**Root-Cause Analytics MVP Architecture**

This document explains the **proposed MVP architecture** for the Root-Cause Delivery Failure Analytics system. It contains a **deterministic-first natural language understanding (NLU)** approach with **LLM fallback**, ensuring predictability, safety, and cost-efficiency while still supporting flexible queries.

**1. Objectives of the MVP**

The MVP aims to:

* **Aggregate multi-domain data**: orders, fleet logs, warehouse logs, customer feedback, external conditions (traffic, weather).
* **Correlate events automatically**: identify systemic issues behind delivery failures.
* **Provide human-readable insights**: concise narratives instead of raw dashboards.
* **Suggest actionable recommendations**: staffing, rescheduling, address verification.
* **Support six key use cases** (e.g., “Why were deliveries delayed in city X yesterday?”).
* **Ensure safety and auditability**: every query and result is validated and logged.

**2. Components of the MVP**

**2.1 Ingestion & Storage**

* **Raw Data Sources**: Order, warehouse, fleet, and feedback CSV/JSON files, plus external traffic/weather feeds.
* **S3 – Raw Landing**: Central storage for immutable raw data.
* **Glue Crawler & Glue Data Catalog**: Automatically infer and register schema.
* **Glue ETL Jobs**: Clean and normalize data, join across domains, derive features (e.g., delivery delay minutes, inferred causes).
* **S3 – Processed (Parquet)**: Curated, optimized datasets ready for analytics.

**2.2 Analytics Layer**

* **Athena**: For ad hoc SQL queries directly over S3.
* **Redshift Serverless**: Primary analytics engine for fast, frequent queries.
* **Materialized Views**: Pre-aggregated tables (e.g., failures by city/day, by warehouse/month) to accelerate common queries.

**2.3 Serving & Compute**

* **API Gateway**: Entry point for user queries.
* **API Service (ECS/Fargate)**: Stateless service orchestrating NLU, validation, execution, and responses.
* **Deterministic NLU / Template Engine**:
  + Matches queries against a library of templates (regex/patterns).
  + If matched with high confidence, generates SQL/plan directly.
  + Covers common business questions (the 6 sample use cases and variants).
* **Template Library**: Repository of NL patterns mapped to parametric SQL skeletons.
* **Plan Generator**: Converts template matches into structured JSON plans.
* **LLM (OpenAI)**: Used only when no template match is found. Converts NL query → JSON plan, or polishes narrative output.
* **Plan Validator & Sandbox**:
  + Validates column names, allowed operations, date ranges, and sensitive data policies.
  + Applies safety checks before execution.
* **Confirmation Engine**: Requests user confirmation for high-risk queries (large ranges, sensitive data).
* **Query Executor**: Runs validated plans on Athena or Redshift.
* **Post-Processor & Recommendations**: Summarizes results, computes percentages, ranks causes, and generates actionable recommendations.
* **Optional LLM Narrative Polish**: Polishes output into clear, user-friendly text.
* **Audit Store (S3/DynamoDB)**: Logs queries, chosen path (deterministic vs. LLM), validated plan, results, and timestamps.
* **Secrets Manager**: Secures API keys (e.g., OpenAI, DB credentials).
* **Monitoring & Alerts (CloudWatch)**: Monitors API latency, ETL health, query cost, and LLM usage.

**2.4 LLM & Models**

* **OpenAI API**: Provides fallback NL → Plan translation and optional narrative polish.
* **SageMaker (future phase)**: For training ML models (e.g., probabilistic cause classification, predictive risk forecasting).

**2.5 User Interfaces**

* **Web Chat / CLI / Slack Bot**: Interfaces for users to ask NL queries.
* **Ops Dashboard (BI tools)**: Connects to Redshift for advanced visualization.

**3. User Flow (End-to-End)**

**Deterministic-first with LLM fallback**

1. **User submits query** (e.g., “Why were deliveries delayed in Ahmedabad yesterday?”).
2. **API Gateway** receives and forwards to API Service.
3. **Deterministic NLU check**:
   * The query is compared against a library of templates.
   * If high-confidence match → slots filled (e.g., city=Ahmedabad, date=2025-09-14).
   * A structured plan is generated directly.
4. **If no template match**:
   * The query is passed to the **LLM translator**.
   * LLM returns a candidate JSON plan.
5. **Plan Validator & Sandbox**:
   * Checks schema compliance, allowed ops, date/range limits, sensitive column policy.
   * Applies cost heuristics to prevent expensive queries.
   * Rejects unsafe/invalid plans.
6. **Confirmation Engine**:
   * For high-risk queries (wide ranges, sensitive columns), the plan is shown to the user for approval.
7. **Query Execution**:
   * Plan runs on Redshift (fast queries) or Athena (ad hoc queries).
8. **Post-Processing**:
   * Results aggregated, top causes ranked, percentages computed.
   * Recommendations generated (e.g., “Increase warehouse staff during peak hours”).
   * Optionally polished by the LLM for readability.
9. **Results returned to user**: Narrative + tables + recommendations.
10. **Audit Logging**: Entire interaction recorded in S3/DynamoDB.

**4. Example Query Lifecycle**

* **User Input**:

“Explain the top reasons for delivery failures linked to Warehouse B in August.”

* **Deterministic Template Match**:

Template recognized: *“Explain the top reasons for failures at  in ”*.

Slots filled: { warehouse: 'Warehouse B', date\_from: '2025-08-01', date\_to: '2025-08-31' }.

* **Generated Plan (JSON)**:

{

"intent": "warehouse\_diagnostic",

"slots": { "warehouse": "Warehouse B", "date\_from": "2025-08-01", "date\_to": "2025-08-31" },

"operations": [

{ "op": "aggregate", "by": ["inferred\_causes"], "filters": {"warehouse": "Warehouse B"} }

]

}

* **Execution Result**:
  + warehouse\_prep\_delay: 45%
  + address\_issue: 30%
  + high\_traffic: 15%
* **Narrative**:

*“In August, Warehouse B’s delivery delays were primarily due to prep delays (45%), address issues (30%), and high traffic (15%). Recommendation: add temporary staff during peak hours and enforce address verification.”*

**5. Why Deterministic-First?**

* **Predictability**: Templates ensure reliable answers for the most common questions.
* **Cost control**: Minimizes LLM calls, reducing token cost.
* **Safety**: Templates + validation reduce the chance of invalid SQL or hallucinations.
* **Flexibility**: LLM still available for long-tail queries.

**6. Next Steps Beyond MVP**

* Add more templates based on usage analytics.
* Train ML classifiers (SageMaker) for inferred causes.
* Enable real-time ingestion (Kinesis) for live monitoring.
* Expand dashboards and workflow integrations.

**7. Conclusion**

The **deterministic-first architecture with LLM fallback** provides the best of both worlds:

* **Predictability and safety** for common queries.
* **Flexibility and adaptability** for unexpected questions.
* **Auditability and scalability** for long-term growth.