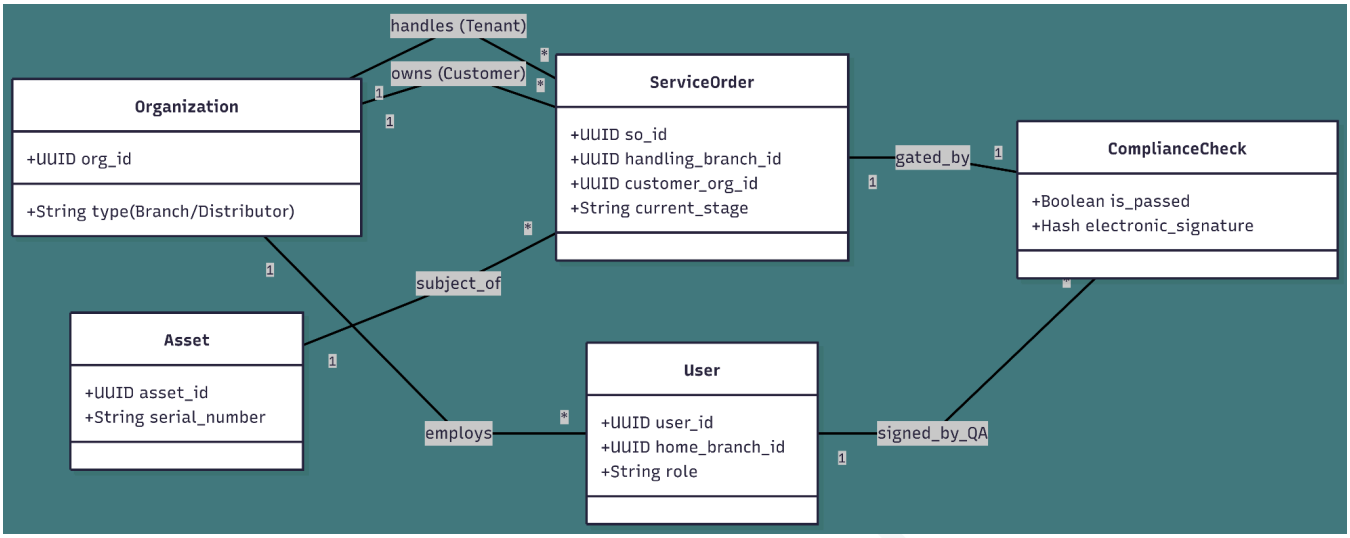
Architecting the Cognitive Enterprise: A Hybrid Transactional/Analytical Processing (HTAP) Framework for High-Value Asset Service Operations



Concept Highlight

By: Mohamed Adnan Palli Konda

02/2026

Listen to this tab

Part 1: The Strategic Data Architecture & Research Framework.
The blueprint for transforming a traditional Product Service Industry into a data-driven "Cognitive Enterprise".

Abstract

This research presents a blueprint for the **Cognitive Enterprise** transforming High-Value Asset Management (MedTech) from reactive, spreadsheet-dependent silos into a predictive, AI-native ecosystem.

By replacing the "Hidden Factory" of disjointed Excel and Smartsheet workflows with a **Hybrid Transactional/Analytical Processing (HTAP)** framework, we eliminate data latency and operational opacity. The proposed Event-Driven, Domain-Driven Design (DDD) schema does more than streamline operations; it enforces **"Compliance-by-Design."** This ensures that every digital interaction inherently adheres to **ISO 13485** and **21 CFR Part 11** standards turning regulatory audits from a risk into a routine while establishing the foundational data estate required for Machine Learning and predictive forecasting.

.

---

1. Business Process Analysis: From Reactive to Predictive

1.1. The "As-Is" State: Fragmented Silos

Currently, the industry operates in a disjointed "Hybrid-Manual" state, characterized by high friction and data opacity.

- **The Excel Trap:** Reception logs tickets in Excel; Workshops track repairs in Smartsheets; Sales manages approvals via email threads; Finance utilizes SAP. These systems are disconnected.
- **Data Latency:** Information is manually re-entered across systems, causing a 24-48 hour lag in visibility.
- **Compliance Risk:** Audit trails are scattered across physical paper and email archives, making regulatory adherence (e.g., FDA audits) laborious and error-prone.
- **Inventory Blindness:** Spare parts consumption is recorded post-repair, leading to stockouts and delayed service delivery.

1.2. The "To-Be" State: A Unified, Application-Driven Ecosystem

The proposed solution moves beyond static data storage to a dynamic Service Operating System. This transformation relies on two distinct but interconnected architectural layers: the Business Domain (the workflow engine) and the Application Domain (the user interface and security) - Capture data that is actually needed.

- **The Business Schema (Organizational Workflow):**
    - Single Source of Truth: A centralized relational database governs all core business entities (`ServiceOrders`, `Assets`, `Inventory`).
    - Event-Driven Logic: The system enforces business rules at the database level (e.g., blocking a dispatch if `QA_PASSED = FALSE`), ensuring process integrity regardless of the user interface.
    - Real-Time Traceability: A "Service Order" acts as a digital thread, linking Customer, Asset, Technician, and Invoice in real-time.
- **The Application Layer (The "Vehicle"):**
    - Web Application Requirement: To interact with this architecture, a custom Service Web Application / Power APP (Front-End) is required. This can also be implemented with smartsheet or any interconnected data management that supports this model. This replaces Excel/Smartsheet as the primary input method.
    - Application-Specific Data: The complete solution extends the data model to include Identity & Access Management (IAM) entities not detailed in the workflow schema, such as:
        - `UserCredentials` (Secure password hashing/OAuth tokens).
        - `SessionLogs` (Tracking active user login duration).
        - `RolePermissions` (Granular UI access control, e.g., "Technician View" vs. "Manager View").
        - `AppSettings` (User preferences, dark mode, default branch selection).
    - The "Human-in-the-Loop" Interface: This application layer serves as the secure gateway, translating human actions (clicks, scans) into structured database transactions.

---

2. Organizational Structure & Functional Interdependencies

The data model reflects the operational reality of a multinational Service Hub (e.g., Dubai Branch).

1. **Customer Care (The Frontline):** Responsible for Ticket Creation (ServiceOrder), Warranty Validation (AssetRegistry), and Logistics Coordination (ShipmentBatch).
2. **Technical Services (The Engine):** Responsible for Triage (TriageEvent), Execution (RepairJob), and Data

Logging (TimeLog).
3. **Commercial Operations (The Bridge):** Manages the "Approval Ping-Pong" between Internal Sales and Distributors. Handles QuoteVersion, PriceList, and PurchaseOrder validation.
4. **Supply Chain (The Backbone):** Manages InventoryTransaction, StockLevel, and Procurement.
5. **Quality & Compliance (The Shield):** A distinct oversight layer responsible for AuditLog, ComplianceCheck, and ElectronicSignature.

---

3. Internal Workflow & Data Flow Architecture

The lifecycle of a service request follows a strict state-machine logic:

1. **Initiation:**
   - *Input:* Customer Email/Portal -> *Action:* ServiceOrder Created (Status: INITIATED).
   - *Data:* Customer ID linked, Asset Warranty Status auto-verified.
2. **Triage & Inspection:**
   - *Input:* Physical Device -> *Action:* Technician logs TriageReport.
   - *Data:* Fault codes identified, Spare Parts estimated (BillOfMaterials).
3. **Commercial Negotiation (The Iterative Loop):**
   - *Input:* Cost Estimate -> *Action:* ServiceQuote (V1) generated.
   - *Branching:* If Rejected -> New Version (V2) created. If Approved -> LPO linked.
4. **Execution:**
   - *Input:* LPO Verified -> *Action:* RepairJob Starts.
   - *Data:* Parts consumed (InventoryTransaction), Labor hours logged (TimeSheet).
5. **Quality Assurance (The Gatekeeper):**
   - *Input:* Completed Repair -> *Action:* ComplianceCheck.
   - *Constraint:* System blocks dispatch until QA_PASSED = TRUE.
6. **Closure:**
   - *Input:* Dispatch -> *Action:* Invoice Generated -> Ticket Closed.

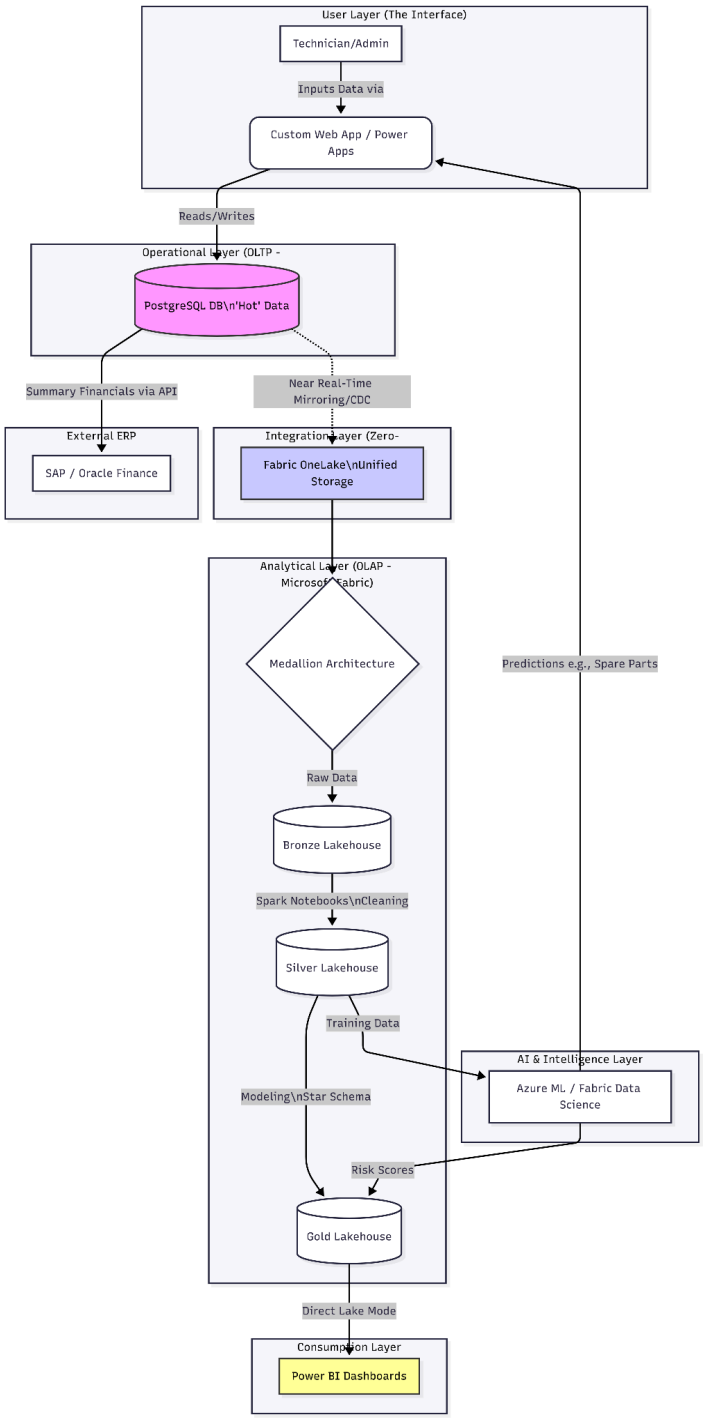


Diagram 1: High-Level End-to-End Architecture (The "Big Picture")

---

4. Database Design: A Domain-Driven Design (DDD) Approach

To ensure scalability, maintainability, and strict regulatory compliance (ISO 13485), the schema is

compartmentalized into five core high-cohesion domains.

4.1. Domain: MASTER_DATA (The Static Registry)

- Purpose: Stores immutable entities referenced by all other domains.
- Key Entities:
    - Organizations: A unified registry for all legal entities. Uses a `Type` attribute to distinguish between Internal Branches (e.g., Koltech Dubai), External Distributors (e.g., Al Futtaim Medical), and End Customers (Hospitals). This supports complex "Inter-Company" workflows where a Branch can act as a service provider to another Branch.
    - Users: Explicitly links employees to a specific `Home_Branch_ID`, ensuring that a technician in Dubai cannot accidentally access data from a European branch (Tenant Isolation).
    - Assets: The "Digital Twin" of the physical device, tracking Serial Number, Warranty Logic, and ownership.

4.2. Domain: SERVICE_OPS (The Transactional Core)

- Purpose: Manages the high-velocity operational workflow.
- Key Entities:
    - ServiceOrders: The central ticket. Includes `Handling_Branch_ID` (who is fixing it) and `Customer_Org_ID` (who owns it) to support multi-region transfers.
    - Task-Based Actors: Unlike legacy systems, tasks are linked to specific users:
        - `TriageEvents` are linked to a Senior Engineer (`Inspector_ID`).
        - `RepairJobs` are linked to a Technician (`Technician_ID`).
    - SlaPauseLogs: Tracks valid delays (e.g., "Waiting for Customer Approval") to ensure accurate KPI calculation.

4.3. Domain: COMMERCIAL (The Financial Ledger)

- Purpose: Manages the versioned history of negotiations.
- Key Entities:
    - ServiceQuotes: Supports version control (`V1`, `V2`, `V3`) to track negotiation history.
    - PurchaseOrders: Links the Customer's LPO to the Quote with a `Verification_Status` to prevent billing mismatches.
    - QuoteLifecycleEvents: An audit log capturing exactly *when* a quote was Viewed, Rejected, or Approved.

4.4. Domain: LOGISTICS (The Physical Flow)

- Purpose: Optimized for batch processing.
- Key Entities:
    - ShipmentBatches: Aggregates multiple `ServiceOrders` into a single `Tracking_Number`. Critical for Distributors shipping bulk devices.
    - InventoryTransactions: Uses double-entry logic to track part consumption against specific Service Orders.

4.5. Domain: COMPLIANCE (The Audit Trail)

- Purpose: Read-only security and verification.

- Key Entities:
    - ComplianceChecks: A "Gatekeeper" table. The system creates a hard block on Dispatch until `QA_Passed = TRUE`, signed by a specific `QA_Manager_ID`.
    - AuditLogs: A shadow table capturing the `Old_Value` and `New_Value` of every change for forensic history.

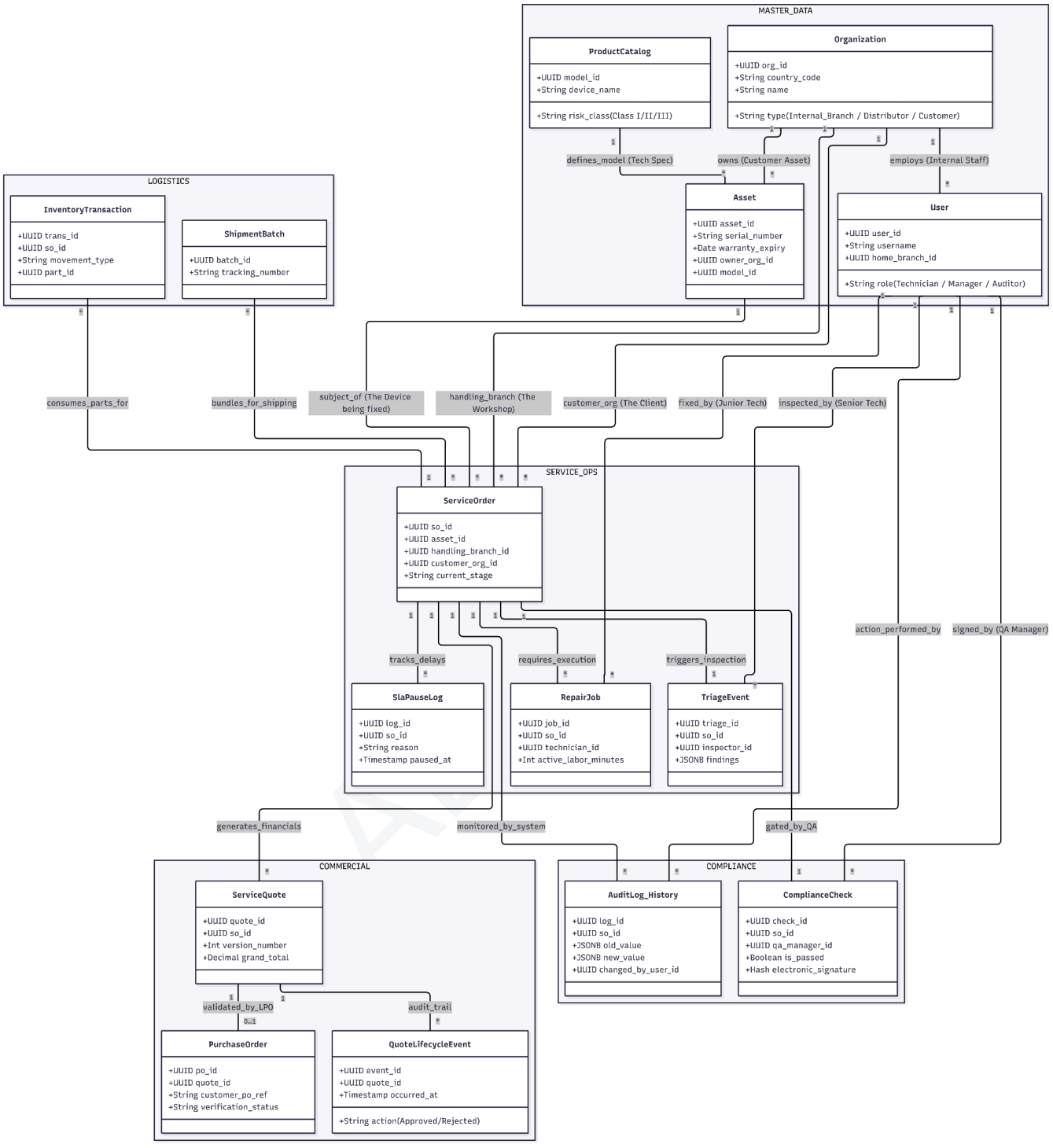


Diagram 2: Domain-Driven Database Schema (The "Engine Design")

5. Handling Non-Structural Data (Unstructured)

Operational reality involves data that cannot fit into rows and columns.

- **Architecture:** Object Storage (e.g., AWS S3 or Google Cloud Storage) linked via URL references in the DB. In the Microsoft Fabric edition (Part 2), this is handled via **OneLake Shortcuts** to Azure Blob Storage, avoiding data duplication.
- **Use Cases:**
  - **PDFs:** Customer LPOs, Signed Contracts.
  - **Images:** Photos of damaged devices (Triage), Photos of packed boxes (Logistics proof).
  - **Logs:** Raw machine logs (JSON/Text) uploaded from the device during repair.
- **Integration:** The database stores the file_url, upload_timestamp, and checksum_hash (for tamper verification), while the heavy blob sits in cost-effective object storage.

---

6. Entity Relations, Security, and Governance

6.1. Entity Integrity (The "ACID" Promise)

- **Foreign Key Constraints:** Strict enforcement (e.g., A RepairJob cannot exist without a valid ServiceOrder).
- **Cascading Rules:** RESTRICT delete operations to prevent accidental data loss (e.g., preventing the deletion of a Customer who has active history).

6.2. Security Architecture

- **Row-Level Security (RLS):** Implementation of "Tenant Isolation" at the database level. A user logged in with BranchID: DUBAI receives zero results when querying BranchID: GERMANY data, regardless of application-level bugs.
- **Role-Based Access Control (RBAC):**
  - Technician: Read/Write ServiceOrder, Read-Only Inventory.
  - Finance: Read-Only ServiceOrder, Read/Write Invoice.
  - Auditor: Read-Only *Global Access*.

6.3. Governance (21 CFR Part 11)

- **Immutable Audit Logs:** Database triggers automatically copy every INSERT, UPDATE, or DELETE action to a separate _HISTORY schema. This log is append-only and immutable.
- **Electronic Signatures:** Critical state changes (e.g., "QA Pass") require a re-authentication event, stored as a signed hash.

---

7. Performance Optimization Strategy

7.1. Read/Write Optimization

- **Indexing Strategy:**
  - **B-Tree Indexes:** On High-Cardinality columns (UUIDs, Serial Numbers).
  - **GIN Indexes:** On JSONB columns (e.g., flexible technical_parameters data).
  - **Partial Indexes:** For active workflows (e.g., WHERE status != 'CLOSED') keeps index size small & fast.

- **Partitioning:** Time-based partitioning of ServiceOrders and Logs (e.g., by Year/Month). Old data is moved to "Cold Storage" partitions to keep active queries fast.

7.2. Analytical Performance

- **Separation of Concerns:** The Operational Database (OLTP - PostgreSQL) is isolated from the Analytical Database (OLAP - BigQuery/Snowflake). This prevents heavy reporting queries from slowing down the daily operations.

---

8. Architecture for Streaming Analytics & ML Integration

This architecture facilitates the transition from "Data Entry" to "Data Science". The following architecture is based on GCP, another version with Azure and MS fabric is designed in as part 2 section.

Step 1: Change Data Capture (CDC)

- **Tool:** Google Datastream (or Debezium).
- **Process:** The system listens to the PostgreSQL Write-Ahead Log (WAL). Every change (New Ticket, Status Change) is streamed in real-time (<1 sec latency) without querying the operational database.

Step 2: The Data Warehouse (The "Lakehouse")

- **Destination:** Google BigQuery.
- **Structure:**
  - **Raw Layer:** Exact replica of PostgreSQL (JSON format).
  - **Curated Layer (dbt):** Transformed, cleaned, and denormalized tables (Star Schema) ready for BI.

Step 3: Machine Learning Pipeline (Vertex AI)

- **Training:** An automated pipeline fetches historical data from BigQuery to train models (e.g., "Predict Repair Duration").
- **Versioning:** Implementation of **MLOps** principles. Models are versioned (Model_v1.2), and performance is tracked.
- **Inference:**
  - **Batch:** Nightly predictions for "Inventory Forecasting".
  - **Online:** An API endpoint serves real-time predictions to the application (e.g., "Suggested Spare Parts" during Triage).

Step 4: Visualization (BI analytic tool - Power BI)

- **Connectivity:** Power BI / Tableau / Databrics connects to the **Curated Layer** in BigQuery via DirectQuery.
- **Output:** Dashboards reflect near real-time status (CDC latency) and embed ML predictions (e.g., "Churn Risk High" column populated by the ML model using python/R).

Listen to this tab

Part 2: The "Perfect" Enterprise Architecture (Microsoft Fabric Edition)

This is the refined technical stack. It replaces the "Generic/GCP" plan with a **Microsoft Azure + Fabric** stack. This is what you sell to a CTO.

1. The Stack Overview

- **Operational Database (The App): Azure Database for PostgreSQL (Flexible Server)**.
  - *Why?* It's open-source standard (easy to hire for) but managed by Azure.
- **Analytics Platform: Microsoft Fabric**.
  - *Why?* It unifies Data Engineering, Data Science, and Power BI into one SaaS platform.
- **The Connector: Fabric Mirroring (Zero-ETL).**
  - *The Magic:* You do NOT write python scripts to copy data. You turn on "Mirroring" in Fabric. It replicates your PostgreSQL data to **OneLake** in near real-time automatically.

2. The Medallion Architecture (In OneLake)

Once data hits Fabric (OneLake), we organize it into three "Lakehouses" to make it analytics-ready.

- **Bronze Layer (Raw Mirror):**
  - *State:* Exact replica of your PostgreSQL tables (`service_orders`, `logs`).
  - *Format:* Delta Parquet (Compressed, super fast).
  - *Action:* No transformation. Just raw history.
- **Silver Layer (Cleansed/Logic):**
  - *Action:* We use **Fabric Notebooks (PySpark)** to clean data (Now we also have **Dataflowgen 2** which is also a powerful tool for transformation).
  - *Transformation:*
    - Fix date formats (UTC conversion).
    - Calculate "Duration" from start/end timestamps.
    - Join `service_orders` with `customer` names for readability.
  - *Outcome:* "Clean" tables ready for data science.
- **Gold Layer (The Power BI Star Schema):**
  - *Action:* We model data into **Facts** (Events) and **Dimensions** (Context).
  - *Fact Tables:* `Fact_Repair_Jobs`, `Fact_Sales_Transactions`.
  - *Dim Tables:* `Dim_Customer`, `Dim_Device_Model`, `Dim_Time`.
  - *The Killer Feature:* **Direct Lake Mode.** Power BI reads this *directly* from storage. No "Import Mode." No size limits. Blazing fast.

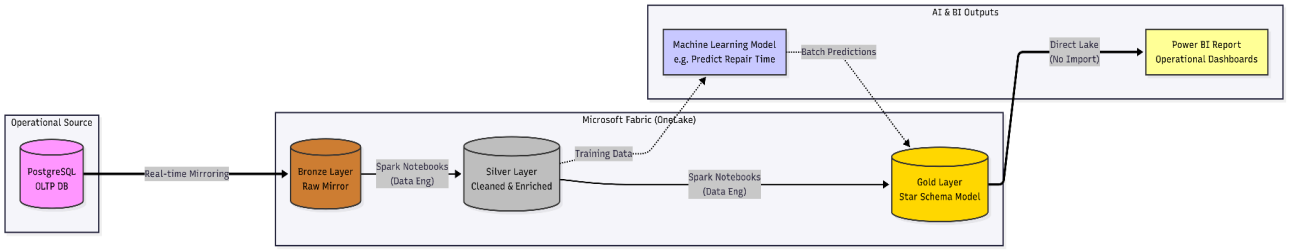


Diagram 3: The Analytics & AI Pipeline (The "Future Value")

3. Machine Learning (The AI Loop)

- **Training:** Your Data Scientist opens a Fabric Notebook, loads data from the **Silver Layer**, and trains a model (e.g., "Predict Spare Part Usage"). More AI RAG-like architecture can also be implemented.
- **Deployment:** The model is saved to **MLflow** (inside Fabric).
- **Prediction:** You run a daily batch job that predicts "Likely Failure" for every active contract and saves it to a new table in the **Gold Layer**.
- **Visualizing:** The Power BI report automatically shows a new column: "Churn Risk: High".

Part 3: But Why? (The Business Case)

Who is this for?

- **Scale:** Companies processing 500+ tickets/month. Below that, Excel is fine. Above that, Excel is a liability.
- **Industry:** MedTech, Aviation, Heavy Machinery (Caterpillar/Komatsu dealers), Oil & Gas Service.
- **Role:** COOs (who want speed), CTOs (who want security), and CFOs (who want billing accuracy).

Why implement this? (The Value Prop)

1. **Single Source of Truth:** No more "Finance says we made $1M, Ops says we did 500 jobs." The data is unified.
2. **Regulatory Immunity:** When the FDA/ISO auditor comes, you don't panic. You show them the **Gold Layer Compliance Report**. Every record is traceable.
3. **Revenue Leakage Plug:** The system physically prevents "free work." If a warranty is expired, the system forces a Quote generation.
4. **AI Readiness:** You cannot do AI on messy Excel sheets. This architecture builds the **Data Estate** required to turn on Copilot or Predictive ML tomorrow.

How does it help in Analytics?

- **Eliminates "Data Drudgery":** Analysts stop spending 80% of their time cleaning Excel files. They connect Power BI to the **Gold Lakehouse** and start analyzing immediately.
- **Real-Time Decisions:** "Mirroring" means the dashboard is only seconds behind reality, not 24 hours.

Conclusion

This architecture represents a shift from "**Systems of Record**" to "**Systems of Intelligence**". By enforcing strict data typing, auditable workflows, and decoupled analytical pipelines, the organization enables a future-proof foundation capable of advanced automation, regulatory compliance, and predictive AI integration. More entities and process can be added as needed such as maintaining internal regional regulatory policy active information and Training licence information of the engineer and more.

Author

Mohamed Adnan Palli Konda

MBA Business Analyst | MSc. Data Science | Operational Specialist

A Data Strategist specializing in AI-driven Digital Transformation for the MedTech, Manufacturing, Service, MediaTech, Automobile, BFSI, Oil & Gas Service sectors. Connect on LinkedIn for more open-source architecture discussions.

Link: linkedin.com/in/adnan-pk