Technical Specifications: Analytics Engine (v1.0)

1. Project Overview

- **Status:** Under Review.
- **Primary Objective:** To provide a low-latency (<200ms) query engine for multi-dimensional telemetry data.
- **Target Users:** Data Scientists and Operations Analysts.

2. System Architecture

The system follows a decoupled architecture, separating data ingestion, processing, and the presentation layer to ensure horizontal scalability.

2.1 Component Overview

- **Ingestion Tier:** Distributed message queue (Kafka) to buffer incoming raw JSON payloads.
- **Processing Tier:** A Spark-based transformation layer that flattens nested structures and performs schema validation.
- **Storage Tier:** A columnar OLAP database (ClickHouse) optimized for rapid aggregations.
- **API Tier:** A GraphQL gateway to allow flexible frontend queries without over-fetching data.

3. Data Schema & Models

To maximize query performance, the model utilizes a star schema approach for our analytical tables.

3.1 Core Telemetry Table (telemetry_events)

Field	Type	Description
event_id	UUID	Primary Key
timestamp	DateTime64	Event occurrence (UTC)
metric_name	LowCardinality(String)	Type of data (CPU, Latency, etc/)

value	Float64	The numeric observation
tags	Map(String, String)	Metadata for filtering (Region, Environment)

4. API Design (GraphQL)

The frontend communicates with the engine via a unified GraphQL schema to support dynamic dashboarding.

4.1 Example Query: Metric Aggregation

```
query GetMetricAvg($metric: String!, $start: ISO8601!, $end: ISO8601!) {
  metrics(name: $metric, timeframe: {start: $start, end: $end}) {
    ...ValueFields
  }
}
fragment ValueFields on MetricType {
  timestamp
  averageValue
  p95Value
}
```

5. Technical Implementation Details

5.1 Real-time Aggregation Logic

Instead of recalculating averages on every request, the system utilizes Materialized Views.

1. Raw data enters the events_local table.
2. A trigger moves data into a summary_minute view.
3. The API queries the summary_minute view for time-series charts, reducing disk I/O by 80%.

5.2 Concurrency Handling

To prevent latency during peak analysis hours, we implement:

- **Query Quotas:** Limits on execution time per user.
- **Result Caching:** Redis-based caching for identical queries within a 30-second window.

6. Infrastructure & Deployment

- **Containerization:** All services packaged via Docker.
- **Orchestration:** Managed Kubernetes (EKS/GKE).
- **Monitoring:** Prometheus for system health; ELK stack for log aggregation.

7. Security & Compliance

- **Authentication:** Integration with Corporate SSO (OpenID Connect).
- **Encryption:** AES-256 for data at rest; TLS 1.3 for data in transit.
- **Data Masking:** Automated redaction of PII (Personally Identifiable Information) during the processing tier before storage.

8. Development Roadmap

- **Phase 1:** Core Ingestion & SQL Query Interface.
- **Phase 2:** GraphQL Layer & UI Dashboard Components.
- **Phase 3:** AI-driven Anomaly Detection (Agentic Reasoning).