KIP-511: Collect and Expose Client's Name and Version in the Brokers

Status

Current state: Accepted

Discussion thread: Thread

JIRA: ⚠ Unable to render Jira issues macro, execution error.

Please keep the discussion on the mailing list rather than commenting on the wiki (wiki discussions get unwieldy fast).

Motivation

Operators of Apache Kafka clusters have little information about the type of clients connected to their clusters besides the *clientId*. Having more information about the connected clients such as their software name and version could tremendously help them to (1) troubleshoot misbehaving clients; and (2) understand the impact of a broker upgrade to their clients and inform them proactively.

Public Interfaces

ApiVersions Request/Response

ApiVersionsRequest is bumped to version 3 with two new fields. *ApiVersionsRequest* version is a flexible version (KIP-482: The Kafka Protocol should Support Optional Tagged Fields).

```
{
  "apiKey": 18,
  "type": "request",
  "name": "ApiVersionsRequest",
  "validVersions": "0-3",
  "flexibleVersions": "3+",
  // Versions 0 through 2 of ApiVersionsRequest are the same.
  // Version 3 is the first flexible version and adds ClientSoftwareName and ClientSoftwareVersion.
  "fields": [
        {"name": "ClientSoftwareName", "type": "string", "versions": "3+", "about": "The name of the client."},
        {"name": "ClientSoftwareVersion", "type": "string", "versions": "3+", "about": "The version of the
client."}
  ]
}
```

ApiVersionsResponse is bumped to version 3 but does not have any changes in the schema. Note that *ApiVersionsResponse* is flexible version but the response header is not flexible. This is necessary because the client must look at a fixed offset to find the error code, regardless of the response version, to remain backward compatible.

```
{
  "apiKey": 18,
  "type": "response",
  "name": "ApiVersionsResponse",
  // Version 1 adds throttle time to the response.
  // Starting in version 2, on quota violation, brokers send out responses before throttling.
  //
  // Version 3 is the first flexible version. Tagged fields are only supported in the body but
  // not in the header. The length of the header must not change in order to guarantee the
  // backward compatibility.
  //
  // Starting from Apache Kafka 2.4, ApiKeys field is populated with the supported versions of
  // the ApiVersionsRequest when an UNSUPPORTED_VERSION error is returned.
  "validVersions": "0-3",
  "flexibleVersions": "3+",
  "fields": [
    { "name": "ErrorCode", "type": "int16", "versions": "0+",
      "about": "The top-level error code." },
    { "name": "ApiKeys", "type": "[]ApiVersionsResponseKey", "versions": "0+",
      "about": "The APIs supported by the broker.", "fields": [
      { "name": "Index", "type": "int16", "versions": "0+", "mapKey": true,
        "about": "The API index." },
      { "name": "MinVersion", "type": "int16", "versions": "0+",
        "about": "The minimum supported version, inclusive." },
      { "name": "MaxVersion", "type": "int16", "versions": "0+",
        "about": "The maximum supported version, inclusive." }
    ]},
    { "name": "ThrottleTimeMs", "type": "int32", "versions": "1+", "ignorable": true,
      "about": "The duration in milliseconds for which the request was throttled due to a quota violation, or
zero if the request did not violate any quota." }
  ]
}
```

Metrics

We will add a new metric in the Selector to surface information about the connected clients. The mbean will be:

```
kafka.server:clientSoftwareName=(client-software-name),clientSoftwareVersion=(client-software-version),listener=
(listener),networkProcessor=(processor-index),type=(type)
```

It will contain a single value name "connections". This will contain the number of currently open connections using the given client software name and version. If the number of connections drops to 0, the mbean will be removed.

A typical example of how this will look:

```
kafka.server:clientSoftwareName=apache-kafka-java,clientSoftwareVersion=2.4.0,listener=PLAINTEXT,
networkProcessor=1,type=socket-server-metrics
```

Request Log

While the Request Log is not a public interface, it is worth mentioning that we will enrich it with the client information.

```
[2019-07-02 14:11:16,137] DEBUG Completed request:RequestHeader(apiKey=FIND_COORDINATOR, apiVersion=2,
clientId=consumer-1, correlationId=11) -- {coordinator_key=console-consumer-17661,coordinator_type=0},response:
{throttle_time_ms=0,error_code=15,error_message=null,coordinator={node_id=-1,host=,port=-1}} from
connection 192.168.12.241:9092-192.168.12.241:52149-3;totalTime:3.187,requestQueueTime:0.137,localTime:2.899,
remoteTime:0.0,throttleTime:0.098,responseQueueTime:0.048,sendTime:0.124,securityProtocol:PLAINTEXT,principal:
User:ANONYMOUS,listener:PLAINTEXT,clientInformation:ClientInformation(softwareName=apache-kafka-java,
softwareVersion=2.4.0)(kafka.request.logger)
```

Proposed Changes

The idea is to re-use the existing *ApiVersionsRequest* to provide the name and the version of the client to the broker. Clients are responsible to provide their name and version.

Broker

ApiVersions Request/Response Handling

The client does not know which ApiVersions versions the broker supports as the ApiVersions is used for this purpose. Today, the client sends an *ApiVersionsRequest* with the latest schema it is aware of. The broker handles it with the correct version if it knows it or sends back an *ApiVersionsResponse* v0 with an *UNSUPPORTED_VERSION* error to the client if it doesn't. When the client receives such error, it retries the whole process with the *ApiVersionsRequest* v0. It means that any fields added after version 0 but before the highest version supported by the broker won't be provided by the client. In our case, we would like to ensure that any future version of the ApiVersionsRequest won't impact the availability of the *ClientSoftwareName* and the *ClientSoftwareVersion*.

To circumvent this, we propose to enhance the fail back mechanism as follow:

1. When an unsupported version of the ApiVersionsRequest is received by the broker, it fails back to ApiVersionsRequest v0 and sends back an *ApiVersionsResponse* v0 with the *UNSUPPORTED_VERSION* error (as today) but the broker also populates the *api_versions* field with the supported version of the *ApiVersionsRequest*.
2. When the client receives an unsupported version fo the ApiVersionResponse, it fails back to version 0 (as today). As version 0 contains both the ErrorCode and ApiKeys fields, the client checks the error and, in case of an UNSUPPORTED_VERSION error, it checks the ApiKeys to get the supported versions or default to versions 0 if not present.

At the moment, the *ApiVersionsRequest* is handled in two different places in the broker: 1) in the *SaslServerAuthenticator* (when used); and 2) in the KafkaApis. Both places will be updated to ensure that all clients work. We have decided to not refactor the handling of the *ApiVersionsRequest* for now and to leave it for further improvements.

Metadata

We propose to attach the various metadata captured to the connection alongside existing metadata such as the principal or the listener. A registry will be created to store metadata about all the active connections. Connections will be removed when they are closed.

Validation

We propose to validate the client name and the client version with the following regular expression: ([\.\-a-zA-Z0-9])+. The *INVALID_REQUEST* error is returned to the client if the validation fails. When the client receives an *INVALID_REQUEST*, it must error out and close the connection.

Metrics & Log

The various metrics described above will be created based on the metadata available in the connection registry. Metrics will be removed when they are inactive (gauge equals to zero). The request log will be extended to include the metadata collected.

Client (Java)

ApiVersions Request/Response Handling

As mentioned earlier, when the client receives an unsupported version fo the ApiVersionResponse, it fails back to version 0 (as today). As version 0 contains both the ErrorCode and ApiKeys fields, the client checks the error and, in case of an UNSUPPORTED_VERSION error, it checks the ApiKeys to get the supported versions or default to versions 0 if not present. Then, it restarts the process with this version.I have

When the client receives an *INVALID_REQUEST* error, it will error out and close the connection.

When SASL is used, the (Java) client sends two *ApiVersionsRequest* to the broker. The first one is sent by the *SaslClientAuthenticator* and the second one is send by the *NetworkClient* when the *KafkaChannel* is established. The *SaslClientAuthenticator* always sends version 0 of the AVR. We have decided to not change this for now and to only update the second call which always happens. The reasoning behind this choice is to avoid multiplying the round trip when an unknown version is used by the client, version 0 always works.

ClientSoftwareName and ClientSoftwareVersion

The client uses the version provided in the *kafka/kafka-version.properties* file and the name *apache-kafka-java*.

Compatibility, Deprecation, and Migration Plan

What impact (if any) will there be on existing users?

Existing users extracting and parsing the Request Log may have to update their parsing logic to accommodate the new fields.

Rejected Alternatives

Put ClientSoftwareName and ClientSoftwareVersion in the RequestHeader

ClientSoftwareName and ClientSoftwareVersion could be sent in every request alongside to the clientId in the header. While this would be fairly simple to implement once KIP-482 is implemented, we believe it is not suitable if we want to collect more information in the future and would wast few bytes in every request for something which does not change within a session. It also makes the error handling weird as a request could be rejected due to its headers. Another issue is that we haven't found a way to evolve the header of the ApiVersionsResponse to support tagged fields.

Put ClientSoftwareName and ClientSoftwareVersion in the RequestHeader but provide it only once

ClientSoftwareVersion could be added to the RequestHeader but sent only in the first request to save bytes in the subsequent requests. The best would be to have it in the ApiVersionsRequest's header but it is impossible (see previous point). It would be weird to have the information in random requests and could make clients inconsistent.

Add a new request to communicate the client metadata to the broker

A new separate request/response could be used for the purpose. This option has been discarded because it would add another round trip to the broker in the establishment of the KafkaChannel.

ApiVersionsRequest combined with "prefix-based" compatibility

We have considered removing the extra round-trip to the broker when the version of the AVR is unknown by ensuring that new fields would be added to the end of the ApiVersions Request and Response. This way, we could parse newer version of the request or the response with any previous version. We have discovered this solution because it would have obliged us to freeze the RequestHeader forever which is not wise.