CalSAWS

California Statewide Automated Welfare System

Design Document

CA-213257

Build Streams Architecture for Batch

CalSAWS	DOCUMENT APPROVAL HISTORY	
	Prepared By	Kevin Hooke
	Reviewed By	[individual(s) from build and test teams that reviewed document]

DATE	DOCUMENT VERSION	REVISION DESCRIPTION	AUTHOR
02/07/2020	1	Initial Revision	Kevin Hooke
02/10/2020	2	Updated few sections	Milind Nirgun
03/06/2020	3	Updated class diagram and added sequence diagrams	Kevin Hooke

Table of Contents

1 OVERVIEW

This document outlines the technical design for the architecture framework to support refactoring existing Batch jobs to use an event-driven, Streaming approach. This architecture framework is the basis for building new Streaming applications.

1.1 Current Design

The current Batch job design follows a traditional batch processing pattern: a driving query selects rows to be processed, then processing logic is invoked for each of the returned rows. An architecture BatchDriver is used as the 'job runner'. Each Batch job implements an architecture interface called BatchModule, which provides processing lifecycle methods, launch(), execute(), terminate(), which are called by the BatchDriver and supporting architecture framework.

1.2 Requests

As the CalSAWS system is expanded to include all California counties, the volume of data to be processed during each nightly batch window by certain Batch jobs is projected to run longer than the time available. An alternative processing solution is needed to ensure that daily batch processing completed within the batch window and meets SLA requirements.

1.3 Overview of Recommendations

1. The operations and support processes for the current Batch architecture are well defined and tested. The startup of the new Streams Processing applications should be automated. Even though they can run indefinitely, designing a runtime architecture for the new Streams processing apps that can take advantage of current operations processes will be an easier transition from the current traditional Batch processing approach to a new Streams processing approach. This Streams Processing architecture will allow for the Streams applications to be started and terminated as needed by the existing Batch scheduler.

2. Running the Streams Processing applications via the existing Batch Scheduler will allow any log output from the new applications to be captured as batch logs, similarly to how job logs are captured today. This enables the Batch Operations team to consistently manage and monitor all Batch operations regardless of their type.

3. Build a runtime Streams architecture framework that provides a similar launch(), execute(), terminate() template that is familiar to existing Batch developers, and also provides Kafka specific setup and configuration that will be common for each Streams application.

4. Use the Apache Avro serialization framework, to take advantage of
    a. Compressed, binary messages for lower bandwidth usage
    b. Schemas that define the content of each message, and support changes to the message structure providing inherent backward compatibility with previous versions.

5. Provide a framework for processing changes in the application data in near realtime as they happen in the online transaction system. This method, called Stream Processing, processes smaller chunks of data throughout the day as opposed to processing the full days' worth of data changes during nightly batch using a traditional Batch Processing approach.

1.4 Assumptions

1. Continual processing of smaller volumes of data throughout the day by a Streams processing application on a near-realtime basis will not have a functional impact on the Online system. The Streams jobs will be designed to ensure any functional impacts are avoided (for example, using the Case Lock functionality).

2. Existing Batch jobs that are targeted for migration to this new framework will be redesigned to take advantage of processing data on a near-realtime basis, and changed to handle any consequences from this fundamental change in approach.

2 RECOMMENDATIONS

This section describes the design for the Streams Architecture Framework.

2.1 Streams Processing Architecture

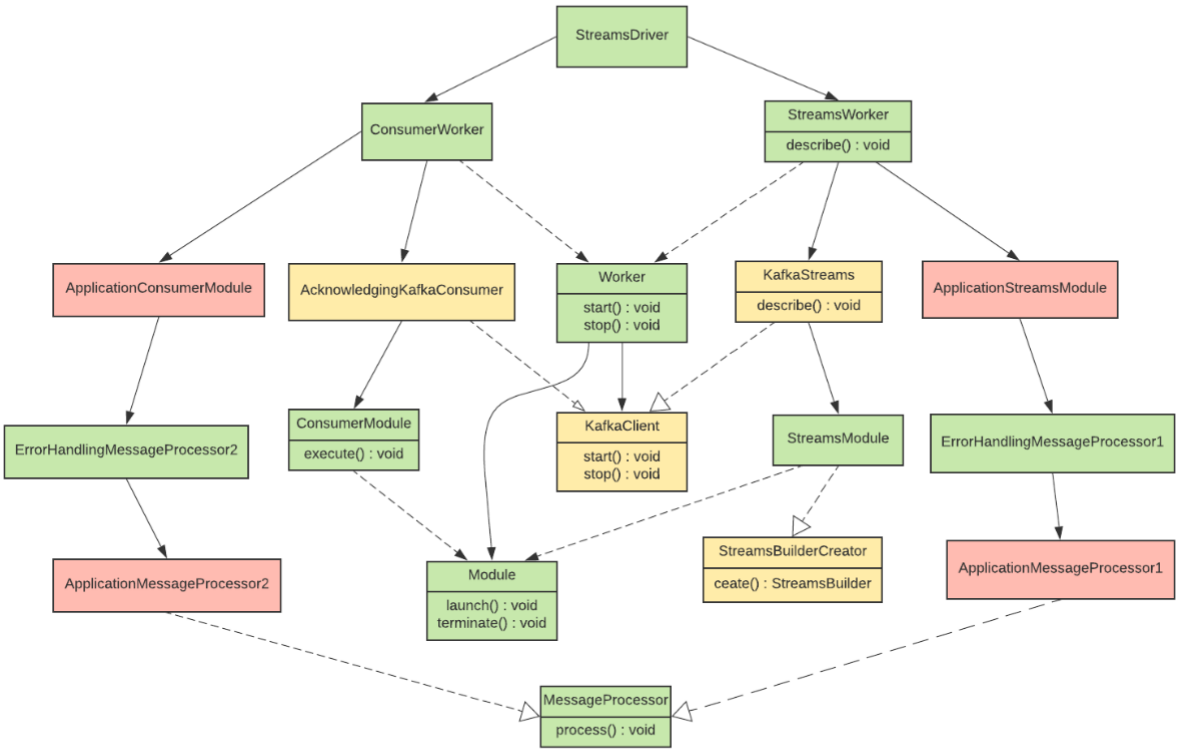
2.1.1 Overview

The Streaming Architecture support classes provide support for developing and running Streams application.

2.1.2 Class Diagram

Streams Architecture classes

Application implementation classes



2.1.3 Class Descriptions

StreamsDriver

- Provides the runtime entry point for a Kafka Streams API or Kafka Consumer API Application. Invoked by the Batch Scheduler with class name of the application Module to start executing.

StreamsModule

- Abstract class implemented by functional application where Kafka Streams API usage is needed
- Application provides implementations for:
    - launch() – initialization and setup for the Streams application
    - create() – creates the StreamsBuilder for the Stream topology
    - terminate() – teardown and cleanup logic for the Streams application

ConsumerModule

- Abstract class implemented by functional application where Kafka Consumer API usage is needed
- Application provides implementation for:
    - execute() –Consumer API usage by functional application

ErrorHandlngMessageProcessor

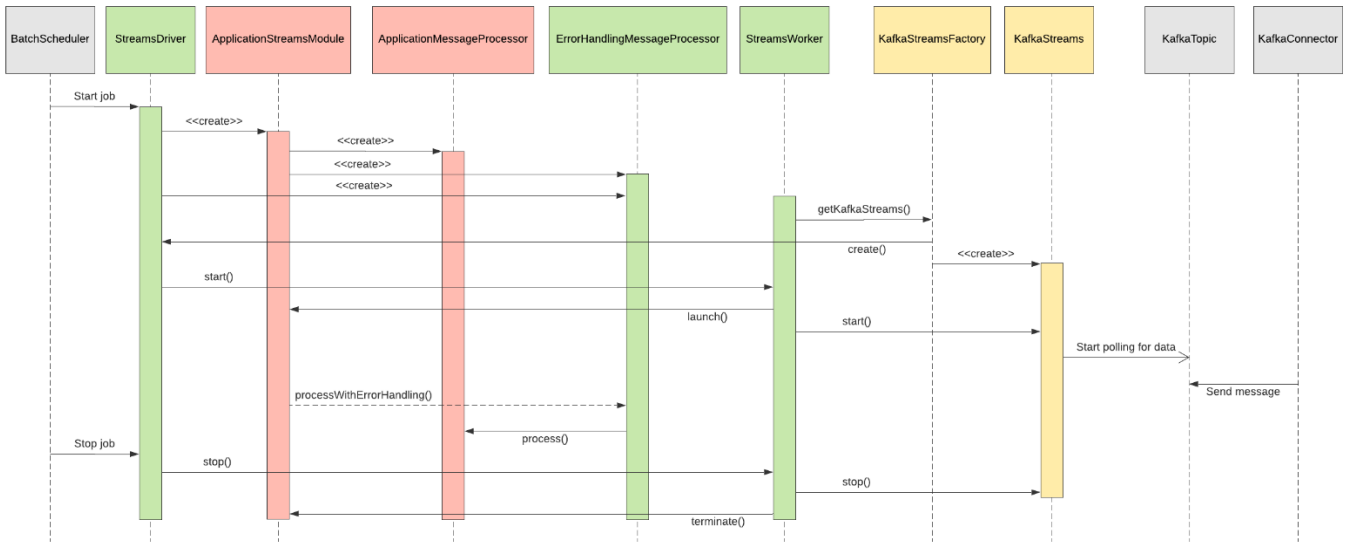
- A wrapper for functional application MessageProcessor implementations that provides standard error handling approaches. Also provides a location where additional common error handling approaches can be added as needed.
- Each of the provided handlers wrap the application's MessageProcessor.processor() method
    - processSkipOnException() – catches and logs exceptions, then continues processing next message
    - processSendToDLTOnException() – catches exceptions and forwards the message that caused the exception to a 'Dead Letter Topic' for later followup and triage (or additional error handling processing in another Streams application)

MessageProcessor

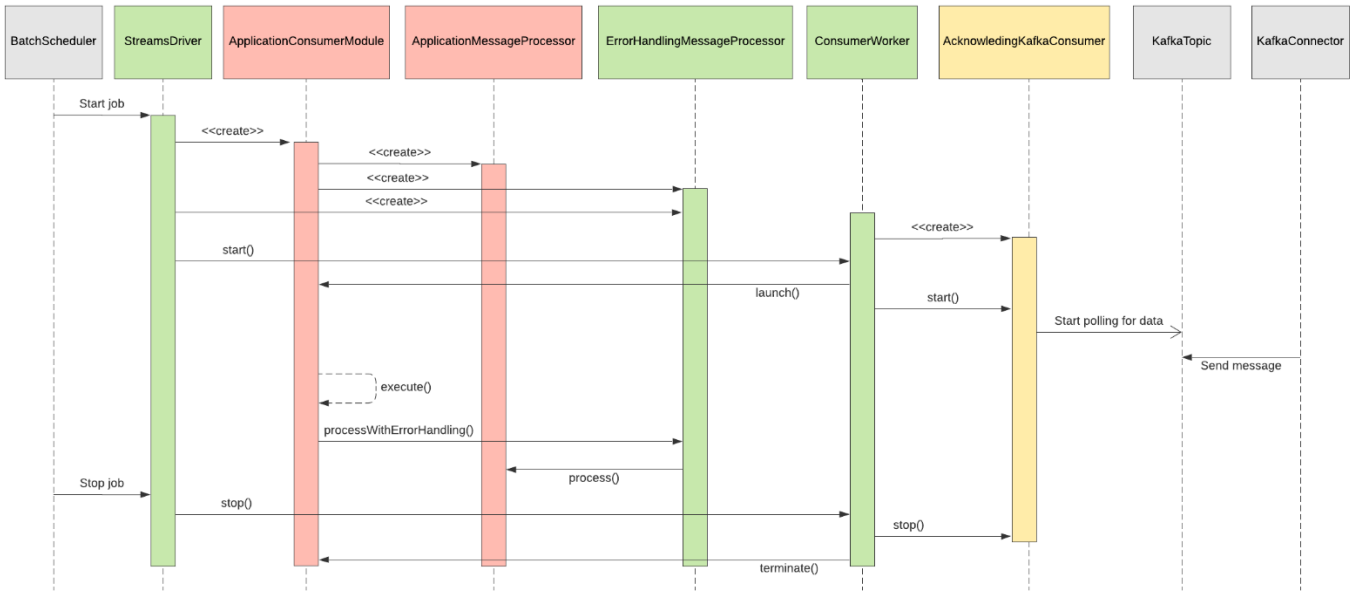
- Interface providing the process() method. Functional application implementations of this interface provide the business logic to handle a message from a source Topic or Stream. MessageProcessor implementations are intended to be independent of Kafka and Kafka library dependencies, which means the logic is easily unit tested outside of the Streams application itself.

2.1.4 Sequence Diagrams

2.1.4.1 Sequence diagram for StreamsModules



2.1.4.2 Sequence diagram for ConsumerModules



2.1.5 Impacts to Existing Batch Architecture / Batch Jobs

The new Streaming architecture classes have no impact on the existing Batch architecture classes as they do not change any dependencies and are designed to be used in parallel to the existing Batch architecture.

This new Streams architecture can be deployed in production along with the existing Batch architecture and it only gets used by the new Streams processing jobs when they are deployed.

2.1.6 Execution Frequency

No impact. This will be configured for the application classes using the Streaming architecture.

3 SUPPORTING DOCUMENTS

None.

Number	Functional Area	Description	Attachment

4 REQUIREMENTS

4.1 Project Requirements

REQ #	REQUIREMENT TEXT	How Requirement Met

4.2 Migration Requirements

DDID #	REQUIREMENT TEXT	Contractor Assumptions	How Requirement Met

5 MIGRATION IMPACTS

None.

6 OUTREACH

None.

7 APPENDIX

None.