**✅ Step-by-Step Project Plan**

**Phase 1: Metadata Extraction and Normalization**

**Goal:** Read Informatica XMLs, extract detailed metadata, and store in SQL.

**1. Store XML File Paths**

* Create table: INFORMATICA\_WORKFLOW\_XML\_FILE
* Store folder, layer, file path, and file name for each of the 1616 XMLs.

**2. Write Python Parser Script**

* Read rows from INFORMATICA\_WORKFLOW\_XML\_FILE
* Parse each XML using lxml or xml.etree.ElementTree
* Extract info and populate normalized tables (e.g., MAPPING, SESSION, SOURCE, etc.)

✅ **Output:** Full metadata extraction layer populated in SQL

**🔹 Phase 2: AI/ML/LLM Analysis for Common Patterns**

**Goal:** Use AI to identify reusable components across 800 workflows.

**1. Train/Use LLM for Workflow Analysis**

* Use OpenAI/GPT or fine-tuned model
* Give input like:
  + Workflow name
  + List of transformations
  + Mapping structure
  + Source/Target types
* Ask LLM to:
  + Find repeating mapping patterns (delta loads, SCDs, validation layers, etc.)
  + Group workflows by functionality (preland, delta, publish, etc.)
  + Suggest reusable modules

**2. Output JSON Format**

{

"DeltaLoad": {

"transformations": ["EXP\_DELTA\_FLAG", "FIL\_UPDATED\_ROWS", "RTR\_DELTA\_PROCESS"],

"reusable": true

},

"PublishLayer": {

...

}

}

✅ **Output:** Component classification for framework generation

**Phase 3: Framework Generation (Kedro + PySpark)**

**Goal:** Create a standardized Kedro pipeline from workflow metadata

**1. Design Kedro Pipelines**

* Create pipelines for layers: initialize, preland, delta, publish
* Generate nodes.py using transformation logic from metadata
* Create parameterized catalog.yml for source/target paths

**2. Create Code Generator**

Write a Python script to:

* Loop over metadata in SQL
* Based on AI-classified component
* Generate Kedro node functions dynamically

Example:

def exp\_delta\_flag(df):

return df.withColumn("IS\_DELTA", when(col("LAST\_UPDATED") > lit(run\_date), lit("Y")).otherwise("N"))

✅ **Output:** A Kedro codebase dynamically generated from Informatica workflow patterns

**Phase 4: Execution in AWS Glue**

**Goal:** Deploy Kedro-based ETL to Glue jobs

**1. Package Kedro Code**

bash

CopyEdit

kedro package

**2. Deploy to S3 and Create Glue Job**

* Upload to s3://your-bucket/kedro\_etl.zip
* Create Glue Spark job (glue\_run.py) to call Kedro

from kedro.framework.session import KedroSession

with KedroSession.create() as session:

session.run()

✅ **Output:** Glue job executes Kedro-generated Spark ETL

**Phase 5: Workflow Orchestration**

**Goal:** Run multiple Glue jobs in sequence or parallel using Autosys

**Option A: Autosys**

* Schedule Glue job using AWS CLI call from Autosys job

aws glue start-job-run --job-name kedro-etl-delta

**Option B: Airflow**

* Use Managed Airflow in AWS or self-hosted
* Define DAGs for each layer (delta -> publish)

start\_glue\_job = AwsGlueJobOperator(

job\_name="kedro-etl-delta", task\_id="start\_delta", dag=dag

)

**📦 Tools Used**

|  |  |
| --- | --- |
| **Component** | **Tool/Framework** |
| XML Parsing | Python (lxml, ElementTree) |
| Metadata Store | SQL Server or Postgres |
| AI Workflow Analysis | OpenAI/GPT or custom LLM |
| ETL Framework | Kedro + PySpark |
| Execution Engine | AWS Glue |
| Orchestration | Autosys / Airflow |
| Version Control | GitHub / GitLab |