**Informatica Workflow Modernization Approach**

Capital Group is migrating approximately > 800 Informatica PowerCenter workflows under the CDM (Customer Data Management) umbrella to AWS Glue using PySpark. The project is driven by the need to modernize legacy systems, reduce operational overhead, and adopt a modular, cloud-native architecture.

**Responsibilities:**

* Assess and convert CDM workflows (Initialize → Preland → Today → Delta).
* Handle SFDC to CDM integration (SFDC is a data source only).
* Exclude MDM (Master Data Management) workflows, which are managed by GlobalLogic.

**Layer Categorization:**

|  |  |  |
| --- | --- | --- |
| Layer | Workflow Count | Description |
| Initialize | 41 | Prepares previous day snapshots from today's layer. |
| Preland | 154 | Loads raw file data to staging tables with minimal transformation. |
| Today | 372 | Implements business rules; joins and transformations. |
| Delta | 237 | Performs CDC by comparing Today vs Previous. |
| Publish | 127 | Pushes curated data to MDM. Only non-MDM dependent jobs are in scope. |

Source Systems (e.g., SFDC)

↓

CDM (Initialize → Preland → Today → Delta)

↓

Publish Layer (from CDM to MDM)

↓

MDM SaaS (managed by GlobalLogic)

**Migration Challenge:**

* Incomplete understanding of every Informatica workflow and transformation.
* Need for parameterization to reduce redundant jobs (e.g., workflows differing only by table name).
* Alignment of code to CG's modular framework.
* Time constraint: 5–6 months for end-to-end migration.

**Approach 1: Custom AI-Based Tool**

**Description:** Develop a proprietary tool using LLMs to parse Informatica XMLs and generate modular PySpark code plus metadata (e.g., YAML config).

**Pros:**

* Fully customizable and reusable.
* Supports parameterization and modular design.
* Aligned to CG framework standards.

**Cons:**

* Limited understanding of workflow logic today increases risk.
* Requires initial metadata extraction to discover patterns.
* Needs strong validation due to AI inaccuracies in edge cases.

**Recommendation:**

* Begin by extracting transformation metadata from 20–30 XMLs.
* Use AI to identify patterns, tag complexity, and structure logic classification.

**Approach 2: DataSwitch + Manual Refactoring**

**Description:** Use DataSwitch to convert Informatica XMLs to PySpark (~60% automated), then manually restructure the code into a modular format.

**Pros:**

* Immediate head-start with auto-conversion.
* Saves effort on standard transformation mapping.

**Cons :**

* Non-modular flat code.
* Not Glue-compatible out-of-the-box.
* Cannot handle parameterization (1:many mappings).
* Post-conversion refactoring is manual and time-consuming.
* In 5–6 months, DataSwitch likely cannot convert all mappings to usable, tested code.

**Recommendation:**

* Use only for complex 1:1 workflows (e.g., Today, Delta).
* Allocate extra time (~1 month) for framework alignment.

**Approach 3: Hybrid Strategy (Recommended)**

**Description:** Split workloads by complexity and design:

* Simple/repeatable jobs → AI-based tool.
* Complex, unique logic → DataSwitch.

**Pros:**

* Balanced approach leverages speed + control.
* Enables modularization and standardization.

**Cons :**

* Lack of deep knowledge makes dividing workload difficult.
* No clear sync between DataSwitch output and reusable framework.
* DataSwitch can't consolidate jobs (e.g., 3 similar workflows with different table names).
* High risk of mistakes while parameterizing unless detailed analysis is done on converted code.

**Recommendation:**

* First extract metadata from XMLs.
* Classify workflows into:
  + Reusable (parameterizable)
  + Complex (unique transformation)
  + MDM dependent (exclude)
* Sync converted code with internal modular design.

**Approach 4: Phased Delivery Model (Stable Output Focus)**

**Description:** Divide the project into two structured phases:

**Phase 1: Data Stage Workflow Conversion**

* Focus solely on converting Informatica workflows to working PySpark scripts.
* Ensure seamless execution at a high level before modularization.

**Phase 2: Code Optimization and Refactoring**

* Focus on job reduction and modular restructuring.
* Apply CG framework, parameterization, and testing.
* Reduce redundant workflows by merging similar logic.
* Optimize PySpark code for performance and scalability.
* Minimize server resource usage (CPU/memory) through efficient logic.
* Reduce AWS Glue and infrastructure cost via optimization.
* Refactor jobs to remove unnecessary transformations.
* Enable auto-scalable processing using dynamic resource allocation.
* Implement performance metrics tracking per job.
* Generate operational analytics for workload and system monitoring.
* Automate data quality validation as part of the pipeline.
* Integrate automated incident creation workflows for process/data quality failures.

**Pros:**

* Produces a stable PySpark codebase early.
* No need to fully analyze deep transformation metadata initially.
* Allows better control and quality in the final optimization phase.
* Direct conversion-to-optimization model can outperform raw tool-based conversion.

**Cons:**

* May increase cost due to longer delivery window.
* Requires client agreement on phased handoff.
* Overall time may slightly increase due to two-stage validation.