PROJECT DOCUMENTATION

Product Analytics Agent (PAA)

End-to-End Intelligent Natural Language Analytics System

(Enhanced with Full Data Pipeline: API → Kafka → Event Hubs → Bronze → Silver → Gold → MCP → UI)

1. Introduction

This project is a full **data analytics & AI engineering solution** that combines:

- **Real-time data ingestion**
- **Databricks lakehouse architecture**
- **Bronze → Silver → Gold structured refinement**
- **Apache Spark data transformations**
- **An interpretable NL → SQL engine**
- **LLM-assisted structured analytics generation**
- **A clean web UI for querying business metrics**

The final result is a complete **Product Analytics Agent (PAA)** — a conversational interface that sits on top of our entire data pipeline and makes analytics accessible to everyone.

2. Motivation & Vision

Originally, the aim was to let business stakeholders ask natural language questions and receive immediate insights from product data. But as the project evolved, we extended the scope to build **the full pipeline**:

Real-world data ingestion

Streaming integration

Quality layers (Bronze/Silver/Gold)

LLM-powered analytics

So the final system is no longer just a query engine — it is an **end-to-end analytics platform**.

3. High-Level Architecture

External API (Product Telemetry / Usage Data)
↓
Apache Kafka (Producers)
↓
Azure Event Hubs (Kafka-Compatible Endpoint)
↓
Databricks Autoloader → Bronze Delta Tables
↓
Apache Spark ETL → Silver Delta Tables
↓
Spark Business Logic → Gold Delta Tables
↓
MCP Server (NL → SQL Engine)
↓
LLM Adapter (Optional)
↓
Databricks SQL Warehouse
↓
Clean UI (Natural Language Interface)

This is a **full lakehouse + AI application stack**, covering ingestion to consumption.

4. Data Engineering Pipeline (Day 1–20)

4.1 Raw Data Ingestion via External API

Our system begins by calling a **custom HTTP API** that produces raw product usage events.

These include:

- User actions
- Monthly subscription revenue
- Feature usage
- Login events
- Churn signals
- Retention events

A Python script periodically **fetches batches of events** and pushes them into Kafka.

Examples of ingested fields:


```
{
  "user_id": "U12345",
  "timestamp": "2025-01-13T10:40:22Z",
  "event_type": "feature_used",
  "feature": "dashboard",
  "region": "uk-england-london",
  "subscription_plan": "pro",
  "amount": 29.0
}
```


4.2 Streaming to Apache Kafka

We configured:

Topic design:

- product-events
- subscription-events
- churn-events

Kafka Producer

The API ingestor writes JSON messages directly to Kafka:

producer.send("product-events", json.dumps(event).encode("utf-8"))

High throughput

Kafka ensures durability, scalability, and buffering.

4.3 Kafka → Azure Event Hubs Integration

To move data into the cloud, we used **Event Hubs' Kafka-compatible endpoint**.

Kafka is abstracted behind:

```
bootstrap.servers = "<event-hubs-namespace>.servicebus.windows.net:9093"
security.protocol = SASL_SSL
sasl.mechanisms = PLAIN
```

This meant:

- No code change in producers
- Kafka API sends data into Azure seamlessly
- Event Hubs acts as a scalable ingestion layer for Databricks

4.4 Bronze Layer in Databricks

The Bronze layer contains **raw** unprocessed JSON data.

A Databricks **Autoloader** pipeline reads Event Hubs streams:

```
raw_df = spark.readStream.format("eventhubs") \
    .options(event_hub_conf) \
    .load()

bronze_df = raw_df.selectExpr("cast(body as string) as json_str")

bronze_df.writeStream \
    .format("delta") \
    .option("checkpointLocation", "/mnt/bronze/_checkpoints") \
    .table("bronze.product_events")
```

Characteristics:

- Raw JSON preserved
- No schema enforcement

- Duplicate-tolerant
- Immutable append-only

4.5 Silver Layer — Cleaned, Normalized Spark Tables

From Bronze, we created Silver tables using Spark structured transforms.

Steps performed:

1. JSON parsing
2. Data type validation
3. Region normalization
4. Plan standardization
5. Dropping malformed records
6. Deduplication using watermarks
7. Converting timestamps

Example transformation:

```
silver_df = bronze_df.select(
  col("json.user_id"),
  to_timestamp(col("json.timestamp")).alias("event_ts"),
  col("json.event_type"),
  lower(col("json.region")).alias("region"),
  col("json.subscription_plan"),
  col("json.amount").cast("double")
).dropDuplicates(["user_id", "event_ts"])
```

Resulting Silver tables:

- silver.product_events
- silver.subscription_events
- silver.churn_events

Silver is **clean, reliable, and schema-enforced**.

4.6 Gold Layer — Final Business Aggregates

Using Silver as source, we built six analytics-oriented tables:

1. gold.churn_rate

Churn per month, region, and cohort.

2. gold.feature_adoption

Feature usage trends across segments.

3. gold.ltv

Historical customer lifetime value.

4. gold.ltv_predicted

Machine-learning-based predictions

(using regression/ARIMA or mocked).

5. gold.mrr

MRR trends by region, month, and plan.

6. gold.user_retention_cohort

Cohort tables used in retention analytics.

Example Spark transformation:

```
mrr_df = silver_subscription.groupby("year", "month", "region", "subscription_plan") \
   .agg(
      sum("amount").alias("total_revenue"),
      avg("amount").alias("avg_revenue_per_user")
   )

mrr_df.write.format("delta").mode("overwrite").saveAsTable("gold.mrr")
```

These Gold tables are the **foundation** for we MCP analytics system.

5. MCP Server (Day 20–35)

The MCP server is our **analytics brain**.

It exposes:

- /nlq
- /query_structured
- /query
- /template/<name>

It converts natural language → structured payload → SQL → results.

Key modules:

- nl_parser.py
- llm_adapter.py
- schema_cache.py
- sql_validator.py
- main.py

6. Natural Language Parsing

Two modes:

1. Rule-based (fast, deterministic)

- Intent detection
- Canonical alias mapping
- Region inference
- Date range parsing
- Grouping logic

2. LLM-based refinement

LLM improves understanding and adjusts:

- Filters
- Grouping
- Metrics
- Explicit/implicit constraints

But **never generates SQL** — only structured JSON.

This preserves complete control & safety.

7. SQL Builder & Validation

Our system produces fully safe SQL:

Allowed:

- SELECT
- GROUP BY
- SUM/COUNT
- ORDER BY
- LIMIT

Blocked:

- UPDATE
- DELETE
- INSERT
- MERGE
- DROP

- Multiple statements
- Comments

The SQL validator ensures no injection or accidental mutation can occur.

8. Web UI (Day 35–45)

We transformed a plain index.html into a full minimalistic web app:

Pages:

- base.html
- home.html
- static/css/style.css
- static/js/app.js

Features:

- Clean black & white theme
- Brand logo (PAA) on the top right
- One text input box
- “Ask” button
- SQL output panel
- JSON output panel
- Fully responsive layout

The UI feels like an internal analytics tool used in real SaaS companies.

9. How the LLM + MCP Work Together

The LLM is **not** running SQL.

It only outputs a **controlled JSON payload**.

Pipeline:

User NL
↓
UI sends NL to MCP (/nlq)
↓
LLM Adapter (optional mode)
↓
Structured payload
↓
SQL Builder (safe)
↓
SQL Validator
↓
Databricks execution
↓
Results returned to UI

This is the safest and most enterprise-compliant pattern for NL→SQL analytics.

10. Full End-to-End Demo Query

NL:

> “Show revenue growth for London from March to July 2025, grouped by month.”

LLM or rule-based yields:

```
{
  "table": "gold.mrr",
  "group_by": ["year","month"],
  "filters": [
     {"col":"region","op":"=","val":"uk-england-london"},
     {"col":"year","op":"=","val":2025},
     {"col":"month","op":">=","val":3},
     {"col":"month","op":"<=","val":7}
  ],
  "agg": {"total_revenue_sum":"total_revenue"},
  "order_by":[{"col":"month","dir":"asc"}],
  "limit":50
```

}

SQL:

```
SELECT year, month, SUM(total_revenue) AS total_revenue_sum
FROM gold.mrr
WHERE region = 'uk-england-london'
  AND year = 2025
  AND month >= 3
  AND month <= 7
GROUP BY year, month
ORDER BY month ASC
LIMIT 50
```

Result:

Displayed in UI.

11. Strengths of This End-to-End System

Full Modern Data Stack

API → Kafka → Event Hubs → Databricks → Delta Layers → UI

Enforced Data Quality

Bronze → Silver → Gold

Safe & Controlled SQL

Never expose raw SQL execution to LLM.

Human-friendly Analytics

Any team member can ask questions using plain English.

Extremely Extensible

Add new metrics, tables, or queries easily.

Production-style Architecture

Matches design of modern SaaS data platforms like:

- Mixpanel
- Amplitude
- Segment
- Snowflake Cortex

12. Future Roadmap

- Authentication & user roles
- Row-level security
- Automatic visualizations in UI
- Slack/Teams bot integration
- Query explanations
- Suggesting next-best questions
- NoSQL or streaming analytics support (e.g., user journeys)

13. Conclusion

We have built a truly **end-to-end intelligent analytics platform**, not just an LLM project. This system includes:

- Real raw data ingestion
- Cloud-scale ETL
- Business logic layers
- NL parsing and reasoning
- Safety-critical SQL generation

- Clean professional UI

It represents **full-stack data engineering + data analytics + AI systems design** — an impressive production-grade system.