Using the AMQ Streams Kafka Bridge

Use the AMQ Streams Kafka Bridge to connect with a Kafka cluster
Use the AMQ Streams Kafka Bridge to connect with a Kafka cluster
Abstract

The AMQ Streams Kafka Bridge provides a RESTful interface for HTTP-based clients to interact with a Kafka cluster.
# Table of Contents

**MAKING OPEN SOURCE MORE INCLUSIVE** ................................................................. 8

**CHAPTER 1. KAFKA BRIDGE OVERVIEW** ................................................................. 9
  1.1. RUNNING THE KAFKA BRIDGE ........................................................................ 9
    1.1.1. Running the Kafka Bridge on OpenShift ................................................. 10
  1.2. KAFKA BRIDGE INTERFACE ........................................................................ 10
    1.2.1. HTTP requests ......................................................................................... 10
  1.3. KAFKA BRIDGE OPENAPI SPECIFICATION ................................................. 11
  1.4. SECURING CONNECTIVITY TO THE KAFKA CLUSTER ............................... 11
  1.5. SECURING THE KAFKA BRIDGE HTTP INTERFACE .................................... 11
  1.6. REQUESTS TO THE KAFKA BRIDGE ............................................................ 12
    1.6.1. Content Type headers ............................................................................. 12
    1.6.2. Embedded data format .......................................................................... 12
    1.6.3. Message format ...................................................................................... 13
    1.6.4. Accept headers ....................................................................................... 14
  1.7. CORS ............................................................................................................ 14
    1.7.1. Simple request ....................................................................................... 14
    1.7.2. Preflighted request ................................................................................ 15
  1.8. CONFIGURING LOGGERS FOR THE KAFKA BRIDGE ................................. 16

**CHAPTER 2. KAFKA BRIDGE QUICKSTART** .............................................................. 18
  2.1. DOWNLOADING A KAFKA BRIDGE ARCHIVE .............................................. 18
  2.2. INSTALLING THE KAFKA BRIDGE ............................................................... 18
  2.3. PRODUCING MESSAGES TO TOPICS AND PARTITIONS ............................ 19
  2.4. CREATING A KAFKA BRIDGE CONSUMER .............................................. 25
  2.5. SUBSCRIBING A KAFKA BRIDGE CONSUMER TO TOPICS ...................... 26
  2.6. RETRIEVING THE LATEST MESSAGES FROM A KAFKA BRIDGE CONSUMER 27
  2.7. COMMITTING OFFSETS TO THE LOG ......................................................... 28
  2.8. SEEKING TO OFFSETS FOR A PARTITION .................................................. 28
  2.9. DELETING A KAFKA BRIDGE CONSUMER ................................................ 29

**CHAPTER 3. KAFKA BRIDGE CONFIGURATION** ...................................................... 31
  3.1. CONFIGURING KAFKA BRIDGE PROPERTIES .......................................... 31
  3.2. CONFIGURING DISTRIBUTED TRACING ..................................................... 32
    3.2.1. Specifying tracing systems with OpenTelemetry .................................... 34

**CHAPTER 4. AMQ STREAMS KAFKA BRIDGE API REFERENCE** ............................ 35
  4.1. OVERVIEW ................................................................................................. 35
    4.1.1. Version information ............................................................................... 35
    4.1.2. Tags ....................................................................................................... 35
    4.1.3. Consumes ............................................................................................. 35
    4.1.4. Produces ............................................................................................... 35
  4.2. DEFINITIONS .............................................................................................. 35
    4.2.1. AssignedTopicPartitions ....................................................................... 35
    4.2.2. BridgeInfo ............................................................................................ 35
    4.2.3. Consumer ............................................................................................. 35
    4.2.4. ConsumerRecord .................................................................................. 36
    4.2.5. ConsumerRecordList ........................................................................... 37
    4.2.6. CreatedConsumer ................................................................................ 37
    4.2.7. Error .................................................................................................... 37
    4.2.8. KafkaHeader ......................................................................................... 37
    4.2.9. KafkaHeaderList .................................................................................. 37
4.2.10. OffsetCommitSeek
4.2.11. OffsetCommitSeekList
4.2.12. OffsetRecordSent
4.2.13. OffsetRecordSentList
4.2.14. OffsetsSummary
4.2.15. Partition
4.2.16. PartitionMetadata
4.2.17. Partitions
4.2.18. ProducerRecord
4.2.19. ProducerRecordList
4.2.20. ProducerRecordToPartition
4.2.21. ProducerRecordToPartitionList
4.2.22. Replica
4.2.23. SubscribedTopicList
4.2.24. TopicMetadata
4.2.25. Topics

4.3. PATHS
4.3.1. GET /
   4.3.1.1. Description
   4.3.1.2. Responses
   4.3.1.3. Produces
   4.3.1.4. Example HTTP response
      4.3.1.4.1. Response 200
4.3.2. POST /consumers/{groupid}
   4.3.2.1. Description
   4.3.2.2. Parameters
   4.3.2.3. Responses
   4.3.2.4. Consumes
   4.3.2.5. Produces
   4.3.2.6. Tags
   4.3.2.7. Example HTTP request
      4.3.2.7.1. Request body
   4.3.2.8. Example HTTP response
      4.3.2.8.1. Response 200
      4.3.2.8.2. Response 409
      4.3.2.8.3. Response 422
4.3.3. DELETE /consumers/{groupid}/instances/{name}
   4.3.3.1. Description
   4.3.3.2. Parameters
   4.3.3.3. Responses
   4.3.3.4. Consumes
   4.3.3.5. Produces
   4.3.3.6. Tags
   4.3.3.7. Example HTTP response
      4.3.3.7.1. Response 404
4.3.4. POST /consumers/{groupid}/instances/{name}/assignments
   4.3.4.1. Description
   4.3.4.2. Parameters
   4.3.4.3. Responses
   4.3.4.4. Consumes
   4.3.4.5. Produces
   4.3.4.6. Tags
   4.3.4.7. Example HTTP request
4.3.9.5. Tags
4.3.9.6. Example HTTP response
  4.3.9.6.1. Response 200
  4.3.9.6.2. Response 404
  4.3.9.6.3. Response 406
  4.3.9.6.4. Response 422
4.3.10. POST /consumers/{groupid}/instances/{name}/subscription
  4.3.10.1. Description
  4.3.10.2. Parameters
  4.3.10.3. Responses
  4.3.10.4. Consumes
  4.3.10.5. Produces
  4.3.10.6. Tags
  4.3.10.7. Example HTTP request
    4.3.10.7.1. Request body
  4.3.10.8. Example HTTP response
    4.3.10.8.1. Response 404
    4.3.10.8.2. Response 409
    4.3.10.8.3. Response 422
4.3.11. GET /consumers/{groupid}/instances/{name}/subscription
  4.3.11.1. Description
  4.3.11.2. Parameters
  4.3.11.3. Responses
  4.3.11.4. Produces
  4.3.11.5. Tags
  4.3.11.6. Example HTTP response
    4.3.11.6.1. Response 200
    4.3.11.6.2. Response 404
4.3.12. DELETE /consumers/{groupid}/instances/{name}/subscription
  4.3.12.1. Description
  4.3.12.2. Parameters
  4.3.12.3. Responses
  4.3.12.4. Tags
  4.3.12.5. Example HTTP response
    4.3.12.5.1. Response 404
4.3.13. GET /healthy
  4.3.13.1. Description
  4.3.13.2. Responses
4.3.14. GET /metrics
  4.3.14.1. Description
  4.3.14.2. Responses
  4.3.14.3. Produces
4.3.15. GET /openapi
  4.3.15.1. Description
  4.3.15.2. Responses
  4.3.15.3. Produces
4.3.16. GET /ready
  4.3.16.1. Description
  4.3.16.2. Responses
4.3.17. GET /topics
  4.3.17.1. Description
  4.3.17.2. Responses
  4.3.17.3. Produces
4.3.17.4. Tags
4.3.17.5. Example HTTP response
  4.3.17.5.1. Response 200
4.3.18. POST /topics/{topicname}
  4.3.18.1. Description
  4.3.18.2. Parameters
  4.3.18.3. Responses
  4.3.18.4. Consumes
  4.3.18.5. Produces
  4.3.18.6. Tags
  4.3.18.7. Example HTTP request
    4.3.18.7.1. Request body
  4.3.18.8. Example HTTP response
    4.3.18.8.1. Response 200
    4.3.18.8.2. Response 404
    4.3.18.8.3. Response 422
4.3.19. GET /topics/{topicname}
  4.3.19.1. Description
  4.3.19.2. Parameters
  4.3.19.3. Responses
  4.3.19.4. Produces
  4.3.19.5. Tags
  4.3.19.6. Example HTTP response
    4.3.19.6.1. Response 200
4.3.20. GET /topics/{topicname}/partitions
  4.3.20.1. Description
  4.3.20.2. Parameters
  4.3.20.3. Responses
  4.3.20.4. Produces
  4.3.20.5. Tags
  4.3.20.6. Example HTTP response
    4.3.20.6.1. Response 200
    4.3.20.6.2. Response 404
4.3.21. POST /topics/{topicname}/partitions/{partitionid}
  4.3.21.1. Description
  4.3.21.2. Parameters
  4.3.21.3. Responses
  4.3.21.4. Consumes
  4.3.21.5. Produces
  4.3.21.6. Tags
  4.3.21.7. Example HTTP request
    4.3.21.7.1. Request body
  4.3.21.8. Example HTTP response
    4.3.21.8.1. Response 200
    4.3.21.8.2. Response 404
    4.3.21.8.3. Response 422
4.3.22. GET /topics/{topicname}/partitions/{partitionid}
  4.3.22.1. Description
  4.3.22.2. Parameters
  4.3.22.3. Responses
  4.3.22.4. Produces
  4.3.22.5. Tags
  4.3.22.6. Example HTTP response
4.3.22.6.1. Response 200
4.3.22.6.2. Response 404

4.3.23. GET /topics/{topicname}/partitions/{partitionid}/offsets
4.3.23.1. Description
4.3.23.2. Parameters
4.3.23.3. Responses
4.3.23.4. Produces
4.3.23.5. Tags
4.3.23.6. Example HTTP response
4.3.23.6.1. Response 200
4.3.23.6.2. Response 404

APPENDIX A. USING YOUR SUBSCRIPTION
Accessing Your Account
Activating a Subscription
Downloading Zip and Tar Files
Installing packages with DNF
MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see our CTO Chris Wright’s message.
Use the AMQ Streams Kafka Bridge to make HTTP requests to a Kafka cluster.

You can use the Kafka Bridge to integrate HTTP client applications with your Kafka cluster.

HTTP client integration

1.1. RUNNING THE KAFKA BRIDGE
Install the AMQ Streams Kafka Bridge to run in the same environment as your Kafka cluster.

You can download and add the Kafka Bridge installation artifacts to your host machine. To try out the Kafka Bridge in your local environment, see the Kafka Bridge quickstart.

It’s important to note that each instance of the Kafka Bridge maintains its own set of in-memory consumers (and subscriptions) that connect to the Kafka Brokers on behalf of the HTTP clients. This means that each HTTP client must maintain affinity to the same Kafka Bridge instance in order to access any subscriptions that are created. Additionally, when an instance of the Kafka Bridge restarts, the in-memory consumers and subscriptions are lost. It is the responsibility of the HTTP client to recreate any consumers and subscriptions if the Kafka Bridge restarts.

1.1. Running the Kafka Bridge on OpenShift

If you deployed AMQ Streams on OpenShift, you can use the AMQ Streams Cluster Operator to deploy the Kafka Bridge to the OpenShift cluster. Configure and deploy the Kafka Bridge as a KafkaBridge resource. You’ll need a running Kafka cluster that was deployed by the Cluster Operator in an OpenShift namespace. You can configure your deployment to access the Kafka Bridge outside the OpenShift cluster.

HTTP clients must maintain affinity to the same instance of the Kafka Bridge to access any consumers or subscriptions that they create. Hence, running multiple replicas of the Kafka Bridge per OpenShift Deployment is not recommended. If the Kafka Bridge pod restarts (for instance, due to OpenShift relocating the workload to another node), the HTTP client must recreate any consumers or subscriptions.

For information on deploying and configuring the Kafka Bridge as a KafkaBridge resource, see the AMQ Streams documentation.

1.2. KAFKA BRIDGE INTERFACE

The Kafka Bridge provides a RESTful interface that allows HTTP-based clients to interact with a Kafka cluster. It offers the advantages of a web API connection to AMQ Streams, without the need for client applications to interpret the Kafka protocol.

The API has two main resources – consumers and topics – that are exposed and made accessible through endpoints to interact with consumers and producers in your Kafka cluster. The resources relate only to the Kafka Bridge, not the consumers and producers connected directly to Kafka.

1.2.1. HTTP requests

The Kafka Bridge supports HTTP requests to a Kafka cluster, with methods to:

- Send messages to a topic.
- Retrieve messages from topics.
- Retrieve a list of partitions for a topic.
- Create and delete consumers.
- Subscribe consumers to topics, so that they start receiving messages from those topics.
- Retrieve a list of topics that a consumer is subscribed to.
- Unsubscribe consumers from topics.
Assign partitions to consumers.

Commit a list of consumer offsets.

Seek on a partition, so that a consumer starts receiving messages from the first or last offset position, or a given offset position.

The methods provide JSON responses and HTTP response code error handling. Messages can be sent in JSON or binary formats.

Clients can produce and consume messages without the requirement to use the native Kafka protocol.

Additional resources

- AMQ Streams Kafka Bridge API reference

1.3. KAFKA BRIDGE OPENAPI SPECIFICATION

Kafka Bridge APIs use the OpenAPI Specification (OAS). OAS provides a standard framework for describing and implementing HTTP APIs.

The Kafka Bridge OpenAPI specification is in JSON format. You can find the OpenAPI JSON files in the src/main/resources/ folder of the Kafka Bridge source download files. The download files are available from the Customer Portal.

You can also use the GET /openapi method to retrieve the OpenAPI v2 specification in JSON format.

Additional resources

- OpenAPI initiative

1.4. SECURING CONNECTIVITY TO THE KAFKA CLUSTER

You can configure the following between the Kafka Bridge and your Kafka cluster:

- TLS or SASL-based authentication
- A TLS-encrypted connection

You configure the Kafka Bridge for authentication through its properties file.

You can also use ACLs in Kafka brokers to restrict the topics that can be consumed and produced using the Kafka Bridge.

NOTE

Use the KafkaBridge resource to configure authentication when you are running the Kafka Bridge on OpenShift.

1.5. SECURING THE KAFKA BRIDGE HTTP INTERFACE

Authentication and encryption between HTTP clients and the Kafka Bridge is not supported directly by the Kafka Bridge. Requests sent from clients to the Kafka Bridge are sent without authentication or encryption. Requests must use HTTP rather than HTTPS.
You can combine the Kafka Bridge with the following tools to secure it:

- Network policies and firewalls that define which pods can access the Kafka Bridge
- Reverse proxies (for example, OAuth 2.0)
- API gateways

**1.6. REQUESTS TO THE KAFKA BRIDGE**

Specify data formats and HTTP headers to ensure valid requests are submitted to the Kafka Bridge.

**1.6.1. Content Type headers**

API request and response bodies are always encoded as JSON.

- When performing consumer operations, **POST** requests must provide the following **Content-Type** header if there is a non-empty body:

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

- When performing producer operations, **POST** requests must provide **Content-Type** headers specifying the *embedded data format* of the messages produced. This can be either **json** or **binary**.

<table>
<thead>
<tr>
<th>Embedded data format</th>
<th>Content-Type header</th>
</tr>
</thead>
<tbody>
<tr>
<td>JSON</td>
<td><strong>Content-Type: application/vnd.kafka.json.v2+json</strong></td>
</tr>
<tr>
<td>Binary</td>
<td><strong>Content-Type: application/vnd.kafka.binary.v2+json</strong></td>
</tr>
</tbody>
</table>

The embedded data format is set per consumer, as described in the next section.

The **Content-Type** must **not** be set if the **POST** request has an empty body. An empty body can be used to create a consumer with the default values.

**1.6.2. Embedded data format**

The embedded data format is the format of the Kafka messages that are transmitted, over HTTP, from a producer to a consumer using the Kafka Bridge. Two embedded data formats are supported: JSON and binary.

When creating a consumer using the `/consumers/groupid` endpoint, the **POST** request body must specify an embedded data format of either JSON or binary. This is specified in the **format** field, for example:

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

1 A binary embedded data format.
The embedded data format specified when creating a consumer must match the data format of the Kafka messages it will consume.

If you choose to specify a binary embedded data format, subsequent producer requests must provide the binary data in the request body as Base64-encoded strings. For example, when sending messages using the /topics/topicname endpoint, records.value must be encoded in Base64:

```json
{
  "records": [
    {
      "key": "my-key",
      "value": "ZWR3YXJkdGhldGhyZWWsZWNnZWRjYXQ="
    },
  ]
}
```

Producer requests must also provide a Content-Type header that corresponds to the embedded data format, for example, Content-Type: application/vnd.kafka.binary.v2+json.

### 1.6.3. Message format

When sending messages using the /topics endpoint, you enter the message payload in the request body, in the records parameter.

The records parameter can contain any of these optional fields:

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

**Example POST request to /topics**

```bash
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": "QXBhY2hhIEthZmthIGlzIHRoZSBib21iQ==" ①
c          }
        ]
      }
    ]
  }'
```
The header value in binary format and encoded as Base64.

### 1.6.4. Accept headers

After creating a consumer, all subsequent GET requests must provide an `Accept` header in the following format:

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

The `EMBEDDED-DATA-FORMAT` is either `json` or `binary`.

For example, when retrieving records for a subscribed consumer using an embedded data format of JSON, include this Accept header:

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

### 1.7. CORS

In general, it is not possible for an HTTP client to issue requests across different domains.

For example, suppose the Kafka Bridge you deployed alongside a Kafka cluster is accessible using the `http://my-bridge.io` domain. HTTP clients can use the URL to interact with the Kafka Bridge and exchange messages through the Kafka cluster. However, your client is running as a web application in the `http://my-web-application.io` domain. The client (source) domain is different from the Kafka Bridge (target) domain. Because of same-origin policy restrictions, requests from the client fail. You can avoid this situation by using Cross-Origin Resource Sharing (CORS).

CORS allows for *simple* and *preflighted* requests between origin sources on different domains.

Simple requests are suitable for standard requests using `GET`, `HEAD`, `POST` methods.

A preflighted request sends a `HTTP OPTIONS` request as an initial check that the actual request is safe to send. On confirmation, the actual request is sent. Preflight requests are suitable for methods that require greater safeguards, such as `PUT` and `DELETE`, and use non-standard headers.

All requests require an `origins` value in their header, which is the source of the HTTP request.

CORS allows you to specify allowed methods and originating URLs for accessing the Kafka cluster in your Kafka Bridge HTTP configuration.

#### Example CORS configuration for Kafka Bridge

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

### 1.7.1. Simple request

For example, this simple request header specifies the origin as `http://my-web-application.io`.

```
Origin: http://my-web-application.io
```
The header information is added to the request to consume records.

```
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'
```

In the response from the Kafka Bridge, an **Access-Control-Allow-Origin** header is returned. It contains the list of domains from where HTTP requests can be issued to the bridge.

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

1. Returning an asterisk (*) shows the resource can be accessed by any domain.

### 1.7.2. Preflighted request

An initial preflight request is sent to Kafka Bridge using an **OPTIONS** method. The **HTTP OPTIONS** request sends header information to check that Kafka Bridge will allow the actual request.

Here the preflight request checks that a **POST** request is valid from **http://my-web-application.io**.

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

1. Kafka Bridge is alerted that the actual request is a **POST** request.

2. The actual request will be sent with a **Content-Type** header.

**OPTIONS** is added to the header information of the preflight request.

```
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 responds to the initial request to confirm that the request will be accepted. The response header returns allowed origins, methods and headers.

```
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
```

If the origin or method is rejected, an error message is returned.

The actual request does not require **Access-Control-Request-Method** header, as it was confirmed in the preflight request, but it does require the origin header.
1.8. CONFIGURING LOGGERS FOR THE KAFKA BRIDGE

You can set a different log level for each operation that is defined by the Kafka Bridge OpenAPI specification.

Each operation has a corresponding API endpoint through which the bridge receives requests from HTTP clients. You can change the log level on each endpoint to produce more or less fine-grained logging information about the incoming and outgoing HTTP requests.

Loggers are defined in the log4j2.properties file, which has the following default configuration for healthy and ready endpoints:

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

The log level of all other operations is set to INFO by default. Loggers are formatted as follows:

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

Where `<operation_id>` is the identifier of the specific operation.

List of operations defined by the OpenAPI specification

- createConsumer
- deleteConsumer
- subscribe
- unsubscribe
- poll
- assign
- commit
- send
• sendToPartition
• seekToBeginning
• seekToEnd
• seek
• healthy
• ready
• openapi

Where $<LOG\_LEVEL>$ is the logging level as defined by log4j2 (i.e. INFO, DEBUG, ...).
CHAPTER 2. KAFKA BRIDGE QUICKSTART

Use this quickstart to try out the AMQ Streams Kafka Bridge in your local development environment.

You will learn how to do the following:

- Produce messages to topics and partitions in your Kafka cluster
- Create a Kafka Bridge consumer
- Perform basic consumer operations, such as subscribing the consumer to topics and retrieving the messages that you produced

In this quickstart, HTTP requests are formatted as curl commands that you can copy and paste to your terminal.

Ensure you have the prerequisites and then follow the tasks in the order provided in this chapter.

In this quickstart, you will produce and consume messages in JSON format.

Prerequisites for the quickstart

- A Kafka cluster is running on the host machine.

2.1. DOWNLOADING A KAFKA BRIDGE ARCHIVE

A zipped distribution of the AMQ Streams Kafka Bridge is available for download.

Procedure

- Download the latest version of the AMQ Streams Kafka Bridge archive from the Customer Portal.

2.2. INSTALLING THE KAFKA BRIDGE

Use the script provided with the Kafka Bridge archive to install the Kafka Bridge. The application.properties file provided with the installation archive provides default configuration settings.

The following default property values configure the Kafka Bridge to listen for requests on port 8080.

Default configuration properties

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

Prerequisites

- The Kafka Bridge installation archive is downloaded

Procedure

1. If you have not already done so, unzip the Kafka Bridge installation archive to any directory.
2. Run the Kafka Bridge script using the configuration properties as a parameter:
   For example:
   ```
   /bin/kafka_bridge_run.sh --config-file=<path>/application.properties
   ```

3. Check to see that the installation was successful in the log.
   ```
   HTTP-Kafka Bridge started and listening on port 8080
   HTTP-Kafka Bridge bootstrap servers localhost:9092
   ```

What to do next

- Produce messages to topics and partitions.

### 2.3. PRODUCING MESSAGES TO TOPICS AND PARTITIONS

Use the Kafka Bridge to produce messages to a Kafka topic in JSON format by using the topics endpoint.

You can produce messages to topics in JSON format by using the `topics` endpoint. You can specify destination partitions for messages in the request body. The `partitions` endpoint provides an alternative method for specifying a single destination partition for all messages as a path parameter.

In this procedure, messages are produced to a topic called `bridge-quickstart-topic`.

**Prerequisites**

- The Kafka cluster has a topic with three partitions.
  You can use the `kafka-topics.sh` utility to create topics.

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

  **Verifying the topic was created**
  ```
  bin/kafka-topics.sh --bootstrap-server localhost:9092 --describe --topic bridge-quickstart-
  topic
  ```

  **NOTE**
  If you deployed AMQ Streams on OpenShift, you can create a topic using the `KafkaTopic`
  custom resource.

**Procedure**

1. Using the Kafka Bridge, produce three messages to the topic you created:

   ```
   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 is sent to a partition based on the hash of the key.
- sales-lead-0002 is sent directly to partition 2.
- sales-lead-0003 is sent to a partition in the bridge-quickstart-topic topic using a round-robin method.

2. If the request is successful, the Kafka Bridge returns an offsets array, along with a 200 code and a content-type header of application/vnd.kafka.v2+json. For each message, the offsets array describes:
   - The partition that the message was sent to
   - The current message offset of the partition

Example response

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

Additional topic requests

Make other curl requests to find information on topics and partitions.

List topics

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

Example response


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

Get topic configuration and partition details

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

Example response

{
  "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": "864000000",
    "segment.ms": "604800000",
    "message.timestamp.difference.max.ms": "9223372036854775807",
    "segment.index.bytes": "10485760"
  },
  "partitions": [
  
  
  ]
}
{  
  "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  
    }  
  ]  
}
List the partitions of a specific topic

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

Example response

```
[
  {
    "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
      }
    ]
  }
]
```
List the details of a specific topic partition

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

Example response

```
{
  "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
    }
  ]
}
```

List the offsets of a specific topic partition
After producing messages to topics and partitions, create a Kafka Bridge consumer.

Additional resources

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

2.4. CREATING A KAFKA BRIDGE CONSUMER

Before you can perform any consumer operations in the Kafka cluster, you must first create a consumer by using the consumers endpoint. The consumer is referred to as a Kafka Bridge consumer.

Procedure

1. Create a Kafka Bridge consumer in a new consumer group named bridge-quickstart-consumer-group:

```
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
  }'
```

- The consumer is named bridge-quickstart-consumer and the embedded data format is set as json.

- Some basic configuration settings are defined.

- The consumer will not commit offsets to the log automatically because the enable.auto.commit setting is false. You will commit the offsets manually later in this quickstart.

  If the request is successful, the Kafka Bridge returns the consumer ID (instance_id) and base URL (base_uri) in the response body, along with a 200 code.

  Example response

```json
{
  "beginning_offset": 0,
  "end_offset": 1
}
```
2. Copy the base URL (base_uri) to use in the other consumer operations in this quickstart.

What to do next
Now that you have created a Kafka Bridge consumer, you can subscribe it to topics.

Additional resources
- POST /consumers/{groupid}

2.5. SUBSCRIBING A KAFKA BRIDGE CONSUMER TO TOPICS
After you have created a Kafka Bridge consumer, subscribe it to one or more topics by using the subscription endpoint. When subscribed, the consumer starts receiving all messages that are produced to the topic.

Procedure
- Subscribe the consumer to the bridge-quickstart-topic topic that you created earlier, in Producing messages to topics and partitions:

```
-H 'content-type: application/vnd.kafka.v2+json'
-d '{
  "topics": ["bridge-quickstart-topic"]
}
```

The topics array can contain a single topic (as shown here) or multiple topics. If you want to subscribe the consumer to multiple topics that match a regular expression, you can use the topic_pattern string instead of the topics array.

If the request is successful, the Kafka Bridge returns a 204 (No Content) code only.

When using an Apache Kafka client, the HTTP subscribe operation adds topics to the local consumer’s subscriptions. Joining a consumer group and obtaining partition assignments occur after running multiple HTTP poll operations, starting the partition rebalance and join-group process. It’s important to note that the initial HTTP poll operations may not return any records.

What to do next
After subscribing a Kafka Bridge consumer to topics, you can retrieve messages from the consumer.

Additional resources
- POST /consumers/{groupid}/instances/{name}/subscription
2.6. RETRIEVING THE LATEST MESSAGES FROM A KAFKA BRIDGE CONSUMER

Retrieve the latest messages from the Kafka Bridge consumer by requesting data from the records endpoint. In production, HTTP clients can call this endpoint repeatedly (in a loop).

Procedure

1. Produce additional messages to the Kafka Bridge consumer, as described in Producing messages to topics and partitions.

2. Submit a GET request to the records endpoint:

   ```shell
   -H 'accept: application/vnd.kafka.json.v2+json'
   ```

   After creating and subscribing to a Kafka Bridge consumer, a first GET request will return an empty response because the poll operation starts a rebalancing process to assign partitions.

3. Repeat step two to retrieve messages from the Kafka Bridge consumer.
   The Kafka Bridge returns an array of messages — describing the topic name, key, value, partition, and offset — in the response body, along with a 200 code. Messages are retrieved from the latest offset by default.

   ```json
   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 }
   ],
   #...
   ```

   **NOTE**

   If an empty response is returned, produce more records to the consumer as described in Producing messages to topics and partitions, and then try retrieving messages again.

What to do next

After retrieving messages from a Kafka Bridge consumer, try committing offsets to the log.
2.7. COMMITTING OFFSETS TO THE LOG

Use the offsets endpoint to manually commit offsets to the log for all messages received by the Kafka Bridge consumer. This is required because the Kafka Bridge consumer that you created earlier, in Creating a Kafka Bridge consumer, was configured with the enable.auto.commit setting as false.

Procedure

- Commit offsets to the log for the bridge-quickstart-consumer:


  Because no request body is submitted, offsets are committed for all the records that have been received by the consumer. Alternatively, the request body can contain an array (OffsetCommitSeekList) that specifies the topics and partitions that you want to commit offsets for.

  If the request is successful, the Kafka Bridge returns a 204 code only.

What to do next

After committing offsets to the log, try out the endpoints for seeking to offsets.

Additional resources

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

2.8. SEEKING TO OFFSETS FOR A PARTITION

Use the positions endpoints to configure the Kafka Bridge consumer to retrieve messages for a partition from a specific offset, and then from the latest offset. This is referred to in Apache Kafka as a seek operation.

Procedure

1. Seek to a specific offset for partition 0 of the quickstart-bridge-topic topic:

  -H 'content-type: application/vnd.kafka.v2+json'
  -d '{
      "offsets": [
        {
          "topic": "bridge-quickstart-topic",
          "partition": 0,
          "offset": 2
        }
      ]
  }'
If the request is successful, the Kafka Bridge returns a 204 code only.

2. Submit a GET request to the records endpoint:

   -H 'accept: application/vnd.kafka.json.v2+json'

   The Kafka Bridge returns messages from the offset that you seeked to.

3. Restore the default message retrieval behavior by seeking to the last offset for the same partition. This time, use the positions/end endpoint.

   -H 'content-type: application/vnd.kafka.v2+json'
   -d '{
     "partitions": [
       {
         "topic": "bridge-quickstart-topic",
         "partition": 0
       }
     ]
   }'

   If the request is successful, the Kafka Bridge returns another 204 code.

   **NOTE**
   You can also use the positions/beginning endpoint to seek to the first offset for one or more partitions.

**What to do next**

In this quickstart, you have used the AMQ Streams Kafka Bridge to perform several common operations on a Kafka cluster. You can now delete the Kafka Bridge consumer that you created earlier.

**Additional resources**

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

**2.9. DELETING A KAFKA BRIDGE CONSUMER**

Delete the Kafka Bridge consumer that you used throughout this quickstart.

**Procedure**

- Delete the Kafka Bridge consumer by sending a DELETE request to the instances endpoint.

If the request is successful, the Kafka Bridge returns a 204 code.

Additional resources

- DELETE /consumers/{groupid}/instances/{name}
CHAPTER 3. KAFKA BRIDGE CONFIGURATION

Configure a deployment of the Kafka Bridge using configuration properties. Configure Kafka and specify the HTTP connection details needed to be able to interact with Kafka. You can also use configuration properties to enable and use distributed tracing with the Kafka Bridge. Distributed tracing allows you to track the progress of transactions between applications in a distributed system.

NOTE

Use the KafkaBridge resource to configure properties when you are running the Kafka Bridge on OpenShift.

3.1. CONFIGURING KAFKA BRIDGE PROPERTIES

This procedure describes how to configure the Kafka and HTTP connection properties used by the Kafka Bridge.

You configure the Kafka Bridge, as any other Kafka client, using appropriate prefixes for Kafka-related properties.

- **kafka.** for general configuration that applies to producers and consumers, such as server connection and security.
- **kafka.consumer.** for consumer-specific configuration passed only to the consumer.
- **kafka.producer.** for producer-specific configuration passed only to the producer.

As well as enabling HTTP access to a Kafka cluster, HTTP properties provide the capability to enable and define access control for the Kafka Bridge through Cross-Origin Resource Sharing (CORS). CORS is a HTTP mechanism that allows browser access to selected resources from more than one origin. To configure CORS, you define a list of allowed resource origins and HTTP methods to access them. Additional HTTP headers in requests describe the CORS origins that are permitted access to the Kafka cluster.

Prerequisites

- The Kafka Bridge installation archive is downloaded

Procedure

1. Edit the application.properties file provided with the Kafka Bridge installation archive. Use the properties file to specify Kafka and HTTP-related properties.
   
   a. Configure standard Kafka-related properties, including properties specific to the Kafka consumers and producers.
      Use:

      - **kafka.bootstrap.servers** to define the host/port connections to the Kafka cluster
      - **kafka.producer.acks** to provide acknowledgments to the HTTP client
      - **kafka.consumer.auto.offset.reset** to determine how to manage reset of the offset in Kafka

For more information on configuration of Kafka properties, see the [Apache Kafka website](https://kafka.apache.org/documentation/)
b. Configure HTTP-related properties to enable HTTP access to the Kafka cluster. For example:

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

1. The default HTTP configuration for the Kafka Bridge to listen on port 8080.
2. Set to `true` to enable CORS.
3. Comma-separated list of allowed CORS origins. You can use a URL or a Java regular expression.
4. Comma-separated list of allowed HTTP methods for CORS.

2. Save the configuration file.

### 3.2. Configuring Distributed Tracing

Enable distributed tracing to trace messages consumed and produced by the Kafka Bridge, and HTTP requests from client applications.

Properties to enable tracing are present in the `application.properties` file. To enable distributed tracing, do the following:

- Set the `bridge.tracing` property value to enable the tracing you want to use. Possible values are `jaeger` and `opentelemetry`.
- Set environment variables for tracing.

With the default configuration, OpenTelemetry tracing uses OTLP as the exporter protocol. By configuring the OTLP endpoint, you can still use a Jaeger backend instance to get traces.

**NOTE**

Jaeger has supported the OTLP protocol since version 1.35. Older Jaeger versions cannot get traces using the OTLP protocol.

OpenTelemetry and OpenTracing are API specifications for collecting tracing data as spans of metrics data. Spans represent a specific operation. A trace is a collection of one or more spans.

Traces are generated when the Kafka Bridge does the following:

- Sends messages from Kafka to consumer HTTP clients
- Receives messages from producer HTTP clients to send to Kafka

Jaeger implements the required APIs and presents visualizations of the trace data in its user interface for analysis.
To have end-to-end tracing, you must configure tracing in your HTTP clients.

**CAUTION**

The OpenTracing project is now archived, so AMQ Streams has deprecated support for OpenTracing. If possible, we will maintain the support for `bridge.tracing=jaeger` tracing until June 2023 and remove it afterwards. Please migrate to OpenTelemetry as soon as possible.

**Prerequisites**

- The Kafka Bridge installation archive is downloaded.

**Procedure**

1. Edit the application.properties file provided with the Kafka Bridge installation archive. Use the `bridge.tracing` property to enable the tracing you want to use.

   **Example configuration to enable OpenTelemetry**

   ```properties
   #bridge.tracing=jaeger
   bridge.tracing=opentelemetry
   
   1 The property for enabling OpenTracing (deprecated). Here left commented.
   2 The property for enabling OpenTelemetry is uncommented.
   
   With tracing enabled, you initialize tracing when you run the Kafka Bridge script.
   
2. Save the configuration file.

3. Set the environment variables for tracing.

   **Environment variables for OpenTelemetry**

   ```properties
   OTEL_SERVICE_NAME=my-tracing-service
   OTEL_EXPORTER_OTLP_ENDPOINT=http://localhost:4317
   
   1 The name of the OpenTelemetry tracer service.
   2 The gRPC-based OTLP endpoint that listens for spans on port 4317.
   
   **Environment variables for OpenTracing**

   ```properties
   JAEGGER_SERVICE_NAME=my-jaeger-service
   JAEGGER_AGENT_HOST=localhost
   JAEGGER_AGENT_PORT=6831
   
   1 The name of the OpenTracing Jaeger tracer service.
   2 The hostname for communicating with the Jaeger agent that listens for spans.
3. The port for communicating with the Jaeger agent. Port 6831 is exposed by the Jaeger agent.

4. Run the Kafka Bridge script with the property enabled for tracing:

```
Running the Kafka Bridge with OpenTelemetry enabled

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

The internal consumers and producers of the Kafka Bridge are now enabled for tracing.

### 3.2.1. Specifying tracing systems with OpenTelemetry

Instead of the default OTLP tracing system, you can specify other tracing systems that are supported by OpenTelemetry.

If you want to use another tracing system with OpenTelemetry, do the following:

1. Add the library of the tracing system to the Kafka classpath.

2. Add the name of the tracing system as an additional exporter environment variable.

**Additional environment variable when not using OTLP**

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

1. The name of the tracing system. In this example, Zipkin is specified.

2. The endpoint of the specific selected exporter that listens for spans. In this example, a Zipkin endpoint is specified.

**Additional resources**

- OpenTelemetry exporter values
4.1. OVERVIEW

The AMQ Streams Kafka Bridge provides a REST API for integrating HTTP based client applications with a Kafka cluster. You can use the API to create and manage consumers and send and receive records over HTTP rather than the native Kafka protocol.

4.1.1. Version information

Version : 0.1.0

4.1.2. Tags

- Consumers : Consumer operations to create consumers in your Kafka cluster and perform common actions, such as subscribing to topics, retrieving processed records, and committing offsets.
- Producer : Producer operations to send records to a specified topic or topic partition.
- Seek : Seek operations that enable a consumer to begin receiving messages from a given offset position.
- Topics : Topic operations to send messages to a specified topic or topic partition, optionally including message keys in requests. You can also retrieve topics and topic metadata.

4.1.3. Consumes

- application/json

4.1.4. Produces

- application/json

4.2. DEFINITIONS

4.2.1. AssignedTopicPartitions

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

4.2.2. BridgelInfo

Information about Kafka Bridge instance.

<table>
<thead>
<tr>
<th>Name</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>bridge_version</td>
<td>string</td>
</tr>
<tr>
<td>optional</td>
<td></td>
</tr>
</tbody>
</table>

4.2.3. Consumer
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>auto.offset.reset</td>
<td>Resets the offset position for the consumer. If set to <strong>latest</strong> (default), messages are read from the latest offset. If set to <strong>earliest</strong>, messages are read from the first offset.</td>
<td>string</td>
</tr>
<tr>
<td>consumer.request.timeout.ms</td>
<td>Sets the maximum amount of time, in milliseconds, for the consumer to wait for messages for a request. If the timeout period is reached without a response, an error is returned. Default is <strong>30000</strong> (30 seconds).</td>
<td>integer</td>
</tr>
<tr>
<td>enable.auto.commit</td>
<td>If set to <strong>true</strong> (default), message offsets are committed automatically for the consumer. If set to <strong>false</strong>, message offsets must be committed manually.</td>
<td>boolean</td>
</tr>
<tr>
<td>fetch.min.bytes</td>
<td>Sets the minimum amount of data, in bytes, for the consumer to receive. The broker waits until the data to send exceeds this amount. Default is <strong>1</strong> byte.</td>
<td>integer</td>
</tr>
<tr>
<td>format</td>
<td>The allowable message format for the consumer, which can be <strong>binary</strong> (default) or <strong>json</strong>. The messages are converted into a JSON format.</td>
<td>string</td>
</tr>
<tr>
<td>isolation.level</td>
<td>If set to <strong>read_uncommitted</strong> (default), all transaction records are retrieved, independent of any transaction outcome. If set to <strong>read_committed</strong>, the records from committed transactions are retrieved.</td>
<td>string</td>
</tr>
<tr>
<td>name</td>
<td>The unique name for the consumer instance. The name is unique within the scope of the consumer group. The name is used in URLs. If a name is not specified, a randomly generated name is assigned.</td>
<td>string</td>
</tr>
</tbody>
</table>

### 4.2.4. ConsumerRecord

<table>
<thead>
<tr>
<th>Name</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>headers</td>
<td><strong>KafkaHeaderList</strong></td>
</tr>
<tr>
<td>offset</td>
<td>integer (int64)</td>
</tr>
<tr>
<td>partition</td>
<td>integer (int32)</td>
</tr>
<tr>
<td>topic</td>
<td>string</td>
</tr>
</tbody>
</table>
4.2.5. ConsumerRecordList

Type: `<ConsumerRecord> array`

4.2.6. CreatedConsumer

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>base_uri</td>
<td>Base URI used to construct URIs for subsequent requests against this consumer instance.</td>
<td>string</td>
</tr>
<tr>
<td>instance_id</td>
<td>Unique ID for the consumer instance in the group.</td>
<td>string</td>
</tr>
</tbody>
</table>

4.2.7. Error

<table>
<thead>
<tr>
<th>Name</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>error_code</td>
<td>integer (int32)</td>
</tr>
<tr>
<td>message</td>
<td>string</td>
</tr>
</tbody>
</table>

4.2.8. KafkaHeader

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>key</td>
<td>required</td>
<td>string</td>
</tr>
<tr>
<td>value</td>
<td>required</td>
<td>string (byte)</td>
</tr>
</tbody>
</table>

`key` and `value` contain the header value in binary format, base64-encoded. The header value must be a string of bytes matching the following pattern:

```
^\([^A-Za-z0-9+/>\]{4}\)[^A-Za-z0-9+/>\]{2}=|[^A-Za-z0-9+/>\]{3}=\)?$`
```

4.2.9. KafkaHeaderList

Type: `<KafkaHeader> array`

4.2.10. OffsetCommitSeek

<table>
<thead>
<tr>
<th>Name</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>offset</td>
<td>integer (int64)</td>
</tr>
</tbody>
</table>

CHAPTER 4. AMQ STREAMS KAFKA BRIDGE API REFERENCE
<table>
<thead>
<tr>
<th>Name</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>partition</td>
<td>integer (int32)</td>
</tr>
<tr>
<td></td>
<td>required</td>
</tr>
<tr>
<td>topic</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td>required</td>
</tr>
</tbody>
</table>

### 4.2.11. OffsetCommitSeekList

<table>
<thead>
<tr>
<th>Name</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>offsets</td>
<td>&lt; OffsetCommitSeek &gt; array</td>
</tr>
<tr>
<td></td>
<td>optional</td>
</tr>
</tbody>
</table>

### 4.2.12. OffsetRecordSent

<table>
<thead>
<tr>
<th>Name</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>offset</td>
<td>integer (int64)</td>
</tr>
<tr>
<td></td>
<td>optional</td>
</tr>
<tr>
<td>partition</td>
<td>integer (int32)</td>
</tr>
<tr>
<td></td>
<td>optional</td>
</tr>
</tbody>
</table>

### 4.2.13. OffsetRecordSentList

<table>
<thead>
<tr>
<th>Name</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>offsets</td>
<td>&lt; OffsetRecordSent &gt; array</td>
</tr>
<tr>
<td></td>
<td>optional</td>
</tr>
</tbody>
</table>

### 4.2.14. OffsetsSummary

<table>
<thead>
<tr>
<th>Name</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>beginning_offset</td>
<td>integer (int64)</td>
</tr>
<tr>
<td></td>
<td>optional</td>
</tr>
<tr>
<td>end_offset</td>
<td>integer (int64)</td>
</tr>
<tr>
<td></td>
<td>optional</td>
</tr>
</tbody>
</table>

### 4.2.15. Partition
<table>
<thead>
<tr>
<th>Name</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>partition</td>
<td>integer (int32)</td>
</tr>
<tr>
<td>topic</td>
<td>string</td>
</tr>
</tbody>
</table>

### 4.2.16. PartitionMetadata

<table>
<thead>
<tr>
<th>Name</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>leader</td>
<td>integer (int32)</td>
</tr>
<tr>
<td>partition</td>
<td>integer (int32)</td>
</tr>
<tr>
<td>replicas</td>
<td><code>&lt;Replica</code> array</td>
</tr>
</tbody>
</table>

### 4.2.17. Partitions

<table>
<thead>
<tr>
<th>Name</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>partitions</td>
<td><code>&lt;Partition</code> array</td>
</tr>
</tbody>
</table>

### 4.2.18. ProducerRecord

<table>
<thead>
<tr>
<th>Name</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>headers</td>
<td>KafkaHeaderList</td>
</tr>
<tr>
<td>partition</td>
<td>integer (int32)</td>
</tr>
</tbody>
</table>

### 4.2.19. ProducerRecordList

<table>
<thead>
<tr>
<th>Name</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>records</td>
<td><code>&lt;ProducerRecord</code> array</td>
</tr>
</tbody>
</table>
### 4.2.20. ProducerRecordToPartition

<table>
<thead>
<tr>
<th>Name</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>headers</td>
<td>KafkaHeaderList</td>
</tr>
</tbody>
</table>

**optional**

### 4.2.21. ProducerRecordToPartitionList

<table>
<thead>
<tr>
<th>Name</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>records</td>
<td><code>&lt; ProducerRecordToPartition &gt; array</code></td>
</tr>
</tbody>
</table>

**optional**

### 4.2.22. Replica

<table>
<thead>
<tr>
<th>Name</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>broker</td>
<td>integer (int32)</td>
</tr>
</tbody>
</table>

**optional**

<table>
<thead>
<tr>
<th>Name</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>in_sync</td>
<td>boolean</td>
</tr>
</tbody>
</table>

**optional**

<table>
<thead>
<tr>
<th>Name</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>leader</td>
<td>boolean</td>
</tr>
</tbody>
</table>

**optional**

### 4.2.23. SubscribedTopicList

<table>
<thead>
<tr>
<th>Name</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>partitions</td>
<td><code>&lt; AssignedTopicPartitions &gt; array</code></td>
</tr>
</tbody>
</table>

**optional**

<table>
<thead>
<tr>
<th>Name</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>topics</td>
<td>Topics</td>
</tr>
</tbody>
</table>

**optional**

### 4.2.24. TopicMetadata

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>configs</td>
<td>Per-topic configuration overrides</td>
<td><code>&lt; string, string &gt; map</code></td>
</tr>
</tbody>
</table>
### 4.2.25. Topics

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the topic</td>
<td>string</td>
</tr>
<tr>
<td>partitions</td>
<td></td>
<td>&lt; PartitionMetadata array &gt;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>topic_pattern</td>
<td>A regex topic pattern for matching multiple topics</td>
<td>string</td>
</tr>
<tr>
<td>topics</td>
<td></td>
<td>&lt; string array &gt;</td>
</tr>
</tbody>
</table>

### 4.3. PATHS

#### 4.3.1. GET /

**4.3.1.1. Description**

Retrieves information about the Kafka Bridge instance, in JSON format.

**4.3.1.2. Responses**

<table>
<thead>
<tr>
<th>HTTP Code</th>
<th>Description</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>Information about Kafka Bridge instance.</td>
<td>BridgInfo</td>
</tr>
</tbody>
</table>

**4.3.1.3. Produces**

- application/json

**4.3.1.4. Example HTTP response**

**4.3.1.4.1. Response 200**

```json
{
    "bridge_version" : "0.16.0"
}
```
4.3.2. POST /consumers/{groupid}

4.3.2.1. Description

Creates a consumer instance in the given consumer group. You can optionally specify a consumer name and supported configuration options. It returns a base URI which must be used to construct URLs for subsequent requests against this consumer instance.

4.3.2.2. Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>Path</td>
<td>groupid</td>
<td>ID of the consumer group in which to create the consumer.</td>
<td>string</td>
</tr>
<tr>
<td>Body</td>
<td>body</td>
<td>Name and configuration of the consumer. The name is unique within the scope of the consumer group. If a name is not specified, a randomly generated name is assigned. All parameters are optional. The supported configuration options are shown in the following example.</td>
<td>Consumer</td>
</tr>
</tbody>
</table>

4.3.2.3. Responses

<table>
<thead>
<tr>
<th>HTTP Code</th>
<th>Description</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>Consumer created successfully.</td>
<td>CreatedConsumer</td>
</tr>
<tr>
<td>409</td>
<td>A consumer instance with the specified name already exists in the Kafka Bridge.</td>
<td>Error</td>
</tr>
<tr>
<td>422</td>
<td>One or more consumer configuration options have invalid values.</td>
<td>Error</td>
</tr>
</tbody>
</table>

4.3.2.4. Consumes

- application/vnd.kafka.v2+json

4.3.2.5. Produces

- application/vnd.kafka.v2+json

4.3.2.6. Tags

- Consumers

4.3.2.7. Example HTTP request

4.3.2.7.1. Request body
4.3.2.8. Example HTTP response

4.3.2.8.1. Response 200

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

4.3.2.8.2. Response 409

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

4.3.2.8.3. Response 422

```json
{
    "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. Description

Deletes a specified consumer instance. The request for this operation MUST use the base URL (including the host and port) returned in the response from the POST request to /consumers/{groupid} that was used to create this consumer.

4.3.3.2. Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>Path</td>
<td>groupid</td>
<td>ID of the consumer group to which the consumer belongs.</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td>required</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
4.3.3.3. Responses

<table>
<thead>
<tr>
<th>HTTP Code</th>
<th>Description</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>204</td>
<td>Consumer removed successfully.</td>
<td>No Content</td>
</tr>
<tr>
<td>404</td>
<td>The specified consumer instance was not found.</td>
<td>Error</td>
</tr>
</tbody>
</table>

4.3.3.4. Consumes

- application/vnd.kafka.v2+json

4.3.3.5. Produces

- application/vnd.kafka.v2+json

4.3.3.6. Tags

- Consumers

4.3.3.7. Example HTTP response

4.3.3.7.1. Response 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. Description

Assigns one or more topic partitions to a consumer.

4.3.4.2. Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>Path</td>
<td>groupid</td>
<td>ID of the consumer group to which the consumer belongs.</td>
<td>string</td>
</tr>
<tr>
<td>Type</td>
<td>Name</td>
<td>Description</td>
<td>Schema</td>
</tr>
<tr>
<td>------</td>
<td>--------------</td>
<td>-----------------------------------------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>Path</td>
<td>name</td>
<td>Name of the consumer to assign topic partitions to.</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td>required</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Body</td>
<td>body</td>
<td>List of topic partitions to assign to the consumer.</td>
<td>Partitions</td>
</tr>
<tr>
<td></td>
<td>required</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 4.3.4.3. Responses

<table>
<thead>
<tr>
<th>HTTP Code</th>
<th>Description</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>204</td>
<td>Partitions assigned successfully.</td>
<td>No Content</td>
</tr>
<tr>
<td>404</td>
<td>The specified consumer instance was not found.</td>
<td>Error</td>
</tr>
<tr>
<td>409</td>
<td>Subscriptions to topics, partitions, and patterns are mutually exclusive.</td>
<td>Error</td>
</tr>
</tbody>
</table>

### 4.3.4.4. Consumes

- application/vnd.kafka.v2+json

### 4.3.4.5. Produces

- application/vnd.kafka.v2+json

### 4.3.4.6. Tags

- Consumers

### 4.3.4.7. Example HTTP request

#### 4.3.4.7.1. Request body

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

### 4.3.4.8. Example HTTP response
4.3.4.8.1. Response 404

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

4.3.4.8.2. Response 409

```json
{
  "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. Description

Commits a list of consumer offsets. To commit offsets for all records fetched by the consumer, leave the request body empty.

4.3.5.2. Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>Path</td>
<td>groupid</td>
<td>ID of the consumer group to which the consumer belongs.</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td>required</td>
<td>required</td>
<td></td>
</tr>
<tr>
<td>Path</td>
<td>name</td>
<td>Name of the consumer.</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td>required</td>
<td>required</td>
<td></td>
</tr>
<tr>
<td>Body</td>
<td>body</td>
<td>List of consumer offsets to commit to the consumer offsets commit log.</td>
<td>OffsetCommitSeekList</td>
</tr>
<tr>
<td></td>
<td>optional</td>
<td>List of consumer offsets to commit to the consumer offsets commit log.</td>
<td></td>
</tr>
</tbody>
</table>

4.3.5.3. Responses

<table>
<thead>
<tr>
<th>HTTP Code</th>
<th>Description</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>204</td>
<td>Commit made successfully.</td>
<td>No Content</td>
</tr>
<tr>
<td>404</td>
<td>The specified consumer instance was not found.</td>
<td>Error</td>
</tr>
</tbody>
</table>

4.3.5.4. Consumes

- application/vnd.kafka.v2+json
4.3.5.5. Produces

- application/vnd.kafka.v2+json

4.3.5.6. Tags

- Consumers

4.3.5.7. Example HTTP request

4.3.5.7.1. Request body

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

4.3.5.8. Example HTTP response

4.3.5.8.1. Response 404

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

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

4.3.6.1. Description

Configures a subscribed consumer to fetch offsets from a particular offset the next time it fetches a set of records from a given topic partition. This overrides the default fetch behavior for consumers. You can specify one or more topic partitions.

4.3.6.2. Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>Path</td>
<td>groupid</td>
<td>ID of the consumer group to which the consumer belongs.</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td><strong>required</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Path</td>
<td>name</td>
<td>Name of the subscribed consumer.</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td><strong>required</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
List of partition offsets from which the subscribed consumer will next fetch records.

### 4.3.6.3. Responses

<table>
<thead>
<tr>
<th>HTTP Code</th>
<th>Description</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>204</td>
<td>Seek performed successfully.</td>
<td>No Content</td>
</tr>
<tr>
<td>404</td>
<td>The specified consumer instance was not found, or the specified consumer instance did not have one of the specified partitions assigned.</td>
<td>Error</td>
</tr>
</tbody>
</table>

### 4.3.6.4. Consumes

- application/vnd.kafka.v2+json

### 4.3.6.5. Produces

- application/vnd.kafka.v2+json

### 4.3.6.6. Tags

- Consumers
- Seek

### 4.3.6.7. Example HTTP request

#### 4.3.6.7.1. Request body

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

### 4.3.6.8. Example HTTP response

#### 4.3.6.8.1. Response 404
4.3.7. POST /consumers/{groupid}/instances/{name}/positions/beginning

4.3.7.1. Description

Configures a subscribed consumer to seek (and subsequently read from) the first offset in one or more given topic partitions.

4.3.7.2. Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>Path</td>
<td>groupid</td>
<td>ID of the consumer group to which the subscribed consumer belongs.</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td>name</td>
<td>Name of the subscribed consumer.</td>
<td>string</td>
</tr>
<tr>
<td>Body</td>
<td>body</td>
<td>List of topic partitions to which the consumer is subscribed. The consumer will seek the first offset in the specified partitions.</td>
<td>Partitions</td>
</tr>
</tbody>
</table>

4.3.7.3. Responses

<table>
<thead>
<tr>
<th>HTTP Code</th>
<th>Description</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>204</td>
<td>Seek to the beginning performed successfully.</td>
<td>No Content</td>
</tr>
<tr>
<td>404</td>
<td>The specified consumer instance was not found, or the specified consumer instance did not have one of the specified partitions assigned.</td>
<td>Error</td>
</tr>
</tbody>
</table>

4.3.7.4. Consumes

- application/vnd.kafka.v2+json

4.3.7.5. Produces

- application/vnd.kafka.v2+json

4.3.7.6. Tags

- Consumers
- Seek
4.3.7.7. Example HTTP request

4.3.7.7.1. Request body

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

4.3.7.8. Example HTTP response

4.3.7.8.1. Response 404

```json
{
  "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. Description

Configures a subscribed consumer to seek (and subsequently read from) the offset at the end of one or more of the given topic partitions.

4.3.8.2. Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>Path</td>
<td>groupid</td>
<td>ID of the consumer group to which the subscribed consumer belongs.</td>
<td>string</td>
</tr>
<tr>
<td>Path</td>
<td>name</td>
<td>Name of the subscribed consumer.</td>
<td>string</td>
</tr>
<tr>
<td>Body</td>
<td>body</td>
<td>List of topic partitions to which the consumer is subscribed. The consumer will seek the last offset in the specified partitions.</td>
<td>Partitions</td>
</tr>
</tbody>
</table>

4.3.8.3. Responses
<table>
<thead>
<tr>
<th>HTTP Code</th>
<th>Description</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>204</td>
<td>Seek to the end performed successfully.</td>
<td>No Content</td>
</tr>
<tr>
<td>404</td>
<td>The specified consumer instance was not found, or the specified consumer instance did not have one of the specified partitions assigned.</td>
<td>Error</td>
</tr>
</tbody>
</table>

4.3.8.4. Consumes

- application/vnd.kafka.v2+json

4.3.8.5. Produces

- application/vnd.kafka.v2+json

4.3.8.6. Tags

- Consumers
- Seek

4.3.8.7. Example HTTP request

4.3.8.7.1. Request body

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

4.3.8.8. Example HTTP response

4.3.8.8.1. Response 404

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

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

4.3.9.1. Description
Retrieves records for a subscribed consumer, including message values, topics, and partitions. The request for this operation MUST use the base URL (including the host and port) returned in the response from the POST request to `/consumers/{groupid}` that was used to create this consumer.

### 4.3.9.2. Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>Path</td>
<td>groupid</td>
<td>ID of the consumer group to which the subscribed consumer belongs.</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td>required</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Path</td>
<td>name</td>
<td>Name of the subscribed consumer to retrieve records from.</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td>required</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Query</td>
<td>max_bytes</td>
<td>The maximum size, in bytes, of unencoded keys and values that can be included in the response. Otherwise, an error response with code 422 is returned.</td>
<td>integer</td>
</tr>
<tr>
<td></td>
<td>optional</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Query</td>
<td>timeout</td>
<td>The maximum amount of time, in milliseconds, that the HTTP Bridge spends retrieving records before timing out the request.</td>
<td>integer</td>
</tr>
<tr>
<td></td>
<td>optional</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 4.3.9.3. Responses

<table>
<thead>
<tr>
<th>HTTP Code</th>
<th>Description</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>Poll request executed successfully.</td>
<td>ConsumerRecordList</td>
</tr>
<tr>
<td>404</td>
<td>The specified consumer instance was not found.</td>
<td>Error</td>
</tr>
<tr>
<td>406</td>
<td>The <strong>format</strong> used in the consumer creation request does not match the embedded format in the Accept header of this request or the bridge got a message from the topic which is not JSON encoded.</td>
<td>Error</td>
</tr>
<tr>
<td>422</td>
<td>Response exceeds the maximum number of bytes the consumer can receive</td>
<td>Error</td>
</tr>
</tbody>
</table>

### 4.3.9.4. Produces

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

### 4.3.9.5. Tags

- Consumers
4.3.9.6. Example HTTP response

4.3.9.6.1. 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,
  }
]
```

4.3.9.6.2. Response 404

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

4.3.9.6.3. 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."
}
```
4.3.10. POST /consumers/{groupid}/instances/{name}/subscription

4.3.10.1. Description

Subscribes a consumer to one or more topics. You can describe the topics to which the consumer will subscribe in a list (of Topics type) or as a topic_pattern field. Each call replaces the subscriptions for the subscriber.

4.3.10.2. Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>Path</td>
<td>groupid</td>
<td>ID of the consumer group to which the subscribed consumer belongs.</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td>required</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Path</td>
<td>name</td>
<td>Name of the consumer to subscribe to topics.</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td>required</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Body</td>
<td>body</td>
<td>List of topics to which the consumer will subscribe.</td>
<td>Topics</td>
</tr>
<tr>
<td></td>
<td>required</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4.3.10.3. Responses

<table>
<thead>
<tr>
<th>HTTP Code</th>
<th>Description</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>204</td>
<td>Consumer subscribed successfully.</td>
<td>No Content</td>
</tr>
<tr>
<td>404</td>
<td>The specified consumer instance was not found.</td>
<td>Error</td>
</tr>
<tr>
<td>409</td>
<td>Subscriptions to topics, partitions, and patterns are mutually exclusive.</td>
<td>Error</td>
</tr>
<tr>
<td>422</td>
<td>A list (of Topics type) or a topic_pattern must be specified.</td>
<td>Error</td>
</tr>
</tbody>
</table>

4.3.10.4. Consumes

- application/vnd.kafka.v2+json

4.3.10.5. Produces

- application/vnd.kafka.v2+json

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

- Consumers

4.3.10.7. Example HTTP request

4.3.10.7.1. Request body

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

4.3.10.8. Example HTTP response

4.3.10.8.1. Response 404

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

4.3.10.8.2. Response 409

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

4.3.10.8.3. Response 422

```json
{
    "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. Description

Retrieves a list of the topics to which the consumer is subscribed.

4.3.11.2. Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>Path</td>
<td>groupid</td>
<td>ID of the consumer group to which the subscribed consumer belongs.</td>
<td>string</td>
</tr>
</tbody>
</table>
### Path

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>Path</td>
<td>name</td>
<td>Name of the subscribed consumer.</td>
<td>string</td>
</tr>
</tbody>
</table>

#### 4.3.11.3. Responses

<table>
<thead>
<tr>
<th>HTTP Code</th>
<th>Description</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>List of subscribed topics and partitions.</td>
<td>SubscribedTopicList</td>
</tr>
<tr>
<td>404</td>
<td>The specified consumer instance was not found.</td>
<td>Error</td>
</tr>
</tbody>
</table>

#### 4.3.11.4. Produces

- application/vnd.kafka.v2+json

#### 4.3.11.5. Tags

- Consumers

#### 4.3.11.6. Example HTTP response

##### 4.3.11.6.1. Response 200

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

##### 4.3.11.6.2. Response 404

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

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

#### 4.3.12.1. Description

Unsubscribes a consumer from all topics.
4.3.12.2. Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>Path</td>
<td>groupid</td>
<td>ID of the consumer group to which the subscribed consumer belongs.</td>
<td>string</td>
</tr>
<tr>
<td>Path</td>
<td>name</td>
<td>Name of the consumer to unsubscribe from topics.</td>
<td>string</td>
</tr>
</tbody>
</table>

4.3.12.3. Responses

<table>
<thead>
<tr>
<th>HTTP Code</th>
<th>Description</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>204</td>
<td>Consumer unsubscribed successfully.</td>
<td>No Content</td>
</tr>
<tr>
<td>404</td>
<td>The specified consumer instance was not found.</td>
<td>Error</td>
</tr>
</tbody>
</table>

4.3.12.4. Tags

- Consumers

4.3.12.5. Example HTTP response

4.3.12.5.1. Response 404

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

4.3.13. GET /healthy

4.3.13.1. Description

Check if the bridge is running. This does not necessarily imply that it is ready to accept requests.

4.3.13.2. Responses

<table>
<thead>
<tr>
<th>HTTP Code</th>
<th>Description</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>204</td>
<td>The bridge is healthy</td>
<td>No Content</td>
</tr>
<tr>
<td>500</td>
<td>The bridge is not healthy</td>
<td>No Content</td>
</tr>
</tbody>
</table>
4.3.14. GET /metrics

4.3.14.1. Description
Retrieves the bridge metrics in Prometheus format.

4.3.14.2. Responses

<table>
<thead>
<tr>
<th>HTTP Code</th>
<th>Description</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>Metrics in Prometheus format retrieved successfully.</td>
<td>string</td>
</tr>
</tbody>
</table>

4.3.14.3. Produces

- text/plain

4.3.15. GET /openapi

4.3.15.1. Description
Retrieves the OpenAPI v2 specification in JSON format.

4.3.15.2. Responses

<table>
<thead>
<tr>
<th>HTTP Code</th>
<th>Description</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>204</td>
<td>OpenAPI v2 specification in JSON format retrieved successfully.</td>
<td>string</td>
</tr>
</tbody>
</table>

4.3.15.3. Produces

- application/json

4.3.16. GET /ready

4.3.16.1. Description
Check if the bridge is ready and can accept requests.

4.3.16.2. Responses

<table>
<thead>
<tr>
<th>HTTP Code</th>
<th>Description</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>204</td>
<td>The bridge is ready</td>
<td>No Content</td>
</tr>
</tbody>
</table>
4.3.17. GET /topics

4.3.17.1. Description
Retrieves a list of all topics.

4.3.17.2. Responses

<table>
<thead>
<tr>
<th>HTTP Code</th>
<th>Description</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>List of topics.</td>
<td>&lt; string &gt; array</td>
</tr>
</tbody>
</table>

4.3.17.3. Produces
- application/vnd.kafka.v2+json

4.3.17.4. Tags
- Topics

4.3.17.5. Example HTTP response

4.3.17.5.1. Response 200

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

4.3.18. POST /topics/{topicname}

4.3.18.1. Description
Sends one or more records to a given topic, optionally specifying a partition, key, or both.

4.3.18.2. Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>Path</td>
<td>topicname</td>
<td>Name of the topic to send records to or retrieve metadata from.</td>
<td>string</td>
</tr>
</tbody>
</table>
4.3.18.3. Responses

<table>
<thead>
<tr>
<th>HTTP Code</th>
<th>Description</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>Records sent successfully.</td>
<td>OffsetRecordSentList</td>
</tr>
<tr>
<td>404</td>
<td>The specified topic was not found.</td>
<td>Error</td>
</tr>
<tr>
<td>422</td>
<td>The record list is not valid.</td>
<td>Error</td>
</tr>
</tbody>
</table>

4.3.18.4. Consumes

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

4.3.18.5. Produces

- application/vnd.kafka.v2+json

4.3.18.6. Tags

- Producer
- Topics

4.3.18.7. Example HTTP request

4.3.18.7.1. Request body

```json
{
  "records" : [
    {
      "key" : "key1",
      "value" : "value1"
    },
    {
      "value" : "value2",
      "partition" : 1
    },
    {
```
4.3.18.8. Example HTTP response

4.3.18.8.1. Response 200

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

4.3.18.8.2. Response 404

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

4.3.18.8.3. Response 422

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

4.3.19. GET /topics/{topicname}

4.3.19.1. Description

Retrieves the metadata about a given topic.

4.3.19.2. Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>Path</td>
<td>topicname</td>
<td>Name of the topic to send records to or retrieve metadata from.</td>
<td>string</td>
</tr>
</tbody>
</table>

4.3.19.3. Responses
<table>
<thead>
<tr>
<th>HTTP Code</th>
<th>Description</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>Topic metadata</td>
<td>TopicMetadata</td>
</tr>
</tbody>
</table>

### 4.3.19.4. Produces
- application/vnd.kafka.v2+json

### 4.3.19.5. Tags
- Topics

### 4.3.19.6. Example HTTP response

#### 4.3.19.6.1. Response 200

```json
{
  "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. Description
Retrieves a list of partitions for the topic.

4.3.20.2. Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>Path</td>
<td>topicname</td>
<td>Name of the topic to send records to or retrieve metadata from.</td>
<td>string</td>
</tr>
</tbody>
</table>

4.3.20.3. Responses

<table>
<thead>
<tr>
<th>HTTP Code</th>
<th>Description</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>List of partitions</td>
<td><code>&lt;PartitionMetadata&gt; array</code></td>
</tr>
<tr>
<td>404</td>
<td>The specified topic was not found.</td>
<td>Error</td>
</tr>
</tbody>
</table>

4.3.20.4. Produces
- application/vnd.kafka.v2+json

4.3.20.5. Tags
- Topics

4.3.20.6. Example HTTP response

4.3.20.6.1. Response 200

```json
[ {
  "partition": 1,
  "leader": 1,
  "replicas": [{
    "broker": 1,
    "leader": true,
    "in_sync": true
  }, {
    "broker": 2,
    "leader": false,
    "in_sync": true
  }]
}, {
  "partition": 2,
  "leader": 2,
  "replicas": [{
```
4.3.21. POST /topics/{topicname}/partitions/{partitionid}

4.3.21.1. Description

Sends one or more records to a given topic partition, optionally specifying a key.

4.3.21.2. Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>Path</td>
<td>partitionid</td>
<td>ID of the partition to send records to or retrieve metadata from.</td>
<td>integer</td>
</tr>
<tr>
<td>Path</td>
<td>topicname</td>
<td>Name of the topic to send records to or retrieve metadata from.</td>
<td>string</td>
</tr>
<tr>
<td>Query</td>
<td>async</td>
<td>Whether to return immediately upon sending records, instead of waiting for metadata. No offsets will be returned if specified. Defaults to false.</td>
<td>boolean</td>
</tr>
<tr>
<td>Body</td>
<td>body</td>
<td>List of records to send to a given topic partition, including a value (required) and a key (optional).</td>
<td>ProducerRecordToPartitionList</td>
</tr>
</tbody>
</table>

4.3.21.3. Responses

<table>
<thead>
<tr>
<th>HTTP Code</th>
<th>Description</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>Records sent successfully.</td>
<td>OffsetRecordSentList</td>
</tr>
<tr>
<td>404</td>
<td>The specified topic partition was not found.</td>
<td>Error</td>
</tr>
<tr>
<td>HTTP Code</td>
<td>Description</td>
<td>Schema</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
<td>--------</td>
</tr>
<tr>
<td>422</td>
<td>The record is not valid.</td>
<td>Error</td>
</tr>
</tbody>
</table>

4.3.21.4. Consumes

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

4.3.21.5. Produces

- application/vnd.kafka.v2+json

4.3.21.6. Tags

- Producer
- Topics

4.3.21.7. Example HTTP request

4.3.21.7.1. Request body

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

4.3.21.8. Example HTTP response

4.3.21.8.1. Response 200

```json
{
  "offsets": [
    {
      "partition": 2,
      "offset": 0
    },
    {
      "partition": 1,
      "offset": 1
    },
    {
      "partition": 2,
      "offset": 2
    }
  ]
}
```
4.3.22. GET /topics/{topicname}/partitions/{partitionid}

4.3.22.1. Description
Retrieves partition metadata for the topic partition.

4.3.22.2. Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>Path</td>
<td>partitionid</td>
<td>ID of the partition to send records to or retrieve metadata from.</td>
<td>integer</td>
</tr>
<tr>
<td></td>
<td><strong>required</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Path</td>
<td>topicname</td>
<td>Name of the topic to send records to or retrieve metadata from.</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td><strong>required</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4.3.22.3. Responses

<table>
<thead>
<tr>
<th>HTTP Code</th>
<th>Description</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>Partition metadata</td>
<td>PartitionMetadata</td>
</tr>
<tr>
<td>404</td>
<td>The specified topic partition was not found.</td>
<td>Error</td>
</tr>
</tbody>
</table>

4.3.22.4. Produces

- application/vnd.kafka.v2+json

4.3.22.5. Tags

- Topics

4.3.22.6. Example HTTP response

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

Retrieves a summary of the offsets for the topic partition.

4.3.23.2. Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>Path</td>
<td>partitionid</td>
<td>ID of the partition.</td>
<td>integer</td>
</tr>
<tr>
<td></td>
<td>required</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Path</td>
<td>topicname</td>
<td>Name of the topic containing the partition.</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td>required</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4.3.23.3. Responses

<table>
<thead>
<tr>
<th>HTTP Code</th>
<th>Description</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>A summary of the offsets for the topic partition.</td>
<td>OffsetsSummary</td>
</tr>
<tr>
<td>404</td>
<td>The specified topic partition was not found.</td>
<td>Error</td>
</tr>
</tbody>
</table>
4.3.23.4. Produces
   - application/vnd.kafka.v2+json

4.3.23.5. Tags
   - Topics

4.3.23.6. Example HTTP response

4.3.23.6.1. Response 200

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

4.3.23.6.2. Response 404

```
{
   "error_code" : 404,
   "message" : "The specified topic partition was not found."
}
```
APPENDIX A. USING YOUR SUBSCRIPTION

AMQ Streams is provided through a software subscription. To manage your subscriptions, access your account at the Red Hat Customer Portal.

Accessing Your Account

1. Go to access.redhat.com.
2. If you do not already have an account, create one.
3. Log in to your account.

Activating a Subscription

1. Go to access.redhat.com.
2. Navigate to My Subscriptions.
3. Navigate to Activate a subscription and enter your 16-digit activation number.

Downloading Zip and Tar Files
To access zip or tar files, use the customer portal to find the relevant files for download. If you are using RPM packages, this step is not required.

1. Open a browser and log in to the Red Hat Customer Portal Product Downloads page at access.redhat.com/downloads.
2. Locate the AMQ Streams for Apache Kafka entries in the INTEGRATION AND AUTOMATION category.
3. Select the desired AMQ Streams product. The Software Downloads page opens.
4. Click the Download link for your component.

Installing packages with DNF
To install a package and all the package dependencies, use:

```bash
dnf install <package_name>
```

To install a previously-downloaded package from a local directory, use:

```bash
dnf install <path_to_download_package>
```

Revised on 2023-09-14 10:24:59 UTC