Red Hat AMQ Streams 2.1

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

Last Updated: 2022-07-12

Red Hat AMQ Streams 2.1 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

Enter your first name here. Enter your surname here.
Enter your organisation's name here. Enter your organisational division here.
Enter your email address here.

法律上の通知

概要

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 インターフェースが提供されます。

目次

多様性を受け入れるオープンソースの強化

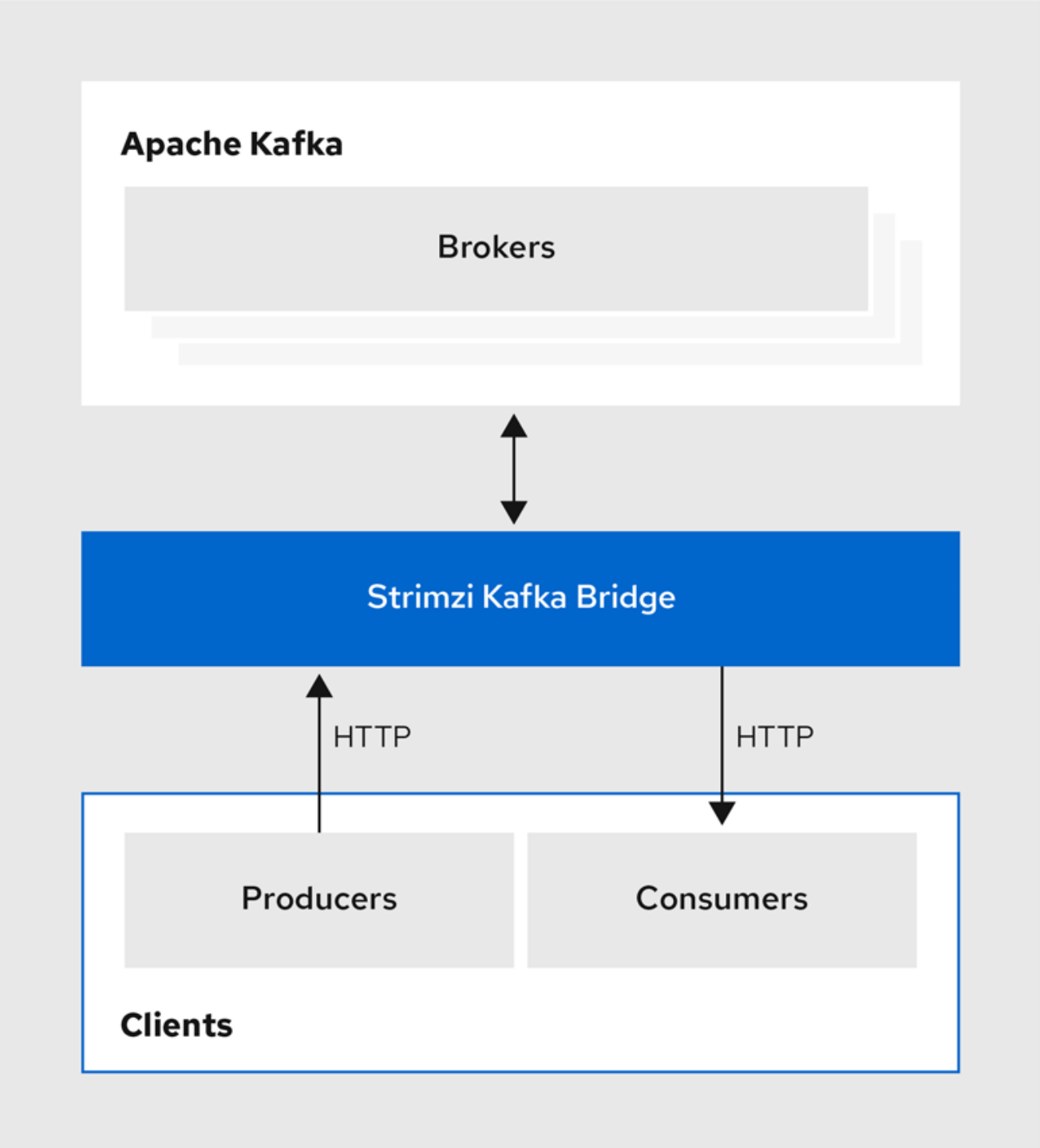
Red Hat では、コード、ドキュメント、Web プロパティーにおける配慮に欠ける用語の置き換えに取り組んでいます。まずは、マスター (master)、スレーブ (slave)、ブラックリスト (blacklist)、ホワイトリスト (whitelist) の 4 つの用語の置き換えから始めます。これは大規模な取り組みであるため、これらの変更は今後の複数のリリースで段階的に実施されます。詳細は、Red Hat CTO である Chris Wright のメッセージ をご覧ください。

第1章 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。

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。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。

関連情報

- AMQ Streams のドキュメント

1.2. 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。

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 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

- AMQ Streams Kafka Bridge API リファレンス

1.3. 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

プロパティーファイル 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 を使用することで、Kafka Bridge を使用して消費および生成できるトピックを制限することができます。

関連情報

- AMQ Streams のドキュメント

1.4. KAFKA BRIDGE HTTP インターフェイスの保護

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

Kafka Bridge を次のツールと組み合わせて、保護することができます。

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

1.5. 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。

1.5.1. コンテンツタイプヘッダー

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、**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どおり、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リクエストのボディが空の場合は、**Content-Type**を**設定しないでください**。空のボディーを使用して、デフォルト値のコンシューマーを作成できます。

1.5.2. 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。サポートされる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**)。

1.5.3. 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ボディー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": "QXBhY2hlIEthZmthIGlzIHRoZSBib21iIQ==" 1
              }
            ]
        },
    ]
}'
```

1 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。

1.5.4. 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、サブスクライ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
```

1.6. CORS

CORS (Cross-Origin Resource Sharing) を使用すると、Kafka Bridge HTTP の設定で Kafka クラスターにアクセスするために許可される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=https://strimzi.io
http.cors.allowedMethods=GET,POST,PUT,DELETE,OPTIONS,PATCH
```

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**の各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。

すべての要求には、HTTP 要求のソースであるヘッダーの **origins** 値が必要です。

1.6.1. 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、オリジンを **https://strimzi.io** と指定します。

```
Origin: https://strimzi.io
```

ヘッダー情報がリクエストに追加されます。

```
curl -v -X GET HTTP-ADDRESS/bridge-consumer/records \
-H 'Origin: https://strimzi.io'\
-H 'content-type: application/vnd.kafka.v2+json'
```

Kafka Bridgeからの応答では、 **Access-Control-Allow-Origin**ヘッダーが返されます。

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

(1) アスタリスク(*)を返すと、リソースをどのドメインでもアクセスできることが分かります。

1.6.2. 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。

ここでは、プリフライトリクエストは **https://strimzi.io** から **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-user/subscription HTTP/1.1
Origin: https://strimzi.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アラートされます。

(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: https://strimzi.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応答し、リクエストが受け入れられることを確認します。応答ヘッダーは、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: https://strimzi.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-ADDRESS/topics/bridge-topic \
-H 'Origin: https://strimzi.io' \
-H 'content-type: application/vnd.kafka.v2+json'
```

応答は、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: https://strimzi.io
```

関連情報

- Fetch CORS 仕様

1.7. KAFKA BRIDGE のロガーの設定

AMQ Streams Kafka ブリッジを使用すると、関連する OpenAPI 仕様で定義される操作ごとに異なるログレベルを設定できます。

各操作には、ブリッジが HTTP クライアントから要求を受信する対応するための API エンドポイントがあります。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 要求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および貼り付けできる 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。

データ形式について

このクイックスタートでは、バイナリーではなく 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ます。

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 プロパティーの設定

この手順では、AMQ Streams Kafka Bridge によって使用される Kafka および HTTP 接続プロパティーを設定する方法を説明します。

Kafka 関連のプロパティー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ーなど、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 プロパティーは、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ー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. AMQ Streams Kafka Bridge のインストールアーカイブにある **application.properties** ファイルを編集します。
   プロパティーファイルを使用して、Kafka および HTTP 関連のプロパティーを指定し、分散トレースを有効にします。

   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ーなど、標準の Kafka 関連のプロパティーを設定します。
   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。
   - **kafka.producer.acks**: HTTP クライアントに確認を提供。
   - **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。
     Kafka プロパティー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 関連のプロパティーを設定します。
   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。

2.3. KAFKA BRIDGE のインストール

この手順に従って、AMQ Streams Kafka Bridge をインストールします。

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。
- Kafka Bridge 設定プロパティーが設定されている。

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解凍します。
2. 設定プロパティー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>/configfile.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
```

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

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、全メッセージの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 クラスターには、3 つのパーティションを持つトピックがあります。
  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を確認する

```
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** アレイを **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** アレイ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。

   応答例

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

```
    }
   ]
}
```

追加のトピック要求

他の curl 要求を実行して、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
```

応答例

```
[
  "__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
```

応答例

```
{
  "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
```

応答例

```
[
  {
    "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
```

応答例

```
{
  "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
```

応答例

```
{
  "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}

2.5. KAFKA BRIDGE コンシューマーの作成

Kafka クラスターで何らかの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、オフセットを跡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。

  応答例

```
#...
{
  "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

上記で作成した Kafka Bridge 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}

2.6. 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 アレイには、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** アレイの代わりに **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

2.7. 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生成」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アレイ (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生成」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をコミットします。

関連情報

- GET /consumers/{groupid}/instances/{name}/records

2.8. 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ので、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。この代わりに、リクエスト本文に、オフセットをコミットするトピックおよびパーティションを指定するアレイ (OffsetCommitSeekList) を含めることができます。

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、オフセットをシークのエンドポイントを試行します。

関連情報

- POST /consumers/{groupid}/instances/{name}/offsets

2.9. 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。この時点で、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

また、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、すでに作成した Kafka Bridge コンシューマーを削除 できます。

関連情報

- POST /consumers/{groupid}/instances/{name}/positions
- POST /consumers/{groupid}/instances/{name}/positions/beginning
- POST /consumers/{groupid}/instances/{name}/positions/end

2.10. KAFKA BRIDGE コンシューマーの削除

このクイックスタートを通して使用した 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、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 AMQ STREAMS KAFKA BRIDGE API リファレンス

3.1. 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 クラスターと統合するための RESTAPI 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。

3.1.1. バージョン情報

バージョン : 0.1.0

3.1.2. タグ

- コンシューマー: Kafka クラスター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。

3.1.3. 消費

- **application/json**

3.1.4. 生成されるもの

- **application/json**

3.2. 定義

3.2.1. AssignedTopicPartitions

Type : < string, < integer (int32) > array > map

3.2.2. BridgeInfo

Kafka Bridge インスタンスに関する情報。

名前	スキーマ
bridge_version **optional**	文字列

3.2.3. コンシューマー

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。**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。**latest** に設定すると、メッセージは最新のオフセットから読み込まれます。	文字列
consumer.request.timeout.ms optional	コンシューマーが要求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。応答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。	整数
enable.auto.commit optional	**true** 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。	ブール値
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。	整数
format optional	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。	文字列
isolation.level optional	**read_uncommitted** に設定すると、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。	文字列
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。	文字列

3.2.4. ConsumerRecord

名前	スキーマ
headers optional	KafkaHeaderList
key **任意**	文字列
offset optional	integer (int64)
partition optional	整数 (int32)

名前	スキーマ
topic optional	文字列
value **任意**	文字列

3.2.5. ConsumerRecordList

Type : < ConsumerRecord > array

3.2.6. 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のリクエストの URI を構築するのに使用されるベース URI。	文字列
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。	文字列

3.2.7. Error

名前	スキーマ
error_code optional	整数 (int32)
message **任意**	文字列

3.2.8. KafkaHeader

名前	説明	スキーマ
key **必須**		文字列
value required	バイナリー形式のヘッダー値: base64 でエンコードした Pattern : **"^(?:[A-Za-z0-9+/]{4})*(?:[A-Za-z0-9+/]{2}==|[A-Za-z0-9+/]{3}=)?$"**	文字列 (バイト)

3.2.9. KafkaHeaderList

Type : < KafkaHeader > array

3.2.10. OffsetCommitSeek

名前	スキーマ
offset **required**	integer (int64)
partition **required**	整数 (int32)
topic **required**	文字列

3.2.11. OffsetCommitSeekList

名前	スキーマ
offsets **optional**	< OffsetCommitSeek > array

3.2.12. OffsetRecordSent

名前	スキーマ
offset **optional**	integer (int64)
partition **optional**	整数 (int32)

3.2.13. OffsetRecordSentList

名前	スキーマ
offsets **optional**	< OffsetRecordSent > array

3.2.14. OffsetsSummary

名前	スキーマ
beginning_offset **optional**	integer (int64)

名前	スキーマ
end_offset optional	integer (int64)

3.2.15. パーティション

名前	スキーマ
partition optional	整数 (int32)
topic optional	文字列

3.2.16. PartitionMetadata

名前	スキーマ
leader optional	整数 (int32)
partition optional	整数 (int32)
replicas **任意**	< Replica > アレイ

3.2.17. Partitions

名前	スキーマ
partitions optional	< Partition > アレイ

3.2.18. ProducerRecord

名前	スキーマ
headers optional	KafkaHeaderList

名前	スキーマ
partition optional	整数 (int32)

3.2.19. ProducerRecordList

名前	スキーマ
records optional	< ProducerRecord > array

3.2.20. ProducerRecordToPartition

Type : おぷジェクト

3.2.21. ProducerRecordToPartitionList

名前	スキーマ
records optional	< ProducerRecordToPartition > array

3.2.22. レプリカ

名前	スキーマ
broker optional	整数 (int32)
in_sync optional	ブール値
leader optional	ブール値

3.2.23. SubscribedTopicList

名前	スキーマ
partitions optional	< AssignedTopicPartitions > array
topics optional	トピック

名前	スキーマ

3.2.24. TopicMetadata

名前	説明	スキーマ
configs optional	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	文字列
partitions optional		< PartitionMetadata > array

3.2.25. トピック

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	文字列
topics optional		< 文字列 > 配列

3.3. パス

3.3.1. GET /

3.3.1.1. 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。

3.3.1.2. 応答

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

3.3.1.3. 生成されるもの

- **application/json**

3.3.1.4. HTTP 応答の例

3.3.1.4.1. Response 200

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

3.3.2. POST /consumers/{groupid}

3.3.2.1. 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 を構築するために使用する必要があるベース URI を返します。

3.3.2.2. パラメーター

タイプ	Name	説明	スキーマ
パス	**groupid** **required**	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。	文字列
ボディー	**body** **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。	コンシューマー

3.3.2.3. 応答

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。	Error
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。	Error

3.3.2.4. 消費

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

3.3.2.5. 生成されるもの

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

3.3.2.6. タグ

- コンシューマー

3.3.2.7. HTTP 要求の例

3.3.2.7.1. 要求のボディー

```
{
  "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"
}
```

3.3.2.8. HTTP 応答の例

3.3.2.8.1. Response 200

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

3.3.2.8.2. Response 409

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

3.3.2.8.3. Response 422

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

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

3.3.3.1. 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** リクエストからの応答で返されたベース 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。

3.3.3.2. パラメーター

タイプ	Name	説明	スキーマ
パス	**groupid** **required**	コンシューマーが属するコンシューマーグループの ID。	文字列
パス	**name** **必須**	削除するコンシューマーの名前。	文字列

3.3.3.3. 応答

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。	Error

3.3.3.4. 消費

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

3.3.3.5. 生成されるもの

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

3.3.3.6. タグ

- コンシューマー

3.3.3.7. HTTP 応答の例

3.3.3.7.1. Response 404

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

3.3.4. POST /consumers/{groupid}/instances/{name}/assignments

3.3.4.1. 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。

3.3.4.2. パラメーター

タイプ	Name	説明	スキーマ
パス	groupid **required**	コンシューマーが属するコンシューマーグループの ID。	文字列
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。	文字列
ボディー	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

3.3.4.3. 応答

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ンスが見つかりませんでした。	Error
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。	Error

3.3.4.4. 消費

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

3.3.4.5. 生成されるもの

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

3.3.4.6. タグ

- コンシューマー

3.3.4.7. HTTP 要求の例

3.3.4.7.1. 要求のボディー

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

```
    "partition" : 1
  } ]
}
```

3.3.4.8. HTTP 応答の例

3.3.4.8.1. Response 404

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

3.3.4.8.2. Response 409

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

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

3.3.5.1. 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。

3.3.5.2. パラメーター

タイプ	Name	説明	スキーマ
パス	**groupid** **required**	コンシューマーが属するコンシューマーグループの ID。	文字列
パス	**name** **必須**	コンシューマーの名前。	文字列
ボディー	**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

3.3.5.3. 応答

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。	Error

3.3.5.4. 消費

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

3.3.5.5. 生成されるもの

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

3.3.5.6. タグ

- コンシューマー

3.3.5.7. HTTP 要求の例

3.3.5.7.1. 要求のボディー

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

3.3.5.8. HTTP 応答の例

3.3.5.8.1. Response 404

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

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

3.3.6.1. 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ティショ

ンを指定できます。

3.3.6.2. パラメーター

タイプ	Name	説明	スキーマ
パス	**groupid** **required**	コンシューマーが属するコンシューマーグループの ID。	文字列
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。	文字列
ボディー	**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

3.3.6.3. 応答

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。	Error

3.3.6.4. 消費

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

3.3.6.5. 生成されるもの

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

3.3.6.6. タグ

- コンシューマー
- Seek

3.3.6.7. HTTP 要求の例

3.3.6.7.1. 要求のボディー

```
{
  "offsets" : [ {
    "topic" : "topic",
    "partition" : 0,
    "offset" : 15
```

```
  }, {
    "topic" : "topic",
    "partition" : 1,
    "offset" : 42
  } ]
}
```

3.3.6.8. HTTP 応答の例

3.3.6.8.1. Response 404

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

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

3.3.7.1. 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。

3.3.7.2. パラメーター

タイプ	Name	説明	スキーマ
パス	**groupid** **required**	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。	文字列
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。	文字列
ボディー	**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

3.3.7.3. 応答

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
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。	Error

3.3.7.4. 消費

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

3.3.7.5. 生成されるもの

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

3.3.7.6. タグ

- コンシューマー
- Seek

3.3.7.7. HTTP 要求の例

3.3.7.7.1. 要求のボディー

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

3.3.7.8. HTTP 応答の例

3.3.7.8.1. Response 404

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

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

3.3.8.1. 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構成します。

3.3.8.2. パラメーター

タイプ	Name	説明	スキーマ
パス	**groupid** **required**	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。	文字列

タイプ	Name	説明	スキーマ
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。	文字列
ボディー	**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探します。	Partitions

3.3.8.3. 応答

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。	Error

3.3.8.4. 消費

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

3.3.8.5. 生成されるもの

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

3.3.8.6. タグ

- コンシューマー
- Seek

3.3.8.7. HTTP 要求の例

3.3.8.7.1. 要求のボディー

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

3.3.8.8. HTTP 応答の例

3.3.8.8.1. Response 404

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

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

3.3.9.1. 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** リクエストからの応答で返されたベース 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。

3.3.9.2. パラメーター

タイプ	Name	説明	スキーマ
パス	**groupid** **required**	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。	文字列
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。	文字列
クエリー	**max_bytes** **optional**	応答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 のエラー応答が返されます。	整数
クエリー	**timeout** **optional**	HTTP Bridge が要求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

3.3.9.3. 応答

HTTP コード	説明	スキーマ
200	ポーリング要求は正常に実行されました。	ConsumerRecordList
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。	Error
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、ブリッジがトピックから JSON エンコードされていないメッセージを受け取りました。	Error
422	応答が、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	Error

3.3.9.4. 生成されるもの

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

3.3.9.5. タグ

- コンシューマー

3.3.9.6. HTTP 応答の例

3.3.9.6.1. Response 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,
  }
]
```

3.3.9.6.2. Response 404

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

```
}
```

3.3.9.6.3. Response 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."
}
```

3.3.9.6.4. Response 422

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

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

3.3.10.1. 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) リストで、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。

3.3.10.2. パラメーター

タイプ	Name	説明	スキーマ
パス	**groupid** **required**	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。	文字列
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。	文字列
ボディー	**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。	トピック

3.3.10.3. 応答

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ンスが見つかりませんでした。	Error

HTTP コード	説明	スキーマ
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。	Error
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。	Error

3.3.10.4. 消費

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

3.3.10.5. 生成されるもの

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

3.3.10.6. タグ

- コンシューマー

3.3.10.7. HTTP 要求の例

3.3.10.7.1. 要求のボディー

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

3.3.10.8. HTTP 応答の例

3.3.10.8.1. Response 404

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

3.3.10.8.2. Response 409

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

3.3.10.8.3. Response 422

```
{
  "error_code" : 422,
```

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

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

3.3.11.1. 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。

3.3.11.2. パラメーター

タイプ	Name	説明	スキーマ
パス	**groupid** **required**	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。	文字列
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。	文字列

3.3.11.3. 応答

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。	Error

3.3.11.4. 生成されるもの

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

3.3.11.5. タグ

- コンシューマー

3.3.11.6. HTTP 応答の例

3.3.11.6.1. Response 200

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

```
    "my-topic2" : [ 1 ]
  } ]
}
```

3.3.11.6.2. Response 404

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

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

3.3.12.1. 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。

3.3.12.2. パラメーター

タイプ	Name	説明	スキーマ
パス	**groupid** **required**	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。	文字列
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。	文字列

3.3.12.3. 応答

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ンスが見つかりませんでした。	Error

3.3.12.4. タグ

- コンシューマー

3.3.12.5. HTTP 応答の例

3.3.12.5.1. Response 404

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

3.3.13. GET /healthy

3.3.13.1. 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。

3.3.13.2. 応答

HTTP コード	説明	スキーマ
200	ブリッジは問題なし	コンテンツなし

3.3.14. GET /openapi

3.3.14.1. 説明

OpenAPI v2 仕様を JSON 形式で取得します。

3.3.14.2. 応答

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。	文字列

3.3.14.3. 生成されるもの

- **application/json**

3.3.15. GET /ready

3.3.15.1. 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。

3.3.15.2. 応答

HTTP コード	説明	スキーマ
200	ブリッジの準備完了	コンテンツなし

3.3.16. GET /topics

3.3.16.1. 説明

すべてのトピックのリストを取得します。

3.3.16.2. 応答

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

3.3.16.3. 生成されるもの

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

3.3.16.4. タグ

- トピック

3.3.16.5. HTTP 応答の例

3.3.16.5.1. Response 200

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

3.3.17. POST /topics/{topicname}

3.3.17.1. 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。

3.3.17.2. パラメーター

タイプ	Name	説明	スキーマ
パス	**topicname** **required**	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。	文字列
ボディー	**body** **必須**		ProducerRecordList

3.3.17.3. 応答

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。	OffsetRecordSentList
404	指定されたトピックが見つかりませんでした。	Error
422	レコードリストが無効です。	Error

3.3.17.4. 消費

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

3.3.17.5. 生成されるもの

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

3.3.17.6. タグ

- プロデューサー
- トピック

3.3.17.7. HTTP 要求の例

3.3.17.7.1. 要求のボディー

```
{
  "records" : [ {
    "key" : "key1",
    "value" : "value1"
  }, {
    "value" : "value2",
    "partition" : 1
  }, {
    "value" : "value3"
  } ]
}
```

3.3.17.8. HTTP 応答の例

3.3.17.8.1. Response 200

```
{
  "offsets" : [ {
    "partition" : 2,
```

```
    "offset" : 0
  }, {
    "partition" : 1,
    "offset" : 1
  }, {
    "partition" : 2,
    "offset" : 2
  } ]
}
```

3.3.17.8.2. Response 404

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

3.3.17.8.3. Response 422

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

3.3.18. GET /topics/{topicname}

3.3.18.1. 説明

特定のトピックに関するメタデータを取得します。

3.3.18.2. パラメーター

タイプ	Name	説明	スキーマ
パス	topicname required	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。	文字列

3.3.18.3. 応答

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

3.3.18.4. 生成されるもの

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

3.3.18.5. タグ

- トピック

3.3.18.6. HTTP 応答の例

3.3.18.6.1. Response 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
    } ]
  } ]
}
```

3.3.19. GET /topics/{topicname}/partitions

3.3.19.1. 説明

トピックのパーティションのリストを取得します。

3.3.19.2. パラメーター

タイプ	Name	説明	スキーマ

タイプ	Name	説明	スキーマ
パス	topicname required	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。	文字列

3.3.19.3. 応答

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

3.3.19.4. 生成されるもの

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

3.3.19.5. タグ

- トピック

3.3.19.6. HTTP 応答の例

3.3.19.6.1. Response 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
  } ]
} ]
```

3.3.19.6.2. Response 404

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

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

3.3.20.1. 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。

3.3.20.2. パラメーター

タイプ	Name	説明	スキーマ
パス	**partitionid** **required**	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** **required**	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。	文字列
ボディー	**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

3.3.20.3. 応答

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。	OffsetRecordSentList
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。	Error
422	レコードが無効です。	Error

3.3.20.4. 消費

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

3.3.20.5. 生成されるもの

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

3.3.20.6. タグ

- プロデューサー
- トピック

3.3.20.7. HTTP 要求の例

3.3.20.7.1. 要求のボディー

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

3.3.20.8. HTTP 応答の例

3.3.20.8.1. Response 200

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

3.3.20.8.2. Response 404

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

3.3.20.8.3. Response 422

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

```
}
```

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

3.3.21.1. 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。

3.3.21.2. パラメーター

タイプ	Name	説明	スキーマ
パス	partitionid required	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 required	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。	文字列

3.3.21.3. 応答

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。	Error

3.3.21.4. 生成されるもの

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

3.3.21.5. タグ

- トピック

3.3.21.6. HTTP 応答の例

3.3.21.6.1. Response 200

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

```
    "leader" : false,
    "in_sync" : true
  } ]
}
```

3.3.21.6.2. Response 404

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

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

3.3.22.1. 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。

3.3.22.2. パラメーター

タイプ	Name	説明	スキーマ
パス	partitionid required	パーティションの ID。	整数
パス	topicname required	パーティションを含むトピックの名前。	文字列

3.3.22.3. 応答

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。	Error

3.3.22.4. 生成されるもの

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

3.3.22.5. タグ

- トピック

3.3.22.6. HTTP 応答の例

3.3.22.6.1. Response 200

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

3.3.22.6.2. Response 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. **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、**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。

改訂日時： 2022-07-12 13:29:11 +1000