Rapido Platform Layer Analysis and Recommendations

TAB 2 : Rapido Platform Layer - Analysis and Recommendations

Data Platform Analysis

Each ride, click, cancellation, and payment generates vast amounts of data.

To manage and make sense of this data, the team at Rapido:

- Collects, cleans, and organizes billions of data points daily.
- Enables product managers, analysts, and business teams to quickly access insights and reports—without overloading systems or driving up costs.

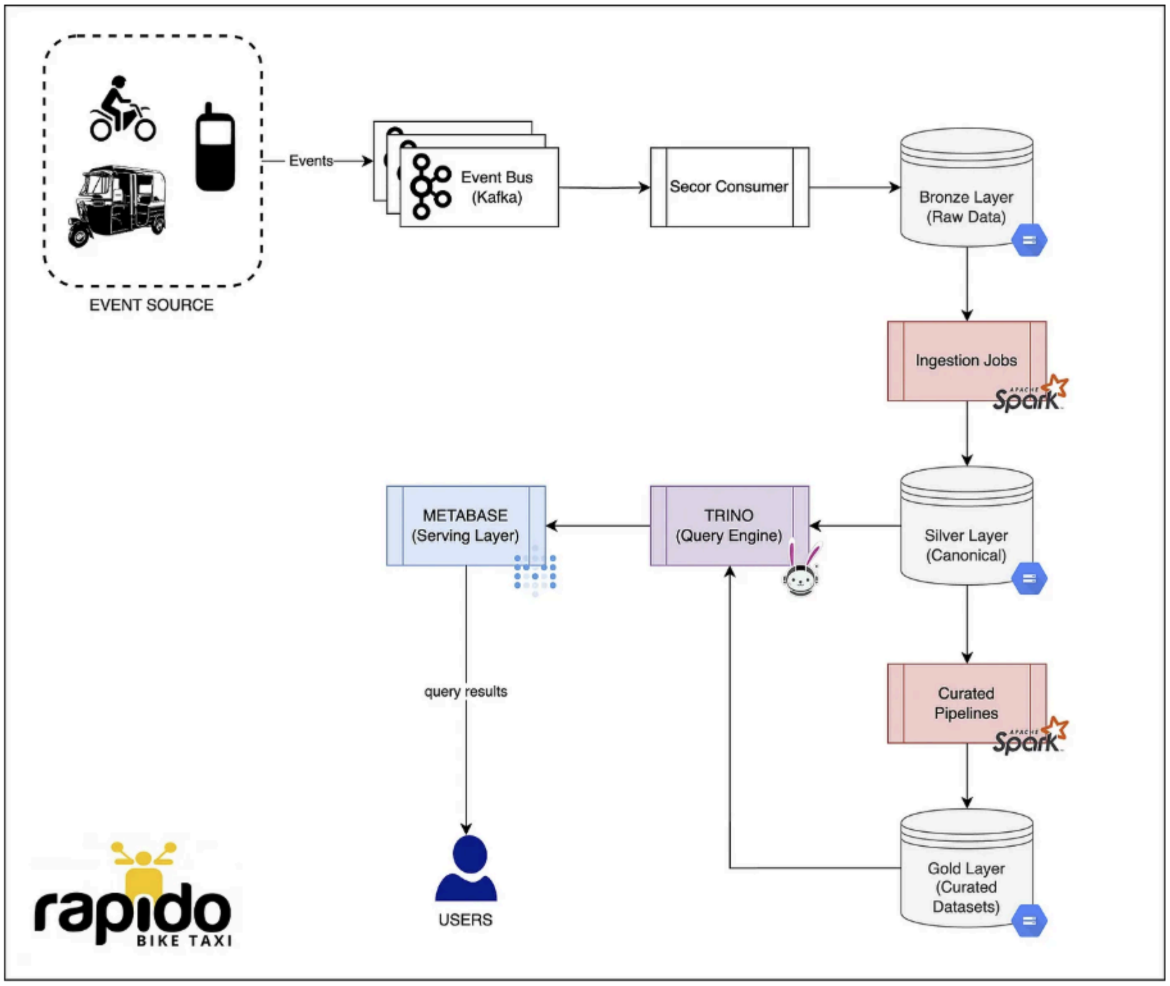
How Rapido's Data Platform Works

Rapido's data infrastructure is designed in three layers:

1. **Bronze (Raw) Layer** – Ingests millions of events daily through Kafka, persisted in the warehouse with an "at least once" guarantee. This raw data captures the full fidelity of user and system activity.

2. **Silver (Cleaned) Layer** – Automated pipelines process and clean this raw data, correcting formats, handling schema issues, and ensuring consistency for downstream use.

3. **Gold (Curated) Layer** – Further pipelines aggregate, enrich, and transform silver datasets into highly curated, business-ready datasets used for analytics and reporting.



Rapido Architecture

Challenge 1: Query Overload

With many users sending queries—from small operational checks to large analytical workloads—everything went through a single, overloaded server.

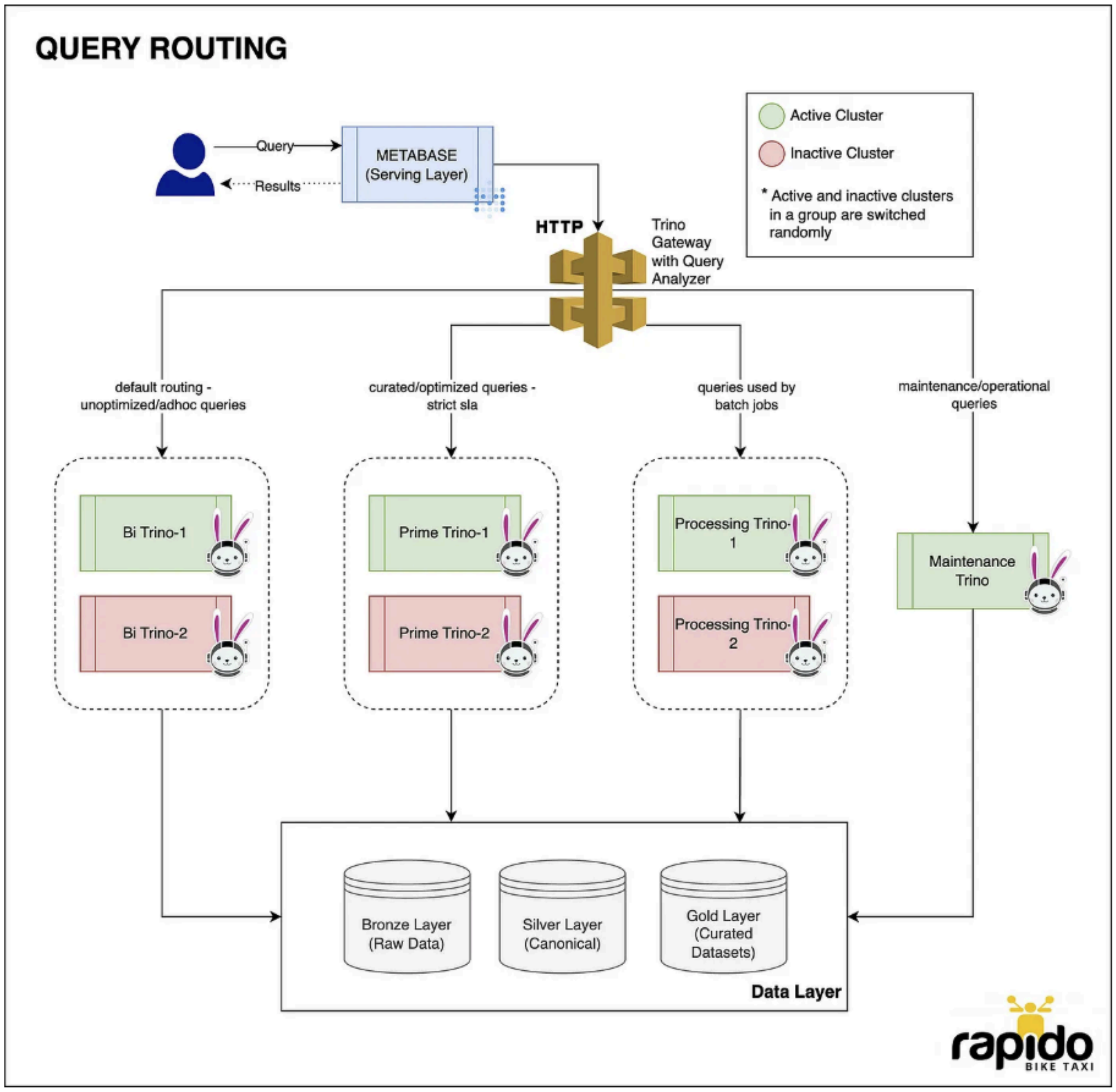
Problem: The centralized system became a bottleneck, slowing down or failing under heavy load.

Solution: Rapido implemented four separate data clusters:

- One for product managers (lightweight reports)
- One for analysts (heavy data exploration)

- One for batch processes (scheduled data tasks)
- One for system maintenance

They introduced a **smart Trino Gateway** that automatically routes queries to the appropriate cluster, ensuring balanced performance without user intervention.


QUERY ROUTING
Query
Results
METABASE (Serving Layer)
HTTP
Trino Gateway with Query Analyzer
Active Cluster
Inactive Cluster
* Active and inactive clusters in a group are switched randomly
default routing - unoptimized/adhoc queries
curated/optimized queries - strict sla
queries used by batch jobs
maintenance/operational queries
Bi Trino-1
Bi Trino-2
Prime Trino-1
Prime Trino-2
Processing Trino-1
Processing Trino-2
Maintenance Trino
Bronze Layer (Raw Data)
Silver Layer (Canonical)
Gold Layer (Curated Datasets)
Data Layer
rapido BIKE TAXI

[Trino Gateway]

Challenge 2: High Infrastructure Costs

Running multiple clusters 24/7 was proving too expensive.

Solution: The team switched to **spot VMs** (cost-effective, short-term cloud servers) and adopted a rotating cluster strategy. Clusters are gracefully shut down and restarted every few hours, resulting in **33% cost savings with no downtime.**

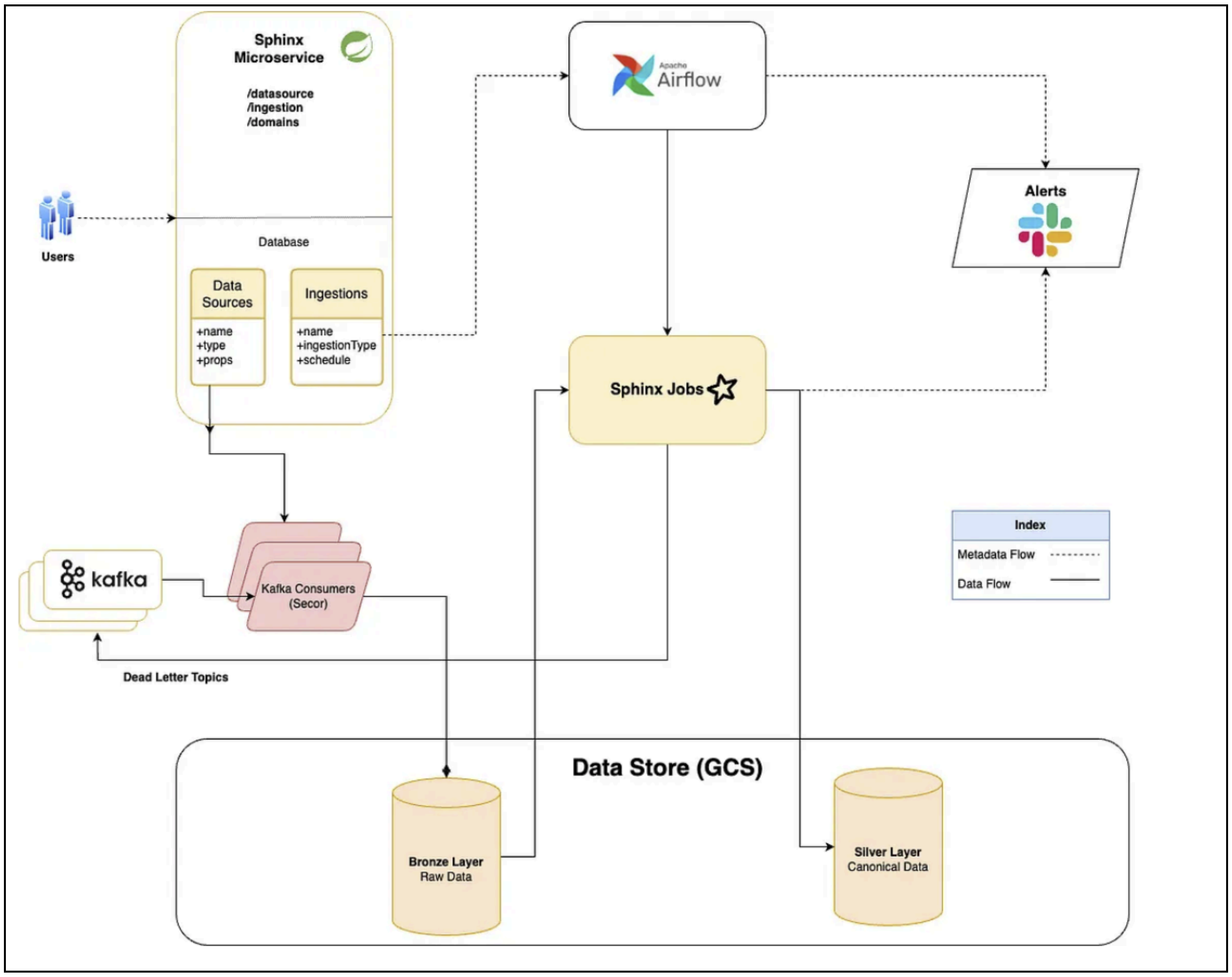
Challenge 3: Complex Ingestion Pipelines

Previously, data ingestion (from sources like Kafka or files) required manual, YAML-based configuration—difficult to maintain, error-prone, and slow to approve.

Solution: SPHINX — A Self-Serve Ingestion Platform

SPHINX enables teams to set up and manage data pipelines through APIs. It:

- Validates incoming data for quality and usability
- Provides clear alerts with specific error messages
- Automatically handles schema changes and data formatting issues
  Transforms nested or complex data into flat, analysis-ready tables



Sphinx Platform

Business Impact of SPHINX

- Over 200 pipelines deployed in just 3 months
- 98% reduction in time to production
- Over 2 billion events ingested daily

Technical Advancements

- Auto-scaling infrastructure that adjusts based on load
- Intelligent query handling that prevents system overload
- Real-time data transformation for better analytics

- Detailed error reporting to resolve issues proactively

Future plans

- A no-code interface for pipeline creation and monitoring
- Real-time data ingestion capabilities within SPHINX
- Integration with Rapido’s internal data catalog for better data discovery and usage

Suggestions And Recommendations

In this section, I have outlined potential future pain points Rapido may encounter in scaling and evolving its data platform, along with assumptions/conditions where needed, and recommendations tailored to their current architecture.

1. Problem: Ride Count on Dashboard is Wrong or Missing

A PM opens the dashboard in the morning and sees 0 rides for the last hour, even though there were lakhs of rides.

Where's the Problem?

- **Backend:** The pipeline that processes ride events (from Bronze to Silver) failed.
- **Database:** The Gold dataset is empty or outdated.
- **Frontend:** Dashboard just shows blank/zero without explanation.

Why This Happens?

- A field in Kafka message changed (e.g., ride_status field was removed or renamed).
- Pipeline didn't catch it and silently failed or skipped rows.

What's Needed?

- UI should show: "Data not updated since 6:00 AM – pipeline failed due to schema mismatch."
- DB/pipeline should have alerts and auto-pausing if schema changes break processing.

Recommendation:

- Add **pipeline health indicators on dashboards** (green = healthy, red = broken).
- Use **data contracts** so backend producers can't accidentally break schema.
- Auto-alert teams when data is late or incorrect.

2. Problem: Same Metric, Different Numbers in Different Dashboards

Ops team sees “active riders = 10,000” but Finance sees “active riders = 9,200” for the same date.

Where’s the Problem?

- **Backend:** Different teams built separate pipelines for “active riders.”
- **Database:** Duplicated Gold datasets with slight variation in logic.
- **Frontend:** Each dashboard is connected to different datasets.

Why This Happens?

- No centralized definition of metrics.
- Teams independently built logic to count active users with slightly different filters.

What’s Needed?

- One single definition of “active rider” used across all tools.
- Central metrics layer in the data platform.

Recommendation:

- Create a **central metrics dictionary** with business-approved definitions.
- In dashboards, only allow use of certified metrics from this layer.
- Build re-usable data models in dbt or LookML to avoid duplication.

3. Problem: PM Wants to Create New Report but Has No Idea What Data Exists

A PM wants to track “daily cancellations due to driver no-shows” but doesn’t know if this data even exists.

Where’s the Problem?

- **Frontend:** No interface to explore available datasets.
- **Backend:** Data might exist in Bronze or Silver but is not discoverable.

- **Database:** No tags, descriptions, or owner info on tables.

Why This Happens?

- No easy way to search or understand existing data.
- Too much tribal knowledge in data teams.

What's Needed?

- A UI that lets users explore what data exists, see sample rows, check freshness, and contact owner.

Recommendation:

- Build a **data catalog UI** like Google Search for data:
  - Search by keyword: "no-show"
  - See table name, when it was last updated, how it's built, and contact info.
- Auto-tag tables by domain: Rides, Payments, Users, etc.

4. Problem: Real-Time Use Case But Data is Always Delayed

You want to show surge pricing triggers based on current demand, but the data you're using is 2 hours old.

Where's the Problem?

- **Backend:** Pipelines are running in hourly batch mode.
- **Database:** Gold tables are updated only a few times a day.
- **Frontend:** Surge pricing UI is using stale data.

Why This Happens?

- Current pipelines are batch-oriented.
- No real-time ingestion or transformation yet.

What's Needed?

- Data should be ingested and processed in real-time (within seconds).

Recommendation:

- Upgrade Sphinx to support **real-time ingestion using Kafka streams or Flink**.
- Create a **real-time Gold layer** for operational use cases like:
  - Surge pricing
  - Fraud alerts
  - Live ops dashboards

5. Problem: Sensitive Data Shown to Wrong Teams

An intern runs a query and gets access to rider phone numbers or payment method details.

Where's the Problem?

- **Database:** No row-level or column-level access controls.
- **Frontend:** No filters on what data different users can query.

Why This Happens?

- Data is open to all users in Trino or warehouse.
- No clear roles, ownership, or access policies.

What's Needed?

- Fine-grained access control at dataset and column level.

Recommendation:

- Implement **role-based access control**:
  - Interns can only see anonymized data.
  - Finance can access payment details.

- Use tools like Unity Catalog or Apache Ranger to enforce this.

Summary

Situation	Where's the Problem	Recommendation
Wrong data in dashboard	Pipeline broke silently	Add pipeline status + schema validation
Conflicting metrics	Different logic by teams	Centralized metrics layer (dbt/LookML)
PM can't find data	No data exploration UI	Build data catalog + sample viewer
Real-time use case fails	Batch pipelines only	Add streaming ingestion in Sphinx
Sensitive data leaked	No access control	Role-based access, column-level security