Kafka Streams Data (Re)Processing Scenarios

Overview

This page gives an overview of data (re)processing scenarios for Kafka Streams. In particular, it summarizes which use cases are already support to what extend and what is future work to enlarge (re)processing coverage for Kafka Streams.

Use this page as a feature wish list for data (re)processing.

If your use case is missing, feel free to add it (let us know if you need access credentials: http://kafka.apache.org/contact)! Any issue you observe with regard to (re)processing can only be fixed if the community is aware that there is an issue. Just describe your scenario with expected behavior (there is no need to provide a solution for adding new scenarios). It would of course also be helpful if you describe why your scenario is currently not covered.

Use Case Scenarios

In the list below, each scenario is described in detail with regard to use case, expected behavior, available tooling, and best practice guidelines. The goal is to provide an comprehensive overview and step-by-step guideline for all kind of (re)processing scenarios.

The table is color coded as follows:

- ***green*** for supported scenarios
- ***yellow*** for scenario that are not fully supported
  - missing tooling, but scenario is clear
  - manual workaround available
- ***red*** for not supported scenarios
  - hard to solve

Scenario	Use Cases	Expected Behavior / Requirements	Available Tooling	Step-by-Step Guideline/ Best Practice	Limitations/ Known Issues	External Resources

Data reprocessing from scratch	• development and testing • rollback after bug fixes in production • A/B testing • demoing for customers or other stakeholders • replay for new business logic (Kappa architecture)	• After running and stopping an application you want to reset your application back to "zero". • Thus, on restart, the application reprocesses your data the same way as in its original run (assuming that the original input data still exists in Kafka in its entirety). Requirements: • Application must start consuming input topics from scratch (no committed offsets) • The application's internal state must be empty • Auto-created created topics must be empty (or deleted)	• Application Reset Tool: `bin/kafka-streams-application-reset.sh` • Local cleanup API: `KafkaStreams#cleanUp()`	1. stop all running application instances 2. if required: a. delete and re-create output topics manually b. use different/new output topics 3. run application reset tool 4. before restart, make sure to call `KafkaSteams#cleanUp()` for each application instance	• all data from input topics must still be available (i.e., no input data is lost due to log retention or compaction) • no support to handle output topics: ◦ by default, new application run appends data to originally used output topics ◦ manual fixed: ▪ delete and recreate output topic manually ▪ change application and use different/new output topics	• https://www.confluent.io/blog/data-reprocessing-with-kafka-streams-resetting-a-streams-application/ • https://groups.google.com/forum/?utm_medium=email&utm_source=footer#!msg/confluent-platform/3OrEmEM46z8/ai5B-jHkBQAJ
Data reprocessing with specific starting point (reprocessing from scratch; i.e., empty state)	• partial rollback after bug fixes in production • A/B testing	Similar to "Data Reprocessing from Scratch". However, instead of restarting the application at offsets zero, the user wants to specify a specific starting point. Requirement: • Same as "Data Reprocessing from Scratch" • Allow user to specify a (valid/consistent) starting point (offsets?, timestamp?)	• Application Reset Tool: `bin/kafka-streams-application-reset.sh` • Local cleanup API: `KafkaStreams#cleanUp()` ***Missing:*** API/tooling to set starting point.	Similar to "Data reprocessing from scratch". Manual workaround: Use a consumer client to `seek()` to desired starting offsets and `commit()` than. This step must be done after the reset tool was used and before the application gets restarted.	• see "Data Reprocessing from Scratch"	• https://groups.google.com/forum/?utm_medium=email&utm_source=footer#!msg/confluent-platform/3OrEmEM46z8/ai5B-jHkBQAJ
Data reprocessing using old application state	• A/B testing with stateful start • rollback after bug fix in production (application was redeployed include a bug at time X, go back to X and reprocess data with fixed application)	Requirement: • New application needs (historical) state of old application at point X.				• http://data-artisans.com/turning-back-time-savepoints/ • https://www.mapr.com/blog/savepoints-apache-flink-stream-processing-whiteboard-walkthrough

Processing cold data	• development • A/B testing	• processing cold/old /offline topics (i.e., process topics that do not have active producers) • application stops automatically after it processed all available data Requirement: • application should have an auto-stop feature (KIP-95)		***Workaround*** Manual stop required at the moment: 1. monitor consumer lag via `bin /kafka-consumer-groups.sh` 2. when consumer lag is zero, stop application manually		
Incremental processing (time driven)	• "batch like" processing	• start application in regular intervals (like cron job) and application automatically stops processing after a processing data for a specific time (wall-clock)	Not required.	• Put a `sleep()` after application startup and close application after sleep-time passed. To make it robust for failure restart, `sleep()` should not get a hard coded parameter passed in, but rather the difference to `endTime - startupTime`. or • Run app "forever" as for regular stream processing case and terminate application from outside when "stop time" is reached.	• not very precise with regard to event-time processing (i.e., stopping point is not related to application progress)	
Incremental processing (data driven)	• "batch like" processing	• start application in regular intervals (like cron job) • application stops automatically at some point • on application restart, it resumes from previous run • while application is running, new data might be appended to input topics Requirement: • application must have an auto-stop feature (KIP-95)		***Workaround*** • follow approach for "Incremental processing (time driven)"	• processing elapse time must be shorter than startup interval (i.e., start processing each hour, processing takes less than an hour)	• http://stackoverflow.com/questions/39048923/stop-a-kafka-streams-app
Offline application upgrade	• application bug fixes / improvements in production	• an application should be replaced with a newer version • new version resumes where old version left off • no reprocessing of old data	Not required.	• stop all running application instances • start new version of your application (same `application.id`) New and old application must be "compatible". Compatible changes: • changing a filter condition • inserting a new filters/map (record-by-record operation) Incompatible changes: • changing the structure of topology DAG • changing data types of stateful operations (like aggregations / joins)	• works only if application downtime is acceptable • new application must have similar structure than old one • Only newly produced output is "fixed"	
Online application upgrade	• application bug fixes / improvements in production • downstream application consumer data live and are not interesting in "correcting" previous result (because computation happened already and there is no interest in "correcting" old stuff)	• an application should be replaced with a newer version • new application is deployed in parallel • when the new application is "ready to take over", the old application is stopped • new application might start from an older offset and reprocess some data (w/ or w/o initial state)				

Reprocessing of "historical" data	• reprocess all data from yesterday / last week / April • "batch like" processing	• old data should be reprocessed (new version of application or completely different application) • result must be exact with regard to even-time (i.e., not include any older data and also take late arrivals into account) • new result might replace old results (i.e., update downstream database)				