

Discounts Processor

Git commit with doc: 3214b38

[HTML version](#)

| | |
|---|----|
| 1. Overview | 2 |
| 2. Discount Conditions | 3 |
| 2.1. Continuous View Discount | 3 |
| 2.2. Most Viewed Product Discount | 3 |
| 2.2.1. Basic Rules | 3 |
| 2.2.2. Detailed Specifications | 3 |
| Time Window | 3 |
| Event Flow | 3 |
| View Counting | 3 |
| Discount Generation | 3 |
| Tiebreaker Rules | 4 |
| 3. Event examples | 5 |
| 3.1. Input (in <code>snowplow-enriched-good</code> topic) | 5 |
| 3.2. Output (in <code>shopper-discounts</code> topic) | 5 |
| 4. Running the Application | 7 |
| 4.1. Prerequisites | 7 |
| 4.2. Version Information | 7 |
| 4.3. Running manually | 7 |
| 4.3.1. Starting Redpanda | 7 |
| 4.3.2. Running the application | 8 |
| 4.3.3. Generating events | 8 |
| 4.3.4. Verifying discounts | 8 |
| 4.4. Running via Docker Compose | 8 |
| 5. Testing Procedures | 10 |
| 5.1. Unit and Mock Tests | 10 |
| 5.2. Integration Tests | 10 |
| 6. Discount processors | 10 |
| 7. Troubleshooting | 11 |
| 7.1. Common Issues | 11 |
| 7.2. Logs | 11 |

1. Overview

This project (**Discounts Processor**) is a Java/ [Apache Flink](#) 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.

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. Discount Conditions

2.1. 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.

2.2. Most Viewed Product Discount

2.2.1. Basic Rules

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

2.2.2. 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_tstamp`

Event Flow

1. A `product_view` event initiates a viewing session
2. Subsequent `page_ping` 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_ping` events count towards view totals
- View duration is calculated using `page_ping` 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 | product_view P1 | Start tracking P1 |
| 10:00:15-00:45 | 4 page_ping P1 | P1: 4 views |
| 10:01:00 | product_view P2 | Start tracking P2 |
| 10:01:05-01:50 | 6 page_ping P2 | P2: 6 views |
| 10:02:00 | product_view P1 | Start tracking P1 again |
| 10:02:05-02:30 | 3 page_ping P1 | P1: total 7 views |
| 10:05:00 | Window ends | P1 wins (7 views) |

3. Event examples

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

product_view event:

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

3.2. 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:

```
{
```

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

4. Running the Application

4.1. Prerequisites

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

4.2. 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
```



The first [Apache Flink](#) version supporting Java 21 is 1.19.2 (see its [Release Notes](#)). Currently, the Flink's version used by this project can be found by this command:

```
$ sed -n 's/val flinkVersion = "(.*)"/\1/p' build.gradle.kts
1.20.1
```

4.3. Running manually

4.3.1. Starting Redpanda

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

```
$ docker compose -f compose.redpanda.yaml up -d
```



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

```
$ docker compose -f compose.redpanda.yaml down -v --remove-orphans
```

4.3.2. 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.

4.3.3. Generating events

To generate events in order to trigger discounts, use [the Simulator project](#).

4.3.4. Verifying discounts

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

```
$ docker exec -it redpanda rpk topic consume shopper-discounts
```

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

4.4. 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
```

5. Testing Procedures

5.1. Unit and Mock 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.

To run a specific test class (e.g., `DiscountEventSerializationSchemaTest`):

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

To run a mocked test for one of the two `discount processors`:

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

5.2. Integration Tests

The integration tests cover tests for the `discount processors` and is running with `TestContainers`.

```
$ ./gradlew test --tests "com.example.DiscountsProcessorIntegrationTest"
```

To run all integration tests:

```
$ ./gradlew integrationTest
```

6. 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: `src/test/kotlin/com/example/processor/ContinuousViewProcessorTest.kt`

`src/main/java/com/example/processor/MostViewedProcessor.java`

Cover all the rules for the `Most Viewed Product Discount`.

7. Troubleshooting

7.1. 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](#).

7.2. Logs

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

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

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