

— GUIDE · RABBITMQ

RabbitMQ 3.x to 4.x Upgrade Runbook

The supported hop sequence to RabbitMQ 4.4, the go/no-go gates to clear before the window, the per-node sequence inside it, and what you can and cannot undo at each stage. Checked against rabbitmq.com on 29 October 2026, together with the 4.x release notes.



By **Scott Sternloff**, Senior Enterprise Architect

LinkedIn • Updated October 29, 2026 • 11 min read • 8 sections

— TWO HOPS FROM 3.13, FORWARD ONLY ONCE FLAGS ARE ON

4.4.0 is current, and rolling upgrades to 4.4 are supported only from 4.2.x or 4.3.x. So 3.13, 4.0 and 4.1 clusters go through 4.2.x first, and 3.12 or older reach 3.13 before that. A 3.13 cluster with Khepri enabled has no in-place path at all: it needs a blue-green migration.

- **Before the window:** every stable feature flag enabled, no classic queue mirroring in use, khepri_db enabled before leaving 4.2, an Erlang version both ends of each hop support.
- **Inside the window:** one node at a time, drained first, health-checked before the next one stops.
- **Rollback:** downgrades are not supported, and a feature flag cannot be disabled once enabled. The only documented way back is blue-green, by switching applications to the old cluster.

What this runbook is for

This is the checklist you run on the day: hops, gates, commands and a rollback table, in order. For the long-form walkthrough of why a 3 to 4 upgrade can be done without downtime, read the upgrade article; for migrations between brokers or a wholesale classic to quorum move, read the migration guide. This page links to both rather than repeating them.

Confirm the hop is a supported rolling upgrade

Only the hops below are supported as rolling upgrades, so plan the route before anything else. Patch-to-patch upgrades within a series are also supported. If two versions cannot be rolled between, they also cannot coexist in one cluster.

FROM	TO	ROLLING UPGRADE?	NOTES
4.3.x	4.4.x	Supported	Both run on Erlang 27. Move to Erlang 28 or 29 after reaching 4.4.0.
4.2.x	4.4.x	Supported	The shortest route from 4.2. 4.2.9 and later need Erlang 27.
4.2.x	4.3.x	Supported	The only route into 4.3; the 4.3.0 release notes require the latest 4.2.x patch first.
4.1.x	4.2.x	Supported	Then 4.2.x to 4.4.x.
4.0.x	4.2.x or 4.1.x	Supported	Go to 4.2.x, then 4.2.x to 4.4.x.
3.13.x	4.2.x, 4.1.x or 4.0.x	Supported	Use 4.2.x. 3.13 runs on Erlang 26, so land on a 4.2 patch that also supports Erlang 26.2.
3.13.x, 4.0.x, 4.1.x	4.3.x or 4.4.x	Not supported	Upgrade to 4.2.x first.
3.13.x with Khepri enabled	Any 4.x	Not supported	Khepri in 3.13 was experimental and incompatible with 4.x. Use a blue-green deployment.
3.12.x	3.13.x	Supported	There is no rolling path from 3.12 to 4.x; the 4.0 release notes offer blue-green from 3.12.14.
3.11.18 to 3.7.18	One series up	Supported	3.11.18 to 3.12, 3.10 to 3.11, 3.9 to 3.10, 3.8 to 3.9, 3.7.18 to 3.8: one series at a time.

Table • Supported RabbitMQ rolling upgrade paths (rabbitmq.com upgrade guide, 4.4 docs, checked 29 October 2026). Scroll sideways on small screens.

Sequence the route from 3.13 so Erlang never blocks a roll

Order the steps so every node always runs an Erlang version supported by both the RabbitMQ version it leaves and the one it joins. That is how we sequence 3.13 upgrades, and it keeps every step a rolling one. The docs recommend upgrading Erlang together with RabbitMQ where a single step allows it.

1. **3.13.x on Erlang 26.2.** Move to the latest 3.13.x patch available to you and enable every stable feature flag; the 4.0 release notes require all 3.13 flags before leaving 3.x.
2. **Roll to a 4.2 patch that supports Erlang 26.2 (4.2.0 to 4.2.8).** 3.13 tops out at Erlang 26.2.x, and 4.2.9 and later need Erlang 27, so this landing patch is what lets both versions share a runtime.
3. **Roll Erlang from 26.2 to 27.x on 4.2.** 4.2.0 to 4.2.8 support 26.2 to 27.x, so this is a rolling change too.
4. **Roll to the latest 4.2.x patch, then enable every stable flag, including `khepri_db`.** Khepri is the only metadata store from 4.3, Mnesia is gone, and the 4.3.0 release notes list `khepri_db` as required.
5. **Let 4.2 settle, then roll from 4.2.x to 4.4.0 on Erlang 27.** Then enable the 4.4 feature flags. We do not run the two hops back to back; 4.4.0 also fixed an issue that could trip a 3.13 to 4.2 upgrade followed immediately by a move to 4.3.
6. **Optionally, roll Erlang to 28 or 29.** 4.4.0 fully supports both for rolling upgrades. Do not do this on 4.2 or 4.3 before 4.3.6: the docs allow Erlang 28 there for new clusters only, because Khepri clusters moving from 27 to 28 can hit a known mixed-version issue.

If the 3.13 cluster served AMQP 1.0 clients through the old `rabbitmq_amqp1_0` plugin, the 4.3.0 release notes ask for at least one rolling update after enabling the `rabbitmq_4.0.0` flag and before upgrading to 4.3.0; we apply the same rule before 4.4.

Clear the go/no-go gates before the window

Every gate below has to pass days before the window, because most of them cannot be fixed inside it. Run each check on the version you are leaving.

1. **All stable feature flags enabled.** The upgrade guide says an upgrade may fail otherwise, and a flag cannot be disabled once enabled, so this is a decision to make calmly, not at 2am. `rabbitmqctl list_feature_flags` then `rabbitmqctl enable_feature_flag all`.
2. **No deprecated features in use.** `rabbitmq-diagnostics check_if_any_deprecated_features_are_used` exits non-zero if any are. It detects classic queue mirroring; it cannot detect global QoS, so check client code for that separately.
3. **No classic queue mirroring left.** Mirroring was removed in 4.0, and mirroring keys in policies simply stop having any effect, so a mirrored queue becomes a single-replica classic queue without an error. Migrate the queues that need replication to quorum queues, then clear the ha- keys from every policy in every vhost (`rabbitmqctl list_policies --vhost <vhost>`). Leftover ha-mode policies were among the findings in a 3.13.7 to 4.2 upgrade we were called in to recover.
4. **Khepri was not enabled on 3.13.** Check `khepri_db` in `rabbitmqctl list_feature_flags` . If it is enabled on 3.13, stop: there is no in-place path, only blue-green.
5. **Applications survive the 4.3 defaults.** From 4.3.0, transient non-exclusive queues, global QoS and queue_master_locator are denied by default, and declaring a queue with `x-queue-mode` or `x-queue-version=1` fails because CQv1 is removed. If an application truly needs transient non-exclusive queues, set `deprecated_features.permit.transient_nonexcl_queues = true` on every node, and restart them, before the upgrade.
6. **Partition handling keys removed.** From 4.3 the `cluster_partition_handling` keys are accepted but have no effect, and with Khepri a majority of nodes must

be online for the cluster to be available. Remove the keys so nobody reads them as protection.

7. **An Erlang plan for every hop.** Each node must run an Erlang version both versions in that hop support; see step 2 and the matrix.
8. **The cluster is healthy and can run one node short.** No alarms (`rabbitmq-diagnostics check_alarms`), no queue or stream replica sync in progress, and `rabbitmq-diagnostics check_if_node_is_quorum_critical` passing on every node. Connections and queue leaders move to the remaining nodes while one is down, so size for that.
9. **Backups taken off the node.** The docs strongly advise backing up each node's data directory, and an exported definitions file turns a rebuild into minutes:
`rabbitmqctl export_definitions /path/off-node/definitions.json`.
0. **Clients reconnect on their own.** A node being stopped closes its client connections; applications need automatic recovery and either a load balancer or a host list.
1. **4.4 breaking changes reviewed.** OpenID Connect discovery is validated more strictly (known to affect Cloud Foundry UAA), `GET /api/auth/hash_password` became a POST, and the `x-amqp-1.0-*` headers written by 3.13 and earlier nodes are no longer parsed.
2. **A baseline and a link inventory recorded.** Queue depths, consumer counts, connections and per-node headroom, plus every federation upstream and shovel (`rabbitmqctl list_parameters --vhost <vhost>`). Without it you cannot tell a regression from a normal day, or spot the one link that did not come back.
3. **CLI tools match the broker version.** After a 3.12.14 to 3.13.14 hop we have seen `rabbitmq-diagnostics` and `rabbitmqctl` fail because the packaged tools were not replaced with the server. Verify them before you need them for a diagnosis.

Run the per-node sequence, one node at a time

Upgrade one node, prove it healthy, then move to the next; never two at once. In a three-node cluster two nodes must stay up for quorum queues, streams and Khepri to keep a majority.

1. **Gate: the node is safe to stop.** `rabbitmq-diagnostics`
`check_if_node_is_quorum_critical` must pass. In automation, `rabbitmq-upgrade await_online_quorum_plus_one` blocks until enough nodes are up.
2. **Drain it.** `rabbitmq-upgrade drain` puts the node in maintenance mode: it stops accepting connections, closes existing ones and hands its quorum queue leaders to peers. A node is expected to stay in this mode for minutes, not hours.
3. **Stop, upgrade, start.** RabbitMQ, its CLI tools and, where this hop moves it, Erlang. A restart takes the node out of maintenance mode automatically; `rabbitmq-upgrade revive` is only for a drained node you decided not to restart.
4. **Gate: it rejoined properly.** `rabbitmq-diagnostics check_running`, `rabbitmqctl cluster_status` and `rabbitmq-diagnostics check_alarms`, then compare consumers and queue depths with the baseline. Move on only when they match.
5. **Repeat for the next node.** Mixed-version clusters exist for rolling upgrades and, per the 4.3.0 release notes, should not run for more than a few hours.
6. **After the last node: enable the new flags and rebalance.** `rabbitmqctl enable_feature_flag all`, then `rabbitmq-queues rebalance all`, because leaders end up unevenly spread after a roll. Ask users of the management UI to clear browser cache and storage for its domain.

Know what you can undo at each stage

Rollback options shrink at every stage, and the docs do not support downgrades at all: they are untested, and some patch releases could not be downgraded even to the previous patch. Decide the abort point before the window opens.

STAGE REACHED	CAN YOU GO BACK?	WHAT TO DO
Gates done, no node stopped yet	Yes	Abort; nothing has changed apart from flags you enabled on the old version.
Some nodes upgraded, new flags not enabled	Not supported	Recover forward: finish the roll, or rebuild a failed node at the majority version and rejoin it. Restore from the data-directory backup only as a last resort.
New version's feature flags enabled	No	A feature flag cannot be disabled once enabled. Forward only; the definitions export is the rebuild path.
On 4.4 with classic queue data written	No	4.4 writes a new v2 message store format; a node with v2 files cannot be downgraded to 4.3.x or earlier.
Blue-green: applications moved, old cluster still running	Yes	Switch applications back to the old cluster. This is the only rollback the docs describe; keep the old cluster until the last consumer is verified.

Table • RabbitMQ upgrade rollback options by stage (rabbitmq.com upgrade and feature flag guides, 4.4.0 release notes). Scroll sideways on small screens.

When it goes wrong: symptom, cause, fix

- **A node will not rejoin after its upgrade.** A node can rejoin only if it supports every feature flag enabled in the cluster, so a flag enabled while it was down, or a version mismatch, keeps it out. Bring its software to the majority version and rejoin it; in our recoveries, resetting that one node and letting it take the cluster's metadata has been faster and safer than reversing the many.
- **Declarations start failing after the move to 4.3 or 4.4.** Transient non-exclusive queues, `x-queue-mode` and `x-queue-version=1` are now rejected, and global QoS is denied by default. Fix the application or permit the deprecated feature on every node.

- **A queue that used to be mirrored lost its messages with a node.** Its ha- policy stopped doing anything at 4.0 and it became a single-replica classic queue. Move it to a quorum queue.
- **CLI tools stopped working.** The packaged tools no longer match the broker. Reinstall the matching package and check with `rabbitmq-diagnostics status`.
- **Federation links all restarted.** Expected during a roll: queue federation links run on the queue leader's node and are recreated after every leader election. Compare against the link inventory and re-establish anything missing.
- **Throughput dropped after moving mirrored queues to quorum queues.** Quorum queues persist every message before confirming and replicate it. Use the fastest disks available, and give each node at least three times the effective WAL limit in memory (512 MiB by default), four times for high-throughput systems. We recommend against chasing the old numbers by cutting the replica count.
- **OAuth 2 logins fail after 4.4.** The identity provider fails the stricter discovery validation. Set `auth_oauth2.discovery.verify_issuer` and/or `auth_oauth2.discovery.verify_https_endpoints` to false before upgrading for providers such as Cloud Foundry UAA.

When to choose blue-green instead

Choose a blue-green deployment when any of the conditions below holds. The docs call it the safest strategy and recommend it wherever a rolling upgrade is not an option, at the cost of running two clusters for a while.

- **Khepri was enabled on 3.13.** There is no in-place path to 4.x.
- **The cluster is on 3.12 or older.** The rolling route to 4.4 is at least three hops (3.12 to 3.13 to 4.2 to 4.4), each with its own window.
- **The estate is full of mirrored queues.** `rabbitmqadmin` v2 automates the move from 3.13.x mirrored queues to quorum queues as a blue-green migration.

- **The nodes are on an operating system you were replacing anyway.**
- **You need a rehearsal.** A parallel cluster lets the application run against 4.4 before any production traffic does.

The documented sequence is: deploy the new cluster, synchronise metadata, set up federation, move consumers, drain messages, move publishers, then decommission the old cluster. Where we have priced both routes for a large or old estate, the parallel cluster has often been the one we recommended.

Frequently asked questions

Can I upgrade RabbitMQ 3.13 directly to 4.4?

No. Rolling upgrades to 4.4 are supported only from 4.2.x or 4.3.x, so a 3.13 cluster goes 3.13.x to 4.2.x, then 4.2.x to 4.4.x. A 3.13 cluster with Khepri enabled cannot be upgraded in place at all and needs a blue-green deployment.

Can I roll back a RabbitMQ upgrade?

Not in place. RabbitMQ does not support downgrades, a feature flag cannot be disabled once enabled, and a 4.4 node that has written its new classic queue message store format cannot go back to 4.3.x. The only documented rollback is blue-green: switch applications back to the old cluster.

Does a RabbitMQ 3 to 4 upgrade need downtime?

Not for the broker. A rolling upgrade stops one node at a time and the docs state there is no need to stop all nodes. Erlang can move in the same rolling way if every node runs a version both RabbitMQ versions in that hop support, which is why the route from 3.13 lands on a 4.2 patch that still supports Erlang 26.2 before moving to Erlang 27.

Why did my RabbitMQ upgrade fail partway through?

The usual causes are a stable feature flag left disabled, a node that missed a flag being enabled while it was down, or applications declaring queues 4.3 now rejects, such as transient non-exclusive queues or x-queue-mode. All three show up in the pre-flight gates if you run them before the window.

What happens to classic mirrored queues after upgrading to RabbitMQ 4?

Mirroring was removed in 4.0. The mirroring parts of policies stop having any effect and the queues keep working as single-replica classic queues, so they lose their redundancy without an error. Migrate the queues that need replication to quorum queues before the upgrade.

Related