DATA ENGINEER TECHNICAL TEST

Sun Valley Technical Test – Data Engineer

AUGUST 3, 2025

JULIAN ANDRES VALLEJO LONDOÑO
Julianvl.2003@gmai.com

Content

Part 1 – Strategy for Scalable Table Extraction

1. Landing & Ingestion

- **Blob Storage**
  • Set up an Azure Blob container named bronze-zone/ to receive all quarterly PDFs (2021–2025).
- **Automated Ingestion**
  • Build an ADF pipeline that triggers a Databricks notebook running your Selenium scraper:
  1. Scrape the Mineros investor site and download any new quarterly PDFs to a local staging area.
  2. Upload each new PDF into the bronze-zone/ container in Azure Blob Storage.
     • Persist an ingestion log (filename, timestamp, SHA-256) into a Delta Live Table (DLT) or Azure SQL for lineage and audit.

2. Pre-processing & PDF Type Detection

- **Classification**
  • Inspect the blob's metadata or attempt a light pdfplumber.open() to see if a text layer exists:
  - **Vector PDF** if text is present.
  - **Image PDF** otherwise.
- **Vector-PDF Path**
  • In Databricks, call a Python notebook that uses pdfplumber.open(pdf_url) to extract raw tables via the PDF's text and line objects.
- **Image-PDF Path**
  • Run OCR (Azure Cognitive Services or Tesseract via docTR) to generate searchable text + layout JSON.

• Use layoutparser (or a lightweight CV model) to detect table regions on the rendered page images.

3. Table Detection & Extraction

- **Box-drawn (“lattice”) tables**: use pdfplumber.open(pdf_url) + page.find_tables({vertical_strategy:"lines",…}) to locate cells bounded by drawing primitives.
- **Whitespace-based (“stream”) tables**: apply Camelot–stream or tabula-py on the vector text layer or on the OCR output to pick up aligned columns.
- **Merged/complex tables**: for headers spanning multiple columns or heavily formatted layouts, feed the raw table snippet into a small LLM (e.g. Donut) or use a specialized CV model to split and normalize cells.

Part 2 – Orchestration and Automation

00_Extraction_PDF Pipeline

Purpose: scrape URLs, download PDFs to bronze, track what ran, and notify stakeholders

1. **Trigger**
   - Time-based (e.g. quarterly calendar) or blob-event if we land PDFs another way.
2. **Databricks Notebook Activity** (Extract_PDFs)
   - Runs an existing extraction_notebook.ipynb logic under a Cluster-or-Job-cluster (job cluster optimize costs).
   - Saves each PDF into our Bronze storage account container (bronze-zone/…).
   - Writes an audit table (e.g. Delta table in Unity Catalog) recording: URL, download timestamp, file size, SHA-256.
3. **Logic App Notification**

   - After the notebook succeeds, ADF calls a Logic App (via Web Activity) that reads the audit Delta table and emails a summary to executives (who requested confirmation).

4. **Error Handling & Retries**
   - Configure ADF to retry the Notebook up to 3 with exponential backoff.
   - On final failure, send an urgent alert via Teams or Logic App.
5. **Outputs**
   - Raw PDFs in bronze-zone/
   - Audit Delta table bronze_zone.ingestion_log

01_Bronze_Silver_Gold Pipeline

Purpose: take those Bronze PDFs all the way to a Gold-ready dataset

1. **Execute Pipeline**
   - First step: Execute Pipeline activity to invoke 00_Extraction_PDF as an execution-pipeline.
2. **Databricks Notebook Activity** (Bronze_To_Silver)
   - Runs the bronze_to_silver.ipynb (the table-extraction code) on the same or an auto-scaled cluster.
   - Reads from bronze-zone/{year}_Q{quarter}/, writes normalized Parquet into silver/ (e.g. partitioned by year/quarter) in our Data Lake.
   - Writes a silver-layer metadata table (Delta) with row counts, schema, execution timestamp.
3. **Databricks Notebook Activity** (Silver_To_Gold)
   - Runs the business-rules notebook: enriches, cleans, or aggregates Silver into a Gold dataset (Delta or SQL), ready for consumption.

4. **Power BI Refresh**
    - ADF uses a **Web Activity** with a Service Principal against the Power BI REST API to trigger a dataset refresh.
5. **Monitoring & Lineage**
    - All Delta tables (bronze, silver, gold) live in Unity Catalog, so we get built-in data lineage and access control.
    - ADF pipeline runs are tracked in ADF's monitoring dashboard.

Variations & Alternative Deployment

- **Pure Databricks Workflows**: instead of ADF, chain these notebooks in Databricks Jobs (Notebook Tasks + Job Dependencies). Store all secrets (storage keys, service-principal credentials, API tokens) in Azure Key Vault and reference them via Databricks Secret Scopes. Unity Catalog still captures lineage.
- **HTTP vs. Selenium**: if one day there's a PDF endpoint, swap in an ADF Web Activity or Databricks Python HTTP call and skip the Selenium notebook entirely.
- **OCR/Image Tables**: later on, we can add a 4th notebook or step in Bronze_To_Silver that calls Azure Cognitive OCR or a LayoutParser model ADF/Workflows will orchestrate it the same way.

Why This Works

- **Modularity**: separate concerns ingestion + notification vs. ETL
- **Scalability**: both ADF and Databricks auto-scale for compute
- **Observability**: audit tables + ADF + Unity Catalog give you end-to-end lineage
- **Extensibility**: you can plug in OCR, AI/LLM-based cleanup, or switch to HTTP ingestion without reworking the orchestration

Part 3 – Gold Layer Design for ML & LLMs

To build a Gold layer that serves both classical ML and LLM/RAG use cases, we need a flexible, well-governed datastore of high-quality, semantically consistent records. Below is a sketch of how I'd structure it:

1. Rule-Driven Normalization & Enrichment

Rather than treating every extracted table identically, implement a table-type registry in the Bronze_To_Silver logic:

- **Identify table categories** (Income Statement, Balance Sheet, Cash Flow, Subsidiaries, etc.) via header patterns or an LLM classifier.
- **Apply per-type cleaning rules**:
  - Income Statement: ensure "Revenue" and "Cost of Sales" map to canonical row_labels.
  - Balance Sheet: enforce assets = liabilities + equity check, fill missing sub-totals.
- **Link metrics across quarters** by assigning a stable entity_id (e.g. company) and metric_code for each row_label, so time series queries can join on (entity_id, metric_code).

This yields a clean Silver with strongly typed fields:

entity_id, metric_code, quarter, numeric_value, currency, extracted_on

2. Multilingual & Standardized Formats

- **Currency**: store as ISO 4217 codes (e.g. "USD", "COP").
- **Dates/Quarters**: use an ISO timestamp or standardized YYYY-Q# string.
- **Taxonomy**: maintain a lookup table that maps metric_code: human-readable labels in English and Spanish (for both UI and LLM prompts).

A small dimension table in Gold might look like:

metric_code	label_en	label_es
REVENUE	"Revenue"	"Ingresos"

TOTAL_ASSETS	“Total Assets”	“Activos Totales”

3. **Gold Tables for ML Workloads**

    - **Wide-table for time series**: pivot the long Silver into one row per (entity_id, quarter) with columns for each metric_code value. Ideal for regression, anomaly detection, forecasting.
    - **Delta Lake**: store these as partitioned (year, quarter) Delta tables with schema enforcement and ACID guarantees.
    - **Feature Store Integration**: register key metrics into a Feature Store (Databricks Feature Store or Azure ML Feature Store) so downstream models can easily consume them.

Gold Outputs for RAG & LLMs

- **JSONL documents**: emit one JSON per analytical record:

```
{
  "entity_id": "MINEROS",
  "quarter": "2025-Q1",
  "metrics": {
    "REVENUE": 123456,
    "NET_INCOME": 7890,
    …
  },
  "currency": "USD",
  "date_extracted": "2025-04-15T12:00:00Z"
}
```

- **Embeddings-ready format**: store a flattened table of (doc_id, metric_code, value, context_text) for generating vector embeddings and building a RAG index.

- **Metadata**: include provenance fields (source_pdf, page_number, pipeline_run_id) so any LLM query can trace back to original document snippets.

5. Automation & Governance

- **Delta Live Tables** or **Purview integration** to track lineage and enforce quality rules.
- **Versioning**: tag each Gold dataset with the pipeline run date and Git commit of the notebooks.
- **Access Control**: leverage Unity Catalog – grant analysts read-only on Gold; data scientists on both Silver & Gold.

Result: a layered Gold repository that is:

- **Consistent** (all metrics follow the same codes & formats)
- **Linked** (time series are easily joined across quarters / entities)
- **Multilingual** (UI and LLM prompts can select English or Spanish labels)
- **ML-ready** (wide tables, feature store integration)
- **LLM-ready** (JSONL export, RAG-friendly embedding artifacts)

Part 4 – Cloud Architecture in Azure

Storage

- **Azure Blob Storage**:
  - Raw PDFs in the bronze-zone/ container
  - Silver Parquet files in silver-zone/
  - Gold outputs (JSON, wide tables) in gold-zone/
- **Azure Data Lake Storage Gen2** (hierarchical namespace) to host Delta Lake tables for Silver and Gold.

Compute

- **Azure Databricks**

    - Run Notebook jobs for:
        - PDF scraping & ingestion (Extraction notebook)
        - Table extraction & normalization (Bronze→Silver notebook)
        - Business-rule enrichment & RAG/ML exports (Silver→Gold notebook)
    - Auto-scaling clusters + Unity Catalog for data governance
- **Azure Functions**
    - Lightweight orchestrated tasks (e.g., post-pipeline notifications, RAG index kicks)
- **Azure ML**
    - Train and serve ML models (time series, anomaly detection) on your Silver/Gold datasets

Orchestration

- **Azure Data Factory (ADF)**
    - Two pipelines:
        1. 00_Extraction_PDF – triggers the scraping notebook and writes ingestion logs
        2. 01_Bronze_To_Gold – chains Extraction, Bronze→Silver, and Silver→Gold notebooks
    - Uses time or blob-event triggers, built-in retry policies, and monitoring

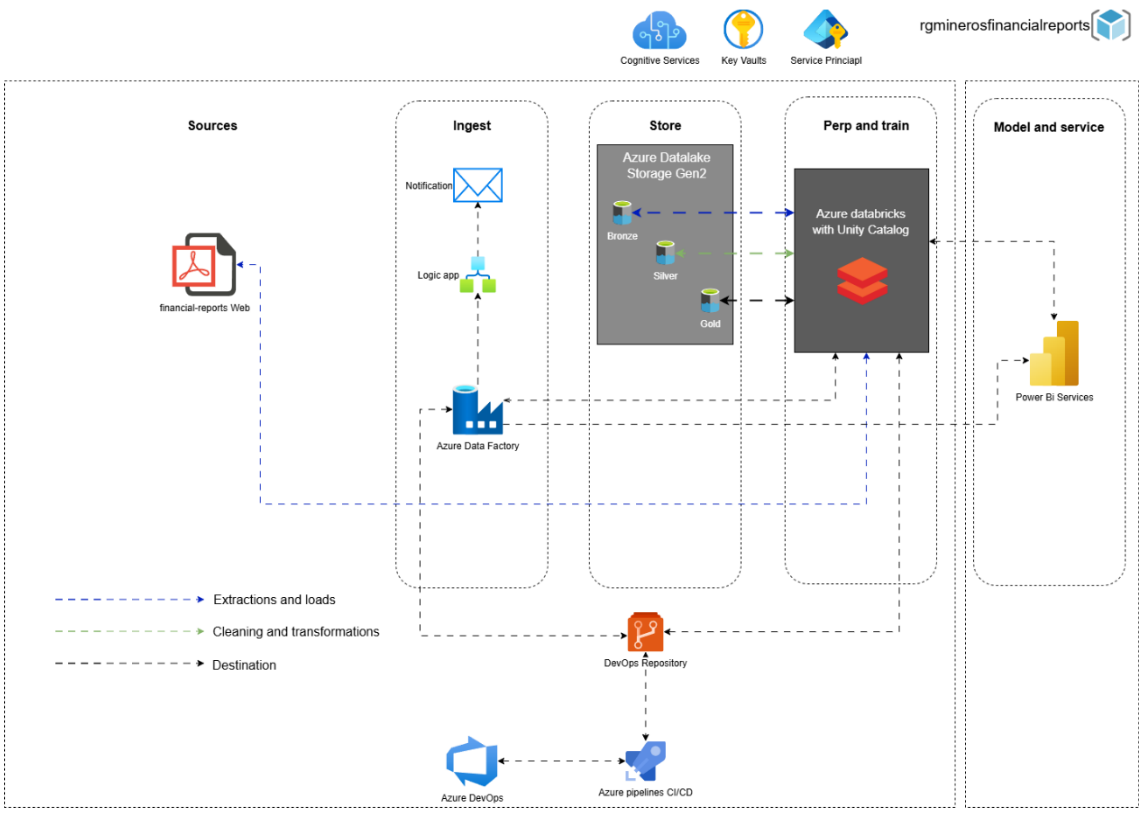
Security

- **Azure Key Vault**
    - Securely store credentials (storage keys, service principal secrets, API tokens)
- **Managed Identities & Service Principals**
    - Grant ADF and Databricks secure access to Key Vault and storage without embedding secrets
- **RBAC & Unity Catalog**
    - Enforce least-privilege access on storage containers and Delta tables

Monitoring & Logging

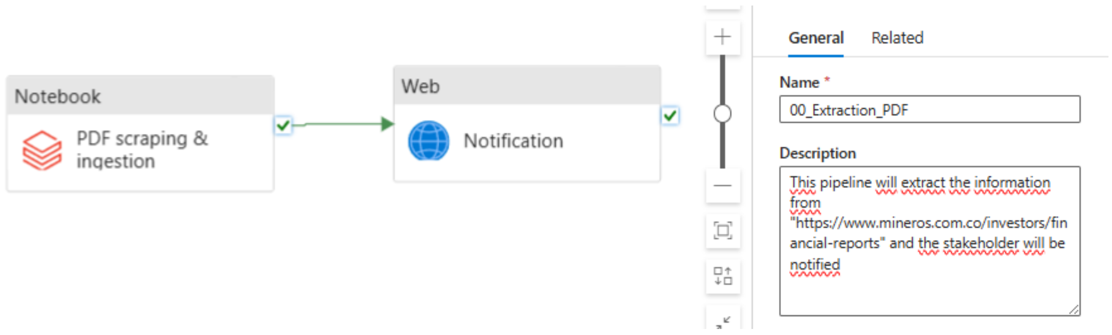
- **Azure Monitor & Log Analytics**
  - Aggregate logs from ADF, Databricks, and Functions
  - Create dashboards and set alerts for failures or SLA breaches
- **Application Insights**
  - Instrument custom Python code or Functions for detailed telemetry and performance tracing

Solution architecture



Orchestration architecture

00_Extraction_PDF



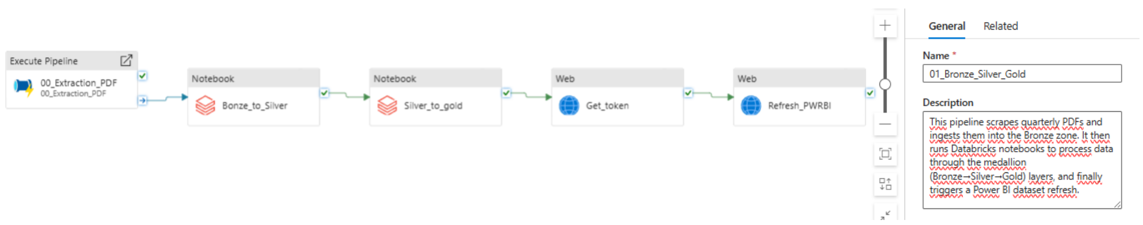
Name *

00_Extraction_PDF

Description

This pipeline will extract the information from "https://www.mineros.com.co/investors/financial-reports" and the stakeholder will be notified

01_Bronze_Silver_Gold



Name *

01_Bronze_Silver_Gold

Description

This pipeline scrapes quarterly PDFs and ingests them into the Bronze zone. It then runs Databricks notebooks to process data through the medallion (Bronze→Silver→Gold) layers, and finally triggers a Power BI dataset refresh.