

Serverless Data Processing with Dataflow - Using Dataflow for Streaming Analytics (Python)

2 hours No cost

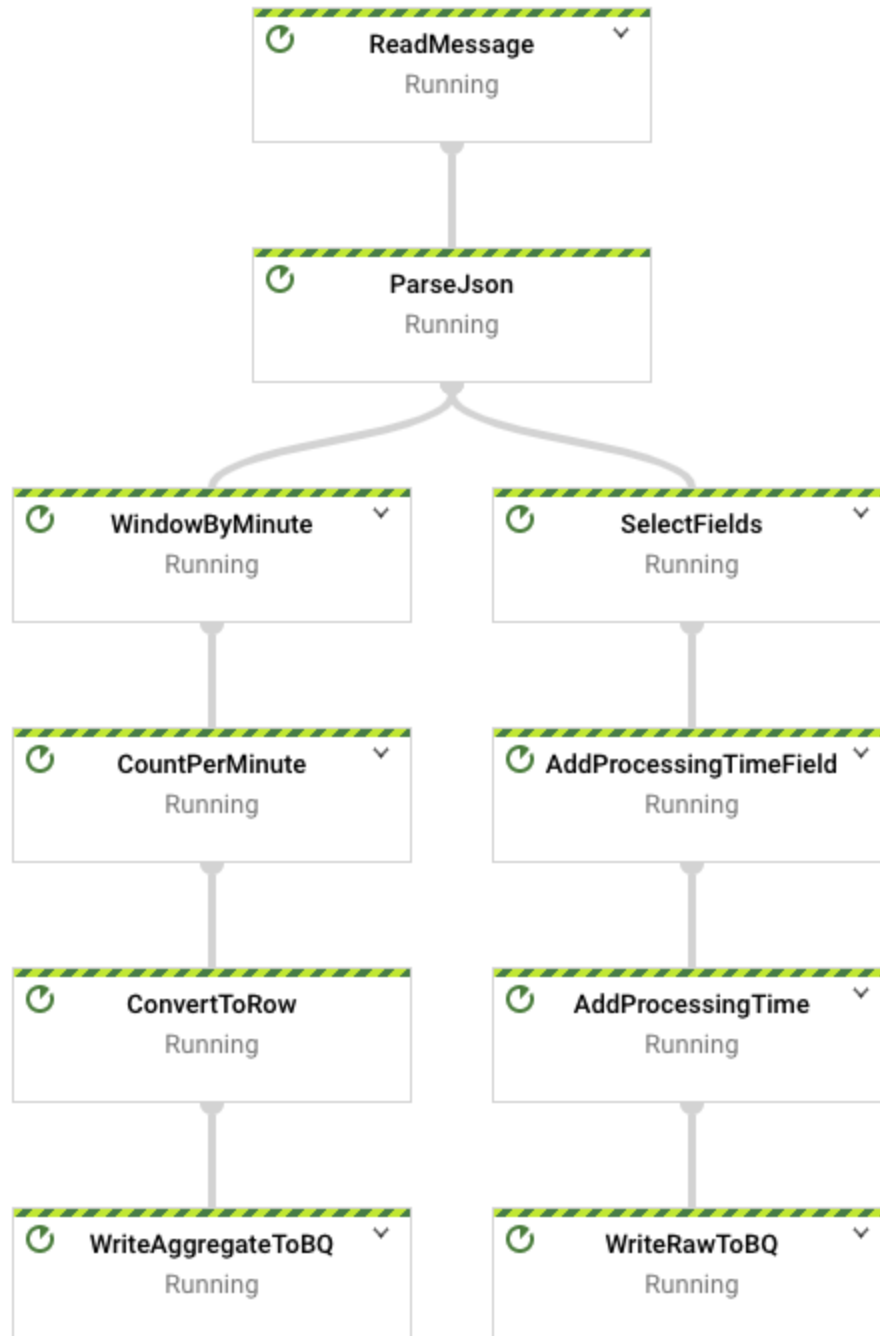
Overview

In this lab, you take many of the concepts introduced in a batch context and apply them in a streaming context to create a pipeline similar to `batch_minute_traffic_pipeline`, but which operates in real time. The finished pipeline will first read JSON messages from Pub/Sub and parse those messages before branching. One branch writes some raw data to BigQuery and takes note of event and processing time. The other branch windows and aggregates the data and then writes the results to BigQuery.

Objectives

- Read data from a streaming source.
- Write data to a streaming sink.
- Window data in a streaming context.
- Experimentally verify the effects of lag.

You will build the following pipeline:



Setup and requirements

Before you click the Start Lab button

Note: Read these instructions.

Labs are timed and you cannot pause them. The timer, which starts when you click **Start Lab**, shows how long Google Cloud resources will be made available to you.

This Qwiklabs hands-on lab lets you do the lab activities yourself in a real cloud environment, not in a simulation or demo environment. It does so by giving you new, temporary credentials that you use to sign in and access Google Cloud for the duration of the lab.

What you need

To complete this lab, you need:

- Access to a standard internet browser (Chrome browser recommended).
- Time to complete the lab.


Note: If you already have your own personal Google Cloud account or project, do not use it for this lab. **Note:** If you are using a Pixelbook, open an Incognito window to run this lab.


How to start your lab and sign in to the Console


1. Click the **Start Lab** button. If you need to pay for the lab, a pop-up opens for you to select your payment method. On the left is a panel populated with the temporary credentials that you must use for this lab.

Open Google Console

Caution: When you are in the console, do not deviate from the lab instructions. Doing so may cause your account to be blocked. [Learn more.](#)

Username
google2727032_student@qwiklabs.n 

Password
k68CZXsxMZ 

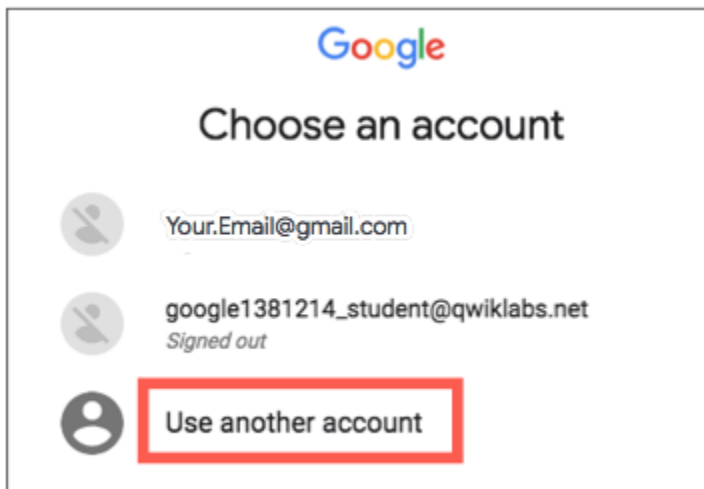
GCP Project ID
qwiklabs-gcp-4fbfecac8667e457 

[New to labs? View our introductory video!](#)

- Copy the username, and then click **Open Google Console**. The lab spins up resources, and then opens another tab that shows the **Choose an account** page.

Note: Open the tabs in separate windows, side-by-side.

- On the Choose an account page, click **Use Another Account**. The Sign in page opens.



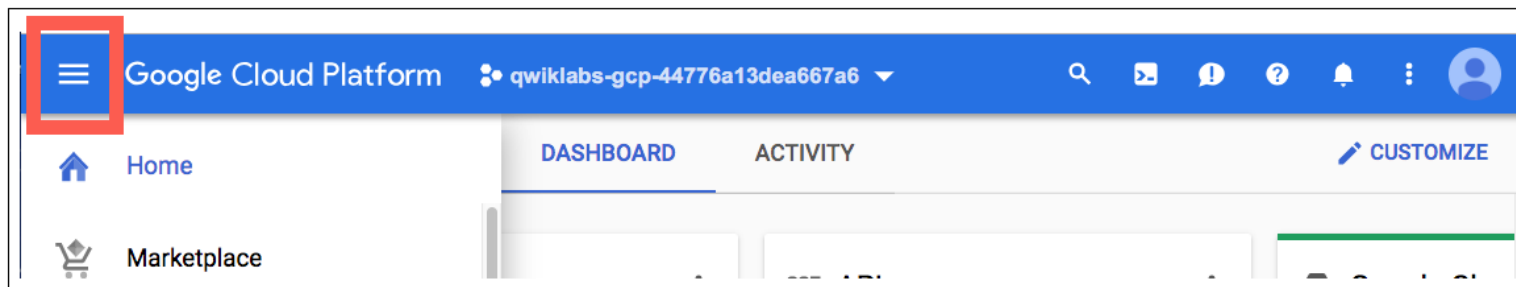
4. Paste the username that you copied from the Connection Details panel. Then copy and paste the password.

Note: You must use the credentials from the Connection Details panel. Do not use your Google Cloud Skills Boost credentials. If you have your own Google Cloud account, do not use it for this lab (avoids incurring charges).

5. Click through the subsequent pages:
 - Accept the terms and conditions.
 - Do not add recovery options or two-factor authentication (because this is a temporary account).
 - Do not sign up for free trials.

After a few moments, the Cloud console opens in this tab.

Note: You can view the menu with a list of Google Cloud Products and Services by clicking the **Navigation menu** at the top-left.



Jupyter notebook-based development environment setup

For this lab, you will be running all commands in a terminal from your notebook.

1. In the Google Cloud Console, on the **Navigation Menu**, click **Vertex AI > Workbench**.
2. Enable **Notebooks API**.
3. On the Workbench page, click **CREATE NEW**.
4. In the **New instance** dialog box that appears, set the region to and zone to .
5. For Environment, select **Apache Beam**.
6. Click **CREATE** at the bottom of the dialog box.

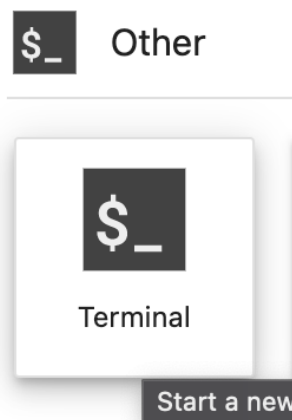
Note: The environment may take 3 - 5 minutes to be fully provisioned. Please wait until the step is complete. **Note:** Click **Enable Notebook API** to enable the notebook api.

- Once the environment is ready, click the **OPEN JUPYTERLAB** link next to your Notebook name. This will open up your environment in a new tab in your browser.

Filter

<input type="checkbox"/>	<input checked="" type="checkbox"/>	Notebook name ↑		Zone	Auto upgrade	Environment	Machine Type
<input type="checkbox"/>	<input checked="" type="checkbox"/>	instance-20231219-123340	OPEN JUPYTERLAB	us-west1-a	—	NumPy/SciPy/scikit-learn	Efficient Instance: 4 vCPUs, 16 GB RAM

- Next, click **Terminal**. This will open up a terminal where you can run all the commands in this lab.



Download Code Repository

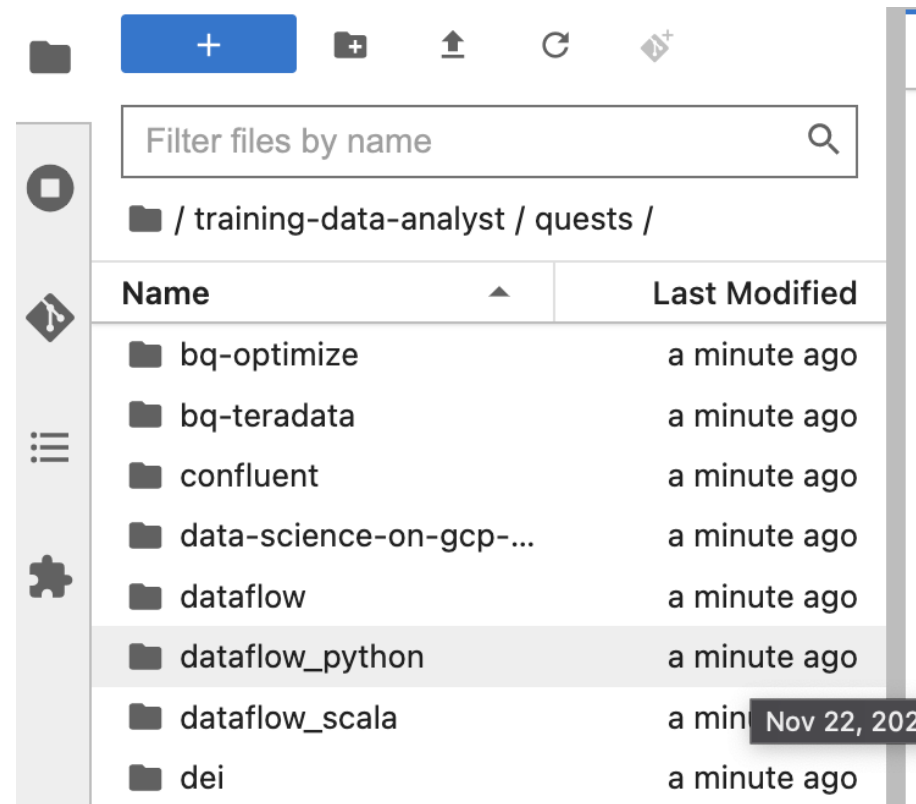
Next you will download a code repository for use in this lab.

- In the terminal you just opened, enter the following:

```
git clone https://github.com/GoogleCloudPlatform/training-data-analyst cd /home/jupyter/training-data-analyst/quests/dataflow_python/
```

- On the left panel of your notebook environment, in the file browser, you will notice the **training-data-analyst** repo added.

3. Navigate into the cloned repo `/training-data-analyst/quests/dataflow_python/`. You will see a folder for each lab, which is further divided into a `lab` sub-folder with code to be completed by you, and a `solution` sub-folder with a fully workable example to reference if you get stuck.



Note: To open a file for editing purposes, simply navigate to the file and click on it. This will open the file, where you can add or modify code.

Open the appropriate lab

- In your terminal, run the following commands to change to the directory you will use for this lab:

```
# Change directory into the lab cd 5_Streaming_Analytics/lab export BASE_DIR=$(pwd)
```

Setting up dependencies

Before you can begin editing the actual pipeline code, you need to ensure that you have installed the necessary dependencies.

1. Execute the following to install the packages you will need to execute your pipeline:

```
python3 -m pip install -q --upgrade pip setuptools wheel python3 -m pip install apache-beam[gcp]
```

2. Ensure that the Dataflow API is enabled:

```
gcloud services enable dataflow.googleapis.com
```

3. Finally, grant the `dataflow.worker` role to the Compute Engine default service account:

```
PROJECT_ID=$(gcloud config get-value project) export PROJECT_NUMBER=$(gcloud projects list --filter="$PROJECT_ID" --format="value(PROJECT_NUMBER)")  
export serviceAccount=""$PROJECT_NUMBER"-compute@developer.gserviceaccount.com"
```

4. In the Cloud Console, navigate to **IAM & ADMIN > IAM**, click on **Edit principal** icon for Compute Engine default service account.
5. Add **Dataflow Worker** as another role and click **Save**.

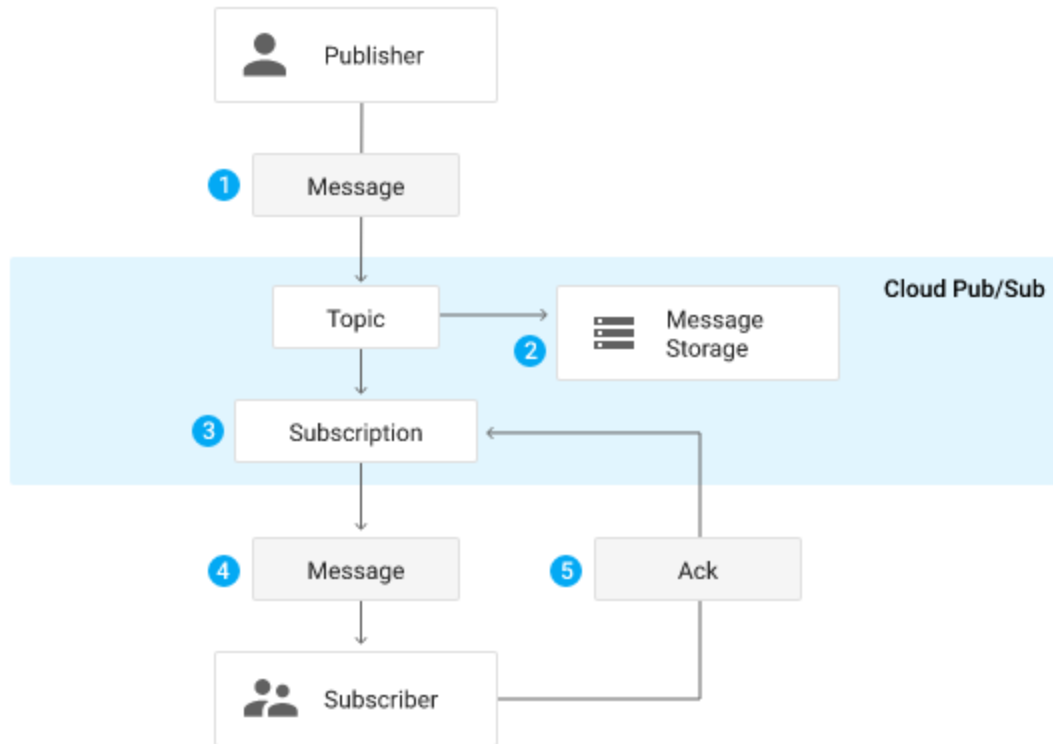
Set up the data environment

```
# Create GCS buckets and BQ dataset cd $BASE_DIR/../../ source create_streaming_sinks.sh # Change to the directory containing the practice version of the code  
cd $BASE_DIR
```

Click *Check my progress* to verify the objective. Set up the data environment

Task 1. Reading from a streaming source

In the previous labs, you used [beam.io.ReadFromText](#) to read from Google Cloud Storage. In this lab, instead of Google Cloud Storage, you use Pub/Sub. Pub/Sub is a fully managed real-time messaging service that allows publishers to send messages to a "topic," to which subscribers can subscribe via a "subscription."



The pipeline you create subscribes to a topic called `my_topic` that you just created via `create_streaming_sinks.sh` script. In a production situation, this topic will often be created by the publishing team. You can view it in the [Pub/Sub portion of the console](#).

1. In the file explorer, navigate to `training-data-analyst/quest/dataflow_python/5_Streaming_Analytics/lab/` and open the `streaming_minute_traffic_pipeline.py` file.
- To read from Pub/Sub using Apache Beam's IO connectors, add a transform to the pipeline which uses the [beam.io.ReadFromPubSub\(\)](#) class. This class has attributes for specifying the source topic as well as the `timestamp_attribute`. By default, this attribute is set to the message publishing time.

Note: Publication time is the time when the Pub/Sub service first receives the message. In systems where there may be a delay between the actual event time and publish time (i.e., late data) and you would like to take this into account, the client code publishing the message needs to set a 'timestamp' metadata attribute on the message and provide the actual event timestamp, since Pub/Sub will not natively know how to extract the event timestamp embedded in the payload. You can see the [client code generating the messages you'll use here](#).

To complete this task:

- Add a transform that reads from the Pub/Sub topic specified by the `input_topic` command-line parameter.
- Then, use the provided function, `parse_json` with `beam.Map` to convert each JSON string into a `CommonLog` instance.
- Collect the results from this transform into a `PCollection` of `CommonLog` instances using `with_output_types()`.

2. In the first `#TODO`, add the following code:

```
beam.io.ReadFromPubSub(input_topic)
```

Task 2. Window the data

In the [previous non-SQL lab](#), you implemented fixed-time windowing in order to group events by event time into mutually-exclusive windows of fixed size. Do the same thing here with the streaming inputs. Feel free to reference the [previous lab's code](#) or the [solution](#) if you get stuck.

Window into one-minute windows

To complete this task:

1. Add a transform to your pipeline that accepts the `PCollection` of `CommonLog` data and windows elements into windows of `window_duration` seconds long, with `window_duration` as another command-line parameter.
2. Use the following code to add a transform to your pipeline that windows elements into one-minute windows:

```
"WindowByMinute" >> beam.WindowInto(beam.window.FixedWindows(60))
```

Task 3. Aggregate the data

In the previous lab, you used the `CountCombineFn()` combiner to count the number of events per window. Do the same here.

Count events per window

To complete this task:

1. Pass the windowed `PCollection` as input to a transform that counts the number of events per window.
2. After this, use the provided `DoFn`, `GetTimestampFn`, with `beam.ParDo` to include the window start timestamp.
3. Use the following code to add a transform to your pipeline that counts the number of events per window:

```
"CountPerMinute" >> beam.CombineGlobally(CountCombineFn()).without_defaults()
```

Task 4. Write to BigQuery

This pipeline writes to BigQuery in two separate branches. The first branch writes the aggregated data to BigQuery. The second branch, which has already been authored for you, writes out some metadata regarding each raw event, including the event timestamp and the actual processing timestamp. Both write directly to BigQuery via streaming inserts.

Write aggregated data to BigQuery

Writing to BigQuery has been covered extensively in previous labs, so the basic mechanics will not be covered in depth here.

To complete this task:

- Create a new command-line parameter called `agg_table_name` for the table intended to house aggregated data.
- Add a transform as before that writes to BigQuery.

Note: When in a streaming context, `beam.io.WriteToBigQuery()` does not support `write_disposition` of `WRITE_TRUNCATE` in which the table is dropped and recreated. In this example, use `WRITE_APPEND`.

BigQuery insertion method

`beam.io.WriteToBigQuery` will default to either [streaming inserts](#) for unbounded PCollections or [batch file load jobs](#) for bounded PCollections. Streaming inserts can be particularly useful when you want data to show up in aggregations immediately, but does incur [extra charges](#). In streaming use cases where you are OK with periodic batch uploads on the order of every couple minutes, you can specify this behavior via the `method` keyword argument, and also set the frequency with the `triggering_frequency` keyword argument. Learn more from the [Write data to BigQuery section of the apache beam.io.gcp.bigquery module documentation](#).

- Use the following code to add a transform to your pipeline that writes aggregated data to the BigQuery table.

```
'WriteAggToBQ' >> beam.io.WriteToBigQuery( agg_table_name, schema=agg_table_schema,
create_disposition=beam.io.BigQueryDisposition.CREATE_IF_NEEDED, write_disposition=beam.io.BigQueryDisposition.WRITE_APPEND )
```

Task 5. Run your pipeline

- Return to the terminal and execute the following code to run your pipeline:

```
export PROJECT_ID=$(gcloud config get-value project) export REGION='us-central1' export BUCKET=gs://{PROJECT_ID} export PIPELINE_FOLDER=${BUCKET}
export RUNNER=DataflowRunner export PUBSUB_TOPIC=projects/{PROJECT_ID}/topics/my_topic export WINDOW_DURATION=60 export
AGGREGATE_TABLE_NAME=${PROJECT_ID}:logs.windowed_traffic export RAW_TABLE_NAME=${PROJECT_ID}:logs.raw python3
streaming_minute_traffic_pipeline.py \ --project={PROJECT_ID} \ --region={REGION} \ --staging_location={PIPELINE_FOLDER}/staging \ --
temp_location={PIPELINE_FOLDER}/temp \ --runner={RUNNER} \ --input_topic={PUBSUB_TOPIC} \ --window_duration={WINDOW_DURATION} \ --
agg_table_name={AGGREGATE_TABLE_NAME} \ --raw_table_name={RAW_TABLE_NAME} Note: If you get a Dataflow pipeline failed error saying that it is
unable to open the pipeline.py file, run the pipeline again and it should run with no issues.
```

Ensure in the [Dataflow UI](#) that it executes successfully without errors. Note that there is no data yet being created and ingested by the pipeline, so it will be running but not processing anything. You will introduce data in the next step.

Click *Check my progress* to verify the objective. Run your pipeline

Task 6. Generate lag-less streaming input

Because this is a streaming pipeline, it subscribes to the streaming source and will await input; there is none currently. In this section, you generate data with no lag. Actual data will almost invariably contain lag. However, it is instructive to understand lag-less streaming inputs.

The code for this quest includes a script for publishing JSON events using Pub/Sub.

- To complete this task and start publishing messages, open a **new terminal** side-by-side with your current one and run the following script. It will keep publishing messages until you kill the script. Make sure you are in the `training-data-analyst/quests/dataflow_python` folder.

```
bash generate_streaming_events.sh
```

Click *Check my progress* to verify the objective. Generate lag-less streaming input

Examine the results

1. Wait a couple minutes for the data to start to populate. Then navigate to [BigQuery](#) and query the `logs.minute_traffic` table with the following query:

```
SELECT timestamp, page_views FROM `logs.windowed_traffic` ORDER BY timestamp ASC
```

You should see that the number of pageviews hovered around 100 views a minute.

2. Alternatively, you can use the BigQuery command-line tool as a quick way to confirm results are being written:

bq head logs.raw bq head logs.windowed_traffic

3. Now, enter the following query:

```
SELECT UNIX_MILLIS(TIMESTAMP(event_timestamp)) - min_millis.min_event_millis AS event_millis, UNIX_MILLIS(TIMESTAMP(processing_timestamp)) - min_millis.min_event_millis AS processing_millis, user_id, -- added as unique label so we see all the points CAST(UNIX_MILLIS(TIMESTAMP(event_timestamp)) - min_millis.min_event_millis AS STRING) AS label FROM `logs.raw` CROSS JOIN ( SELECT MIN(UNIX_MILLIS(TIMESTAMP(event_timestamp))) AS min_event_millis FROM `logs.raw`) min_millis WHERE event_timestamp IS NOT NULL ORDER BY event_millis ASC
```

This query illustrates the gap between event time and processing time. However, it can be hard to see the big picture by looking at just the raw tabular data. We will use Looker Studio, a lightweight data visualization and BI engine.

4. To enable Looker Studio:

- Visit [Looker Studio](#).
- Click **Create** in the upper left.
- Click **Report**.
- Select **Country** name and enter a **Company** name. Check the checkbox to acknowledge you have read and agree to the Looker Studio Additional Terms, then click **Continue**.
- Select "No" to all options, then click **Continue**.
- Return to the BigQuery UI.

5. In the BigQuery UI, click on the **Explore data** button and choose **Explore With Looker Studio**.

This will open a new window.

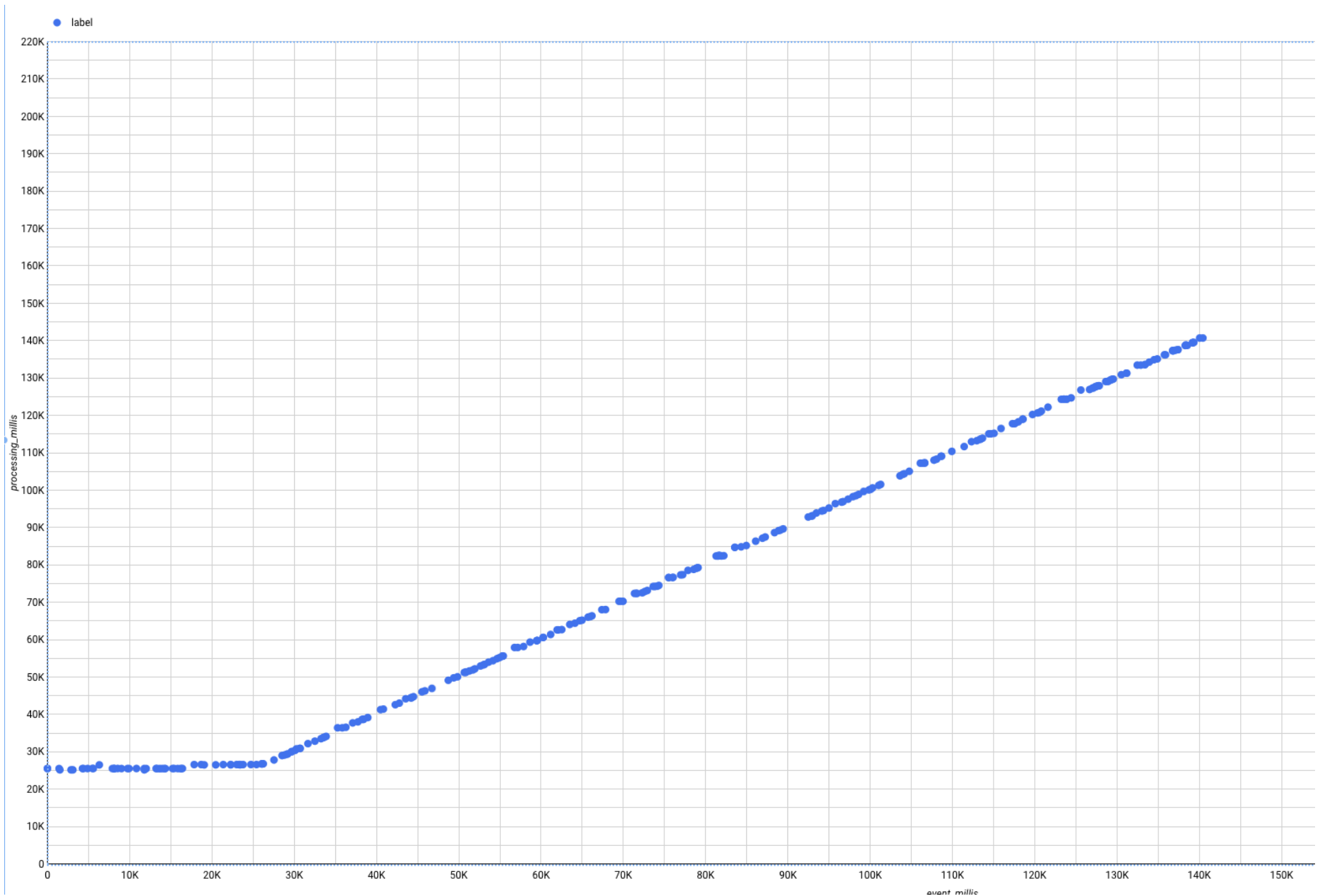
6. Click **Get Started**.

Notice that default visualizations are created for the data.

7. To remove the default visualizations, right-click on each, and select **Delete**.
8. Click **Add a chart** on the top menu bar.
9. Select the **Scatter chart** type.
10. In the **Data** column of the panel on the right hand side, set the following values:
 - Dimension: label
 - Metric X: event_millis
 - Metric Y: processing_millis

The chart will transform to be a scatterplot, where all points are on the diagonal. This is because in the streaming data currently being generated, events are processed immediately after they were generated — there was no lag. If you started the data generation script quickly, i.e. before the Dataflow job was fully up and running, you may see a hockey stick, as there were messages queuing in Pub/Sub that were all processed more or less at once.

But in the real world, lag is something that pipelines need to cope with.



Task 7. Introduce lag to streaming input

The streaming event script is capable of generating events with simulated lag.

This represents scenarios where there is a time delay between when the events are generated and published to Pub/Sub, for example when a mobile client goes into offline mode if a user has no service, but events are collected on the device and all published at once when the device is back online.

Generate streaming input with lag

1. First, close the Looker Studio window.
2. Then, to turn on lag, return to the terminal and stop the running script using `CTRL+C`.
3. Then, run the following:

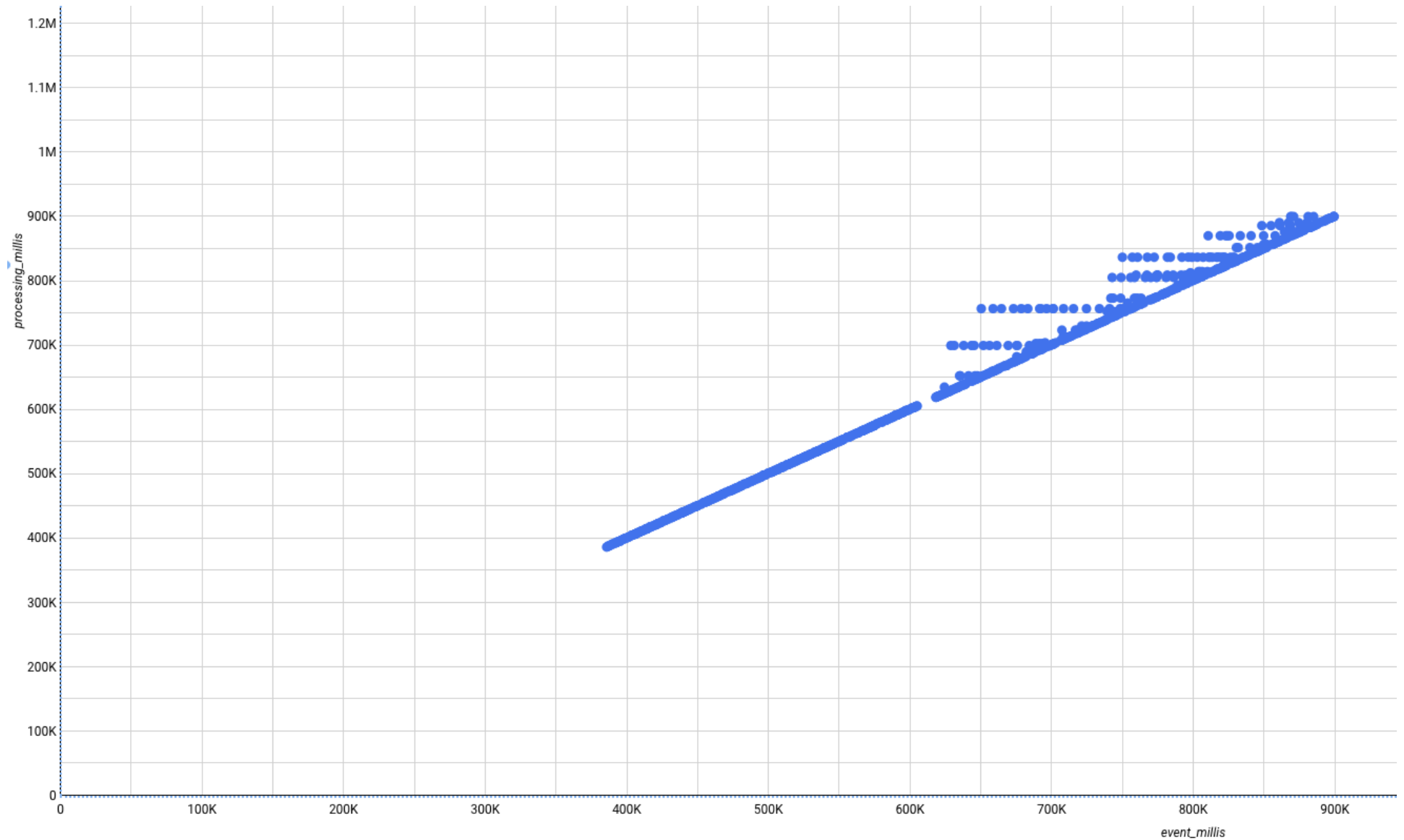
```
bash generate_streaming_events.sh true
```

Examine the results

- Return to the BigQuery UI, rerun the query, and then recreate the Looker Studio view as before. The new data that arrive, which should appear on the right side of the chart, should no longer be perfect; instead, some will appear above the diagonal, indicating that they were processed after the events transpired.

Chart Type: Scatter

- Dimension: label
- Metric X: event_millis
- Metric Y: processing_millis



End your lab

When you have completed your lab, click **End Lab**. Google Cloud Skills Boost removes the resources you've used and cleans the account for you.

You will be given an opportunity to rate the lab experience. Select the applicable number of stars, type a comment, and then click **Submit**.

The number of stars indicates the following:

- 1 star = Very dissatisfied
- 2 stars = Dissatisfied
- 3 stars = Neutral
- 4 stars = Satisfied
- 5 stars = Very satisfied

You can close the dialog box if you don't want to provide feedback.

For feedback, suggestions, or corrections, please use the **Support** tab.

Copyright 2022 Google LLC All rights reserved. Google and the Google logo are trademarks of Google LLC. All other company and product names may be trademarks of the respective companies with which they are associated.

- [Overview](#)
- [Setup and requirements](#)
- [Task 1. Reading from a streaming source](#)
- [Task 2. Window the data](#)
- [Task 3. Aggregate the data](#)
- [Task 4. Write to BigQuery](#)
- [Task 5. Run your pipeline](#)
- [Task 6. Generate lag-less streaming input](#)
- [Task 7. Introduce lag to streaming input](#)
- [End your lab](#)