E6Data Telemetry and Observability

Problem Statement:

Design and implement a system that:

1. Ingests high-volume query events in real-time

2. Transforms and stores these events in an OLAP-optimized format

3. Ensures data is available for analytical queries within T+20 seconds (where T is the event generation timestamp)

4. Handles the specified load sustainably

E6Data's analytical query engine processes SQL queries continuously with the following operational characteristics:

• **Query Processing Rate**: 100 queries per second

• **P95 Latency**: 2 seconds per query

• **Event Generation**: Each query produces approximately 5 events during its lifecycle

• **Event Size**: Individual events range from 1 KB to 5 KB

• **Business Requirement**: All events must be queryable for analytical purposes within 20 seconds of generation

Functional Requirements

• Process minimum 500 events per second

• Store data in columnar/OLAP format suitable for analytical queries

• Data must be queryable within 20 seconds of event generation

• Support basic analytical queries (aggregations, time-series analysis, filtering)

• Handle event ordering and potential out-of-order arrivals

• Ensure no data loss under normal operating conditions

Non-Functional Requirements

• System should be horizontally scalable

• Demonstrate fault tolerance for single component failures

• Provide observability into system health and performance

• Resource-efficient operation

• Clear separation of concerns in architecture

Design Discussions

Assumptions:

1. Data Generator would be a simple standalone application that writes to the ingestion end-point. Not much has been done to make the data generator fail-safe. It is assumed that the failure-handling, retry mechanisms, recovery of lost/missed data is handled within the Query Processor Engine (QPE) itself.

    a. Typically, if the event is not reaching the end-point due to any reason, there would be appropriate retry mechanisms with exponential backoff to ensure proper delivery.
    b. The delivery of telemetry data shouldn't block the core business operations of the QPE itself. It is assumed that this is handled by either:
        i. Handling logging asynchronously, or
        ii. Having a separate service which takes care of sending events to the ingestion end-point.

    c. The time stamp of the logs are correctly marked at the correct time of generating it, and there is no induced delay.

2. Event guarantees is assumed to be **at-least-once** delivery (considering this is a telemetry and observability service). This would mean that there would be chances of duplicate events, which needs to be handled in the system.

3. Assuming that 1-5KB of each data packet is a hard-bound and there are no outliers.

4. Schema changes of the event are possible and should be handled within the system.

5. Failure handling – There are 2 types of failures that needs to be handled:

    a. Failures at QPE level – This means that there is an outage at the Event Source itself, which would mean that – a. No Queries are being processed; b.

Queries are being processed, but the telemetry is not going through with some exceptions.

   b. Failures at the Telemetry Ingestion layer – This would mean that the packets are being sent by QPE, but there is some failure at the Telemetry service itself. This would typically be handled within the service and will explain it in the design discussions.

6. The solution would be cloud based, though the container-based design would be discussed in this document.

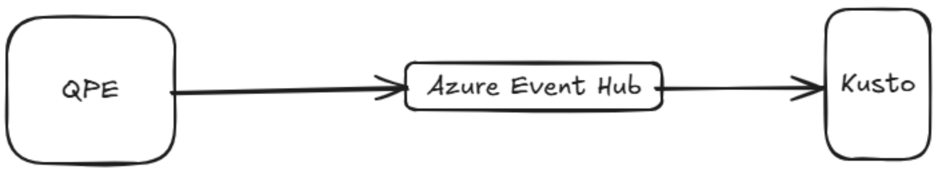
7. <T+20s delivery would be ensured for P95 data events.

Design decision walkthrough

Below are the design considerations done for observability.

1. **Event Hub -> Azure Data Explorer**

   The simplest way to handle this would be to have QPS send the events to Event Hub. The Event Hub, would connect to Azure Data Explorer. Any event that comes into Event Hub can be directly sent into Azure Data Explorer, which acts as a OLAP storage layer, where the data can be queried directly with KQL.



   Pros:

   1. Low latency can be achieved as the Events are being written directly from the Ingestion queue to the Storage/Query layer.
   2. With Ingestion Streaming option enabled, the events can directly be streamed to Kusto/Azure Data Explorer with low latency.
   3. Low cost, lesser moving parts implies lesser ops cost.
   4. Perfectly aligns with the requirement of <20s availability on the Querying layer.

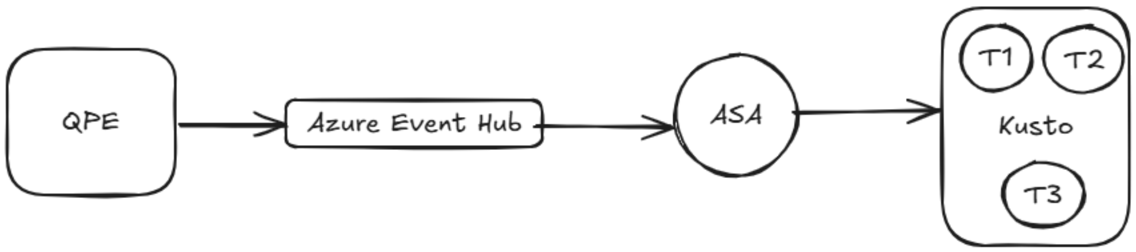
   Cons:

   1. The transformation layer is missed out. It is possible to update schema mappings to store the data in ADX.

2. Flexibility to add enrichment, filtering, complex stream joins is not easy and adds to latency.

2. **Event Hub -> Azure Stream Analytics -> Azure Data Explorer**

   This involves adding the extra transformation layer Azure Streaming Analytics, which is a scalable streaming service to publish data in near-real-time. The data is distributed into 2 tables for this requirement -> All sink events and Error events table. This is done for 2 reasons- a. To isolate critical error details, which can be queried and worked analyzed independently. b. To demonstrate some of the latency shortcomings in ADX, and how it was overcome.



   Pros:

   1. Transformation layer allows data to be cleaned, enriched, processed in real time.
   2. Can be simultaneously written to multiple output sinks with minimal latency. (ADX for querying, Cold storage for compliance, stream to PowerBI for dashboard etc)
   3. Provides flexibility.
   4. Handles out-of-order events, provides de-duplication to an extent.
   5. Though it is a slightly costlier option, it would reduce the operational costs in the long run.

   Cons:

   1. Added Cost and design complexity
   2. Added latency (~5-10 seconds)

3. **Hybrid approach (Local + Cloud)**

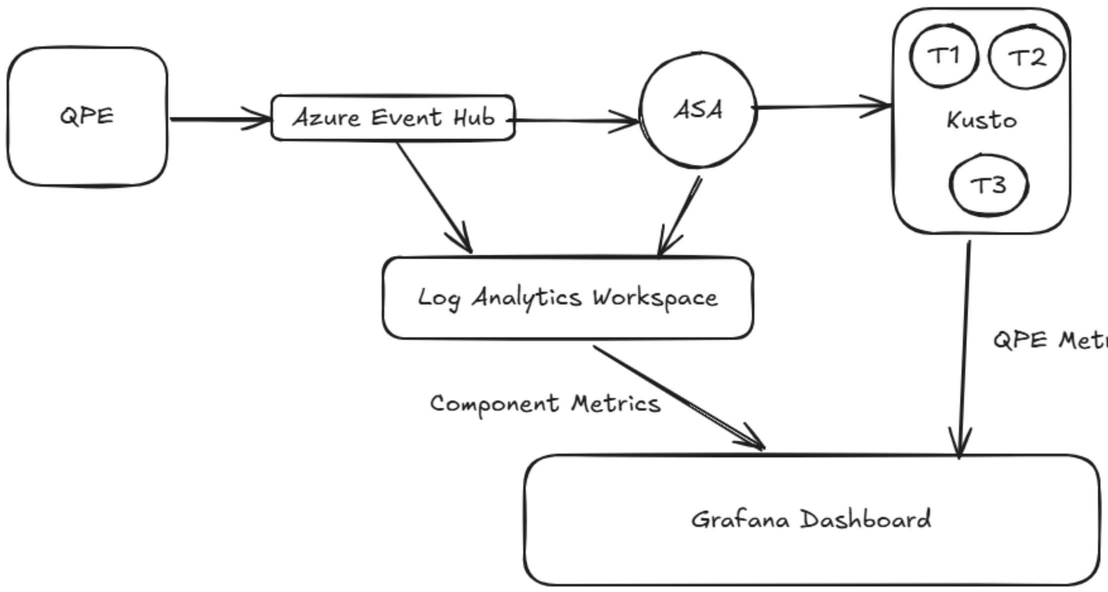
   This involves QPE writing to both Azure Event hub and container-based Kafka. The architecture is same, just that the components of the container-based pipeline are different:

   Cloud approach : EH -> ASA -> ADX
   Container based approach : Kafka -> Flink -> Clickhouse

   One can act as a failover for the other.

The telemetry can be split into Business critical and non-critical. Each one can be handled in different pipelines.

For this, the requirement cloud-based approach is chosen due to implementation time constraints and familiarity.



Component level metrics are exported to Grafana via Log Analytics Wrokspace and Kusto data, which contains the QPE metrics is connected to Grafana via Azure Data Explorer plugin available on Grafana.

Data Model and Schema

The Event Data is given to be in the below format:

```
{
"query_id": "unique_identifier",
"timestamp": "ISO-8601 timestamp",
"event_type": "enumerated_type",
"query_text": "actual SQL query",
"metadata": {
"user_id": "string",
"database": "string",
"duration_ms": "number",
"rows_affected": "number",
"error": "optional_error_details"
},
"payload": "event_specific_data"
}
```

Data schema in ADX:

For this flow, 2 tables are created:

a. Sink table – containing all events from QEP without errors.
b. Error table – containing all error events from QEP.

This is done to ensure more focus on tracking of reducing the error count for the Product's success.

Schema for sink table:

query_id: string, event_type: string, query_text: string, user_id: string, database_name: string, duration_ms: long, rows_affected: long, payload: string, event_time: datetime, ingestion_time: datetime

Schema for error table:

query_id: string, event_type: string, query_text: string, user_id: string, database_name: string, error: string, event_time: datetime, ingestion_time: datetime

Transformation Layer

The transformation layer – in this case ASA transforms the data into OLAP-optimized format. There are advantages of adding this transformation layer:

1. The incoming stream of data can be split into multiple tables or multiple sinks to process separately.
2. The responsibility of transformation lies with ASA alone.
3. It can be scaled independently.
4. Late packet policy, out of order events can be set. De-duplication can also be done with a change in query.

Data delivery and Ordering

Scalability

Each component of the Observability pipeline can be scaled independently.

1. Event Hub – Currently, namespace set to 4 Throughput Units, which can auto scale to 10 TUs based on load. Partition count is set to 16 to increase parallelism on processing downstream.

In case of abnormal data spikes or recovery data processing (post-outage) causing Throttling errors - the TU can be increased temporarily to handle the spike and gradually reduced once the data flow is normal.

2. Streaming Analytics – Currently running at 2 Streaming Units – consuming < 30% CPU and Memory. This can be reduced for cost-saving. The Watermark Delay (latency) on ASA side is ~10 seconds with this configuration.

   In case of abnormal data spikes or recovery data processing (post-outage) causing high water mark delays – ASA can be scaled horizontally by increasing the SU to handle the load spikes. (Though ASA does have auto-scale feature that was developed recently, it is flaky and could add to additional consumption costs if not monitored)

3. Data Explorer – This is currently at the lowest setting having Dev SKU (Dev(No SLA)_Standard_D11_v2). Auto-scale feature is available for higher SKUs. For the current load, this would be just fine.

   ADX is typically a cold-start service, which allocates resources based on the incoming traffic. So it is expected to have higher latency when data flow starts, but the latency gradually minimizes with time and continuous data flow.

Failure Scenarios and Data Recovery

Below highlights the failure and recovery scenarios in each of the components:

1. QPE failures: This would typically mean that no queries are being processed, resulting in no telemetry OR Queries are being processed, but telemetry is not being sent. Either way, the recovery steps for this scenario would be:
   a. Local Storage of telemetry and push once service is back online.
   b. Send telemetry to a secondary source (in case of hybrid model)

2. Event Hub Failures: Event hubs could fail due to several reasons – Throttling errors, Server errors, User errors etc. Data drops are generally not expected for Event Hubs. Either the producer would fail to write or any failures on EH could cause:

   a. Write errors on QPE writes to EH – Strategy to handle this would be same as QPE failure handling.

   b. Read errors on ASA side – This shows up as watermark delay increase on ASA side and would generally show throttling errors on EH side. Strategy to handle this would be to increase the TU to mitigate the throttling first and then increase the SU on ASA side to ease the data spike (if any). This can be done temporarily if it is a data spike. If this is an organic growth of data, it would be a more permanent change like increasing partition count to increase parallel processing or permanently increase the TU and SU on EH and ASA.

3. Failures on ASA Layer – This would show up as no data in sink or high WMD. ASA maintains a checkpoint on the Event Hub reads.
   If ASA Fails, the recovery steps would be to restart ASA from the last checkpoint. This would cause a burst in event volume, which can be handled by scaling up the ASA job.

4. Failure on ADX layer – This would mean the telemetry is not available to query and process. To resolve this, we could connect to a secondary sink – like Storage blob or other OLAP compliant storages. But this would be an overhead on the cost and would mostly be redundant as

Observability

There are 2 dashboards configured for the end-to-end observability.

1. The Component health dashboard – This shows metrics from the components used for the pipeline. This would show the overall health of the telemetry pipeline itself.

2. The QPE health dashboard – This shows the metrics from the event data received from QPE. Some of the metrics are projected – Error query volume, query volume etc. These metrics can be sliced and diced on all available fields on the event data.

Cost Estimates

Component	Estimate / Configuration	Monthly Cost (USD) Approx
Event Hubs	Standard Tier, 1 namespace, 2 throughput units, small retention	$20-$50 / month
Stream Analytics (ASA)	2 SUs, running 24×7 (~720 hours)	SU price around $0.178 / SU-hour for higher tier in India; so 2 × 720 × $0.178 ≈ $256 / month
Azure Data Explorer (Compute)	Dev cluster small (say 2 vCores), streaming ingest enabled	Estimate $200-$400 / month depending on cluster size & usage
Storage for ADX	Data storage + hot cache + retention (7 days)	Maybe $50-$100 / month depending on amount of data stored (compressed)
Other costs (diagnostic logs, monitoring, networking)	modest overhead	$10-$30 / month

Total expenditure would come upto ~$600 per month based on the query volume and usage.