Spring Batch + Spring Integration: Detailed Step-by-Step Guide

1. Objective

To execute a Spring Batch job only after the Spring Boot application has fully started. This is achieved by integrating Spring Batch with Spring Integration and triggering the job using ApplicationReadyEvent.

2. Components Overview

1. Spring Batch: Defines a batch job with one or more steps and executes them sequentially or conditionally.

2. Spring Integration: Orchestrates message flow between components, handles job launch requests.

3. ApplicationReadyEvent: Listens for application startup completion and initiates the job execution.

3.1 Integration Configuration

This configuration defines the message channels, the transformation logic to create a JobLaunchRequest, and the gateway to launch the job. See below for code snippet.

3.2 Batch Configuration

Defines the batch job and its steps. Each step is executed with a tasklet. Refer to the code example below.

3.3 Triggering Job on Application Startup

This component listens to ApplicationReadyEvent and sends a message to the integration flow's inputChannel.

4. Execution Flow

1. ApplicationReadyEvent fires when the Spring context is fully initialized.

2. Listener sends a message to the inputChannel.

3. Integration Flow processes the message: transforms payload to JobLaunchRequest and sends to jobLaunchChannel.

4. JobLaunchingGateway launches the batch job using JobLauncher.

5. Batch job processes steps and completes execution.

5. Diagram

1. Application Startup

2. ApplicationReadyEvent

3. Listener -> inputChannel

4. Integration Flow -> Transform payload -> Send to jobLaunchChannel

5. JobLaunchingGateway -> Launch Batch Job

6. Batch Job Execution

6. Logs Output

Expected logs during execution:

INFO - Application is ready. Triggering batch job...

INFO - Transforming payload into JobLaunchRequest.

INFO - Batch job launched successfully.

Executing step...

Step completed.

INFO - Batch job completed successfully.

7. Key Notes

1. Thread-Safe Execution: Spring Integration ensures thread-safe execution of the job.

2. Dynamic Parameters: You can pass dynamic job parameters in the JobLaunchRequest.

3. Avoid Deprecated Classes: Ensure no deprecated classes are used (as demonstrated).