



Using Streaming Queues on IBM MQ 9.3 in a distributed environment

Introduction

This lab demonstrates the Streaming Queue feature added to MQ in the 9.2.3 CD release.

Streaming Queues allow you to configure any local or model queue with the name of a second named queue. This second queue is referred to as a Stream Queue. When messages are put to the original queue, a duplicate copy of each message is also placed on the stream queue. The Streaming Queue feature allows you to create a duplicate stream of messages which can be used for later analysis, logging, or storage without affecting the business applications using the original queue.

In this lab, you will:

- 1) Create 2 queues. An application queue that messages are put to by a sending application, and a stream queue for duplicate messages to be streamed to.
- 2) Configure the application queue to stream messages to the stream queue
- 3) Send messages to the application queue and observe them being duplicated to the stream queue
- 4) Modify the stream queue and the streaming quality of service to demonstrate the behavior when duplicate messages cannot be streamed
- 5) Inspect some of the messages to see that they are identical

We will not demonstrate messages being consumed from either of the queues. Stream queues are regular MQ local queues and getting messages from them is done in the same way as any other MQ queue.

We will not demonstrate using a queue alias to topic as the stream queue, although this is supported. More information on using queue alias as the stream queue is available in the IBM Documentation at <https://www.ibm.com/docs/en/ibm-mq/9.2?topic=queues-streaming-remote-alias>

Diagram 1: Topology overview for stream queues

The topology diagram shows the basic capability of stream queues. A regular local queue that is currently being used by MQ applications can be configured to stream a duplicate of every message put to that queue, to a second destination called a stream queue.

Basic Requirements

A Windows or Unix operating system

Acknowledgements

This lab was written by Matthew Whitehead (mwhitehead@uk.ibm.com) and modified by Dorothy Quincy (dorothy.quincy@ibm.com) for use by the Washington Systems Center.

Entering commands for this lab

In this lab, you will come across some lengthy commands to enter. You will be using a Linux Shell like bash. To avoid a lot of typing, you may copy the commands from this document and execute on the Shell.

Setup configuration for the lab

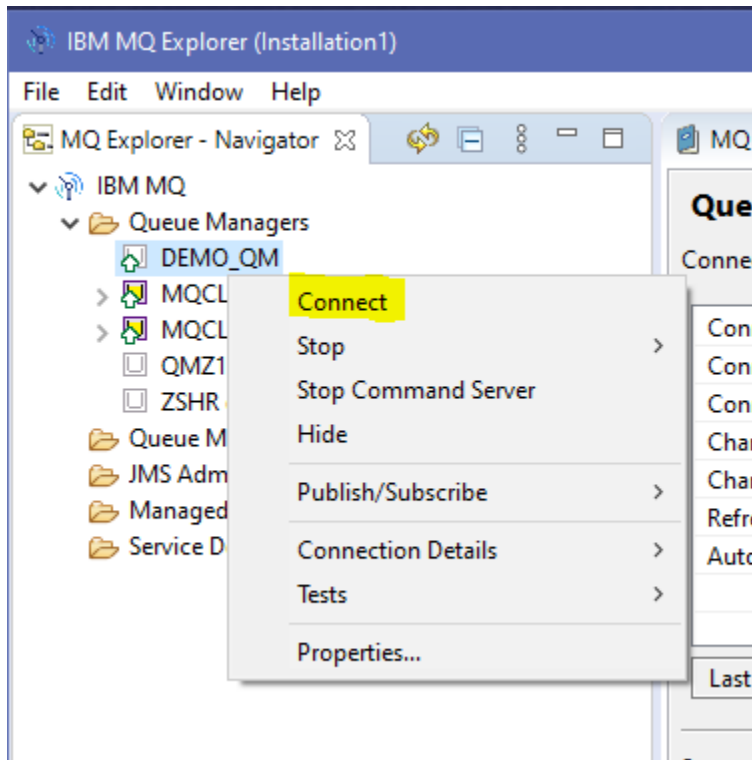
In this lab you will create and start a single queue manager.

The MQSC snippets have been shown being written to the console and piped in to runmqsc each time. You could choose to start runmqsc once and paste just the MQSC commands into runmqsc instead if you prefer

Running the lab

1. On your Windows home screen, select the MQ Explorer application from the Desktop. You'll notice that as the application opens, the loading screen indicates that we are using MQ version 9.3.
2. For this lab, you will be using a queue manager already in existence on the demo environment. The queue manager is called DEMO_QM. Go ahead and connect to the

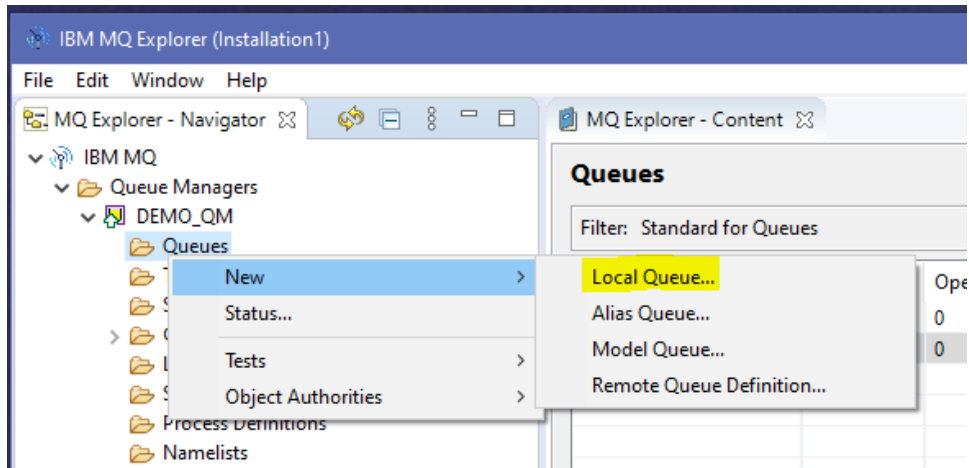
DEMO_QM on MQ Explorer. You will know you're connected if the white box to the left of the DEMO_QM turns into a yellow box.



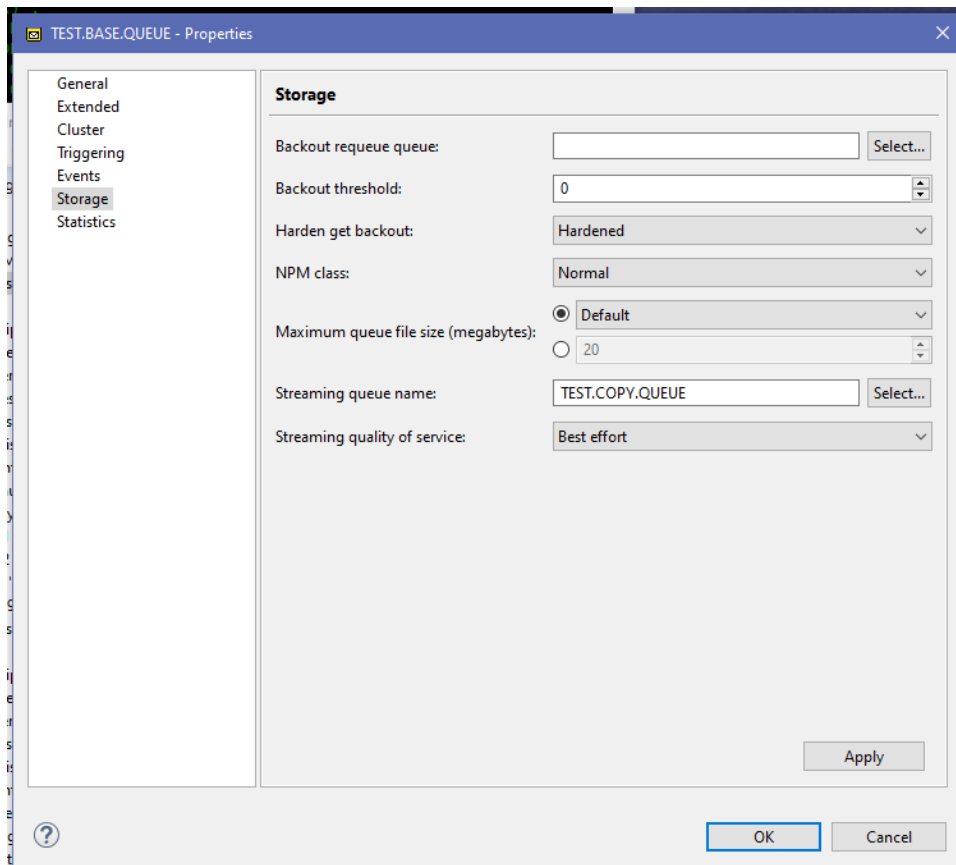
TECH TIP: You'll notice there are other queue managers connected to this instance of MQ Explorer. DEMO_QM, MQCLUS1, and MQCLUS2 are all local queue managers. QMZ1 and ZSHR are remote queue managers, each connected to z/OS over the network. For this lab, we are only interested in DEMO_QM.

3. Now define a queue that messages will be streamed to by selecting the dropdown arrow to the left of DEMO_QM on the menu panel and then right-clicking 'Queues' when it pops up.

In the lab, we will call our stream queue 'TEST.COPY.QUEUE'. One way you might use a streaming queue is to take a copy of every application message and log it to an external data store such as a database.



4. Next, we will create our application queue that the putting application will send messages to. This is also called our base queue. When we create the application queue we will specify TEST.COPY.QUEUE as the streaming queue to put duplicate messages on.

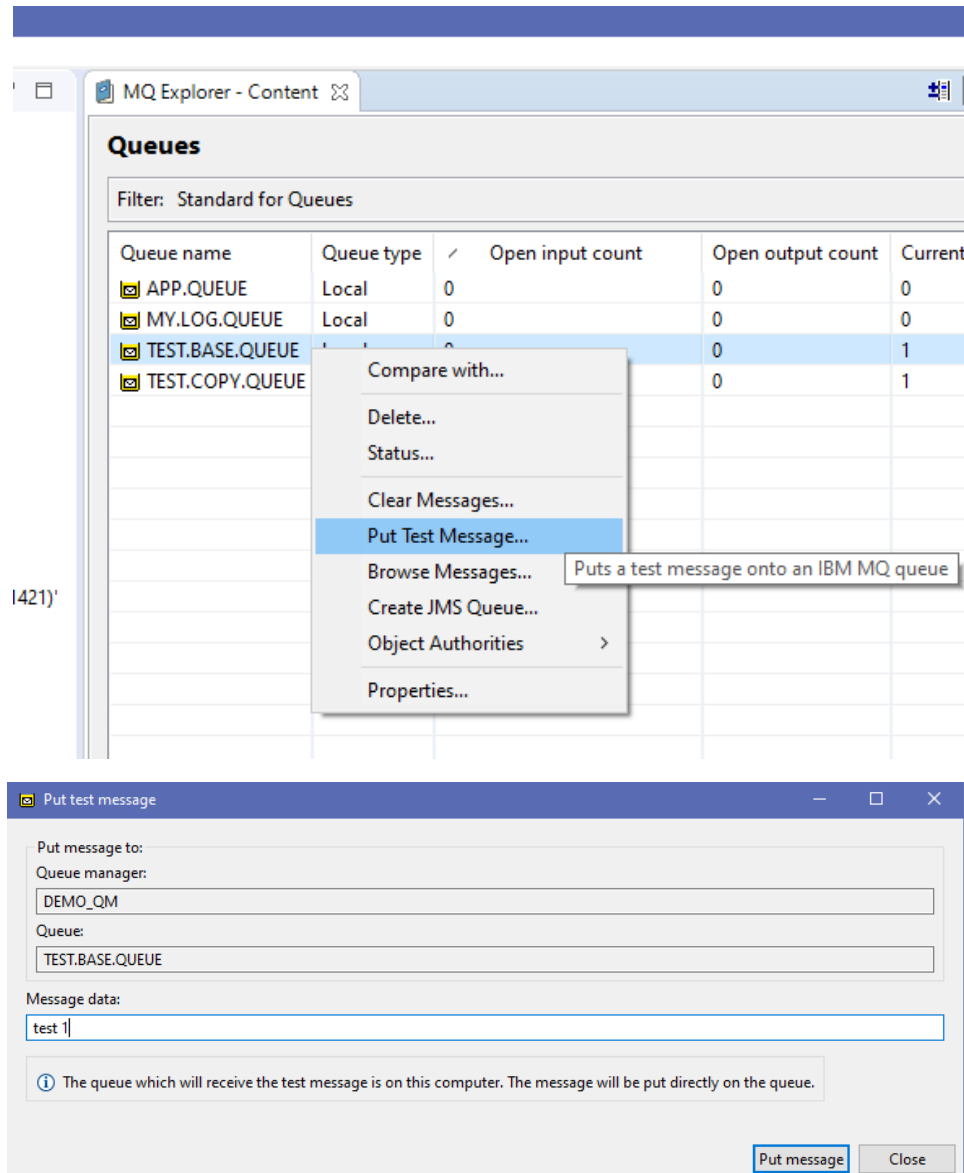


No additional configuration is required to set up our stream queue with the default settings.

5. Now that we have an application queue and a stream queue we will put some messages to the application queue. First let's check that the queues are both empty (obviously we have only just created them but we'll re-use this command throughout the lab):

Current queue depth
0
0
0
0

- We are going to put messages onto our base queue. Go ahead and put 3 messages on TEST.BASE.QUEUE. Note: we will not put any messages directly to our TEST.COPY.QUEUE – the queue manager will do that for us.



- Having put some messages to TEST.BASE.QUEUE we can check to see how many messages are on the two queues by looking under the 'Current queue depth' for both the base and stream queue:

Current queue depth
0
0
3
3

Note that even though we didn't put any messages to TEST.COPY.QUEUE it has the same number of messages on it as TEST.BASE.QUEUE. The stream queue feature has taken a copy of each message we put to TEST.BASE.QUEUE and streamed it to TEST.COPY.QUEUE.

8. What happens if there are problems streaming the messages? We'll simulate this by restricting the MAXDEPTH of TEST.COPY.QUEUE to a very low number. Use the command below to reduce the MAXDEPTH of TEST.COPY.QUEUE to a very low number (we'll use 5 in the lab which means we can put a few more messages and then it will be full):

TEST.COPY.QUEUE - Properties

General
Extended
Cluster
Triggering
Events
Storage
Statistics

Extended

Max queue depth: 5

Maximum message length (bytes): 4194304

Shareability: Shareable

Default input open option: Input shared

Message delivery sequence: Priority

Retention interval (hours): 99999999

Definition type: Predefined

Distribution lists: Not Supported

Default read ahead: No

Default put response type: Asynchronous

Property control: Compatibility

Custom:

Cluster channel names:

Queue media image recoverability: As Queue Manager

Apply

OK Cancel

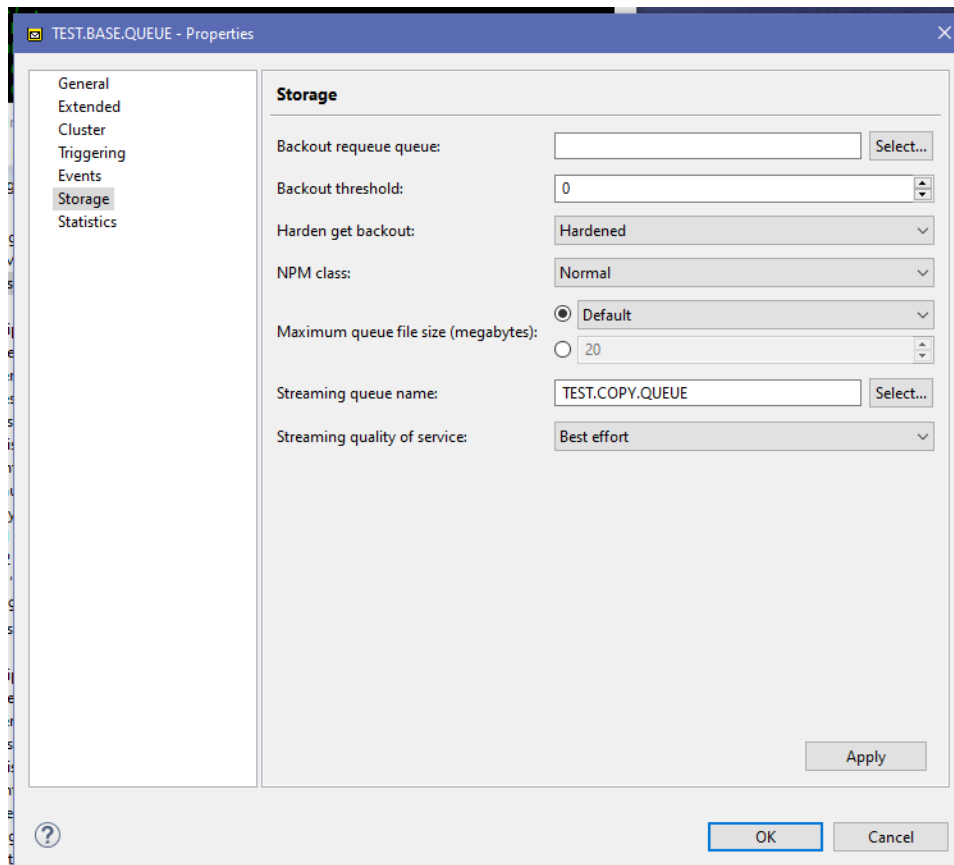
9. Having reduced the MAXDEPTH of TEST.COPY.QUEUE we are going to put some more messages to TEST.BASE.QUEUE and see how the stream queue feature behaves. Put 3 more test messages on the TEST.BASE.QUEUE using the same procedure we used above to PUT messages on the queue.

10. Note how we don't see any errors and MQ continues to allow us to put messages to TEST.BASE.QUEUE. This is due to the default quality-of-service that stream queues are configured to use.

Queue name	Queue type	✓	Open input count	Open output count	Current queue depth
APP.QUEUE	Local	0	0	0	0
MY.LOG.QUEUE	Local	0	0	0	0
TEST.BASE.QUEUE	Local	0	0	0	6
TEST.COPY.QUEUE	Local	0	0	0	5

TEST.BASE.QUEUE has 6 messages on it but TEST.COPY.QUEUE only has 5. MQ could not stream all of the new messages to TEST.COPY.QUEUE because it had reached its MAXDEPTH.

11. Look at the stream queue quality-of-service that is configured:



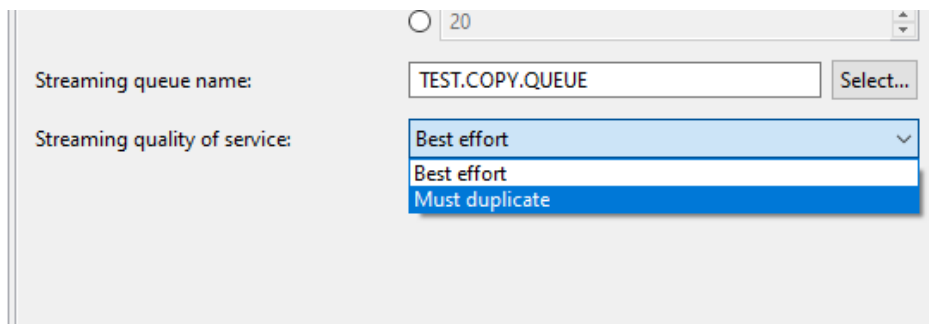
Note that the 'Streaming quality of service' attribute is set to best effort. The best effort quality of service is the default value and indicates that MQ should attempt to stream the duplicate

messages to the stream queue, but that it should deliver the original message to TEST.BASE.QUEUE even if there is a problem delivering the duplicate. This quality-of-service is best suited to applications where you want to make sure that the original behavior of the application is not affected by the stream queue feature. In this mode a putting application will never receive an error from the MQPUT API call due to an error with the streaming feature.

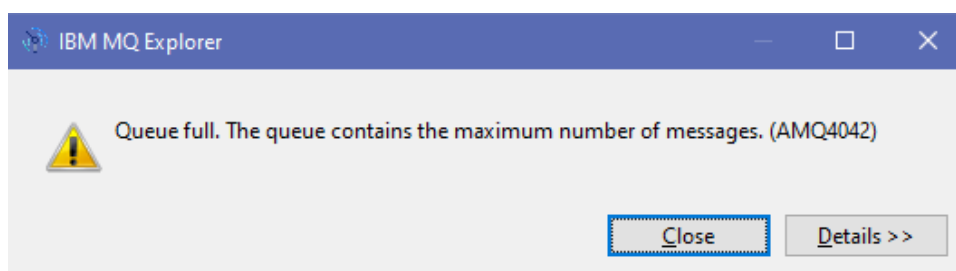
Stream queues offer a second quality-of-service, must duplicate. When 'must duplicate' is used, MQ will either deliver both messages, or neither of them. If there is a problem delivering the duplicate to the stream queue then the original message will not be delivered to the application queue and the putting application will receive an error reason code.

Let's change our APP.QUEUE to use the MUSTDUP quality of service and see what effect that has.

12. Change TEST.BASE.QUEUE to have 'Must duplicate'. Apply those changes to the queue.



13. Now we will see what happens when we try to put more messages to TEST.BASE.QUEUE. Remember that TEST.COPY.QUEUE is still full. Try putting another message on TEST.BASE.QUEUE.



Note: if there was a problem with TEST.BASE.QUEUE then the application would receive that error instead and MQ would not attempt to deliver the message to the stream queue.

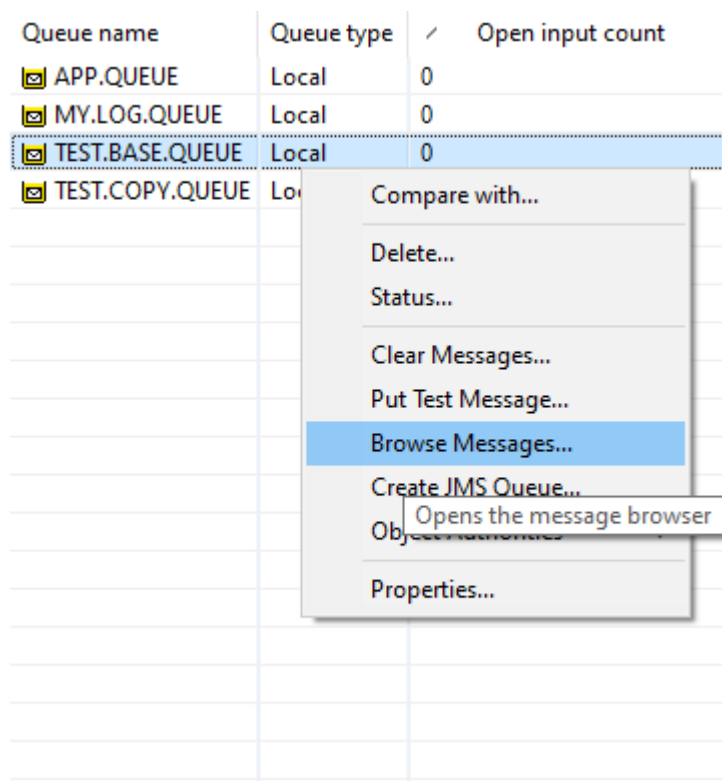
14. We'll now change the setting on TEST.COPY.QUEUE back to max depth of 5000 so it has plenty of space for additional messages.

Then we will try putting 5 more messages to TEST.BASE.QUEUE. This time we shouldn't see any errors because MQ is able to stream duplicates to TEST.COPY.QUEUE again:

15. If we check the queue depths we should now see 11 messages on TEST.BASE.QUEUE, and 10 on TEST.COPY.QUEUE (because 1 of them couldn't be delivered when the setting was 'Must duplicate'):

16. Lastly, browse the messages on first TEST.BASE.QUEUE, then TEST.COPY.QUEUE. How similar are the original and the streamed messages?

Queue name	Queue type	✓	Open input count
APP.QUEUE	Local	0	
MY.LOG.QUEUE	Local	0	
TEST.BASE.QUEUE	Local	0	
TEST.COPY.QUEUE	Local		



Compare with...

Delete...

Status...

Clear Messages...

Put Test Message...

Browse Messages...

Create JMS Queue...

Opens the message browser

Object Navigator

Properties...

They are the same!