

— GUIDE · RABBITMQ

RabbitMQ Queue Type Decision Worksheet

Answer the questions below once per workload and you get a queue type, a reason you can put in a design review, and the list of behaviours that change if the queue is moving from another type. Checked against rabbitmq.com on 29 October 2026 (RabbitMQ 4.4 documentation).



By **Scott Sternloff**, Senior Enterprise Architect

LinkedIn • Updated October 29, 2026 • 8 min read • 6 sections

— **QUORUM BY DEFAULT, CLASSIC FOR TEMPORARY QUEUES, STREAM FOR REPLAY**

Quorum for any queue whose contents you would be paged about losing. **Classic** for exclusive, server-named and short-lived queues such as RPC replies, and where latency matters more than data safety. **Stream** when consumers need to re-read, when many consumers need the same messages, or when the backlog runs to millions.

- **Classic is not replicated.** Mirroring was removed in 4.0, so "classic with an HA policy" is no longer an option.
- **The type is fixed at declaration.** It cannot be set or changed by policy, so moving a queue to another type means a new queue and a message move.
- **Some behaviour does not carry over.** Exclusivity, global QoS, reject-publish-dlx, unlimited redelivery and TTL all behave differently or not at all on the new type; the migration table below lists each one.

What this worksheet is for

Use this page to decide a queue type per workload and to find what a migration between types will break. It does not explain how quorum queues or streams work internally: the quorum queue article covers Raft, fsync and failover, the streams guide covers retention, offsets and super streams, and the durability article covers what persistence actually guarantees.

The worksheet: answer in order, stop at the first yes

Work down the questions for each workload; the first yes decides the type, and later questions only refine the configuration. If every answer is no, the workload is low-stakes and a classic queue is fine.

#	QUESTION	IF YES	WHY
1	Is the queue tied to one connection, server-named, or meant to vanish when the client goes?	Classic	Quorum queues and streams cannot be exclusive, non-durable or server-named. RPC reply queues live here.
2	Do consumers need to re-read messages or start from a point in time?	Stream	Streams are non-destructive: consumers attach at an offset or timestamp and can replay.
3	Do many independent consumers need the same messages?	Stream	One stream serves any number of consumers, instead of one bound queue per consumer.
4	Will the backlog regularly reach millions of messages?	Stream	The docs name 5M+ backlogs as a case where streams are likely the better fit than quorum queues.
5	Would losing this queue's contents with one node be an incident?	Quorum	Classic queues are single-replica since 4.0; quorum queues survive a minority of members failing.
6	Do you need a delivery limit, consumer timeout or delayed retry?	Quorum	Poison message handling is quorum-only; consumer timeouts and delayed retry are quorum features from 4.3.
7	Is lowest possible latency the priority and data safety not (no confirms, no manual acks)?	Classic	The docs list both as reasons not to use quorum queues: consensus adds latency.

#	QUESTION	IF YES	WHY
8	Will there be more than about 5,000 queues of this kind?	Review topology	The docs recommend reviewing whether some quorum queues should be classic queues or streams past roughly 5,000.

Table • Queue type worksheet, one pass per workload. Scroll sideways on small screens.

The feature matrix the answers rest on

Each worksheet answer comes from a row below, so check the rows your application depends on before you commit. Version notes reflect RabbitMQ 4.4; several rows changed in 4.3.

BEHAVIOUR	CLASSIC	QUORUM	STREAM
Replicated across nodes	No, since 4.0	Yes, Raft; up to three members by default	Yes
Exclusive or server-named	Yes	No	No
Non-durable (transient)	Yes, but transient non-exclusive queues are denied by default from 4.3	No, always durable	No, always durable
Message removed when consumed	Yes	Yes	No, consumers track an offset
Message TTL	Yes	Yes	No, use retention (max-age)
Length limits	Yes	Yes, except x-overflow reject-publish-dlx	No, use retention (max-length-bytes)
Message priority	Yes, with x-max-priority	Yes, strict 0 to 31, always on from 4.3	No
Consumer priority	Yes	Yes	No
Single active consumer	Yes	Yes	Yes, for named consumers
Dead-lettering	Yes, at-most-once only	Yes, at-least-once available	No
Poison message handling	No	Yes, delivery limit 20 by default since 4.0	No
Consumer timeout	Not evaluated from 4.3	Yes, from 4.3	Not evaluated

BEHAVIOUR	CLASSIC	QUORUM	STREAM
Global QoS prefetch	Deprecated, denied by default from 4.3	No, channel error	No, channel error
Messages kept in memory	Up to 2,048, depending on consumer rate	Never; at least 32 bytes of index per message	Never
Lazy mode or x-queue-mode	Ignored since 3.12; declaring it fails from 4.3	Not applicable	Not applicable

Table • Classic vs quorum vs stream on RabbitMQ 4.4 (rabbitmq.com docs, checked 29 October 2026). Scroll sideways on small screens.

What does not survive a move between types

A queue's type cannot be changed in place, and the behaviours below do not carry over, so check the application against this list before you declare the new queue. Migrations that skip it tend to fail in the application rather than in the broker.

Classic to quorum

- **Exclusive, server-named and transient declarations fail.** Keep those queues classic; they are usually RPC replies and per-connection scratch queues.
- **Global QoS consumers get a channel error.** Switch to per-consumer prefetch, the default in several popular clients.
- **x-overflow reject-publish-dlx is not supported.** Use reject-publish, which is also what at-least-once dead-lettering requires.
- **Redelivery is no longer unlimited.** A message redelivered more than the delivery limit (20 by default) is dropped, or dead-lettered if a DLX is set. The docs recommend a dead-letter configuration on every quorum queue; a policy with `delivery-limit` changes the limit, and from 4.3 that applies without redeclaring the queue.

- **Priority semantics change.** x-max-priority is ignored, every quorum queue has 32 strict levels, and a message with no priority is treated as 4 rather than the classic default of 0.
- **Queue TTL is only partial.** The lease is not renewed when the queue is redeclared.
- **Consumer exclusivity is not available.** Use single active consumer instead.
- **Memory scales with depth.** At least 32 bytes per message (about 1 MB per 30,000 messages), plus 16 bytes when any message TTL is set.

Queue to stream

- **Consuming no longer removes the message.** The application must track its offset and handle replays.
- **Acknowledgements and prefetch are mandatory.** Acks act as the credit that advances the consumer.
- **TTL and length limits become retention.** Without max-age or max-length-bytes the log grows until the disk runs out.
- **No dead-lettering, priorities, consumer priorities or poison message handling.** Anything relying on them needs another design.
- **Streams do not react to memory alarms.** They use minimal RAM, so watch disk instead.

Any change of type

- **The type is set only at declaration.** `x-queue-type` cannot be set or changed by a policy.
- **Redeclaring with a different type fails.** The property equivalence check raises a channel error;

`quorum_queue.property_equivalence.relaxed_checks_on_redeclaration = true` exists for applications that cannot be redeployed quickly.

- **Messages do not move by themselves.** Declare the new queue under a new name or on a new cluster, move messages with a shovel, then repoint bindings, policies and consumers.

Moving off classic mirrored queues

Any queue that relied on mirroring has had no replica since 4.0, so the first step is to find those queues and decide which ones need replication back. This is the order we run it in.

1. **Find mirroring in use.** `rabbitmq-diagnostics`
`check_if_any_deprecated_features_are_used` detects classic queue mirroring; then list queues and policies in every vhost, not just the default: `rabbitmqctl list_queues name type policy --vhost <vhost>` and `rabbitmqctl list_policies --vhost <vhost>`.
2. **Separate the queues that cannot become quorum queues.** Exclusive, server-named and transient queues stay classic.
3. **Check the application against section 3.** Global QoS, reject-publish-dlx, priorities and the delivery limit are the four we find most often.
4. **On 3.13.x, migrate with rabbitmqadmin v2.** It automates the move from mirrored classic queues to quorum queues as a blue-green deployment.
5. **Clear the ha- keys from the policies.** On 4.x they do nothing, and leaving them makes the next person think the queues are still replicated.
6. **Re-baseline throughput before tuning.** Quorum queues persist every message before confirming, so latency differs from an unreplicated classic queue. We recommend measuring first and not chasing the old numbers by cutting the replica count.

Size quorum queues before they go live

Quorum queues trade memory and disk I/O for safety, so size those two before the first workload lands.

- **Members: three unless you need to survive two failures.** A quorum queue is declared with up to three members by default; three tolerate one node failure and five tolerate two, and the docs note that more members generally means lower throughput.
- **Memory per message: at least 32 bytes of index.** That is about 1 MB per 30,000 messages per queue, whatever the message size, so set max-length on every queue to cap it.
- **Memory for the WAL: three times its limit.** The write-ahead log defaults to 512 MiB, and the docs recommend at least three times the effective WAL limit in node memory, four times for high-throughput systems.
- **Disk: the fastest you have.** Quorum queues persist all data before doing anything else.
- **Prefetch: higher than you used with classic queues.** The docs note quorum queue consumers benefit from higher prefetch so they are not starved while acks flow.
- **Backlog: heading past millions means a stream.** Do not answer it with a bigger quorum queue.

Frequently asked questions

Should I use classic or quorum queues in RabbitMQ 4?

Quorum queues for anything whose contents must survive a node failure, because classic queues have not been replicated since mirroring was removed in 4.0. Keep classic queues for exclusive, server-named and short-lived queues such as RPC replies, and where lowest latency matters more than data safety.

When should I use a RabbitMQ stream instead of a queue?

When consumers need to re-read messages or start from an offset or time, when many consumers need the same messages, or when the backlog regularly reaches millions of messages. Streams are not a work queue: consuming does not remove messages, and they have no dead-lettering or priorities.

Can I change a RabbitMQ queue from classic to quorum with a policy?

No. The queue type is set by the x-queue-type argument at declaration and cannot be set or changed by a policy. Declare a new quorum queue, move the messages, for example with a shovel, and repoint bindings and consumers.

Why are messages disappearing from my RabbitMQ quorum queue?

Often the delivery limit. Since 4.0 quorum queues drop a message redelivered more than 20 times, or dead-letter it if a dead-letter exchange is configured. Add a dead-letter configuration to every quorum queue, and raise the limit with the delivery-limit policy key if your workload needs it.

Do RabbitMQ quorum queues support message priority?

Yes. From 4.3 every quorum queue has strict priorities from 0 to 31 with no opt-in argument, x-max-priority is ignored, and a message without a priority is treated as priority 4.

Related