Snowplow Live Viewer Profile

Author: Paulo Jeronimo (paulo@oso.sh)

Git commit with doc: d979b5c

HTML version

1. Current status (as of 2024-12-10)

Right now, this project can work my machine (running all inside a Windows/WSL2 environment) without worrying about setting up an infrastructure to run everything as a Databricks Accelerator, as requested in the document. But, this will be my next step ([databricks-setup]) and will be delivered soon (question here).

1.1. What is already in operation at this point?

A sequence of steps, defined below, to permit you to run Snowplow, via Docker Compose, sending data to Kafka through via Snowplow Micro and Snowbridge.

These steps demonstrate how to test, locally, the sending of events from this application (but running it locally) until its consumption, via Kafka Connector, by an application written in Java 21.

1.2. Limitations

1. In this version, as I am using Snowplow Micro to generate events, and it does not have native support for generating events for Kinesis, this is not addressed.

1.3. TODO or ~~DONE or CANCELED~~)

1. Create a entire Docker Compose infrastructure to run components locally.
    a. ~~Configure a Snowplow Micro container~~. ← **DONE**
    b. ~~Create a customized image to call Snowbridge and send events to it by reading data from a micro.tsv file~~. ← **DONE** ← **snowbridge-caller**
    c. ~~Configure a Node.js container to run the media app~~. ← **DONE**
    d. ~~Configure the Snowbridge container to sent events to Kafka.~~ ← **DONE**
    e. ~~Configure a Kafka container~~. ← **DONE**
    f. ~~Configure a Kafka UI container~~. ← **DONE**
    g. Configure a DynamoDB container.
    h. ~~Create a Java 21 application to consume messages from Kafka~~. ← **DONE**
        i. ~~Consume messages from Kafka (coming from snowbridge)~~. ← **DONE**
        ii. Record messages received to DynamoDB.
        iii. ~~Determine a state from a viewer using a video state machine~~. ← **DONE**
    i. ~~Create a Docker image to run the Java app~~. ← **DONE**
2. ~~Create and keep this documentation updated~~. ← **DONE**
3. Maybe:
    a. Refactor the code to make it cleaner and create more test code.
    b. ~~Separate the frontend code from the Java backend~~. ← **DONE**
        i. So the Spring backend will skip to use Thymeleaf and be more focused on manage backend messages (Kafka and WebSockets).
        ii. Since it is only a static page, it can be in a separate container managed by Ngnix (as an example).
    c. ~~frontend → Detect if WebSocket Server (backend) is up or down~~. ← **DONE**
    d. ~~frontend → Change to use Snowplow's website colors~~. ← **DONE**
4. ~~Configure Snowplow to send messages to Kinesis and test it locally with LocalStack~~. ← **DONE**
5. ~~Create a Databricks Accelerator. ← **question here**~~ ← **CANCELED**

My questions (doubts):

question-localstack) Do we need to create a version of this project to run with LocalStack?

Reason: this is a way to insert Kinesis in this solution and continue to test it locally. Currenty, to skip this limitation and use only Snowplow Micro in this version, I created the [snowbridge-caller] project.
This was one of the things I chatted with Trent.

Snowplow's response: Yes. Base code: https://github.com/snowplow-incubator/snowplow-local/

question-databricks) Do we need to create a version of this project to run inside Databricks (as a real [databricks-accelerator])?

Snowplows's response: No. Message: https://osodevops.slack.com/archives/C07RAQVAAJH/p1732132934565859

2. Steps (to run this application as is)

Step 0 → Prerequisites

1. Start a Ubuntu Linux (it can be running on a WSL2 environment) terminal.
2. Make sure you have docker (and docker compose) installed.
3. Clone this project with Git and cd to it.

```
$ repo=git@github.com:osodevops/snowplow-live-viewer-profile-generator.git
$ git clone $repo && cd $(basename $repo .git)
```

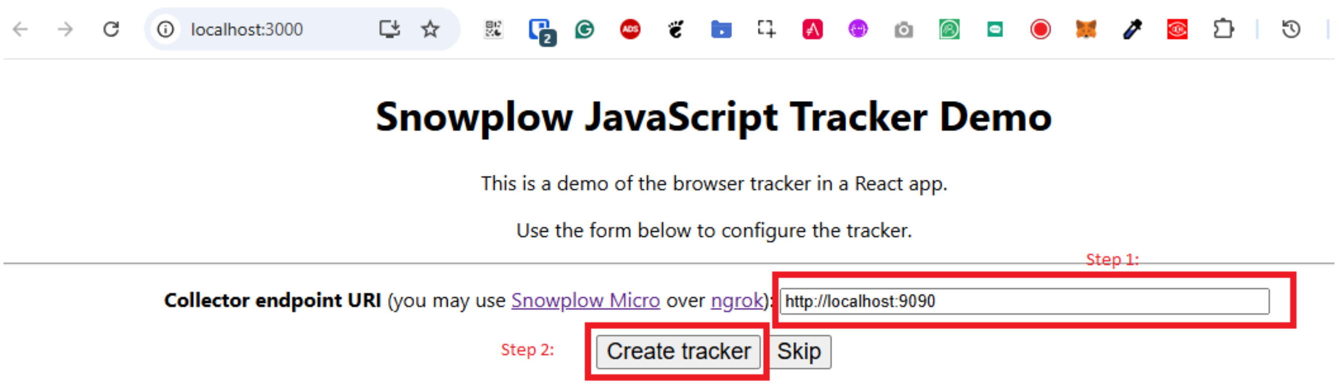
You don’t need Java or Node.js configured on your machine to follow the steps below.

Step 1 → Start the containers

```
$ ./up.sh
```

Step 2 → Open http://localhost:3000 to generate the events

After open this link, configure the collector endpoint:



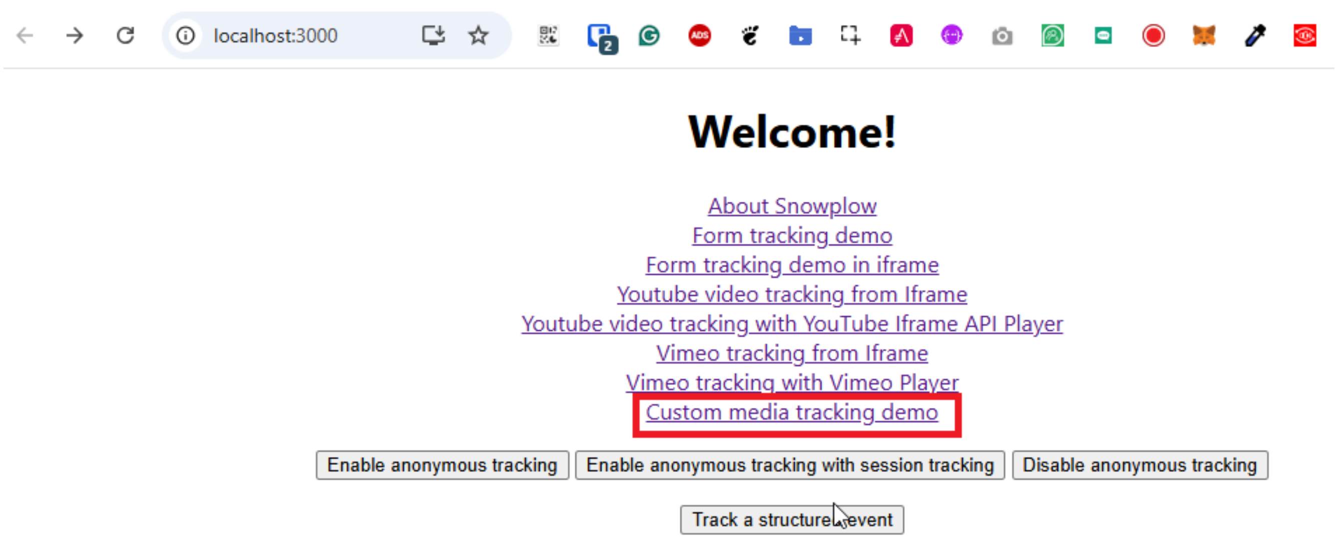
Snowplow JavaScript Tracker Demo

This is a demo of the browser tracker in a React app.

Use the form below to configure the tracker.

Collector endpoint URI (you may use Snowplow Micro over ngrok): http://localhost:9090

Open the "Custom media tracking demo":



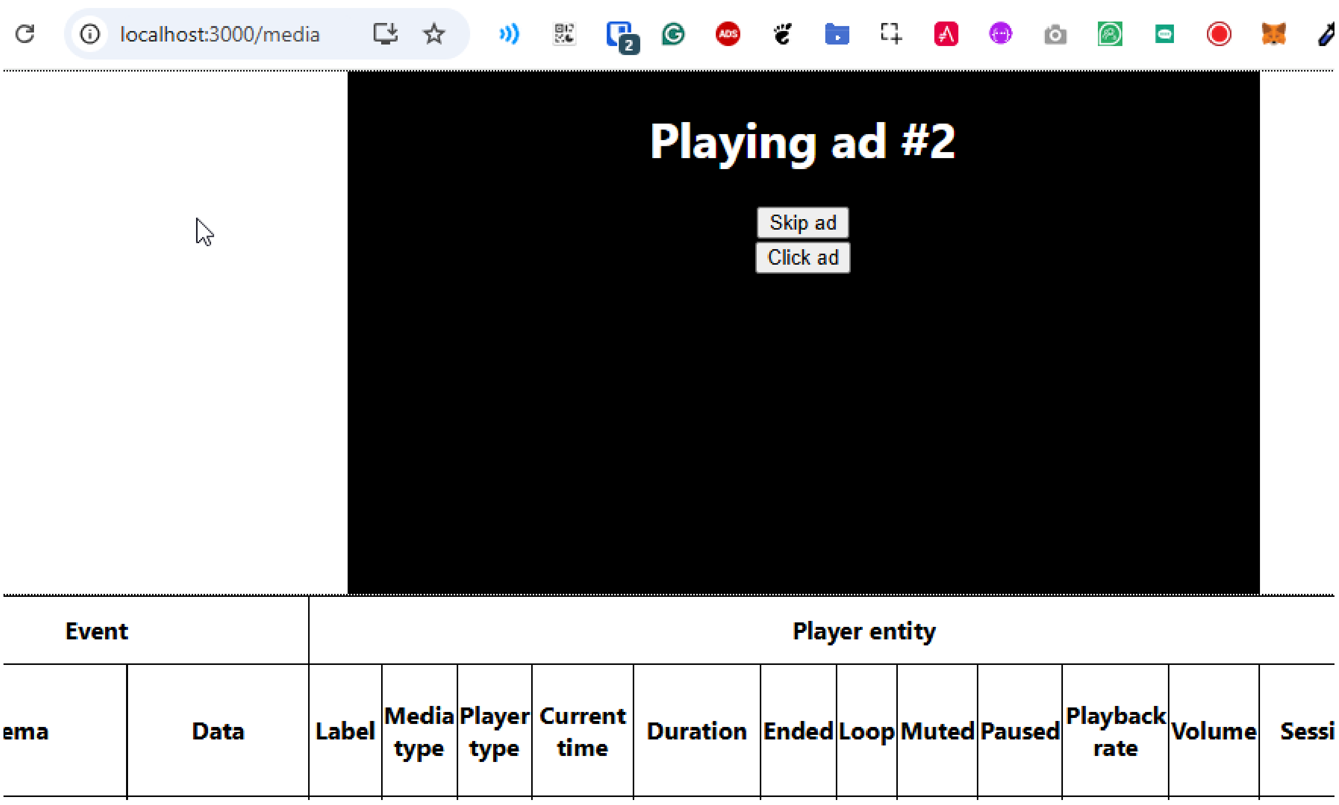
Welcome!

About Snowplow
Form tracking demo
Form tracking demo in iframe
Youtube video tracking from Iframe
Youtube video tracking with YouTube Iframe API Player
Vimeo tracking from Iframe
Vimeo tracking with Vimeo Player
Custom media tracking demo

Enable anonymous tracking | Enable anonymous tracking with session tracking | Disable anonymous tracking

Track a structured event

You will get a page like this one:



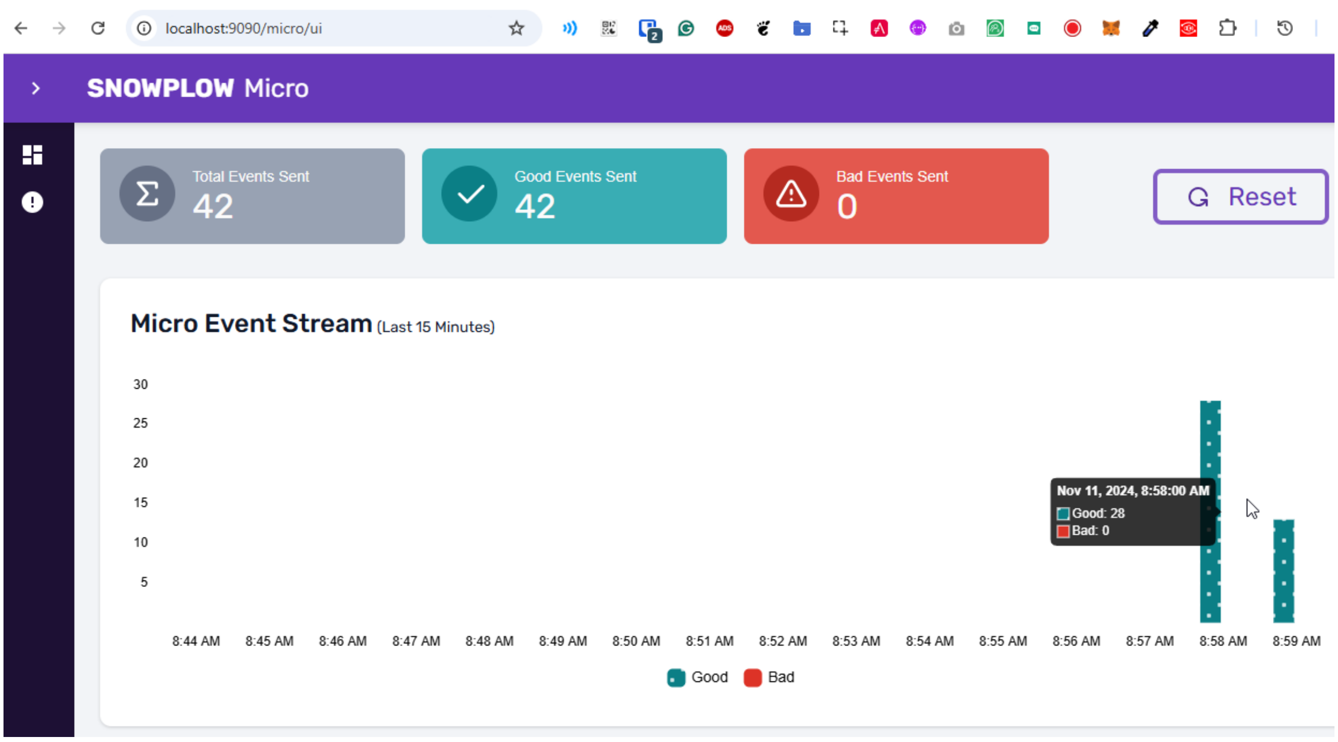
Event		Player entity											
ema	Data	Label	Media type	Player type	Current time	Duration	Ended	Loop	Muted	Paused	Playback rate	Volume	Sessi

Step 3 → Open http://localhost:8280 to see the "Snowplow Live Viewer Profile" UI.

See details on the [video2].

Step 4 → (optional) Open http://localhost:9090/micro/ui to watch events

You will get a page like this one:



Step 5 → (optional) Open a terminal to watch events sent by snowbridge

To watch the number of events sent by snowbridge, type:

```
$ ./data/snowbridge.watch.sh
```

Step 6 → (optional) Open http://localhost:8080 to see the events exported to Kafka UI.

See details on the [video1].

Step 7 → (optional) Open the LocalStack UI

Open https://app.localstack.cloud/inst/default/overview.

Step 8 → (optional) Use LazyDocker to monitor the containers and logs

Step 9 → Stop the containers

To stop all the containers:

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

Step 10 → Restart this lab (to run it from scratch again)

To restart this lab:

```
$ ./restart.sh
```

Warnings:

1. Make sure you call the script `down.sh` before calling `restart.sh`.
2. The script `restart.sh` will call the script `clean.sh` as its firts step.
3. The script `clean.sh` will destroy any data generated by these containers.

3. References

1. **databricks-acelerator**:
   a. https://github.com/databricks-industry-solutions/
   b. https://www.databricks.com/solutions/accelerators

4. Videos demonstrating the with status of this project

- **video2** → https://www.youtube.com/watch?v=CZ5gGOPkGtY → Published on YouTube (unlisted) on Nov 18, 2024.
- **video1** → https://www.youtube.com/watch?v=94U1-Ryjv20 → Published on YouTube (unlisted) on Nov 11, 2024.

5. What I learned in the space of two weeks

Since beginning this project on November 4 and working with Snowplow for about two weeks (by November 18), I've gained some insight into the tool. I've discovered that it's incredibly useful for collecting behavioral data from applications running across various environments. Before diving into the solution requested by Snowplow (as documented here), I decided to experiment with it on a personal project: a WhatsApp chatbot I built in JavaScript. I plan to write an article about that experience soon.

As for this web application, creating this solution gave me a good initial insight into the power that Snowplow offers. It was not easy, however, to set up a local structure to make it work only on the machine. However, I have good experience with Docker, Docker Compose, Bash, JavaScript, and Java. All of this, added to the documentation provided by Snowplow, helped me build this solution.

Initially, I was particularly intrigued by the fact that Snowbridge did not always produce the same output every time I gave it an input file and asked it to generate the events (in stdout) according to the transformation I configured. The events generated by it, in this output, were out of order, and, in my head, a warning light went off that this would be a problem for processing the state that a viewer would be in. I spent a good amount of time developing a lab to understand this. Apparently (I haven't investigated it yet) this should be implemented by implementing multiple threads within Snowbridge. But, to avoid spending too much time trying to understand this, I reduced its call time in snobridge-caller by just 1 second. Then, the result was exactly what I expected.

So far, learning from Snowplow has been a lot of fun. I hope to learn even more about it so I can continue to help your team with whatever they need.

From now on I will focus on the implementation that will save the records from Kafka to DynamoDB. I will give more feedback when I finish this.

6. About this document

This document is written in AsciiDoc format. Its source code is the file `README.adoc` (inside the GitHub repo of the project).

The script `README.sh` generates the files `README.html` and `README.pdf`.