Schema implementation and testing

Author(s): Paulo Jerônimo

HTML version
Last updated: 2025-05-05 01:54:19 -0300
Last commit: 98860e2

1. Introduction

Here you have details about the schema for the generated discount event.

As implemented by the ../discounts-processor, there are two possible discount formats.

A **Discount Event** can be exemplified by these two examples:

../discounts-processor/e2e/data/MultiProduct.json:

```
{
  "discount": {
    "rate": 0.1,
    "by_view_time": {
      "duration_in_seconds": 130
    }
  },
  "user_id": "user1",
  "product_id": "product3",
  "generated_at": "2025-04-26T19:04:37.949Z"
}
```

../discounts-processor/e2e/data/MostViewedMultipleViewsPerProduct.json:

```
{
  "discount": {
    "rate": 0.1,
    "by_number_of_views": {
      "views": 5,
      "duration_in_seconds": 130
    }
```

```
  },
  "user_id": "user1",
  "product_id": "product1",
  "generated_at": "2025-05-02T01:50:24.813Z"
}
```

So, the Custom schema implementation must be able to attend both formats.

2. Custom schema implementation

The schema for for the events above is definined below.

./com.snowplow/shopper_discount_applied/jsonschema/1-0-0:

```
{
  "$schema": "http://iglucentral.com/schemas/com.snowplowanalytics.self-
desc/schema/jsonschema/1-0-0#",
  "description": "Schema for tracking shopper discounts based on user behavior",
  "self": {
    "vendor": "com.snowplow",
    "name": "shopper_discount_applied",
    "format": "jsonschema",
    "version": "1-0-0"
  },
  "type": "object",
  "properties": {
    "discount": {
      "type": "object",
      "description": "Discount configuration and trigger conditions",
      "properties": {
        "rate": {
          "type": "number",
          "minimum": 0,
          "maximum": 1,
          "description": "Discount rate as decimal (e.g., 0.1 for 10%)"
        },
        "by_view_time": {
          "type": "object",
          "description": "Discount triggered by viewing duration",
          "properties": {
            "duration_in_seconds": {
              "type": "number",
              "minimum": 0,
              "description": "Duration in seconds that triggered the discount"
            }
          },
          "required": [
            "duration_in_seconds"
          ],
          "additionalProperties": false
```

```
        },
        "by_number_of_views": {
          "type": "object",
          "description": "Discount triggered by number of views",
          "properties": {
            "views": {
              "type": "number",
              "minimum": 1,
              "description": "Number of views that triggered the discount"
            },
            "duration_in_seconds": {
              "type": "number",
              "minimum": 0,
              "description": "Duration in seconds that triggered the discount"
            }
          },
          "required": [
            "views",
            "duration_in_seconds"
          ],
          "additionalProperties": false
        }
      },
      "required": [
        "rate"
      ],
      "oneOf": [
        {
          "required": [
            "by_view_time"
          ]
        },
        {
          "required": [
            "by_number_of_views"
          ]
        }
      ],
      "additionalProperties": false
    },
    "user_id": {
      "type": "string",
      "minLength": 1,
      "description": "The ID of the user who received the discount"
    },
    "product_id": {
      "type": "string",
      "minLength": 1,
      "description": "The product SKU that received the discount"
    },
    "generated_at": {
```

```
      "type": "string",
      "format": "date-time",
      "description": "ISO 8601 timestamp when the discount was generated"
    }
  },
  "required": [
    "discount",
    "user_id",
    "product_id",
    "generated_at"
  ],
  "additionalProperties": false
}
```

3. Registering the custom schema in Iglu Server

The script ./register.sh is responsible for registering the custom discount event schema in the Iglu Server. It uses a simple `curl` command to POST the schema file (located at `com.snowplow/shopper_discount_applied/jsonschema/1-0-0`) to the Iglu Server's `/api/schemas` endpoint, authenticating with the API key.

For demonstration purposes, the Iglu Server configuration (see ../docker/compose.snowplow.yaml) is set to use the "dummy" database mode, which means all schemas are stored only in memory. **Any schema registered will be lost if the Iglu Server container is restarted.** This setup is ideal for local development and testing, but not for production.

To register the schema, simply run the `register.sh` script.

4. Validating events against the schema in Iglu Server

After registering the schema, you can use the ./validate_events.sh script (which calls ./validate_events.js) to verify two things:

1. That the schema is actually available in the Iglu Server (the script fetches it via HTTP).
2. That your discount event examples (such as ../discounts-processor/e2e/data/MultiProduct.json and ../discounts-processor/e2e/data/MostViewedMultipleViewsPerProduct.json) are valid according to the registered schema.

The validation script will:

1. Fetch the schema from the Iglu Server using the configured URL and API key.
2. Validate each event file against the fetched schema using the Ajv JSON Schema validator (with support for formats like `"date-time"`).

3. Print a message for each event indicating whether it is valid or, if not, what validation errors were found.

To run the validation, simply run the `validate_events.sh` script. It will ensures your schema is correctly registered and your events conform to it, simulating the validation that will occur in the Snowplow pipeline.

5. Demonstration

Inside the `schemas` directory, with all other containers stopped, run the following commands:

```
# Start the Iglu Server:
$ (cd ../docker; docker compose up -d iglu-server)

# Register the schema:
$ ./register.sh
{"message":"Schema
created","updated":false,"location":"iglu:com.snowplow/shopper_discount_applied/jsonsc
hema/1-0-0","status":201}

# Validate the events:
$ ./validate_events.sh
Schema fetched from:
http://localhost:8180/api/schemas/com.snowplow/shopper_discount_applied/jsonschema/1-
0-0
OK: MultiProduct.json is valid.
OK: MostViewedMultipleViewsPerProduct.json is valid.

# Stop the Iglu Server:
$ (cd ../docker; docker compose down -v)
```