# Prompt-Driven Microservice & Database Evolution Using LLMs for Automated Schema Migration and DevOps Delivery

By Niraj Katkamwar

Intuit

Conf42 Database DevOps 2026

The Problem

# Traditional Change Management Can't Keep Pace

## Schema Drift

Application and database changes evolve out of sync, breaking contracts across service boundaries.

## Manual Bottlenecks

Engineers hand-craft migrations, API updates, and rollback scripts  slowing every release cycle.

## Ambiguous Requirements

Business intent gets lost in translation between stakeholders, developers, and data engineers.

## Deployment Risk

Unvalidated schema changes in distributed systems cause outages, data loss, and rollback failures.

# A New Paradigm: Define Requirements in Natural Language

Instead of writing migration scripts and boilerplate code by hand, engineers express intent in plain English. The framework handles the rest from schema evolution to API generation to test scaffolding.

Session Agenda

# What We'll Cover Today

01

## The Architecture

Prompt interpretation, semantic validation, and type-safe code generation

02

## Database-Aware Transformation

Automated schema evolution with backward compatibility guarantees

03

## Unified DevOps Pipeline

Aligning application and database changes in a single delivery flow

04

## Safe Releases

Automated test generation and rollback-aware migration strategies

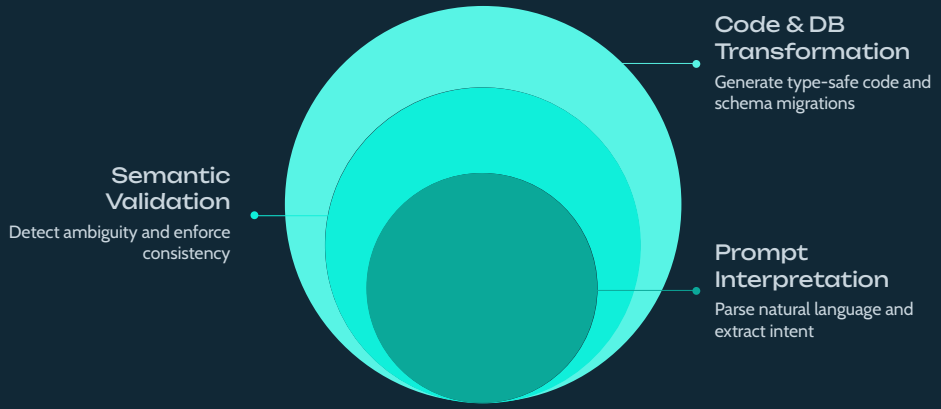
05

## Practical Integration

Fitting LLM-driven automation into your existing pipelines today

Core Architecture

# The Prompt-Driven Framework




Each layer has a discrete responsibility, ensuring that errors are caught early before they propagate into production migrations or API contracts.

## Prompt Interpretation

Extracts structured intent from free-form natural language input

## Semantic Validation

Detects ambiguity and enforces cross-service consistency before generation

## Code + Schema Generation

Produces type-safe service code and database-aware migration artefacts

Layer 1

# Prompt Interpretation & Semantic Validation

## Intent Extraction

The LLM parses natural language to identify entities, relationships, constraints, and change type add, modify, or deprecate.

## Ambiguity Detection

Conflicting or underspecified requirements are flagged before any code or migration is generated, preventing downstream failures.

## Consistency Enforcement

Cross-service contracts and shared data models are validated against the existing schema registry before proceeding.

Layer 2

# Type-Safe Code Generation

The framework generates production-ready microservice code, including:

- **API contracts** — endpoint definitions, request/response models, versioning strategies
- **Data access layers** — repository classes aligned with the target schema
- **Service logic stubs** — scaffolded business logic ready for engineer review
- **Integration tests** — automatically generated test cases covering happy paths and edge conditions

LAYER 3

# Database-Aware Schema Transformation

Schema evolution is the highest-risk step in any microservice change. The framework addresses this with a multi-guarantee transformation layer.

- ## Backward Compatibility

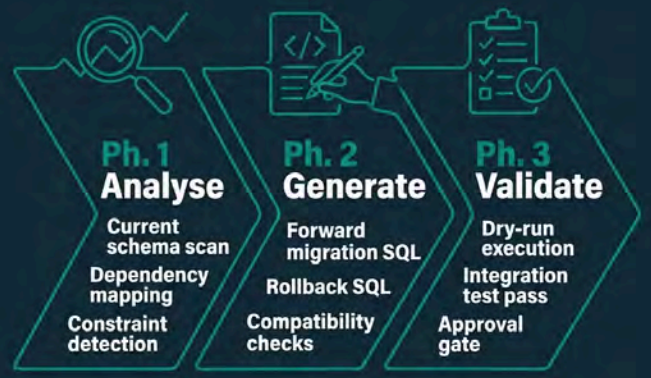
  Generated migrations preserve existing API contracts and consumer expectations across service versions.

- ## Data Integrity

  Constraint analysis prevents destructive operations  column drops, type changes without explicit human approval gates.

- ## Rollback Plans

  Every migration artefact ships with a tested, version-pinned rollback script generated at the same time as the forward migration.



Unified Pipeline

# Application and Database Changes in a Single Flow

The most dangerous pattern in Database DevOps is deploying application code and schema changes through *separate, uncoordinated pipelines*. This framework eliminates that gap.

- **Prompt Submitted**

  Engineer or stakeholder submits a natural language requirement

- **Artefacts Generated**

  Service code, API spec, migration, and tests produced atomically

- **Validation Gate**

  Semantic checks, schema dry-run, and integration test pass required

- **Coordinated Deploy**

  Application and database changes released together with full rollback coverage

Safety Mechanisms

# Built-In Validation Before Every Deployment


Every generated artefact must clear a four-layer validation pyramid before reaching the deployment pipeline. Failures at any layer halt the pipeline and return structured feedback to the engineer.

- **Schema syntax** — parsed and linted before execution
- **Semantic consistency** — cross-service contracts verified against the registry
- **Dry-run execution** — migration applied against a shadow database clone
- **Integration tests** — auto-generated tests must pass in CI before promotion


Rollback Strategy

# Rollback-Aware Migrations in Distributed Systems

## Atomic Artefact Pairs

Every forward migration is generated alongside its inverse. The two are version-pinned and stored together rollback is never an afterthought.

## Expand/Contract Pattern

Additive changes first, destructive changes only after consumer adoption is confirmed preserving service reliability during multi-phase rollouts.

## Automated Rollback Trigger

Health check failures post-deployment automatically initiate the rollback script — no manual intervention required in the critical window.

Collaboration Impact

# Closing the Gap Between Stakeholders and Engineers

One of the most underestimated costs in feature delivery is the translation overhead between business intent and technical implementation.

### Business Stakeholders

Express requirements in natural language — no SQL or API design knowledge required

### Developers

Review and approve generated code rather than author boilerplate from scratch

### Data Engineers

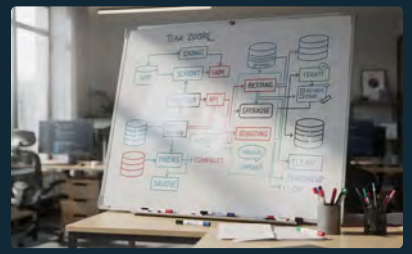
Focus on governance and optimisation instead of hand-crafting routine migrations

Practical Integration

# Fitting LLM Automation into Your Existing Pipeline

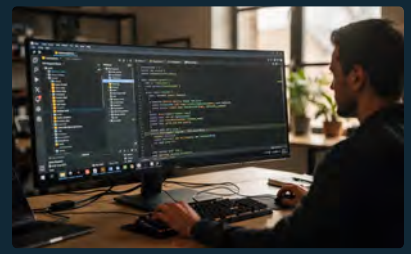
## CI/CD Integration

The framework exposes a CLI and REST API  drop it into any Jenkins, GitHub Actions, or GitLab pipeline as a generation and validation stage.



## Schema Registry Compatibility

Works alongside Confluent Schema Registry, AWS Glue, or custom registries consuming and publishing schema versions without disrupting existing governance.



## Human-in-the-Loop Gates

Approval gates at each layer keep engineers in control. LLMs accelerate generation; engineers own the decision to promote.

# Key Takeaways

- **Natural language as the interface**

  Requirements expressed in plain English drive the full stack  service code, APIs, migrations, and tests  through a single validated flow.

- **Safety is non-negotiable**

  Semantic validation, schema dry-runs, and atomic rollback artefacts ensure that speed gains do not come at the expense of system reliability.

- **Fits your pipeline today**

  CLI and API integration points make adoption incremental  no need to rebuild existing DevOps toolchains to realise meaningful gains.

# Thank You!