LANGGRAPH ARCHITECTURE

Multi-Agent AI Workflow for MSSQL to dbt Migration

Author: Alexander Garcia Angus
Property of: OKO Investments

Generated: December 02, 2025

1. OVERVIEW

The migration workflow uses **LangGraph**, a framework for building stateful, multi-agent workflows. This provides structured state management, visual workflow, checkpointing, AWS integration, and security guardrails.

Key Benefits:

* Structured State Management - TypedDict-based state with Pydantic validation

* Visual Workflow - Clear graph structure with conditional routing

* Checkpointing - State persistence for resumable migrations

* AWS Integration - Lambda functions and Step Functions deployment

* Security Guardrails - LLM input/output validation and SQL sanitization

2. ARCHITECTURE COMPONENTS

Component	File	Purpose
State Management	agents/state.py	TypedDict state structure
LangGraph Workflow	agents/graph.py	StateGraph orchestration
Agent Nodes	agents/nodes.py	Node functions for each agent
Security Guardrails	agents/guardrails.py	LLM input/output validation
Lambda Handlers	agents/lambda_handlers.py	AWS Lambda wrappers
CDK Infrastructure	aws/cdk_stack.py	Cloud infrastructure

3. STATE MANAGEMENT

MigrationState Structure:

Field	Type	Description
phase	Literal	assessment, planning, execution, evaluation, complete
models	List[Dict]	List of models to generate
current_model_index	int	Index of current model being processed
completed_count	int	Number of successfully completed models
failed_count	int	Number of failed models
assessment_complete	bool	Whether assessment phase is done
plan_complete	bool	Whether planning phase is done
assessment	Dict	Assessment results from first agent
planning	Dict	Planning results with execution order

metadata	Dict	MSSQL metadata input
project_path	str	Path to dbt project
errors	List[str]	Accumulated error messages
max_retries	int	Maximum rebuild attempts per model

4. WORKFLOW GRAPH STRUCTURE

The LangGraph StateGraph orchestrates a 6-agent workflow with conditional routing:

Agent Flow:

1. **START** -> Assessment Agent

2. **Assessment Agent** - Evaluates MSSQL metadata

3. **Planner Agent** - Creates migration plan, initializes model list

4. **Executor Agent** - Generates dbt model for current model (loop)

5. **Tester Agent** - Validates generated model

6. **Rebuilder Agent** - Fixes errors if test failed (conditional)

7. **Evaluator Agent** - Final validation of all models

8. **END** - Migration complete

Conditional Edges:

* **should_continue_migration** - After planner, check if models exist

* **should_rebuild_or_continue** - After tester, decide rebuild or advance

* **after_advance_check** - After advance, check if more models exist

5. AGENT NODES

Node Function	Agent	Purpose
assessment_node()	Assessment Agent	Analyze metadata, create assessment
planner_node()	Planner Agent	Create migration plan, initialize model list
executor_node()	Executor Agent	Generate dbt model for current model
tester_node()	Tester Agent	Validate generated model
rebuilder_node()	Rebuilder Agent	Fix errors, regenerate model
evaluator_node()	Evaluator Agent	Final validation of all models

6. SECURITY GUARDRAILS

Input Validation:

* Prompt injection detection

* Maximum length checks

* Dangerous pattern detection

Output Validation:

* JSON extraction from markdown

* SQL sanitization

* Dangerous SQL pattern blocking

Blocked SQL Patterns:

- DROP TABLE/DATABASE/SCHEMA/VIEW/INDEX

- DELETE FROM ... WHERE 1=1

- TRUNCATE TABLE

- EXEC xp_cmdshell

Rate Limiting:

Per-agent rate limits with time-windowed request tracking.

7. AWS INFRASTRUCTURE (CDK)

Resource	Purpose
S3 Bucket	State storage with versioning
6 Lambda Functions	One per agent
IAM Roles	Permissions for S3 and Secrets Manager
Secrets Manager	Stores Anthropic API key
Step Functions	Orchestrates workflow
CloudWatch Logs	Centralized logging

8. STATE FLOW EXAMPLE

Initial State:

phase: 'assessment', models: [], current_model_index: 0, assessment_complete: false, plan_complete: false

After Assessment:

phase: 'planning', assessment_complete: true, assessment: {total_objects: 7, tables: [...], strategy: {...}}

After Planning:

phase: 'execution', plan_complete: true, models: [{name: 'stg_customers', status: 'pending'}, ...]

During Execution:

phase: 'execution', current_model_index: 0, models: [{name: 'stg_customers', status: 'in_progress', attempts: 1}, ...]

After Completion:

phase: 'complete', completed_count: 7, failed_count: 0, models: [{name: 'stg_customers', status: 'completed', validation_score: 0.95}, ...]

9. ORIGINAL VS LANGGRAPH COMPARISON

Aspect	Original	LangGraph
State Management	JSON files	TypedDict + Pydantic
Workflow	Custom orchestrator	StateGraph
Persistence	Manual save/load	Built-in checkpointing
Visualization	None	Mermaid diagrams
Cloud Deployment	Manual	CDK infrastructure
Type Safety	Minimal	Full type hints
Error Handling	Custom	Framework-integrated
Testing	End-to-end only	Node + integration

10. LANGGRAPH BENEFITS SUMMARY

* **Type Safety** - Pydantic models catch errors early

* **Observability** - Clear state transitions, structured logging

* **Resumability** - Built-in checkpointing, S3 state persistence

* **Scalability** - Nodes on different machines, Lambda serverless

* **Testability** - Each node independently testable
