Stream Processing with Cloud Pub/Sub and Dataflow: Python

Create Project Resources

In Cloud Shell, create variables for your bucket, project, and region.

PROJECT\_ID=$(gcloud config get-value project)

BUCKET\_NAME=$PROJECT\_ID

TOPIC\_ID=my-id

REGION=us-central1

AE\_REGION=us-central

Cloud Storage bucket names must be globally unique. Your Qwiklabs Project ID is always unique, so that is used for your bucket name in this lab.

Create a Cloud Storage bucket owned by this project:

gsutil mb gs://$BUCKET\_NAME

Create a Pub/Sub topic in this project:

gcloud pubsub topics create $TOPIC\_ID

Create an App Engine app for your project:

gcloud app create --region=$AE\_REGION

Create a Cloud Scheduler job in this project. The job publishes a message to a Pub/Sub topic at one-minute intervals.

gcloud scheduler jobs create pubsub publisher-job --schedule="\* \* \* \* \*" \

--topic=$TOPIC\_ID --message-body="Hello!"

If prompted to enable the Cloud Scheduler API, press y and enter.

Start the job.

gcloud scheduler jobs run publisher-job

Use the following commands to clone the quickstart repository and navigate to the sample code directory:

docker run -it -e DEVSHELL\_PROJECT\_ID=$DEVSHELL\_PROJECT\_ID python:3.7 /bin/bash

git clone <https://github.com/GoogleCloudPlatform/python-docs-samples.git>

cd python-docs-samples/pubsub/streaming-analytics

pip install -U -r requirements.txt # Install Apache Beam dependencies

**Note**: Execute the python commands individually.

**Stream messages from Pub/Sub to Cloud Storage**

Code Sample

This sample code uses Dataflow to:

* Read Pub/Sub messages.
* Window (or group) messages into fixed-size intervals by publish timestamps.
* Write the messages in each window to files in Cloud Storage.

import argparse

from datetime import datetime

import logging

import random

from apache\_beam import DoFn, GroupByKey, io, ParDo, Pipeline, PTransform, WindowInto, WithKeys

from apache\_beam.options.pipeline\_options import PipelineOptions

from apache\_beam.transforms.window import FixedWindows

class GroupMessagesByFixedWindows(PTransform):

"""A composite transform that groups Pub/Sub messages based on publish time

and outputs a list of tuples, each containing a message and its publish time.

"""

def \_\_init\_\_(self, window\_size, num\_shards=5):

# Set window size to 60 seconds.

self.window\_size = int(window\_size \* 60)

self.num\_shards = num\_shards

def expand(self, pcoll):

return (

pcoll

# Bind window info to each element using element timestamp (or publish time).

| "Window into fixed intervals"

>> WindowInto(FixedWindows(self.window\_size))

| "Add timestamp to windowed elements" >> ParDo(AddTimestamp())

# Assign a random key to each windowed element based on the number of shards.

| "Add key" >> WithKeys(lambda \_: random.randint(0, self.num\_shards - 1))

# Group windowed elements by key. All the elements in the same window must fit

# memory for this. If not, you need to use `beam.util.BatchElements`.

| "Group by key" >> GroupByKey()

)

class AddTimestamp(DoFn):

def process(self, element, publish\_time=DoFn.TimestampParam):

"""Processes each windowed element by extracting the message body and its

publish time into a tuple.

"""

yield (

element.decode("utf-8"),

datetime.utcfromtimestamp(float(publish\_time)).strftime(

"%Y-%m-%d %H:%M:%S.%f"

),

)

class WriteToGCS(DoFn):

def \_\_init\_\_(self, output\_path):

self.output\_path = output\_path

def process(self, key\_value, window=DoFn.WindowParam):

"""Write messages in a batch to Google Cloud Storage."""

ts\_format = "%H:%M"

window\_start = window.start.to\_utc\_datetime().strftime(ts\_format)

window\_end = window.end.to\_utc\_datetime().strftime(ts\_format)

shard\_id, batch = key\_value

filename = "-".join([self.output\_path, window\_start, window\_end, str(shard\_id)])

with io.gcsio.GcsIO().open(filename=filename, mode="w") as f:

for message\_body, publish\_time in batch:

f.write(f"{message\_body},{publish\_time}\n".encode("utf-8"))

def run(input\_topic, output\_path, window\_size=1.0, num\_shards=5, pipeline\_args=None):

# Set `save\_main\_session` to True so DoFns can access globally imported modules.

pipeline\_options = PipelineOptions(

pipeline\_args, streaming=True, save\_main\_session=True

)

with Pipeline(options=pipeline\_options) as pipeline:

(

pipeline

# Because `timestamp\_attribute` is unspecified in `ReadFromPubSub`, Beam

# binds the publish time returned by the Pub/Sub server for each message

# to the element's timestamp parameter, accessible via `DoFn.TimestampParam`.

# https://beam.apache.org/releases/pydoc/current/apache\_beam.io.gcp.pubsub.html#apache\_beam.io.gcp.pubsub.ReadFromPubSub

| "Read from Pub/Sub" >> io.ReadFromPubSub(topic=input\_topic)

| "Window into" >> GroupMessagesByFixedWindows(window\_size, num\_shards)

| "Write to GCS" >> ParDo(WriteToGCS(output\_path))

)

if \_\_name\_\_ == "\_\_main\_\_":

logging.getLogger().setLevel(logging.INFO)

parser = argparse.ArgumentParser()

parser.add\_argument(

"--input\_topic",

help="The Cloud Pub/Sub topic to read from."

'"projects//topics/".',

)

parser.add\_argument(

"--window\_size",

type=float,

default=1.0,

help="Output file's window size in minutes.",

)

parser.add\_argument(

"--output\_path",

help="Path of the output GCS file including the prefix.",

)

parser.add\_argument(

"--num\_shards",

type=int,

default=5,

help="Number of shards to use when writing windowed elements to GCS.",

)

known\_args, pipeline\_args = parser.parse\_known\_args()

run(

known\_args.input\_topic,

known\_args.output\_path,

known\_args.window\_size,

known\_args.num\_shards,

pipeline\_args,

)

To explore the sample code further, visit the respective GitHub pages: [Java](https://github.com/GoogleCloudPlatform/java-docs-samples/blob/HEAD/pubsub/streaming-analytics/src/main/java/com/examples/pubsub/streaming/PubSubToGcs.java), [Python](https://github.com/GoogleCloudPlatform/python-docs-samples/blob/HEAD/pubsub/streaming-analytics/PubSubToGCS.py).

**Start the Pipeline**

To start the pipeline, run the following command:

python PubSubToGCS.py \

--project=gcp\_project\_id \

--region=us-central1 \

--input\_topic=projects/gcp\_project\_id/topics/my-id \

--output\_path=gs://BUCKET\_NAME/samples/output \

--runner=DataflowRunner \

--window\_size=2 \

--num\_shards=2 \

--temp\_location=gs://BUCKET\_NAME/temp

**Note**: Replace gcp\_project\_id and BUCKET\_NAME with the GCP Project ID mentioned in connection details panel.

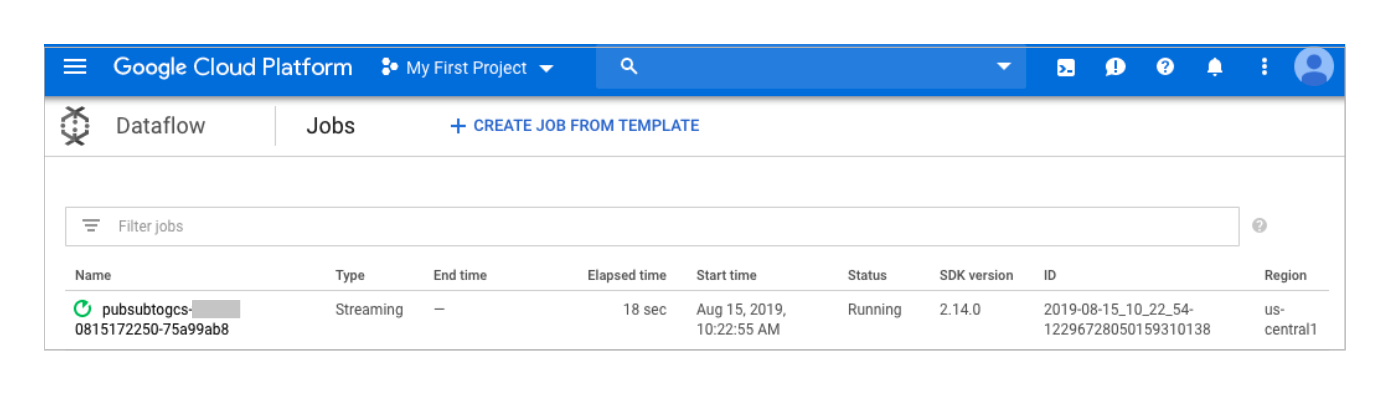
The preceding command runs locally and launches a Dataflow job that runs in the cloud. When the command returns JOB\_MESSAGE\_DETAILED: Workers have started successfully, exit the local program using Ctrl+C.

**Note**: To exit your Python development environment, type and enter exit.

**Observe Job and Pipeline Progress**

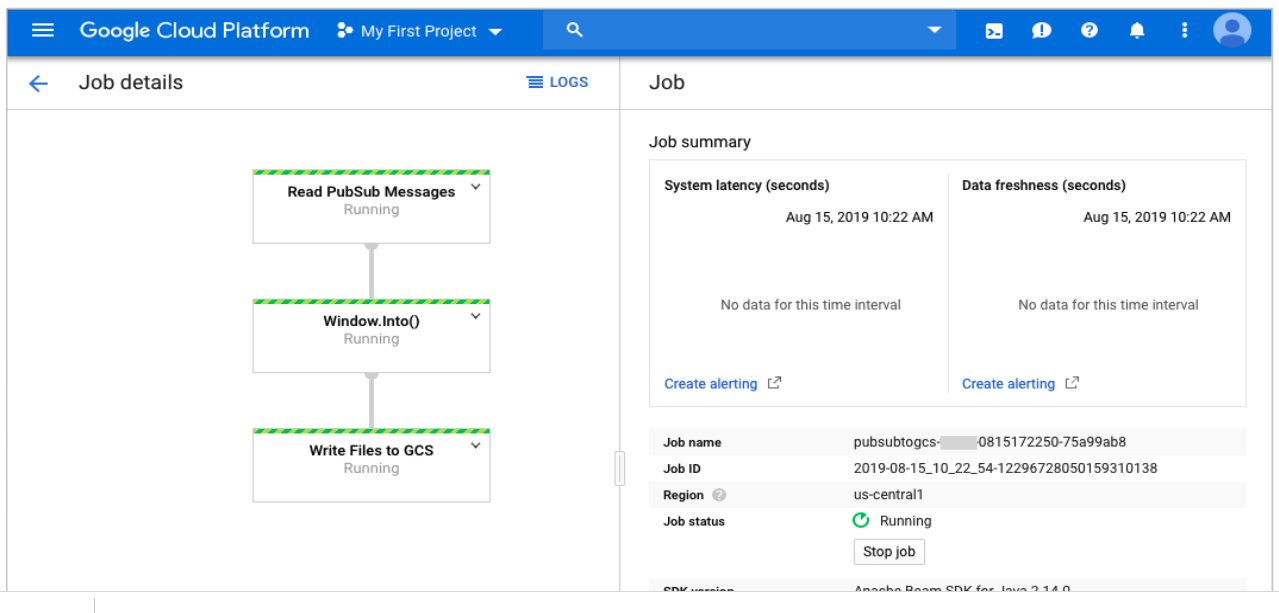
You can observe the job's progress in the Dataflow console.

[Go to the Dataflow console](https://console.cloud.google.com/dataflow?project=).



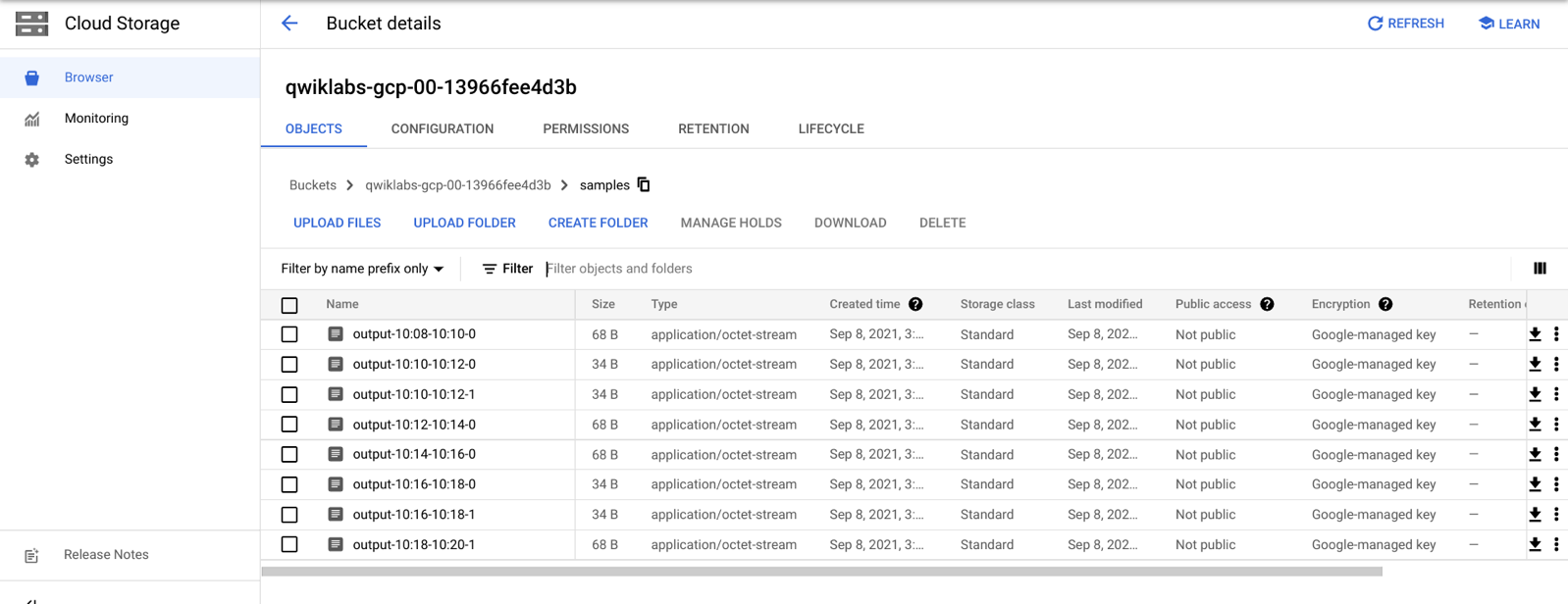
Open the job details view to see:

* Job structure
* Job logs
* Stage metrics



You may have to wait a few minutes to see the output files in Cloud Storage.

You can see the output files by navigating to **Navigation menu** > **Cloud Storage**. Click on your bucket name and then click **Samples**.



Alternatively, use the command line below to check which files have been written out.

gsutil ls gs://${BUCKET\_NAME}/samples/

The output should look like the following:

gs://{$BUCKET\_NAME}/samples/output-22:30-22:32-0-of-1

gs://{$BUCKET\_NAME}/samples/output-22:32-22:34-0-of-1

gs://{$BUCKET\_NAME}/samples/output-22:34-22:36-0-of-1

gs://{$BUCKET\_NAME}/samples/output-22:36-22:38-0-of-1

**Cleanup**

Delete the Cloud Scheduler job:

gcloud scheduler jobs delete publisher-job

If prompted, do you want to continue press Y and enter.

Press ctrl + c in your Cloud Shell if it's still busy printing output of your Dataflow job.

In the Dataflow console, stop the job.

With your job selected from the Dataflow Console, press the **Stop** button. Select the **Cancel** bubble to cancel the pipeline without draining.

Delete the topic:

gcloud pubsub topics delete $TOPIC\_ID

Delete the files created by the pipeline:

gsutil -m rm -rf "gs://${BUCKET\_NAME}/samples/output\*"

gsutil -m rm -rf "gs://${BUCKET\_NAME}/temp/\*"

Remove the Cloud Storage bucket:

gsutil rb gs://${BUCKET\_NAME}