🚀 End-to-End Data Engineering Pipeline with Azure Databricks

1. Executive Summary

This project delivers a **production-ready, end-to-end data engineering pipeline** using **Azure Databricks** and follows the **medallion architecture** (bronze-silver-gold). It showcases how to build scalable, modular, and governed data pipelines using **PySpark**, **Delta Live Tables**, and **Unity Catalog**. A strong focus is placed on **reusability through object-oriented programming (OOP) in PySpark**, building an optimized **star schema** data model, and preparing analytics-ready data with governance and reliability in mind.

---

2. Project Architecture Overview

🔷 Medallion Architecture: Bronze → Silver → Gold

This layered design helps enforce data reliability, reusability, and security across ETL stages.

- **Bronze Layer** – Raw data ingestion from external sources using **Autoloader** and **Structured Streaming**, stored in **Parquet** format.
- **Silver Layer** – Data cleansing, deduplication, and transformation using **modular PySpark code with OOP principles**, stored in **Delta Lake** format.
- **Gold Layer** – Creation of an **analytics-ready star schema**, implementing **Slowly Changing Dimensions (SCD)** using **Delta Live Tables** and **PySpark upsert logic**.

---

3. Technology Stack

Component	Tool
Cloud Platform	Microsoft Azure
Data Storage	Azure Data Lake Storage Gen2

Processing Engine	Apache Spark (via Azure Databricks)
Languages	Python, PySpark
Governance	Unity Catalog
ETL Orchestration	Databricks Jobs
BI/Analytics	Power BI, Databricks SQL
File Formats	Parquet, Delta

---

4. Data Flow & Processing Pipeline

🔁 Bronze Layer – Ingestion

- Real-time ingestion using **Spark Structured Streaming + Autoloader**.
- Ensures **exactly-once** file processing using **checkpointing** and **RocksDB-backed metadata**.
- Stored in Parquet format for space and speed efficiency.

🧪 Silver Layer – Transformation

- Complex transformations using **PySpark**.
- **Applied OOP principles**:
    - Created **Python classes** for:
        - Common data validation
        - Date normalization
        - Metadata logging
        - Surrogate key generation

    - Used class inheritance and encapsulation to make transformation logic **modular and reusable** across datasets.

- Registered reusable **functions in Unity Catalog**, both SQL and Python, for company-wide logic reuse.
- Output stored in **Delta format**, enabling ACID compliance and schema evolution.

⭐ Gold Layer – Warehousing with Star Schema

- Designed a **Star Schema** consisting of:
    - **Fact Tables**: e.g., `fact_sales`, `fact_transactions`
    - **Dimension Tables**: e.g., `dim_customer`, `dim_product`, `dim_date`, `dim_branch`
- Implemented **SCD Type 1 & Type 2**:
    - Used **Delta Merge** operations to handle updates and maintain history.
    - **Delta Live Tables** handled:
        - Historical tracking with automated logic for `effective_date`, `end_date`, and `is_current`
        - Declarative ETL with **@dlt.table** decorators for cleaner transformation code
- Enabled **BI reporting** via Power BI dashboards connected to gold tables.

---

5. PySpark with Python OOP – In-Depth

Unlike traditional monolithic scripts, this project emphasizes **clean architecture** in data engineering using **Object-Oriented Programming (OOP)** in PySpark.

✨ OOP Concepts Applied:

Concept	Applied For
Encapsulation	Transformation logic encapsulated within utility classes
Inheritance	Base class created for shared logic; subclasses for dataset-specific rules
Modularity	Each transformation function was a method within structured classes
Reusability	Methods were reused across bronze/silver layers and across notebooks
Parameterization	Classes accepted configurations via constructor (`__init__`) for dynamic behavior

This structure:

- Reduced **code duplication**
- Simplified **testing and debugging**
- Enabled **unit testability**
- Enhanced **readability** and **collaboration**

---

6. Governance & Reusability

- **Unity Catalog** centralized governance:
    - Managed permissions at schema/table/function levels
    - Defined **external locations** for secure data lake access
    - Registered **custom Python and SQL functions** for transformations
- Enforced **row-level security and role-based access control** (RBAC)

---

7. Workflow Orchestration

- All processing steps were orchestrated via **Databricks Jobs**:
    - Notebook-to-notebook chaining
    - Parameter passing using **widgets**
    - Retry and timeout configurations
- Created **dynamic, parameterized notebooks** to scale across datasets and pipelines.

---

8. Challenges & Mitigation

Challenge	Resolution
Schema changes in streaming data	Handled using Autoloader’s schema evolution and default column types
SCD logic complexity	Simplified with DLT's declarative syntax and merge condition rules
Large dataset maintainability	Modularized code using OOP; used Unity Catalog for reusability
Governance for multiple teams	Applied Unity Catalog-managed catalogs and audit logs
Cluster cost control	Used job clusters and auto-termination features effectively

---

9. Business Impact & Value

- Demonstrates how to build **governed, modular, and cost-efficient** data pipelines.
- Enables **real-time data ingestion** with high reliability using exactly-once guarantees.
- Promotes maintainable architecture through **code reusability** and **clean coding practices**.
- Prepares data engineers for real-world production challenges with a complete understanding of governance, streaming, cost tuning, and ETL best practices.

---

11. Future Enhancements

- Add **unit testing** for PySpark classes using `pytest` + `chispa`
- Integrate **data observability tooling** (e.g., OpenLineage, Great Expectations)
- Expand gold model to **data marts** serving different departments
- Introduce **incremental CDC feeds** using Azure Event Hubs or Kafka
- Incorporate **infrastructure as code** via Terraform for environment provisioning

---

12. Conclusion

This Azure Databricks project is a comprehensive, end-to-end **data engineering solution** that uses modern design patterns, software engineering practices, and governance models. With robust streaming, transformation, and warehousing capabilities, it serves as a strong reference for enterprise-ready big data systems and demonstrates excellence in cloud-native data pipeline development.