

Serverless Data Processing with Dataflow - Branching Pipelines (Python)

2 hours No cost

Overview

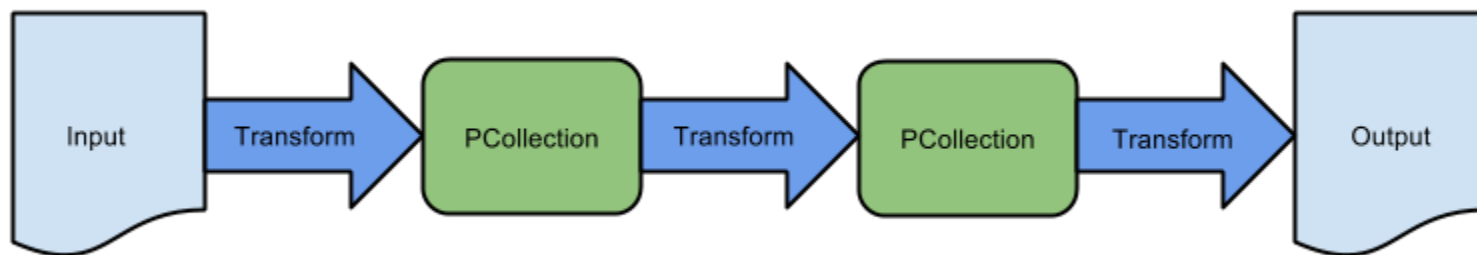
In this lab, you:

- Implement a pipeline that has branches.
- Filter data before writing.
- Add custom command-line parameters to a pipeline.

Prerequisites:

- Basic familiarity with Python.

In the previous lab, you created a basic Extract-Transform-Load sequential pipeline and used an equivalent Dataflow Template to ingest batch data storage on Google Cloud Storage. This pipeline consists of a sequence of transformations:



Many pipelines will not exhibit such simple structure, though. In this lab, you build a more sophisticated, non-sequential pipeline.

The use case here is to optimize resource consumption. Products vary with respect to how they consume resources. Additionally, not all data is used in the same way within a business; some data will be regularly queried, for example, within analytic workloads, and some data will only be used for

recovery. In this lab, you optimize the pipeline from the first lab for resource consumption, by storing only data that analysts will use in BigQuery while archiving other data in a very low-cost, highly durable storage service: Coldline storage in Google Cloud Storage.

Setup and requirements


For each lab, you get a new Google Cloud project and set of resources for a fixed time at no cost.

1. Sign in to Qwiklabs using an **incognito window**.
2. Note the lab's access time (for example, 1:15:00), and make sure you can finish within that time.
There is no pause feature. You can restart if needed, but you have to start at the beginning.
3. When ready, click **Start lab**.
4. Note your lab credentials (**Username** and **Password**). You will use them to sign in to the Google Cloud Console.
5. Click **Open Google Console**.
6. Click **Use another account** and copy/paste credentials for **this** lab into the prompts.
If you use other credentials, you'll receive errors or **incur charges**.
7. Accept the terms and skip the recovery resource page.

Note: Do not click **End Lab** unless you have finished the lab or want to restart it. This clears your work and removes the project.

Check project permissions

Before you begin your work on Google Cloud, you need to ensure that your project has the correct permissions within Identity and Access Management (IAM).

1. In the Google Cloud console, on the **Navigation menu** () , select **IAM & Admin > IAM**.
2. Confirm that the default compute Service Account `{project-number}-compute@developer.gserviceaccount.com` is present and has the `editor` role assigned. The account prefix is the project number, which you can find on **Navigation menu > Cloud Overview > Dashboard**.

Permissions for project "qwiklabs-gcp-00-3f97701829bb"

These permissions affect this project and all of its resources. [Learn more](#)

☐ Include Google-provided role grants

VIEW BY PRINCIPALS

VIEW BY ROLES

+ GRANT ACCESS

- REMOVE ACCESS

Filter Enter property name or value

?

III

<input type="checkbox"/>	Type	Principal ↑	Name	Role	Security insights ?	Inheritance
<input type="checkbox"/>		96496971506-compute@developer.gserviceaccount.com	Compute Engine default service account	Editor Owner		
<input type="checkbox"/>		admiral@qwiklabs-services-prod.iam.gserviceaccount.com		Owner		
<input type="checkbox"/>		qwiklabs-gcp-00-3f97701829bb@qwiklabs-gcp-00-3f97701829bb.iam.gserviceaccount.com	Qwiklabs User Service Account	BigQuery Admin Owner Storage Admin		
<input type="checkbox"/>		student-03-93dbfa673ace@qwiklabs.net	student 7451284e	App Engine Admin BigQuery Admin Dataflow Admin Dataflow Developer Editor Owner Viewer		

Note: If the account is not present in IAM or does not have the `editor` role, follow the steps below to assign the required role.

1. In the Google Cloud console, on the **Navigation menu**, click **Cloud Overview > Dashboard**.
2. Copy the project number (e.g. 729328892908).
3. On the **Navigation menu**, select **IAM & Admin > IAM**.
4. At the top of the roles table, below **View by Principals**, click **Grant Access**.

5. For **New principals**, type:

{project-number}-compute@developer.gserviceaccount.com

6. Replace {project-number} with your project number.
7. For **Role**, select **Project** (or Basic) > **Editor**.
8. Click **Save**.


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 **Python 3 (with Intel® MKL)**.
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.

7. 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

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



Other



Terminal

Start a new

Download Code Repository

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

1. 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/
```

2. 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.

Filter files by name

/ training-data-analyst / quests /

Name	Last Modified
bq-optimize	a minute ago
bq-teradata	a minute ago
confluent	a minute ago
data-science-on-gcp-...	a minute ago
dataflow	a minute ago
dataflow_python	a minute ago
dataflow_scala	a minute ago Nov 22, 202
dei	a minute ago

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.

Click **Check my progress** to verify the objective. Create notebook instance and clone course repo

Multiple transforms process the same PCollection

In this lab, you write a branching pipeline that writes data to both Google Cloud Storage and to BigQuery.

One way of writing a branching pipeline is to apply two *different* transforms to the same PCollection, resulting in two *different* PCollections:

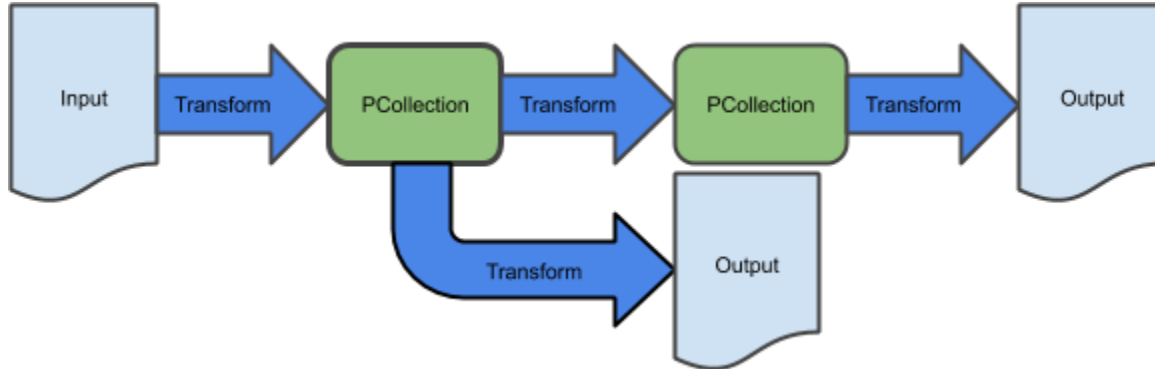
`[PCollection1] = [Initial Input PCollection] | [A Transform]` `[PCollection2] = [Initial Input PCollection] | [A Different Transform]`

Implementing a branching pipeline

If you get stuck in this or later sections, the solution is available from the [Google Cloud training-data-analyst page](#).

Task 1. Add a branch to write to Cloud Storage

To complete this task, modify an existing pipeline by adding a branch that writes to Cloud Storage.



Open the appropriate lab

- In the terminal in your IDE environment, and run the following commands:

```
# Change directory into the lab cd 2_Branching_Pipelines/lab export BASE_DIR=$(pwd)
```

Setting up virtual environment and dependencies

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

1. In the terminal in your IDE environment, run the commands below create a virtual environment for your work in this lab:

```
sudo apt-get update && sudo apt-get install -y python3-venv python3 -m venv df-env source df-env/bin/activate
```

2. Next, 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]
```

3. Finally, ensure that the Dataflow API is enabled:

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

Set up the data environment

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

1. Open up `my_pipeline.py` in your IDE, which can be found in `training-data-analyst/quests/dataflow_python/2_Branching_Pipelines/labs/`.
2. Scroll down to the `run()` method, where the body of the pipeline is defined. It currently looks as follows:

```
(p | 'ReadFromGCS' >>> beam.io.ReadFromText(input) | 'ParseJson' >>> beam.Map(parse_json) | 'WriteToBQ' >>> beam.io.WriteToBigQuery( output,
schema=table_schema, create_disposition=beam.io.BigQueryDisposition.CREATE_IF_NEEDED,
write_disposition=beam.io.BigQueryDisposition.WRITE_TRUNCATE ) )
```

3. Modify this code by adding a new branching transform that writes to Cloud Storage using [textio.WriteToText](#) before each element is converted from `json` to `dict`.

If you get stuck in this or later sections, refer to the solution, which can be found on the [Google Cloud training-data-analyst page](#).

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

Task 2. Filter data by field

At the moment, the new pipeline doesn't actually consume fewer resources, since all data are being stored twice. To start improving resource consumption, we need to reduce the amount of duplicated data.

The Google Cloud Storage bucket is intended to function as archival and backup storage, so it's important that all data be stored there. However, not all data necessarily need to be sent to BigQuery.

1. Let's assume that data analysts often look at what resources users access on the website, and how those access patterns differ as a function of geography and time. Only a subset of the fields would be necessary. Since we have parsed the JSON elements into dictionaries, we can easily use the `pop` method to drop a field from within a Python callable:

```
def drop_field(element): element.pop('field_name') return element
```

2. To complete this task, use a Python callable with `beam.Map` to drop the field `user_agent`, which our analysts will not be using in BigQuery.

Task 3. Filter data by element

There are many ways of filtering in Apache Beam. Since we are working with a `PCollection` of Python dictionaries, the easiest manner will be to leverage a lambda (anonymous) function as our filter, a function returning a boolean value, with `beam.Filter`. For example:

```
purchases | beam.Filter(lambda element : element['cost_cents'] > 20*100)
```

- To complete this task, add a `beam.Filter` transform to the pipeline. You may filter on whatever criteria you wish, but as a suggestion, try eliminating rows where `num_bytes` is greater than or equal to 120.

Task 4. Adding custom command-line parameters

The pipeline currently has a number of parameters hard-coded into it, including the path to the input and the location of the table in BigQuery. However, the pipeline would be more useful if it could read *any* JSON file in Cloud Storage. Adding this feature requires adding to the set of command-line parameters.

Currently, we use an `ArgumentParser` to read in and parse command-line arguments. We then pass these arguments into the `PipelineOptions()` object we specify when creating our pipeline:

```
parser = argparse.ArgumentParser(description='...') # Define and parse arguments
options = PipelineOptions() # Set options values from options
p = beam.Pipeline(options=options)
```

The [PipelineOptions](#) is used to interpret the options being read by the `ArgumentParser`. To add a new command-line argument to the parser, we can use the syntax:

```
parser.add_argument('--argument_name', required=True, help='Argument description')
```

To access a command-line parameter in code, parse the arguments and refer to the field in the resulting dictionary:

```
opts = parser.parse_args() arg_value = opts.arg_name
```

- To complete this task, add command-line parameters for the input path, the Google Cloud Storage output path, and the BigQuery table name, and update the pipeline code to access those parameters instead of constants.

Task 5. Add nullable fields to your pipeline


You may have noticed that the BigQuery table created in the last lab had a schema with all `REQUIRED` fields like this:

logs

[Schema](#)

[Details](#)

[Preview](#)

Field name	Type	Mode	Policy tags 	Description
user_id	STRING	REQUIRED		
ip	STRING	REQUIRED		
lat	FLOAT	REQUIRED		
lng	FLOAT	REQUIRED		
timestamp	STRING	REQUIRED		
http_request	STRING	REQUIRED		
user_agent	STRING	REQUIRED		
http_response	INTEGER	REQUIRED		
num_bytes	INTEGER	REQUIRED		

Edit schema

It may be desirable to create an Apache Beam schema with `NULLABLE` fields where data is missing, both for the pipeline execution itself and then for the resulting BigQuery table.

We can update the JSON BigQuery schema by adding a new property `mode` for a field we wish to be nullable:

```
{ "name": "field_name", "type": "STRING", "mode": "NULLABLE" }
```

- To complete this task, mark the `lat` and `lon` fields as nullable in the BigQuery schema.

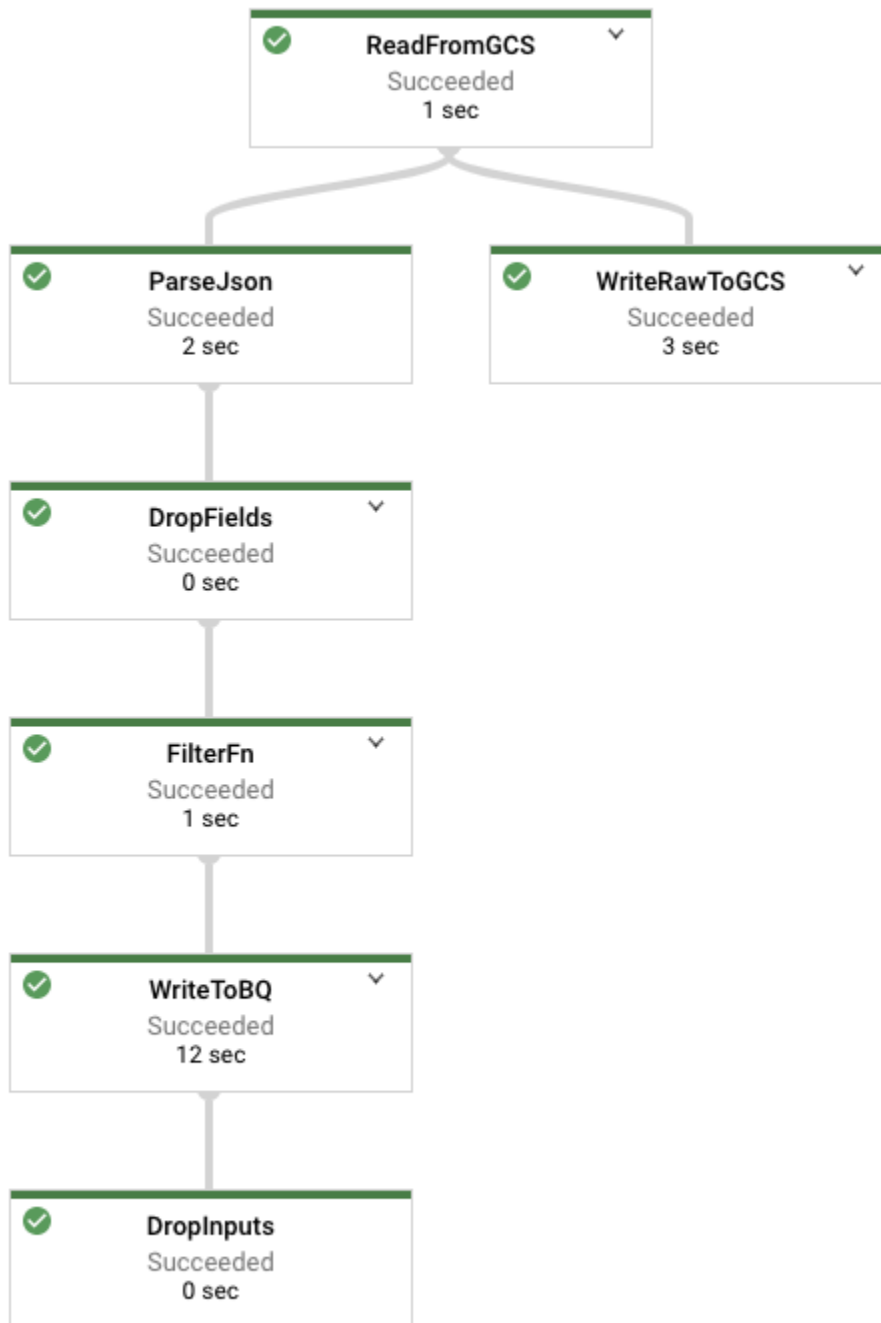
Task 6. Run your pipeline from the command line

- To complete this task, run your pipeline from the command line and pass the appropriate parameters. Remember to take note of the resulting BigQuery schema for `NULLABLE` fields. Your code should look something like this:

```
# Set up environment variables export PROJECT_ID=$(gcloud config get-value project) export
REGION={{ {project_0.startup_script.lab_region|Region} }} export BUCKET=gs://{PROJECT_ID} export COLDLINE_BUCKET=${BUCKET}-
coldline export PIPELINE_FOLDER=${BUCKET} export RUNNER=DataflowRunner export
INPUT_PATH=${PIPELINE_FOLDER}/events.json export OUTPUT_PATH=${PIPELINE_FOLDER}-coldline/pipeline_output export
TABLE_NAME=${PROJECT_ID}:logs.logs_filtered cd $BASE_DIR python3 my_pipeline.py \ --project=${PROJECT_ID} \ --
region=${REGION} \ --stagingLocation=${PIPELINE_FOLDER}/staging \ --tempLocation=${PIPELINE_FOLDER}/temp \ --
runner=${RUNNER} \ --inputPath=${INPUT_PATH} \ --outputPath=${OUTPUT_PATH} \ --tableName=${TABLE_NAME} Note: If your
pipeline is building successfully, but you're seeing a lot of errors due to code or misconfiguration in the Dataflow service, you can set `runner` back
to 'DirectRunner' to run it locally and receive faster feedback. This approach works in this case because the dataset is small and you are not using any
features that aren't supported by DirectRunner.
```

Task 7. Verify the pipeline's results

1. Navigate to the [Cloud Dataflow Jobs page](#) and look at the job as it's running. Its graph should resemble the following:



2. Click on the node representing your `Filter` function, which in the above picture is called `FilterFn`. In the panel that appears on the right-hand side, you should see that more elements were added as inputs than were written as outputs.
3. Now click on the node representing the write to Cloud Storage. Since all elements were written, this number should agree with the number of elements in the input to the Filter function.
4. Once the pipeline has finished, examine the results in [BigQuery](#) by querying your table. Note that the number of records in the table should agree with the number of elements that were output by the Filter function.

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

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