Cost of Risk Engine: Project Report

Candidate: Analytics Engineer Specialist Candidate

Project: Credix Cost of Risk Engine

Status: Production-Hardened

1. Executive Summary

This project establishes a scalable, trusted data foundation for calculating **Cost of Risk** (Expected Loss) across the credit portfolio. Moving beyond a functional prototype, this engine has been "production-hardened" to handle real-world data anomalies, contractual versioning, and complex risk logic.

The solution transforms raw Postgres logs into a high-fidelity **Gold Mart** and leverages **MetricFlow** to provide a Semantic Layer that answers the critical business question: *"How much capital is at risk based on current portfolio performance?"*

2. Infrastructure & Stack

The solution is built on a modern, modular analytics stack designed for auditability and scale:

- **Warehouse:** Google BigQuery (Sandbox) for serverless, high-performance computing.
- **Transformation:** dbt Cloud (connected via GitHub). Sources defined in `_sources.yml` map to the `raw_data` schema.
- **Semantic Layer:** MetricFlow (dbt Semantic Layer) for centralized metric definitions.
- **BI & Visualization:** Google Looker Studio (connected to `mart_monthly_risk_summary`).
- **Orchestration:** dbt Cloud Job Scheduler.

3. Data Engineering Challenges & Solutions

During the transition from "functional" to "production-grade," three critical data challenges were identified and resolved.

A. The "Ghost Loan" Anomalies (Data Quality)

- **Problem:** The raw source system contained ~450 records with `face_value = 0.00` marked as `Settled`. These were identified as system artifacts/logs rather than valid financial liabilities.

- **Impact:** Including these records would artificially inflate the "Loan Count" metric, heavily skewing "Average Ticket Size" analysis.
- **Solution:** Implemented a hard filter in the Staging Layer (`stg_assets.sql`) to exclude records where `face_value <= 0`.

B. Identity Resolution & Contractual Versioning

- **Problem:** Loans in the source system had multiple "versions." A single loan might appear multiple times with different `due_date` values (renegotiations) or `settled_at` timestamps. Standard hashing or deduplication would result in data loss.
- **Impact:** Using a simple ID hash caused "Unique Key Collisions," causing dbt build failures.
- **Solution:** Implemented the **"Ultimate" Surrogate Key** strategy.
    - **Logic:** A composite hash of *all* business-critical attributes: `created_at`, `buyer_tax_id`, `face_value`, `settled_at`, `buyer_state`, `due_date`, and `status`.
    - **Result:** Contractual history is preserved. A renegotiation is treated as a distinct ledger entry, allowing for accurate historical risk analysis without key collisions.
    - **Metric:** Resolved **22,000+ duplicate key errors** during testing.

C. The "Implicit Default" Logic Gap

- **Problem:** A dangerous gap was detected where loans were marked `Active` in the source system but were **>30 days overdue**.
- **Impact:** Relying solely on the source system status led to massive under-provisioning (assigning a 1-40% risk rating to assets that should be 100% provisioned).
- **Solution:** Implemented a Business Logic Override in the Fact Layer (`fct_credit_risk.sql`).
    - **Rule:** `IF days_overdue > 30 THEN Status = 'Defaulted' (Provision Rate = 1.0)`.
    - **Validation:** This logic caught and corrected **322 instances** where risk was previously underestimated.

4. The Semantic Layer

The Semantic Layer abstracts complex SQL into business-friendly metrics, ensuring consistency across all downstream tools.

Metric	Definition	Business Value

Total Exposure	`Sum(Face Value)`	The total principal amount currently outstanding.
Total Expected Loss	`Sum(Face Value * Provision Rate)`	The trusted "Cost of Risk" amount we expect to write off.
Portfolio Provision Rate	`Total Loss / Total Exposure`	A normalized risk index (%) allowing comparison between Cohorts/States.

Key Feature: The Time Spine A custom time spine (`metricflow_time_spine.sql`) was implemented using BigQuery's `generate_date_array` function. This facilitates continuous time-series analysis, allowing stakeholders to view risk evolution "Month-over-Month" (even for months with zero activity) without writing complex date-spine SQL.

5. Reliability & Testing

Trust is paramount in financial data. The pipeline is guarded by a comprehensive testing suite.

- **Generic Tests:**
  - `unique` & `not_null` constraints applied to the complex surrogate keys in `stg_assets` and `fct_credit_risk`.
  - `accepted_values` ensures statuses never deviate from the normalized list: `['Settled', 'Active', 'Defaulted', 'Canceled']`.
  - `accepted_range` ensures `provision_rate` is always between 0.0 and 1.0.
- **Singular Tests (Logic Integrity):**
  - **Test:** `assert_risk_logic_integrity.sql`
  - **Function:** Scans every row in the final fact table to ensure no "illegal states" exist (e.g., a `Settled` loan with a `Provision Rate > 0` or a `Defaulted` loan with `Provision Rate < 1.0`).
  - **Status: PASSED (0 Failures)**.

6. Business Intelligence & Insights

The data model now powers a **Risk Concentration Dashboard** in Looker Studio.

- **Key Insight:** We can now slice the "Risk Index" by **State**.
- **Drill-Down Capability:** Stakeholders can filter by `Cohort` (Month of Origination) to see if newer vintages are performing better or worse than older ones.

- **Auditability:** Every number on the dashboard can be traced back to the `fct_credit_risk` table, with transparent lineage showing exactly how the `provision_rate` was derived.

7. Conclusion & Next Steps

This project has successfully delivered a **production-hardened Cost of Risk Engine**. By resolving deep data quality issues in the Staging layer and enforcing strict business logic in the Fact layer, we have transformed raw, noisy logs into a strategic asset.

Recommendations:

1. **Ingestion:** Automate the manual CSV upload process via an Airbyte/Fivetran connector to S3/GCS.
2. **Alerting:** Configure dbt Source Freshness alerts to notify the team if `assets.csv` is not updated within 24 hours.
3. **CI/CD Configuration:** The current `_sources.yml` points to a static `credix-challenge` database. For full production readiness, this should be parameterized using `{{ env_var('DBT_SOURCE_DATABASE') }}` to support dynamic environments (Dev vs. Prod).
4. **History:** Implement `dbt snapshots` to track changes in `ratings` over time more granularly than the current "Latest Rating" logic.