

PostgreSQL 14 upgrade checklist

Community support for PostgreSQL 14 ends 12 November 2026; RDS and Aurora standard support ends 28 February 2027. This is the order of work I use to leave PG14 without a surprise blocker in the window or a slow application the next morning.

Updated 2026-10-21. Dates verified 2026-10-13; re-check the [RDS](#) and [Aurora](#) release calendars before scheduling.

1. Inventory

- ☐ List every PG14 instance and cluster, including read replicas, staging copies and forgotten reporting databases. On AWS: `aws rds describe-db-instances --query "DBInstances[?starts_with(EngineVersion, '14')].[DBInstanceIdentifier,EngineVersion]"`.
- ☐ Note the minor version. RDS 14.18 and 14.19: 31 October 2026; Aurora 14.17: 30 November 2026; 14.18: 31 December 2026. See [what happens at a minor-version deadline](#).

- ☐ Record extensions and versions in every database:

```
SELECT current_database(), extname, extversion FROM pg_extension ORDER BY 2;
```

- ☐ Find logical replication: slots, publications and subscriptions. These feed CDC tools (Debezium, AWS DMS, Fivetran, Airbyte) and need their own plan.

```
SELECT slot_name, slot_type, plugin, active FROM pg_replication_slots;
SELECT pubname FROM pg_publication;
SELECT subname, subenabled FROM pg_subscription;
```

- ☐ List client drivers, ORM and pooler versions (PgBouncer, RDS Proxy). Note anything still using MD5 password authentication; PostgreSQL 18 deprecates it.

2. Target version and path

- ☐ Pick the target from your platform's supported list, not the newest number. On RDS: `aws rds describe-db-engine-versions --engine postgres --engine-version <current> --query "DBEngineVersions[*].ValidUpgradeTarget[*].EngineVersion"`.
- ☐ Choose the path: in-place major upgrade (downtime for the whole `pg_upgrade`), RDS/Aurora Blue/Green (short switchover, logical replication limits), or self-managed logical replication. The trade-offs are in [RDS and Aurora 14 → 18: downtime and Blue/Green](#).
- ☐ Confirm each extension exists at a compatible version on the target. RDS removes `plrust` from version 18; check the provider's extension list for the rest.

3. Clear known blockers

- ☐ No open prepared transactions: `SELECT count(*) FROM pg_prepared_xacts;`
- ☐ No user columns of `regproc`, `regprocedure`, `regoper`, `regoperator`, `regconfig`, `regdictionary`, `regnamespace` or `regcollation`.
- ☐ No invalid databases left by an interrupted `DROP DATABASE`: `SELECT datname FROM pg_database WHERE datconnlimit = -2;`
- ☐ Logical slots: RDS and Aurora block an in-place upgrade from 14 while they exist. Self-managed `pg_upgrade` from 14 does not block; it drops them without a warning. Plan the CDC restart either way: [what happens to logical replication slots](#).
- ☐ Count large objects: `SELECT count(*) FROM pg_largeobject_metadata;` Millions of them make the schema restore slow and memory-hungry, and Blue/Green does not replicate them.
- ☐ Self-managed to 18: PostgreSQL 18's `initdb` enables data checksums by default and `pg_upgrade` refuses a mismatch. Use `--no-data-checksums` for the new cluster if the old one has none.
- ☐ Run the check on a copy: `pg_upgrade --check` self-managed, or a trial upgrade of a snapshot restore on RDS and read `pg_upgrade_precheck.log`. [Common pg_upgrade --check failures and their fixes](#).

4. Capture a baseline

- ☐ Save the top statements by total execution time from `pg_stat_statements` over a normal busy period. The new cluster starts with empty cumulative statistics, so this is the only comparison point. [How to rank them](#).

- ☐ Save `EXPLAIN (ANALYZE, BUFFERS)` for the 10–20 queries that matter most to users, with real parameter values. `FORMAT JSON` keeps them machine-readable; [PlanReader](#) opens either format and flags estimate errors and spills.
- ☐ Record p95 latency, CPU, connections and error rate for the same window.

5. Rehearse on a copy

- ☐ Restore a recent snapshot or clone and run the real upgrade on it. Time each phase: upgrade, statistics rebuild, extension updates, application smoke test.
- ☐ Run the baseline queries on the upgraded copy before and after `ANALYZE`. Plans that change shape are the risk list.
- ☐ Point a test build of the application, its migrations and its CDC consumer at the copy.

6. Cutover

- ☐ Write the runbook with owners and timings from the rehearsal, including the go/no-go moment.
- ☐ Take a snapshot or backup immediately before. With `pg_upgrade --link`, the old cluster cannot be started safely once the new one has run.
- ☐ Rebuild statistics before full traffic: `ANALYZE` per database on RDS and Aurora; self-managed `18, vacuumdb --all --analyze-in-stages --missing-stats-only` then `vacuumdb --all --analyze-only`. Analyze partitioned parents explicitly. [Why this step matters](#).
- ☐ Update extensions after reading their release notes: `ALTER EXTENSION ... UPDATE;`
- ☐ Recreate replication slots and restart CDC consumers with a re-snapshot or reconciliation plan for the gap.

7. Validate and keep the way back

- ☐ Compare top statements and baseline plans with the pre-upgrade capture within the first hour, then after a full business day.
- ☐ Watch for a nested loop where there was a hash join, or row estimates far from actual rows: that is a missing or stale statistic, not a new-version bug, until proven otherwise.
- ☐ Agree the rollback trigger in advance (for example, p95 above an agreed limit for 30 minutes after statistics are rebuilt) and how long the old instance or snapshot is kept.

