Sample JMeter Test Plan for MQ JMS.

Load testing MQ solutions can be performed using a variety of tools. We recommend PerfHarness for JMS and MQ-CPH for native applications. These are lightweight command line tools providing a host of options to control the load presented to the queue manager.

Both are available for download from GitHub as stand-alone or containerised tools:

PerfHarness - https://github.com/ot4i/perf-harness
MQ-CPH - https://github.com/ibm-messaging/mq-cph

PerfHarness (container version) - https://github.com/ibm-messaging/jmstestp
MQ-CPH (container version) - https://github.com/ibm-messaging/cphtestp

Another solution is JMeter, but care must be taken to configure your JMeter test plan correctly, to ensure optimal performance. In particular, *each client thread must create its own session objec*t (which encapsulates a thread-scope connection to the queue manager) - see https://docs.oracle.com/javaee/7/api/javax/jms/Session.html.

Ideally producers and consumers should also be re-used to avoid opening and closing destinations on every iteration.

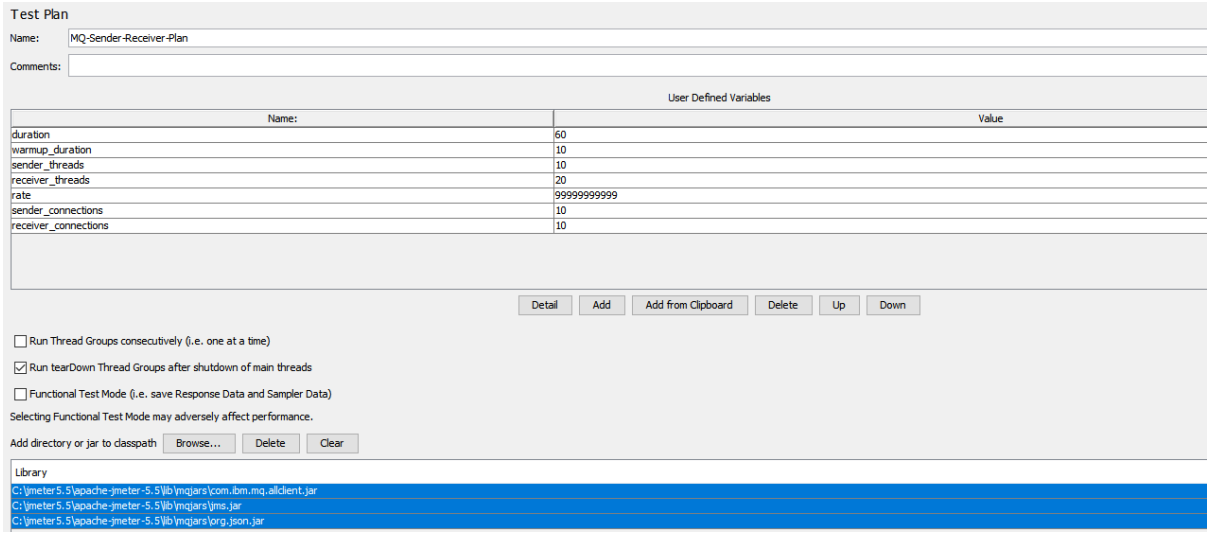
A sample JMeter test plan for MQ (mqperf-sender-receiver.jmx) is available for download which you can use as a starting point in developing your own.

This is a point-to-point test which runs senders (producers) and/or receivers (consumers). It's recommended that both senders and receivers are run in parallel to avoid queue build-up, unless a specific, exception case is being simulated.

How Do I Run This Now?

Assuming you have installed JMeter already, then you can run an initial test in the GUI by …

1. Adding com.ibm.mq.allClient.jar, jms.jar and org.json.jar to the classpath in the main test plan element. These jars are included in the IBM MQ classes for JMS (see https://www.ibm.com/docs/en/ibm-mq/9.3?topic=umcjm-obtaining-mq-classes-jms-mq-classes-jakarta-messaging-separately). There are already entries in this JMeter test plan, but you'll need to alter them to point to the location of those jars on your host. E.g.,



   Alternatively, you can add these to the classpath in the JMeter user.properties file to avoid setting them for every MQ JMS test plan you have.

2. Review and set the MQ connection settings in element 'MQ Connection Settings' The settings are self-documented and include the queue manager name, optional credentials, and host machine name & listener port for client connections.

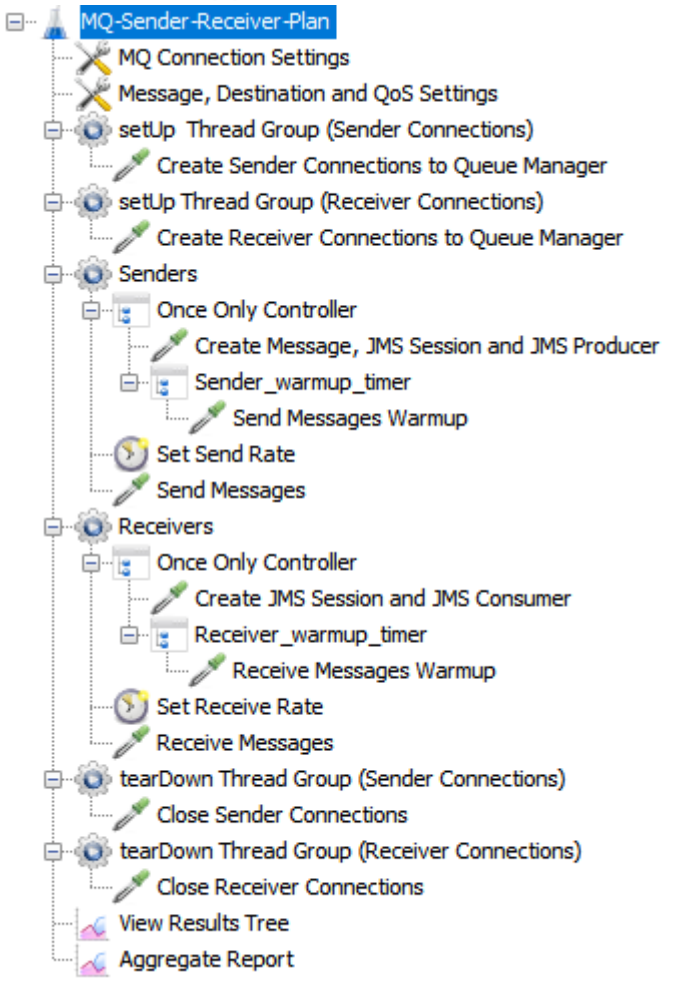
3. Review and set the destination(s), required qos (persistent? transacted?) and message contents in element 'Message, Destination and QoS Settings'.

4. Review the top-level test parameters to specify the duration of the test, the number of threads to execute etc. These settings are described below, if you leave them as-is, the test will run 10 senders and 20 receivers, unrated for 60 seconds, with no warmup.

5. Clear any previous test data, by clicking 'clear all' in JMeter.

6. Click the Start button and review results appearing in the 'Aggregate Report' listener. Note that this approach is only for testing – run the tests from the command line (see below) for final metrics. If there are JMS errors, they will appear in the 'View Results Tree' listener element.

What Just Happened When I Ran That Test?



The figure above shows the fully exploded MQ-Sender-Receiver-Plan test plan. Assuming you ran with the default settings on the test plan itself. Here's what just happened:

1. Test, connection, and other JMS settings were picked up from the test element itself and the two User Defined Variables elements ("MQ Connection Settings" and "Message, Destination & QoS Settings")
2. Thread groups are now run. The main test element specifies that regular test groups ("Senders" & "Receivers") are run at the same time, but four thread groups are special JMeter thread groups which are run at the start of the test (the "setUp Thread Groups") and at the end (the "tearDown Thread Groups"). The test will execute the two "setUp Thread Groups" then loop around, executing "Senders" and "Receivers" in parallel for the time set by the 'duration' setting then execute the two "tearDown Thread Groups". The two defined listeners 'View Results Tree' and 'Aggregate Report' will start to show any results.

    a. SetupThread Group (Sender Connections)
    The first setupThread Group is run for multiple threads (defined by the 'sender_connections' setting in the main test plan element) to create the

JMS Connection objects to be shared by all the sender threads in the test in a round robin fashion.

b. SetupThread Group (Receiver Connections)
   The second setupThread Group is run for multiple threads (defined by the 'receiver_connections' setting in the main test plan element) to create the JMS Connection objects to be shared by all the receiver threads in the test in a round robin fashion.
c. Senders
   This thread groups runs in parallel with the 'Receivers' thread group. A once-only block is run first which creates a session, destination, and message object for each thread in the group (the number of threads is controlled by the 'sender_threads' setting). Following this, a warmup element 'Send Messages Warmup' is run, sending messages to MQ unrated for the duration specified by 'warmup_duration'.
   After the once-only block has been run, the thread group loops around the 'Senders' element for 'duration' – 'warmup-duration' sending additional messages to MQ. i.e., if duration=60 and warmup-duration=10, then the 'Senders' element is looped round for 50 seconds.
d. Receivers
   This thread groups runs in parallel with the 'Senders' thread group. Its structure is very similar to the 'Senders' thread group except that the warmup and main loop are receiving messages instead of sending them, no message object needs to be created and the number of threads in the group is controlled by the setting 'receiver_threads'
e. tearDown Thread Group (Sender Connections)
   Closes all connections created in the SetupThread Group (Sender Connections). This will also cause all associated sessions to be closed.
f. tearDown Thread Group (Receiver Connections)
   Closes all connections created in the SetupThread Group (Receiver Connections). This will also cause all associated sessions to be closed.

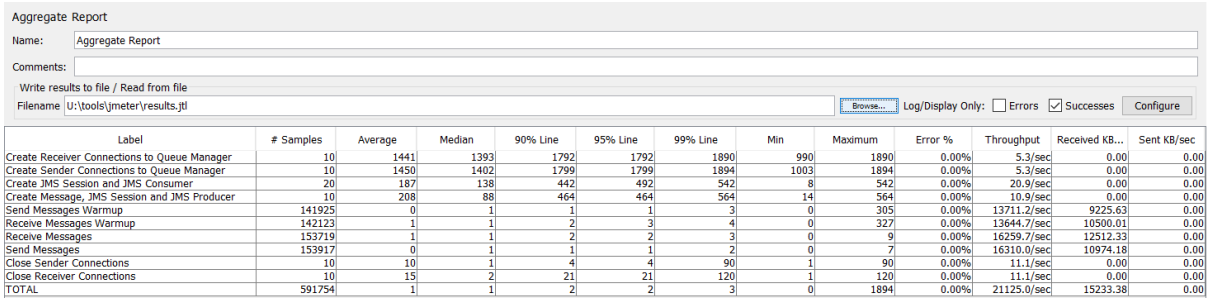
Once the test completes (controlled by the 'duration' setting) you'll see something like this in the aggregate results listener:

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received KB...	Sent KB/sec
Create Sender Connections to Queue Manager	10	37	16	82	82	169	12	169	0.00%	10.9/sec	0.00	0.00
Create Receiver Connections to Queue Manager	10	35	14	66	66	171	10	171	0.00%	11.0/sec	0.00	0.00
Create Message, JMS Session and JMS Producer	10	36	16	73	73	152	13	152	0.00%	11.0/sec	0.00	0.00
Create JMS Session and JMS Consumer	20	34	17	93	94	150	12	150	0.00%	20.5/sec	0.00	0.00
Send Messages Warmup	85050	1	1	1	2	2	0	88	0.00%	7906.5/sec	5319.89	0.00
Receive Messages Warmup	85488	2	2	3	3	5	0	101	0.00%	7894.4/sec	6074.95	0.00
Send Messages	82372	1	1	1	2	2	0	9	0.00%	8353.3/sec	5620.54	0.00
Receive Messages	81934	2	2	3	3	5	0	12	0.00%	8317.3/sec	6400.44	0.00
Close Sender Connections	10	9	5	6	6	51	3	51	0.00%	11.2/sec	0.00	0.00
Close Receiver Connections	10	14	9	13	13	56	8	56	0.00%	11.0/sec	0.00	0.00
TOTAL	334914	1	1	3	3	4	0	171	0.00%	12354.3/sec	8907.99	0.00

For optimal performance you should run the test from a command line. e.g.:

```
jmeter.bat -n -t <test dir>/mqperf-sender-receiver.jmx -l results.jtl
```

You can then load the results file (results.jtl in this case) into a listener like 'aggregate report' or process it with a spreadsheet. There are examples on the internet on how to post-process a results file. Results from a batch invocation are shown below.



Aggregate Report

Name: Aggregate Report

Comments:

Write results to file / Read from file

Filename U:\tools\jmeter\results.jtl Browse... Log/Display Only: Errors Successes Configure

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received KB...	Sent KB/sec
Create Receiver Connections to Queue Manager	10	1441	1393	1792	1792	1890	990	1890	0.00%	5.3/sec	0.00	0.00
Create Sender Connections to Queue Manager	10	1450	1402	1799	1799	1894	1003	1894	0.00%	5.3/sec	0.00	0.00
Create JMS Session and JMS Consumer	20	187	138	442	492	542	8	542	0.00%	20.9/sec	0.00	0.00
Create Message, JMS Session and JMS Producer	10	208	88	464	464	564	14	564	0.00%	10.9/sec	0.00	0.00
Send Messages Warmup	141925	0	1	1	1	3	0	305	0.00%	13711.2/sec	9225.63	0.00
Receive Messages Warmup	142123	1	1	2	3	4	0	327	0.00%	13644.7/sec	10500.01	0.00
Receive Messages	153719	1	1	2	2	3	0	9	0.00%	16259.7/sec	12512.33	0.00
Send Messages	153917	0	1	1	1	2	0	7	0.00%	16310.0/sec	10974.18	0.00
Close Sender Connections	10	10	1	4	4	90	1	90	0.00%	11.1/sec	0.00	0.00
Close Receiver Connections	10	15	2	21	21	120	1	120	0.00%	11.1/sec	0.00	0.00
TOTAL	591754	1	1	2	2	3	0	1894	0.00%	21125.0/sec	15233.38	0.00

Note that the initial connect time to MQ in this case is higher than for the GUI. I found that running multiple tests in the GUI meant that initial one-off actions like this became cheaper as, the jvm was more ‘warmed up’. If the same test was run straight after starting the GUI, the connect time was then longer than the batch invocation. The main loop should always be faster in a batch invocation.

Make sure you understand the performance behaviour of JMeter and that results are run in the most optimal fashion and are repeatable. A significant drawback of the ‘aggregate listener’ for instance, is that any dips in the performance during a run won’t be easily seen. These are immediately visible running a tool such as PerfHarness where rates for controllable intervals are continuously reported. Post-processing the results file is probably a better option here, but beyond the scope of this article.

The Test Plan provided here should suffice as a starting point but modify it to be specific to you own needs. You may want each thread to have its own initial connection to MQ, for instance, to emulate standalone applications.

JMS Connections and Sessions

Each JMS Connection and Session configured in the test is effectively a separate MQ connection. In a simple JMS application, a lightweight JMS Connection object is created which establishes an initial connection to MQ (including any required authentication). The JMS Connection may then be used to create a JMS Session which is used to do the real work (messages are sent and/or received via the MQ connection represented by the JMS Session). It’s possible to create more than one JMS Session per JMS Connection.

From an MQ perspective the number of connections for an application is the sum of the JMS Connection plus its associated JMS Sessions.

A JMS Connection can be shared between threads, but a JMS Session is single threaded, so every thread most have exclusive access to one or more sessions when executing JMS methods on those sessions.

In this JMeter test plan, each thread owns its own JMS session object for the duration of the test, but a smaller number of JMS Connections can be specified so that you could configure 2 sessions per connection. In an application manager there would be connection and session pools and some environments may impose a constraint of one session per connection.

Consider the application you are simulating when setting the number of sessions per connection. When in doubt set the number of connections to the same value as the sessions for a 1:1 relationship (so there will be 2 MQ connections per application on the queue manager).

The number of sockets used by the application will depend on the SHRCNV setting of the queue manager

SHRCNV: https://www.ibm.com/docs/en/ibm-mq/9.3?topic=application-sharing-tcpip-connection-in-mq-classes-jms

Each JMS connection and session is effectively a separate MQ connection. So a simple JMS app with a connection+session will usually show up as 2 MQCONN handles. MQCONNs then use the channel SHRCONV value to determine how many connections are established on the same TCP socket. Though the best value for SHRCONV is usually 1, so each MQCONN = 1 socket.

Settings for the Sample Test Plan

There are three main configuration elements in the plan:

MQ-Send-Receiver-Plan

High level settings for duration, number of threads etc. Namely:

duration	The number of seconds to run the main Send Messages and/or Receive Messages loop(s).
warmup_duration	The number of seconds to run the warmup loop(s). Set this to 0 for no warmup, otherwise the warmup elements are run for this number of seconds before moving onto the main Send Messages and/or Receive Messages loop(s).
sender_threads	Number of Senders to instantiate (one JMS session is created per thread).
receiver_threads	Number of Receivers to instantiate (one JMS session is created per thread) – typically set this higher than the number of Senders to ensure there are always waiting Receivers.
rate	Set the target msg/sec rate for the main Send Messages and/or Receive Messages loop(s), (or set high for unrated).
sender_connections	Number of JMS connections to be created for the senders. JMS Sessions (one per sender thread) will be created from these connections, using them in a round-robin fashion.
receiver_connections	Number of JMS connections to be created for the receivers. JMS Sessions (one per receiver thread) will be created from these connections, using them in a round-robin fashion.

MQ Connection Settings

These setting specify how you connect to an MQ Queue Manager.

qm_name	Queue manager Name
qm_channel	Server connection channel to connect to
qm_conn_mode	Connection Mode (client | bindings*)
qm_host	Queue manager host (only required for client mode)
qm_port	Queue manager listener port (only required for client mode)
qm_userid	Userid (will attempt to connect without credentials if not specified)
qm_pwd	Password (used if userid is set)

*If you specify bindings mode connection, the jmeter test must be run on the same host as the queue manager and the MQ supplied mqjbnd library must be in the java native library path.

E.g., on Linux this could be specified by executing the following commands before running the test:

```
export LD_LIBRARY_PATH=$MQ_INSTALLATION_PATH/java/lib64:$LD_LIBRARY_PATH
```

$MQ_INSTALLATION_PATH/java/lib64 would typically resolve to /opt/mqm/java/lib64

For more details see: https://www.ibm.com/docs/en/ibm-mq/9.3?topic=jms-configuring-java-native-interface-jni-libraries

Message, Destination and QoS Settings

These setting are used to create the thread specific JMS objects such as the JMS Session, Destination and Message objects. You also specify whether the messages are persistent and whether transactions are used here.

q_name	Queue name (used as-is, if q_range is set to 1)
q_range	Queue range (if >1 then queues ${q_name}1 to ${qname}${q_range} are used round-robin by threads)
persistent	Set to true for persistent messages (true|false)
transacted	Set to true for transacted sends/receives. This should be set to the same value as 'persistent' above (true|false)
msg_len	Length of message to be generated (only used if msg_file is not set below)
msg_file	File containing data to populate messages with (e.g. /tmp/testMsg.dat). Paths relative to JMeter bin directory can be used. For windows use / or \\ (i.e. c:/testfile.txt or c:\\testfile.txt)
msg_type	Send message as String object (text) or stream of uninterpreted bytes (binary) (text|binary)
receive_timeout	Timeout for consumer.receive call (specified in milliseconds)