Design Notes for Kafka Driver for Queuing Streaming Interface PoC

This document contains some notes from designing the Kafka driver for the queuing streaming interface.

Table of Contents

Operating Environment 1

Notes on Kafka-python 1

Produce-direction Kafka-python interface: 2

SimpleProducer 2

Keyed 3

low level 4

Consume-direction Kafka-python interface: 4

SimpleConsumer 4

MultiProcessConsumer 6

Common for both 7

Design 8

KafkaDriver.\_\_init\_\_(driver\_args, event\_loop) 8

produce direction 8

KafkaDriver.prepare\_for\_append\_stream(self, queue\_name) 8

KafkaDriver.append(self, payload, ttl) 8

KafkaDriver.cancel\_append\_stream(self) 9

consume direction 9

KafkaDriver.init\_get\_stream(self, get\_message\_stream, queue\_name, starting\_marker, echo\_requested, include\_claimed) 9

KafkaDriver.periodically\_check\_for\_new\_messages(self) 9

KafkaDriver.check\_for\_new\_messages(self) 9

KafkaDriver.cancel\_get\_stream(self) 10

Future work 10

# Operating Environment

Driver classes get the calls described in the “Interface between server interface and driver in Queuing Streaming Interface PoC“ document

# Design

Following the driver interface definition we’ll implement a KafkaDriver class, making use of [Kafka-python](https://github.com/mumrah/kafka-python).

## KafkaDriver.\_\_init\_\_(driver\_args, event\_loop)

something like:

kafka = KafkaClient("localhost:9092")

get\_message\_count=0 # count of message we have sent from various queues

args get passed in from server interface when setting up driver in driver\_args

we’ll store the event loop for use in the consume direction (we could also use if to for any periodic maintenance that is needed)

## produce direction

### KafkaDriver.prepare\_for\_append\_stream(self, queue\_name)

save provided queue name as current topic

create a kafka-python SimpleProducer

* async=True
* req\_acks=SimpleProducer.ACK\_AFTER\_LOCAL\_WRITE
* ack\_timeout=5000
* batch\_send=True
* batch\_send\_every\_n= TBD
* batch\_send\_every\_t=1000
* random\_start=True

### KafkaDriver.append(self, payload, ttl)

discard ttl

producer.send\_messages(<current topic>, [ payload ]) # this makes some copies, not sure how many

### KafkaDriver.cancel\_append\_stream(self)

producer.stop()

delete producer

current topic=none

## consume direction

### KafkaDriver.init\_get\_stream(self, get\_message\_stream, queue\_name, starting\_marker, echo\_requested, include\_claimed)

save provided queue name as current topic

save get\_message\_stream (instance of MessageStream)

consume group taken from starting marker param

discard other params with warning

get/save max-get-request-batch from get\_message\_stream.space\_avail()

create a SimpleConsumer:

* topic=<current topic>
* partitions (omit)
* auto\_commit\*: default values
* fetch\_size\_bytes= max-get-request-batch\*4K
* buffer\_size/max\_buffer\_size (omit)
* iter\_timeout=None

call self.periodically\_check\_for\_new\_messages() # kick of periodic checking for new messages

### KafkaDriver.periodically\_check\_for\_new\_messages(self)

if get\_message\_stream is not None: # still providing messages

call KafkaDriver.check\_for\_new\_messages()

self.new\_msg\_check\_callback= event\_loop.call\_later(0.010, self.periodically\_check\_for\_new\_messages) # schedules self to run again after 10ms

### KafkaDriver.check\_for\_new\_messages(self)

max-msgs-to-get= run get\_message\_stream.space\_avail()

return if max-msgs-to-get == 0

messages=consumer.get\_messages(count=max-msgs-to-get, block=False)

* count= max-msgs-to-get # how many message slots we open
* block/timeout: no block, no timeout

for each message in messages:

self.get\_message\_count += 1

get\_message\_stream.add\_message() with:

* payload=message
* marker=get\_message\_count
* id=get\_message\_count
* ttl=2^31-1
* age= 0
* omit claim\_\*

### KafkaDriver.cancel\_get\_stream(self)

self.new\_msg\_check\_callback.cancel() # cancel call to periodically\_check\_for\_new\_messages()

consumer.stop()

delete consumer

current topic=none

delete get\_message\_stream including anything currently present

# Future work

There were skipped in PoC driver implementation:

* get cooperative read working properly; seems to be a kafka-python issue with Kafka 0.8.0 (seems related: <https://github.com/mumrah/kafka-python/issues/147>)
* allow consumer to set starting marker (currently always starts from beginning)
* actually provide a valid offset as a marker (may have to use lower level kafka interface)
* admin should be able to configure Kafka driver (no hard-coded params)
  + async or not (currently always async)
  + buffer size/timeout
* allow KafkaDriver to possibly send a backlog of messages more quickly by immediately checking again for more messages from Kafka if there is still space in MessageStream
* need to more correctly get consume group in from interface (protocol needs to provide way)
* if the single process being shared between the driver and the interface code starts to become a bottleneck (using 100% CPU), we could use asyncio to start a separate thread for the driver’s grabbing of messages
* to get offset commits via kafka-python, we may need to use kafka 0.8.1 or later