Kafka Connectors to/from IBM MQ – an MQ for z/OS perspective

Tony Sharkey
Published on 24/07/2019

What is Apache Kafka?

There are many descriptions available on the internet of what Kafka is, including the Apache Kafka introduction site which states that Apache Kafka® is a distributed streaming platform.

Some of the differences between Kafka and a typical MQ messaging system are:

- Producers of the messages publish to Topics
- Consumers subscribe to the Topics
- Messages are arrays of bytes and can be JSON objects, Strings etc.
- Topics are logs of messages.
- Kafka is run as a cluster of servers, each of which is called a broker.

How can I use Apache Kafka with my IBM MQ subsystem?

Kafka connectors allow data to be passed between your MQ subsystems and your Apache Kafka cluster(s).

Kafka-connect-mq-source is a Kafka Connect source connector for copying data from IBM MQ into Apache Kafka, i.e. MQ is the source of the data.

Kafka-connect-mq-sink is a Kafka Connect sink connector for copying data from Apache Kafka into IBM MQ, i.e. Apache Kafka is the source, and IBM MQ is the target.

The documentation provided with these connectors makes it relatively straightforward to configure even for a first-time Kafka user (and z/OS dinosaur!).

Configuration options

The Kafka connectors do not have to be run locally to the Kafka broker, so there are 3 configurations we have considered:

1. Connectors local to Kafka but remote from z/OS, connecting as a client via SVRCONN channel to z/OS queue manager.
2. Connectors local to z/OS, running in the Unix System Services (USS) environment, connecting as a client via SVRCONN to an IBM Advanced MQ queue manager on z/OS.

3. Connectors local to z/OS, located in the USS environment, connecting via bindings to z/OS queue manager.

As the Kafka connectors are written in Java™, when running in the USS environment the connectors are largely eligible for offload to zIIP processors.

Gotcha's

The default configuration for Kafka sink connectors into MQ is to use persistent messages, rather than setting the MQMD Persistence option to "MQPER_PERSISTENCE_AS_Q_DEF".
Persistence can be overridden in the properties file, but only set to true or false.

"Lazy" commits – when sending or receiving messages from IBM MQ, the connector does not issue the commit for each message, rather it will issue the commit periodically. We have seen instances with workloads that have bursts of activity interspersed with idle periods that have resulted in messages not being committed for a period of minutes, or until the next burst of work occurs.

When using client connections with the Kafka MQ Connectors, it is necessary to ensure the SVRCONN channel is configured with a SHARECNV value greater than 0 in order to support JMS2.0 APIs.

IBM MQ configurations

For the measurements run in this blog, we made relatively few MQ definitions:

```
DEFINE STGCLASS(KAFKA) PSID(3)

DEF QL(KAFKA_MQ) SHARE DEFSOPT(SHARED) STGCLASS(KAFKA) +
INDXTYPE(CORRELID) MAXMSGL(100000001) REPLACE

DEF QL(MQ_KAFKA) SHARE DEFSOPT(SHARED) STGCLASS(KAFKA) +
INDXTYPE(CORRELID) MAXMSGL(100000001) REPLACE

ALTER CHL(SYSTEM.DEF.SVRCONN) CHLTYPE(SVRCONN) SHARECNV(10)
```

Note that whilst we have set the INDXTYPE, by default we are not using the CORRELID to identify specific messages.

Kafka Connector configurations

When starting the standalone Kafka connector, the default properties can be overridden using the properties file.

For each of the configurations used we set the following *key* values:
MQ Sink – Bindings

```
topics=TO_MQ
mq.queue=KAFKA_MQ
mq.persistent=false
mq.queue.manager=VTS1
mq.connection.mode=bindings
```

MQ Sink – Client

```
topics=TO_MQ
mq.queue.manager=VTS1
mq.queue=KAFKA_MQ
mq.persistent=false
mq.connection.mode=client
mq.channel.name=SYSTEM.DEF.SVRCONN
mq.connection.name.list=10.20.37.21(2201)
```

MQ Source – Bindings

```
topics=FROM_MQ
mq.queue=MQ_KAFKA
mq.queue.manager=VTS1
mq.connection.mode=bindings
```

MQ Source – Client

```
topics=FROM_MQ
mq.queue.manager=VTS1
mq.queue=MQ_KAFKA
mq.connection.mode=client
mq.channel.name=SYSTEM.DEF.SVRCONN
mq.connection.name.list=10.20.37.21(2201)
```

Note that the contents of the client properties file are the same regardless of whether running locally to the Kafka broker or not.

Measurements

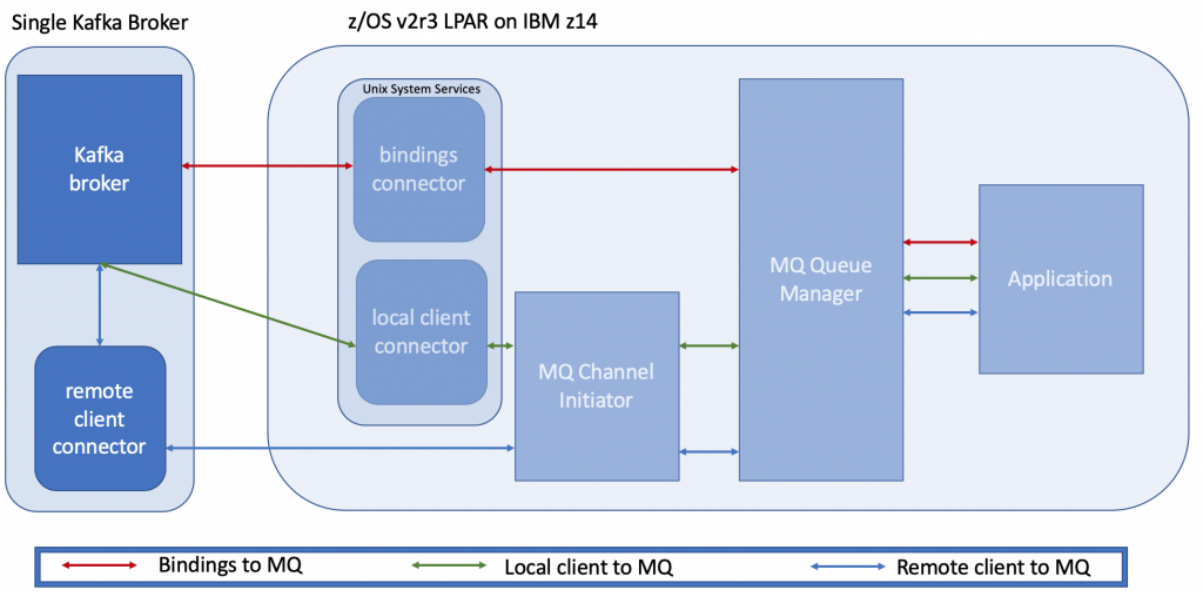
Whilst not the recommended configuration, for the purposes of this blog we are not considering load balancing, failover and high availability or indeed any of those features that make Apache Kafka such a desired technology and as such we have used a single broker in the Kafka Cluster – this is partly for simplicity and also partly due to availability of test machines.

As mentioned previously we have looked at 3 configurations:–

1. Kafka connector remote from z/OS using client connection to MQ
2. Kafka connector local to z/OS, using USS-based client connection to MQ.

3. Kafka connector local to z/OS, using USS-based bindings mode connection to MQ.

These configurations can be pictured thus:



Kafka connectors into MQ

For each of these configurations we look at the cost of a message flow for a fixed rate workload.

We have chosen a sustainable fixed rate workload of 3000 messages per second, to remove variability from disk response time on the machine hosting the Kafka broker and so we see similar behavior from the 'lazy' commits.

With this fixed transaction rate, the maximum depth of the MQ queues is less than 200, indicating that transaction times are less than 100 milliseconds, i.e. in both the MQ as a source and MQ as a sink configurations, the consumer is able to keep pace with the producer.
Note we have calculated this from:
1,000,000 microseconds per second divided by rate (3000), gives the time for each message (333 microseconds).
Backlog is never more than 200, so 200 x 333 is 66.6 milliseconds.

In all cases the message size is 1KB and we have chosen to enforce non-persistent messages.

When using the connectors in client mode, the SVRCONN channel is started once, and remains running until the connector ends.

MQ as the source

When running the MQ source configuration, we use an application to put messages to the MQ_KAFKA queue at the specified rate.

The Kafka source connector then gets the message from the queue and publishes to the topic "FROM_MQ".

Messages are then consumed from the “FROM_MQ” by the “kafka-consumer-perf-test.sh” harness.

In terms of cost per messages, we saw the following:

Connector location	Connection type to queue manager	Cost on z/OS per queue manager CPU Microseconds		
		Total	MQ MSTR+CHIN	Connector
Unix System Services	Bindings	100.6 (4.9)	2.4	98.2 (2.5)
Unix System Services	Client	152.9 (60.9)	58.4	94.5 (2.5)
Remote from z/OS	Client	55.7 (55.7)	55.7	N/A

Notes:
When the connector runs on z/OS, the work is largely zIIP eligible, so with sufficient zIIPs available these *connector* costs will be negligible.

The costs shown in blue are those observed when maximum offload of zIIP-eligible work occurs.

There is some variability between these measurements as the workload is not driving the z/OS LPAR particularly hard.

In terms of MQ costs, the client mode tests are significantly higher cost than the bindings mode test which is due to the cost incurred in the channel initiator. The benefit of using the client mode configuration is that the connector does not have to be located on the same LPAR or even machine as the queue manager but still can be managed with z/OS and make use of transport technologies such as SMC-R, SMC-D or hipersockets.

MQ as the sink *(Kafka as the source)*

When running the MQ sink configuration, we use the “kafka-producer-perf-test.sh” utility to publish messages to the topic “TO_MQ” at the specified rate.

The Kafka sink connector then consumes (non-destructively) the message from the topic and uses MQPUT1 to put the message to the queue “KAFKA_MQ”.

Messages are then got-destructively from the queue by a server application.

In terms of cost per messages, we saw the following:

		Cost on z/OS per queue manager CPU Microseconds		

Connector location	Connection type to queue manager	Total	MQ MSTR+CHIN	Connector
Unix System Services	Bindings	61.5 (6.2)	4.5	57 (1.7)
Unix System Services	Client	71.5 (30.9)	29.6	41.9 (1.3)
Remote from z/OS	Client	28.8 (28.8)	28.8	N/A

Notes:
Once more, when the connector runs on z/OS the work is largely zIIP eligible, so with sufficient zIIPs available these connector costs will be negligible.

The costs shown in blue are those observed when maximum offload of zIIP-eligible work occurs.

There is some variability between these measurements as the workload is not driving the z/OS LPAR particularly hard.

Why does MQ as the source cost more than MQ as a sink?

In client mode we have seen that the average cost to MQ address spaces of MQ source is 55-58 CPU microseconds but the MQ sink configuration is only 29 CPU microseconds, so why the difference?

If we only include the MQ CHIN address space cost, we have the following approximate cost per message:

MQ Sink: 25 CPU microseconds
MQ Source: 54 CPU microseconds

From the MQ accounting and statistics data (`TRACE(A) CLASS(3) and TRACE(S) CLASS(4)`) we can see that different work occurs for each message.

For the MQ Sink measurement we see MQPUT1's being used, but in addition we also see the lazy commit occurring on average every 7.4 messages. When we look at the average cost in the channel initiator per adapter request we see an average of 20 CPU microseconds (based on adapter and dispatcher calls).

The MQ Source measurement sees a similar cost per adapter request, ~20 CPU microseconds, but in the MQ Source measurement we have additional requests per message successfully gotten:

- A number of failed (or empty) gets, where there is no message on the MQ queue for the connector task to get.
- The ratio of commits per MQGET is lower – on average 2.6 messages are successfully gotten between commits.

- The MQ Source connector is using MQ JMS to call the MQCTL API with MQOP_SUSPEND and MQOP_RESUME to control the flow of messages. Whilst the class(3) accounting cost of these is 0, i.e. less than 0.5 microseconds, there is cost due to the flow between connector and channel initiator.

What this means is whereas the average number of requests per message for the MQ Sink averages 1.3 adapter requests per message, the number of requests per message for the MQ Source averages 2.9.

Given that each request costs 20 CPU microseconds, the additional requests for the MQ Source connector make a significant difference to the cost on the MQ channel initiator address space.

What if the messaging rate is unsustainable?

In the measurements described so far, we have used the term ‘sustainable rate’, which for the purposes of this blog means that all the configurations would be able to sustain this messaging rate for sustained periods without constraints.

What if the messaging rate is higher? For example when we try to drive 10,000 messages per second through the “MQ as the source” configuration, we see different behaviour depending on how the connector connects to the MQ queue manager as well as where the connector is located.

In this instance, by unsustainable we mean that there is a build-up of messages on the queue due to the getting task being unable to get the messages as fast as they are being put. By design, MQ buffer pools and page sets allow messages to remain in queues but should the put rate exceed the get rate for long enough, capacity limits will be reached unless some mitigating action occurs.

In the bindings configuration, the connector was able to keep pace with the z/OS-based putting application such that the queue depth never exceeded 3,500 messages. Given the messaging rate, this means that the end to end latency was approximately 1/3 of a second.

With the client configuration where the connector was running in the Unix System Services environment, we saw that the z/OS-based application was putting at a faster rate than the Kafka consumer was able to get the messages. In this measurement the consumer was able to sustain approximately 9,400 messages per second.
In this configuration, despite the queue manager having large buffer pools (4GB), the queue depth built up sufficiently that MQ initiated immediate (synchronous) writes to page set, which slowed the putting application. Despite this, the queue depth reached 100,000 messages, suggesting a latency of the order of 10 seconds.

In the final configuration, where the connector is running locally to the Kafka broker, but remote from the z/OS queue manager, the Kafka consumer was only able to get messages at 7,200 messages per second. This meant that the MQ queue depth increased more rapidly, once more causing the buffer pool to fill, page set to expand and the application to be affected by immediate writes to page set. The queue depth exceeded 200,000 messages, which with the consumer rate of 7,200 per second, gives a latency of almost 30 seconds.

What else might higher message rates do?

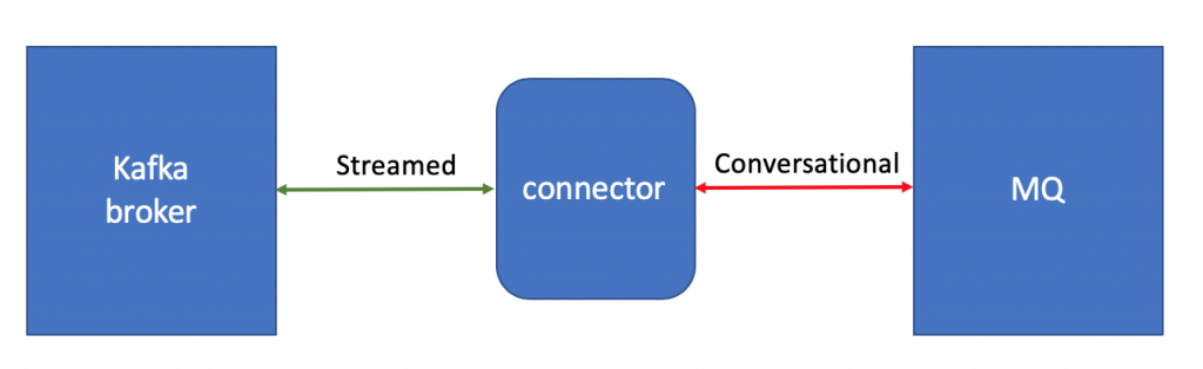
The higher messaging rate affected the MQ work the connector performed – when the desired messaging rate was increased, we saw that the ratio of messages per commit also increase – in bindings we saw an average of 17 messages being gotten between commits but in the client mode we saw on average 230 messages between commits.

Additionally with the increased throughput, there were significantly less MQCTL calls – which brought the average cost per message in the MQ channel initiator from 54 microseconds to approximately 30 microseconds!

Does the location of the connector matter?

In short, yes, and I will explain why..

Consider the following diagram:



Data flows from Kafka Connectors to MQ and Kafka

This consists of 2 sets of flows:

1. Kafka broker to/from connector – this flow is largely a unidirectional stream.
2. Connector to/from MQ – this is a pseudo-conversational flow.

For the connector to/from MQ running in a conversational mode, there is more impact from network latency as the connector must wait for a response from MQ before performing the next action.

Also consider that TCP/IP is optimized for streaming type workloads, as TCP is able to use features such as dynamic right sizing to increase the size of the send windows, to send data more efficiently.

Finally, the data flowing between MQ and the connector is typically larger as it contains MQ's implementation headers – so a 1KB message payload may result in 1.5KB being transferred, i.e. more data is flowed between MQ and the connector.

As a result, we would suggest the following order in terms of preference to be able to achieve the best throughput:

1. Connector in Unix System Services using binding-type connections to MQ
2. Connector in Unix System Services using client-type connections to MQ
3. Connector remote from MQ and using client-type connections.

Do persistent messages make a difference to the cost?

Yes – and the difference is primarily in the MQ MSTR address space. For these 1KB messages we see the average cost per message increase by 17 CPU microseconds. This 17 CPU microsecond increase includes the cost of logging but we are unable to differentiate between the work driven by the Kafka Connectors through the channel initiator, and the applications running on z/OS.

And finally…

Hopefully that gives an insight into the performance you might expect to see when connecting Apache Kafka with your IBM MQ z/OS queue manager, as well as some of the considerations to take into account when deciding where to run your connectors.