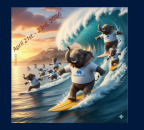
RDS Maintenance: Strategies for Patching and Version Upgrades

Karen Ng
Senior Technical Account Manager
AWS

Abhimanyu Tomar
Senior Database Specialist Technical Account Manager
AWS

Defense Strategy

- RDS PostgreSQL Updates: What You Need to Know
- The Upgrade Decision Framework
- Minimizing Downtime: From In-Place to Near Zero-Downtime
- Proven Best Practices and Common Problems
- Optimize Upgrade Process at Scale



RDS PostgreSQL Updates: What You Need to Know

PostgreSQL Upgrades

- Types of Upgrades
  - Operating System (OS)
  - Database Engine

Operating System Updates

- Types of OS patches
  - Distribution upgrades → new Operating System versions
  - System patches → security, bug fixes, performance improvements(rare)
- Key behavior
  - Can be mandatory or optional
  - Security-related updates are mandatory and must be applied
  - Applied via maintenance window or adhoc by customer

Minor Version Upgrades

- Minor version upgrades
  - Frequent releases
  - Bug Fixes/Patches to the binaries
  - Rarely contains new minor functionality/performance enhancements
  - May contain important security fixes
  - Requires spot-testing by AWS+Customer
  - Automate with AMVU (RDS)

Major Version Upgrades

- Infrequent Releases
- Tracks the community yearly release cycle
- Introduces new functionality/performance enhancements
- May change system catalogs and page formats
- Supports skip version in-place upgrade

Differences between Major and Minor version upgrades

	Minor Version Upgrade	Major Version Upgrade
Multi-AZ	Upgrades primary + Standby together	Same Behavior
In-Region Read Replica	Upgrades Replicas First	Upgrades with primary
Parameter Group	No Change	New Custom Group Required
Upgrades automatically	Yes (with AmVU enabled)	No
Database Data Files	No Change	Update
Table Statistics	Preserved	Rebuild for < PostgreSQL 18 Use vacuumdb --all --analyze-in-stages --missing-stats-only to update extended statistics.
Backward compatible	Yes	Not guaranteed
Extensions	No action required	Require Upgrades

The Upgrade Decision Framework

Who decides Upgrades & Versions?

- **Customer Responsibility**
  - Choose major version
  - Plan timeline for major version upgrades
  - Stay current with minor version upgrades/patches
  - Decide when and how to perform:
    - Major version Upgrades
    - Minor version Upgrades
- **Amazon RDS Service Team**
  - Make new database versions available
    - After release by the Postgres Community
    - After AWS testing and validation
  - Provide automation options (AMVU – Auto Minor Version Upgrades)
    - Enhance upgrade experience for customers

RDS vs Self-managed

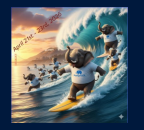
	RDS	Self-Managed
Upgrade control	AWS controls the binary; you control when but not how.	Full control over timing, pg_upgrade flags, custom scripts.
Extension flexibility	OS binary update managed by RDS. Limit to supported extensions. Trusted language extension allow custom extensions.	Install anything, any version, compile from source. Require to update OS binaries.
Superuser & lower-level access	rds_superuser role with meaningful limitations. No file system access. Use parameter group.	Full superuser, pg_hba.conf, custom postgresql.conf tuning.
Upgrade Testing	Snapshot restore gets you close, but parameter groups and IAM layers can add variables	Can mirror production exactly in a test environment
Rollout upgrade	Use AWS Organization rollout policy for upgrade fleets at scale	Require custom tooling for self-managed.

Bottom line. When this matters?
Need full control over the upgrade process? Self-managed is the right choice. To reduce operational burden at scale, choose RDS.

Extension Compatibility

Extension	Good to know
PostGIS	Major version upgrades require manual ALTER EXTENSION
pgvector	RDS support different extension version in different RDS versions. Newest extension version is not guarantee.
pg_cron	Needs re-enabling post-upgrade
pg_partman	Catalog changes between major version can break it.

Key Take Away
Check Extension upgrade requirement, test in a copy, don’t assume compatibility. Check pg_available_extension_versions view for supported extensions. Most extension needs to be added to shared_preload_libraries in custom parameter group.

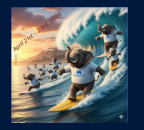


Minimizing Downtime: From In-Place to Near Zero-Downtime

Downtime by Maintenance Type

	OS Update	Minor Upgrade	Major Upgrade
Single-AZ	Minutes (restart)	Minutes (restart)	High (full downtime)
Multi-AZ Instance	Seconds (failover)	Minutes (restart)	High (full downtime)
Aurora PostgreSQL	Near-zero (rolling)	Near-zero (ZDP*)	High (full downtime / switchover)
Aurora Global Database	Rolling per region	Requires regional sequencing	High (cross-region coordination)

[1] Minor version upgrade and ZDP and Limitation of ZDP.
[2] Patch level compatibility for managed cross-region switchover & failover.
[3] AMVU for Aurora clusters.



RDS upgrade options

Major version Upgrade Approaches

In-Place Upgrade

- Direct upgrade path
- Downtime required

Snapshot Migration

- Isolated Testing Environment
- Risk-free validation of new version

Database Migration Service (DMS)

- Minimal Downtime
- Suitable for critical applications.

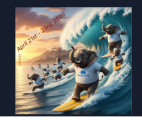
Logical Replication

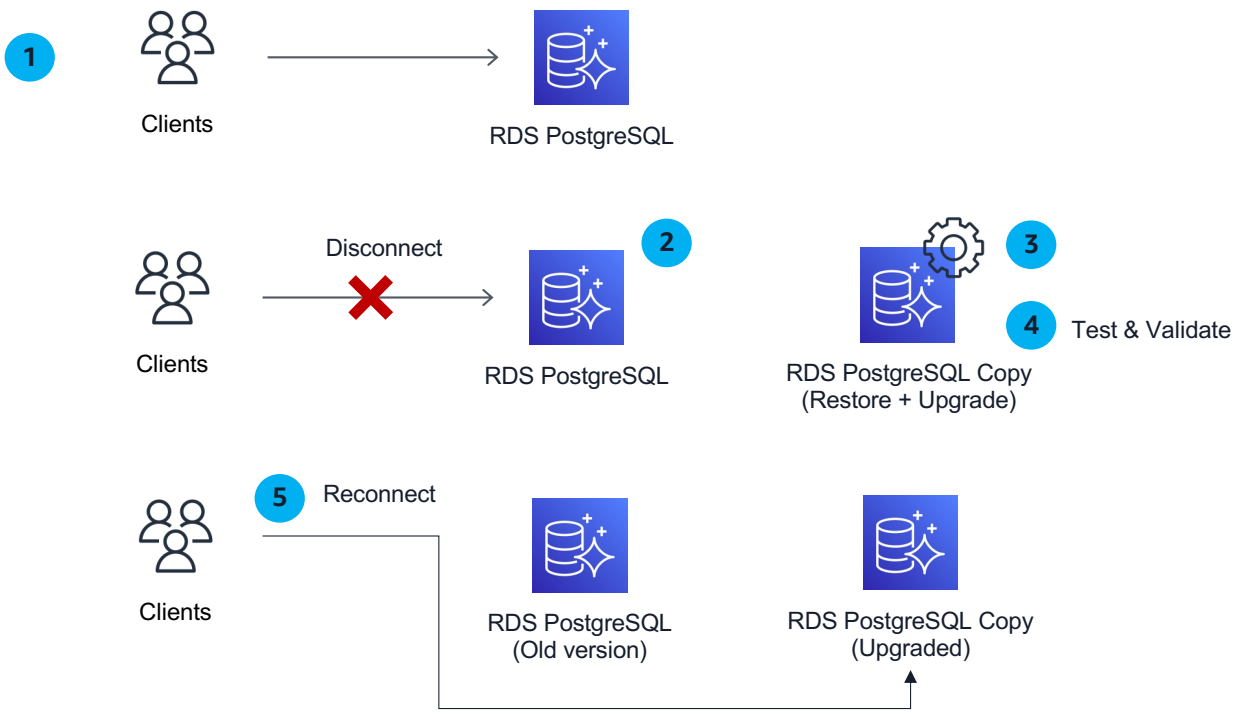
- Native PostgreSQL
- Flexible version compatibility

Blue/Green Deployment

- Managed switchover process.
- Minimize risk upgrade path

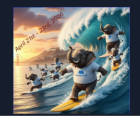
Snapshot Restore



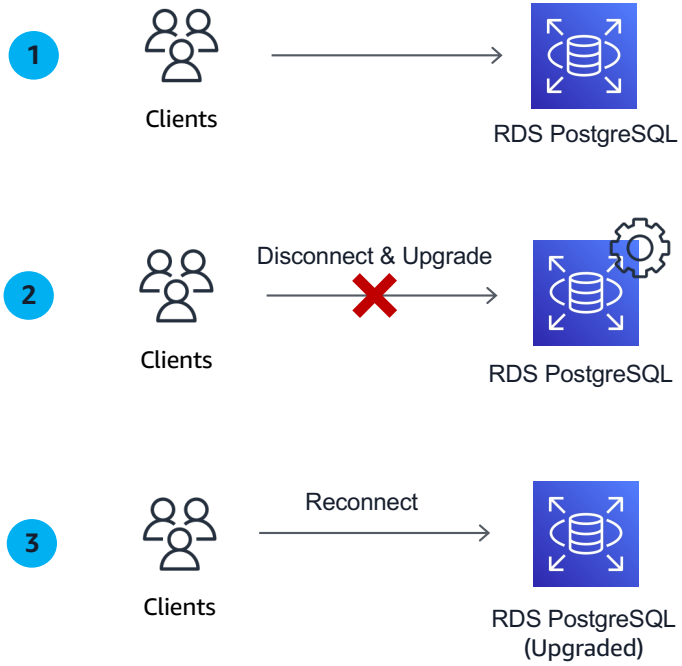


Blog: https://aws.amazon.com/blogs/database/amazon-rds-snapshot-restore-and-recovery-demystified/

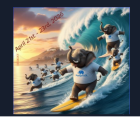
In-place Upgrade



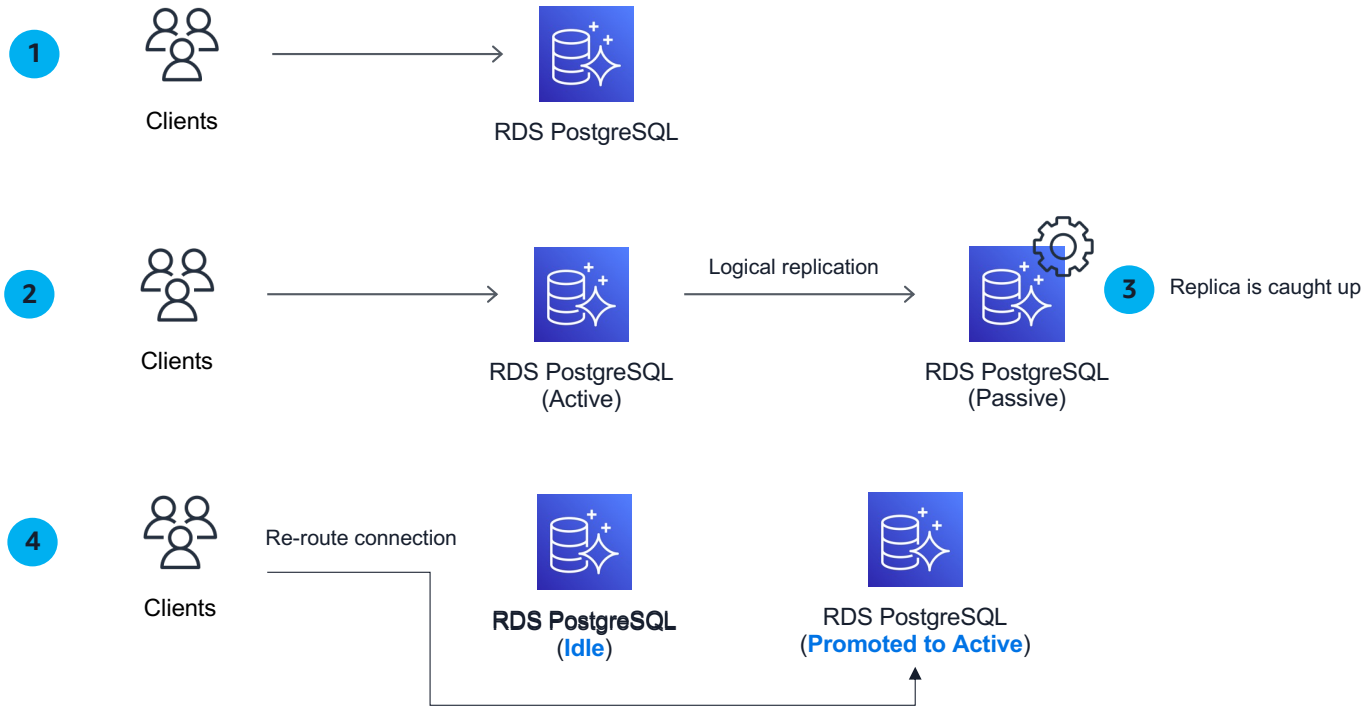
Downtime



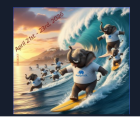
Upgrade option: active-passive (blue-green)



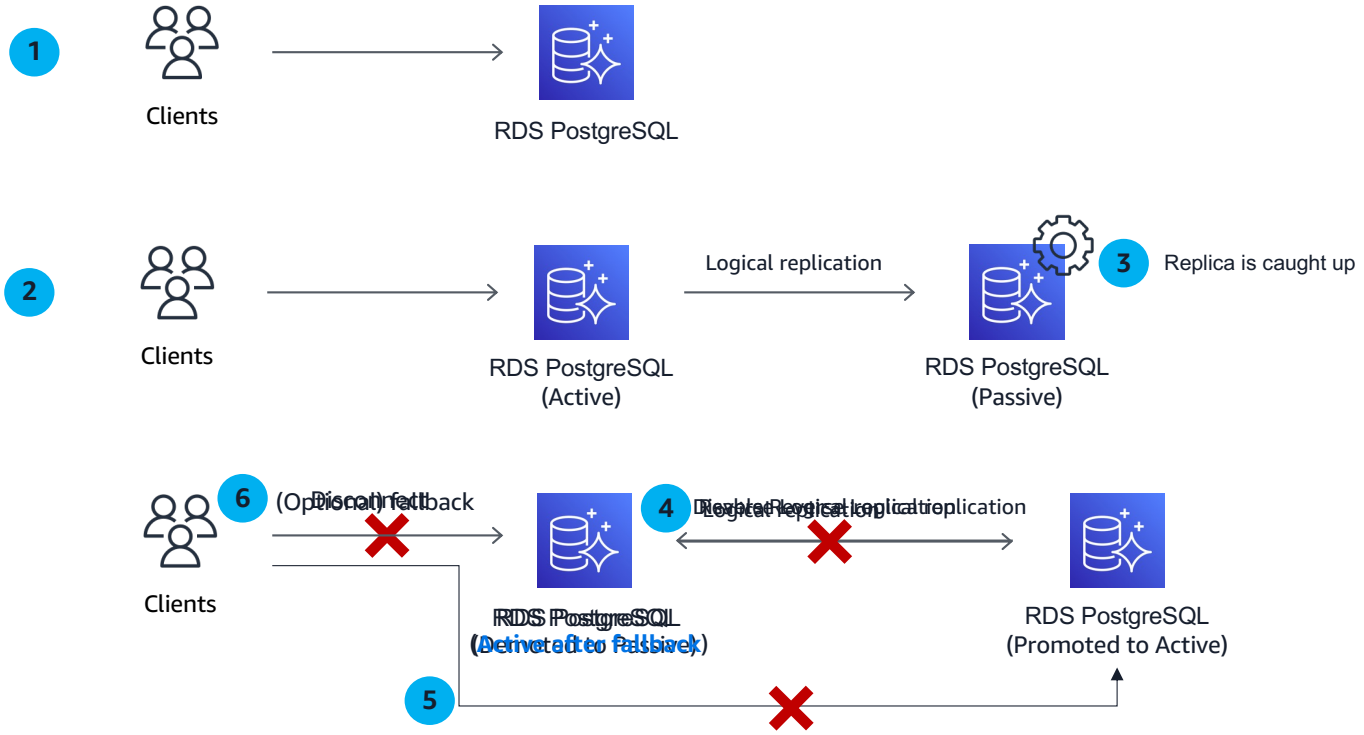
Near-zero Downtime



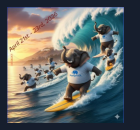
Upgrade option: active-passive with fallback

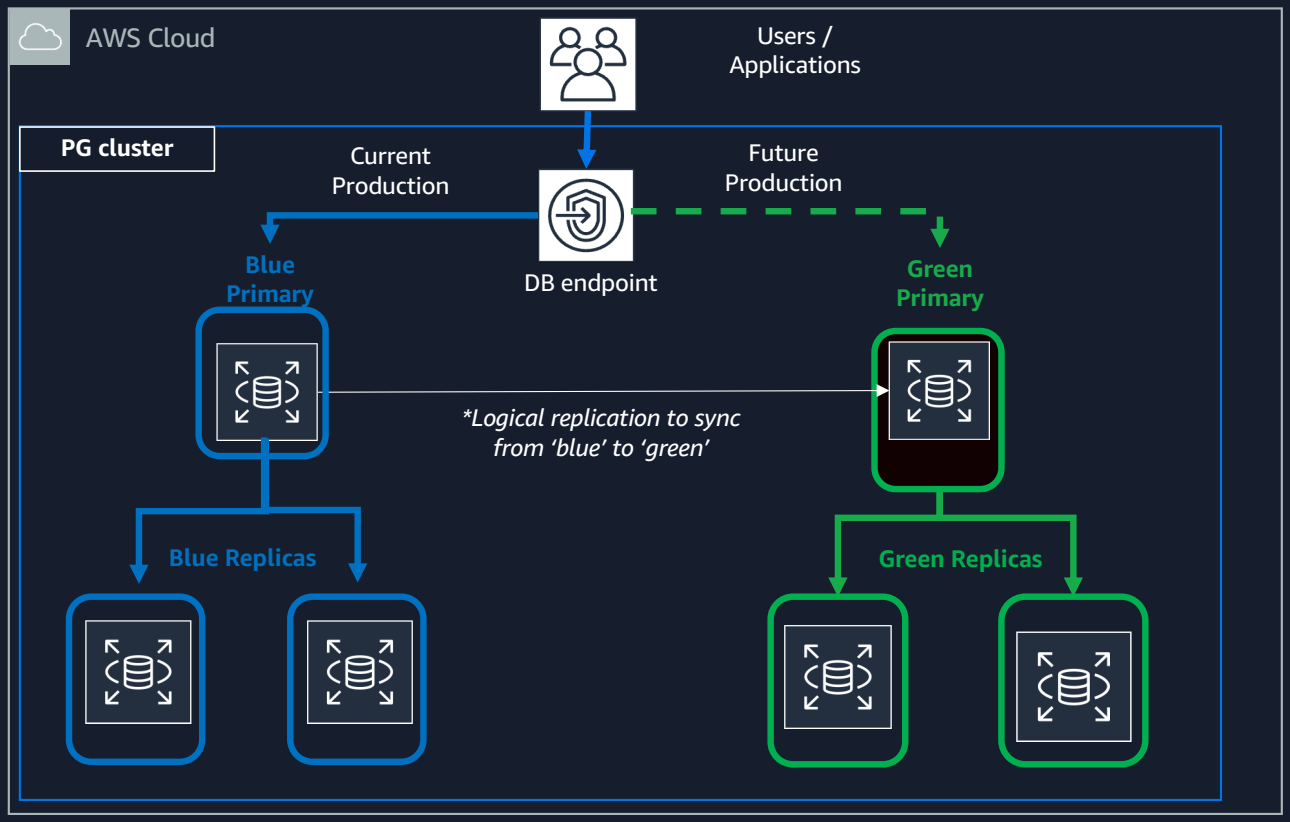


Near-zero Downtime With fallback



RDS Blue/Green Deployments





What it does:

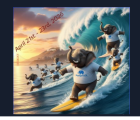
- Creates a mirrored copy of the current production environment (blue) as the green environment (future production)
- Sets up logical replication between blue primary and green primary.
- Modify green, add/remove replicas, and test changes in green environment before switchover

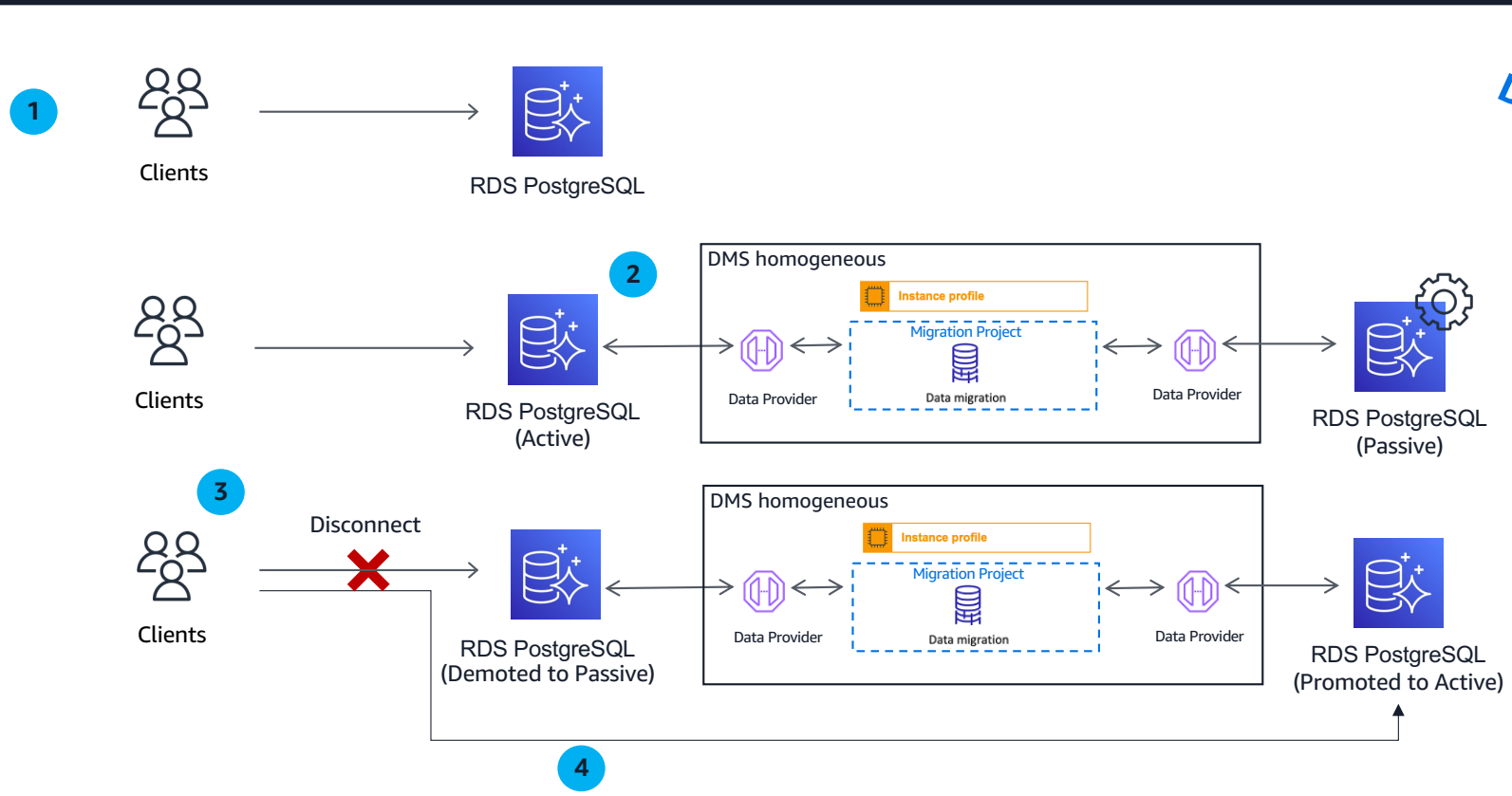
Tasks in Green before switchover:

- Monitor replication lag.
- Run Analyze in Green (major version).

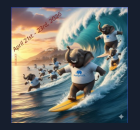
- RDS PostgreSQL - Default Physical Replication, use logical replication for major version in Green environment.
- Aurora PostgreSQL – Logical replication

Upgrade option: Database Migration Service

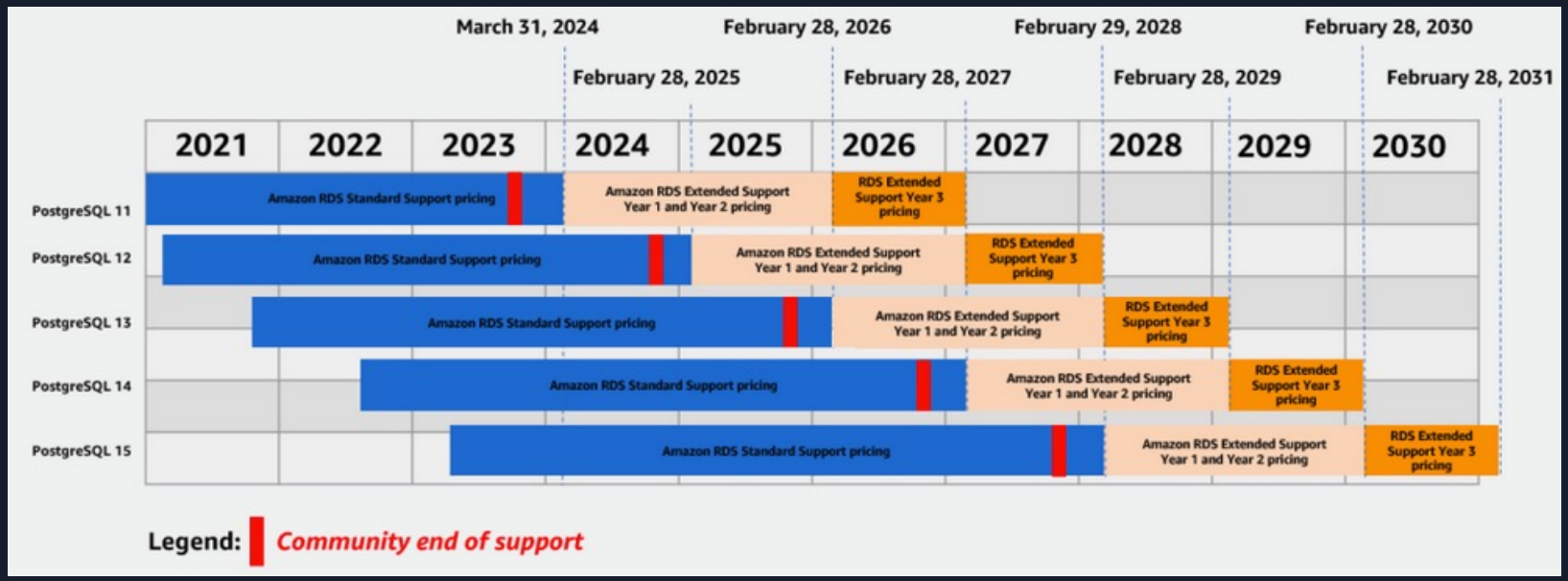




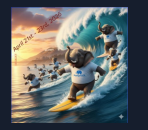
Extended Major Version Support



- PostgreSQL 12 & 13 are supporting 3 years past community end-of-life

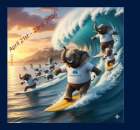


- Release calendars for Aurora PostgreSQL
- Release calendars for Amazon RDS for PostgreSQL



Proven Best Practices and Common Problems

Best Practices



- Minor Version Upgrade:
  - Update DB engine, when a new release becomes available (recommended)
  - Choose automatic minor upgrades vs. manual based on need
  - Set an appropriate maintenance window
  - Required updates are scheduled automatically during a maintenance window
  - Physical Read replica can have different minor version than primary, but not major
- Major Version Upgrade - To minimize downtime:
  - Use pglogical or native logical replication and/or Aurora Fast Cloning
  - Use Blue/Green Deployment
  - Test engine update process in a pre-prod/testing environment
  - Check for unsupported database classes, see Supported DB engines for DB instance classes.
  - Check for unsupported usages (i.e. Prepared transactions, data type, invalid database).
  - Run Analyze post major version upgrade
  - Upgrade the extensions

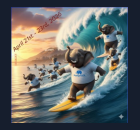
Downtime Strategy

Feature	RDS Proxy	PgBouncer
Managed by AWS	Yes	No (self-managed)
Major Version Upgrades	Yes (reduces impact)	Yes (reduces impact)
Minor Version Patches	Yes	Yes
Typical Downtime	5–30 seconds failover	Depends on drain time
Connection Pooling	Yes (built-in)	Yes (primary purpose)
Additional Cost	~$0.015/hr per vCPU	EC2 instance cost
Application Changes	Endpoint change only	Endpoint + config
Best For	Production apps	High-connection apps

Rollback Strategy

Criteria	pgactive	Logical Replication
RDS Compatibility	Limited (custom setup)	Fully supported
Setup Complexity	High	Medium
Rollback Speed	Near-instant	Seconds (DNS change)
Data Loss Risk	Low (bi-directional sync)	Low (if subscriber is current)
Cross-Version Support	Yes	Yes
Resource Overhead	High (two active nodes)	High (two running instances)
Operational Complexity	Very High	Medium
Recommendation	Complex workloads	Most RDS use cases

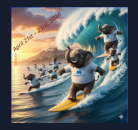
Common problems during Upgrade



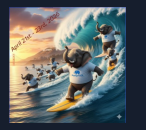
- Sometimes RDS Postgres upgrade encounters certain issues
- pg_upgrade.log contains details of these issues

Error	Reason
INCOMPATIBLE_PARAMETER	Could occur if a memory-related parameter such as shared_buffers or work_mem was set too large
STORAGE_FULL	Instance ran out of space
Logical Replication slots	If the database is using logical replication slots

Common problems during Upgrade

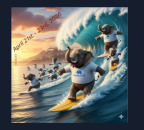


Error	Reason
Release Date Dependency	If the release date of the target version is older than the release date of the current version
Master Username	If master username starts with pg_, the upgrade fails
Upgrades taking long time	The duration is dependent on the number of database objects not their physical size
Slowness after the upgrade	Optimizer statistics are not upgraded so they need to be re-gathered after the upgrade by running ANALYZE. Hence this time has to be accounted in the overall upgrade timeline



Optimize Upgrade Process at Scale

AWS Organizations Upgrade Rollout Policy



Staged Rollouts with Confidence

- Automatically deploy updates to dev/test → production → mission-critical environments
- Built-in delays between stages minimize disruption risk

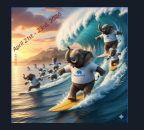
Enterprise-Scale Orchestration

- Manage upgrades across thousands of databases in multiple accounts and regions
- Policy-based control—no custom scripts or manual scheduling required

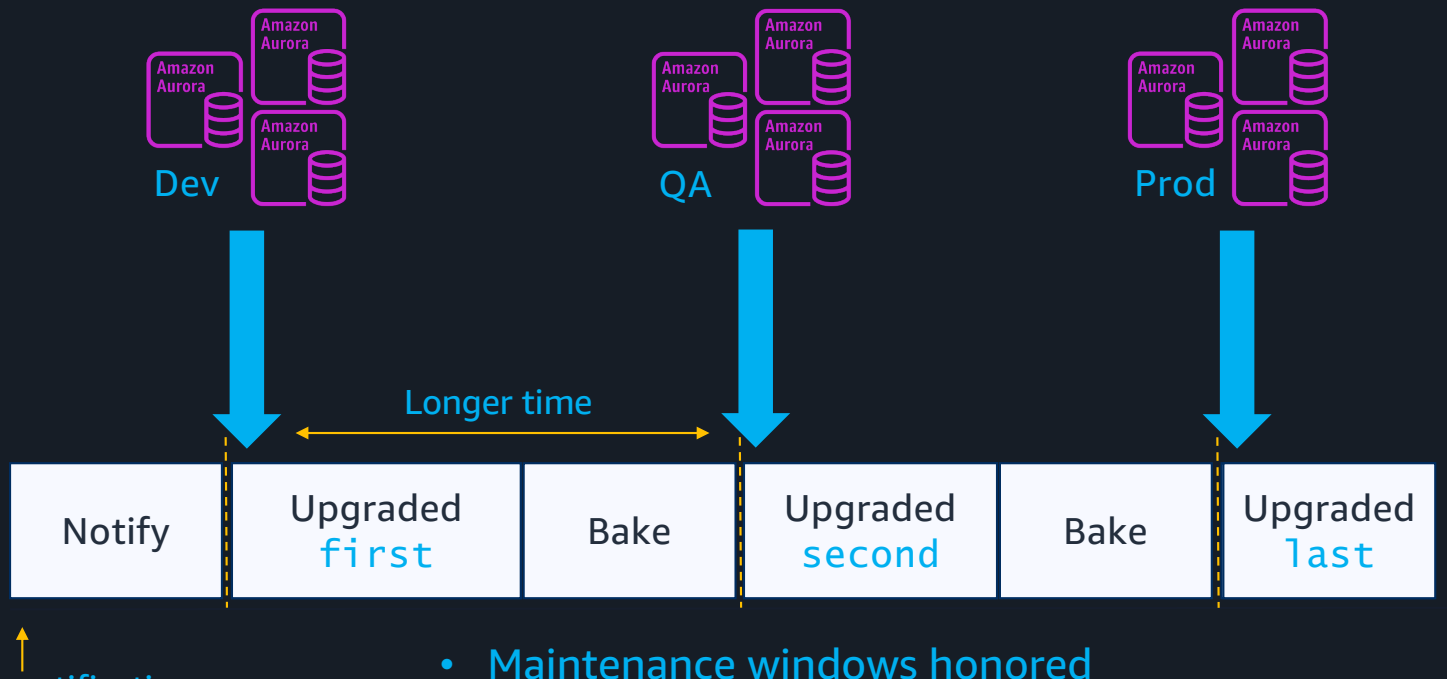
Full Observability & Control

- Advance notifications via AWS Health Dashboard
- Real-time progress tracking with pause/override capabilities

Fleet-wide upgrade orchestration

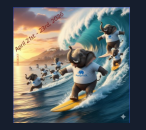


AWS Organizations Upgrade Rollout Policy



- Maintenance windows honored
- Bake time
- Upgraded in sequence

Don't guess. Know.



```
postgres=> select version();
```

Github Repo: https://bit.ly/4tuXPGq

Thank you!

https://pulse.aws/survey/M486ADHP