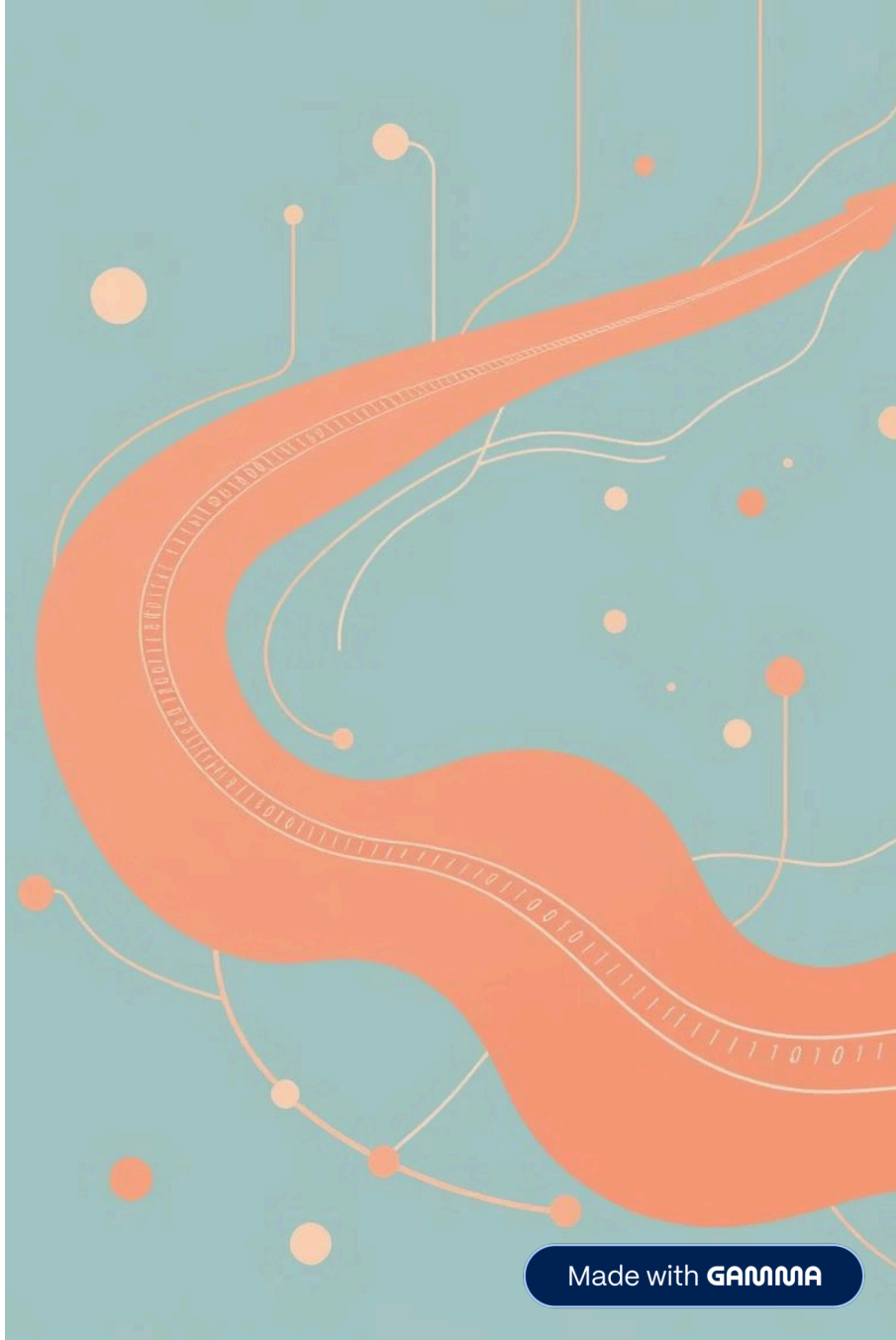
SSIS to PySpark Conversion Pipeline

This presentation outlines a robust methodology for converting SQL Server Integration Services (SSIS) packages (.dtsx) into equivalent, executable PySpark scripts.



Presentation Agenda

- Project Requirements & Scope
- Multi-Phase Conversion Approach
- Key Findings & Architecture
- Project Structure & Outputs
- Current Progress & Next Steps
- Technical Dependencies & Limitations
- Execution Flow Overview
- Conclusion & Future Outlook

Project Requirements: Bridging SSIS and PySpark

The core objective is to automate the migration of existing SSIS ETL logic into PySpark scripts, ensuring functional equivalence and maintainability.

1

Logic Replication

Translate SSIS ETL logic, including source/target configurations and transformations, into PySpark.

2

Configuration Extraction

Accurately extract source/target connection details and transformation rules from .dtsx packages.

3

Execution Flow

Replicate the original execution sequence and precedence constraints in the PySpark environment.

Important Note: We strictly utilize **metadata** extracted from SSIS packages; no sensitive data is processed or stored during this conversion.

Multi-Phase Conversion Approach

Our methodology is structured into three distinct phases, from metadata extraction to executable code generation.

1

Phase 1: .dtsx Parsing & Metadata Extraction

Parse .dtsx files to extract all essential ETL metadata, including source/target configurations and execution control. This phase generates detailed Excel outputs.

2

Phase 2: Visualization Generation

Utilize extracted metadata to generate visual flowcharts of data pipelines, aided by generative AI for clarity and understanding. This phase produces PNG diagrams.

3

Phase 3: PySpark Code Generation (In Progress)

Leverage AI models to generate runnable PySpark scripts based on the comprehensive metadata. Validation is pending due to environment setup.

Key Findings: Standard ETL Architecture

Our analysis of the SSIS solution revealed a common three-layer ETL architecture, orchestrated by a central scheduler package.

- **ODS (Operational Data Store):** Extracts and loads raw data from diverse source systems.
- **STA (Staging Area):** Performs intermediate transformations and processing on ODS data.
- **DWH (Data Warehouse):** Loads final, transformed data into fact and dimension tables for analytical consumption.
- **Scheduler.dtsx:** Acts as the master orchestrator, sequencing ODS, STA, and DWH package executions.

Project File Structure & Outputs

The project is organized to facilitate clear separation of SSIS source files, generated outputs, and visualization assets.

```
SSIS_Project/
├── SSIS/
│   └── SSIS_DWH/
│       └── SSIS_DWH/
│           ├── *.dtsx # ODS, STA, DWH, and Scheduler packages
│           ├── Code/
│           ├── Outputs/ # Multi-sheet Excel files
│           │   ├── .xlsx # Sheets: Source, Data Flow, Meta-Data, Params, Execution-Control
│           ├── Visualization/ # Graphviz-generated PNG flowcharts
│           │   ├── _flowchart.png
│           ├── requirement.txt # Dependency listing
│           ├── testing.ipynb # Initial prototyping
│           └── testing1.ipynb # Final implementation notebook
```

All generated outputs, including Excel metadata and visual flowcharts, are systematically stored in their respective directories.

Output File Descriptions

The conversion process generates detailed Excel sheets, each serving a specific purpose in documenting the SSIS package's metadata.

Sheet Name	Description
Source	Extracted connection and configuration details from SSIS data sources.
Data-Flow	Lists SSIS components, their types, and the data movement pipeline within the package.
Meta-Data	Describes column-level metadata, including name, data type, and length for each column.
Params	Includes parameters and variable mappings used in SSIS transformations.
Execution-Control	Captures control flow between tasks using precedence constraints.

Progress & Next Steps

We have successfully completed the initial phases, with code generation underway and future efforts focused on validation and expansion.

Current Status:

- **Phase 1:** ✅ Done – .dtsx parsing and metadata extraction completed.
- **Phase 2:** ✅ Done – Flowchart visualizations generated using AI and Graphviz.
- **Phase 3:** ⚠️ Partial – PySpark code generation is complete, but validation is pending.

Next Steps:

- Validate PySpark code in a test SQL Server environment.
- Expand conversion capabilities to STA & DWH layers.
- Replicate Scheduler.dtsx orchestration in Databricks Workflow.
- Improve AI prompts for higher quality PySpark output.
- Train/fine-tune AI model for consistent and accurate responses.

Technical Dependencies & Limitations

The solution relies on several key libraries and external tools, and it's important to be aware of certain constraints.

Key Dependencies:

- **Core Libraries:** `json`, `os`, `re`, `xml.etree.ElementTree`
- **Data Handling:** `pandas`, `xlsxwriter`
- **Visualization:** `graphviz`
- **Generative AI:** `google.generativeai`
- **Configuration:** `python-dotenv` (for Gemini API key)
- **Graphviz:** Requires external installation and path configuration.

Current Limitations:

- Generative AI output can be inconsistent and may "hallucinate" code.
- Lack of live validation for generated PySpark output due to environment limitations.
- Requires manual Graphviz installation for visualization generation.

Conclusion & Future Outlook

This project lays a robust foundation for automating SSIS to PySpark migrations, streamlining complex ETL transformations.

Automated Discovery

Successful pipeline for automated discovery of SSIS package logic.

Modular System

Structured for extensibility and scalability across various ETL layers.

AI-Assisted

Leverages generative AI for visualization and code transformation.

Future efforts will focus on achieving end-to-end validation, expanding coverage to more complex transformation logic, and enabling enterprise-grade data flows.