MVP Telemetry Analytics Schema for Apache Iceberg

Author: Martin Arroyo
Date: October 2, 2025
Version: 0.1.1

1. Overview

This document provides the complete data lake schema for the MVP analytics layer, architected on **Apache Iceberg**. This design replaces the original PostgreSQL concept to better handle the high volume and analytical nature of telemetry data.

The schema follows a **denormalized, flat-table model** to optimize for high-speed data ingestion and fast analytical queries by avoiding the need for joins. All tables are designed to be **multi-tenant** from the ground up, ensuring that data for each organization is securely isolated while stored within a single, unified data lake.

While the immediate focus is on capturing detailed trace and span data, this schema holistically includes traces, logs, and metrics to provide a complete, future-proof observability platform.

---

2. Architecture: Partitioning and Sorting Strategy 🏛️

The performance and security of our multi-tenant data lake are built on a specific partitioning and sorting strategy. This approach is designed to keep our most common queries fast without sacrificing the ability to perform crucial platform-wide analytics.

- **Partitioning**: Tables are partitioned first by **month** and then by **organization ID**.
  - PARTITIONED BY (months(event_timestamp), organization_id)
  - **Why this order?** Nearly all queries will filter on a time range. By partitioning by month first, the query engine can instantly discard the vast majority of data (e.g., all previous months), making both organization-specific and platform-wide queries extremely efficient. This creates a manageable folder structure like .../event_timestamp_month=2025-10/organization_id=org_abc/.
- **Sorting**: Within each partition's data files, records are sorted by **workload run ID**.
  - TBLPROPERTIES ('write.sorted-by' = 'workload_run_id, ...')
  - **Why sort?** The workload_run_id is a high-cardinality field (it has many unique values). Sorting by this ID allows the query engine to use file metadata to quickly locate all events related to a single execution, dramatically speeding up requests

for a specific trace.

This layered strategy ensures that whether we are fetching the latest data for one organization or running a global query across all tenants, the performance remains high.

---

3. Table: trace_events (Primary Focus)

This table stores the enriched data from OpenTelemetry spans. Each row represents a single node invocation within a workload.

Iceberg SQL Schema

SQL

```sql
CREATE TABLE trace_events (
  -- Core OTel Identifiers
  span_id STRING,
  trace_id STRING NOT NULL,
  parent_span_id STRING,

  -- Execution Grouping Identifier
  workload_run_id STRING NOT NULL,

  -- Event Timing & Core Metadata
  event_timestamp TIMESTAMP NOT NULL,
  duration_ms BIGINT, -- Latency in milliseconds

  -- Hierarchical Identifiers for Roll-ups
  organization_id STRING NOT NULL,
  org_namespace STRING, -- e.g., 'Codon', 'Pepsi'
  logic_id STRING NOT NULL,
  nodespec_id STRING NOT NULL,
```

```
  -- Denormalized Descriptive Attributes
  service_name STRING, -- The name of the microservice emitting the trace
  workload_name STRING,
  node_name STRING,
  node_role STRING,
  model_name STRING,
  model_vendor STRING,

  -- Key Performance Metrics
  input_tokens BIGINT,
  output_tokens BIGINT,
  token_usage_json STRING,

  -- Payloads & Status
  node_input STRING,
  node_output STRING,
  status_code STRING, -- e.g., 'OK' or 'ERROR'
  error_message STRING,

  -- Raw data for debugging and future migration
  raw_attributes_json STRING
)
USING iceberg
PARTITIONED BY (months(event_timestamp), organization_id)
TBLPROPERTIES ('write.sorted-by' = 'workload_run_id, logic_id');
```

---

4. Table: log_events

This table stores log messages, correlated to specific traces and workload runs for comprehensive debugging.

Iceberg SQL Schema

SQL

```sql
CREATE TABLE log_events (
  log_id STRING,
  event_timestamp TIMESTAMP NOT NULL,
  workload_run_id STRING,
  trace_id STRING,
  span_id STRING,
  organization_id STRING,
  org_namespace STRING,
  service_name STRING,
  severity_text STRING,
  body STRING,
  raw_attributes_json STRING
)
USING iceberg
PARTITIONED BY (months(event_timestamp), organization_id)
TBLPROPERTIES ('write.sorted-by' = 'workload_run_id, trace_id');
```

---

5. Table: metric_events

This table stores time-series numerical data points for monitoring and alerting.

Iceberg SQL Schema

SQL

```sql
CREATE TABLE metric_events (
```

```
  event_timestamp TIMESTAMP NOT NULL,
  metric_name STRING NOT NULL,
  organization_id STRING,
  org_namespace STRING,
  workload_run_id STRING,
  service_name STRING,
  metric_value DOUBLE NOT NULL,
  metric_type STRING,
  raw_attributes_json STRING
)
USING iceberg
PARTITIONED BY (months(event_timestamp), organization_id)
TBLPROPERTIES ('write.sorted-by' = 'workload_run_id, metric_name');
```

Sample Data

Sample data for the tables can be found here:

Codon SDK Telemetry Schema