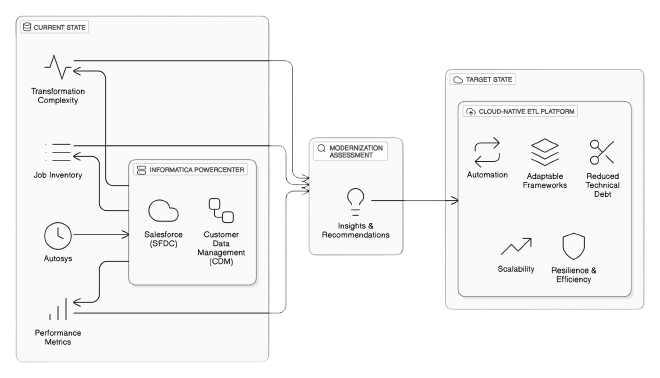
A blue and black logo

AI-generated content may be incorrect.



**ETL ASSESSMENT REPORT**

This report provides a comprehensive evaluation of Capital Group’s existing ETL ecosystem, currently implemented using Informatica PowerCenter and orchestrated with Autosys. The assessment focuses on analyzing current workflows to identify opportunities for modernization



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

**Section Overview**

[**Executive Foundations**](#MC_Executive_Foundations)

1. [Executive Summary](#S1_1)

2. [Scope & Objectives](#S1_2)

3. [Assessment Methodology](#S1_3)

[**CDM ETL Current State & Workload Analysis**](#MC2)

1. [Workflow Inventory Overview](#S2_1)

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

2. [Functional Categorization](#S2_2)

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

3. [Transformation Complexity](#S2_3)

* Mapping Complexity Description

4. [ETL Layer Job Analysis](#S2_4)

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

5. [Job Performance & Utilization](#S2_5)

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

6. [Operational Stability](#S2_6)

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

[**Cloud Modernization Plan**](#MC3)

1. [Migration of Informatica Workflows to Py Spark](#S3_1)

2. [Deploying the Converted Pipelines to AWS – Cloud Modernization](#S3_2)

[**Evaluation of Informatica Mapping Conversion Methods**](#MC4)

1. Metadata-Driven Common Ingestion Framework

2. Data Switch Tool-Based Conversion

3. AI-Driven Generator Tool

4. Hybrid Approach

[**Current State Analysis**](#MC6)

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**](#MC7)

1. Cloud Deployment Approaches Considered

2. Detailed Architecture Discussion

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

|  |
| --- |
|  |

**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 * Nagesh |

🧾

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.

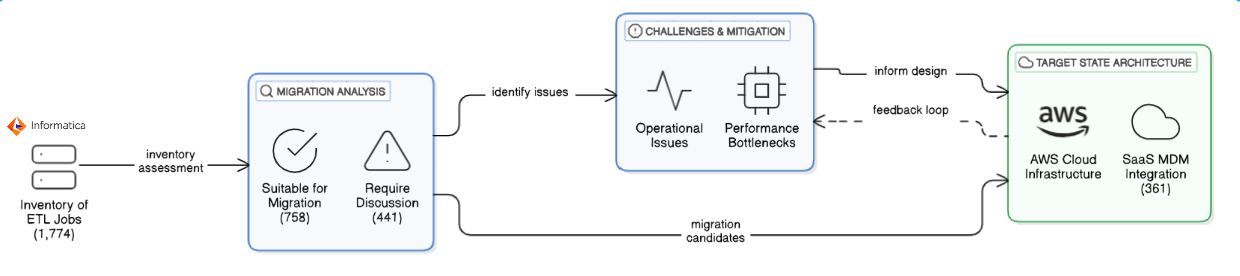
## 1.2 Scope & Objective

**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 examining the workflows and mappings

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

**Informatica repository for SFDC**

Repository – pc105\_repo\_dev2\_3

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

|  |  |
| --- | --- |
| Source | Folder |
| **SFDC** | 📁 SFDC\_AF |
|  | 📁 SFDC\_CDM\_INBOUND |
|  | 📁 SFDC\_CDM\_PUBLISH |
|  | 📁 SFDC\_CONCUR |
|  | 📁 SFDC\_COVERAGE |
|  | 📁 SFDC\_EACG\_INBOUND |
|  | 📁 SFDC\_INVESTMENT |
|  | 📁 SFDC\_IPBIOS |
|  | 📁 SFDC\_MAIN |
|  | 📁 SFDC\_MSSBI |
|  | 📁 SFDC\_NADIA |
|  | 📁 SFDC\_PUBLISH |
|  | 📁 SFDC\_SFD\_INBOUND |

**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.

[**7. Analysis of Long Running Jobs and Jobs that did not run for longer time. - MSS CLIENT DATA MASTER - Confluence**](https://confluence.capgroup.com/pages/viewpage.action?pageId=1081144656)

Confluence Page

**1.3.4 Informatica Workflows**

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

multiple layers like pre-land, 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 Autosys Jobs Categorization and Migration Path Strategy**

The Autosys workload has been comprehensively analysed to identify migration-eligible components and determine the most suitable modernization approach per job type. The strategy distinguishes between jobs to be excluded and those selected for migration, followed by a structured classification of the migration paths based on the functional role of each job.

**1. Jobs Excluded from Migration**

A set of Autosys jobs were excluded from the modernization initiative due to one or more of the following criteria:

* Jobs classified under **MOCA**.
* Jobs that have **not run in the last six months**, as identified from execution logs.

These jobs are designated as *“Not to Convert”* and excluded from the migration scope.

**2. Jobs Identified for Migration**

From the remaining Autosys inventory, a refined set of jobs was shortlisted for migration. These were categorized into five distinct functional groups based on their operational behavior:

|  |  |  |
| --- | --- | --- |
| **Job Category** | **Job Count** | **Migration Approach** |
| **ETL Jobs**  **(convertible via DataSwitch or manual)** | **804** | DataSwitch Conversion |
| **File checks, Notifications, IDQ Status** | **124** | Manual Python Scripts on AWS |
| **Core and Additional MDM Jobs** | **408** | Migrated to MDM SaaS |
| **Reports and Publish Jobs** | **187** | Rewritten using MDM SaaS API or PySpark |
| **Autosys Commands, Shell Scripts** | **93** | Translated to AWS-native commands or executed directly using AutoSys on AWS |

Each category is assigned a tailored modernization strategy aligned with technical feasibility, business criticality, and system dependencies.

**3. Migration Path Mapping**

The table below summarizes the selected migration paths and the technologies/platforms aligned with each:

|  |  |
| --- | --- |
| **Migration Type** | **Execution Target** |
| **DataSwitch Conversion** | AWS Glue / PySpark via automated ETL conversion |
| **Manual Python Script** | AWS Lambda or AWS Batch |
| **MDM SaaS** | Cloud-native SaaS-based Master Data Management solution |
| **Rewritten Jobs (API / PySpark)** | AWS Glue using PySpark with API integration |
| **Shell/Command Jobs** | AutoSys migrated to cloud or rehosted as AWS-native commands |

This migration strategy ensures optimal reuse of existing patterns (e.g., DataSwitch conversions), minimizes redevelopment effort, and aligns all migrated jobs with AWS-based deployment models, enabling improved scalability, automation, and operational consistency.

A diagram of a software flow

AI-generated content may be incorrect.

A screenshot of a table

AI-generated content may be incorrect.

**2.1.3 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.

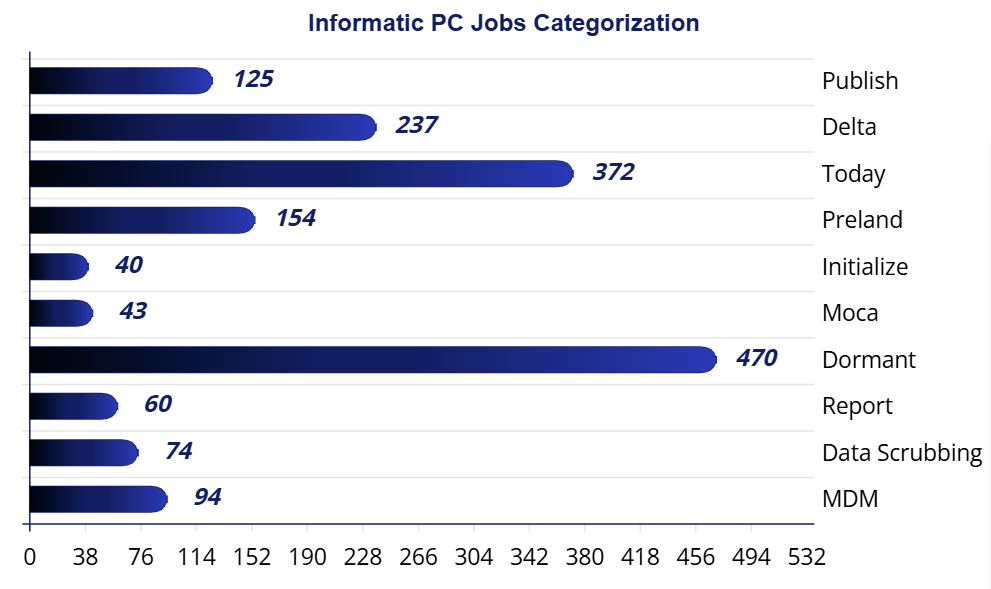
**2.2. Functional Categorization**

**2.2.1 Categorization of based on functional and business entities**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| # | Source | Advisor | EACG | Intermediary | Investor | IR Trading ID | Plan | Plan/Plan Sponsor | Business Entity Not Identified | Total |
| 1 | Brightscope |  |  |  |  |  |  | 28 | 42 | 89 |
| 2 | Broadridge |  |  |  |  |  |  |  | 3 | 3 |
| 3 | CDM |  |  |  |  |  |  |  |  | 461 |
| 4 | DMI | 62 |  |  |  |  |  |  |  | 62 |
| 5 | DORIS | 28 |  |  |  |  |  |  |  | 28 |
| 6 | DST |  |  |  |  |  |  |  |  | 19 |
| 7 | EI |  |  |  |  |  |  | 21 | 36 | 57 |
| 8 | Empower |  |  |  |  |  |  | 15 | 18 | 33 |
| 9 | Eventsource |  |  |  |  |  |  |  | 5 | 5 |
| 10 | FC |  | 27 |  |  |  |  |  |  | 27 |
| 11 | Fidelity |  |  |  |  |  |  |  |  | 2 |
| 13 | Fund\_Famis |  |  |  |  |  |  |  |  | 1 |
| 14 | GCP |  |  |  |  |  |  |  |  | 6 |
| 15 | HNW |  |  |  |  |  |  |  | 14 | 14 |
| 16 | IPS | 80 |  |  |  |  |  |  |  | 80 |
| 17 | MMD |  |  |  |  |  |  |  | 23 | 23 |
| 18 | MultiEmployer |  |  |  |  |  |  |  | 2 | 2 |
| 19 | Newport |  |  |  |  |  |  |  | 8 | 8 |
| 20 | Event/Publish |  |  |  |  |  |  |  | 223 | 223 |
| 21 | Picsar |  |  |  |  |  |  |  | 9 | 9 |
| 22 | PO |  |  |  |  |  |  |  | 5 | 5 |
| 23 | REM |  |  |  |  |  |  |  | 5 | 5 |
| 24 | RIA |  |  |  |  |  |  |  | 1 | 1 |
| 25 | RPA |  |  |  |  |  |  | 58 |  | 58 |
| 26 | RPH |  |  |  |  |  |  |  | 21 | 21 |
| 27 | RPM |  |  |  |  |  |  |  | 6 | 6 |
| 28 | RPO |  |  |  |  |  | 3 |  |  | 3 |
| 29 | Salesconnect |  |  | 46 |  |  |  |  |  | 46 |
| 30 | SC |  |  | 25 |  |  |  |  |  | 25 |
| 31 | Schwab |  |  |  |  |  |  |  | 5 | 5 |
| 32 | SFDC | 81 |  |  |  |  |  | 12 |  | 93 |
| 33 | SFDCLEAD | 12 |  |  |  |  |  |  |  | 12 |
| 34 | SOFI |  |  |  |  |  |  |  | 6 | 6 |
| 35 | SSC |  |  | 25 | 5 |  |  |  |  | 30 |
| 36 | TA2000 |  |  |  |  |  | 7 | 50 | 62 | 62 |
| 37 | Trac |  |  |  |  |  | 27 |  | 64 | 91 |

**2.2.2 Categorization of Power Centre 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** falls 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 * Pre-land * 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**

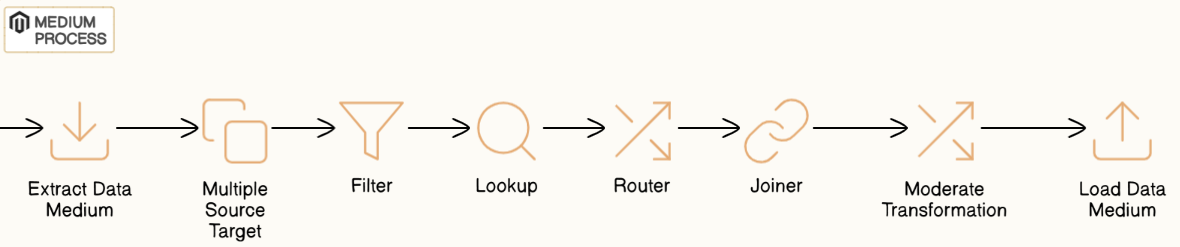
Measuring the complexity of a mapping can be approached by evaluating several factors related to transformations. Based on assessment, here are some key aspects that have been considered.

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. |

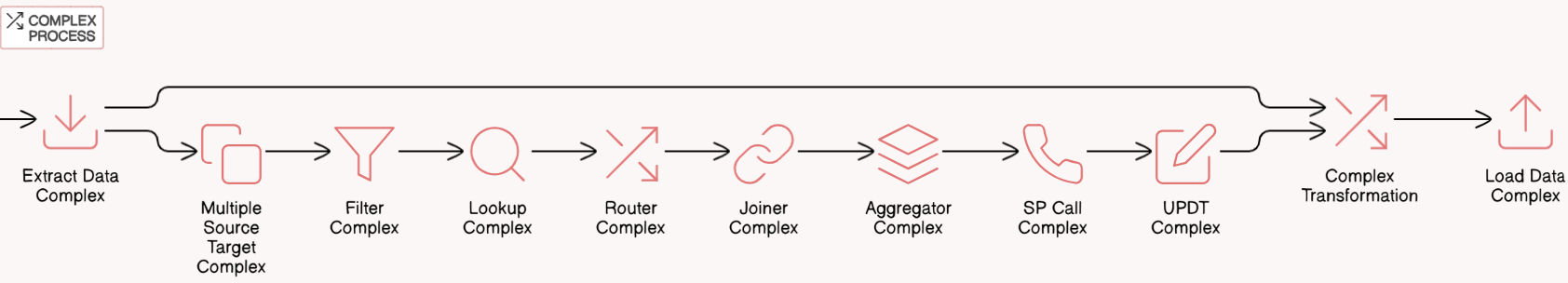
**🟠 Medium**

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



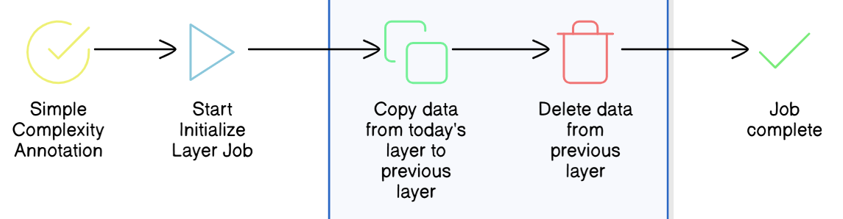
**🔴 Complex**

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 41 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

|  |  |
| --- | --- |
|  | wf\_load\_delta\_prev\_tables |
| Previous Layer | wf\_Load\_Prev\_Plan\_Tables |
|  | wf\_insert\_plan\_contact\_batch\_cycle\_date |

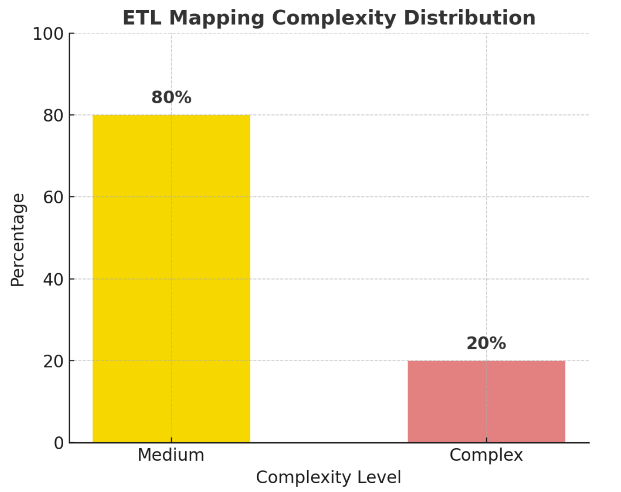
**Complexity – 10**0% of the total jobs in this category falling under **simple**.

**👁️ Observation –**

* Copying the data from one table to another.
* Set the batch cycle and control table date.
* 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 154 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. TA2000 & TRAC

[[3. Analyze File to Pre-Land Data Ingestion - MSS CLIENT DATA MASTER - Confluence](https://confluence.capgroup.com/display/CDM/3.+Analyze+File+to+Pre-Land+Data+Ingestion)](https://confluence.capgroup.com/pages/viewpage.action?pageId=1081144656)

Confluence Link

|  |
| --- |
| **💡 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 372 jobs populating the data from pre-land tables into today layer tables. Multiple joins within pre-land tables are done while generating the dataset. All business rules and logic are 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.

**Examples**

|  |  |
| --- | --- |
| Complex job | wf\_landing\_sales\_connect\_to\_cdm\_firm\_cntct\_mchsm |
| Simple | wf\_landing\_FC\_contact\_party\_address |
| Medium | wf\_today\_trac\_Sponsco\_party |

[4. Analyze Today Ingestion - MSS CLIENT DATA MASTER - Confluence](https://confluence.capgroup.com/display/CDM/4.+Analyze+Today+Ingestion)

Confluence Link -

**👁️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 237 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 |

[7. Analysis of Long Running Jobs and Jobs that did not run for longer time. - MSS CLIENT DATA MASTER - Confluence](https://confluence.capgroup.com/pages/viewpage.action?pageId=1081144656)

Confluence link -

**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.

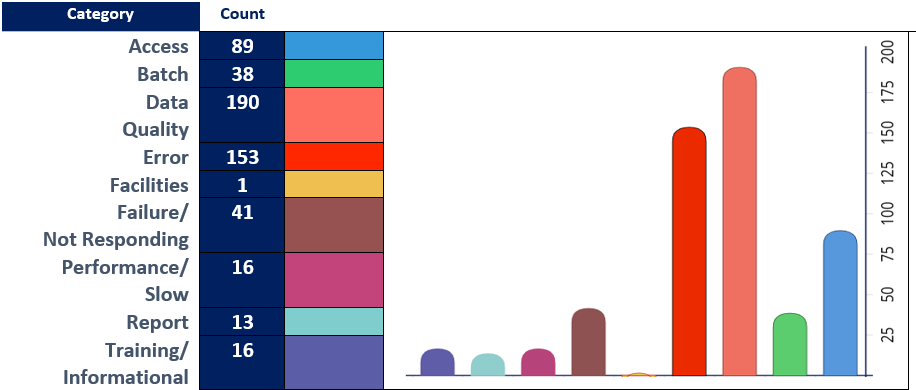
A graph with numbers and a bar

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.

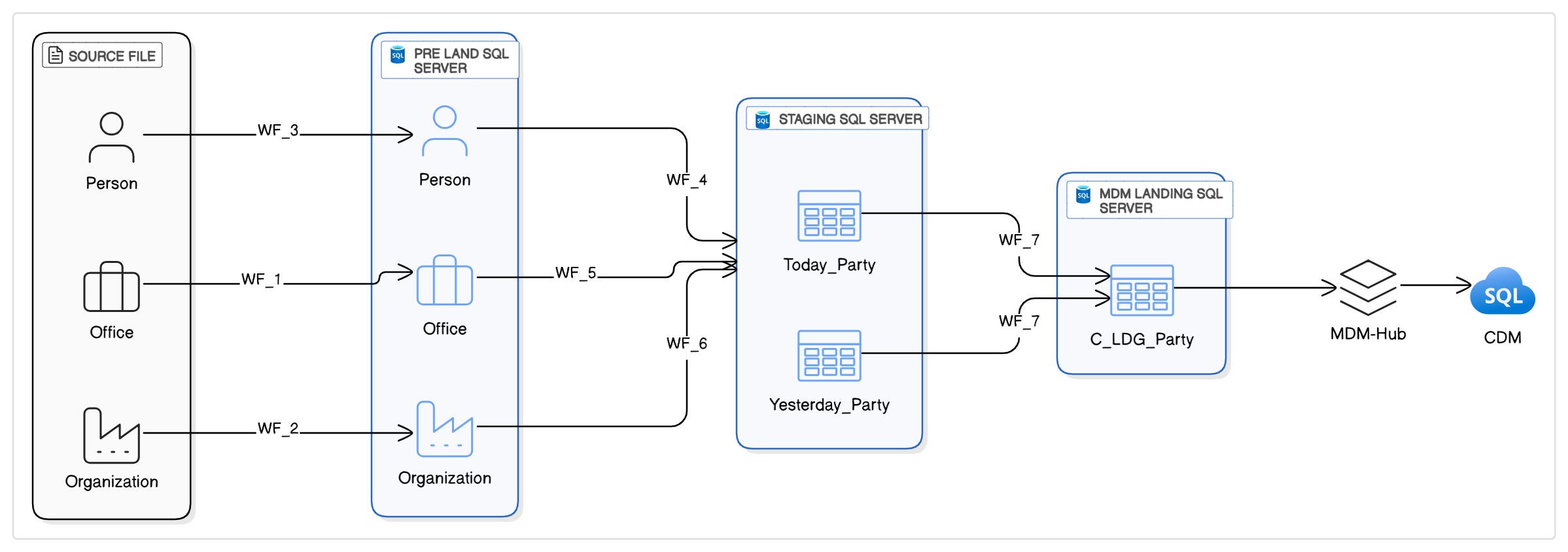
**2.6.2 High Level Inbound Dataflow**

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**



A group of rectangular boxes with text

AI-generated content may be incorrect.

**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 two strategic approaches for converting existing Informatica PowerCenter workflows into PySpark as part of the broader ETL modernization initiative at Capital Group. These approaches are designed to balance automation, reusability, and scalability, and are selected based on the complexity and repeatability of each workflow.

* **Section 4.1 – One-to-One (1:1) Conversion:**Targets unique or complex Informatica mappings by generating a dedicated PySpark script for each workflow. This method ensures fidelity to original business logic and is suited for workflows that require specific handling or contain intricate transformation rules.
* **Section 4.2 – Many-to-One (n:1) Framework-Based Conversion:**  
  Focuses on consolidating multiple similar Informatica workflows into a single, reusable, metadata-driven PySpark framework. This approach leverages pattern recognition, configuration abstraction (via YAML/JSON), and modular design to streamline development and promote maintainability across standardized ETL layers (e.g., Initialize, Preland, Today, Delta).

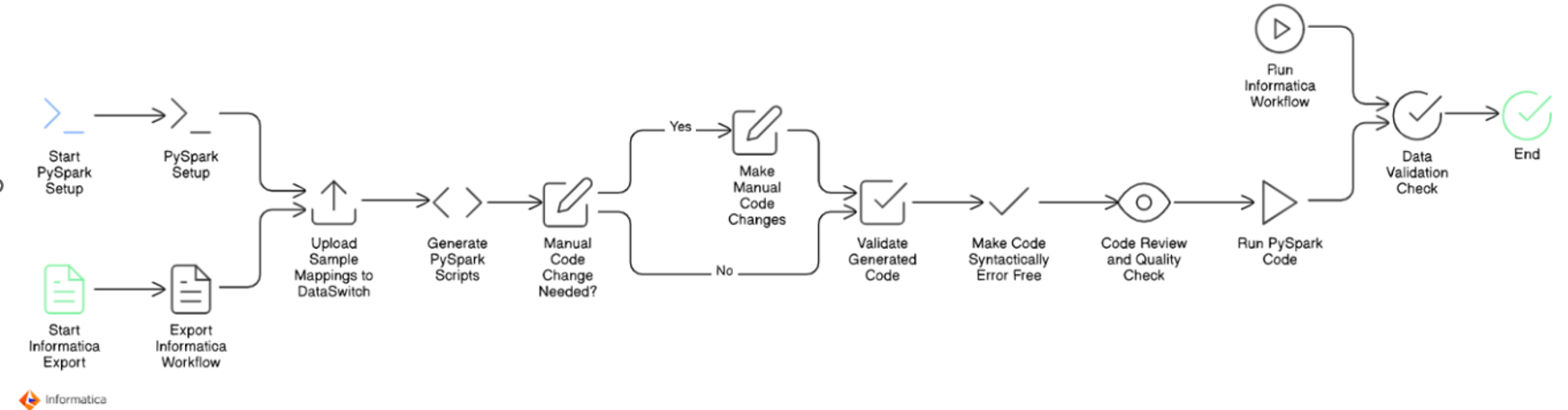
**4.1 Approach Matrix: One-to-One (1:1) Conversion**

This approach involves converting each individual Informatica XML mapping into a dedicated, standalone PySpark job. It is most effective when dealing with:

* Unique logic that doesn’t lend itself to parameterization
* High-complexity workflows with embedded business rules

**4.1.1 One-to-One (1:1) Conversion: Step-by-Step Process**

This step-by-step process outlines how individual Informatica mappings are transformed into standalone PySpark scripts using automation tools (e.g., Data Switch) and manual refinement. This method is particularly valuable for mappings with unique logic or minimal potential for standardization.



|  |  |  |
| --- | --- | --- |
|  | Step | Description |
| Step 1 | Identify & Export Informatica Mappings | Select mappings for migration and export them as XML files. These are the raw input for conversion. |
| Step 2 | Set Up PySpark Development Environment | Set up Spark clusters (on-prem/cloud), install necessary libraries, configure dev/test environments. |
| Step 3 | Export Associated Informatica Workflows | Export the full workflow, including source/target metadata, sessions, dependencies, and transformation flows. |
| Step 4 | Upload Mappings into Tool of Choice | Load the exported XML files into either Data Switch, a custom-built parser, or an AI-based code generator. |
| Step 5 | Auto-Generate Base PySpark Scripts | Use automation tools to convert XML logic into base PySpark code. This typically covers standard logic. |
| Step 6 | Assess the Need for Manual Intervention | Review the generated code for missing logic, complexity gaps, or inaccuracies. Flag where manual coding is needed. |
| Step 7 | Apply Manual Enhancements (If required) | Modify and refine the generated code to match the original mapping logic precisely.  Tasks include:   * Custom transformation logic implementation * Complex join conditions * Date or format conversions * Lookup replication using broadcast joins or cache tables * Parameter replacement and reusable functions |
| Step 8 | Logic Validation Against Original Mapping | Cross-check the **PySpark logic flow** against the original Informatica workflow:   * Transformation sequence * Join paths and conditions * Filter criteria * Data derivations   This ensures **functional parity** before testing. |
| Step 9 | Code Cleanup & Syntax Validation | Ensure the code is syntactically correct and ready for execution:   * Fix imports, indentation, type errors * Verify variable scoping and naming conventions * Remove unused functions or placeholder code   Optionally, apply formatting tools (e.g., black, isort) for consistent styling. |
| Step 10 | Peer Review & Quality Assurance | Perform a structured **peer review** of the script:   * Confirm adherence to internal coding standards * Check for performance bottlenecks (e.g., skewed joins, non-partitioned reads) |
| Step 11 | Execute PySpark Job in Test Environment | Run the PySpark job with test or sanitized data in the target environment.  **Monitor**:   * Execution time and log messages * Task distribution and resource utilization * Output file/database location |
| Step 12 | Execute Original Informatica Workflow (Parallel Run) | In parallel, run the original Informatica workflow with the **same source data**.  This helps create a **baseline output** for comparison and validation. |
| Step 13 | Perform Data Validation | Conduct **record-level** and **field-level comparisons** between PySpark and Informatica outputs:   * Row counts * Primary key match rate * Column value-level differences (nulls, formats, calculated fields) |

**4.1.1 Data Switch (No-Code Tool)**

A commercial no-code platform designed to automate ~50–60% of the Informatica-to-PySpark translation process.

**Pros:**

* Quick initial conversion of standard mappings
* Reduces manual development effort
* User-friendly and configuration-driven

**Cons:**

* Generates flat, non-modular PySpark code
* Requires significant manual effort for optimization and Glue compatibility
* Lacks built-in support for parameterized or reusable logic patterns

**4.1.2 Custom Tool Development**

A proprietary tool that parses Informatica XML files and generates PySpark scripts according to Capital Group’s framework and coding standards.

**Pros:**

* Fully customizable to internal coding and documentation standards
* Enables enforcement of naming conventions, logging, and error handling
* Supports integration with CI/CD pipelines

**Cons:**

* Moderate-to-high initial development effort
* Requires ongoing maintenance for enhancements and bug fixes
* Limited reusability unless designed for pattern recognition

**4.1.3 LLM/AI-Based Code Generator**

Uses Large Language Models (e.g., GPT) to interpret transformation logic in XML and generate corresponding PySpark code automatically.

**Pros:**

* High automation potential (~80–90% for standard patterns)
* Excellent for rapid prototyping and high-volume mapping generation
* Can adapt to different transformation styles with sufficient prompt engineering

**Cons:**

* Requires thorough human validation for edge cases and business-critical logic
* May lack interpretability or debugging support
* Limited contextual memory in complex or chained workflows

**4.2 Many-to-One (n:1) Framework-Based Conversion**

As part of a modern data transformation strategy, the AI-powered code generation framework offers a scalable and intelligent solution for ETL modernization. Instead of converting each Informatica XML mapping into a separate PySpark script, this approach consolidates multiple mappings into a **unified, reusable metadata-driven architecture**.

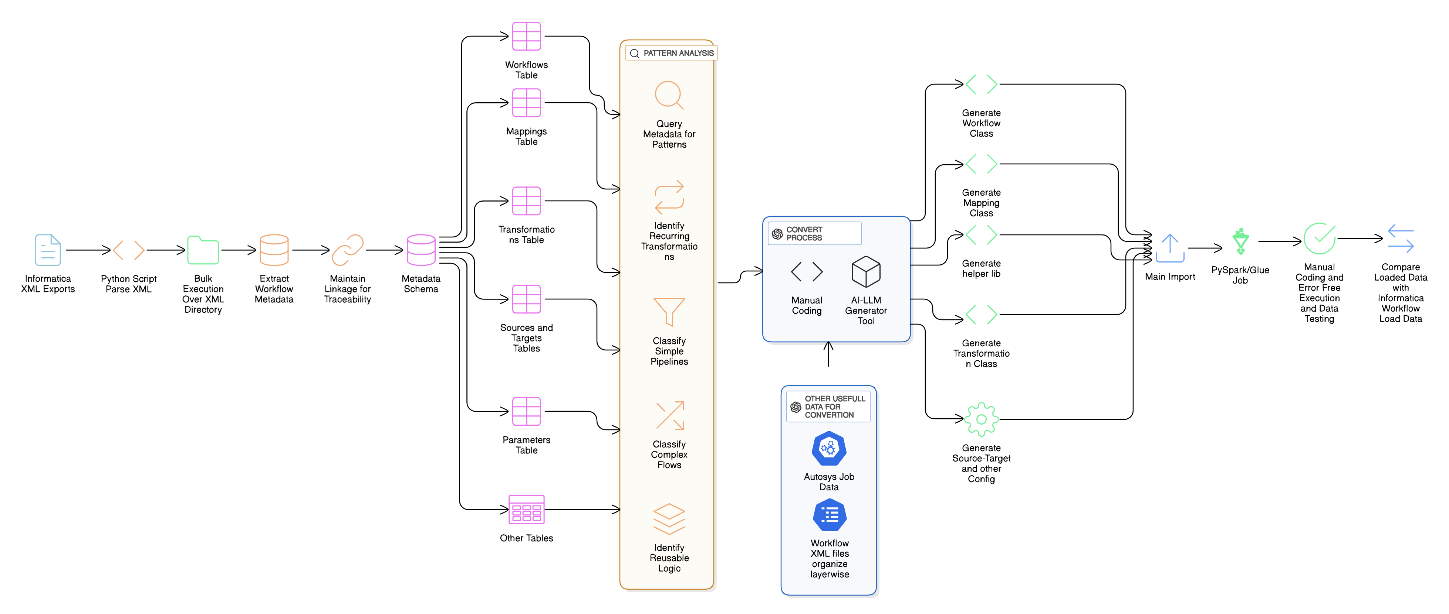
Transformation logic is extracted from Informatica XMLs and structured into metadata configurations (YAML, JSON, or relational metadata tables), organized by established data layers—**Initialize, Pre-land, Today, Delta, and Publish**. A reusable PySpark or AWS Glue engine then interprets this metadata at runtime to dynamically perform extraction, transformation, and load operations.

In essence, this method transforms traditional hard-coded ETL into a **configuration-driven system**, enabling faster delivery, improved maintainability, and greater consistency across the data pipeline ecosystem.

**AI-Driven Informatica-to-PySpark/Glue framework Conversion Tool**

The **AI-Driven Conversion Tool** is a purpose-built, intelligent framework designed to automate the modernization of legacy Informatica PowerCenter workflows. It transforms complex XML-based mappings into clean, maintainable, and reusable **PySpark or AWS Glue jobs** by leveraging AI/LLM models, pattern recognition, and metadata-driven architecture.

This tool significantly reduces manual coding effort, enforces consistency across layers, and accelerates the ETL modernization lifecycle.

****

**4.2.1 AI-Driven Informatica-to-PySpark/Glue framework Conversion Tool : Key Features**

**Step 1: Metadata Extraction & Ingestion**

Objective: Parse and store Informatica XML metadata.

* Collect Informatica XML exports.
* Use a Python script to bulk-parse workflows, mappings, sources, targets, sessions, transformations, and parameters.
* Store extracted metadata in a relational database schema, maintaining full lineage and traceability.

**Step 2: Metadata Schema & Table Design**

Tables Created:

* Workflows Table
* Mappings Table
* Transformations Table
* Sources & Targets Table
* Parameters Table
* Other Tables (e.g., sessions, variables, configs)

This structure forms the metadata backbone for downstream analysis and AI processing.

**Step 3: Layer-Wise Workflow Classification**

Objective: Identify the ETL layer for each workflow based on source/target patterns.

Pattern Rules:

* SharePoint → msscdm\_\* → Preland
* msscdm\_\* → msscdm\_\* → Initialize / Today
* msscdm\_\* → cmx\_ors\_\* → Delta
* cmx\_ors\_\* → SharePoint → Publish

Outcome: Enables clean separation of logic and correct placement into the target architecture.

**Step 4: Autosys Job Integration**

Objective: Map job orchestration logic.

* Parse Autosys JIL files to extract jobs and box configurations.
* Use naming and command-line patterns to associate each job with the corresponding Informatica workflow.
* Store mappings:
  + Autosys Job → Workflow Name → Layer → Box

This supports dependency mapping and orchestration during cloud migration.

**Step 5: Pattern Analysis & Workflow Categorization**

Purpose: Detect transformation patterns and classify workflows.

* Identify:
  + Repetitive transformation patterns
  + Join/filter logic
  + Reusable logic blocks
* Categorize workflows as:
  + Simple Pipelines
  + Complex Flows
  + Reusable Logic Candidates

This step helps prioritize automation versus manual attention.

**Step 6: AI-Driven Conversion Engine**

Objective: Use AI + templating to generate code from metadata.

* Feed parsed XML + metadata to an AI/LLM Generator Tool.
* Generate modular classes like WorkflowClass, MappingClass etc
* Store logic in YAML or JSON configs.

Manual override is supported for complex edge cases.

**Step 7: Code Generation & Assembly**

**Output**: Reusable PySpark or AWS Glue Job

* Generate and assemble:
  + Job import’s structure
  + Workflow logic
  + Helper utilities
* Final job file composed of centralized logic for each pipeline.

This supports many-to-one transformation and maximizes code reuse.

**Step 8: CDC Handling for Delta Layer**

**Redesign Objectives:**

* Ensure presence of last\_updated\_timestamp or similar fields.
* If missing, recommend:
  + Adding audit columns
  + Creating control tables for surrogate CDC
* Implement SCD Type 1 / Type 2 logic within the Glue jobs.

CDC compatibility ensures accurate change capture and scalability.

**Step 9: Autosys Execution Compatibility Analysis**

**Goal**: Ensure converted jobs can be orchestrated reliably.

* Analyze Autosys JILs for:
  + Multi-instance support
  + Job dependencies
  + Retry and alert logic
* Design new invoker scripts using:
  + AWS Glue triggers
  + AWS Step Functions
  + PySpark job schedulers

**Step 10: Testing & Validation**

**Quality Assurance:**

* Run generated PySpark/Glue jobs with test data.
* Validate output against original Informatica load results.
* Perform:
  + Manual review of errors
  + Runtime validation
  + Row-level and schema-level data comparison

**✅Pro:**

1. **Centralized Logic (Many:1):**  
   Combines multiple similar workflows into one optimized PySpark/Glue framework, reducing duplication and simplifying maintenance.
2. **Layer-Aware Automation:**  
   Logic is mapped intelligently to Initialize, Preland, Today, Delta, and Publish layers, preserving the current architecture.
3. **Reusable Components:**  
   Common patterns (joins, lookups, filters) are abstracted into helper classes, improving code reuse and modularity.
4. **AI-Driven Code Generation:**  
   Leverages AI to parse Informatica XML, extract transformation logic, and auto-generate workflow, mapping, and transformation classes.
5. **Future-Proof:**  
   Designed to support changes like timestamp-based CDC or schema evolution, making it adaptable for modern cloud ETL.
6. **Operational Integration:**  
   Integrates with Autosys job definitions and provides provisions for multi-instance parallel execution or job chaining logic.

**❌Con:**

1. **Complexity of Initial Setup:**Building a fully functional AI-driven parser and generator requires significant upfront investment in logic parsing and training/feedback loops.
2. **Dependency on Metadata Accuracy:**  
   Relies heavily on well-defined and consistent metadata in the XML files. Inconsistencies can break or confuse the conversion logic.
3. **Autosys Adaptation Required:**  
   Existing job orchestration (like retries, alerts, dependencies) may need rewriting or adaptation for compatibility with AWS Step Functions or Glue triggers.

**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.

**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**

**🌩️**

**1. Cloud Deployment Approaches**

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. |

**2. Detailed Architecture Discussion**

**Detailed Architecture Discussion**

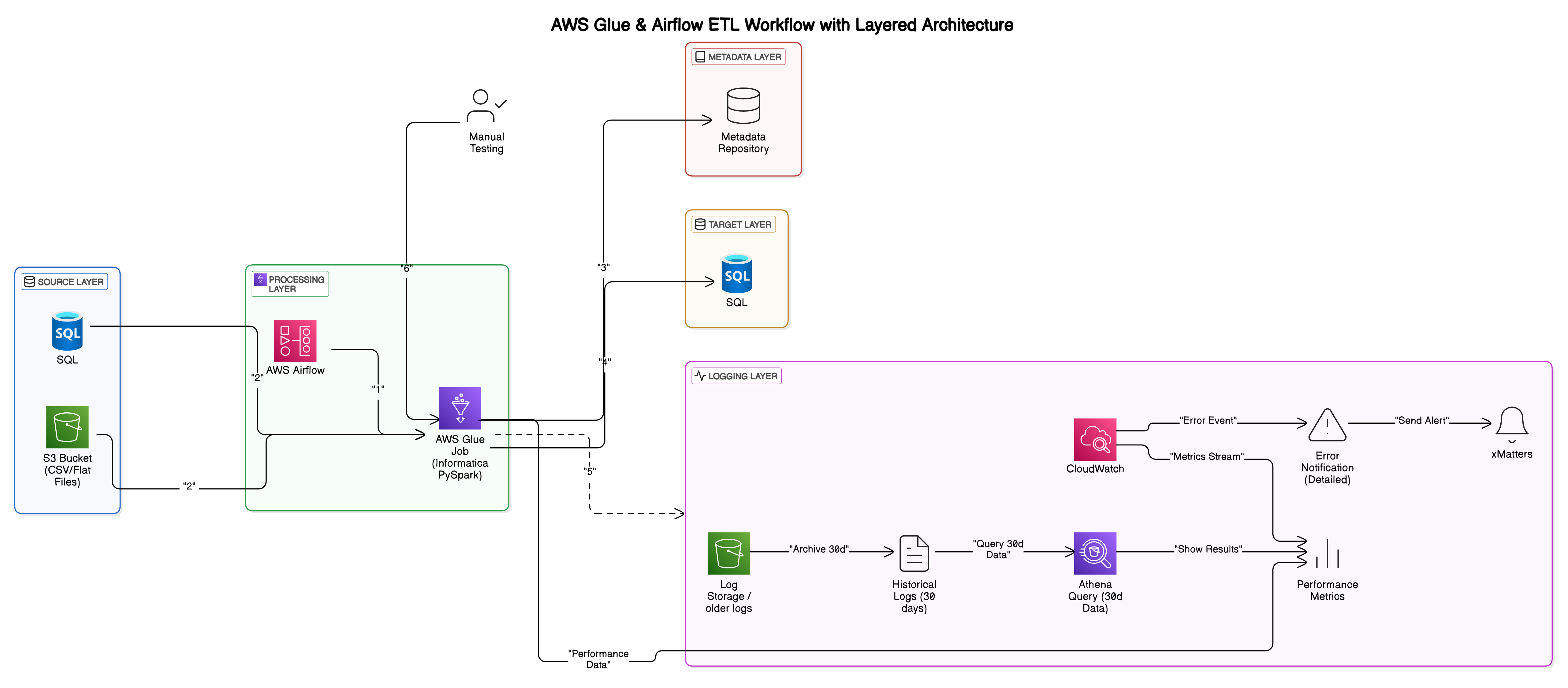
Given the analysis above, we plan to focus further discussions specifically on:

* **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.

**2.1 Approach 1: AWS Glue + Apache Airflow**

**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.

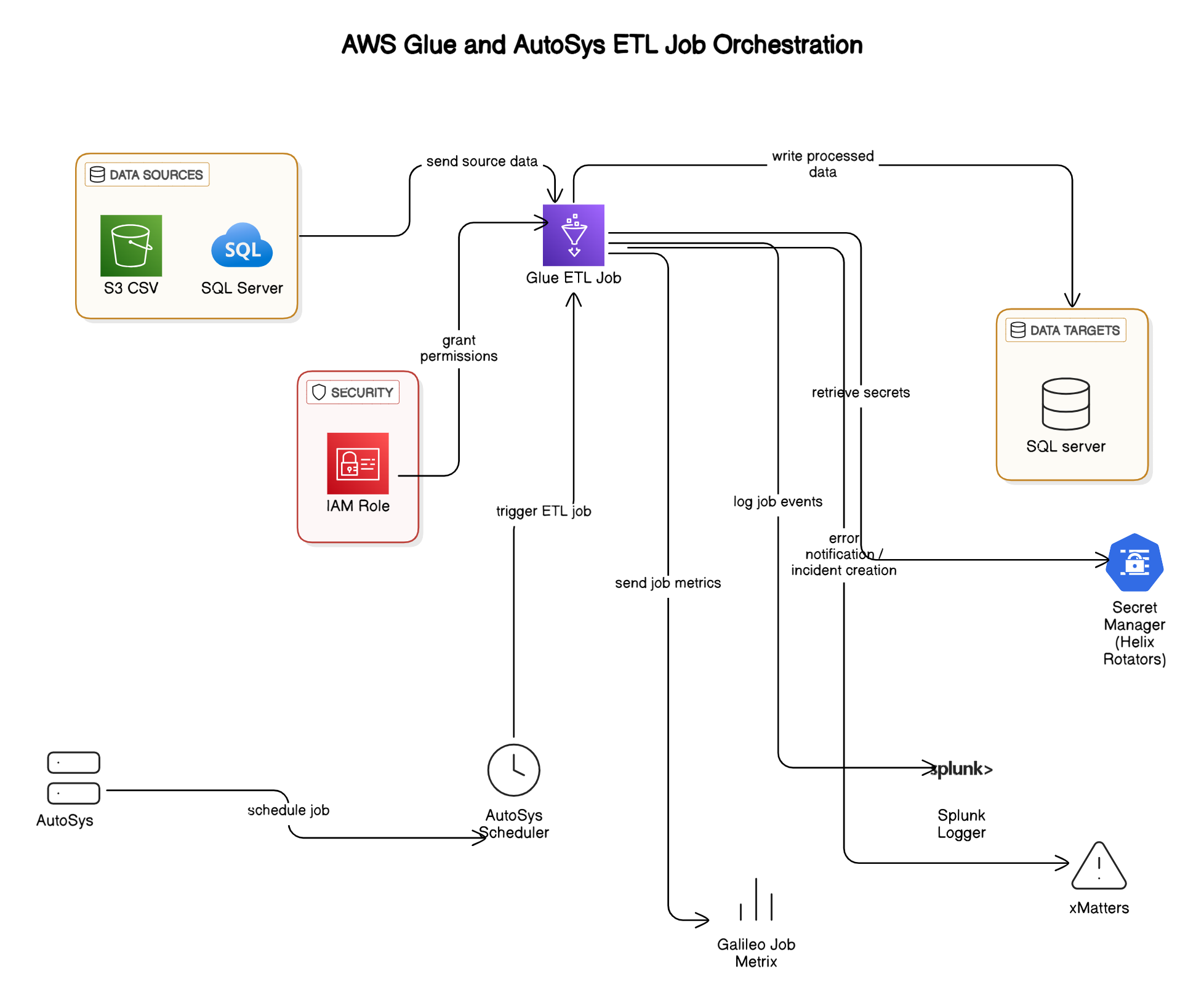
**6. Manual Testing**

* 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.

**2.2 Approach 1: AWS Glue + AutoSys**



**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.