Red Hat Streams for Apache Kafka 2.7

使用 Apache Kafka Bridge 的流

使用 AMQ Streams Kafka Bridge 与 Kafka 集群连接

Last Updated: 2024-05-29

Red Hat Streams for Apache Kafka 2.7 使用 Apache Kafka Bridge 的流

使用 AMQ Streams Kafka Bridge 与 Kafka 集群连接

法律通告

摘要

AMQ Streams Kafka Bridge 为基于 HTTP 的客户端提供了一个 RESTful 接口，以便与 Kafka 集群交互。

目录

前言

对红帽文档提供反馈

我们感谢您对我们文档的反馈。

要改进，创建一个 JIRA 问题并描述您推荐的更改。提供尽可能多的详细信息，以便我们快速解决您的请求。

前提条件

- 您有红帽客户门户网站帐户。此帐户可让您登录到 Red Hat Jira Software 实例。
  如果您没有帐户，系统会提示您创建一个帐户。

流程

1. 点以下内容： Create issue。
2. 在 **Summary** 文本框中输入问题的简短描述。
3. 在 **Description** 文本框中提供以下信息：
   - 找到此问题的页面的 URL。
   - 有关此问题的详细描述。
     您可以将信息保留在任何其他字段中的默认值。
4. 添加 reporter 名称。
5. 点 **Create** 将 JIRA 问题提交到文档团队。

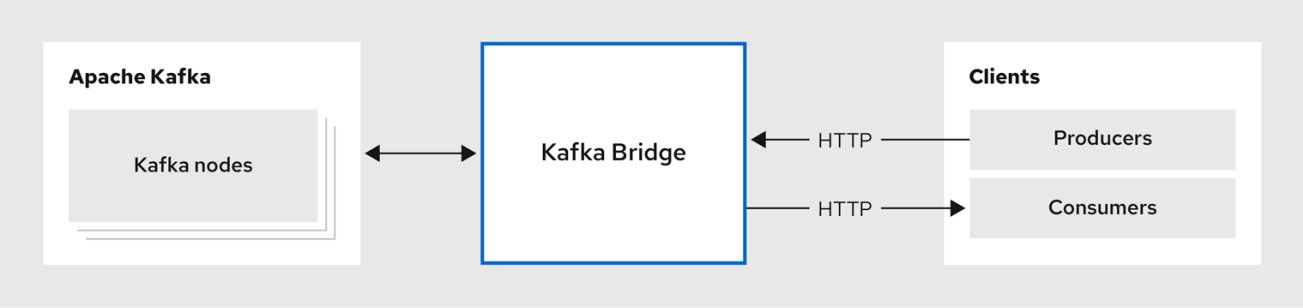
感谢您花时间来提供反馈。

第 1 章 KAFKA BRIDGE 概述

使用 Apache Kafka Bridge 的 Streams 向 Kafka 集群发出 HTTP 请求。

您可以使用 Kafka Bridge 将 HTTP 客户端应用程序与 Kafka 集群集成。

HTTP 客户端集成



1.1. 运行 KAFKA BRIDGE

安装 Apache Kafka Bridge 的 Streams，以便在与 Kafka 集群相同的环境中运行。

您可以将 Kafka Bridge 安装工件下载到主机机器中。要在本地环境中尝试 Kafka Bridge，请参阅 Kafka Bridge quickstart。

务必要注意，Kafka Bridge 的每个实例都维护自己的一组内存消费者（和订阅），它们代表 HTTP 客户端连接到 Kafka Broker。这意味着，每个 HTTP 客户端都必须维护同一 Kafka Bridge 实例的关联性，才能访问创建的任何订阅。另外，当 Kafka Bridge 的实例时，内存消费者和订阅将会丢失。**如果 Kafka Bridge 重启，HTTP 客户端负责重新创建任何使用者和订阅。**

1.1.1. 在 OpenShift 上运行 Kafka Bridge

如果在 OpenShift 上部署了 Apache Kafka 的 Streams，您可以使用 Apache Kafka Cluster Operator 的 Streams 将 Kafka Bridge 部署到 OpenShift 集群。配置和部署 Kafka Bridge 作为 **KafkaBridge** 资源。您需要一个正在运行的 Kafka 集群，该集群由 Cluster Operator 在 OpenShift 命名空间中部署。您可以配置部署以访问 OpenShift 集群外的 Kafka Bridge。

HTTP 客户端必须保持与 Kafka Bridge 相同的实例的关联性，才能访问它们创建的任何用户或订阅。因此，不建议每个 OpenShift Deployment 运行多个 Kafka Bridge 副本。如果 Kafka Bridge pod 重启（例如，因为 OpenShift 将工作负载重新定位到另一节点），则 HTTP 客户端必须重新创建任何消费者或订阅。

有关部署和配置 Kafka Bridge 作为 **KafkaBridge** 资源的详情，请参考 Apache Kafka 文档的流。

1.2. KAFKA BRIDGE 接口

Kafka Bridge 提供了一个 RESTful 接口，它允许基于 HTTP 的客户端与 Kafka 集群交互。 它提供与 Apache Kafka 的 Streams 的 Web API 连接的优点，而无需客户端应用程序来解释 Kafka 协议。

API 有两个主要资源 - **消费者（consumer）**和**主题（topic）** - 通过端点公开并可访问，以便与 Kafka 集群中的消费者和制作者交互。资源只与 Kafka Bridge 相关，而不是直接连接到 Kafka 的用户和制作者。

1.2.1. HTTP 请求

Kafka Bridge 支持对 Kafka 集群的 HTTP 请求，使用以下方法：

- 发送消息到主题。
- 从主题检索消息。
- 检索主题的分区列表。
- 创建和删除用户。
- 订阅消费者到主题，以便他们开始从这些主题接收信息。
- 检索消费者订阅的主题列表。
- 取消订阅消费者的主题.
- 为消费者分配分区。
- 提交使用者偏移列表。
- 定位分区，以便消费者开始收到来自第一或最后一个偏移位置的信息，或给定的偏移位置。

该方法提供 JSON 响应和 HTTP 响应代码错误处理。消息可以使用 JSON 或二进制格式发送。

客户端可以在不需要使用原生 Kafka 协议的情况下生成和使用消息。

其他资源

- Apache Kafka Bridge API 参考流

1.3. KAFKA BRIDGE OPENAPI 规格

Kafka Bridge API 使用 OpenAPI 规格(OAS)。OAS 提供用于描述和实施 HTTP API 的标准框架。

Kafka Bridge OpenAPI 规格采用 JSON 格式。您可以在 Kafka Bridge 源下载文件的 **src/main/resources/** 文件夹中找到 OpenAPI JSON 文件。可以通过客户门户网站下载文件。

您还可以使用 **GET /openapi** 方法 以 JSON 格式检索 OpenAPI v2 规格。

其他资源

- OpenAPI 计划

1.4. 保护到 KAFKA 集群的连接

您可以在 Kafka Bridge 和 Kafka 集群间配置以下内容：

- 基于 TLS 或 SASL 的身份验证
- TLS 加密的连接

您可以通过其 属性文件 配置 Kafka Bridge 进行身份验证。

您还可以在 Kafka 代理中使用 ACL 来限制可使用 Kafka Bridge 消耗和生成的主题。

注意

在 OpenShift 上运行 Kafka Bridge 时，使用 **KafkaBridge** 资源配置身份验证。

1.5. 保护 KAFKA BRIDGE HTTP 接口

Kafka Bridge 不支持 HTTP 客户端和 Kafka Bridge 之间的身份验证和加密。从客户端发送到 Kafka Bridge 的请求在没有身份验证或加密的情况下发送。请求必须使用 HTTP 而不是 HTTPS。

您可以将 Kafka Bridge 与以下工具相结合来保护它：

- 定义哪些 pod 可以访问 Kafka Bridge 的网络策略和防火墙
- 反向代理（例如，OAuth 2.0）
- API 网关

1.6. 对 KAFKA BRIDGE 的请求

指定数据格式和 HTTP 标头，以确保将有效的请求提交到 Kafka Bridge。

1.6.1. 内容类型标头

API 请求和响应正文始终编码为 JSON。

- 在执行消费者操作时，如果有一个非空正文，**POST** 请求必须提供以下 **Content-Type** 标头：

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

- 在执行制作者操作时，**POST** 请求必须提供 **Content-Type** 标头来指定 *生成的消息的嵌入式数据格式*。这可以是 **json**、**二进制** 或**文本**。

嵌入式数据格式	Content-Type 标头
JSON	**content-Type: application/vnd.kafka.json.v2+json**
二进制	**content-Type: application/vnd.kafka.binary.v2+json**
文本	**content-Type: application/vnd.kafka.text.v2+json**

嵌入式数据格式是为每个消费者设置的，如下一节中所述。

如果 **POST** 请求带有空的正文，则一定 *不能* 设置 **Content-Type**。可以使用空正文来创建具有默认值的消费者。

1.6.2. 嵌入式数据格式

嵌入式数据格式是通过 HTTP 传输的 Kafka 消息的格式，使用 Kafka Bridge 将生产者传输到消费者。支持三种嵌入式数据格式：JSON、二进制和文本。

当使用 **/consumers/*groupid*** 端点创建消费者时，**POST** 请求正文必须指定 JSON、二进制或文本的嵌入式数据格式。这在 **format** 字段中指定，例如：

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

1 二进制嵌入式数据格式。

创建消费者时指定的嵌入式数据格式必须与它将使用的 Kafka 信息的数据格式匹配。

如果您选择指定二进制嵌入的数据格式，后续的制作者请求必须在请求正文中提供二进制数据作为 Base64 编码的字符串。例如，当使用 **/topics/*topicname*** 端点发送消息时，**records.value** 必须使用 Base64 编码：

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

制作者请求还必须提供与嵌入式数据格式对应的 **Content-Type** 标头，如 **Content-Type: application/vnd.kafka.binary.v2+json**。

1.6.3. 消息格式

使用 **/topics** 端点发送消息时，您可以在请求正文中输入消息有效负载，在 **records** 参数中输入消息有效负载。

records 参数可以包含这些可选字段：

- **消息标头**
- message **键**
- 消息 **值**
- 目标 **分区**

到 /topics 的 POST 请求示例

```
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 二进制格式的标头值，并编码为 Base64。

请注意，如果您的使用者配置为使用文本嵌入的数据格式，则 **records** 参数中的 **value** 和 **key** 字段必须是字符串，而不是 JSON 对象。

1.6.4. 接受标头

创建消费者后，所有后续 GET 请求都必须以以下格式提供 **Accept** 标头：

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

EMBEDDED-DATA-FORMAT 是 json、二进制 或文本。

例如，当使用 JSON 的嵌入式数据格式检索订阅消费者的记录时，请包含此 Accept 标头：

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

1.7. CORS

通常，HTTP 客户端无法在不同域中发布请求。

例如，假设 Kafka Bridge 您与 Kafka 集群一起部署，可以使用 **http://my-bridge.io** 域访问。HTTP 客户端可以使用 URL 与 Kafka Bridge 交互，并通过 Kafka 集群交换信息。但是，您的客户端作为 **http://my-web-application.io** 域中的 Web 应用程序运行。客户端（源）域与 Kafka Bridge （目标）域不同。由于同一原始策略限制，来自客户端的请求会失败。您可以使用 Cross-Origin Resource Sharing (CORS)来避免这种情况。

CORS 允许在不同的域中的原始源之间的 *simple* 和 *preflighted* 请求。

简单的请求适用于使用 **GET**、**HEAD**、**POST** 方法的标准请求。

preflighted 请求发送 *HTTP OPTIONS* 请求，作为初始检查实际请求是否安全发送。在确认时，会发送实际请求。preflight 请求适合需要更大的保护方法，如 **PUT** 和 **DELETE**，并使用非标准标头。

所有请求都需要在其标头中有一个 *origin* 值，即 HTTP 请求的来源。

CORS 允许您指定允许的方法和原始 URL，以便在 Kafka Bridge HTTP 配置中访问 Kafka 集群。

Kafka Bridge 的 CORS 配置示例

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

1.7.1. 简单请求

例如：这个简单请求标头将源指定为 **http://my-web-application.io**。

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

标头信息添加到请求中，以消耗记录。

```
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'
```

在 Kafka Bridge 的响应中，会返回 **Access-Control-Allow-Origin** 标头。它包含可以从向网桥发出 HTTP 请求的域列表。

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

1. 返回星号(*)显示可由任何域访问的资源。

1.7.2. preflighted 请求

初始 preflight 请求使用 **OPTIONS** 方法发送到 Kafka Bridge。 *HTTP OPTIONS* 请求发送标头信息，以检查 Kafka Bridge 是否允许实际请求。

此处的 preflight 请求检查 **POST** 请求是否从 **http://my-web-application.io** 有效。

```
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 会警告实际请求是 **POST** 请求。
2. 实际请求将附带 **Content-Type** 标头。

OPTIONS 添加到 preflight 请求的标头信息中。

```
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 响应初始请求，以确认请求被接受。响应标头返回允许的源、方法和标头。

```
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
```

如果 origin 或 方法被拒绝，则返回错误消息。

实际请求不需要 **Access-Control-Request-Method** 标头，因为它在 preflight 请求中已确认，但它需要 origin 标头。

```
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'
```

响应显示允许原始 URL。

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

其他资源

- 获取 CORS 规格

1.8. 为 KAFKA BRIDGE 配置日志记录器

您可以为 Kafka Bridge OpenAPI 规格定义的每个操作设置不同的日志级别。

每个操作都有一个对应的 API 端点，网桥通过该端点从 HTTP 客户端接收请求。您可以更改每个端点的日志级别，以生成有关传入和传出 HTTP 请求的更多或更精细的日志信息。

日志记录器在 **log4j2.properties** 文件中定义，该文件对 **healthy** 和 **ready** 端点有以下默认配置：

```
logger.healthy.name = http.openapi.operation.healthy
logger.healthy.level = WARN
logger.ready.name = http.openapi.operation.ready
logger.ready.level = WARN
```

所有其他操作的日志级别默认设置为 **INFO**。日志记录器的格式如下：

```
logger.<operation_id>.name = http.openapi.operation.<operation_id>
logger.<operation_id>_level = _<LOG_LEVEL>
```

其中 ***<operation_id>*** 是特定操作的标识符。

OpenAPI 规格定义的操作列表

- **createConsumer**
- **deleteConsumer**
- **订阅**
- **unsubscribe**
- **poll**
- **分配**
- **commit**
- **send**

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

其中 *<LOG_LEVEL >* 是 log4j2 定义的日志级别（例如 **INFO**,**DEBUG**, ...）。

第 2 章 KAFKA BRIDGE QUICKSTART

使用此快速入门来在本地开发环境中尝试 Apache Kafka Bridge 的流。

您将了解如何进行以下操作：

- 生成信息到 Kafka 集群中的主题和分区
- 创建 Kafka Bridge 使用者
- 执行基本的消费者操作，如将消费者订阅到主题并检索您生成的消息

在这个快速入门中，HTTP 请求被格式化为 curl 命令，您可以复制并粘贴到终端。

确定您有先决条件，然后按照本章中提供的顺序按照任务进行操作。

在本快速入门中，您将以 JSON 格式生成和使用消息。

Quickstart 的先决条件

- Kafka 集群在主机机器上运行。

2.1. 下载 KAFKA BRIDGE 归档

可供下载 Apache Kafka Bridge 的 Streams 的 zipped 发行版。

流程

- 从客户门户网站下载 Apache Kafka Bridge 归档的最新版本。

2.2. 安装 KAFKA BRIDGE

使用 Kafka Bridge 归档提供的脚本来安装 Kafka Bridge。安装存档提供的 **application.properties** 文件提供默认配置设置。

以下默认属性值将 Kafka Bridge 配置为侦听端口 8080 上的请求。

默认配置属性

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

先决条件

- Kafka Bridge 安装存档下载

流程

1. 如果您还没有这样做，请将 Kafka Bridge 安装存档解压缩到任何目录中。
2. 使用配置属性作为参数运行 Kafka Bridge 脚本：
   例如：

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

3. 检查日志中的安装是否成功。

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

接下来要做什么

- 生成消息到主题和分区。

2.3. 向主题和分区生成信息

使用 Kafka Bridge 使用 topics 端点向 JSON 格式的 Kafka 主题生成信息。

您可以使用 topics 端点将消息生成到 JSON 格式的主题。您可以为请求正文中的消息指定目标分区。分区 端点提供了一种替代方法，为所有消息指定单一目标分区作为路径参数。

在此过程中，消息被生成为名为 **bridge-quickstart-topic** 的主题。

先决条件

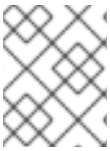
- Kafka 集群有一个具有三个分区的主题。
  您可以使用 **kafka-topics.sh** 实用程序创建主题。

  具有三个分区的主题创建示例

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

  验证主题是否已创建

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



注意

如果在 OpenShift 上部署了 Apache Kafka 的 Streams，您可以使用 **KafkaTopic** 自定义资源创建一个主题。

流程

1. 使用 Kafka Bridge，为您创建的主题生成三个信息：

```
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** 发送到基于密钥哈希的分区。
- **Sales-lead-0002** 直接发送到分区 2。
- **Sales-lead-0003** 使用 round-robin 方法发送到 **bridge-quickstart-topic** 主题中的分区。

2. 如果请求成功，Kafka Bridge 会返回一个 **偏移** 数组，以及 **application/vnd.kafka.v2+json** 的 **content-type** 标头。对于每个消息，**偏移** 数组描述：

- **消息发送到的分区**
- **分区的当前消息偏移**

响应示例

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

其他主题请求

发出其他 curl 请求以查找有关主题和分区的信息。

列出主题

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

响应示例

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

获取主题配置和分区详情

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

响应示例

```
{
  "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
     }
    ]
   }
  ]
}
```

列出特定主题的分区

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

响应示例

```
[
  {
    "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
    }
   ]
  }
]
```

列出特定主题分区的详情

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

响应示例

```
{
  "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
    }
  ]
}
```

列出特定主题分区的偏移量

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

响应示例

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

接下来要做什么

在生成消息到主题和分区后，创建一个 Kafka Bridge 消费者。

其他资源

- POST /topics/{topicname}
- POST /topics/{topicname}/partitions/{partitionid}

2.4. 创建 KAFKA BRIDGE 使用者

在 Kafka 集群中执行任何消费者操作前，您必须首先使用 consumers 端点创建消费者。消费者被称为 *Kafka Bridge 消费者*。

流程

1. 在名为 bridge-quickstart-consumer-group 的新消费者组中创建一个 Kafka Bridge 使用者：

```
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
  }'
```

- 消费者名为 bridge-quickstart-consumer，嵌入式数据格式被设置为 json。
- 定义了一些基本配置设置。
- 消费者不会自动向日志提交偏移，因为 enable.auto.commit 设置为 false。稍后您将在此快速入门中手动提交偏移量。

如果请求成功，Kafka Bridge 会返回响应正文中的使用者 ID (instance_id)和基本 URL (base_uri)，以及 200 代码。

响应示例

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

2. 复制基础 URL (base_uri)，以便在这个快速入门中的其他消费者操作中使用。

接下来要做什么

现在，您已创建了 Kafka Bridge 消费者，您可以将其订阅到主题。

其他资源

- POST /consumers/{groupid}

2.5. 将 KAFKA BRIDGE 消费者订阅到主题

创建 Kafka Bridge 消费者后，使用 订阅 端点将其订阅到一个或多个主题。订阅后，消费者开始接收生成到该主题的所有消息。

流程

- 将消费者订阅之前创建的 bridge-quickstart-topic 主题，在 Produc ing 信息到主题和分区：

```
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 数组可以包含单个主题（如此处所示）或多个主题。如果要将消费者订阅与正则表达式匹配的多个主题，您可以使用 topic_pattern 字符串而不是 topics 数组。

如果请求成功，Kafka Bridge 只会返回一个 204 (No Content)代码。

当使用 Apache Kafka 客户端时，HTTP 订阅操作会为本地消费者的订阅添加主题。加入消费者组，并在运行多个 HTTP 轮询操作后获取分区分配，开始分区重新平衡和加入-组进程。务必要注意，初始 HTTP 轮询操作可能无法返回任何记录。

接下来要做什么

将 Kafka Bridge 消费者订阅到主题后，您可以从 消费者检索信息。

其他资源

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

2.6. 从 KAFKA BRIDGE CONSUMER 检索最新的信息

通过从 records 端点请求数据，从 Kafka Bridge 使用者检索最新的消息。在生产环境中，HTTP 客户端可以重复调用此端点（在循环中）。

流程

1. 为 Kafka Bridge consumer 生成额外消息，如 Produc ing 消息到主题和分区 中所述。

2. 向 记录 端点提交 GET 请求：

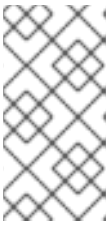
```
curl -X GET http://localhost:8080/consumers/bridge-quickstart-consumer-group/instances/bridge-quickstart-consumer/records \
  -H 'accept: application/vnd.kafka.json.v2+json'
```

创建并订阅 Kafka Bridge 消费者后，第一个 GET 请求将返回空响应，因为轮询操作会启动一个重新平衡过程来分配分区。

3. 重复步骤 2，以从 Kafka Bridge 使用者检索信息。

Kafka Bridge 返回一条信息 数组，它带有 200 代码，它包括主题名称、键、值、分区、分区以及 200 代码。默认情况下，消息从最新的偏移中检索。

```
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
  },
#...
```



注意

如果返回空响应，请生成更多记录到消费者，如 Produc ing 消息到主题和分区 中所述，然后尝试再次检索消息。

接下来要做什么

从 Kafka Bridge 消费者检索信息后，尝试向 日志提交偏移。

其他资源

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

2.7. 向日志提交偏移量

使用 偏移 端点手动向日志提交 Kafka Bridge 消费者接收的所有消息。这是必要的，因为在 创建 Kafka Bridge 消费者 中的 Kafka Bridge 消费者 被配置，其 enable.auto.commit 设置为 false。

流程

- 将偏移提交到 bridge-quickstart-consumer 的日志：

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

因为没有提交请求正文，所以针对消费者收到的所有记录提交偏移。另外，请求正文可以包含数组(OffsetCommitSeekList)，用于指定您要为其提交偏移的主题和分区。

如果请求成功，Kafka Bridge 只会返回一个 204 代码。

接下来要做什么

向日志提交偏移后，尝试查找查找 偏移的端点。

其他资源

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

2.8. 寻找分区偏移量

使用 位置 端点配置 Kafka Bridge 消费者，从特定偏移中检索分区的消息，然后从最新的偏移中检索。这在 Apache Kafka 中被称为 seek 操作。

流程

1. 请参阅 quickstart-bridge-topic 主题的分区 0 的特定偏移：

```
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
        }
    ]
}'
```

如果请求成功，Kafka Bridge 只会返回一个 204 代码。

2. 向 记录 端点提交 GET 请求：

```
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 从您看到的偏移返回信息。

3. 通过查找同一分区的最后一个偏移来恢复默认消息检索行为。这一次，使用 位置/结束 端点。

```
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
      }
    ]
}'
```

如果请求成功，Kafka Bridge 会返回另一个 204 代码。

注意

您还可以使用 positions/beginning 端点查找一个或多个分区的第一个偏移。

接下来要做什么

在这个快速入门中，您使用 Apache Kafka Bridge 的 Streams 在 Kafka 集群上执行几个常见操作。现在，您可以删除之前创建的 Kafka Bridge 使用者。

其他资源

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

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

2.9. 删除 KAFKA BRIDGE 使用者

删除您在此快速入门中使用的 Kafka Bridge 使用者。

流程

- 通过向 实例 端点发送 DELETE 请求来删除 Kafka Bridge 使用者。

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

如果请求成功，Kafka Bridge 会返回 204 代码。

其他资源

- DELETE /consumers/{groupid}/instances/{name}

第 3 章 KAFKA BRIDGE 配置

使用配置属性配置 Kafka Bridge 的部署。配置 Kafka 并指定与 Kafka 交互所需的 HTTP 连接详情。您还可以使用配置属性在 Kafka Bridge 中启用和使用分布式追踪。分布式追踪允许您跟踪分布式系统中的应用程序间事务的进度。

注意

在 OpenShift 上运行 Kafka Bridge 时，使用 KafkaBridge 资源配置属性。

3.1. 配置 KAFKA BRIDGE 属性

这个步骤描述了如何配置 Kafka 和 HTTP 连接属性，供 Kafka Bridge 使用。

您可以将 Kafka Bridge 配置为任何其他 Kafka 客户端，为 Kafka 相关的属性使用适当的前缀。

- Kafka. 对于应用于生产者和消费者的常规配置，如服务器连接和安全性。
- kafka.consumer. 用于只传递给消费者的特定于消费者的配置。
- kafka.producer. for producer 特定配置仅传递给制作者。

除了启用对 Kafka 集群的 HTTP 访问外，HTTP 属性提供通过 Cross-Origin Resource Sharing (CORS)启用和定义 Kafka Bridge 的访问控制。CORS 是一种 HTTP 机制，它允许浏览器从多个来源访问所选资源。要配置 CORS，您可以定义允许的资源源和 HTTP 方法列表来访问它们。请求中的其他 HTTP 标头描述了允许访问 Kafka 集群的 CORS 来源。

先决条件

- Kafka Bridge 安装存档下载

流程

1. 编辑 Kafka Bridge 安装存档提供的 application.properties 文件。

使用属性文件指定 Kafka 和 HTTP 相关属性。

a. 配置标准 Kafka 相关属性，包括特定于 Kafka 用户和制作者的属性。

使用：

- kafka.bootstrap.servers 来定义到 Kafka 集群的主机/端口连接
- kafka.producer.acks 为 HTTP 客户端提供确认
- kafka.consumer.auto.offset.reset 来确定如何在 Kafka 中管理偏移重置

有关配置 Kafka 属性的更多信息，请参阅 Apache Kafka 网站

b. 配置与 HTTP 相关的属性，以启用对 Kafka 集群的 HTTP 访问。

例如：

```
bridge.id=my-bridge
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. Kafka Bridge 的默认 HTTP 配置来侦听端口 8080。
2. 设置为 true 以启用 CORS。
3. 允许 CORS 来源的逗号分隔列表。您可以使用 URL 或 Java 正则表达式。

4 CORS 允许用逗号分隔的 HTTP 方法列表。

2. 保存配置文件。

3.2. 配置指标

通过设置 KAFKA_bridge_METRICS_ENABLED 环境变量来启用 Kafka Bridge 的指标。

先决条件

- Kafka Bridge 安装存档下载。

流程

1. 将启用指标的环境变量设置为 true。

启用指标的环境变量

```
KAFKA_BRIDGE_METRICS_ENABLED=true
```

2. 运行 Kafka Bridge 脚本以启用指标。

运行 Kafka Bridge 以启用指标

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

启用指标后，您可以使用带有 /metrics 端点的 GET /metrics 来检索 Prometheus 格式的 Kafka Bridge 指标。

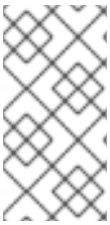
3.3. 配置分布式追踪

启用分布式追踪，追踪 Kafka Bridge 使用和生成的信息，以及来自客户端应用程序的 HTTP 请求。

启用追踪的属性存在于 application.properties 文件中。要启用分布式追踪，请执行以下操作：

- 设置 bridge.tracing 属性值，以启用您要使用的追踪。唯一可能的值是 opentelemetry。
- 设置环境变量以进行追踪。

使用默认配置时，Open OpenTelemetry tracing 使用 OTLP 作为导出器协议。通过配置 OTLP 端点，您仍然可以使用 Jaeger 后端实例来获取 trace。



注意

从版本 1.35 开始，Jaeger 支持 OTLP 协议。旧的 Jaeger 版本无法使用 OTLP 协议获得 trace。

OpenTelemetry 定义了一个 API 规格，用于收集追踪数据 *跨* 指标数据。span 代表特定的操作。trace 是一个或多个范围的集合。

当 Kafka Bridge 执行以下操作时，会生成 trace：

- 将信息从 Kafka 发送到消费者 HTTP 客户端
- 从生成者 HTTP 客户端接收发送到 Kafka 的信息

Jaeger 实现所需的 API，并在其用户界面中显示 trace 数据的视觉化，以便进行分析。

要进行端到端追踪，您必须在 HTTP 客户端中配置追踪。

小心

Apache Kafka 的流不再支持 OpenTracing。如果您之前将 OpenTracing 与 bridge.tracing=jaeger 选项搭配使用，我们建议您改为使用 OpenTelemetry。

先决条件

- Kafka Bridge 安装存档下载。

流程

1. 编辑 Kafka Bridge 安装存档提供的 application.properties 文件。

   使用 bridge.tracing 属性启用您要使用的追踪。

   启用 OpenTelemetry 的配置示例

```
bridge.tracing=opentelemetry (1)
```

(1) 启用 OpenTelemetry 的属性通过在行开头删除 # 来取消。

   启用追踪后，您可以在运行 Kafka Bridge 脚本时初始化追踪。

2. 保存配置文件。

3. 设置用于追踪的环境变量。

   OpenTelemetry 的环境变量

```
OTEL_SERVICE_NAME=my-tracing-service 1
OTEL_EXPORTER_OTLP_ENDPOINT=http://localhost:4317 2
```

1 OpenTelemetry tracer 服务的名称。

2 基于 gRPC 的 OTLP 端点，侦听端口 4317。

4. 使用启用用于追踪的属性运行 Kafka Bridge 脚本。

在启用了 OpenTelemetry 的情况下运行 Kafka Bridge

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

现在，启用了 Kafka Bridge 的内部使用者和制作者。

3.3.1. 使用 OpenTelemetry 指定追踪系统

您可以指定 OpenTelemetry 支持的其他追踪系统，而不是默认的 OTLP 追踪系统。

如果要在 OpenTelemetry 中使用另一个追踪系统，请执行以下操作：

1. 将追踪系统库添加到 Kafka 类路径。

2. 将追踪系统的名称添加为额外的 exporter 环境变量。

不使用 OTLP 时的其他环境变量

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

(1) 追踪系统的名称。在本例中，指定了 Zipkin。

(2) 侦听 span 的特定所选导出器的端点。在本例中，指定了 Zipkin 端点。

3.3.2. 支持的 Span 属性

除了标准的 OpenTelemetry 属性外，Kafka Bridge 还添加了以下属性，用于 HTTP 的 OpenTelemetry 标准惯例 到其 span。

属性键	属性值
peer.service	硬编码为 **kafka**
http.request.method	用于发出请求的 http 方法
url.scheme	URI 方案 组件
url.path	URI 路径组件
url.query	URI 查询 组件
messaging.destination.name	生成或读取的 Kafka 主题的名称
messaging.system	硬编码为 **kafka**

http.response.status_code	**OK** 用于 200 到 300 之间的 http 响应。所有其他状态代码 **的错误**

其他资源

- OpenTelemetry 导出器值

第 4 章 APACHE KAFKA BRIDGE API 参考流

4.1. 概述

Apache Kafka Bridge 的 Streams 提供了一个 REST API，用于将基于 HTTP 的客户端应用程序与 Kafka 集群集成。您可以使用 API 来创建和管理消费者，并通过 HTTP 而不是原生 Kafka 协议发送和接收记录。

4.1.1. 版本信息

版本 : 0.1.0

4.1.2. Tags

- Users : Consumer operations to create consumers in your Kafka cluster 并执行常见操作，如订阅主题、检索已处理记录和提交偏移。
- producer : Producer 操作，将记录发送到指定的主题或主题分区。
- seek : Seek 操作使消费者开始从给定偏移位置接收消息。
- 主题 ：将消息发送到指定主题或主题分区的主题操作，可以选择在请求中包含消息键。您还可以检索主题和主题元数据。

4.1.3. 使用

- application/json

4.1.4. 生成

- application/json

4.2. 定义

4.2.1. AssignedTopicPartitions

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

4.2.2. BridgeInfo

有关 Kafka Bridge 实例的信息。

名称	模式
bridge_version *optional*	字符串

4.2.3. 消费者

名称	描述	模式
auto.offset.reset *optional*	重置消费者的偏移位置。如果设置为 **latest** （默认），则会从最新的偏移中读取消息。如果设置为 **最早** 的，则消息将从第一个偏移中读取。	字符串
consumer.request.timeout.ms *optional*	设置消费者等待请求消息的最大时间（以毫秒为单位）。如果在没有响应的情况下达到超时时间，则返回错误。默认为 **30000** (30 秒)。	整数
enable.auto.commit *optional*	如果设置为 **true** （默认），则消息偏移会自动为消费者提交。如果设置为 **false**，则必须手动提交消息偏移。	布尔值
fetch.min.bytes *optional*	设置要接收的消费者的最小数据量（以字节为单位）。代理会等待数据发送超过这个数量。默认为 **1** 字节。	整数
格式 *可选*	consumer 的允许消息格式，可以是 **二进制** （默认）或 **json**。消息转换为 JSON 格式。	字符串
isolation.level *可选*	如果设置为 **read_uncommitted** （默认），则检索所有事务记录，以免于任何事务结果。如果设置为 **read_committed**，则会检索来自已提交事务的记录。	字符串
name *可选*	消费者实例的唯一名称。名称在消费者组范围内是唯一的。该名称在 URL 中使用。如果没有指定名称，则会分配一个随机生成的名称。	字符串

4.2.4. ConsumerRecord

名称	模式
标头 *可选*	KafkaHeaderList
偏移 *可选*	integer (int64)
分区 *可选*	integer (int32)
主题 *可选*	字符串

4.2.5. ConsumerRecordList

type **: < ConsumerRecord > array**

4.2.6. CreatedConsumer

名称	描述	模式
base_uri *optional*	用于构建针对此消费者实例的后续请求的 URI 的基础 URI。	字符串
instance_id *optional*	组中消费者实例的唯一 ID。	字符串

4.2.7. Error

名称	模式
error_code *optional*	integer (int32)
message *可选*	字符串

4.2.8. KafkaHeader

名称	描述	模式
key *必需*		字符串
*必需*的值	标头值采用二进制格式, base64 编码的 Pattern : **"^(?:[A-Za-z0-9+/]{4})*(?:[A-Za-z0-9+/]{2}==|[A-Za-z0-9+/]{3}=)?$"**	字符串（字节）

4.2.9. KafkaHeaderList

type **: <** KafkaHeader **> array**

4.2.10. OffsetCommitSeek

名称	模式
*需要*偏移	integer (int64)
*需要*分区	integer (int32)
主题 *必需*	字符串

4.2.11. OffsetCommitSeekList

名称	模式
偏移 *可选*	< OffsetCommitSeek > 数组

4.2.12. OffsetRecordSent

名称	模式
偏移 *可选*	integer (int64)
分区 *可选*	integer (int32)

4.2.13. OffsetRecordSentList

名称	模式
偏移 *可选*	< OffsetRecordSent > array

4.2.14. OffsetsSummary

名称	模式
beginning_offset *optional*	integer (int64)
end_offset *optional*	integer (int64)

4.2.15. 分区

名称	模式
分区 *可选*	integer (int32)
主题 *可选*	字符串

4.2.16. PartitionMetadata

名称	模式
领导 *可选*	integer (int32)
分区 *可选*	integer (int32)
replicas *可选*	&lt ; replica > 数组

4.2.17. 分区

名称	模式
分区 *可选*	&lt ; partition > 数组

4.2.18. ProducerRecord

名称	模式
标头 *可选*	KafkaHeaderList
分区 *可选*	integer (int32)

4.2.19. ProducerRecordList

名称	模式
记录 *可选*	< ProducerRecord > array

4.2.20. ProducerRecordToPartition

名称	模式
标头 *可选*	KafkaHeaderList

4.2.21. ProducerRecordToPartitionList

名称	模式
记录 *可选*	< ProducerRecordToPartition > array

4.2.22. replica

名称	模式
代理 *可选*	integer (int32)
in_sync *optional*	布尔值
领导 *可选*	布尔值

4.2.23. SubscribedTopicList

名称	模式
分区 *可选*	< AssignedTopicPartitions > 数组
主题 *可选*	topics

4.2.24. TopicMetadata

名称	描述	模式
configs *可选*	每个主题配置覆盖	< string, string > map
name *可选*	主题的名称	字符串
分区 *可选*		< PartitionMetadata > 数组

4.2.25. topics

名称	描述	模式
topic_pattern *optional*	用于匹配多个主题的 regex 主题模式	字符串
主题 *可选*		< string > 数字

4.3. 路径

4.3.1. GET /

4.3.1.1. 描述

以 JSON 格式检索有关 Kafka Bridge 实例的信息。

4.3.1.2. 响应

HTTP 代码	描述	模式
200	有关 Kafka Bridge 实例的信息。	BridgeInfo

4.3.1.3. 生成

- **application/json**

4.3.1.4. HTTP 响应示例

4.3.1.4.1. 响应 200

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

4.3.2. POST /consumers/{groupid}

4.3.2.1. 描述

在给定的消费者组中创建一个消费者实例。您可以选择指定消费者名称和支持的配置选项。它返回一个基础 URI，它必须用于构建针对此消费者实例的后续请求的 URL。

4.3.2.2. 参数

类型	Name	描述	模式
路径	GroupId *必需*	创建消费者的消费者组的 ID。	字符串
Body	body *必需*	消费者的名称和配置。名称在消费者组范围内是唯一的。如果没有指定名称，则会分配一个随机生成的名称。所有参数都是可选的。以下示例中显示了支持的配置选项。	消费者

4.3.2.3. 响应

HTTP 代码	描述	模式
200	消费者创建成功。	CreatedConsumer

HTTP 代码	描述	模式
409	Kafka Bridge 中已存在具有指定名称的消费者实例。	Error
422	一个或多个消费者配置选项具有无效的值。	Error

4.3.2.4. 使用

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

4.3.2.5. 生成

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

4.3.2.6. Tags

- **消费者**

4.3.2.7. HTTP 请求示例

4.3.2.7.1. 请求正文

```
{
  "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 响应示例

4.3.2.8.1. 响应 200

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

4.3.2.8.2. 响应 409

-

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

4.3.2.8.3. 响应 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. 描述

删除指定的消费者实例。此操作的请求必须使用响应中返回的基本 URL （包括主机和端口），从 POST 请求返回到用于创建此消费者的 /consumers/{groupid}。

4.3.3.2. 参数

类型	Name	描述	模式
路径	GroupId *必需*	消费者所属的消费者组的 ID。	字符串
路径	name *必需*	要删除的消费者的名称。	字符串

4.3.3.3. 响应

HTTP 代码	描述	模式
204	消费者已被成功删除。	无内容
404	未找到指定的消费者实例。	Error

4.3.3.4. 使用

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

4.3.3.5. 生成

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

4.3.3.6. Tags

- **消费者**

4.3.3.7. HTTP 响应示例

4.3.3.7.1. 响应 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. 描述

为消费者分配一个或多个主题分区。

4.3.4.2. 参数

类型	Name	描述	模式
路径	GroupId *必需*	消费者所属的消费者组的 ID。	字符串
路径	name *必需*	将主题分区分配到的消费者名称。	字符串
Body	body *必需*	要分配给消费者的主题分区列表。	分区

4.3.4.3. 响应

HTTP 代码	描述	模式
204	成功分配分区。	无内容
404	未找到指定的消费者实例。	Error

HTTP 代码	描述	模式
409	主题、分区和模式订阅是互斥的。	Error

4.3.4.4. 使用

- application/vnd.kafka.v2+json

4.3.4.5. 生成

- application/vnd.kafka.v2+json

4.3.4.6. Tags

- 消费者

4.3.4.7. HTTP 请求示例

4.3.4.7.1. 请求正文

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

4.3.4.8. HTTP 响应示例

4.3.4.8.1. 响应 404

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

4.3.4.8.2. 响应 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. 描述

提交使用者偏移列表。要为消费者获取的所有记录提交偏移，请将请求正文留空。

4.3.5.2. 参数

类型	Name	描述	模式
路径	GroupId *必需*	消费者所属的消费者组的 ID。	字符串
路径	name *必需*	消费者的名称。	字符串
Body	body *可选*	提交到消费者偏移提交日志的消费者偏移列表。您可以指定一个或多个主题分区来提交偏移。	OffsetCommitSeekList

4.3.5.3. 响应

HTTP 代码	描述	模式
204	已成功提交。	无内容
404	未找到指定的消费者实例。	Error

4.3.5.4. 使用

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

4.3.5.5. 生成

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

4.3.5.6. Tags

- 消费者

4.3.5.7. HTTP 请求示例

4.3.5.7.1. 请求正文

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

4.3.5.8. HTTP 响应示例

4.3.5.8.1. 响应 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. 描述

配置订阅的消费者，以便在下一次从给定主题分区获取一组记录时从特定偏移中获取偏移。这会覆盖使用者的默认获取行为。您可以指定一个或多个主题分区。

4.3.6.2. 参数

类型	Name	描述	模式
路径	GroupId *必需*	消费者所属的消费者组的 ID。	字符串
路径	name *必需*	订阅消费者的名称。	字符串

类型	Name	描述	模式
Body	body *必需*	订阅消费者将在其中获取记录的分区偏移列表。	OffsetCommitSeekList

4.3.6.3. 响应

HTTP 代码	描述	模式
204	seek 已成功执行。	无内容
404	未找到指定的消费者实例，或者指定的消费者实例没有分配指定分区之一。	Error

4.3.6.4. 使用

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

4.3.6.5. 生成

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

4.3.6.6. Tags

- **消费者**
- **seek**

4.3.6.7. HTTP 请求示例

4.3.6.7.1. 请求正文

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

```
    "offset" : 42
  } ]
}
```

4.3.6.8. HTTP 响应示例

4.3.6.8.1. 响应 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. 描述

配置订阅的消费者，以查找一个或多个给定主题分区中的第一个偏移。

4.3.7.2. 参数

类型	Name	描述	模式
路径	GroupId *必需*	订阅消费者所属的消费者组的 ID。	字符串
路径	name *必需*	订阅消费者的名称。	字符串
Body	body *必需*	消费者订阅的主题分区列表。消费者将查找指定分区中的第一个偏移。	分区

4.3.7.3. 响应

HTTP 代码	描述	模式
204	请参阅进入成功执行的开始。	无内容
404	未找到指定的消费者实例，或者指定的消费者实例没有分配指定分区之一。	Error

4.3.7.4. 使用

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

4.3.7.5. 生成

- application/vnd.kafka.v2+json

4.3.7.6. Tags

- 消费者
- seek

4.3.7.7. HTTP 请求示例

4.3.7.7.1. 请求正文

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

4.3.7.8. HTTP 响应示例

4.3.7.8.1. 响应 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. 描述

配置订阅的消费者，以查找一个或多个给定主题分区末尾的偏移值（以及随后从中读取）。

4.3.8.2. 参数

类型	Name	描述	模式
路径	GroupId *必需*	订阅消费者所属的消费者组的 ID。	字符串
路径	name *必需*	订阅消费者的名称。	字符串
Body	body *可选*	消费者订阅的主题分区列表。消费者将查找指定分区中的最后一个偏移量。	分区

4.3.8.3. 响应

HTTP 代码	描述	模式
204	请参阅进入成功执行的最终操作。	无内容
404	未找到指定的消费者实例，或者指定的消费者实例没有分配指定分区之一。	Error

4.3.8.4. 使用

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

4.3.8.5. 生成

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

4.3.8.6. Tags

- **消费者**
- **seek**

4.3.8.7. HTTP 请求示例

4.3.8.7.1. 请求正文

```
{
  "partitions" : [ {
```

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

4.3.8.8. HTTP 响应示例

4.3.8.8.1. 响应 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. 描述

检索订阅的消费者的记录，包括消息值、主题和分区。此操作的请求必须使用响应中返回的基本 URL（包括主机和端口），从 POST 请求返回到用于创建此消费者的 /consumers/{groupid}。

4.3.9.2. 参数

类型	Name	描述	模式
路径	GroupId *必需*	订阅消费者所属的消费者组的 ID。	字符串
路径	name *必需*	要从中检索记录的消费者的名称。	字符串
查询	max_bytes *optional*	响应中可包含未编码的键和值的最大大小（以字节为单位）。否则，会返回带有代码 422 的错误响应。	整数
查询	**超时** *可选*	HTTP Bridge 在超时请求前花费的最大时间（以毫秒为单位）。	整数

4.3.9.3. 响应

HTTP 代码	描述	模式

HTTP 代码	描述	模式
200	轮询请求已成功执行。	ConsumerRecordList
404	未找到指定的消费者实例。	Error
406	消费者创建请求中使用的**格式**与此请求的 Accept 标头中的嵌入式格式不匹配，或者网桥从主题获取消息（非 JSON 编码）。	Error
422	响应超过消费者可接收的最大字节数	Error

4.3.9.4. 生成

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

4.3.9.5. Tags

- **消费者**

4.3.9.6. HTTP 响应示例

4.3.9.6.1. 响应 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. 响应 404

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

4.3.9.6.3. 响应 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. 响应 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. 描述

订阅消费者到一个或多个主题。您可以描述消费者将在列表（主题类型）或 topic_pattern 字段订阅的主题。每个调用都会替换订阅者的订阅。

4.3.10.2. 参数

类型	Name	描述	模式
路径	GroupId *必需*	订阅消费者所属的消费者组的 ID。	字符串
路径	name *必需*	订阅主题的消费者的名称。	字符串
Body	body *必需*	消费者要订阅的主题列表。	topics

4.3.10.3. 响应

HTTP 代码	描述	模式
204	消费者订阅成功。	无内容
404	未找到指定的消费者实例。	Error
409	主题、分区和模式订阅是互斥的。	Error
422	必须指定列表(**主题** 类型)或 **topic_pattern**。	Error

4.3.10.4. 使用

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

4.3.10.5. 生成

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

4.3.10.6. Tags

- **消费者**

4.3.10.7. HTTP 请求示例

4.3.10.7.1. 请求正文

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

4.3.10.8. HTTP 响应示例

4.3.10.8.1. 响应 404

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

4.3.10.8.2. 响应 409

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

4.3.10.8.3. 响应 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. 描述

检索消费者订阅的主题列表。

4.3.11.2. 参数

类型	Name	描述	模式
路径	GroupId *必需*	订阅消费者所属的消费者组的 ID。	字符串
路径	name *必需*	订阅消费者的名称。	字符串

4.3.11.3. 响应

HTTP 代码	描述	模式
200	订阅的主题和分区列表。	SubscribedTopicList
404	未找到指定的消费者实例。	Error

4.3.11.4. 生成

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

4.3.11.5. Tags

- **消费者**

4.3.11.6. HTTP 响应示例

4.3.11.6.1. 响应 200

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

4.3.11.6.2. 响应 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. 描述

取消订阅所有主题的消费者。

4.3.12.2. 参数

类型	Name	描述	模式
路径	GroupId *必需*	订阅消费者所属的消费者组的 ID。	字符串
路径	name *必需*	取消订阅主题的消费者的名称。	字符串

4.3.12.3. 响应

HTTP 代码	描述	模式
204	消费者已成功取消订阅。	无内容
404	未找到指定的消费者实例。	Error

4.3.12.4. Tags

- **消费者**

4.3.12.5. HTTP 响应示例

4.3.12.5.1. 响应 404

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

4.3.13. GET /healthy

4.3.13.1. 描述

检查网桥是否正在运行。这不一定意味着它已准备好接受请求。

4.3.13.2. 响应

HTTP 代码	描述	模式
204	网桥处于健康状态	无内容

HTTP 代码	描述	模式
500	网桥不是健康	无内容

4.3.14. GET /metrics

4.3.14.1. 描述

以 Prometheus 格式检索网桥指标。

4.3.14.2. 响应

HTTP 代码	描述	模式
200	Prometheus 格式的指标成功检索。	字符串

4.3.14.3. 生成

- text/plain

4.3.15. GET /openapi

4.3.15.1. 描述

以 JSON 格式检索 OpenAPI v2 规格。

4.3.15.2. 响应

HTTP 代码	描述	模式
204	JSON 格式的 OpenAPI v2 规格成功检索。	字符串

4.3.15.3. 生成

- application/json

4.3.16. GET /ready

4.3.16.1. 描述

检查网桥是否已就绪，并可以接受请求。

4.3.16.2. 响应

HTTP 代码	描述	模式
204	网桥已就绪	无内容
500	网桥未就绪	无内容

4.3.17. GET /topics

4.3.17.1. 描述

检索所有主题的列表。

4.3.17.2. 响应

HTTP 代码	描述	模式
200	主题列表。	< string > 数字

4.3.17.3. 生成

- application/vnd.kafka.v2+json

4.3.17.4. Tags

- topics

4.3.17.5. HTTP 响应示例

4.3.17.5.1. 响应 200

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

4.3.18. POST /topics/{topicname}

4.3.18.1. 描述

将一个或多个记录发送到给定主题，可以选择指定分区、密钥或两者。

4.3.18.2. 参数

类型	Name	描述	模式
路径	topicName *必需*	将记录发送到或从中检索元数据的主题名称。	字符串
查询	async *可选*	是否在发送记录后立即返回，而不是等待元数据。如果未指定，则不会返回偏移。默认为false。	布尔值
Body	body *必需*		ProducerRecordList

4.3.18.3. 响应

HTTP 代码	描述	模式
200	成功发送的记录。	OffsetRecordSentList
404	未找到指定的主题。	Error
422	记录列表无效。	Error

4.3.18.4. 使用

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

4.3.18.5. 生成

- application/vnd.kafka.v2+json

4.3.18.6. Tags

- 制作者
- topics

4.3.18.7. HTTP 请求示例

4.3.18.7.1. 请求正文

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

4.3.18.8. HTTP 响应示例

4.3.18.8.1. 响应 200

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

4.3.18.8.2. 响应 404

```
{
```

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

4.3.18.8.3. 响应 422

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

4.3.19. GET /topics/{topicname}

4.3.19.1. 描述

检索有关给定主题的元数据。

4.3.19.2. 参数

类型	Name	描述	模式
路径	topicName *必需*	将记录发送到或从中检索元数据的主题名称。	字符串

4.3.19.3. 响应

HTTP 代码	描述	模式
200	主题元数据	TopicMetadata

4.3.19.4. 生成

- application/vnd.kafka.v2+json

4.3.19.5. Tags

- topics

4.3.19.6. HTTP 响应示例

4.3.19.6.1. 响应 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
    } ]
  } ]
}
```

4.3.20. GET /topics/{topicname}/partitions

4.3.20.1. 描述

检索主题的分区列表。

4.3.20.2. 参数

类型	Name	描述	模式
路径	topicName *必需*	将记录发送到或从中检索元数据的主题名称。	字符串

4.3.20.3. 响应

HTTP 代码	描述	模式
200	分区列表	< PartitionMetadata > 数组
404	未找到指定的主题。	Error

4.3.20.4. 生成

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

4.3.20.5. Tags

- **topics**

4.3.20.6. HTTP 响应示例

4.3.20.6.1. 响应 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
  } ]
} ]
```

4.3.20.6.2. 响应 404

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

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

4.3.21.1. 描述

将一个或多个记录发送到给定的主题分区，选择性地指定密钥。

4.3.21.2. 参数

类型	Name	描述	模式
路径	partitionid *必需*	要从中发送记录的分区 ID，或从中检索元数据。	整数
路径	topicName *必需*	将记录发送到或从中检索元数据的主题名称。	字符串
查询	async *可选*	是否在发送记录后立即返回，而不是等待元数据。如果未指定，则不会返回偏移。默认为false。	布尔值
Body	body *必需*	要发送到给定主题分区的记录列表，包括值（必需）和键（可选）。	ProducerRecordToPartitionList

4.3.21.3. 响应

HTTP 代码	描述	模式
200	成功发送的记录。	OffsetRecordSentList
404	未找到指定的主题分区。	Error
422	记录无效。	Error

4.3.21.4. 使用

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

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

4.3.21.5. 生成

- application/vnd.kafka.v2+json

4.3.21.6. Tags

- 制作者
- topics

4.3.21.7. HTTP 请求示例

4.3.21.7.1. 请求正文

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

4.3.21.8. HTTP 响应示例

4.3.21.8.1. 响应 200

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

4.3.21.8.2. 响应 404

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

4.3.21.8.3. 响应 422

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

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

4.3.22.1. 描述

检索主题分区的分区元数据。

4.3.22.2. 参数

类型	Name	描述	模式
路径	partitionid *必需*	要从中发送记录的分区 ID，或从中检索元数据。	整数
路径	topicName *必需*	将记录发送到或从中检索元数据的主题名称。	字符串

4.3.22.3. 响应

HTTP 代码	描述	模式
200	分区元数据	PartitionMetadata
404	未找到指定的主题分区。	Error

4.3.22.4. 生成

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

4.3.22.5. Tags

- topics

4.3.22.6. HTTP 响应示例

4.3.22.6.1. 响应 200

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

4.3.22.6.2. 响应 404

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

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

4.3.23.1. 描述

检索主题分区的偏移摘要。

4.3.23.2. 参数

类型	Name	描述	模式
路径	partitionid *必需*	分区的 ID。	整数
路径	topicName *必需*	包含分区的主题名称。	字符串

4.3.23.3. 响应

HTTP 代码	描述	模式
200	主题分区的偏移摘要。	OffsetsSummary
404	未找到指定的主题分区。	Error

4.3.23.4. 生成

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

4.3.23.5. Tags

- **topics**

4.3.23.6. HTTP 响应示例

4.3.23.6.1. 响应 200

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

4.3.23.6.2. 响应 404

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

附录 A. 使用您的订阅

Apache Kafka 的流通过软件订阅提供。要管理您的订阅，请访问红帽客户门户中的帐户。

访问您的帐户

1. 转至 access.redhat.com。
2. 如果您还没有帐户，请创建一个帐户。
3. 登录到您的帐户。

激活订阅

1. 转至 access.redhat.com。
2. 导航到 My Subscriptions。
3. 导航到 激活订阅 并输入您的 16 位激活号。

下载 Zip 和 Tar 文件

要访问 zip 或 tar 文件，请使用客户门户网站查找下载的相关文件。如果您使用 RPM 软件包，则不需要这一步。

1. 打开浏览器并登录红帽客户门户网站 产品下载页面，网址为 access.redhat.com/downloads。
2. 在 INTEGRATION AND AUTOMATION 目录中找到 Apache Kafka for Apache Kafka 的流。
3. 选择 Apache Kafka 产品所需的流。此时会打开 Software Downloads 页面。

4. 单击组件的 **Download** 链接。

使用 DNF 安装软件包

要安装软件包以及所有软件包的依赖软件包，请使用：

```
dnf install <package_name>
```

要从本地目录中安装之前下载的软件包，请使用：

```
dnf install <path_to_download_package>
```

更新于 2024-04-30