
PERCONA
PostgreSQL Major Version Upgrades:
Choosing the Right Path for Production
Bhargav Kamineni - PostgreSQL DBA
Bangkok, March 2026
©2025 Peroona

Agenda

- Why upgrade PostgreSQL
- Versioning Policy
- PostgreSQL Upgrade types
- Minor version upgrades - Small upgrades, Big Impact
- Major version upgrades - Big Upgrades, Big Changes
- PostgreSQL Upgrade Methods
- Quick Comparison
- How to choose right path for production

Why upgrade PostgreSQL

- Security Fixes
- Bug Fixes
- Performance Improvements
- New Features
- Community support

Quickly Compare PostgreSQL Release Notes

https://why-upgrade.depesz.com/show?from=14.8&to=18.3&keywords=

PostgreSQL Version Numbering

- PostgreSQL 18.3 - > [18] . [3]
  Major Minor
- Minor version releases: quarterly
- Major version releases: once per year
- Explore the PostgreSQL release roadmap to pinpoint the schedule for upcoming minor upgrades.

PostgreSQL Upgrade types

In PostgreSQL, upgrades typically fall into two categories:

- Minor Version Upgrade
- Major Version Upgrade

Minor Version Upgrades: Small upgrade, Big Impact

- They primarily deliver bug fixes and security patches, with no user-visible feature changes.
- Compatibility risks are extremely low, making the process straightforward.
- These upgrades typically require minimal downtime and effort.
- Organizations should stay current with minor releases to maintain stability and security.

How to do minor version upgrades

- Review the release notes for the target minor version
- Install the new minor version binaries (preferably on standbys first)
- Restart PostgreSQL to activate the new version

That’s it — upgrade complete.

Major Version Upgrades: Big upgrade, Big changes

- Requires moving to a new PostgreSQL major version (e.g., 14 → 16)
- May involve compatibility and application validation
- Typically requires planned downtime or migration strategy
- Needs careful testing and rollback planning
- Offers performance improvements and new capabilities

PostgreSQL Major Upgrade Methods

- pg_dumpall
- pg_dump / pg_restore
- Logical replication
- pg_upgrade

Major Version Upgrade Using pg_dumpall

pg_dumpall creates a logical, text-format dump of the entire PostgreSQL cluster
Includes all databases in the instance
Captures global objects such as roles, tablespaces, and configuration grants
Commonly used for small clusters or simple migrations

Advantages

- Works well for small database clusters
- Upgrade can be performed with simple commands
- Produces clean, de-bloated tables after restore
- Ensures globals are fully preserved

Limitations

- Not suitable for large databases (downtime can be significant)
- Supports plain text format only
- No parallelism — backup and restore are single-threaded
- Restore time grows linearly with database size

Major Version Upgrade Using pg_dump

pg_dump performs a logical backup of a single database
One of the most commonly used tools for logical PostgreSQL upgrades
Supports flexible backup formats and parallel restore
Often combined with pg_dumpall to capture global objects

Advantages

- Works well for small and large individual databases
- Allows upgrading selected databases only
- Produces clean, de-bloated tables after restore
- Supports multiple backup formats (plain, custom, directory)
- Enables parallel restore to reduce downtime

Limitations

- Not ideal for clusters with many databases
- Requires pg_dumpall separately for roles and tablespaces
- Downtime can increase with very large databases
- Requires additional disk space for the logical dump
- Migration time scales with data volume

Major Version Upgrade Using pg_upgrade

pg_upgrade is the standard utility for major version upgrades, Performs a fast, in-place upgrade of the PostgreSQL cluster. Much more efficient than dump and restore for large databases

Key Features

- Ensures catalog compatibility between versions
- In-place upgrade of existing data files
- Minimal downtime compared to logical methods
- Upgrade can be executed with simple, well-defined steps

Limitations

- Designed for major version upgrades only
- No partial upgrades — entire cluster is upgraded

pg_upgrade `--copy` mode

--copy mode

- physically copies every data file from old cluster → new cluster.
- Two completely independent clusters
- Requires ~2× disk space
- Slowest
- Best when safety > speed
- Easy rollback

pg_upgrade `--link` mode

--link mode

- Instead of copying files, PostgreSQL creates hard links.
- A hard link means two file names pointing to the same data blocks on disk.
- Requires minimal additional disk
- Fastest approach
- Best only when downtime must be extremely low

pg_upgrade `--clone` mode

--clone mode

- Supported on only XFS,Btrfs,APFS
- Files are logically copied, but data blocks are shared until modified.
- Uses copy-on-write (CoW) filesystem
- Old and new clusters remain logically independent
- Best balance of speed and safety

Upgrading standbys

- When using pg_upgrade `--link` for upgrading primary database - we can use rsync from the upgraded primary to quickly upgrade standby
- If rsync fails use pg_basebackup for rebuilding the standy's
- Always keep one standby untouched during the upgrade

PostgreSQL Upgrade Using Logical Replication

What is Logical Replication?

- Allows data replication between different PostgreSQL versions
- Uses publications and subscriptions to stream changes
- Normally known as Blue-Green deployments

Advantages

- Enables near-zero downtime major version upgrades
- Old and new clusters can run simultaneously
- Applications can switch traffic once the new cluster is ready

Restrictions

- DDL changes are not replicated
- Supports tables only (no sequences or large objects)
- Tables require PRIMARY KEY / UNIQUE or replica identity
- Initial setup is more complex than pg_upgrade

Quick Comparison of Major upgrades

Method	Downtime	Resources	Complexity	Risk
pg_dump / restore	High (size dependent)	~2× disk	Low	Low
pg_upgrade –copy	High (size dependent)	~2× disk	Medium	Low
pg_upgrade –link	Usually < 5 to 10 mins (depends on the number of objects in DB)	Low	Medium	High
pg_upgrade –clone	Usually < 5 to 10 mins (depends on the number of objects in DB)	Low	Medium	Low
Logical replication	Near-zero	~2× disk	High	Medium

How to choose right path for upgrade ?

- Choosing the right upgrade method is primarily about constraints — especially downtime tolerance and database size.
- pg_upgrade fits most production cases, while logical replication is reserved for near-zero downtime requirements.
- Regardless of the method, testing and validation are what make upgrades successful

PERCONA

Questions ?