Dynamic Ecommerce Discounts with Redpanda

Author(s): Paulo Jerônimo

HTML version
Last updated: 2025-05-04 19:03:04 -0300
Last commit: 78b7216

1. Introduction

1.1. The busisess value of this solution

This solution allows Snowplow users to learn how their components can be integrated with e-commerce solutions to offer discounts dynamically when detecting an hesitant shopper.

A hesitant shopper is one who has shown a certain amount of interest in a given product without adding it to cart.

This **solution acclerator** implements two complementary ways of offering discounts.

1. The first way to get a discount on a product is to **view it for more than 90 seconds in a 5-minute window**.
    a. In the Architecture section, the [ContinuousViewProcessor] implements this feature.
2. The second way to get a discount on a product is to **view it more than five times in the same 5-minute window**.
    a. In the Architecture section, the [MostViewedProcessor] implements this feature.

After the discount event is generated, one of the situations that can occur in the solution implemented by this accelerator is that it emits the discount event back to the Snowplow pipeline. In the Architecture, this is done by the [DiscountEventSender] component.

In addition to offering this possibility, the processors of this implementation already record the discount event in a Redpanda topic, allowing any consumer with access to this topic to consume it.

When a discount event is available in the Snowplow pipleline, it can be consumed in some other ways, including:

1. From the data warehouse, use the Census Dataset API to make the discount available to the front end application
2. Using Snowbridge to send the discount to Braze for live couponing.

Detecting high-interest products without purchases is a popular real-time use case for e-commerce.

Although the above definitions seem simple, **implementing dynamic discounts is not trivial (discounts-processor-doc)**.

1.2. Technologies used

This project is a companion to the Example Next.js Ecommerce Store for Snowplow.

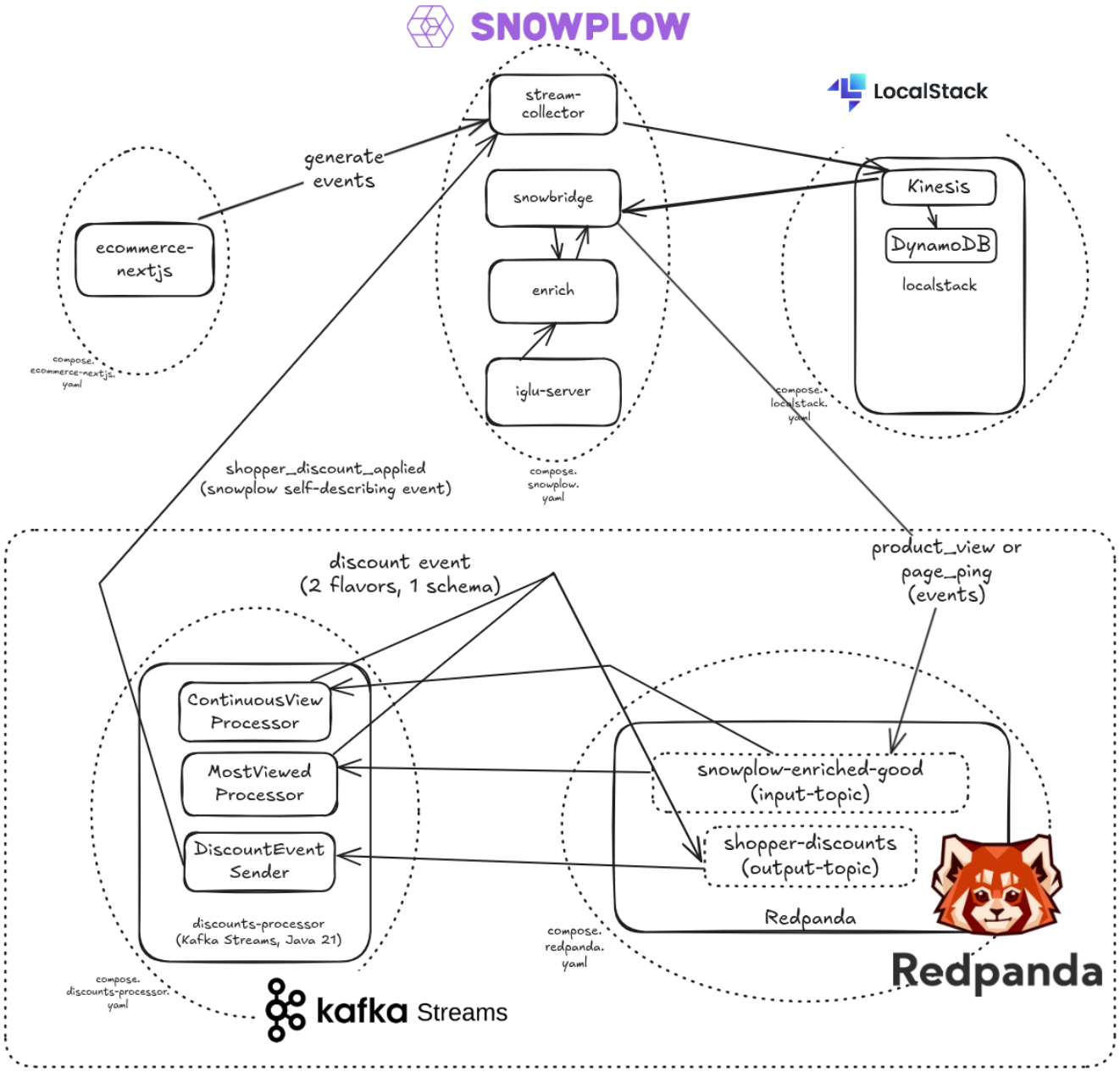
It allows you to test this it locally, using LocalStack, ~~in the AWS's cloud (via Terraform)~~ or Development Containers (read [localstack-devcontainers]).

> The development of this project can be done entirely in the cloud because it uses Development Containers in GitHub Codespaces. This way, you only need a browser to get started!
>
> This solution was designed so developers can quickly and easily set up their environment to create new features or test existing ones.
>
> **Watch the [introduction-video] for details.**

What this project does to complement the existence of [ecommerce-nextjs-example-store] is to create the entire infrastructure that allows it to be executed via Docker Compose, or DevContainer + Docker Compose and, ultimately, meet the business requirements.

So, this solution integrates LocalStack with Snowplow, Redpanda and implements the processors using Kafka Streams.

2. Architecture



- The **ecommerce-nextjs-example-store** is a Next.js application that generates tracking events.
- The **stream-collector** component sends these events via Kinesis to the **[snowbridge]** component.
- The **snowbridge** component enriches these events, inserts more information (via **[enrich]** component), and sends them to [redpanda].
  - Read more about the **enrich** component here: https://docs.snowplow.io/docs/pipeline-components-and-applications/enrichment-components/enrich-kinesis/.
  - Read more about the **snowbridge** component here: https://docs.snowplow.io/docs/destinations/forwarding-events/snowbridge/.
- The **redpanda** broker is receives the events from [snowbridge].
- For the following components, read details in the [discounts-processor-doc]:

    - The **ContinuousViewProcessor** component implements the first processor.
    - The **MostViewedProcessor** component implements the second processor.
    - The **DiscountEventSender** component implements the way to send an event back to Snowplow.
        - Please, also read the additional [schemas-doc].

All components in this Architecture run as Docker containers via `docker compose`:

- The infrastructure to provide the AWS resources locally (Kinesis, DyanmoDB, etc) is created by LocalStack, in the file `compose.localstack.yaml`.
- The Snowplow's components (**[stream-collector]**, **[enrich]**, **[snowbridge]** **and** **[iglu-server]**) are defined in the file `compose.snowplow.yaml`.
- Redpanda's infrastructure is provided by the file `compose.redpanda.yaml`.
- The fron end application (**[ecommerce-nextjs-example-store]**) is defined in the file `compose.ecommerce-nextjs.yaml`.
- The **[ContinuousViewProcessor]**, **[MostViewedProcessor]** and **[DiscountEventSender]** components executed by Kafka Streams which executes a Java application running by Docker through the configuration in the file `compose.discounts-processor.yaml`.

3. Prerequisites

3.1. Running with Docker

1. Make sure you have this tools installed:
   a. Bash (version 5.1 or higher). You will run it in a terminal on your macOS, Linux or Windows (in a WSL2 environment).
   b. Git.
   c. Docker (with `docker compose` support).
2. Even though you can run most of the demonstrations for this project using Docker Compose (see docker-doc), in a test environment, in development, or to generate the documentation you are reading for this project, you will need to have a few more tools installed. To run some scripts provided by this project you will need some additional tools:
   a. jq, yq
   b. docker-asciidoctor-builder
3. To create a full development environment, you will also need to install:
   i. Java 21
   ii. Node.js 18
   iii. Python 3.12

So, here is **the best tip to run this project**: its much more convenient to use the Development Containers to create a development environment. So, check Running with DevContainers.

1. Clone this project with Git and cd to it.
2. Execute this command (← **misc-setup**):

```
$ source scripts/misc/setup.sh
```

3. Create a file `docker/.env` (from `docker/.env.sample`) and configure the AWS variables on it.

You don't need Java or Node.js configured on your machine to follow the steps below. **You only need a Bash terminal, a Docker installation, and some required tools.**

3.2. Running with DevContainers

Watch the video [introduction-video].

3.2.1. Inside GitHub Codespaces

Watch the video [introduction-video].

3.2.2. Locally, inside VSCode

Watch the video [introduction-video].

3.2.3. Locally, With DevContainer CLI

Step 1 → Install the `devcontainer` command:

```
$ npm install -g @devcontainers/cli
```

Step 2 → Start the dev container:

```
$ # cd TO_THE_PROJECT_FOLDER (the directory where you clone this project)
$ devcontainer up --workspace-folder .
```

Step 3 → Open a Bash terminal in the container:

```
$ devcontainer exec --workspace-folder . bash
```

Please, see more details about how Development Containers is configured in this project by viewing its README file (devcontainer-doc).

4. Steps (to run this application as is)

Step 1 → Start the containers

```
$ up.sh
```

Tips:

1. You can press `Ctrl` + `C` at any time. The docker containers will remain running.
2. If there is no file `docker/.env` in the project, this script will try to locate it in a file named `../dynamic-ecommerce-discounts-with-redpanda.env` and copy it to `docker/.env`. This allows you to call `git clean -fdX` at any time you want without losing your configuration.
   a. If the file `../dynamic-ecommerce-discounts-with-redpanda.env` does not exists, it will copy the file `docker/.env.sample` to `docker/.env` and use it.
3. You can pass "services" as an argument option to this script. It will list the options you can pass to it by adding the suffix "-services":

```
$ up.sh services
apps
discounts-processor
ecommerce-nextjs
localstack
redpanda
snowplow
```

4. By adding the "-services" to one of the options listed above, you will start only the services listed in the file `copose.<service>.yaml`. So, this will start only the redpanda services (services listed in `compose.redpanda.yaml`):

```
$ up.sh redpanda-services
```

5. You can also call the script `up.sh` by using the `compose.sh` script this way:

```
$ compose.sh up
```

6. Finally, ***if you can don't want to use this script, you can change your current directory to `docker` and use the `docker` commands you alredy know***. Check this README file.

Step 2 → Know the URL provided by the services

1. **LocalStack**: https://app.localstack.cloud ← localstack
2. **Redpanda**:
   a. **Internal (docker containers access)** http://localhost:9092 ← redpanda-internal
   b. **Console**: http://localhost:8080 ← redpanda-console
      i. User / password: jane / some-other-secret-password
3. **Ecommerce store**: http://localhost:3000 ← ecommerce-store
   a. It connects with **Snowplow collector** configured to run in http://localhost:9090 ← snowplow-collector

Step 3 → Browse the application pages

As expected, in the [ecommerce-store], during every page navigation, we are tracking a page view event.

For ecommerce interactions we track the following:

- When a customer goes to a product page we track a product view event.
- When a customer sees an internal promotion list, e.g. Homepage promotions, we track an internal promotion view event.
- When a customer clicks an internal promotion, we track an internal promotion click event.
- When a customer goes to a product list page, we track a product list view event.
- When a customer clicks a product on a product list page, we track a product list click event.
- When a customer sees a recommended product list on the product page, we track a product list view event.
- When a customer clicks on a recommended product list on the product page, we track a product list click event.
- When the customer adds a product to the cart, we track an add to cart event.
- When the customer goes to the cart page we track a checkout step event.
- When they go to the payment step, another checkout step event is tracked.
- When the customer successfully completes a transaction, we track a transaction event (triggered on the server-side but formulated with the spec of Snowplow ecommerce)

Step 4 → Access the redpanda-console and check the generated events

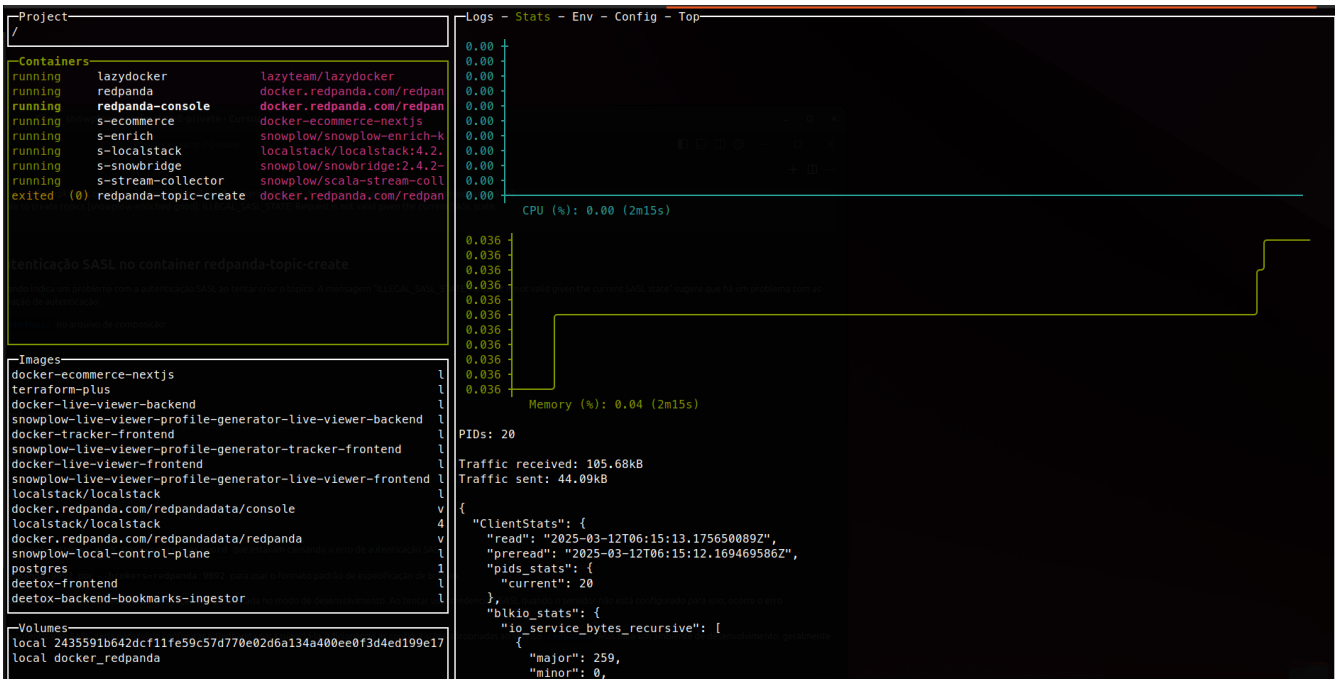
After browsing the , you can access the and check the generated events in the topic `snowplow-enriched-good`. See these images: redpanda-1.png, redpanda-2.png.

You can explore the data format of these events in enriched TSV format. In the scripts/raw-

messages.sample directory, there are examples of the events recorded by Snowplow when they are transferred to Redpanda. These sample TSV files were created by running the script extract-snowplow-raw-messages.sh. Note: *these are not the events final format of the events sent from [snowbridge] component to [redpanda].*

Step N → (optional) Use LazyDocker to monitor the containers and logs

```
$ lazy.sh
```



5. Clean up steps

Step 1 → Stop the containers

To stop all the containers, type:

```
$ ./docker/down.sh
```

Step 2 → Clean up

To remove all the containers and images, type:

```
$ ./docker/clean.sh
```

Warnings:

1. The script `clean.sh` will destroy any data generated by these containers.

6. References

LocalStack

- https://docs.localstack.cloud/
- https://docs.localstack.cloud/user-guide/integrations/devcontainers/ ← **localstack-devcontainers**

Redpanda

- Docker Compose Labs
  - Start a Single Redpanda Broker with Redpanda Console in Docker
- Redpanda Self-Managed Quickstart
- How we engineered our CLI to improve developer productivity
- Some YouTube videos:
  - Why did Redpanda rewrite Apache Kafka? (with Christina Lin)
  - Redpanda Office Hour: HUGE rpk - Redpanda CLI update!

Redpanda Connect

- https://docs.redpanda.com/redpanda-connect/get-started/quickstarts/rpk/
- https://docs.redpanda.com/current/get-started/quick-start/

7. Additional docs

The documentation for this project is designed to be modular, just like all of its code. Therefore, this topic serves as a summary to help you find these additional documents. You can always run a `find . -name README.sh` command from the project root to locate all the scripts that generates additional documents.

Anyway, here are links to the additional documents:

- **discounts-processor-doc**: Discounts Processor Implementation
- **docker-doc**: Some docker commands used in this project
- **devcontainer-doc**: DevContainers Configuration
- **schema-doc**: Schema implementation and testing

8. Demo videos

- Getting started by creating a development environment on GitHub Codespaces ← **introduction-video**

9. Next steps

The following steps describe features or activities that could not be delivered when the demo videos were created.

So, here are some tasks that can be developed for the subsequent releases.

9.1. Fix some known bugs and do some refactoring

1. One of the bugs is related to the ../../discounts-processor/e2e/test/MostViewedSingleViewPerProduct.js, which currently does not pass.
2. Regarding refactoring, one of them concerns the processor code that contains duplicated logic. This can undoubtedly be improved.

9.2. Development of a discounts processor version using Redpanda Connect

1. We tried to implement a solution faster using Bloblang. However, besides forcing a learning curve, testing the solutions with the expected requirements was not easy. Therefore, we abandoned this solution until we had a clearer vision of what we wanted to implement.

9.3. Deployment via Terraform in a cloud, such as AWS (for example)

1. The first accelerator developed in partnership with OSO demonstrates this.

9.4. Presentation of the discount event directly on the front end

1. Due to the time spent on the backend and the difficulty in implementing the discount processors, we did not have enough time to complete the development of a visualisation of discount events on the frontend. So this is a good task to be developed.