Notes from Research Into Queuing Protocols for Symantec CPE Stream Processing Solution

**By:** Karthik Karuppaiya, Jim Hoagland, Keith Newstadt ([cpe.symantec.com](http://cpe.symantec.com))

**Status:** work in progress

**Last Modified:** 2014-11-20

# Marconi and Kafka Gap Analysis

## APIs that are supported by Marconi but not by Kafka

|  |  |  |
| --- | --- | --- |
| **Marconi API** | **Supported by Kafka** | **Proposed Solution** |
| Create Queue | Yes |  |
| List Queues | Yes |  |
| Check Queue Existence | Yes |  |
| Delete Queue | Partially | Kafka 0.8.1 supports delete Topic. With 0.8 deleting topic is still possible with few scripts |
| Set Queue Metadata | No | We don’t need to do anything specifically for this. Marconi captures this information in its own data store |
| Get Queue Metadata | Yes | This is equivalent of List Topic API in Kafka |
| Get Queue Stats | Partially | The response that Marconi provides is not directly derivable from Kafka List Topic - need to do some additional work there |
| List Messages | Yes | Consumer API |
| Get Specific Message | No | We can accomplish this by having the consumer provide the offset of the specific message they want the Kafka to start streaming from - how this will translate into Marconi API is still not very clear. How will Marconi support this offset feature for other backends and how do we identify which partition the offset belongs to?  Question: Do we need to support this feature for SPaaS?  [KK]Answer: I am not aware of any requirements that is asking for this feature in SPaaS. |
| Get Specific messages | No | Please see above |
| Post Messages | Yes |  |
| Delete Message | No | Marconi should return a 405 if using Kafka driver. |
| Delete a set of Messages By Id | No | Marconi should return 405 if using Kafka driver. |
| Claim Messages | Yes | Kafka supports this through Fetch API  [JH] not sure about this; may not be supported for all cases |
| Query Claim | No | Not sure how this will translate into Kafka world.  Question: Is this required for Kafka?  [KK] Answer: This feature is not supported by Kafka directly. We will have to implement something in the driver, if this is required. |
| Update Claim | No | Not sure how this will translate into Kafka world.  Question: Is this equivalent of resetting the offset in Kafka?  [KK] Answer: This is not directly equivalent to resetting the offset. We will have to implement something in the driver for this as well. |
| Release Claim | No | Not sure how this will translate into Kafka world. |

## SPaaS Requirement for Queuing API - Gaps in Marconi

|  |  |  |
| --- | --- | --- |
| **Requirement** | **Supported by Marconi** | **Proposed Solution** |
| Create Queue | Yes |  |
| List Queues | Yes |  |
| Check Queue Existence | Yes |  |
| Delete Queue | Yes |  |
| Authentication | Yes |  |
| Authorization | Maybe | Need to validate that X-Project-Id will satisfy all our needs |
| High Performance Producer API | No | Build a HTTP Streaming API to produce messages |
| High Performance Consumer API | No | Build a HTTP Streaming API to consume messages |
| Non destructive cooperative Read (Group of consumer threads work together to consume the messages from a queue, but do so in a way that the messages still remain in the queue) | No | We can try and use the Client-ID provided by Marconi to see if that satisfies this need. If not, this is something that needs to be built into Marconi. This will be quite a complex piece. |
| Message sizes larger than 4k | No | Question: Is this a real requirement for SPaaS?  [KK] Answer: Not yet. But, it might be in the future. |
| Semantic Partitioning ala Kafka (queue is set up such that the same consumer within a set of a cooperating consumers will get all the related messages) | No | Question: Is this a real requirement for SPaaS? If so, this will be a complex piece to implement  JH: Yes, this seems to be important for upcoming use cases |

# HTTP live streaming vs WebSocket vs HTTP long polling

See top answer here: <http://stackoverflow.com/questions/12555043/my-understanding-of-http-polling-long-polling-http-streaming-and-websockets>

We want native binary support (no need to Base64 encode or use some other inefficient encoding), so WebSocket is the best in that regard.

WebSockets are also good in terms of latency and bandwidth overhead. It's also bi-directional.

Long polling - seems clunky - there is period just after got something from server that the timeliness of something new that came in will not be as high. Forever response is like long polling but the same HTTP response is used for multiple chunks.

Live streaming - it is nice to have text-based API.

Both long polling and live streaming send content in one direction- from server to client (consuming direction). To support client to server (producing direction), we would have to pair that with simple HTTP requests. See for example what Bayeux does (section 1.4.4 to 1.4.6 of <http://svn.cometd.com/trunk/bayeux/bayeux.html>). This breaks symmetry and may have additional overhead for the producing direction due to having a new request for each.

just use streaming interface for data operations and use a RESTful interface for other operations

## WebSocket notes

* WebSocket protocol defined in 65 page RFC 6455 with section 1 giving a good informal overview
* W3C defines a WebSocket API
* WebSocket starts as HTTP but uses HTTP protocol upgrade feature to switch to WebSocket
* supports SSL tunneled (wss vs ws)
* both clients and servers send a non-overlapping sequence of messages; the sides don’t necessarily need to have a complete message ready to start sending a message (in fact the message can span an indefinite amount of time)
* kinds of messages:
  + text (payload data is UTF-8)
  + binary (payload data is raw bytes)
  + connection close
  + ping
  + pong
* messages consist of a contiguous sequence of one or more frames
  + frames have markers for whether they are the final frame for a message and if they are a continuation of a message
  + should not assume frames will be same on client and server (an intermediary may combine or split), so frame boundaries can have no semantic meaning
* after Close message is exchanged both directions, the TCP connection is closed
* client to server messages must be “masked” (XORed with a random mask key) per RFC section 5.1 but masking not allowed in reverse
* must be careful to not be in Connecting state with same server more than once from same client, even if host names are different
* there was a multiplexing WebSocket extension Internet Draft to support multiple logical WebSocket connections over a single TCP connection, but the I-D expired Jan 2014
* Oracle Java 7 EE provides a [facility for using WebSocket](http://docs.oracle.com/javaee/7/tutorial/doc/websocket.htm); you just need to provide methods/classes
* There are a few Python WebSocket implementations (see <http://stackoverflow.com/questions/5839054/websocket-server-in-python>), including:
  + [pywebsocket](http://code.google.com/p/pywebsocket/); server only; standalone or inside Apache HTTP Server; “pywebsocket is intended for **testing** or **experimental** purposes” (does this mean we shouldn’t use it?); Chromium project; we might expect this to be most reliable in terms of complete and correct implementation of RFC 6455
  + [tornado](http://www.tornadoweb.org/en/stable/): server only; “[Tornado](http://www.tornadoweb.org) is a Python web framework and asynchronous networking library, originally developed at [FriendFeed](http://friendfeed.com). By using non-blocking network I/O, Tornado can scale to tens of thousands of open connections, making it ideal for [long polling](http://en.wikipedia.org/wiki/Push_technology#Long_polling), [WebSockets](http://en.wikipedia.org/wiki/WebSocket), and other applications that require a long-lived connection to each user.” ([WebSocket handling](http://www.tornadoweb.org/en/stable/websocket.html))
  + [Autobahn|Python](http://autobahn.ws/python/); server and client; sounds like a robust implementation with examples; supports Twisted (event-driven networking engine) and asyncio. Supports a message frame-based interface via [IWebSocketChannelFrameApi](https://github.com/tavendo/AutobahnPython/blob/master/autobahn/autobahn/websocket/interfaces.py); claims high performance
  + [websockets 2.1](https://pypi.python.org/pypi/websockets); server and client; “websockets is a library for developing WebSocket servers and clients in Python. It implements RFC 6455 with a focus on correctness and simplicity. It passes the Autobahn Testsuite.” ([documentation](http://aaugustin.github.io/websockets/))
  + [websocketserver.py](https://gist.github.com/jkp/3136208); server only; “A simple WebSockets server with no dependencies”; 68 lines of code; doesn’t deal with fragmentation; seems we should only use it for experimentation

# Case studies (other HTTP streaming interfaces)

For the Twitter Streaming APIs, the client connects to the server and receives incremental responses as new tweets become available. There is a single HTTP request and the HTTP response employs a chunked transfer encoding.

MagnetoDB Bulk Load Interface has RESI API and sends one request per chunk of data; seems too high of overhead if we are supporting low latency. Uses separate channel to get status (uncommon design pattern); being able to explicitly check status has its appeal, but you can also get that from the data channel responses.

Force.com Streaming API uses Bayeux.

Narconi: [https://github.com/tjanczuk/narconi](https://github.com/tjanczuk/narconi#httpwebsocket-protocol-details); supports RabitMQ operations via WebSocket

# Queuing technology notes

## Kafka

* queue=topics
* delete only through a topic-level retention parameter
* having consumers in same consumer group has them cooperate on getting all messages from a topic
* horizontally scalable and highly available

## RabbitMQ

* reliable
* flexible routing
* highly available queue
* work queues (cooperative read) and fanout exchanges
* incoming messages not targeted for a particular queue -- just to an exchange
* routing keys can be used to put all messages with a certain routing key in same queue (kind of like semantic partitioning in Kafka - kinda)

## ActiveMQ

* Active Groups provide for locks (like claims?)
* Queue consumer clusters support groups of consumers doing cooperative read (“high performance load balancing of messages on a queue across consumers”), with support for re-assignment of messages if consumer goes away
* supports distributed queues via Network of Brokers
* supports per-message message expiration
* Message Groups provide:
  + guaranteed ordering of the processing of related messages across a single queue
  + load balancing of the processing of messages across multiple consumers
  + high availability / auto-failover to other consumers if a JVM goes down
* horizontally scalable
* can send large blobs as messages
* provides for authorization and other security features (<http://activemq.apache.org/security.html>)