A blue and black logo

AI-generated content may be incorrect.



**CDM ETL ASSESSMENT REPORT**

This report delivers a comprehensive assessment of Capital Group’s existing ETL environment, currently built on Informatica PowerCenter and orchestrated through Autosys. The evaluation focuses on workflows supporting Customer Data Management (CDM) and Salesforce (SFDC), with a detailed analysis of job inventories, transformation complexities, and performance metrics. This assessment offers critical insights into the structure and usage patterns of the current Informatica workflow inventory and serves as a foundation for informed modernization planning.

The findings support a strategic shift toward cloud modernization. The proposed approach aims to rearchitect the ETL landscape into a modern, scalable, and cloud-native ecosystem. It emphasizes automation, reduction of technical debt, and implementation of adaptable frameworks that support future scalability.

This transformation initiative is designed to equip Capital Group with a resilient, efficient, and forward-looking data integration platform—enhancing operational performance today while laying the groundwork for sustained innovation and growth.

**@Date: 15-05-2025 | @Version: 1.0**

**A black and white logo

AI-generated content may be incorrect.**

**Version History**

* The document evolved from initial draft (v0.1) to v0.8 over multiple iterations.
* Key contributors: Kuntal, Nikhil, Pradip, Yesudas John.
* Each version added more detail, from inventory to incident analysis and modernization strategy.

| **Date**  📅 | **Version** | **Author(s)**  ✍️ | **Description**  📄 | **Reviewed By**  🔍 |
| --- | --- | --- | --- | --- |
| 2024-04-01 | **0.1** | 👤 Kuntal | Initial draft of assessment scope and objectives | * Yesudas * Nagesh * Arindam |
| 2024-04-07 | **0.2** | 👤 Yesudas | Incorporated review comments and methodology updates | * Nagesh * Samse * Arindam |
| 2024-05-01 | **0.5** | 👤 Kuntal, 👤 Pradip,  👤 Nikhil | Added detailed inventory breakdown and job categorization | * Nagesh * Samse |
| 2024-05-05 | **0.6** | 👤 Kuntal, 👤 Pradip,  👤 Nikhil | Updated business impact section and removed redundant jobs | * Nagesh * Arindam |
| 2024-05-07 | **0.7** | 👤 Kuntal, 👤 Pradip,  👤 Nikhil | Added job and incident performance analysis | ---— |
| 2024-05-14 | **0.8** | 👤 Kuntal, 👤 Pradip | Finalized job analysis and added modernization recommendations | ---— |
| 2024-05-15 | **0.9** | 👤 Nikhil | Finalized cloud modernization detail  Recommendations and formatting | * Arindam |

**Section Overview**

**Executive Foundations**

1. Executive Summary

2. Scope & Objectives

3. Assessment Methodology

**CDM ETL Current State & Workload Analysis**

1. Workflow Inventory Overview

* Inventory Breakdown (High Level)
* Source system wise job count

2. Functional Categorization

* Categorization of Power Centre Jobs
* Inbound Jobs categorization based on ETL layers

3. Transformation Complexity

* Mapping Complexity Description

4. ETL Layer Job Analysis

* Initialize layers Jobs Analysis Report
* Pre-land layers Jobs Analysis Report
* Today layers Jobs Analysis Report
* Delta Layer Job Analysis
* Publishing Jobs

5. Job Performance & Utilization

* Inbound Jobs Performance Analysis
* Jobs that did not run after Sep 2024

6. Operational Stability

* Last 1 year Incident Analysis –
* High Level Inbound Dataflow

**Cloud Modernization Plan**

1. Migration of Informatica Workflows to Py Spark

2. Deploying the Converted Pipelines to AWS – Cloud Modernization

**Evaluation of Informatica Mapping Conversion Methods**

1. Metadata-Driven Common Ingestion Framework

2. Data Switch Tool-Based Conversion

3. AI-Driven Generator Tool

4. Hybrid Approach

**Approach Evaluation for Migrating Informatica Mappings**

1. Comparative Analysis: Hybrid Approach vs Data Switch

2. Comparative Analysis: Metadata-Driven Framework vs Data Switch

3. Comparative Analysis: AI-Driven Generator Tool vs Data Switch

**Current State Analysis**

1. Job Volume: High Workflow Density

2. Data Volume: Moderate-High Throughput Per Job

3. Delta Load Strategy: Incremental Processing via Separate Workflows

4. Execution Frequency: Non-Uniform Load Distribution

5. Database Environment: Azure-Hosted

6. Current Infrastructure Setup: On-Prem & Custom-Managed

**Cloud Deployment Approaches Considered**

1. Cloud Deployment Approaches Considered

2. Detailed Architecture Discussion

* **Approach 1**: AWS Glue + Apache Airflow
* **Approach 2**: AWS Glue + AutoSys

|  |
| --- |
|  |

🧾

1. **Executive Foundations**

**1.1 Executive Summary**

This assessment presents a strategic evaluation of Capital Group’s enterprise ETL ecosystem, focusing on the modernization of data integration workflows supporting Customer Data Management (CDM) and Salesforce (SFDC). The current architecture, powered by Informatica PowerCenter and orchestrated through AutoSys, spans multiple transformation layers, including ingestion, staging, processing, and publishing.

The analysis uncovers significant opportunities to streamline operations, improve system efficiency, and reduce the technical debt associated with legacy platforms. Key challenges include complex dependency chains, rigid transformation logic, limited automation, and growing maintenance overhead—factors that hinder scalability and adaptability in a cloud-first environment.

To address these challenges, a hybrid modernization strategy is recommended. This strategy advocates for transitioning from legacy ETL tools to a cloud-native architecture powered by PySpark, supported by scalable orchestration and automation frameworks. The approach is centred on building reusable, metadata-driven pipelines that promote modularity, transparency, and long-term maintainability.

This transformation represents more than a technical uplift—it is a strategic enabler for Capital Group’s broader goals of agility, innovation, and operational excellence. By embracing modern data engineering practices and scalable infrastructure, the organization will be better equipped to deliver timely insights, reduce risk, and support future business growth.

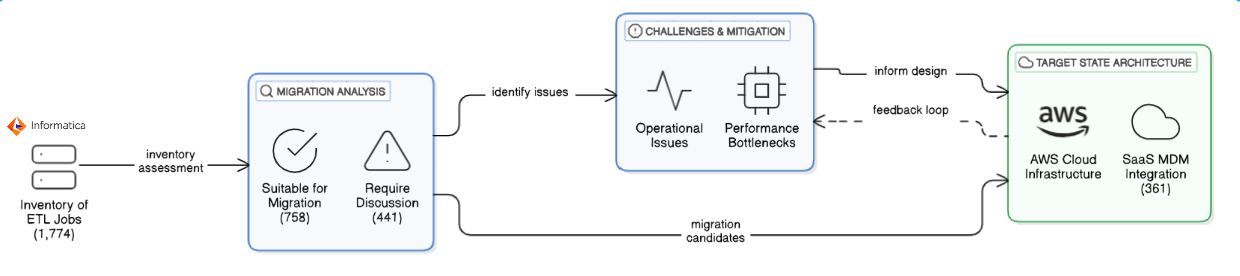
## 1.2 Scope & Objective

**Scope:**

The assessment encompasses the entire ETL integration landscape involving Salesforce (SFDC) and Customer Data Management (CDM), specifically targeting workflows and mappings currently executed in the on-premises Informatica PowerCenter Production environment.

**Objective:**

* 1. Perform an in-depth inventory analysis of Informatica PowerCenter ETL components to identify optimal migration candidates to AWS.
  2. Document existing operational and performance challenges, recommending actionable mitigation plans.
  3. Propose a robust and scalable cloud-based architecture using PySpark, AWS Glue, and AWS Airflow (MWAA), integrating Master Data Management (MDM) services delivered as SaaS solutions



**1.3 Assessment Methodology**

The assessment methodology presents a structured and comprehensive approach to thoroughly evaluate the current state of CDM ETL workflows. It focuses on identifying inefficiencies, redundancies, and technical debt within the existing environment. This methodology also highlights opportunities for modernization and automation, ensuring alignment with evolving business needs. Furthermore, it assesses the overall ecosystem’s preparedness for transitioning to a scalable, cloud-native architecture that leverages modern data integration tools and frameworks for improved performance, flexibility, and maintainability.

**1.3.1 Assessment Data Sources**

|  |  |  |  |
| --- | --- | --- | --- |
| Referenced to extract existing documentation, business logic definitions, and mapping specifications for Informatica workflows. | Used to identify job schedules, command types, execution history, and inactive workflows for ETL readiness analysis. | Reviewed to analyse ETL mappings, transformation complexity, and workflow dependencies across CDM layers. | Accessed to validate source and target schema structures, assess data volumes, and support end-to-end data flow tracing. |

**1.3.2 Data Collection**

**Confluence**

Documentation available on Confluence page Team has followed the existing documentation available in confluence

[MSS CLIENT DATA MASTER - MSS CLIENT DATA MASTER - Confluence](https://confluence.capgroup.com/display/CDM/MSS+CLIENT+DATA+MASTER)

*CTRL + CLICK*

**Autosys Jobs portal**

Team has used PRD instance of Autosys portal - [Autorep Browser - PD1](http://autorep-cpz:8000/autorep_pd1.html) to list all the CDM jobs currently configured to run in production.

**Informatica repository for CDM**

Repository – pc105\_repo\_dev2\_3

The folder structure below was followed for analyzing the workflows and mappings

|  |  |
| --- | --- |
| Source | Folder |
| **SFDC** | 📁 CDM\_DEV3\_SFDC |
| **SFDC\_LEAD** | 📁 CDM\_DEV3\_SFDC |
| **SC** | 📁 CDM\_DEV3\_SSC |
| **Salesconnect** | 📁 CDM\_DEV3\_SSC |
| **PO** | 📁 CDM\_PO |
| **DMI** | 📁 CDM\_EACG |
| **DORIS** | 📁 CDM\_DEV3\_PRELANDING |
| **FC** | 📁 CDM\_DEV3\_FC |
| **TRAC** | 📁 CDM\_DST |
| **TA2000** | 📁 CDM\_DST |
| **Brightscope** | 📁 RPM\_DEV |
| **EI** | 📁 CDM\_EI |
| **RPA** | 📁 CDM\_RPA |

**SQL Server instance (CSSCDM/ORX)**

|  |  |
| --- | --- |
| **SQL Server** | w908925\CGSQL |
| **Schema Name** | |
| **MSSCDM\_PRD/DEV** | Preland, Today, Previous layer |
| **CMS\_ORX\_10\_3/DEV3** | Landing layer tables |

**1.3.3 Autosys Jobs**

**Command Type** – Looking at the Autosys JIL file we categorized jobs based on command

types like sh/ksh/pl calling the shell or perl scripts, ctl calling the powercenter workflows

and fw file watcher are the jobs for monitoring the files.

**Not Running Jobs** – Identify the complete list of jobs (X) and jobs that are not running in

last 6 months (Y). The X-Y will give the list of jobs that need to be confirmed by CG on

migration readiness.

**Long Running Jobs –** Extract the duration of jobs for last 6 months. Find the median

values for each job. Order descending based on duration. Get the confirmation from CG

for an acceptable duration. The jobs falling over the acceptance level need to be identified as long-running jobs.

**1.3.4 Informatica Workflows**

**Categorization based on layers and functionality –** The ETL landscape is consist of

multiple layers like preland, today, previous and delta. Additionality functionality wise

we can categorize the workflows like – ingestion, MDM, publish, report etc.

**Categorization of workflows based on complexity –** The entire ingestion workflows can be categorized as Simple/Medium/Complex based number of transformation and type of transformation used.

**📊**

**2. CDM ETL Current State & Workload Analysis**

**2.1 Workflow Inventory Overview**

This section presents a high-level view of the existing ETL job landscape, helping to quantify and categorize the scope of workflows involved in the current CDM environment. Understanding job distribution across tools and systems is a foundational step in assessing modernization readiness and migration strategy.

**2.1.1 Inventory Breakdown (High Level)**

The total set of jobs has been grouped into four primary categories based on their functionality and implementation:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Total | Informatica PC | Scripts | File Watcher | Others |
| 2334 | **1669** | **587** | **34** | **44** |
| 100% | **72%** | **25%** | **1%** | **2%** |

|  |  |
| --- | --- |
|  | **PowerCenter Jobs:** A total of **1,669 jobs** fall under this category. These jobs are designed to execute Informatica PowerCenter workflows as part of the ETL process.  **Scripts/MDM Jobs:** This group includes **587 script-based jobs** (.sh, .ksh, .pl) that perform operations such as file transfers, event monitoring (e.g., file arrival), and service status checks.  **File Monitoring Jobs:** A total of **34 jobs** are specifically configured to monitor files for availability or changes.  **Uncategorized (Others):** There are **44 jobs** that could not be classified due to the absence of identifiable workflows or scripts. |

**2.1.2 Source system wise job count**

The job list was extracted using <http://wlautility-primary-cpz-prd-pd1:8000/art.html>. The categorization below is based on Jobs run between last year October. (10/24) till March 2025. There are other sources, but the in the below picture we are showing the most contributing sources.

**Job Count as per source**

**DST**

**16**

**DORIS**

**27**

**DMI**

**60**

**CDM**

**520**

**Bright Scope**

**84**

**EM**

**POWER**

**31**

**EL**

**49**

**MMD**

**23**

**IPS**

**77**

**22**

**FC**

**SALES**

**CONNECT**

**RPA**

**56**

**SFDC**

**93**

**SC**

**25**

**46**

**RPH**

**21**

**TA**

**2000**

**SSC**

**SFDC**

**LEAD**

**12**

**TRAC**

**79**

**61**

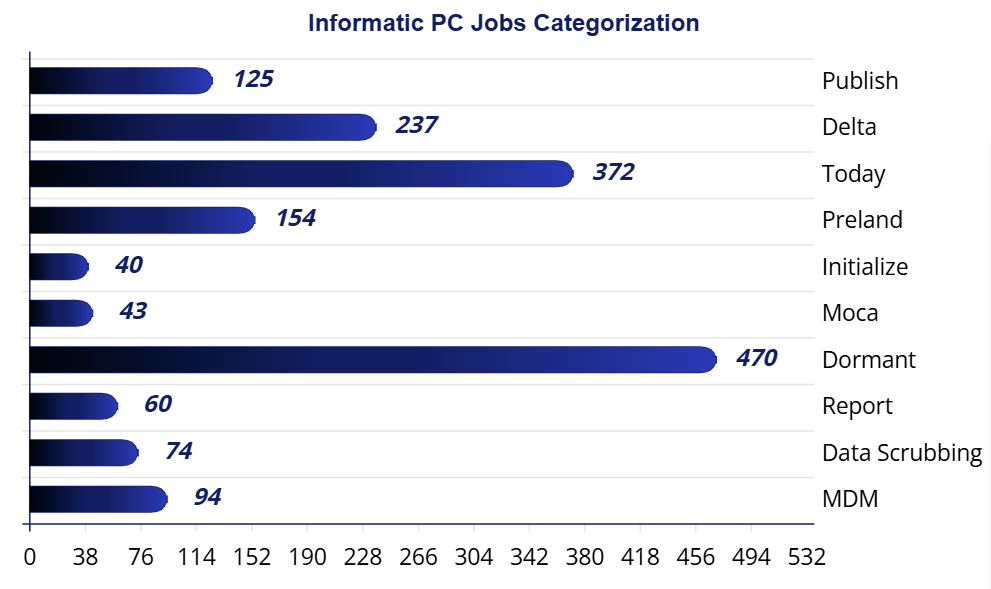
**SSC**

**25**

**2.2. Functional Categorization**

**2.2.1 Categorization of Power Center Jobs**

The complete set of PowerCenter jobs can be categorized into the following groups, based on their functional roles, execution patterns, and relevance to the overall data processing pipeline. This classification helps in better understanding the current ETL landscape, identifying modernization opportunities, and streamlining migration efforts. By organizing the jobs in this manner, we can prioritize transformation efforts, manage dependencies more effectively, and ensure critical business processes are preserved during the transition.



|  |  |
| --- | --- |
| **📥** | **Inbound Jobs**  A total of **803 jobs** fall under this category. These jobs are responsible for **loading data into MDM**, typically moving it from files to the **pre-land zone**, then to the **“today” layer**, and finally into the **MDM landing area**, sourcing from various external systems. |
| **🧩** | **MDM Jobs**  These are **PowerCenter workflows** triggered from the MDM layer, with **94 jobs** identified in this category. |
| **🧼** | **Data Scrubbing Jobs**  These jobs handle **data cleansing and enrichment** either before or after loading into MDM. Tasks may include **survivorship logic**, **flag settings**, and other **data quality operations**. |
| **💤** | **Dormant Jobs**  Jobs in this category have not been executed for an extended period and are considered **inactive**. |
| **📤** | **Publish Jobs**  These jobs **distribute processed data** to downstream systems and users. They typically represent the **final stage** in the data pipeline. |
| **📊** | **Report Jobs**  These jobs generate **intermediate datasets** for further processing, **reporting**, or **sending outputs via email**. |
| **🚫** | **Moca Jobs**  These jobs are currently considered **out of scope** for this analysis. |

**2.2.2 Inbound Jobs categorization based on ETL layers**

|  |  |
| --- | --- |
| Uploaded image | This image presents a clean and structured flowchart titled "Inbound Jobs  Categorization Flow", visually breaking down the classification of inbound data processing jobs. It clearly separates job types—   * Initialize * Preland * Today * Delta   —using distinct colours, icons, and counts, offering a high-level overview of data processing stages in a simplified and intuitive format. |

**Initialize Jobs** – These are precursor to load any source file. It removes the existing data from the previous layer and copy data from today’s layer to the previous layer.

**Preland Jobs** – These jobs load data from file to preland tables. There is no transformation except changing the date to PST time zone. So, there are number of preland tables equal to the number of source files. Preland load is delete and loads, it deletes the existing data in table and then loads the current data.

**Today Jobs** – Today layer data model is like MDM data model. In this layer data is maintained at business entity level like Party, Address, Roles etc. While loading the data in this layer multiple tables are joined together, and results are passed through multiple transformation as per business rules and finally populated into today layer tables. The loading is deleted and load, all the existing data is deleted and then loaded into this table.

**Delta Jobs** – This job calculates the change between previous day snap and today’s snap and mark the record as Insert Update or Delete. The resultant data is loaded into MDM C\_LDG tables. Here also, before loading all the data existing data is deleted and loaded into C\_LDG tables.

**2.3 Transformation Complexity**

**2.3.1 Mapping Complexity Description**

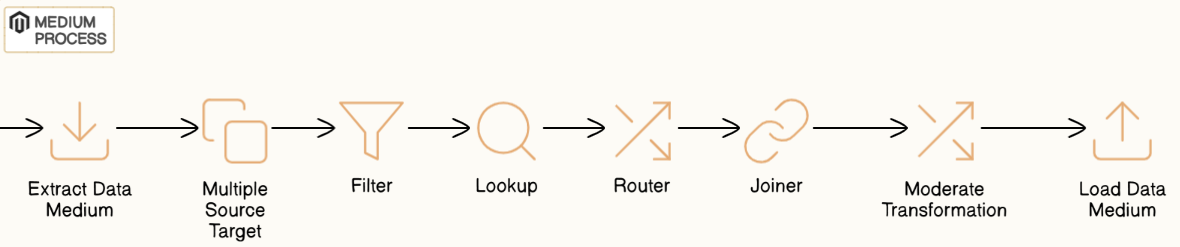
Measuring the complexity of a mapping can be approached by evaluating several factors related to transformations. Based on assessment, here are some key aspects that have been considered.

1. **Number of Transformations**: The more transformations a mapping contains, the more complex it is. Each transformation adds to the processing time and resource usage.
2. **Type of Transformations**: Different transformations have varying levels of complexity. For example, an Aggregator transformation, which performs calculations on groups of data, is generally more complex than a simple Filter transformation.
3. **Transformation Logic**: The complexity of the logic within each transformation also matters. Complex expressions, multiple conditions, and extensive use of functions can increase the complexity.
4. **Dependencies and Links**: The number of links between transformations and the dependencies among them can also contribute to the complexity. More links and dependencies can make the mapping harder to manage and optimize.

|  |  |
| --- | --- |
|  | **Simple**  Basic data mappings where data is directly loaded from source to target with minimal or no transformation logic. Ideal for straightforward file/table transfers. |

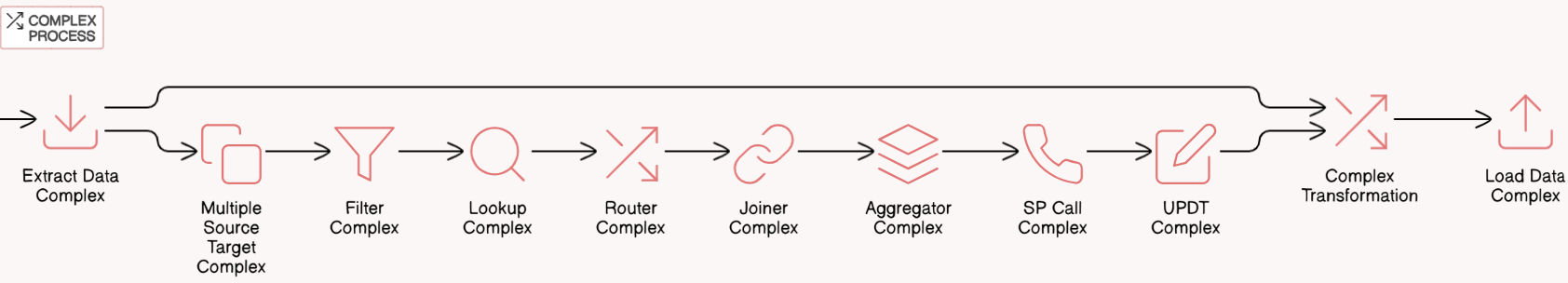
**🟠 Medium**

Moderate complexity mappings involving multiple sources and targets. Includes transformations like filters, lookups, routers, and joiners to shape the data before loading.



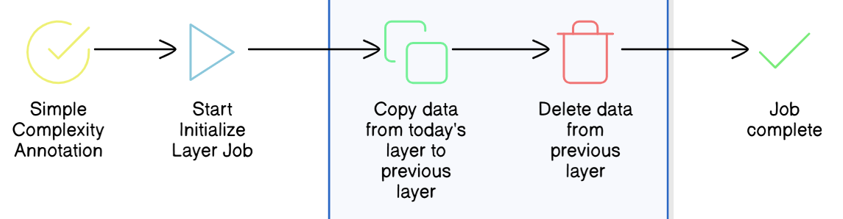
**🔴 Complex**

Advanced mappings combining multiple sources and targets with complex logic such as aggregations, stored procedure calls, updates, and layered transformation rules.



**2.4 ETL Layer Job Analysis**

**2.4.1 Initialize layers Jobs Analysis Report**

****

**Initialize Layer Jobs –** There are 19 jobs falling under this category. The functionality of this job is to copy the original data in today’s layer to previous layer and delete the data from the previous layer.

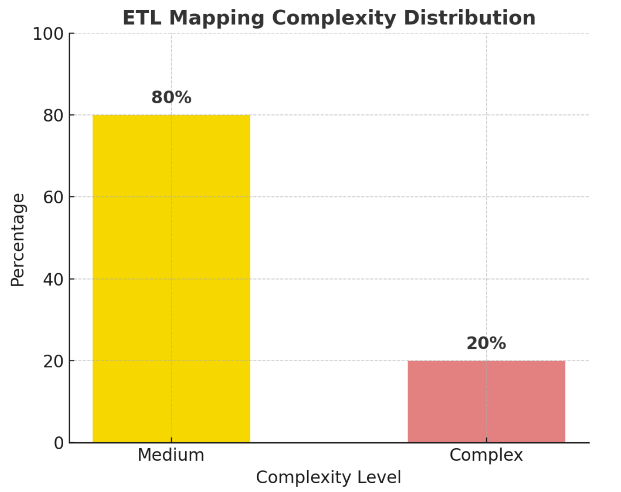
**Complexity – 10**0% of the total jobs in this category falling under simple.

**👁️ Observation –**

* Copying the data from one table to another.
* Deleting huge data volume each time will create free space fragmented. Also, the DB stats will be stale if stats are not gathered immediately.

|  |
| --- |
| **💡 Recommendation –**   * Implement partitioning strategy at table level using application key or source system. * Execute data gather stats scripts immediately after removing data. * Rebuild the index if any. |

**2.4.2 Preland layers Jobs Analysis Report**

**Preland Layer Jobs**

– There are 120 jobs pulling the data from source file and dumping into preland tables with same structure as file.

**Complexity**

– 80% of the total jobs are simple as it’s dumping the data from source file to preland tables. There are a few sources, like TA2000 and TRAC where delta data is loaded and before loading lookup is done to determine the delta. This category of jobs is falling under medium complexity.

***👁️ Observation***

* Copying the data from file to table.
* No transformation logic implemented as is copy of data.
* Only converting the date from source to Timestamp with PST time zone.
* Every time, source file format changes, new code needs to be developed.
* For some sources, only incremental data is being loaded.

|  |
| --- |
| **💡Recommendation**   * Without changing the architecture, date conversion can be included while reading the data from file to target data frame. * If common ingestion framework will be used, it will reduce individual mappings for each file and each source. |

**2.4.3 Today layers Jobs Analysis Report**

|  |  |  |
| --- | --- | --- |
| **30 %** | **60%** | **10%** |
|  | | |

**Today Layer Jobs**

**–** There are 480+ jobs populating the data from preland tables into today layer tables. Multiple joins within preland tables are done while generating the dataset. All business rules and logic is implemented in this layer,

**Complexity**

**–** There are 60% of the mappings using medium complex transformation (Sales connect, SFDC) and complex SQL queries to select the data from multiple sources. 30% are complex as there are multiple sources and number of transformations is more. Some cases SP call is also being used.

**👁️Observation**

* Load the data from pre-land to Today layer by joining multiple preland layer tables.
* Most of the business rules are implemented in this layer.
* Removes all the data from the Today layer before loading.
* Lookup override is used in most of the unconnected lookups.
* Stored Procedure call is being made in this layer.

|  |
| --- |
| **💡Recommendation**   * Implement partitioning strategy at table level using application key or source system. * Execute data gather stats scripts immediately after removing data. * If the common ingestion framework will be used, it will reduce the number of mappings. * For certain cases, we can merge the transformation logic to minimize creating temporary datasets. |

**2.4.4 Delta Layer Job Analysis**

**Delta Layer Jobs –** There are 185 jobs pulling the data from source file and dumping into preland tables with same structure as file.

**Complexity –** The majority of the jobs are of medium category as those are using SQL override at source joining multiple tables and calculating the CDC by comparing today & previous layer, followed by Union, Aggregator and Sorter.

**👁️Observation –**

* Identify the change data by comparing previous & today table and mark each record as “Insert”, “Update” or “Delete”.
* Update Strategy transformation is used but not used optimally as identification is done at SQL query level.
* Delete all the records from C\_LDG tables and load full data with change flag.

|  |
| --- |
| **💡Recommendation –**   * Implement partitioning strategy at table level using application key or source system. * Execute data gather stats scripts immediately after removing data. * Remove unused Update Strategy transformation where only Insert is considered. |

**2.4.5 Publishing Jobs**

**Publish Layer Jobs –**

There are 140+ jobs (105 unique jobs) involved in generating publishing extracts, either full or delta reports which are published to various clients at different scheduled time T1, T2, T3 as per their requirement.

**Complexity –**

The majority of the jobs are of medium category as those are using multiple lookup tables and joiner, aggregator transformation to derive meaningful and user understandable data for an entity.

**👁️Observation –**

* Target is flat file type in 90% of the jobs
* Identify the change data by comparing previous & today table and mark each record as “Insert”, “Update” or “Delete” for delta reports
* Full reports are created on Today layer tables
* Duplicate records are being traced using router or aggregator and rejected rows are being maintained in a common reject table
* Date columns like create date, last update date, deleted date, EffectiveStartDate, EffectiveEndDate is being converted from Pacific date to UTC MS1 format
* Filter transformation is used but has no filter value in most of the pipelines

|  |
| --- |
| **💡Recommendation –**   * Remove Filter transformation where no filter condition is required. * Instead of calling Time conversion using user defined function, use bult in function in-line with SQL queries. |

**2.5. Job Performance & Utilization**

**2.5.1 Inbound Jobs Performance Analysis**

This section presents a performance breakdown of inbound ETL jobs based on their execution durations across the Pre - land, Today, and Delta processing layers. Jobs have been grouped into the following time-based categories to help identify long-running or performance-sensitive workloads:

* **1 Hour** – Jobs exceeding 60 minutes
* **30 Minutes** – Jobs taking between 30 to 60 minutes
* **15 Minutes** – Jobs executing between 15 to 30 minutes

**🔄 Pre-land Layer Job Summary**

*These jobs ingest data from source files into staging tables with minimal transformation—primarily time zone adjustments and file-to-table mappings.*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Job Name | Duration | > 1 hr | > 30 min  < = 1 hr | > 15 min  < 30 min |
| cdm\_pre\_land\_tnsc\_s02\_plan | 4470 | 1 | 0 | 0 |
| cdm\_preland\_dst\_acb\_weekly | 1979 | 0 | 0 | 1 |
| cdm\_preland\_dst\_customer\_legal\_owner\_account\_position | 3234 | 0 | 1 | 0 |
| cdm\_preland\_dst\_customer\_legal\_owner\_b00\_weekly | 1045 | 0 | 0 | 1 |
| cdm\_preland\_dst\_customer\_legal\_owner\_position | 2145 | 0 | 0 | 1 |
| cdm\_preland\_dst\_customer\_legal\_owner\_y09\_daily | 2578 | 0 | 1 | 0 |
| cdm\_preland\_dst\_acb\_weekly | 1031 | 0 | 0 | 1 |
| cdm\_pm\_acp\_and\_bridgecope\_ocp\_inactivation | 27357 | 1 | 0 | 0 |
| cdm\_tis\_gap\_advisor\_preference\_delta | 1400 | 0 | 0 | 1 |
| cdm\_trading\_dq\_prev\_check\_dq | 4420 | 2 | 5 | 3 |

**📅 Today Layer Job Summary**

*Today layer jobs handle complex business rules and data transformation across multiple tables, contributing to higher execution times, with some jobs exceeding the 1-hour mark.*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Job Name | Duration | | > 1 hr | > 30 min  <= 1 hr | > 15 min  < 30 min |
| cdm\_today\_dst.customer\_legal\_owner\_party\_cntct\_mech | 1164 | 0 | | 0 | 1 |
| cdm\_today.ssc\_al.participant\_party\_address | 1020 | 0 | | 0 | 1 |
| cdm\_today\_dst.customer\_legal\_owner\_address | 1008 | 0 | | 0 | 1 |
| cdm\_today\_dst.customer\_legal\_owner\_party\_role | 1320 | 0 | | 0 | 1 |
| cdm\_today.ssc\_grp.party\_contact\_mechanism | 1418 | 0 | | 0 | 1 |
| cdm\_today.ssc\_grp.party\_rel | 9031 | 1 | | 0 | 0 |
| cdm\_today.ssc\_office\_party\_rel | 1239 | 0 | | 0 | 1 |
| cdm\_today.ssc\_al.participant\_address | 1273 | 0 | | 0 | 1 |
| cdm\_today\_dst.customer\_legal\_owner\_party\_restrictor | 1763 | 1 | | 0 | 8 |

**♻️ Delta Layer Job Summary**

*Delta jobs perform change data capture (CDC) by comparing Today vs Previous snapshots, often involving heavy joins and aggregations—resulting in some long-running executions.*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Job Name | Duration | > 1 hr | > 30 min  <= 1 hr | > 15 min  < 30 min |
| cdm\_ihw\_delta\_elastic\_search | 7744 | 1 | 0 | 0 |
| cdm\_publish.pas.party\_broker\_id\_delta | 1736 | 0 | 0 | 1 |
| cdm\_tis.acp\_advisor\_preference\_delta | 1405 | 1 | 0 | 2 |

**2.5.2 Jobs that did not run after Sep 2024**

Based on total CDM jobs and the list of Jobs that run in last 6 months, we arrived at the list of jobs that were not run in last 6 months. We have categorized the list of infrequent jobs yearly including the list of jobs that never run as per Autosys report.

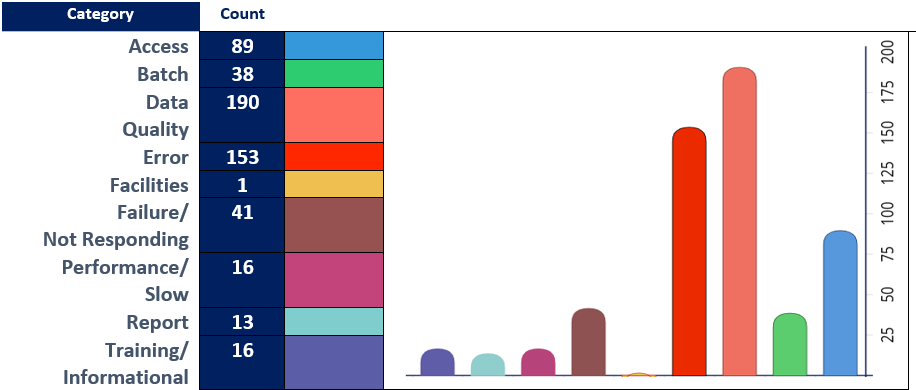
A graph with numbers and a bar

AI-generated content may be incorrect.

**2.6 Operational Stability & Data Flow Optimization**

**2.6.1 Last 1 year Incident Analysis –**

Based on total CDM jobs and the list of Jobs that run in last 6 months, we arrived at the list of jobs that were not run in last 6 months. We have categorized the list of infrequent jobs yearly including the list of jobs that never run as per Autosys report.

****

The above incidents are considered into re-designing certain jobs associated with Data Quality, Performance and Batch.

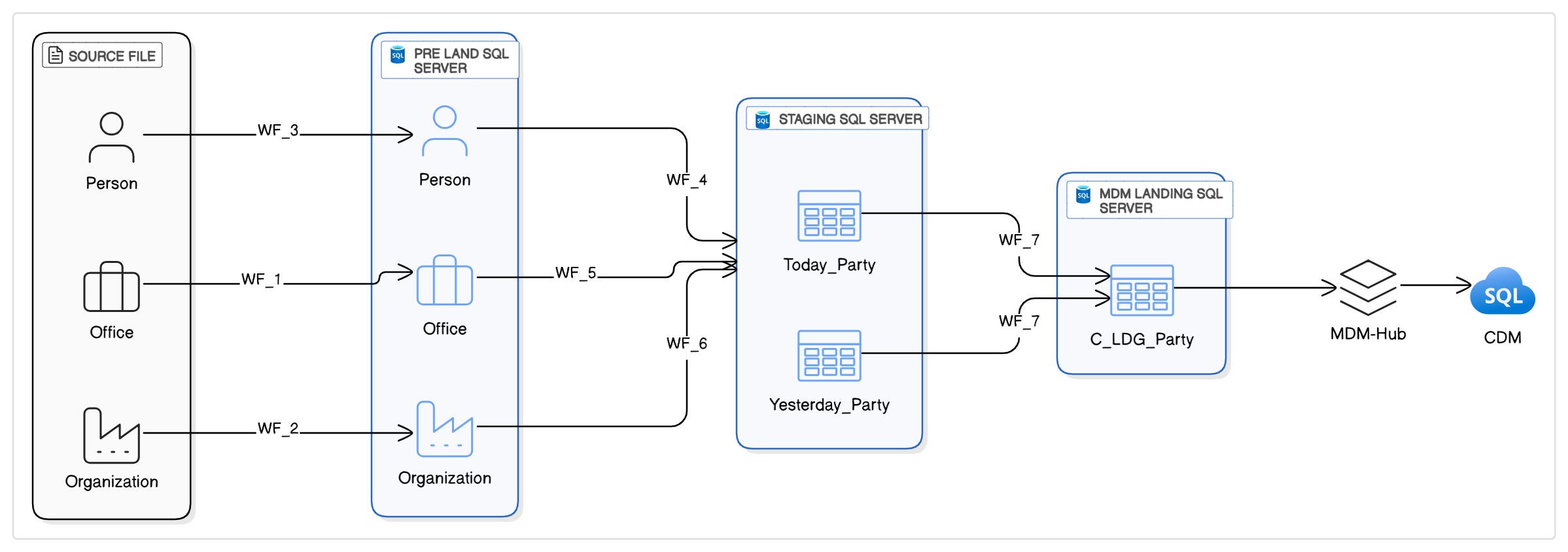
**2.6.2 High Level Inbound Dataflow**

We have analysed the inbound data flow into CDM. There are multiple hops where data is stored and accessed while moving from source to target.

At high level, PARTY entity is combination of Office, Org and Person, where for each subtype there is a table at preland layer as the data is received from source. The different preland tables are converged into Party entity at Today layer and there onwards granularity remains same till MDM load.

High-Level Data Flow Architecture (for Party Entity)

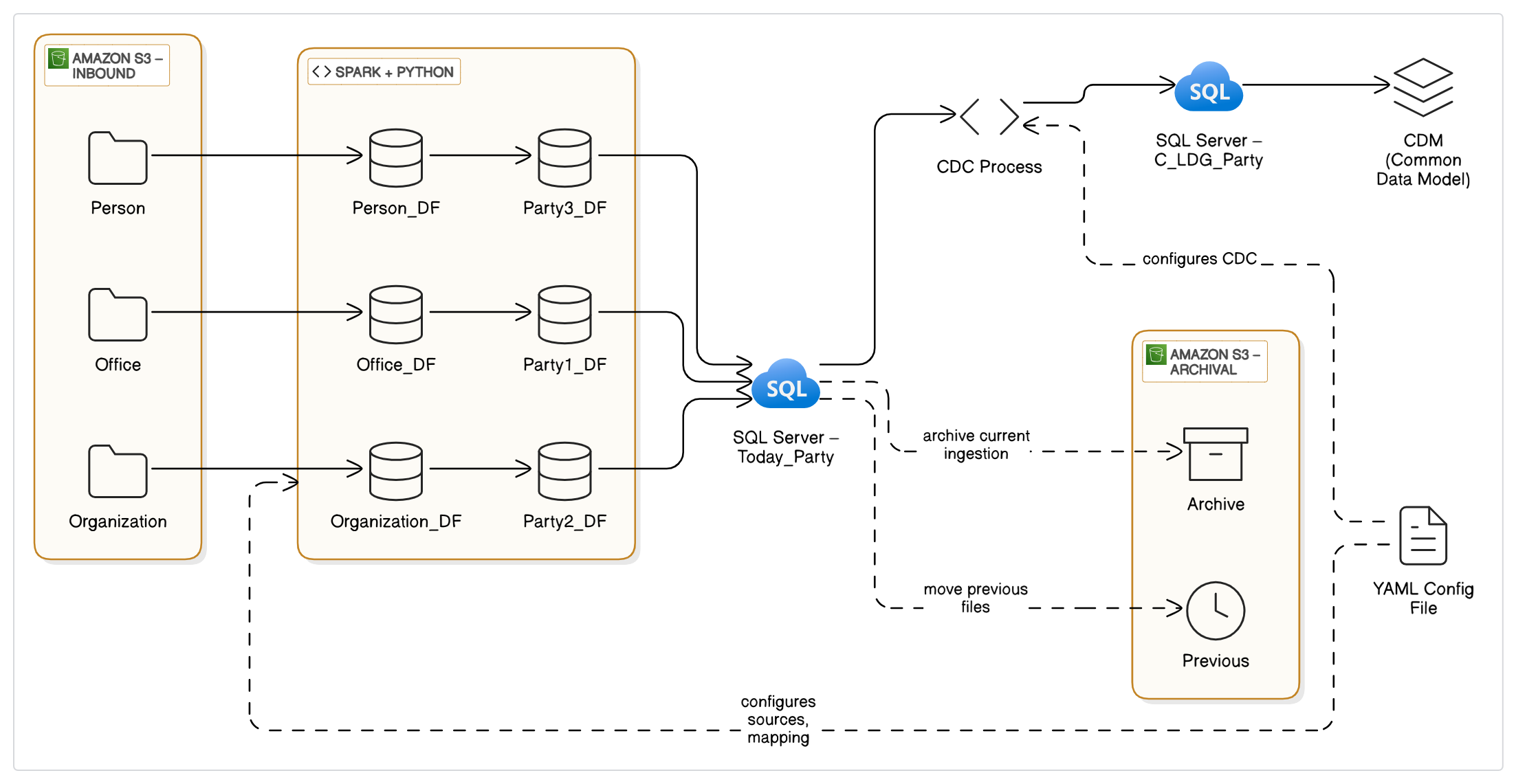
**CURRENT**



In proposed data flow, we are trying to show that different source files can be directly loaded into Today Party table without changing target table structure and removing the additional layer preland. In doing so we could reduce the technical debt of intermediate mappings covering file to preland and preland to today layer load.

High-Level Data Flow Architecture (for Party Entity)

**PROPOSED**



**3. Cloud Modernization Plan**

**☁️**

As part of the broader CDM Modernization initiative, the Cloud Modernization Plan focuses on enabling scalable, cloud-native data integration by transitioning from legacy on-premises ETL infrastructure to modern AWS-based platforms.

The plan consists of two key transformation tracks:

**3.1 Migration of Informatica Workflows to PySpark**

This track involves the automated and semi-automated conversion of ~2,000 existing Informatica PowerCenter workflows into PySpark-based ETL scripts. Using tools like Data Switch, the conversion process ensures the functional parity of business logic while enabling cleaner, more maintainable code that aligns with modern data engineering practices. The migration supports reduced technical debt, improved performance, and easier maintenance across the CDM ecosystem.

**3.2 Deploying the Converted Pipelines to AWS – Cloud Modernization**

Once converted, the PySpark pipelines are deployed on AWS using serverless and scalable data services.

Two primary deployment approaches are under consideration:

* AWS Glue + Apache Airflow: A fully cloud-native orchestration and execution model, recommended for its flexibility, scalability, and cost-efficiency.
* AWS Glue + AutoSys: A transitional hybrid approach leveraging the existing AutoSys scheduler for teams requiring phased adoption.

**4. Evaluation of Informatica Mapping Conversion Methods**

**🛠️**

This section outlines and compares four key strategies for converting Informatica PowerCenter mappings into PySpark: a metadata-driven ingestion framework, a no-code tool-based approach (Data Switch), AI-powered code generation, and a hybrid method combining automation with manual refinement. Each method is evaluated for scalability, automation potential, and suitability for large-scale enterprise data modernization.

**4.1 Approach Matrix: Converting Informatica to PySpark**

Migrating more than 2,000 Informatica workflow mappings to PySpark is a complex but essential step toward modernizing data infrastructure. Several viable methods exist—ranging from manual rewrites to fully automated, no-code tools—each offering different levels of automation, development effort, and technical flexibility. The following summary outlines the most practical and widely adopted approaches for this transformation.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Option Name | Description | Best For | Automation Level (Est.) | Pros | Cons |
| Manual Coding | Fully manual rewrite of each Informatica mapping in PySpark by a developer. | Small volumes, one-off critical jobs | ~0% | Full control, precise logic replication | Not scalable, time-consuming, high cost |
| Custom XML-to-PySpark Parser | Internal tool reads Informatica XML and generates PySpark scripts using business rules. | Orgs with common mapping patterns and in-house engineering | ~50% | Tailored to your needs, reusable | Medium effort to build and maintain |
| DSL-Based Code Generator | Convert Informatica logic into JSON/YAML and use templates to generate PySpark. | Teams with strong metadata discipline and repeatable transformations | ~60% | Decouples logic from code, reusable templates | Requires upfront design of DSL and mappings |
| LLM/AI-Assisted Code Generation | Use LLMs like GPT to convert mappings from XML to PySpark using predefined patterns. | Teams open to AI experimentation and rapid prototyping | ~80-90% | High automation potential, fast output | Needs human QA, may miss complex edge cases |
| No-Code Converter Tool (Data  Switch) | Commercial no-code tool to convert Informatica XML to PySpark with configurable logic. | Enterprises needing speed with moderate customization | ~50-60% (Data  Switch specific) | High automation, minimal coding | usage cost |
| Excel-Based Mapping Templates | Use Excel to document mapping logic and auto-generate PySpark via Python scripts. | Data teams comfortable with low-code interfaces | ~40% | Business-user friendly, simple to implement | Not ideal for complex transformations |
| Internal GUI Mapping Tool | Build a web UI (e.g., Streamlit) where users define mappings, and backend generates PySpark. | Orgs needing user-friendly control over transformation logic | ~70% | Semi-automated, UI-driven | High upfront development effort |
| No-Code ETL Tools with Export | Rebuild workflows in tools like Talend or NiFi that support PySpark export. | Teams already using these tools for new pipelines | ~60% | Quick visual mapping, good for simple jobs | Limited for complex/nested logic |
| Metadata-Driven Code Generation | Use metadata lineage tools to extract logic and auto-generate PySpark code. | Enterprises with mature data governance | ~80% | Centralized control, scalable | High implementation complexity |

**4.2 Metadata driven common ingestion framework**

As part of a modern data transformation strategy, the metadata-driven ingestion framework represents a scalable and strategic approach to simplifying ETL modernization. Rather than rewriting thousands of Informatica mappings into individual PySpark scripts, this method centralizes transformation logic into metadata configurations (YAML, JSON, or relational metadata tables). A reusable PySpark engine interprets this metadata at runtime, dynamically executing data extraction, transformation, and load (ETL) processes. In essence, we replace code with configuration—streamlining delivery while enforcing consistency across the pipeline landscape.

**Best Fit Use Cases**  
This approach is ideal in environments where a significant portion of ETL workloads follow repeatable patterns—such as standardized ingestion from multiple systems or transformations with minimal variations. It particularly excels in enterprise settings with high job volumes and common logic archetypes, enabling rapid onboarding of new data sources by simply adding or modifying configurations.

**Estimated Automation Potential**  
Once the core framework is established, up to 70–85% of the ETL development effort can be automated through configuration. Developers focus on defining metadata—like source/target mappings, transformation rules, and scheduling parameters—rather than writing code. However, building the initial framework demands upfront investment, and complex transformation scenarios may still require tailored PySpark scripts.

**Key Advantages**

* **Reusability at Scale**: A single PySpark engine can execute hundreds of pipelines by simply swapping metadata inputs, dramatically reducing code redundancy and promoting uniformity across ETL layers.
* **Faster Delivery**: New jobs can often be added with minimal coding—speeding up time-to-value, especially in rapidly evolving data environments.
* **Improved Governance and Quality**: Centralizing logic enables built-in enforcement of enterprise data standards, lineage tracking, and validation rules, which are often inconsistent in hand-coded pipelines.
* **Cross-Environment Flexibility**: With execution logic abstracted into metadata, the same framework can support cloud-native platforms (e.g., AWS Glue, EMR) and on-prem Spark clusters with minimal change.

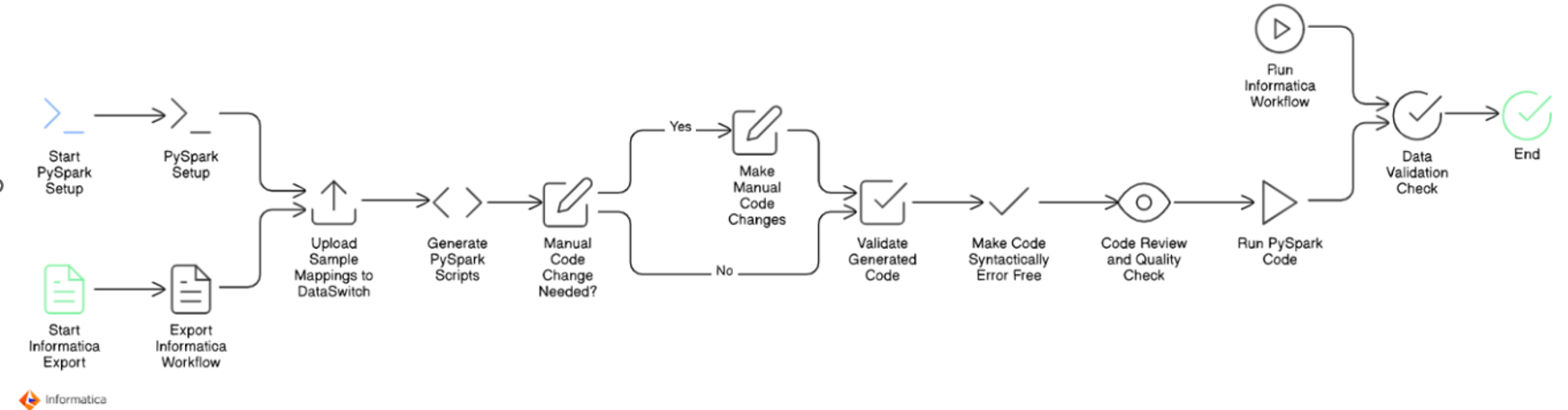
**4.3 Data Switch Tool-Based Conversion**

The Data Switch tool-based approach offers a semi-automated, no-code solution to convert Informatica PowerCenter mappings into PySpark scripts. It is purpose-built to accelerate ETL modernization by translating exported Informatica XML files into runnable PySpark code using a configurable engine. This approach minimizes manual development while maintaining fidelity to the original transformation logic, making it ideal for enterprise environments prioritizing speed, consistency, and lower migration risk.

**Estimated Automation Potential**Automation levels range between 50% and 60%, depending on the complexity of the mapping. For straightforward mappings, the tool can generate production-ready PySpark code with minimal adjustments. For complex workflows involving custom logic, stored procedures, or unusual data patterns, post-generation refinement is typically required.

**Informatica Workflow to PySpark Code Conversion – Step-by-Step Process using Data Switch**

This diagram outlines the end-to-end workflow for converting Informatica PowerCenter mappings (XML exports) into PySpark scripts using a semi-automated approach via Data Switch, a no-code conversion tool. It visualizes how technical teams can efficiently transition legacy ETL workflows to a modern PySpark-based architecture while ensuring validation, quality, and functional equivalence with the original mappings.



**Step-by-Step Explanation**

1. **Start Informatica Export**

* Begin by identifying the Informatica PowerCenter mappings intended for migration.
* Export these mappings in XML format. This serves as the raw input for the conversion process.

2. **Start PySpark Setup**

* In parallel, initiate the setup for the PySpark environment.
* This includes setting up Spark clusters (local or cloud), libraries, configuration files, and dev/test environments where the generated PySpark code will be executed and validated.

3**. Export Informatica Workflow**

* Export the full workflow related to the mappings, including all transformation logic, connections, and dependencies.
* This ensures accurate replication during the conversion to PySpark.

4. **Upload Sample Mappings to Data Switch**

* Load the exported XML files into the Data Switch platform.
* Data Switch analyses the XML files and interprets the mapping logic using its built-in engine.

5. **Generate PySpark Scripts**

* Data Switch generates base PySpark code by automatically translating the Informatica transformation logic into Python scripts compatible with Apache Spark.
* This step significantly reduces manual coding effort.

6. **Manual Code Change Needed?**

* Perform an initial review of the auto-generated code.
* If the code is not fully accurate or misses complex transformation logic, determine whether manual intervention is needed.

7. **Make Manual Code Changes (if required)**

* Developers manually adjust and refine the PySpark scripts where automation falls short.
* This could involve custom joins, filters, lookups, or handling edge cases that are not fully captured by automated tooling.

8. **Validate Generated Code**

* Validate the PySpark code by comparing logic flow and transformation steps against the original Informatica mapping.
* This helps catch any discrepancies early in the migration cycle.

9**. Make Code Syntactically Error-Free**

* Ensure the generated and manually modified code is syntactically correct.
* Fix missing imports, indentation, variable mismatches, or API misuses to make the code runnable.

10**. Code Review and Quality Check**

* Conduct peer reviews to ensure the code meets performance, security, and maintainability standards.
* Apply linting, formatting, and static analysis if applicable.

11. **Run PySpark Code**

* Execute the PySpark job in a controlled test environment.
* Monitor output logs and check if the job runs successfully end-to-end.

12. **Run Informatica Workflow (Parallel Testing)**

* In parallel, run the original Informatica workflow using the same input data.
* This helps in benchmarking and functional validation.

13. **Data Validation Check**

* Compare the outputs from the Informatica job and the PySpark job.
* Perform record-level and field-level comparisons to ensure both jobs produce identical results.

14. **End**

* Once validation passes and functional equivalence is confirmed, the migration is considered complete for that mapping.
* The code is then ready for production deployment or inclusion in the broader migration batch.

**4.4 AI-Driven Generator Tool**

The AI-driven generator tool approach leverages modern Large Language Models (LLMs) and AI capabilities to automate the translation of Informatica XML files into PySpark code. It begins by parsing thousands of workflow XML files and extracting key elements—such as source definitions, transformation logic, SQL overrides, expressions, and target mappings. These extracted components are stored in a structured SQL database, serving as a metadata repository for further processing.

Once stored, LLMs (like GPT-based models) are applied to interpret the logic and generate equivalent PySpark code or integrate with a predefined Spark framework. The AI models are trained or prompted with conversion rules, making them capable of understanding a variety of transformation patterns and intelligently mapping them to Spark constructs.

**Estimated Automation Potential**With a well-structured metadata store and a fine-tuned LLM pipeline, the automation level can exceed 80–90% for standard mapping patterns. Human oversight is still needed for validation, exception handling, and certain advanced transformations, but the majority of code can be generated automatically with consistent quality.

**4.5 Hybrid Approach**

The hybrid approach combines the strengths of AI-driven generation, metadata-based frameworks, and traditional manual coding to achieve a balanced and scalable solution for ETL modernization. This method begins with the AI generator tool extracting and converting Informatica XML mappings into base-level PySpark scripts or configuration metadata. These outputs are then either injected into a common metadata-driven ingestion framework—which executes logic dynamically at runtime—or refined manually for complex use cases.

By using AI for fast initial conversion, metadata frameworks for standardization, and manual intervention for exceptions, this approach maximizes automation without compromising on flexibility or quality.

**Best Fit Use Cases**Best suited for enterprise programs with 1,000–2,000+ workflows of varying complexity. It handles a wide range of scenarios—from repetitive load patterns that fit into a metadata model, to custom transformations that demand manual tuning. This method works well in environments where no single strategy fully covers all use cases.

**Estimated Automation Potential**Depending on the workflow complexity distribution, automation can range from 70% to 90%. AI and metadata frameworks handle bulk conversion, while manual intervention is focused only on truly complex, non-standard mappings. This allows teams to prioritize effort and accelerate delivery without sacrificing accuracy.

🔍

**Approach Evaluation for Migrating Informatica Mappings**

As organizations modernize their data ecosystems, migrating legacy Informatica PowerCenter mappings to scalable, cloud-native architectures becomes a strategic priority. Choosing the right approach for this migration is critical to balancing speed, cost, flexibility, and long-term maintainability.

This section evaluates multiple approaches based on key factors such as automation level, development effort, ability to handle complex transformations, and alignment with enterprise architecture goals.

**1. Comparative Analysis: Hybrid Approach vs Data Switch**

When evaluating strategies for modernizing legacy Informatica workflows into scalable PySpark pipelines, two distinct approaches emerge: the Data Switch Approach and the Architectural Hybrid Conversion Framework. Each offers unique strengths and trade-offs depending on the organization’s goals, timeline, and complexity of ETL processes.

|  |  |
| --- | --- |
| Data switch Approach | AI-Powered Hybrid Migration Framework – Pros |
| ✅ **Pros:**   * Faster initial migration due to pre-built tool * Minimal development effort * Lower upfront cost * Less need for specialized skills (Spark/AWS) * Mostly regression testing only   ❌ **Cons:**   * No code reuse – each XML results in a separate PySpark script * Lacks standardization – inconsistent code structure * Limited flexibility – hard to adapt to custom use cases * Weak support for complex transformations * Minimal optimization opportunities * High post-migration adjustment effort * Dependent on external tool * Not suitable for re-architecting * Not aligned with long-term data strategy | ✅ Pros:  * **AI-assisted code generation** accelerates development for common ETL patterns * **Modular and reusable architecture** improves long-term maintainability * **Standardized coding practices** aligned with enterprise guidelines * **Fully customizable** to Capital Group’s business and data needs * **Handles complex transformations** with better accuracy and control * **Optimized workflows** through intelligent logic tuning and resource efficiency * **No dependency on third-party conversion tools** * **Supports re-architecting MDM workflows** (inbound and outbound) * **Low post-migration effort** due to tailored design and testing * **Strategic alignment** with long-term data modernization goals   ❌ **Cons:**   * **Longer initial development timeline** compared to tool-based conversion * **Higher upfront investment** due to AI integration and engineering effort * **Requires experienced resources** in AI, PySpark, AWS, and ETL design * **Extensive testing scope** across data quality, performance, and business rules * **Change management effort** due to architectural and operational shifts |

**2. Comparative Analysis: Metadata-Driven Framework vs Data Switch**

In the journey to modernize legacy Informatica workflows, organizations must choose between flexible engineering-led solutions and rapid tool-driven conversions. This analysis compares the Metadata-Driven Framework with the Data Switch Approach in terms of scalability, maintainability, flexibility, and alignment with strategic enterprise data goals.

|  |  |
| --- | --- |
| Data switch Approach | Metadata-Driven Framework |
| ✅ **Pros:**   * Faster initial migration due to pre-built tool * Minimal development effort * Lower upfront cost * Less need for specialized skills (Spark/AWS) * Mostly regression testing only   ❌ **Cons:**   * No code reuse – each XML results in a separate PySpark script * Lacks standardization – inconsistent code structure * Limited flexibility – hard to adapt to custom use cases * Weak support for complex transformations * Minimal optimization opportunities * High post-migration adjustment effort * Dependent on external tool * Not suitable for re-architecting * Not aligned with long-term data strategy | ✅ **Pros:**   * Highly reusable: One engine handles multiple ingestion scenarios * Centralized configuration for source/target definitions and transformations * Standardized architecture leads to consistent code quality * Easily extensible for new sources, targets, or business rules * Lower maintenance due to minimal code duplication * Scales well across domains and teams * Aligns with enterprise DevOps and CI/CD standards   ❌ **Cons:**   * Requires initial investment in building the metadata engine * Demands a strong understanding of dynamic PySpark design * May involve a steeper learning curve for developers |

**3. Comparative Analysis: AI-Driven Generator Tool vs Data Switch**

Modernizing legacy Informatica workflows to PySpark involves choosing between static tool-based conversion and intelligent automation. This analysis compares the AI-Driven Generator Tool approach with the Data Switch Approach, focusing on automation intelligence, transformation accuracy, and future readiness.

|  |  |
| --- | --- |
| Data switch Approach | AI-Driven Generator Tool |
| ✅ **Pros:**   * Faster initial migration due to pre-built tool * Minimal development effort * Lower upfront cost * Less need for specialized skills (Spark/AWS) * Mostly regression testing only   ❌ **Cons:**   * No code reuse – each XML results in a separate PySpark script * Lacks standardization – inconsistent code structure * Limited flexibility – hard to adapt to custom use cases * Weak support for complex transformations * Minimal optimization opportunities * High post-migration adjustment effort * Dependent on external tool * Not suitable for re-architecting * Not aligned with long-term data strategy | ✅ Pros:  * Smarter code generation using NLP and LLM pattern recognition * Interprets complex mappings better than rule-based tools * Potential for code optimization and deduplication * Can adapt based on learned patterns across workflows * Faster evolution with feedback loops and model tuning * Reduces manual validation and rework * Future-aligned with intelligent ETL and low-code/no-code paradigms   ❌ **Cons:**   * **Black-box nature** may cause trust or transparency issues * **Needs model monitoring and governance** * **Requires significant upfront integration with metadata/catalog systems** * **Higher compute/resource overhead compared to static tools** |

**Approach and Tool Selection for Converting Informatica Mappings:**

*We have chosen* ***Data Switch****, a no-code automated ETL conversion platform, to facilitate the mapping conversion.* This strategic business decision aims to accelerate the migration by automatically converting PowerCenter mappings to PySpark, reducing manual coding effort and project risk.

**Current State Analysis**

📦

The current data integration environment is built on legacy Informatica PowerCenter workflows running on on-premises servers. This section provides a high-level overview of the existing ETL architecture, its operational patterns, and key considerations as we plan a migration to a modern, cloud-native platform.

**1. Job Volume: High Workflow Density**

The existing ETL ecosystem comprises approximately 2,000 Informatica PowerCenter workflows, each defined using XML configurations. These workflows represent a wide variety of data processing needs—ranging from straightforward source-to-target loads to complex transformations and lookup-heavy jobs with embedded business logic.

Many workflows are modular and designed around source-system groupings or specific business domains. Over time, incremental additions and evolving business requirements have led to the growth of the workflow repository, creating challenges in cataloguing, impact analysis, and dependency management.

**Key Observations:**

* Volume indicates a large-scale transformation effort, which will require effective planning, automated conversion tooling, and rigorous testing frameworks.
* A significant percentage of workflows likely contain redundant or obsolete logic. A pre-migration rationalization phase is recommended to identify reusable patterns, remove deprecated jobs, and standardize before converting to a cloud-native format.

**2. Data Volume: Moderate-High Throughput Per Job**

Each workflow processes an average of 500,000 to 1 million records, which includes transformations, joins, lookups, and inserts/updates to downstream systems. Some workflows may exceed this threshold during peak processing cycles or when loading from staging environments.

**Performance and data volume considerations:**

* Current throughput is manageable by cloud-native platforms (e.g., Azure Data Factory, Spark-based pipelines, or Glue), but design choices must ensure that processing time and resource allocation scale linearly with job volume.
* Certain workflows might use pushdown optimization or custom SQL overrides, bypassing Informatica’s transformation engine—these must be carefully identified during the migration to ensure semantic parity.

**3. Delta Load Strategy: Incremental Processing via Separate Workflows**

Incremental or "delta" loads are managed through separate workflows, which execute based on timestamp filters, change flags, or audit columns. These delta jobs are not integrated within full-load processes but exist as independent entities with distinct scheduling logic.

**Strategic Implications:**

* While effective, this approach adds orchestration overhead and introduces dependencies between full and incremental loads.
* Any future-state platform must support incremental logic natively—whether through:
  + Change Data Capture (CDC) techniques
  + Built-in incremental copy features
  + Timestamp-based filtering logic within the data integration engine

**During migration, a decision must be made on whether to:**

* Consolidate delta and full loads into single parameterized pipelines, or
* Maintain their separation but simplify orchestration using modern

**4. Execution Frequency: Non-Uniform Load Distribution**

Job execution patterns are not standardized across the ecosystem. The majority of workflows run on-demand or based on non-daily schedules. Daily runs account for roughly 300–400 workflows, but peak periods—particularly during financial closings, reconciliations, or data refresh windows—can spike to 1,200 job executions per day.

**Challenges:**

* High job concurrency during peaks imposes strain on the existing on-prem infrastructure and will need cloud elasticity for smooth operation in the future state.
* Schedulers must account for non-daily, time-sensitive batch jobs, and dependencies that span across days or weeks.
* Some jobs are triggered based on data arrival or completion of upstream workflows, suggesting the presence of event-driven orchestration logic that must be replicated or modernized in the cloud.

**5. Database Environment: Azure-Hosted**

The ETL processes primarily interact with Microsoft SQL Server databases, used as both source and target systems. These databases are already hosted on Microsoft Azure, which represents an architectural advantage by reducing network latency and enabling secure, high-throughput data transfers within the cloud.

**Additional Details:**

* Connection strings, firewall rules, authentication models (likely managed identity or connection pools), and credentials are already in place and maintained in AutoSys job definitions or external parameter files.
* Several workflows may rely on linked servers, stored procedures, or inline SQL transformations that directly access SQL Server artifacts.

**Cloud Readiness Implication:**

* Since the data tier is already in Azure, the modernization effort can focus primarily on transforming compute and orchestration layers, rather than data migration.

**5. Current Infrastructure Setup: On-Prem & Custom-Managed**

The entire ETL ecosystem is tightly coupled with legacy infrastructure and customized operational controls:

**Job Scheduling – AutoSys**

* AutoSys is used to trigger and schedule Informatica workflows, manage job dependencies, enforce SLA-based timing, and initiate notification logic.
* Jobs may include conditional branching, file arrival checks, or multi-job dependencies, making the scheduling logic complex and interdependent.

**Workflow Execution – On-Premises Informatica Servers**

* All jobs are processed through on-prem Informatica PowerCenter servers, with fixed compute capacity.
* Scaling to meet peak loads has historically required careful scheduling and resource tuning, given the finite infrastructure footprint.

**Monitoring, Logging & Notifications – Shell Script-Driven**

* Custom shell scripts manage post-execution logging, error detection, metric reporting, and email/SMS notifications.
* Log files are parsed manually or via grep-based pattern matching to generate alerts, and are written to local or NFS-mounted volumes.
* Job metadata (such as start time, end time, row counts, status codes) is extracted and recorded in internal reporting tables or Excel dashboards.

This decentralized, script-based observability is not scalable, lacks central monitoring, and introduces challenges for root-cause analysis or compliance auditing.

**Cloud Deployment Approaches Considered**

**🌩️**

**Cloud Deployment Approaches Considered**

To modernize our existing Informatica workflows by deploying PySpark pipelines on AWS, we carefully evaluated several cloud-based deployment strategies. Each approach was assessed based on criteria like scalability, maintainability, cost-effectiveness, and compatibility with our current infrastructure.

Below is a detailed overview of the evaluated options and our preliminary recommendations:

**❌ Option 1: Databricks (Rejected)**

**Description:** Databricks provides a unified, managed platform for Spark workloads with extensive collaboration and analytics capabilities.

**Pros:**

* Optimized performance and built-in analytics.
* Strong integration with Spark ecosystem and notebooks.
* Data switch pyspark code fully compatible here
* Provides granular control over Spark clusters and compute resources.
* Inbuild schedule system or integrate with third party schedular
* **Databricks ecosystem** is a unified platform designed for **data, analytics, and AI** and data Lakehouse

**Cons:**

* High licensing and operational costs.
* Our workloads (typically processing only 5–7 rows per job across 2,000 jobs) do not justify the expense.

**Conclusion:** **Rejected** because the cost outweighs the benefits for our specific scenario.

Between **Databricks** and **Informatica**, typically **Informatica's enterprise licensing costs are** lesser compared to Databricks, particularly for large-scale, enterprise-grade solutions.

**Databricks**:

* Pay-as-you-go pricing model based on actual compute usage.
* Addition cost of Cloud Infrastructure Expenses (Amazon S3, Azure Blob Storage), networking, data transfer
* Flexible scaling, with costs varying by cloud provider, instance type, and features.
* Additional Licensing Databricks SQL, Delta Lake optimization, advanced security features
* Typically, more cost-effective for moderate workloads or variable usage.
* Administrative and Operational Overhead

**Informatica**:

* Subscription-based, tiered pricing that escalates quickly with increased data volume, users, or advanced features (e.g., MDM, data governance).
* Often higher fixed licensing costs at enterprise scale.

**Conclusion**:  
Informatica generally incurs higher costs at enterprise scale, especially as data volumes and complexity increase, making Databricks potentially more economical depending on usage patterns and needs.

**✅ Option 2: AWS Glue + Apache Airflow (Recommended)**

**Description:** This approach leverages AWS Glue for serverless, scalable ETL execution paired with Apache Airflow for flexible, robust workflow orchestration.

**Pros:**

* Fully managed serverless ETL service, reducing operational overhead.
* Airflow provides advanced scheduling, monitoring, error handling, and retry mechanisms.
* Cost-effective, cloud-native architecture aligned with AWS best practices.

**Cons:**

* Requires setup and maintenance of Airflow (can be managed via AWS MWAA or self-hosted).
* Data switch pyspark code need convert or make compatible as per AWS Glue

**Conclusion:** **Recommended** as the best practice solution due to superior scalability, cost-effectiveness, and robust orchestration capabilities.

**⚙️ Option 3: EMR + AWS Step Functions (Considered)**

**Description:** Amazon EMR is utilized to run Spark workloads, orchestrated through AWS Step Functions, which offer managed workflows.

**Pros:**

* Provides granular control over Spark clusters and compute resources.
* Good fit for large-scale, compute-intensive jobs with specialized tuning.
* Dataswitch pyspark code more compatible here

**Cons:**

* Higher complexity in managing cluster resources.
* Setup and ongoing operational effort are substantial compared to serverless solutions.

**Conclusion:** Technically viable but **less attractive** due to higher operational complexity for our typical workloads.

**✅ Option 4: AWS Glue + AutoSys (Feasible)**

**Description:** Utilize existing AutoSys scheduler to orchestrate jobs while leveraging AWS Glue for execution of PySpark scripts.

**Pros:**

* Minimal changes required in job orchestration logic.
* Enables a gradual transition by maintaining familiarity with AutoSys scheduling.
* Immediate compatibility with current scheduling processes and existing infrastructure.

**Cons:**

* Continues dependence on on-premise legacy scheduling solutions.
* Limited integration with AWS-native monitoring, governance, and management tools.
* Potentially creates long-term complexity in maintaining a hybrid orchestration environment.
* Dataswitch pyspark code need convert or make compatible as per AWS Glue

**Conclusion:** **Feasible as an interim solution** to ease transition, but less ideal from a cloud-native perspective.

**❌ Option 5: EC2 with Spark Cluster (Rejected)**

**Description:** Manually managing Spark clusters on Amazon EC2 infrastructure.

**Pros:**

* Maximum flexibility and control over cluster configuration and Spark tuning.
* Dataswitch pyspark code fully compatible here

**Cons:**

* High operational burden and infrastructure management overhead.
* Lacks scalability, elasticity, and operational simplicity compared to managed serverless offerings.

**Conclusion:** **Rejected** due to operational complexity, maintenance overhead, and limited scalability advantages.

NOTE:  
The following diagrams not defined detailed definitions for:

* Network: Connectivity, firewall, VPC settings.
* Security: IAM roles, encryption, compliance.
* Error Handling: Job retries and monitoring.

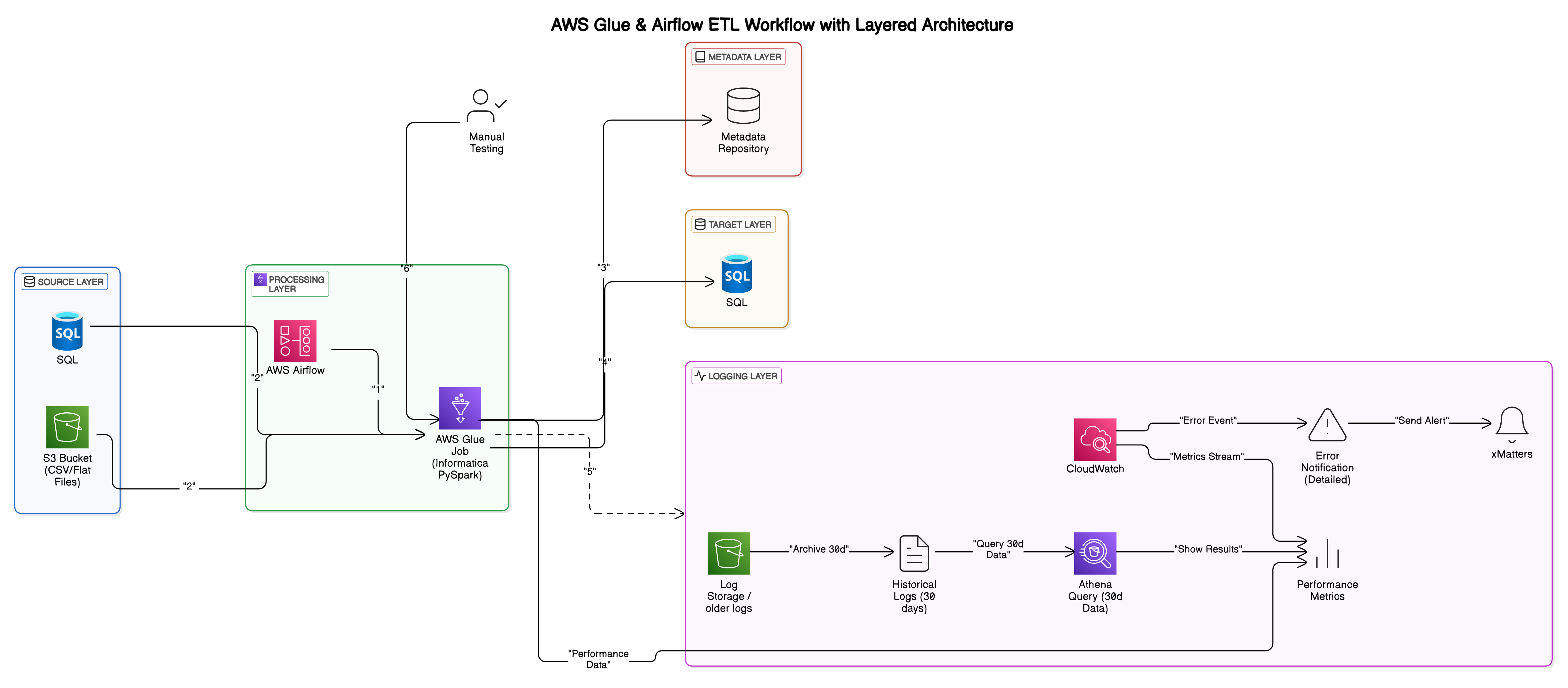
**Detailed Architecture Discussion**

Given the analysis above, we plan to focus further discussions specifically on:

* **Option 2 / Approach 1: AWS Glue + Apache Airflow** *(Recommended, best practice)*
* **Option 4/ Approach 2: AWS Glue + AutoSys** *(Feasible, transitional approach)*

We will provide detailed **architectural diagrams**, cost estimations, technical considerations, and implementation guidelines for both options. These discussions will help us make a well-informed final decision tailored to our short-term migration goals and long-term cloud modernization strategy.

**AWS Glue + Apache Airflow** *(Recommended, best practice)*



**AWS Glue & Airflow ETL Workflow – Layered Architecture Overview**

This architecture replaces legacy Informatica ETL workflows with a scalable AWS-based solution using AWS Glue and Apache Airflow. It is divided into clear, purpose-specific layers:

**1. Source Layer**

* **SQL Database (Azure-hosted)**: Structured data source.
* **Amazon S3**: Storage for flat-file inputs (CSV).

**2. Processing Layer (AWS Glue + Apache Airflow)**

* **AWS Airflow**:
  + Manages scheduling, dependencies, and workflow orchestration.
  + Sends alerts on workflow failures.
* **AWS Glue (PySpark)**:
  + Executes data transformation and loads processed data.
  + Serverless, auto-scalable ETL jobs.

**3. Target Layer**

* **MS SQL Database (Azure-hosted or on frame )**: Receives processed data for analytics.

**4. Metadata Layer**

* Tracks job executions, dependencies, and data lineage.
* Supports operational monitoring and troubleshooting.

**5. Logging Layer**

* **AWS CloudWatch**: Real-time monitoring and alerts.
* **Amazon S3**: Archival of historical logs.

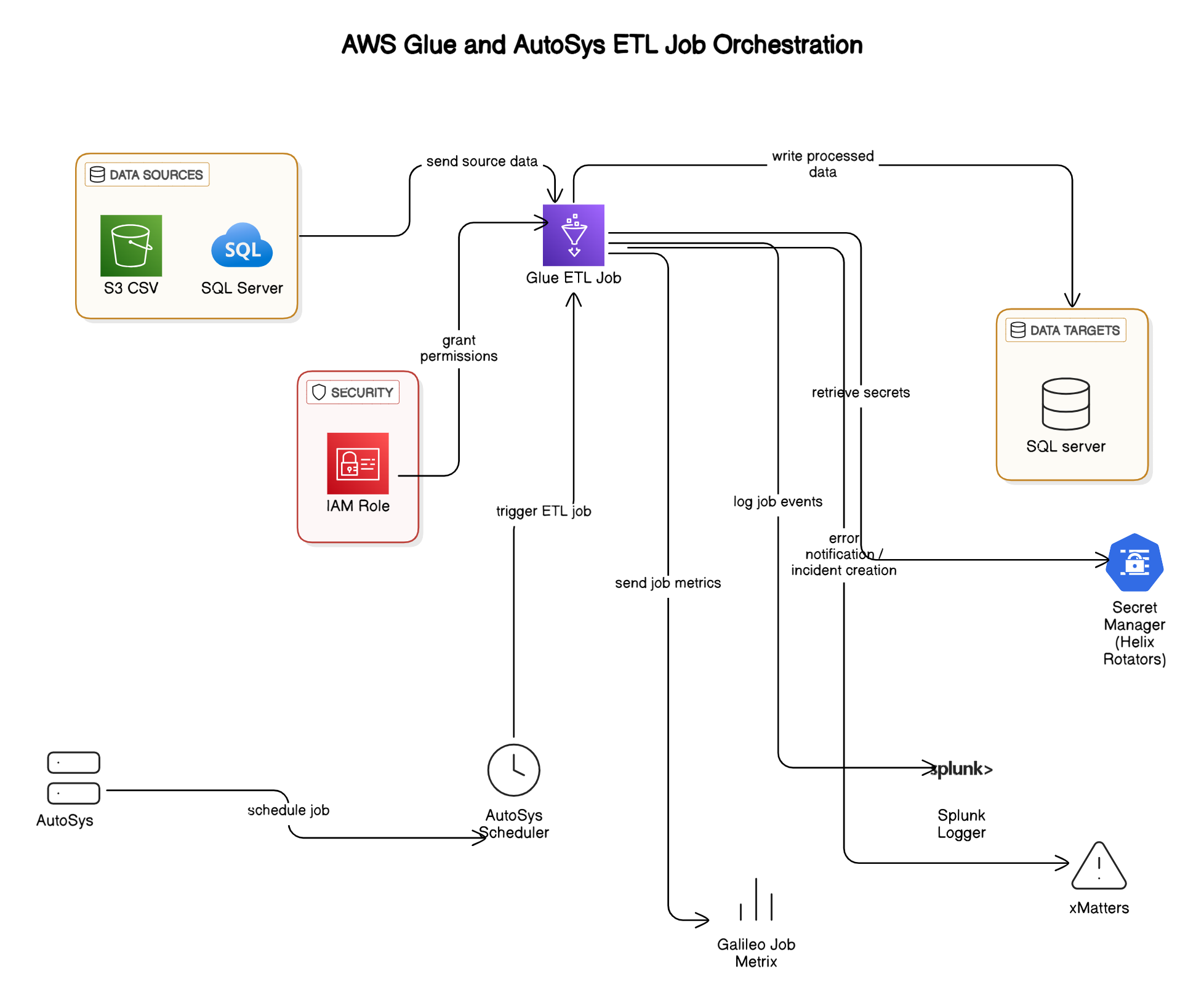
**6. Manual Testing**

* Periodic manual validation for quality assurance.

**Typical Workflow**

1. Airflow initiates Glue jobs.
2. Glue extracts data from sources.
3. Glue transforms and loads data to the target database.
4. Job metadata is recorded.
5. Logs captured in CloudWatch, archived to S3.
6. Manual quality checks performed as needed.

**AWS Glue + AutoSys** *(transitional approach)*



**AWS Glue and AutoSys ETL Job Orchestration**

This architecture illustrates how converted Informatica workflows (now PySpark ETL jobs) can be orchestrated using AWS Glue for processing, while leveraging the existing on-premises AutoSys scheduler for job scheduling. It represents a hybrid architecture effectively bridging cloud-native and on-premises systems.

**Detailed Architecture Overview**

The architecture consists of the following major components:

**1. Data Sources**

* **SQL Server (On-Premise)**:
  + Provides structured data inputs for the ETL process.
* **Amazon S3 (CSV files)**:
  + Serves as a storage source for flat-file data, accessible directly by AWS Glue.

**2. AWS Glue ETL Job**

* Centralized ETL processing using PySpark scripts converted from original Informatica workflows.
* Retrieves data from sources, performs transformations, enrichments, and loads processed data to the target.
* Leverages AWS Glue’s serverless infrastructure for scalability and cost efficiency.

**3. Data Targets**

* **SQL Server Database**:
  + Acts as the final destination for the transformed and cleansed data.
  + Supports analytical queries, reporting, or downstream business applications.

**4. Security Layer**

* **AWS IAM Roles**:
  + Securely grant AWS Glue necessary permissions to access resources (S3, SQL Server, Secrets Manager).
  + Ensure fine-grained access control and secure execution of ETL jobs.
* **AWS Secrets Manager (Helix Rotators)**:
  + Provides secure and managed storage of sensitive credentials required by Glue ETL jobs to access source and target systems.

**5. Job Scheduling & Orchestration**

* **AutoSys Scheduler (On-Prem)**:
  + Existing on-premises scheduler used to trigger AWS Glue jobs.
  + Continues to schedule ETL workloads, ensuring minimal disruption to existing job orchestration practices.
  + Allows a phased migration approach by maintaining familiar job scheduling processes.

**6. Logging and Monitoring**

* **Splunk Logger**:
  + Captures and logs job events, providing centralized logging and monitoring capability.
  + Facilitates troubleshooting, auditing, and compliance reporting.
* **Galileo Job Metrics**:
  + Collects and analyzes job execution metrics, such as performance and operational efficiency data.

**Typical Workflow Execution**

1. **AutoSys (On-Prem)** schedules and triggers the AWS Glue job.
2. **AWS Glue ETL Job** begins execution, authorized via an **IAM role** for secure access.
3. Glue retrieves source data from **S3** and **SQL Server** databases.
4. Glue retrieves credentials securely from **AWS Secrets Manager**.
5. Glue performs transformations and writes processed data to the target **SQL Server**.
6. Glue sends execution logs and metrics to **Splunk Logger** and **Galileo** for monitoring.

**Open Questions for AWS Glue and AutoSys ETL**

Below are critical questions that need clarification and discussion as we proceed with the Informatica-to-PySpark ETL migration using the above architecture:

* **AutoSys Details:** Should we set up a new AutoSys cluster in the cloud, or continue using the existing on-premises setup?
* **Splunk API and Job Metrics:** If a new AutoSys cluster is created, do we need to set up new Splunk APIs and job metrics, or can we reuse the existing configurations?