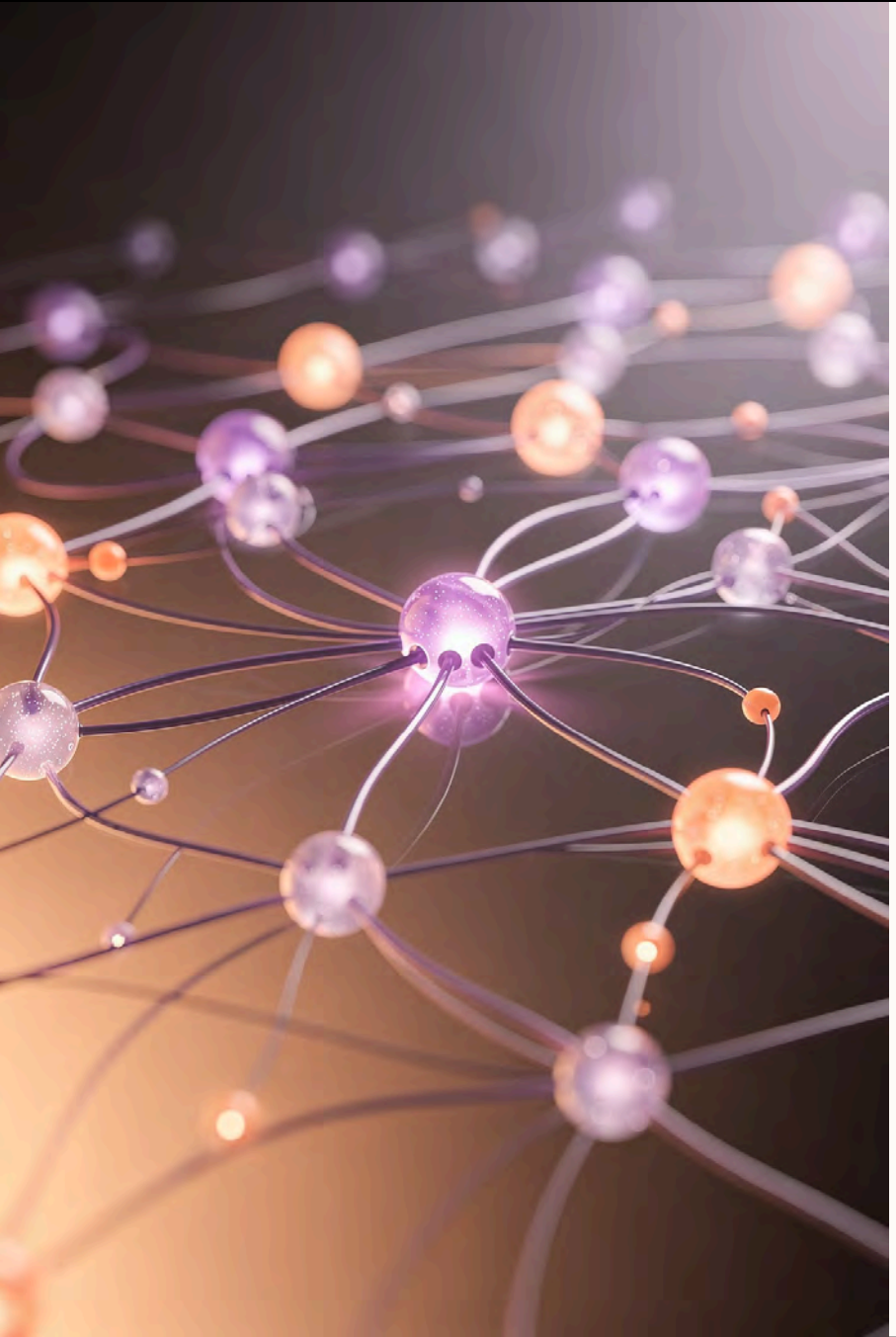


Operationalizing Semantic Consistency: A DevOps Framework for Reliable Cross-Engine Analytics

Muthupalaniappan Ramanathan · Senior Software Developer, DoorDash

The Multi-Engine Reality

Modern data platforms don't run on a single engine: they run on many. Performance, scalability, and workload diversity drive teams to adopt Spark, Trino, Snowflake, and more in parallel.

Performance

Different engines excel at different query shapes and data volumes

Scalability

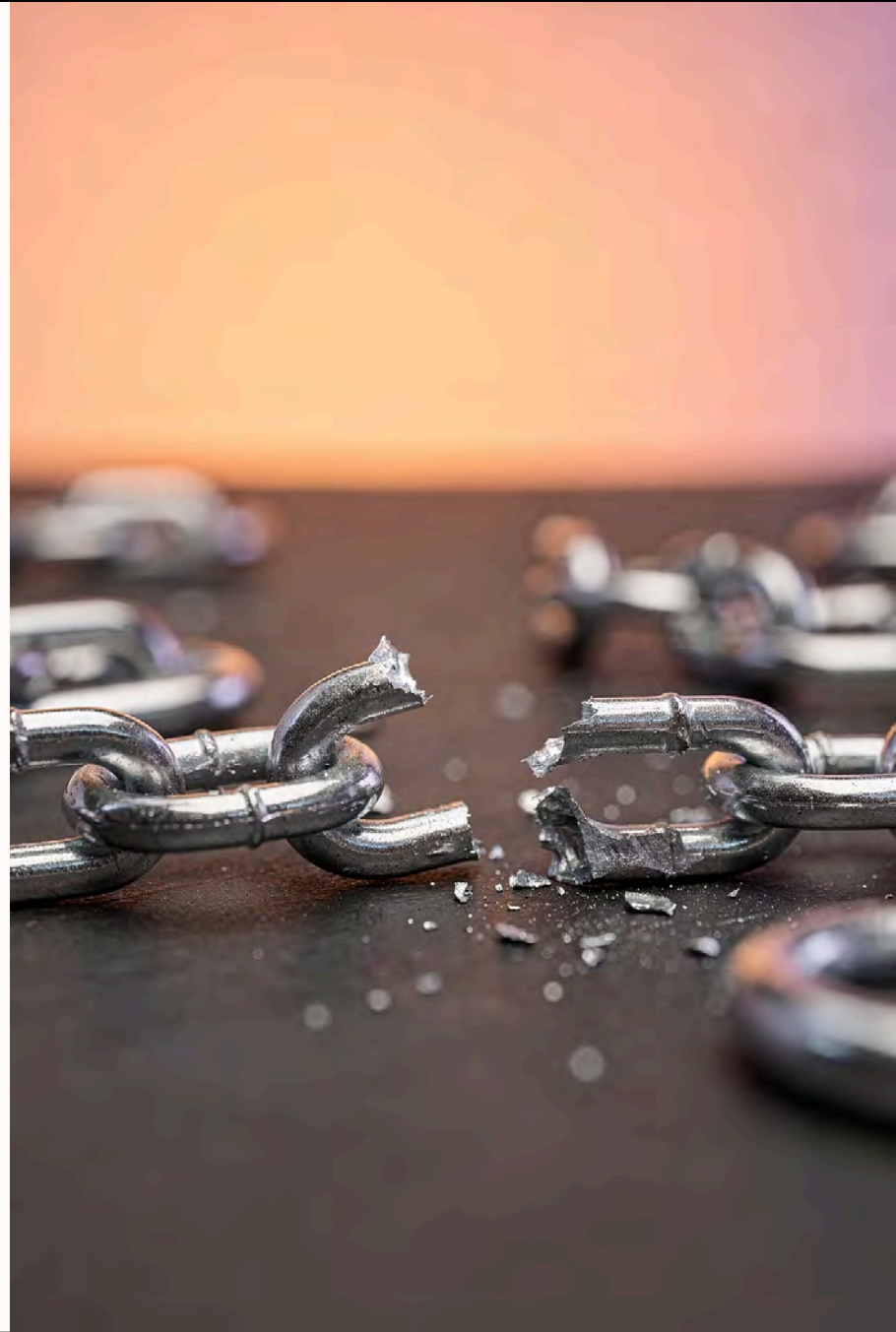
Distributed compute needs vary by team, latency SLA, and data size

Workload Diversity

ETL, ad-hoc exploration, and dashboards each favor different runtimes

SQL Compatibility ≠ Semantic Consistency

A query that compiles and runs on every engine is **not** a query that returns the same answer on every engine. This is the core problem.



Where Divergence Hides

Subtle engine-level differences accumulate into meaningful analytical drift. These aren't bugs, they are engine design choices that compound silently.

Type Handling

Implicit casts, numeric precision, and integer overflow behave differently across engines

Null Semantics

NULL propagation in aggregations, joins, and comparisons varies in subtle but impactful ways

Aggregation Behavior

Floating-point summation order, DISTINCT counts, and GROUP BY edge cases diverge

Optimizer Decisions

Query rewrites and execution plan choices can alter result ordering and row inclusion

The Same Query, Three Answers

Across Spark, Trino, and Snowflake, an identical SQL string can produce divergent results when engine-specific semantics differ in type coercion, null handling, or aggregation precision.

⚠ This creates silent data reliability failures that standard SQL linting and schema tests will never catch.

Core Concept

Introducing Analytical Intent as a First-Class Abstraction

Rather than reasoning about SQL syntax, we reason about **what the query is supposed to mean**, independent of which engine executes it. Analytical intent becomes the source of truth.

1

Intent Definition

Declare the logical meaning of a query in engine-agnostic terms

2

Semantic Layer

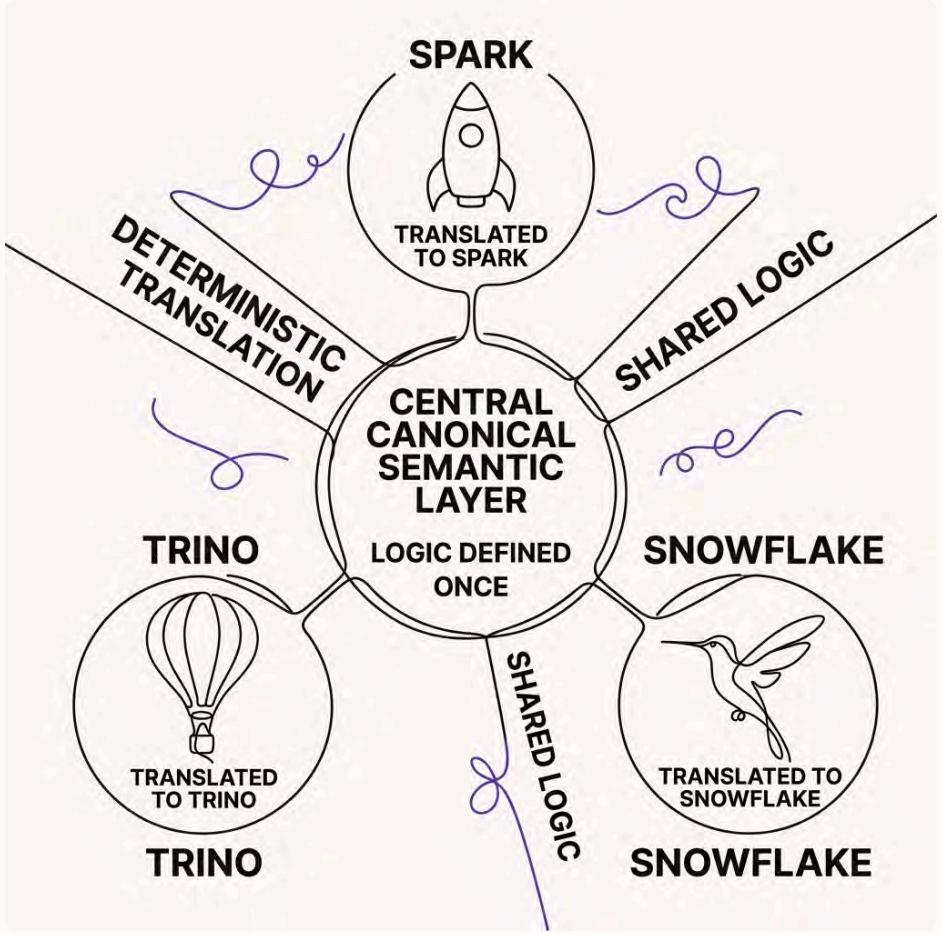
Translate intent into engine-specific SQL with deterministic rules

3

Validated Execution

Run on target engine with guaranteed semantic equivalence

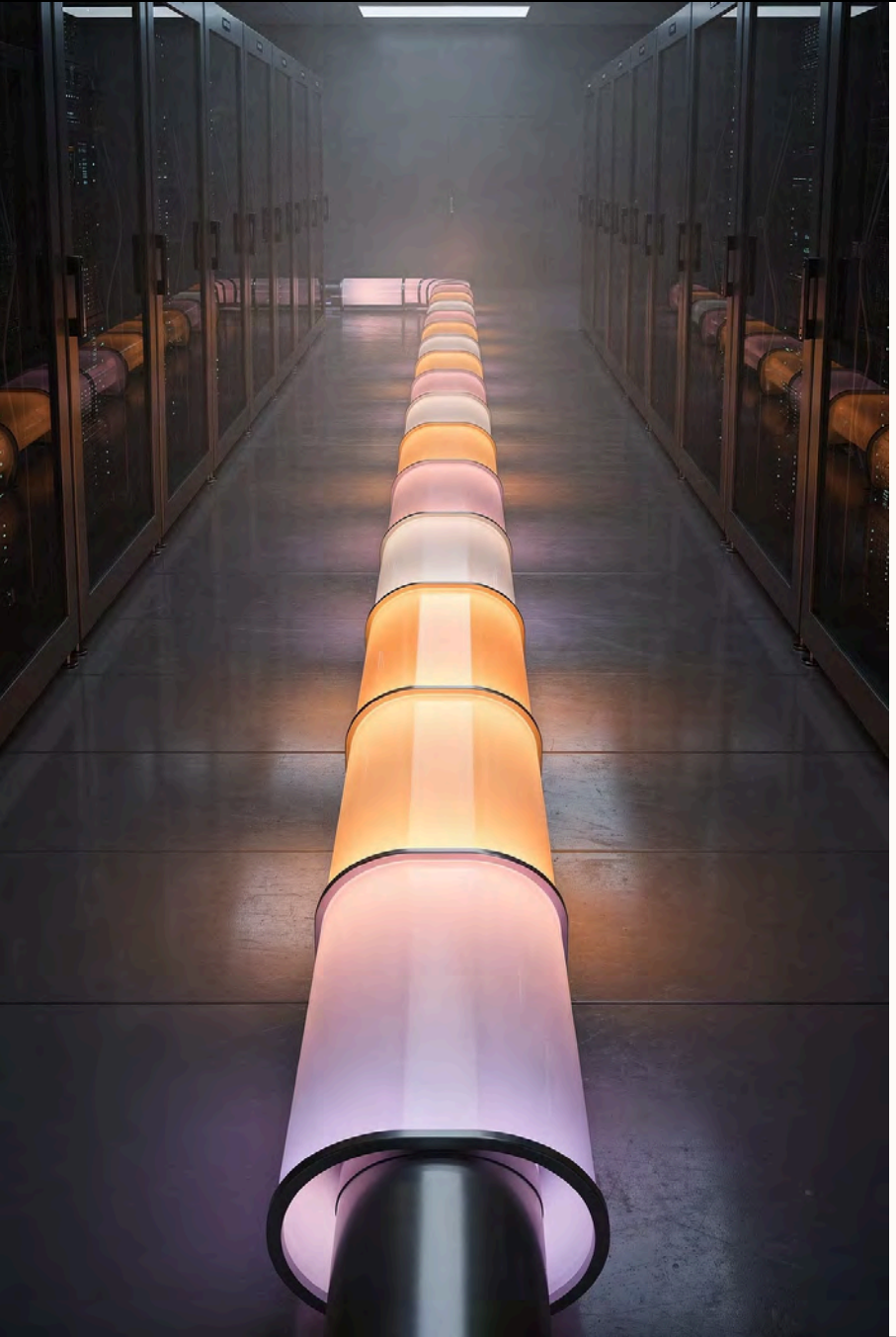
The Canonical Semantic Layer



Decouple Logic from Execution

The canonical semantic layer acts as a single definition point for analytical logic. Engine-specific dialects are generated from it, never authored directly.

- One source of logical truth
- Deterministic query translation per engine
- Eliminates hand-crafted per-engine SQL drift

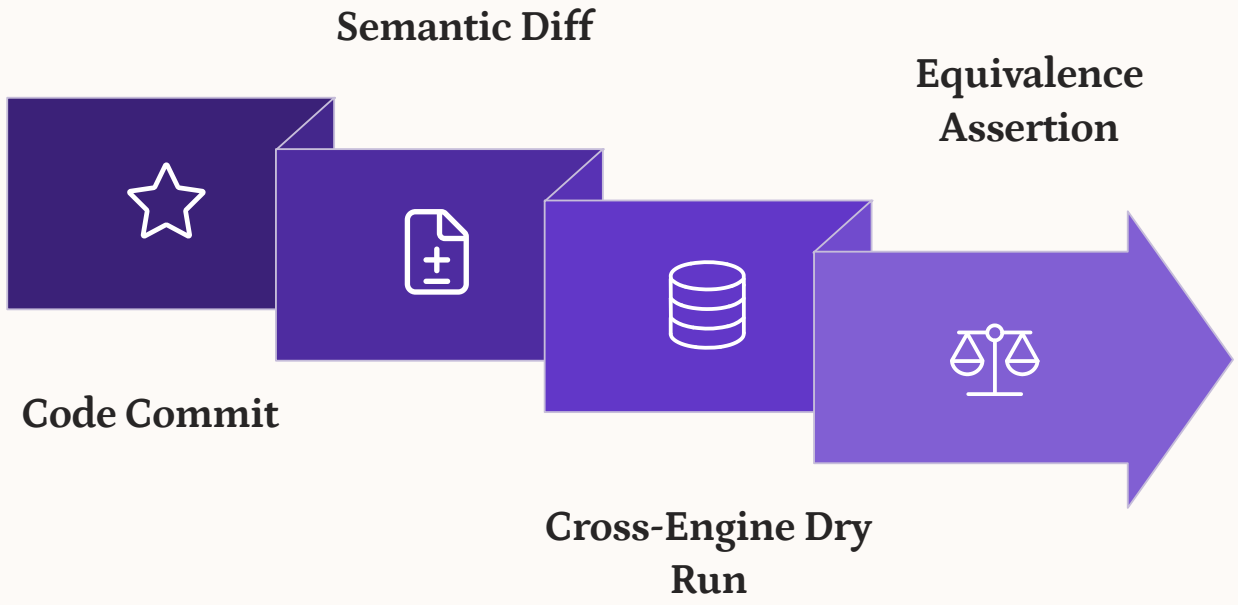


Chapter 2

CI/CD-Integrated Validation Workflows

Semantic guarantees must be enforced at commit time, not discovered in production dashboards.

Embedding Validation in the Pipeline



Every schema or logic change triggers a cross-engine equivalence check before merging. Semantic drift is caught as a PR blocker, not a data incident.

Semantic Testing Strategies

Standard data tests validate shape and freshness. Semantic tests validate **meaning**. These require a distinct testing vocabulary.

1

Equivalence Tests

Assert that the same logical query returns identical results across engine pairs for a reference dataset

2

Null Boundary Tests

Explicitly test NULL propagation paths that are statistically rare in production data but semantically critical

3

Precision Regression Tests

Validate that numeric aggregations stay within acceptable tolerance bounds across engine upgrades

4

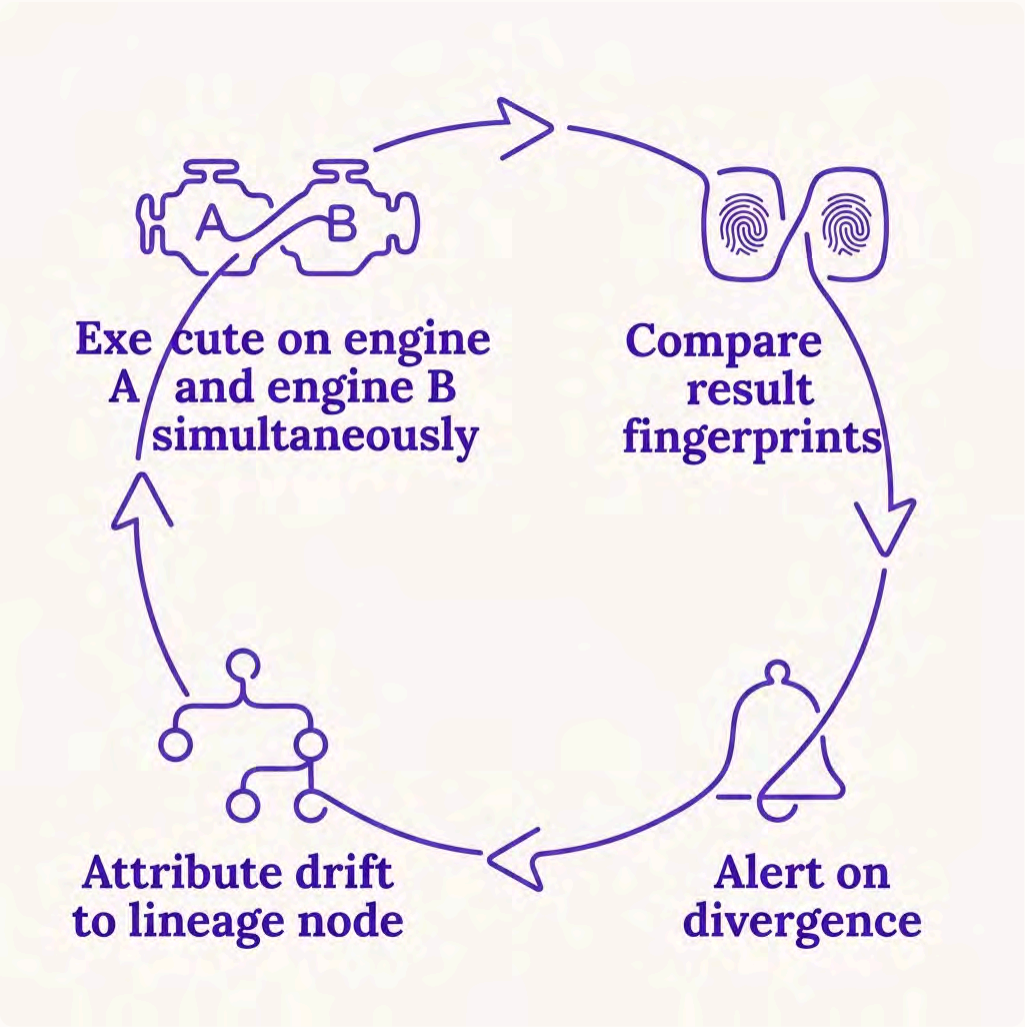
Translation Snapshot Tests

Pin the generated SQL output per engine to detect unintended changes in the translation layer

Observability Patterns for Semantic Drift

Detection doesn't stop at merge time. Production pipelines need runtime observability to surface semantic drift before it propagates into downstream metrics.

- Cross-engine result fingerprinting on scheduled queries
- Metric divergence alerting across engine replicas
- Lineage-aware drift attribution to isolate root cause



Trust as an Operational Property

Reproducibility and trust are not features you add: they are properties you enforce through process, tooling, and cultural commitment to semantic rigor.

Operational Resilience Outcomes

Teams that embed semantic guarantees into their data platform infrastructure see measurable improvements across the reliability spectrum.

Reproducibility

Queries produce the same answer regardless of which engine is selected at execution time

Early Detection

Drift is caught in CI before reaching production, reducing costly data incident investigations

Governance

A canonical semantic layer creates an auditable, version-controlled record of analytical logic

Framework Summary

The Three Pillars

- **Canonical Semantic Layer**, decouple logic from execution, enable deterministic translation
- **CI/CD Validation**, enforce equivalence at merge time across target engines
- **Runtime Observability**, detect and attribute drift before downstream impact

What This Enables

- Heterogeneous engine adoption without sacrificing analytical correctness
- Semantic guarantees as a testable, versionable artifact
- Data platform teams that ship with confidence across engine migrations and upgrades

Key Takeaways

1 **SQL compatibility is necessary but not sufficient**

Engine-level semantic differences create silent analytical divergence

2 **Analytical intent must be a first-class abstraction**

Decouple logical meaning from execution dialect using a canonical semantic layer

3 **Semantic guarantees belong in your CI/CD pipeline**

Test for equivalence, pin translations, and monitor for runtime drift continuously

Thank You

Muthupalaniappan Ramanathan

Senior Software Developer, DoorDash

Questions? Let's discuss.