Database Migration Tools – Analysis & CI/CD Integration

1.1 Tool Overview and Comparison

Flyway

Flyway is an open-source database migration tool focused on simplicity and reliability. It follows a versioned migration approach using SQL or Java-based scripts and is popular for managing database changes across multiple environments. Flyway integrates seamlessly into CI/CD pipelines.

Microsoft Entity Framework Core (EF Core)

EF Core is a modern, open-source Object-Relational Mapper (ORM) from Microsoft, designed for .NET developers. In addition to mapping database tables to C# classes, EF Core provides powerful migration capabilities via code-based, model-driven migrations.

Comparison Table

Criteria	Flyway	EF Core
Ease of Use	Simple setup; intuitive for DBAs and DevOps teams	Developer-friendly for .NET projects; steep learning curve for non-.NET users
Integration with CI/CD	Native support in CLI; plugins for build tools	Tight integration with .NET pipelines (e.g., Azure DevOps, GitHub Actions)
Supported Databases	Broad support including PostgreSQL, MySQL, Oracle, SQL Server, etc.	Limited to supported providers (SQL Server, PostgreSQL, MySQL, SQLite, etc.)

1.2 Integration Strategy for CI/CD Pipelines

To integrate both Flyway and EF Core into a CI/CD pipeline.

CI/CD Pipeline Automation Steps

Using a platform like GitHub Actions or Jenkins:

1. **Pre-Commit (Dev):**
    - Developers run `dotnet ef migrations add` to create new migration classes.
    - `dotnet ef migrations script` is used to generate SQL migration files.
    - Files are committed to the repository under a `migrations` directory.

2. **CI Pipeline (Testing/Staging):**
    - On every merge or pull request:
        - Run unit tests and integration tests using EF Core in the test DB.
        - Validate and apply Flyway migration scripts using `flyway migrate` against the staging DB.
        - Use `flyway validate` to ensure the migration history is consistent.
3. **CD Pipeline (Prod):**
    - Once approved, run Flyway commands in the deploy stage to apply SQL migrations to production.
    - Use `flyway info` for audit and traceability.
    - Enable backup and pre-checks before applying changes.

Versioning and Rollback

- **Versioning:** Adopt a semantic versioning scheme for migration filenames (e.g., `V1.2__add_orders_table.sql`). EF Core migration timestamps should be clearly documented and mapped to Flyway versions.
- **Rollback:** For EF Core, maintain explicit `Down()` methods in migration classes. For Flyway, optionally implement undo scripts (`U1.2__remove_orders_table.sql`) where feasible. Backup production databases before applying irreversible changes.

This hybrid strategy uses EF Core for developer productivity and model-driven migration generation, while utilizing Flyway's strengths in deployment control, auditing, and environment consistency.