Database Management via Harness CI/CD and Flyway/Liquibase

# Overview

This document outlines a secure and developer-friendly approach to manage AWS Aurora PostgreSQL database schemas using Harness pipelines and Flyway or Liquibase, without giving developers direct AWS Console or RDS access.

# Goals

- Restrict direct AWS Console and DB access

- Enable developers to submit DDL/DML changes safely

- Automate database changes using CI/CD

- Maintain schema version control and audit history

# Access Control Model

| Role | Access Type |

|-------------|-------------|

| Developer | Git access only (DDL/DML scripts) |

| CI/CD | Privileged role to run migration scripts |

| DBA/DevOps | Manage Harness and Secrets |

# High-Level Architecture

Developer -> Git push with SQL

Source Control (e.g. GitHub) ->

Harness CI Pipeline (Validate SQL / Dry-run) ->

Harness CD Pipeline (Apply migrations) ->

Aurora PostgreSQL (AWS RDS)

# Repository Structure

/db-migrations/

├── V1\_\_initial\_schema.sql

├── V2\_\_add\_customers\_table.sql

├── V3\_\_fix\_order\_column.sql

└── ...

# Developer Workflow

1. Create SQL file for schema or data changes

2. Follow versioned naming convention (e.g., V4\_\_add\_invoice\_table.sql)

3. Submit PR for review

4. Upon merge, Harness pipeline is triggered to validate and apply changes

# CI Pipeline: Validation Stage

This stage performs:

- SQL syntax validation

- Dry-run using Flyway or Liquibase

- Optional: naming convention check or static analysis

Example Flyway Dry Run:

flyway -url=jdbc:postgresql://dummy-host/db -user=dummy -password=dummy -locations=filesystem:./db-migrations -dryRunOutput=preview.sql migrate

# CD Pipeline: Migration Execution

Steps:

1. Use IAM Role or Secrets Manager for DB credentials

2. Execute Flyway or Liquibase CLI using a Harness shell step

3. Monitor logs and notify on success/failure

Example Flyway Command:

flyway -url=jdbc:postgresql://${DB\_HOST}:${DB\_PORT}/${DB\_NAME} -user=${DB\_USER} -password=${DB\_PASS} -locations=filesystem:./db-migrations migrate

# Rollback Strategy

Flyway: Manual undo scripts or Pro version

Liquibase: Built-in rollback XML/YAML support

Example Liquibase Rollback:

<changeSet id="1" author="alice">

<createTable tableName="users">

<column name="id" type="int"/>

<column name="name" type="varchar(255)"/>

</createTable>

<rollback>

<dropTable tableName="users"/>

</rollback>

</changeSet>

# Flyway vs Liquibase Comparison

| Feature | Flyway | Liquibase |

|----------------------|--------------|-----------------|

| Syntax | SQL only | SQL, XML, YAML |

| Ease of Use | Very easy | Moderate |

| Rollback | Manual/Pro | Built-in |

| Dry-run Support | Yes (Pro) | Yes |

| Diff Support | Limited | Full |

# Tools Checklist

- [ ] Git repository with db-migrations/ folder

- [ ] Harness CI/CD Pipelines

- [ ] Flyway or Liquibase CLI in pipeline

- [ ] Secrets stored securely

- [ ] Aurora PostgreSQL created via pipeline

# Environment Promotion Flow

1. Dev -> QA -> Prod

2. Each stage uses the same pipeline with different configs

3. Track applied migration versions

# Developer Enablement Tips

- Provide SQL templates

- Add pre-commit hooks for validation

- Build dashboards or alerts in Harness

- Offer a local dry-run DB instance

# Benefits

| Benefit | Description |

|-----------------|----------------------------------------|

| Security | No direct DB or AWS access |

| Auditability | Git and Harness track all changes |

| Automation | Fully automated via CI/CD |

| Developer Focus | Devs write SQL, not manage infra |

| Consistency | Schema consistency across environments |

# References

- https://flywaydb.org/documentation/

- https://www.liquibase.org/docs

- https://developer.harness.io/docs/