Agentic AI in Integrations Platform

Primary Author(s)	Zeynep Tuna
Date Created	10/17/25
Related Documentation	AI / Agentic Tool Exploration

Background and Motivation

As part of the Integrations Platform work, the developer experience involves using the sis-config-transforms repository to write SQL transforms that normalize data extracted from external APIs. The repository is configured to allow developers to run make test to verify their normalizations are working as expected. This involves a highly iterative process of writing SQL queries, testing, fixing failures, and retesting until all tests pass. Once successful, the developer commits the necessary files and creates a pull request (PR). After review and merge, custom-normalizer can then use these SQL files for data normalization.

This iterative process of writing and testing SQL queries until all tests pass is an ideal candidate for an agentic AI use case. Currently, engineers may manually perform this using coding assistant tools like Cursor and Claude Code at Clever. The goal is to automate this SQL test-driven development (TDD) loop with a custom agent.

If we imagine our overall Integrations Platform Experience as a process to be enhanced by AI, we can look into the below components as possible optimization areas outlined here:

1. Data source validation & field mapping
2. Data Extraction
3. Data Transformation

Suggested Implementation

For the purpose of increasing developer productivity, a simpler external UI / tool or a local IDE / terminal integration would be more efficient and appropriate.

As we plan to move towards an overall multi-agent system, we could start by envisioning the data normalization task as an automation - probably tightly coupled with GitHub depending on our final Integrations Platform Experience. Possibly a flow like this:

1. User creates a feature branch in repo to push the field mapping / crosswalk file
2. User Push Triggers AWS-GitHub Pipeline
3. Pipeline Invokes AWS Bedrock Agent
4. Agent Creates Expected CSV Outputs
5. Agent Generates SQL Transforms
6. TDD Loop (SQL Gen → Lambda Test → Iterate)
    a. Existing testing logic as a Lambda
7. Conditional Stop on Success or Limit Reach
8. Commit SQL & Open PR for user to review

But before we commit to the above, we might want to focus on a pilot to test if we're able to find the performance lift we're looking for with Generative AI / Agentic AI. We could use the Bedrock Playground to test the below components:

Component	Purpose	Effort
Agent	The core LLM and orchestrator.	**S**
Code Interpreter	The sandbox for **data manipulation** (JSON -> CSV) and **executing the test logic**.	**S**
Input Files	Holds the test JSON data and the Crosswalk document.	**S**

Custom Test Logic	**The Test Function:** DuckDB logic is *re-written* into a Python script to run within the Code Interpreter.	**S**

Tasks / Implementation Estimation

Task	Description	Implementation Estimation	Open Questions
1.1 Test Runner Lambda Setup	Compile the existing Go test runner logic into a Lambda.	**M**	Is this feasible with what we have in sis-config-transforms? We could also replicate the logic in another language and run directly in Code Interpreter?
1.2 Action Group Setup	Define the **Custom Action Group**, linking it to the Lambda.	**M**	
1.3 Data Staging Setup	Create a dedicated, secure **S3 bucket** for input data and test artifacts. Define necessary Bucket Policies.	**S**	Consider how we can connect this piece to the data extraction step to have input data files accessible.
2.1 Bedrock Agent	Create the	**S**	

Creation	**Bedrock Agent** with **Code Interpreter** and link the **Custom Action Group** from 1.2.		
2.2 Prompt Engineering	Write the complex, multi-step **Agent Instructions** that guide the TDD loop (JSON-to-CSV -> SQL Gen -> Call Test -> Fix).	**M**	
2.3 Agent Invocation Script	Script to invoke the Bedrock Agent, passing the initial S3 paths and handling the long-running process.	**M**	
3.1 Github -> AWS CodePipeline Setup	Create the pipeline upon GitHub push.	**M**	
3.2 Final Commit Logic	Create a separate step to download the final, passing SQL file from S3, run git add/commit/push back to the feature branch, and clean-up.	**L**	

3.3 Pull Request Integration	Configure the final step to open a PR or update an existing one with a summary of the agent's work.	**M**	