スキーマ
200	パーティションのリスト	< PartitionMetadata > 配列
404	指定されたトピックが見つかりませんでした。	エラー

4.3.19.4. 生成されるアイテム

- `application/vnd.kafka.v2+json`

4.3.19.5. タグ

- Topics

4.3.19.6. HTTP レスポンスの例

4.3.19.6.1. レスポンス 200

```

[ {
  "partition" : 1,
  "leader" : 1,
  "replicas" : [ {
    "broker" : 1,
    "leader" : true,
    "in_sync" : true
  }
]
}
]

```

```

    }, {
      "broker" : 2,
      "leader" : false,
      "in_sync" : true
    } ]
  }, {
    "partition" : 2,
    "leader" : 2,
    "replicas" : [ {
      "broker" : 1,
      "leader" : false,
      "in_sync" : true
    }, {
      "broker" : 2,
      "leader" : true,
      "in_sync" : true
    } ]
  } ]
} ]

```

4.3.19.6.2. レスポンス 404

```

{
  "error_code" : 404,
  "message" : "The specified topic was not found."
}

```

4.3.20. POST /topics/{topicname}/partitions/{partitionid}

4.3.20.1. 説明

任意でキーを指定して、1つ以上のレコードを特定のトピックパーティションに送信します。

4.3.20.2. パラメーター

型	名前	説明	スキーマ
パス	partitionid 必須	レコードを送信したり、メタデータを取得したりするパーティションの ID。	整数
パス	topicname 必須	レコードの送信先またはメタデータの取得元のトピックの名前。	string
ボディ	body 必須	値 (必須) とキー (任意) を含む、特定のトピックパーティションに送信するレコードのリスト。	ProducerRecordToPartitionList

4.3.20.3. レスポンス

HTTP コード	説明	スキーマ
200	レコードは正常に送信されました。	OffsetRecordSent List
404	指定されたトピックパーティションが見つかりませんでした。	エラー
422	レコードが無効です。	エラー

4.3.20.4. 消費されるアイテム

- `application/vnd.kafka.json.v2+json`
- `application/vnd.kafka.binary.v2+json`

4.3.20.5. 生成されるアイテム

- `application/vnd.kafka.v2+json`

4.3.20.6. タグ

- `producer`
- `Topics`

4.3.20.7. HTTP リクエストの例

4.3.20.7.1. リクエストの本文

```
{
  "records": [{
    "key": "key1",
    "value": "value1"
  }, {
    "value": "value2"
  }]
}
```

4.3.20.8. HTTP レスポンスの例

4.3.20.8.1. レスポンス 200

```
{
  "offsets": [{
    "partition": 2,
    "offset": 0
  }, {
    "partition": 1,
```



```

    "offset" : 1
  }, {
    "partition" : 2,
    "offset" : 2
  }
]
}

```

4.3.20.8.2. レスポンス 404

```

{
  "error_code" : 404,
  "message" : "The specified topic partition was not found."
}

```

4.3.20.8.3. レスポンス 422

```

{
  "error_code" : 422,
  "message" : "The record is not valid."
}

```

4.3.21. GET /topics/{topicname}/partitions/{partitionid}

4.3.21.1. 説明

トピックパーティションのパーティションメタデータを取得します。

4.3.21.2. パラメーター

型	名前	説明	スキーマ
パス	partitionid 必須	レコードを送信したり、メタデータを取得したりするパーティションの ID。	整数
パス	topicname 必須	レコードの送信先またはメタデータの取得元のトピックの名前。	string

4.3.21.3. レスポンス

HTTP コード	説明	スキーマ
200	パーティションメタデータ	PartitionMetadata
404	指定されたトピックパーティションが見つかりませんでした。	エラー

4.3.21.4. 生成されるアイテム

- `application/vnd.kafka.v2+json`

4.3.21.5. タグ

- Topics

4.3.21.6. HTTP レスポンスの例

4.3.21.6.1. レスポンス 200

```
{
  "partition" : 1,
  "leader" : 1,
  "replicas" : [ {
    "broker" : 1,
    "leader" : true,
    "in_sync" : true
  }, {
    "broker" : 2,
    "leader" : false,
    "in_sync" : true
  } ]
}
```

4.3.21.6.2. レスポンス 404

```
{
  "error_code" : 404,
  "message" : "The specified topic partition was not found."
}
```

4.3.22. GET /topics/{topicname}/partitions/{partitionid}/offsets

4.3.22.1. 説明

トピックパーティションのオフセットの概要を取得します。

4.3.22.2. パラメーター

型	名前	説明	スキーマ
パス	<code>partitionid</code> 必須	パーティションの ID。	整数
パス	<code>topicname</code> 必須	パーティションを含むトピックの名前。	string

4.3.22.3. レスポンス

HTTP コード	説明	スキーマ
200	トピックパーティションのオフセットの要約。	OffsetsSummary
404	指定されたトピックパーティションが見つかりませんでした。	エラー

4.3.22.4. 生成されるアイテム

- `application/vnd.kafka.v2+json`

4.3.22.5. タグ

- Topics

4.3.22.6. HTTP レスポンスの例

4.3.22.6.1. レスポンス 200

```
{  
  "beginning_offset" : 10,  
  "end_offset" : 50  
}
```

4.3.22.6.2. レスポンス 404

```
{  
  "error_code" : 404,  
  "message" : "The specified topic partition was not found."  
}
```

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