Java Tools

This page documents some Java-based utility programs (PerfTest, Tracer).

**[PerfTest](http://next.rabbitmq.com/java-tools.html" \l "throughput-load-testing)**

RabbitMQ has a basic throughput testing tool, PerfTest (docs, source code and releases), that is based on the Java client and can be configured to simulate basic workloads. PerfTest has extra tools that produce HTML graphs of the output. A RabbitMQ cluster can be limited by a number of factors, from infrastructure-level constraints (e.g. network bandwidth) to RabbitMQ configuration and topology to applications that publish and consume. PerfTest can demonstrate baseline performance of a node or a cluster of nodes.

**[Installation](http://next.rabbitmq.com/java-tools.html" \l "perf-test-installation)**

PerfTest is distributed as a binary build archive from [Bintray](https://bintray.com/rabbitmq/java-tools/perf-test) and [GitHub releases](https://github.com/rabbitmq/rabbitmq-perf-test/releases) as well. It is also available on [Maven Central](http://search.maven.org/#search%7Cga%7C1%7Cg%3A%22com.rabbitmq%22%20AND%20a%3A%22perf-test%22) if one needs to use it as library. Note the documentation below can cover features available only in milestone releases or release candidates, available in our[Java Tools milestones repository](https://bintray.com/rabbitmq/java-tools-milestones/perf-test) and in [GitHub releases](https://github.com/rabbitmq/rabbitmq-perf-test/releases) as well.

The distribution contains a script (bin/runjava or bin/runjava.bat) to run Java with the class path correctly configured, e.g. bin/runjava com.rabbitmq.perf.PerfTest runs the PerfTest Java class.

To verify a PerfTest installation, use

bin/runjava com.rabbitmq.perf.PerfTest --help

**[Using PerfTest](http://next.rabbitmq.com/java-tools.html" \l "using-perf-test)**

The most basic way of running PerfTest only specifies a URI to connect to, a number of publishers to use (say, 1) and a number of consumers to use (say, 2). Note that RabbitMQ Java client can achieve high rates for publishing (up to 80 to 90K messages per second per connection), given enough bandwidth and when some safety measures (publisher confirms) are disabled, so overprovisioning publishers is rarely necessary (unless that's a specific objective of the test). The following command runs PerfTest with a single publisher without publisher confirms, two consumers (each receiving a copy of every message) that use automatic acknowledgement mode and a single queue named “throughput-test-x1-y2”. Publishers will publish as quickly as possible, without any rate limiting. Results will be prefixed with “test 1” for easier identification and comparison:

bin/runjava com.rabbitmq.perf.PerfTest -x 1 -y 2 -u "throughput-test-1" -a --id "test 1"

This modification will use 2 publishers and 4 consumers, typically yielding higher throughput given enough CPU cores on the machine and RabbitMQ nodes:

bin/runjava com.rabbitmq.perf.PerfTest -x 2 -y 4 -u "throughput-test-2" -a --id "test 2"

This modification switches consumers to manual acknowledgements:

bin/runjava com.rabbitmq.perf.PerfTest -x 1 -y 2 -u "throughput-test-3" --id "test 3"

This modification changes message size from default (12 bytes) to 4 kB:

bin/runjava com.rabbitmq.perf.PerfTest -x 1 -y 2 -u "throughput-test-4" --id "test 4" -s 4000

PerfTest can use durable queues and persistent messages:

bin/runjava com.rabbitmq.perf.PerfTest -x 1 -y 2 -u "throughput-test-5" --id "test-5" -f persistent

When PerfTest is running, it is important to monitor various publisher and consumer metrics provided by the [management UI](http://next.rabbitmq.com/management.html). For example, it is possible to see how much network bandwidth a publisher has been using recently on the connection page.

Queue page demonstrates message rates, consumer count, acknowledgement mode used by the consumers, consumer utilisation and message location break down (disk, RAM, paged out transient messages, etc). When durable queues and persistent messages are used, node I/O and message store/queue index operation metrics become particularly important to monitor.

Consumers can ack multiple messages at once, for example, 100 in this configuration:

bin/runjava com.rabbitmq.perf.PerfTest -x 1 -y 2 -u "throughput-test-6" --id "test-6" -f persistent --multi-ack-every 100

[Consumer prefetch (QoS)](http://next.rabbitmq.com/confirms.html) can be configured as well (in this example to 500):

bin/runjava com.rabbitmq.perf.PerfTest -x 1 -y 2 -u "throughput-test-7" --id "test-7" -f persistent --multi-ack-every 200 -q 500

Publisher confirms can be used with maximum of N outstanding publishes:

bin/runjava com.rabbitmq.perf.PerfTest -x 1 -y 2 -u "throughput-test-8" --id "test-8" -f persistent -q 500 -c 500

PerfTest can publish only a certain number of messages instead of running until shut down:

bin/runjava com.rabbitmq.perf.PerfTest -x 1 -y 2 -u "throughput-test-10" --id "test-10" -f persistent -q 500 -pmessages 100000

Publisher rate can be limited:

bin/runjava com.rabbitmq.perf.PerfTest -x 1 -y 2 -u "throughput-test-11" --id "test-11" -f persistent -q 500 --rate 5000

Consumer rate can be limited as well to simulate slower consumers or create a backlog:

bin/runjava com.rabbitmq.perf.PerfTest -x 1 -y 2 -u "throughput-test-12" --id "test-12" -f persistent --rate 5000 --consumer-rate 2000

Note that the consumer rate limit is applied per consumer, so in the configuration above the limit is actually 2 \* 2000 = 4000 deliveries/second.

PerfTest can be configured to run for a limited amount of time in seconds with the -z option:

bin/runjava com.rabbitmq.perf.PerfTest -x 1 -y 2 -u "throughput-test-13" --id "test-13" -f persistent -z 30

Running PerfTest without consumers and with a limited number of messages can be used to pre-populate a queue, e.g. with 1M messages 1 kB in size each::

bin/runjava com.rabbitmq.perf.PerfTest -y0 -p -u "throughput-test-14" -s 1000 -C 1000000 --id "test-14" -f persistent

Use the -D option to limit the number of consumed messages. Note the -z (time limit), -C(number of published messages), and -D (number of consumed messages) options can be used together but their combination can lead to funny results. -r 1 -x 1 -C 10 -y 1 -D 20 would for example stop the producer once 10 messages have been published, letting the consumer wait forever the remaining 10 messages (as the publisher is stopped).

To consume from a pre-declared and pre-populated queue without starting any publishers, use

bin/runjava com.rabbitmq.perf.PerfTest -x0 -y10 -p -u "throughput-test-14" --id "test-15"

PerfTest is useful for establishing baseline cluster throughput with various configurations but does not simulate many other aspects of real world applications. It is also biased towards very simplistic workloads that use a single queue, which provides [limited CPU utilisation](http://next.rabbitmq.com/queues.html) on RabbitMQ nodes and is not recommended for most cases. Multiple PerfTest instances running simultaneously can be used to simulate more realistic workloads.

**[Customising queues and messages](http://next.rabbitmq.com/java-tools.html" \l "customising-queues-and-messages)**

PerfTest can create queues using provided queue arguments:

bin/runjava com.rabbitmq.perf.PerfTest --queue-arguments x-max-length=10

The previous command will create a [queue with a length limit](http://next.rabbitmq.com/maxlength.html) of 10. You can also provide several queue arguments by separating the key/value pairs with commas:

bin/runjava com.rabbitmq.perf.PerfTest --queue-arguments x-max-length=10,x-dead-letter-exchange=some.exchange.name

You can also specify message properties with key/value pairs separated by commas:

bin/runjava com.rabbitmq.perf.PerfTest --message-properties priority=5,timestamp=2007-12-03T10:15:30+01:00

The supported property keys are: contentType, contentEncoding, deliveryMode, priority, correlationId, replyTo, expiration, messageId, timestamp, type, userId, appId, clusterId. If some provided keys do not belong to the previous list, the pairs will be considered as headers (arbitrary key/value pairs):

bin/runjava com.rabbitmq.perf.PerfTest --message-properties priority=10,header1=value1,header2=value2

You can mimic real messages by specifying their content and content type. This can be useful when plugging real application consumers downstream. The content can come from one or several files and the content-type can be specified:

bin/runjava com.rabbitmq.perf.PerfTest --consumers 0 \

--body content1.json,content2.json --body-content-type application/json

**[Working with many queues](http://next.rabbitmq.com/java-tools.html" \l "working-with-many-queues)**

PertTest supports balancing the publishing and the consumption across a sequence of queues, e.g.:

bin/runjava com.rabbitmq.perf.PerfTest --queue-pattern 'perf-test-%d' \

--queue-pattern-from 1 --queue-pattern-to 10 \

--producers 100 --consumers 100

The previous command would create the perf-test-1, perf-test-2, ..., perf-test-10 queues and spreads the producers and consumers across them. This way each queue will have 10 consumers and 10 producers sending messages to it.

Load is balanced in a round-robin fashion:

bin/runjava com.rabbitmq.perf.PerfTest --queue-pattern 'perf-test-%d' \

--queue-pattern-from 1 --queue-pattern-to 10 \

--producers 15 --consumers 30

With the previous command, queues from perf-test-1 to perf-test-5 will have 2 producers, and queues from perf-test-6 to perf-test-10 will have only 1 producer. Each queue will have 3 consumers.

Note the --queue-pattern value is a [Java printf-style format string](https://docs.oracle.com/javase/7/docs/api/java/util/Formatter.html). The queue index is the only argument passed in. The formatting is very closed to C's printf. --queue-pattern 'perf-test-%03d' --queue-pattern-from 1 --queue-pattern-to 500 would for instance create queues from perf-test-001 to perf-test-500.

**[Simulating High Loads](http://next.rabbitmq.com/java-tools.html" \l "simulating-high-loads)**

PerfTest can easily run hundreds of connections on a simple desktop machine. Each producer and consumer use a Java thread and a TCP connection though, so a PerfTest process can quickly run out of file descriptors, depending on the OS settings. A simple solution is to use several PerfTest processes, on the same machine or not. This is especially handy when combined with the [queue sequence](http://next.rabbitmq.com/java-tools.html#working-with-many-queues) feature:

*# This first PerfTest process creates 500 queues (from 'perf-test-1' to 'perf-test-500').*

*# Each queue will have 3 consumers and 1 producer sending messages to it.*

bin/runjava com.rabbitmq.perf.PerfTest --queue-pattern 'perf-test-%d' \

--queue-pattern-from 1 --queue-pattern-to 500 \

--producers 500 --consumers 1500

*# This second PerfTest process creates 500 queues (from 'perf-test-501' to 'perf-test-1000').*

*# Each queue will have 3 consumers and 1 producer sending messages to it.*

bin/runjava com.rabbitmq.perf.PerfTest --queue-pattern 'perf-test-%d' \

--queue-pattern-from 501 --queue-pattern-to 1000 \

--producers 500 --consumers 1500

Those 2 processes will simulate 1000 producers and 3000 consumers spread across 1000 queues.

A PerfTest process can exhaust its file descriptors limit and throw java.lang.OutOfMemoryError: unable to create new native thread exceptions. A first way to avoid this is to reduce the number of Java threads PerfTest uses with the --heartbeat-sender-threads option:

bin/runjava com.rabbitmq.perf.PerfTest --queue-pattern 'perf-test-%d' \

--queue-pattern-from 1 --queue-pattern-to 1000 \

--producers 1000 --consumers 3000 --heartbeat-sender-threads 10

By default, each producer and consumer connection uses a dedicated thread to send heartbeats to the broker, so this is 4000 threads for heartbeats in the previous sample. Considering producers and consumers always communicate with the broker by publishing messages or sending acknowledgments, connections are never idle, so using 10 threads for heartbeats for the 4000 connections should be enough. Don't hesitate to experiment to come up with the appropriate --heartbeat-sender-threads value for your use case.

Another way to avoid java.lang.OutOfMemoryError: unable to create new native threadexceptions is to tune the number of file descriptors allowed per process at the OS level, as some distributions use very low limits. Here the recommendations are the same as for the broker, so you can refer to our [networking guide](http://next.rabbitmq.com/networking.html#os-tuning).

**[Simulating Workloads with a Large Number of Clients ("IoT" Workloads)](http://next.rabbitmq.com/java-tools.html" \l "iot-workloads)**

A typical connected device workload involves many producers and consumers (dozens or hundreds of thousands) that exchange messages at a low and mostly constant rate, usually a message every few seconds or minutes. Simulating such workloads requires a different set of settings compared to the workloads that have higher throughput and a small number of clients. With the appropriate set of flags, PerfTest can simulate IoT workloads without requiring too many resources.

With an IoT workload, publishers usually don't publish many messages per second, but rather a message every fixed period of time. This can be achieved by using the --publishing-intervalflag instead of the --rate one. For example:

bin/runjava com.rabbitmq.perf.PerfTest --publishing-interval 5

The command above makes the publisher publish a message every 5 seconds. To simulate a group of consumers, use the--queue-pattern flag to simulate many consumers across many queues:

bin/runjava com.rabbitmq.perf.PerfTest --queue-pattern 'perf-test-%d' \

--queue-pattern-from 1 --queue-pattern-to 1000 \

--producers 1000 --consumers 1000 \

--heartbeat-sender-threads 10 \

--publishing-interval 5

To prevent publishers from publishing at roughly the same time and distribute the rate more evenly, use the --producer-random-start-delay option to add an random delay before the first published message:

bin/runjava com.rabbitmq.perf.PerfTest --queue-pattern 'perf-test-%d' \

--queue-pattern-from 1 --queue-pattern-to 1000 \

--producers 1000 --consumers 1000 \

--heartbeat-sender-threads 10 \

--publishing-interval 5 --producer-random-start-delay 120

With the command above, each publisher will start with a random delay between 1 and 120 seconds.

When using --publishing-interval, PerfTest will use one thread for scheduling publishing for all 50 producers. So 1000 producers should keep 20 threads busy for the publishing scheduling. This ratio can be decreased or increased with the --producer-scheduler-threadsoptions depending on the load and the target environment. Very few threads can be used for very slow publishers:

bin/runjava com.rabbitmq.perf.PerfTest --queue-pattern 'perf-test-%d' \

--queue-pattern-from 1 --queue-pattern-to 1000 \

--producers 1000 --consumers 1000 \

--heartbeat-sender-threads 10 \

--publishing-interval 60 --producer-random-start-delay 1800 \

--producer-scheduler-threads 10

In the example above, 1000 publishers will publish every 60 seconds with a random start-up delay between 1 second and 15 minutes (1800 seconds). They will be scheduled by only 10 threads (instead of 20 by default). Such delay values are suitable for long running tests.

By default, PerfTest uses blocking network socket I/O to communicate with the broker. This mode works fine for clients in many cases but the RabbitMQ Java client also supports an [asynchronous I/O mode](http://next.rabbitmq.com/api-guide.html#java-nio), where resources like threads can be easily tuned. The goal here is to use as few resources as possible to simulate as much load as possible with a single PerfTest instance. In the slow publisher example above, a handful of threads should be enough to handle the I/O. That's what the --nio-threads flag is for:

bin/runjava com.rabbitmq.perf.PerfTest --queue-pattern 'perf-test-%d' \

--queue-pattern-from 1 --queue-pattern-to 1000 \

--producers 1000 --consumers 1000 \

--heartbeat-sender-threads 10 \

--publishing-interval 60 --producer-random-start-delay 1800 \

--producer-scheduler-threads 10

--nio-threads 10

PerfTest will use 2 extra threads for internal bookkeeping needs, so the total will be 12 threads for I/O over all the connections. With the default blocking I/O mode, each producer (or consumer) uses a thread for the I/O loop, that is 2000 threads to simulate 1000 producers and 1000 consumers. Using NIO in PerfTest can dramatically reduce the resources used to simulate workloads with a large number of connections with appropriate tuning.

**[Running Producers and Consumers on Different Machines](http://next.rabbitmq.com/java-tools.html" \l "perftest-different-machines)**

If you run producers and consumers on different machines or even in different processes, and you want PerfTest to calculate latency, you need to use the --use-millis flag. E.g. for sending messages from one host:

bin/runjava com.rabbitmq.perf.PerfTest --producers 1 --consumers 0 --predeclared --routing-key rk --queue q --use-millis

And for consuming messages from another host:

bin/runjava com.rabbitmq.perf.PerfTest --producers 0 --consumers 1 --predeclared --routing-key rk --queue q --use-millis

Note that as soon as you use --use-millis, latency is calculated in milliseconds instead of microseconds. Note also the different machines should have their clock synchronised, e.g. by NTP. If you don't run producers and consumers on different machines or if you don't want PerfTest to calculate latency, you don't need the --use-millis flag.

Why does one need to care about the --use-millis flag? PerfTest uses by default System.nanoTime() in messages to calculate latency between producers and senders. System.nanoTime() provides nanosecond precision but must be used only in the same Java process. So PerfTest can fall back to System.currentTimeMillis(), which provides only milliseconds precision, but is reliable between different machines as long as their clocks are synchronised.

**[How It Works](http://next.rabbitmq.com/java-tools.html" \l "how-perftest-works)**

If a queue name is defined (-u "queue-name"), PerfTest will create a queue with this name and all consumers will consume from this queue. The queue will be bound to the direct exchange with its name as the routing key. The routing key will be used by producers to send messages. This will cause messages from all producers to be sent to this single queue and all consumers to receive messages from this single queue.

If the queue name is not defined, PerfTest will create a random UUID routing key with which producers will publish messages. Each consumer will create its own anonymous queue and bind it to the direct exchange with this routing key. This will cause each message from all producers to be replicated to multiple queues (number of queues equals number of consumers), while each consumer will be receiving messages from only one queue.

**[TLS Support](http://next.rabbitmq.com/java-tools.html" \l "perf-test-tls)**

PerfTest can use TLS to connect to a node that is [configured to accept TLS connections](http://next.rabbitmq.com/ssl.html). To enable TLS, simply specify a URI that uses the amqps schema:

bin/runjava com.rabbitmq.perf.PerfTest -h amqps://localhost:5671

By default PerfTest automatically trusts the server and doesn't present any client certificate (a warning shows up in the console). In many benchmarking or load testing scenarios this may be sufficient. If peer verification is necessary, it is possible to use the [appropriate JVM properties](https://docs.oracle.com/javase/8/docs/technotes/guides/security/jsse/JSSERefGuide.html#InstallationAndCustomization) on the command line to override the default SSLContext. For example, to trust a given server:

JAVA\_OPTS="-Djavax.net.ssl.trustStore=/path/to/server\_key.p12 -Djavax.net.ssl.trustStorePassword=bunnies -Djavax.net.ssl.trustStoreType=PKCS12" \

bin/runjava com.rabbitmq.perf.PerfTest -h amqps://localhost:5671

The previous snippet uses a one-liner to define the JAVA\_OPTS environment variable while running PerfTest. Please refer to the [TLS guide](http://next.rabbitmq.com/ssl.html) to learn about how to set up RabbitMQ with TLS. A convenient way to generate a CA and some self-signed certificate/key pairs for development and QA environments is with [tls-gen](https://github.com/michaelklishin/tls-gen/). tls-gen's basic profile is a good starting point. How to run PerfTest with a certificate/key pair generated by the aforementioned profile:

JAVA\_OPTS="-Djavax.net.ssl.trustStore=/path/to/server\_key.p12 -Djavax.net.ssl.trustStorePassword=bunnies -Djavax.net.ssl.trustStoreType=PKCS12 -Djavax.net.ssl.keyStore=/path/to/client\_key.p12 -Djavax.net.ssl.keyStorePassword=bunnies -Djavax.net.ssl.keyStoreType=PKCS12" \

bin/runjava com.rabbitmq.perf.PerfTest -h amqps://localhost:5671

**[Result Reporting in HTML](http://next.rabbitmq.com/java-tools.html" \l "perf-html)**

The PerfTest HTML extension are a set of tools that can help you run automated benchmarks by wrapping around the PerfTest benchmarking framework. You can provide benchmark specs, and the tool will take care of running the benchmark, collecting results and displaying them in an HTML page. Learn more [here](https://github.com/rabbitmq/rabbitmq-perf-test/blob/master/html/README.md).

**[Tracer](http://next.rabbitmq.com/java-tools.html" \l "tracer)**

The tracer is a very basic, very simple AMQP 0-9-1 protocol analyzer, similar in purpose to [Wireshark](http://next.rabbitmq.com/amqp-wireshark.html). Use it with the runtracer or runtracer.bat script:

runtracer *listenPort* *connectHost* *connectPort*

listenPort

port to listen for incoming AMQP connections on - defaults to 5673.

connectHost

hostname to use when making an outbound connection in response to an incoming connection - defaults to localhost.

connectPort

port number to use when making an outbound connection - defaults to 5672.

**[Download and source code](http://next.rabbitmq.com/java-tools.html" \l "tracer-download)**

Releases: [Bintray](https://bintray.com/rabbitmq/java-tools/tracer) [GitHub releases](https://github.com/rabbitmq/rabbitmq-tracer/releases)

[Source code](https://github.com/rabbitmq/rabbitmq-tracer)