APEX Pipeline Orchestration Guide

Version: 1.0 **Date:** 2025-09-05 **Author:** APEX Development Team

Overview

Pipeline Orchestration is APEX's revolutionary approach to YAML-driven data processing workflows. This system embodies the core APEX principle that **all processing logic should be contained in the YAML configuration file**, eliminating hardcoded orchestration in Java code.

This guide provides comprehensive coverage of APEX's pipeline orchestration capabilities, from basic concepts to advanced enterprise implementations.

Table of Contents

1. Introduction to Pipeline Orchestration

What is Pipeline Orchestration?

Pipeline orchestration in APEX allows you to define complete data processing workflows using declarative YAML configuration. Instead of writing Java code to coordinate different processing steps, you describe the entire workflow in YAML and let APEX execute it.

The APEX Principle

Before (Hardcoded Java Orchestration):

```
// Traditional approach - hardcoded orchestration
List<Customer> customers = csvReader.readCustomers("input.csv");
for (Customer customer : customers) {
    Customer enriched = enrichmentService.enrich(customer);
    Customer validated = validationService.validate(enriched);
    databaseService.insert(validated);
    auditService.log(validated);
}
```

After (YAML-Driven Orchestration):

```
// APEX approach - YAML-driven orchestration
pipelineEngine.executePipeline("customer-etl-pipeline");
```

```yaml
# All orchestration logic in YAML
pipeline:
  name: "customer-etl-pipeline"
  steps:
    - name: "extract-customers"
      type: "extract"
      source: "customer-csv-input"
      operation: "getAllCustomers"

    - name: "load-to-database"
      type: "load"
      sink: "customer-h2-database"
      operation: "insertCustomer"
      depends-on: ["extract-customers"]

    - name: "audit-logging"
      type: "audit"
      sink: "audit-log-file"
      operation: "writeAuditRecord"
      depends-on: ["load-to-database"]
      optional: true
```

Key Benefits

1. **Declarative Configuration**: Describe what you want, not how to achieve it
2. **No Java Orchestration Code**: All workflow logic in YAML configuration
3. **Dependency Management**: Automatic step dependency resolution and validation
4. **Error Handling**: Configurable error handling strategies
5. **Monitoring**: Built-in execution tracking and metrics
6. **Maintainability**: Easy to modify workflows without code changes
7. **Testing**: Pipeline configurations can be validated and tested independently

2. Core Concepts

Pipeline

A **pipeline** is a complete data processing workflow consisting of multiple steps executed in a specific order.

Steps

Steps are individual processing units within a pipeline. APEX supports four step types:

- **Extract**: Read data from external sources
- **Load**: Write data to external destinations
- **Transform**: Modify data between steps
- **Audit**: Create audit trails and compliance records

Dependencies

Dependencies define the execution order of steps. Steps can depend on one or more other steps, creating a directed acyclic graph (DAG) of execution.

Data Flow

Data flow is the automatic passing of data between pipeline steps through the pipeline context.

Data Sources and Sinks

- **Data Sources**: External systems that provide input data (CSV files, databases, APIs)
- **Data Sinks**: External systems that receive output data (databases, files, message queues)

3. Getting Started

Prerequisites

- APEX Rules Engine 1.0 or later
- Java 21 or later
- Maven 3.6 or later

Your First Pipeline

Let's create a simple pipeline that reads data from a CSV file and writes it to a database:

Step 1: Create the Pipeline Configuration

```yaml
# my-first-pipeline.yaml
metadata:
  name: "My First Pipeline"
  version: "1.0.0"
  description: "Simple CSV to database pipeline"

pipeline:
  name: "csv-to-db-pipeline"
  description: "Read CSV data and write to database"

  steps:
    - name: "extract-data"
      type: "extract"
      source: "csv-input"
      operation: "getAllRecords"

    - name: "load-data"
      type: "load"
      sink: "database-output"
      operation: "insertRecord"
      depends-on: ["extract-data"]

data-sources:
  - name: "csv-input"
    type: "file-system"
    connection:
      basePath: "./data"
      filePattern: "input.csv"
    fileFormat:
      type: "csv"
      hasHeaderRow: true
    queries:
      getAllRecords: "SELECT * FROM csv"
```

```yaml
data-sinks:
  - name: "database-output"
    type: "database"
    sourceType: "h2"
    connection:
      database: "./output/data"
      username: "sa"
      password: ""
    operations:
      insertRecord: |
        INSERT INTO records (id, name, value)
        VALUES (:id, :name, :value)
```

Step 2: Execute the Pipeline

```
// Load configuration
YamlRuleConfiguration config = YamlConfigurationLoader
    .loadFromFile("my-first-pipeline.yaml");

// Initialize pipeline engine
DataPipelineEngine pipelineEngine = new DataPipelineEngine();
pipelineEngine.initialize(config);

// Execute pipeline
YamlPipelineExecutionResult result = pipelineEngine
    .executePipeline("csv-to-db-pipeline");

// Check results
System.out.println("Pipeline success: " + result.isSuccess());
System.out.println("Duration: " + result.getDurationMs() + "ms");
System.out.println("Steps completed: " + result.getSuccessfulSteps() + "/" + result.getTotalSteps());
```

Working Demo

APEX includes a complete working demo that demonstrates pipeline orchestration:

```
# Run the CSV to H2 Pipeline Demo
java -cp apex-demo/target/apex-demo-1.0-SNAPSHOT-jar-with-dependencies.jar \
     dev.mars.apex.demo.etl.CsvToH2PipelineDemo
```

This demo processes 10 customer records from CSV to H2 database in approximately 23ms, demonstrating the performance and reliability of APEX pipeline orchestration.

4. Pipeline Configuration

Basic Pipeline Structure

Every pipeline configuration follows this structure:

```yaml
metadata:
  name: "Pipeline Name"
```

```
  description: "What this pipeline does"
  version: "1.0.0"

pipeline:
  name: "pipeline-identifier"
  description: "Detailed description"

  steps:
    # Step definitions

  execution:
    # Execution configuration

  monitoring:
    # Monitoring configuration

data-sources:
  # Input data sources

data-sinks:
  # Output data sinks
```

Pipeline Metadata

```
metadata:
  name: "Customer Processing Pipeline"
  version: "1.0.0"
  description: "Complete customer data processing workflow"
  author: "Data Team"
  tags: ["etl", "customers", "production"]
```

Pipeline Definition

```
pipeline:
  name: "customer-processing-pipeline"
  description: "Extract, validate, enrich, and load customer data"

  steps:
    - name: "extract-customers"
      type: "extract"
      source: "customer-csv-input"
      operation: "getAllCustomers"
      description: "Read customer data from CSV file"

    - name: "validate-customers"
      type: "transform"
      description: "Validate customer data quality"
      depends-on: ["extract-customers"]
      transformations:
        - type: "validation"
          rule-group: "customer-validation-rules"

    - name: "enrich-customers"
      type: "transform"
      description: "Enrich customer data with additional information"
      depends-on: ["validate-customers"]
      transformations:
        - type: "enrichment"
          enrichment-id: "customer-profile-enrichment"
```

```yaml
    - name: "load-customers"
      type: "load"
      sink: "customer-database"
      operation: "upsertCustomer"
      description: "Load enriched customer data to database"
      depends-on: ["enrich-customers"]

    - name: "audit-processing"
      type: "audit"
      sink: "audit-log"
      operation: "logProcessingResults"
      description: "Create audit trail"
      depends-on: ["load-customers"]
      optional: true
```

Execution Configuration

```yaml
pipeline:
  execution:
    mode: "sequential"  # or "parallel"
    error-handling: "stop-on-error"  # or "continue-on-error"
    max-retries: 3
    retry-delay-ms: 1000
    timeout-ms: 300000  # 5 minutes
```

Monitoring Configuration

```yaml
pipeline:
  monitoring:
    enabled: true
    log-progress: true
    collect-metrics: true
    alert-on-failure: true
    performance-tracking: true
```

5. Step Types and Configuration

Extract Steps

Extract steps read data from external data sources:

```yaml
steps:
  - name: "extract-customers"
    type: "extract"
    source: "customer-csv-input"  # Data source name
    operation: "getAllCustomers"  # Named query/operation
    description: "Read customer data from CSV file"
    parameters:
      limit: 1000
      offset: 0
      filter: "status = 'ACTIVE'"
```

Common Extract Patterns:

- CSV file extraction
- Database query execution
- REST API data retrieval
- JSON/XML file parsing

Load Steps

Load steps write data to external data sinks:

```yaml
steps:
  - name: "load-to-database"
    type: "load"
    sink: "customer-h2-database"  # Data sink name
    operation: "insertCustomer"   # Named operation
    description: "Insert customers into database"
    depends-on: ["extract-customers"]
    parameters:
      batch-size: 100
      upsert: true
      conflict-resolution: "update"
```

Common Load Patterns:

- Database record insertion/update
- File output generation
- REST API data posting
- Message queue publishing

Transform Steps

Transform steps modify data between extraction and loading:

```yaml
steps:
  - name: "transform-customers"
    type: "transform"
    description: "Apply business transformations"
    depends-on: ["extract-customers"]
    transformations:
      - name: "add-processing-timestamp"
        type: "field-addition"
        field: "processed_at"
        value: "CURRENT_TIMESTAMP"

      - name: "validate-email"
        type: "validation"
        field: "email"
        rule: "email-format"

      - name: "enrich-customer-data"
        type: "enrichment"
        enrichment-id: "customer-profile-lookup"

      - name: "calculate-risk-score"
        type: "calculation"
        field: "risk_score"
```

```
        expression: "#creditScore * 0.6 + #incomeLevel * 0.4"
```

Audit Steps

Audit steps create audit trails and compliance records:

```yaml
steps:
  - name: "audit-processing"
    type: "audit"
    sink: "audit-log-file"
    operation: "writeAuditRecord"
    description: "Create audit trail for processed records"
    depends-on: ["load-to-database"]
    optional: true  # Won't fail pipeline if it fails
    audit-config:
      include-original-data: true
      include-transformed-data: true
      include-metadata: true
      retention-days: 2555  # 7 years
```

6. Dependency Management

Declaring Dependencies

Steps can declare dependencies on other steps:

```yaml
steps:
  - name: "extract-customers"
    type: "extract"
    # No dependencies - runs first

  - name: "extract-orders"
    type: "extract"
    # No dependencies - can run in parallel with extract-customers

  - name: "join-customer-orders"
    type: "transform"
    depends-on: ["extract-customers", "extract-orders"]  # Wait for both

  - name: "load-to-warehouse"
    type: "load"
    depends-on: ["join-customer-orders"]  # Wait for transformation
```

Dependency Validation

APEX automatically validates dependencies:

- **Missing Dependencies**: Ensures all referenced steps exist
- **Circular Dependencies**: Detects and prevents infinite loops
- **Topological Sorting**: Orders steps for correct execution

Dependency Patterns

Linear Dependencies

```yaml
# A → B → C → D
steps:
  - name: "step-a"
    type: "extract"

  - name: "step-b"
    type: "transform"
    depends-on: ["step-a"]

  - name: "step-c"
    type: "transform"
    depends-on: ["step-b"]

  - name: "step-d"
    type: "load"
    depends-on: ["step-c"]
```

Parallel Processing

```yaml
# A → B, A → C, B+C → D
steps:
  - name: "step-a"
    type: "extract"

  - name: "step-b"
    type: "transform"
    depends-on: ["step-a"]

  - name: "step-c"
    type: "transform"
    depends-on: ["step-a"]

  - name: "step-d"
    type: "load"
    depends-on: ["step-b", "step-c"]
```

Fan-Out Pattern

```yaml
# A → B, A → C, A → D (one source, multiple destinations)
steps:
  - name: "extract-data"
    type: "extract"

  - name: "load-to-warehouse"
    type: "load"
    sink: "data-warehouse"
    depends-on: ["extract-data"]

  - name: "load-to-cache"
    type: "load"
    sink: "redis-cache"
    depends-on: ["extract-data"]

  - name: "send-to-api"
    type: "load"
    sink: "external-api"
    depends-on: ["extract-data"]
```

This comprehensive guide continues to build upon the foundation, providing detailed information about each aspect of APEX pipeline orchestration.