ADR: Database Management via Harness CI/CD — Flyway vs Liquibase

Date: 2025-09-22

Status: Draft → Review → Approved

Owners: Data Platform / DevEx / Architecture

Scope: Managing AWS Aurora PostgreSQL schema changes through CI/CD without direct console or DB access.

# Background and Problem Statement

We need a secure, developer-friendly, and auditable way to manage database schema and data migrations for AWS Aurora PostgreSQL. Direct access to AWS Console, RDS endpoints, and production databases should be restricted. All changes must be versioned in Git and executed through Harness pipelines. We evaluated Flyway and Liquibase as the two primary tools to standardize migrations.

# Requirements

|  |  |
| --- | --- |
| Type | Requirements |
| Functional | Versioned schema and data migrations; dry-run validation; rollback/undo strategy; environment promotion; migration history tracking; backward-compatible deploys. |
| Non-functional | Zero/near-zero downtime where possible; idempotent re-runs; deterministic ordering; scale to multiple services and databases; clear observability. |
| InfoSec | No direct DB or AWS Console access; credentials via IAM role/Secrets Manager; principle of least privilege; audit logs retained; network access restricted by SGs. |
| Risk | Destructive changes; drift between environments; long-running locks; failed migrations partially applied; human error in scripts. |
| Data | Migration artifacts stored in Git; checksums tracked; migration history table managed by tool; PII is not written to logs; backups/restores documented. |

# 1-Way / 2-Way Door Decision Record

|  |  |  |  |
| --- | --- | --- | --- |
| # | Checklist Item | Yes/No | Notes |
| 1 | Impacts developer and release experience across org | Yes | Standard tool becomes mandatory for DB changes |
| 2 | Legal/regulatory/compliance implications | Yes | Auditable change history and access controls |
| 3 | Enterprise-wide platform/infrastructure scope | Yes | Used by all product teams with DBs |
| 4 | Cross-domain impact (Ops, Security, Data, Apps) | Yes | Pipelines, secrets, and database operations |
| 5 | Vendor lock-in risk | Low/Med | Both are open-source with commercial options |
| 6 | High reversal cost | Medium | Migration to the other tool is feasible but non-trivial |
| 7 | Enables GenAI new functional area | No |  |
| 8 | Customer impact if wrong | Yes | Production data integrity/perf risks |

Classification: 2-Way Door (high blast radius) — reversible with planned migration, but impacts all teams.

# Access Control Model

|  |  |
| --- | --- |
| Role | Access Type |
| Developer | Git-only contribution of versioned SQL/changesets; cannot connect directly to DB. |
| CI/CD | Harness pipelines assume IAM role to fetch secrets and run Flyway/Liquibase CLI against RDS. |
| Platform/DBA | Manage pipelines, guardrails, approvals; tune DB parameters; review migrations; manage backups. |

# High-Level Flow

• Developer creates migration (SQL for Flyway; changeset for Liquibase) and pushes to Git.

• Pull Request review (architecture/DBA/security as needed).

• Harness CI: validate syntax, naming, dry-run (no-op) against a disposable DB.

• Harness CD: apply migrations in lower envs (DEV → TEST → STAGE).

• Promotion to PROD via gated approval; prechecks (connections, locks, pending migrations).

• Audit: store logs, artifacts, and version/tag applied.

# Repository Structure

/db-migrations/  
 versioned/  
 V1\_\_initial\_schema.sql  
 V2\_\_add\_customers\_table.sql  
 V3\_\_fix\_order\_column.sql  
 repeatable/  
 R\_\_load\_reference\_data.sql  
 rollback/  
 U1\_\_drop\_table.sql  
 baseline/  
 B1\_\_baseline.sql

# Naming Standards

Follow Flyway standards for file names; Liquibase uses changeset IDs.

|  |  |
| --- | --- |
| Type | Standard |
| Versioned migrations | versioned/V1\_\_add\_table.sql (applied exactly once in order) |
| Undo migrations | rollback/U1\_\_drop\_table.sql (manual rollback path) |
| Repeatable migrations | repeatable/R\_\_load\_reference\_data.sql (reapplied on checksum change) |
| Baseline migrations | baseline/B1\_\_baseline.sql (represents state prior to versioning) |

# CI Pipeline: Validation Stage

• SQL syntax validation / lints.

• Naming convention check and checksum verification.

• Dry-run using Flyway (`-dryRunOutput`) or Liquibase (`updateSQL`).

Example Flyway Dry Run:  
flyway -url=jdbc:postgresql://dummy-host/db -user=dummy -password=dummy \  
 -locations=filesystem:./db-migrations/versioned -dryRunOutput=flyway\_dryrun.sql info

# CD Pipeline: Migration Execution

• Obtain DB credentials via IAM role/Secrets Manager; rotate regularly.

• Execute Flyway/Liquibase CLI from a Harness shell step with read-only artifact access.

• Monitor logs and notify on success/failure; emit metrics to CloudWatch.

Example Flyway Command:  
flyway -url=jdbc:postgresql://${DB\_HOST}:${DB\_PORT}/${DB\_NAME} -user=${DB\_USER} -password=${DB\_PASSWORD} \  
 -locations=filesystem:./db-migrations/versioned,filesystem:./db-migrations/repeatable migrate

# Rollback Strategy

Prefer roll-forward fixes for production incidents. For reversible operations, provide undo scripts (Flyway: `U\_\_\*` files; Liquibase: rollback tags/commands). Ensure backups and PITR (Point-In-Time Recovery) are available for catastrophic rollbacks.

Example Flyway Undo Execution (manual):  
flyway -url=jdbc:postgresql://${DB\_HOST}:${DB\_PORT}/${DB\_NAME} -user=${DB\_USER} -password=${DB\_PASSWORD} \  
 -locations=filesystem:./db-migrations/rollback migrate

# Tools Checklist

☐ Git repository with `/db-migrations/` folder

☐ Harness CI/CD pipelines (validation + apply)

☐ Flyway CLI (preferred) or Liquibase CLI installed in pipeline image

☐ Secrets stored in AWS Secrets Manager; access via IAM role

☐ Aurora PostgreSQL provisioned via IaC; no direct psql access in PROD

# Promotion to Prod

• Track applied migration versions and checksums.

• Release tag per migration batch; change ticket linked.

• Mandatory approvals (Platform/DBA/Security) for PROD.

# Developer Enablement Tips

• Provide SQL templates and naming helpers.

• Pre-commit hooks to lint SQL and enforce naming.

• Harness dashboards/alerts for migration outcomes.

• Offer a local disposable DB for dry-run testing.

# Benefits

|  |  |
| --- | --- |
| Benefit | Description |
| Security | No direct DB or AWS access; all through CI/CD and IAM. |
| Auditability | Git history + pipeline logs + migration table provide full traceability. |
| Automation | Repeatable, deterministic deployments via CI/CD. |
| Developer Focus | Engineers write SQL; platform runs it safely. |
| Consistency | Same process across all environments and services. |

# Design / Solution Options

Two primary tools were considered: Flyway and Liquibase.

## Flyway vs Liquibase Comparison

|  |  |  |
| --- | --- | --- |
| Feature | Flyway | Liquibase |
| Syntax | SQL-first (simple). Java callbacks optional. | SQL, XML, YAML, JSON changesets (flexible, more to learn). |
| Rollback | Undo scripts (`U\_\_\*`) or roll-forward; Pro adds more. | Built-in rollback commands/tags. |
| Dry-run | Yes (`-dryRunOutput`). | Yes (`updateSQL`). |
| Diff/Generate | Partial (Pro or via external tools). | Strong diff/changelog generation. |
| Learning Curve | Low for SQL-proficient teams. | Higher due to DSL/changeset model. |
| Operational Footprint | Lightweight CLI; minimal schema table. | Richer features; more config surface. |
| Community & Ecosystem | Large OSS community; commercial support available. | Large OSS community; strong commercial support. |
| Best Fit | Git-based SQL migrations, simple process, many teams. | Complex cross-DB use cases, autogenerated changelogs. |

## Option A — Flyway (Recommended)

• SQL-first workflow aligns with developer skills and code reviews.

• Deterministic ordering with simple naming; easy adoption across many teams.

• Lightweight CLI integrates cleanly with Harness shell steps and containers.

• Supports versioned, repeatable, and undo scripts; dry-run output for review.

• Good Aurora PostgreSQL support; small operational surface area.

## Option B — Liquibase

• Powerful diff and changelog generation; rich rollback features.

• Supports multiple change definition formats (SQL, XML, YAML, JSON).

• Best when teams need auto-generation, complex rollbacks, or heterogeneous databases.

# Recommendation

Adopt Flyway as the standard migration tool. Rationale: SQL-first approach matches team skills and PR review culture; simpler tooling and lower learning curve accelerate adoption; lightweight CLI reduces pipeline complexity; features like repeatable and undo scripts are sufficient for our needs. Liquibase remains an approved alternative for exceptional use cases requiring rich diff/rollback automation or multi-DB heterogeneity.

# Follow-Up Decisions

|  |  |  |
| --- | --- | --- |
| Decision | Status | Next steps |
| Publish Flyway pipeline templates in Harness | IN REVIEW | Provide CI validation & CD apply steps; sample container image |
| Enforce no direct DB access in PROD | APPROVED | IAM/SecOps changes; SG rules; bastion removal where possible |
| Create SQL lint + naming pre-commit hooks | IN REVIEW | Add to language starter repos |
| Document rollback playbook & PITR | IN REVIEW | DBA + Platform draft and test |
| Evaluate Liquibase for complex legacy migrations | PENDING | Pilot if needed on a candidate service |

# References

• Redgate Flyway Documentation

• Liquibase Documentation

• Harness Developer Hub — Pipelines & Secrets