Unified Transport & Ridership Analytics

Design Project

Photo Credit:

Key Identified Data Flows

All datasets are retrieved sequentially (Oracle ⟶ Parquet Files ⟶ JSON Files ⟶ MS SQL Server ⟶ REST API) to maintain referential dependency.

Domain	Raw Data Input	Ingestion & Processing	Advisory Insights	Deliverable
Fare & Revenue	Pre-paid ticket sales Daily train trip records	• Joins ticket sales with trip records • Validates journeys & fare tiers • Reconciles revenue against operator payouts	• Fare Reconciliation • Cost Reconciliation • Passenger Rates	Monthly Regulatory Reports
Routes & Network	▪ Daily train trip records ▪ List of operators ▪ Lists of operating bus stops and train stations ▪ Depot locations ▪ Bus routes	• Joins route/operator reference data with trip records • Aggregates ridership by route, stop & time period	• Public Transport Planning Insights • Demand Forecasts	• Daily Ridership Summaries • Weekly Planning Reports
Assets & Telemetry	GPS telemetry updates Vehicle maintenance events	• Cleans & maps GPS telemetry to fleet records • Flags at-risk vehicles via maintenance history	• Predictive Fleet Maintenance	Next-Morning Predictive Insights
Incidents	Incidents impacting passengers and vehicles	• Batch-ingests incident event logs via REST API • Enriches with affected route and vehicle context	• Incident Prevention • Commuter Traffic Analytics	• Daily Incident Reports (current) • Real-Time On-The-Go Alerts (future)

Design Principles

Cost Optimization & Economic Scaling

- **100% Open Source Frameworks:** Eliminates commercial licensing fees and vendor lock-in reducing Total Cost of Ownership (TCO).
- **Decoupled Infrastructure Scaling:** HDFS storage and Spark compute scale independently supporting both horizontal and vertical growth.
- **Minimized Custom Code Overhead:** Relies on mature, packaged components (Spark MLlib, Apache Superset) to keep long-term maintenance lean.

Simplicity

- **Standardized Tool Selection:** Apache frameworks remove the need for custom software engineering keeping the team focused on delivering analytics rather than engineering overhead.
- **Unified Processing Framework:** Utilizing Apache Spark across multiple pipeline stages minimizes tool fragmentation and simplifies overall platform management.
- **Technical Language Alignment:** Built on SQL and Python which are languages the existing team already knows minimizing onboarding friction.

System Reliability & Completeness

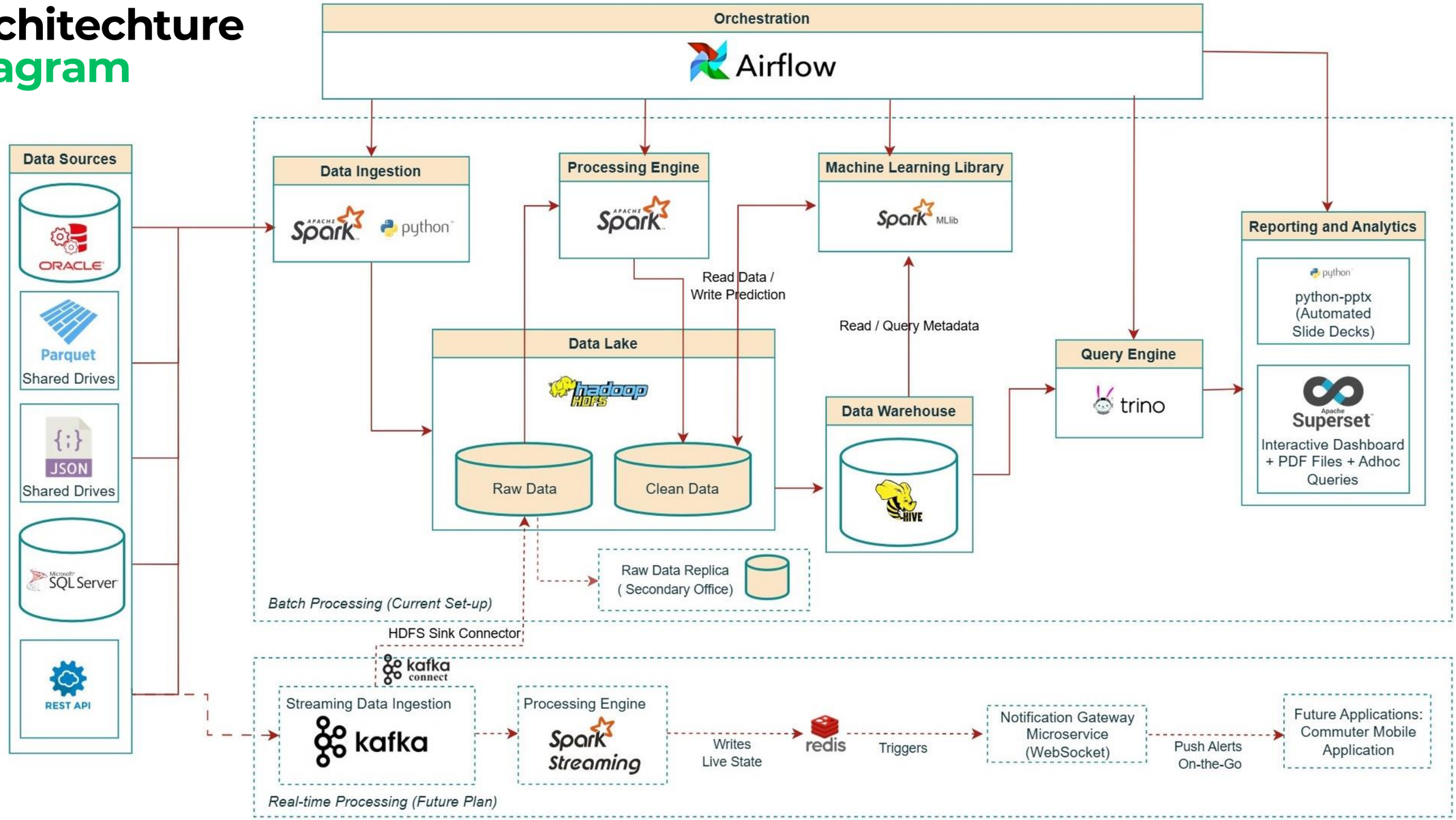
- **Resilient Orchestration:** Airflow's task-level retry logic and explicit dependency chains guarantee sequential dataset ingestion >> no step runs out of order and no failure passes silently without alerting

- **Distributed Infrastructure Resilience:** HDFS block replication for hot data and erasure coding for cold data provide tiered redundancy across the storage layer, NameNode HA with automated failover minimises metadata unavailability, and Spark lineage recovery eliminates manual intervention on compute failures

- **Batch-level Snapshot Isolation:** Airflow-enforced DAG sequencing and atomic Hive partition commits ensure analysts only query fully-loaded, complete datasets **(Strong Consistency Guarantee (CP))**

- **Geographic Disaster Recovery & Storage Isolation:** Partitions storage into Raw and Clean Zones, replicating only the immutable raw data to a secondary office. This safeguards historical records against localized hazards (e.g., fire) and external outages while optimizing backup costs by using Spark to deterministically rebuild the clean layer.

Architechture Diagram



CP-first hybrid architecture

[Batch Processing] **Technology** Components

Scope	Tool	Business Considerations	Unique Selling Points	Limitations
Orchestration	Apache Airflow	• **Strict referential ordering** between dataset pulls + need for reliable reruns on failure	• DAG-based dependency definition • Configurable retry policies per task • Built-in backfill/catchup mechanics	• Needs heavy coding for DAGs vs drag-and-drop tools • **No native versioning of pipelines**
Data Ingestion	APACHE Spark	• Capability to ingest from multiple sources	• Handles all source formats natively without additional connectors • Scales easily through **parallel ingestion** • **Single engine** for both data ingestion and processing • Actively maintained and supported (unlike Sqoop, which was retired in 2021)	• High memory consumption and increased hardware costs
	python	• Capability to ingest from REST API	• **Thin by design** >> authenticates, paginates, lands raw JSON files from REST API to HDFS • **Schema changes** handled downstream by Spark • Credentials via **Airflow Connections store** with Fernet encryption	• **Minimal ingestion layer** >> fetches data and stores raw JSON files from REST API
Data Storage	hadoop HDFS	• Company requires a cost-effective on-premise infrastructure after exiting the cloud to avoid vendor lock-in and reduce TCO	• Runs on commodity hardware keeping **on-premise TCO low** after cloud exit • Built-in replication provides **fault tolerance**: replication for hot data and erasure coding for cold data • **Native storage laye**r for Spark and Hive >> no data movement for batch analytics	• Optimized for **large** immutable datasets • Small-file overhead minimized through **Spark compaction to Parquet**

[Batch Processing] **Technology** Components

Scope	Tool	Business Considerations	Unique Selling Points	Limitations
Data Storage		• Team requires consistent schema management across Spark, Trino • Must integrate natively within the Hadoop ecosystem	• Provides **consistent table definitions** and discovery across the platform • Central metadata catalog connecting Spark and Trino • Metadata-only layer with data stored in HDFS anchoring on HDFS fault tolerance	• **Slow at query execution** hence batch processing runs on PySpark while interactive queries are handled by Trino
Processing Engine		• **High variety of data** coming from different sources with no central management • Capability to provide **machine learning and data analysis** for predictive maintenance	• Distributed in-memory processing handles high variety, multi-source data efficiently • MLlib enables predictive maintenance without additional tools • Offers built-in fault tolerace with RDDs • Python and SQL support aligns with the team's existing skills	• **High memory consumption** and increased hardware costs
		• Team relies heavily on self-service analytics and ad hoc SQL queries against Hive-based warehouse data	• Overcomes Hive's slow query performance >> enables fast adhoc SQL for self-service analytics via Superset's SQL Lab • Horizontally scalable, maintaining performance as query volumes grow	• Limited by **available cluster memory** due to its **in-memory** processing architecture • Single coordinator can become a scalability bottleneck and a potential **single point of failure**

[Batch Processing] **Technology** Components

Scope	Tool	Business Considerations	Unique Selling Points	Limitations
Reporting and Visualization	Superset	• Capability to create interactive dashboards and run adhoc queries • Serves multiple statutory bodies and transport planners >> each requiring access only to their own data	• Offers **interactive dashboards** for business consumers • **SQL Lab** provides ad-hoc query capabilities directly against Hive tables • **Row-level security** allows one dashboard to safely serve multiple tenants each seeing only their own data • **Embedded SDK** with guest tokens enables dashboards to be rendered **inside consumer portals**	• Native reporting is limited to static PDF or PNG snapshots, without support for customizable PDF layouts.
	python™	• Capability to generate customizable slide decks	• **Pure Python library** which leverages on existing team skills to programmatically generate and customize slide decks • Runs as a scheduled Airflow task automating delivery after each daily pipeline run	• More complex slide design needs **more custom scripting** to maintain as layout is fully code-driven

[Real-time Processing] **Technology** Components

Scope	Tool	Business Considerations	Unique Selling Points	Limitations
Data Ingestion	kafka	• Capability to do streaming ingestion **without disrupting the current batch pipeline**	• **No custom ingestion code** required • Streaming data lands in existing HDFS storage • Fault tolerance from replication model	• Future enhancement only >> extends beyond the current Hadoop ecosystem • Applies only to incident data with potential real-time API feeds • **GPS data continues through batch ingestion** until streaming support is provided
Processing Engine	Spark Streaming	• **Unified platform** for data ingestion, processing, and predictive maintenance >> **simplifies** number of specialized tools	• **Micro-batching** process of data to manage spikes in data • Checkpointing and Write-Ahead Logs >> guarantees **data is processed exactly once** (within the spark pipeline)	• High memory consumption and **increased hardware costs** • Micro-batch also results in **inherent latency** as it accumulates records to be processed
Data Storage	redis	• Future support for **real-time commuter alerts** • Adds real-time commuter alert capability **without disrupting the existing batch pipeline** (Kafka's HDFS sink connector)	• Fast **in-memory state store** for live incidents • Drives real-time push notifications to commuters • TTL-based deduplication: prevents the same alert being pushed to a commuter multiple times	• **Persistence must be explicitly enabled**: RDB/AOF persistence needed to protect against state loss on restart • **Single point of failure risk:** Redis clustering or replication should be considered for production