

Discounts Processor (Kafka Streams version)

Git commit with doc: 307097e

Author(s): Paulo Jerônimo

[HTML version](#)

1. Overview	3
2. Processors Rules	4
2.1. Processor Configuration	4
2.2. Continuous View Processor	4
2.2.1. Rules	4
2.2.2. Business Value	5
2.3. Most Viewed Processor	5
2.3.1. Rules	5
2.3.2. Business Value	5
2.3.3. Relationship with Continuous View Processor	5
3. Discount Conditions	7
3.1. Continuous View Discount	7
3.1.1. Viewing Time Calculation	7
3.2. Most Viewed Product Discount	7
3.2.1. Basic Rules	7
3.2.2. Detailed Specifications	7
Time Window	7
Event Flow	8
View Counting	8
Discount Generation	8
Tiebreaker Rules	8
4. Event examples	9
4.1. Input (in <code>snowplow-enriched-good</code> topic)	9
4.2. Output (in <code>shopper-discounts</code> topic)	9
5. Running the application	11
5.1. Prerequisites	11
5.2. Version Information	11
5.3. Running manually	11
5.3.1. Starting Redpanda	11
5.3.2. Running the application	11
5.3.3. Generating events	12
5.3.4. Verifying discounts	13

5.4. Running via Docker Compose	13
6. Testing procedures.	14
6.1. Unit tests	14
6.2. Mock Tests	14
7. Discount processors.	15
8. MostViewed Processor Testing Results Comparator	16
9. Parameters used in generating test events	17
10. Troubleshooting	18
10.1. Common Issues	18
10.2. Logs	18
11. Developer Guide.	19
11.1. Developer Tips	19
11.1.1. Gradle Commands	19
Skipping Tests	19
Code Formatting	19
Running Specific Tests	19
11.1.2. IDE Configuration	19
IntelliJ IDEA	19
VS Code	19
11.1.3. Documentation	20
README Generation	20
11.1.4. Docker Tips.	20
Clean Environment.	20
Quick Redpanda Access.	20
11.2. Troubleshooting	20

1. Overview

This project (Discounts Processor (Kafka Streams version)) is a Java/ [Kafka Streams](#) application that processes event data from [Redpanda](#) and generates [discount events](#) based on some [conditions](#). So, based on the [input events](#), it analyzes if the user is able to get discounts.

! PROJECT STATUS: UNDER DEVELOPMENT "

Status:

1. [Continuous View Discount Processor](#): Working well.
2. [Most Viewed Product Discount Processor](#): Under development/ test.

This module can be treated almost^[1] entirely independently of [the main project](#) that encompasses it. It is an alternative intended to produce the same result as the [connect.yaml](#) file configured using [Redpanda Connect](#). In other words, it is a discount processor that works under the same [rules](#).

The tests in this module are [unit tests](#) and [integration tests](#). Some of them are written in [Kotlin](#) following the [BehaviourSpec](#) provided by [Kotest](#) framework.

The [integration tests](#) developed are executed with the help of [TestContainers](#).

2. Processors Rules

This section describes the rules and business logic implemented by the two discount processors.

2.1. Processor Configuration

The `DiscountsProcessor` provides flexible configuration options for both processors:

- ¥ Both processors can be enabled simultaneously
- ¥ Either processor can be disabled independently
- ¥ At least one processor must be enabled for the system to function
- ¥ Configuration is managed through the `ConfigurationManager`

This flexibility allows for different business scenarios:

- ¥ Running both processors together provides the most comprehensive discount strategy
- ¥ Running only the `MostViewedProcessor` can be useful when focusing on product discovery behavior
- ¥ Running only the `ContinuousViewProcessor` emphasizes deep product engagement

When running only the `MostViewedProcessor`, it's recommended to:

- ¥ Adjust the `processor.continuous-view.min-pings-for-discount` property to a higher value
- ¥ This ensures that discounts are only given for products with significant user engagement
- ¥ Helps maintain the quality of the discount program while focusing on the most viewed products

2.2. Continuous View Processor

The `ContinuousViewProcessor` implements the primary discount strategy, which rewards users for continuous engagement with a single product.

2.2.1. Rules

- ¥ A 10% discount is generated when a user views the same product continuously for 90 seconds
- ¥ The viewing time is calculated as:
 - ¥ Initial viewing time (when `product_view` occurs)
 - ¥ 10-second delay to first ping
 - ¥ Each subsequent `page_ping` adds 10 seconds
- ¥ Only one discount per user per 5-minute window
- ¥ Cooldown period of 5 minutes after receiving any discount

2.2.2. Business Value

This processor encourages deep product engagement by rewarding users who spend significant time examining a single product. This behavior often indicates strong purchase intent and can lead to higher conversion rates.

2.3. Most Viewed Processor

The `MostViewedProcessor` serves as a fallback mechanism when the continuous view criteria aren't met, but the user still shows significant interest in multiple products.

2.3.1. Rules

- ¥ A 10% discount is generated for the most viewed product within a 5-minute window
- ¥ View counting:
 - ! Minimum threshold: 3 views within the window
 - ! Only `page_view` events count towards view totals
 - ! View duration is calculated using `page_view` timestamps
- ¥ Tiebreaker rules:
 - ! Product with longest total viewing time wins
 - ! If still tied, most recently viewed product wins
- ¥ One discount per user per window
- ¥ 5-minute cooldown period after receiving any discount

2.3.2. Business Value

This processor serves as a valuable fallback for several reasons:

1. It captures users who browse multiple products but don't spend enough time on any single one to trigger the continuous view discount
2. It rewards product discovery behavior, encouraging users to explore the catalog
3. It provides a second chance for discounts when the primary continuous view strategy doesn't apply
4. It helps maintain user engagement by offering alternative paths to discounts

2.3.3. Relationship with Continuous View Processor

The `MostViewedProcessor` complements the `ContinuousViewProcessor` by:

- ¥ Providing an alternative discount path for different user behaviors
- ¥ Ensuring users who show interest in multiple products can still receive discounts
- ¥ Creating a more comprehensive discount strategy that covers various shopping patterns
- ¥ Maintaining user engagement through multiple interaction patterns

This dual-processor approach creates a more robust and flexible discount system that can accommodate different user behaviors while maintaining clear business rules and objectives.

3. Discount Conditions

3.1. Continuous View Discount

A single discount of 10% (within a 5-minute window) is generated when:

¥ A user views the same product continuously within a 90-second window.

3.1.1. Viewing Time Calculation

The total viewing time includes:

1. Initial viewing time: When a `product_view` event occurs, it starts the viewing session
2. Delay to first ping: A configured time (default: 10 seconds) between the `product_view` and the first `page_ping`
3. Subsequent pings: Regular intervals between `page_ping` events (default: 10 seconds)

For example, to achieve a 90-second continuous view:

¥ Initial `product_view` event
¥ 10 seconds delay to first ping
¥ 8 subsequent `page_ping` events at 10-second intervals
¥ Total time = 10 (delay) + (8 * 10) = 90 seconds

This means that with default settings (10-second ping interval and 10-second initial delay), only 8 `page_ping` events are needed to achieve a 90-second continuous view, instead of 9 pings that would be needed without considering the initial viewing time.

!

The formula to calculate the total viewing time is: $total_time = delay_to_first_ping + (number_of_pings * ping_interval)$

3.2. Most Viewed Product Discount

3.2.1. Basic Rules

For each user, a 10% discount is generated for their most viewed product within a fixed 5-minute window.

3.2.2. Detailed Specifications

Time Window

¥ Fixed 5-minute windows (00:00-00:05, 00:05-00:10, etc.)
¥ Discount evaluation occurs at the end of each window
¥ Events are processed based on their `collector_timestamp`

Event Flow

1. A `product_view` event initiates a viewing session
2. Subsequent `page_pi ng` events for that product are counted
3. Session continues until another `product_view` event occurs
4. Process repeats for the new product

View Counting

- ¥ Minimum threshold: 3 views within the window
- ¥ Only `page_pi ng` events count towards view totals
- ¥ View duration is calculated using `page_pi ng` timestamps

Discount Generation

- ¥ Scope: Per user
- ¥ Generated at window end
- ¥ One discount per user per window
- ¥ Cooldown: 5-minute period per user after receiving any discount

Tiebreaker Rules

If a user has multiple products with the same number of views: 1. Product with longest total viewing time wins 2. If still tied, first product viewed wins

Table 1. Example Timeline (for user "U1")

Time	Event	Effect
10:00:00	Window starts	
10:00:10	<code>product_view</code> P1	Start tracking P1
10:00:15-00:45	4 <code>page_pi ng</code> P1	P1: 4 views
10:01:00	<code>product_view</code> P2	Start tracking P2
10:01:05-01:50	6 <code>page_pi ng</code> P2	P2: 6 views
10:02:00	<code>product_view</code> P1	Start tracking P1 again
10:02:05-02:30	3 <code>page_pi ng</code> P1	P1: total 7 views
10:05:00	Window ends	P1 wins (7 views)

4. Event examples

4.1. Input (in snowplow-enriched-good topic)

product_view event:

```
{
  "collector_timestamp": "2025-04-04T07:05:00.119Z",
  "event_name": "product_view",
  "user_id": "1",
  "product_id": "5",
  "product_name": "SP Flex Runner 2",
  "product_price": 42.99,
  "webpage_id": "page_5"
}
```

page_ping event:

```
{
  "collector_timestamp": "2025-04-04T07:05:12.130Z",
  "event_name": "page_ping",
  "user_id": "1",
  "webpage_id": "page_5"
}
```

4.2. Output (in shopper-discounts topic)

discount event examples:

Continuous View Discount:

```
{
  "user_id": "1",
  "product_id": "5",
  "discount": {
    "rate": 0.1,
    "by_view_time": {
      "duration_in_seconds": 100
    }
  }
}
```

Most Viewed Product Discount:

```
{
```

```
Ê "user_id": "1",  
Ê "product_id": "5",  
Ê "discount": {  
Ê   "rate": 0.1,  
Ê   "by_number_of_views": {  
Ê     "views": 5,  
Ê     "duration_in_seconds": 30  
Ê   }  
Ê }  
}
```

5. Running the application

5.1. Prerequisites

1. Bash
2. Java
3. [Docker](#)

5.2. Version Information

These were the versions used during development:

```
$ echo $BASH_VERSION
5.2.21(1)-release

$ java --version
openjdk 21.0.6 2025-01-21 LTS
OpenJDK Runtime Environment Temurin-21.0.6+7 (build 21.0.6+7-LTS)
OpenJDK 64-Bit Server VM Temurin-21.0.6+7 (build 21.0.6+7-LTS, mixed mode, sharing)

$ docker --version
Docker version 27.3.1, build ce12230
```

!

[Kafka Streams](#) is fully supported by Java 21. See this page: <https://kafka.apache.org/documentation/#java>

5.3. Running manually

5.3.1. Starting Redpanda

First, you need to start Redpanda using `docker compose`:

```
$ docker compose -f compose.redpanda.yaml up -d # <- same effect as ./redpanda.sh up
```

!

When you're done, use this command to stop Redpanda and remove all data:

```
$ docker compose -f compose.redpanda.yaml down -v --remove-orphans # <-
same effect as ./redpanda.sh down
```

5.3.2. Running the application

The application can be executed using the provided [run.sh](#) script, which handles building and

running the application:

Start the application with an existing JAR:

```
$ ./run.sh
```

!

You can force a rebuild before running the application:

```
$ ./run.sh --build
```

The `run.sh` script will:

- ¥ Build the application if the JAR doesn't exist or if `--build` is specified
- ¥ Configure appropriate JVM options
- ¥ Handle graceful shutdown on SIGTERM/SIGINT
- ¥ Display colored status messages during execution

!

The script automatically manages the application lifecycle and provides proper cleanup on shutdown.

5.3.3. Generating events

To generate events in order to trigger discounts, use [the Simulator project](#). You can use it to generate events (containing `product_view` and `page_ping` events in JSONL format) and send them to Redpanda.

So, the file [data-samples/long.jsonl](#) is an example of such a file. You can send the events in this file to the input topic using the following command:

```
$ docker exec -i redpanda rpk topic produce snowplow-enriched-good < \
  Ê data-samples/long.jsonl # <- same effect as ./redpanda.sh produce
```

!

1. The command above is to produce test events for testing the [Continuous View Discount Processor](#). To test the [Most Viewed Product Discount Processor](#), use the command `data_type=frequent ./redpanda.sh produce`.
2. Using the `tstamp-diff.sh` script (available in main project) you can verify that the time difference between the collector timestamps for the `page_ping` events in this file is greater than 90 seconds as per the rule. Therefore, these `page_ping` events on it should generate a discount event according to [the rule](#).

```
$ {
  Ê f=data-samples/long.jsonl
  Ê start=$(sed -n 2p $f | jq -r .collector_timestamp)
```

```
Ê end=$(sed -n 11p $f | jq -r .collector_timestamp)
Ê ../scripts/tstamp-diff.sh $start $end
}
90.075
```

5.3.4. Verifying discounts

To verify that discounts are being generated correctly, consume from the output topic:

```
$ docker exec -it redpanda rpk topic consume shopper-discounts # <- same effect as
./redpanda.sh consume
```

You should see discount events in the output that match the expected patterns based on the input events.

!

You can also open the Redpanda console at <http://localhost:8080> to observe each of the events produced in the shopper-discounts topic.

5.4. Running via Docker Compose

To run the application, type:

```
$ docker compose up --build -d
```

Watch the logs to ensure the application is running correctly:

```
$ docker compose logs discounts-processor -f
```

Generate events as described in [Generating events](#).

Verify discounts as described in [Verifying discounts](#).

To stop the application and remove the containers, type:

```
$ docker compose down -v --remove-orphans
```

6. Testing procedures

6.1. Unit tests

Unit tests verify the core business logic of the discount processors without external dependencies.

To run all unit and mock tests:

```
$ ./gradlew test
```

!

Since all tests are mocked, there is no need to start Redpanda.

6.2. Mock Tests

These are the two mock tests created for the [discount processors](#):

¥ [src/test/kotlin/com/example/processor/ContinuousViewProcessorTest.kt::](#)

¥ [src/test/kotlin/com/example/processor/MostViewedProcessorTest.kt::](#)

To run a specific test class (e.g., [DiscountEventSerdeTest](#)):

```
$ ./gradlew test --tests "com.example.serialization.DiscountEventSerdeTest"
```

To run a mocked test for one of the two [discount processors](#):

```
$ ./gradlew test --tests "com.example.processor.MostViewedProcessorTest"
```

7. Discount processors

This project contains two discount processors:

[src/main/java/com/example/processor/ContinuousViewProcessor.java](#)

Cover all the rules for the [Continuous View Discount](#).

Test code: [ContinuousViewProcessorTest](#)

[src/main/java/com/example/processor/MostViewedProcessor.java](#)

Cover all the rules for the [Most Viewed Product Discount](#).

Test code: [MostViewedProcessorTest](#)

8. MostViewed Processor Testing Results Comparator

```
$ ./scripts/MostViewed.summary.compare.sh # <- it will compare the results where  
testName=MostViewedSingleViewPerProduct
```

```
$ testName=MostViewedMultipleViewsPerProduct ./scripts/MostViewed.summary.compare.sh
```


9. Parameters used in generating test events

```
$ ./e2e/run.sh -t MostViewedMultipleViewsPerProduct -j --maxProducts 5 --maxPings 4
```

10. Troubleshooting

10.1. Common Issues

- ¥ Connection issues: Verify that you [started Redpanda](#).
- ¥ No discounts generated: Ensure that enough events are being sent to trigger the [discount conditions](#).
- ¥ Serialization errors: Check that the event format matches [the expected schema](#).

10.2. Logs

Application logs can be configured in two files depending on the environment (main or test):

1. [logback.xml](#)
2. [logback-test.xml](#)

11. Developer Guide

11.1. Developer Tips

11.1.1. Gradle Commands

Skipping Tests

To skip test execution during build:

```
$ ./gradlew build -x test
```

Code Formatting

The project uses Spotless for code formatting. Common commands:

```
$ ./gradlew spotlessCheck    # Check code formatting
$ ./gradlew spotlessApply    # Apply code formatting
```

!

To automatically format code before each build, the project already includes `spotlessApply` as a build dependency.

Running Specific Tests

To run a specific test class:

```
$ ./gradlew test --tests "com.example.processor.MostViewedProcessorTest"
```

To run only integration tests:

```
$ ./gradlew integrationTest
```

11.1.2. IDE Configuration

IntelliJ IDEA

TODO

VS Code

¥ Install "[EditorConfig](#)"

! Configure "Editor: Format On Save" setting

¥ Install "[Java extensions for Visual Studio Code](#)"

! Install "[SonarQube for IDE](#)"

! Install "[Gradle for Java](#)"

¥ Install "[Kotlin Language](#)"

¥ Install "[AsciiDoc](#)"

11.1.3. Documentation

README Generation

To rebuild the documentation:

```
$ ./README.sh    # Generates README.html and updates other formats
```

11.1.4. Docker Tips

Clean Environment

To completely clean up Docker resources:

```
$ docker compose down -v --remove-orphans
$ docker system prune -f    # Remove unused containers/images
```

Quick Redpanda Access

The project includes convenient scripts for Redpanda operations:

```
$ ./redpanda.sh up    # Start Redpanda
$ ./redpanda.sh down  # Stop Redpanda
```

11.2. Troubleshooting

Sometimes, you may need to clean up Java processes to start fresh. If you have strange issues while testing the application, like in the following output:

```
$ run
Starting Discounts Processor...
Console output will be logged to: private/logs/run.log
10:08:18.621 Loading configuration from application.properties...
10:08:18.623 Configuration loaded:
10:08:18.626   Enabled processors: ContinuousView
10:08:18.628   Continuous view implementation: 2
10:08:18.628   Bootstrap servers: localhost:19092
10:08:18.628   Input topic: snowplow-enriched-good
10:08:18.628   Output topic: shopper-discounts
```

```

10:08:18.628 Group ID: discounts-processor
10:08:18.628 Auto offset reset: earliest
10:08:18.629 Window duration: 300s
10:08:18.629 Ping interval: 10s
10:08:18.629 Minimum pings: 8
10:08:18.629 Discount rate: 0.1
10:08:18.630 Initializing Discounts Processor...
10:08:18.739 Configuring ContinuousViewProcessor (implementation 2)...
Exception in thread "main" org.apache.kafka.streams.errors.StreamsException: Unable to
initialize state, this can happen if multiple instances of Kafka Streams are running
in the same state directory (current state directory is [/tmp/kafka-streams/discounts-
processor])
    at
org.apache.kafka.streams.processor.internals.StateDirectory.initializeProcessId(StateD
irectory.java:191)
    at org.apache.kafka.streams.KafkaStreams.<init>(KafkaStreams.java:879)
    at org.apache.kafka.streams.KafkaStreams.<init>(KafkaStreams.java:862)
    at org.apache.kafka.streams.KafkaStreams.<init>(KafkaStreams.java:832)
    at org.apache.kafka.streams.KafkaStreams.<init>(KafkaStreams.java:744)
    at com.example.DiscountsProcessor.start(DiscountsProcessor.java:160)
    at com.example.DiscountsProcessor.main(DiscountsProcessor.java:184)
Application terminated successfully

```

Run these commands to clean up:

```

$ sudo kill java
$ rm -rf /tmp/kafka-streams/discounts-processor/

```

[1] It only relies on scripts that generate documentation.