CRED Platform: Data Pipeline Design Doc

Author: Richa Patel
Date: October 2025

1. Goal

Design a scalable, reliable, and cost-effective data platform to support AI-driven analytics and dashboards for high-volume user event data.

Requirements:

- Handle billions of rows efficiently
- Serve both analytics and ML workloads
- Ensure data quality, reproducibility, and incremental processing
- Balance engineering effort with Series A startup resource constraints

2. Event Data Overview

Sample event:

```
{
  "user_id": "abc123",
  "event_type": "purchase",
  "event_timestamp": "2025-09-01T12:34:56Z",
  "metadata": {
    "amount": 42.50,
    "currency": "USD"
  }
}
```

Events originate from web and mobile apps and may arrive in real-time or batch.

3. Pipeline Architecture

Flow Overview:

CRED Data Platform: Event Pipeline Overview

1. Raw Event Ingestion
    - Events are collected from web and mobile apps.
    - Can be ingested via GCP Pub/Sub for streaming or CSV uploads for batch.

2. Raw Table in BigQuery (events_raw)
    - All incoming events are stored in their raw form.
    - Maintains an immutable, auditable history of all events.
3. Staging Layer (stg_events) via dbt
    - Performs data cleaning, such as removing duplicates and correcting types.
    - Flattens JSON fields (e.g., metadata.amount) for easier querying.
    - Validates data types and schema consistency.
4. Fact / Curated Layer (fact_user_activity) via dbt
    - Aggregates key metrics per user for dashboards and ML:
        - first_event – timestamp of first activity
        - last_event – timestamp of last activity
        - total_events – number of events per user
        - total_amount – sum of purchase amounts
    - Supports incremental loading to process only new data.
    - Tables are partitioned by event_date for performance and cost efficiency.
5. Serving Layer: GraphQL API / ML Feature Store
    - Provides curated metrics to dashboards and AI models.
    - Supports daily refresh or near-real-time serving depending on business need.

3.1 Ingestion

- Choice: GCP Pub/Sub for streaming; CSV uploads for batch testing.
- Reasoning:
    - Streaming ensures near-real-time analytics for dashboards and ML features.
    - Batch ingestion provides a simple, low-cost method for testing or small data loads.
- Trade-offs:
    - Streaming introduces complexity (Pub/Sub + Dataflow).
    - Batch is simpler but slower for real-time analytics.

3.2 Storage Strategy

- Raw Layer: events_raw
    - Immutable, captures all events

- Partitioned by ingestion date for cost efficiency
- Curated Layer: fact_user_activity
    - Aggregated metrics for analytics & ML
    - Partitioned by event_date
    - Clustered by user_id for efficient lookups

Trade-offs:

- Raw layer preserves full history; good for audits and backfills
- Curated layer is optimized for query performance

3.3 Transformations (dbt)

- Staging Layer (stg_events):
    - Flatten nested JSON fields
    - Type validation & null checks
    - Standardize timestamps and currency
- Fact Layer (fact_user_activity):
    - Aggregates per user per day:
        - first_event / last_event timestamps
        - total_events / total_amount
    - Incremental logic: only process new events
    - Partitioned & clustered for performance

Trade-offs:

- dbt simplifies transformations and testing
- Incremental models reduce cost and compute time

3.4 Serving Layer

- GraphQL API or ML Feature Store
- Supports dashboards and AI model consumption
- Refresh strategy: daily incremental updates (or near real-time for critical features)

4. Scalability & Reliability

- Partitioning & Clustering: reduces query scan cost
- Incremental loading: avoids full-table recomputation

- Data quality checks:
  - Row counts, null checks, uniqueness
  - dbt tests integrated in CI/CD pipeline
- Error handling & monitoring: alert on schema changes or failed incremental runs

5. Series A Prioritization

- First priorities:
  1. Reliable raw ingestion & staging
  2. Fact table with core metrics (first_event, last_event, total_events, total_amount)
  3. Data quality tests to ensure trust in metrics
- Later priorities:
  - Near real-time dashboards
  - Feature store for ML
  - Lineage tracking & cost optimization

6. Summary

This design provides a modular, scalable, and reliable pipeline:

- Raw → Staging → Fact → Serving layers
- Supports AI/ML workloads with incremental aggregation and partitioning
- Enforces data quality via dbt tests
- Balances engineering effort with Series A resource constraints