Stream Processing with Cloud Pub/Sub and Dataflow: Java

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:

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

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

**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 java.io.IOException;

import org.apache.beam.examples.common.WriteOneFilePerWindow;

import org.apache.beam.sdk.Pipeline;

import org.apache.beam.sdk.io.gcp.pubsub.PubsubIO;

import org.apache.beam.sdk.options.Default;

import org.apache.beam.sdk.options.Description;

import org.apache.beam.sdk.options.PipelineOptions;

import org.apache.beam.sdk.options.PipelineOptionsFactory;

import org.apache.beam.sdk.options.StreamingOptions;

import org.apache.beam.sdk.options.Validation.Required;

import org.apache.beam.sdk.transforms.windowing.FixedWindows;

import org.apache.beam.sdk.transforms.windowing.Window;

import org.joda.time.Duration;

public class PubSubToGcs {

\* Define your own configuration options. Add your own arguments to be processed

\* by the command-line parser, and specify default values for them.

public interface PubSubToGcsOptions extends PipelineOptions, StreamingOptions {

@Description("The Cloud Pub/Sub topic to read from.")

@Required

String getInputTopic();

void setInputTopic(String value);

@Description("Output file's window size in number of minutes.")

@Default.Integer(1)

Integer getWindowSize();

void setWindowSize(Integer value);

@Description("Path of the output file including its filename prefix.")

@Required

String getOutput();

void setOutput(String value);

}

public static void main(String[] args) throws IOException {

// The maximum number of shards when writing output.

int numShards = 1;

PubSubToGcsOptions options =

PipelineOptionsFactory.fromArgs(args).withValidation().as(PubSubToGcsOptions.class);

options.setStreaming(true);

Pipeline pipeline = Pipeline.create(options);

pipeline

// 1) Read string messages from a Pub/Sub topic.

.apply("Read PubSub Messages", PubsubIO.readStrings().fromTopic(options.getInputTopic()))

// 2) Group the messages into fixed-sized minute intervals.

.apply(Window.into(FixedWindows.of(Duration.standardMinutes(options.getWindowSize()))))

// 3) Write one file to GCS for every window of messages.

.apply("Write Files to GCS", new WriteOneFilePerWindow(options.getOutput(), numShards));

// Execute the pipeline and wait until it finishes running.

pipeline.run().waitUntilFinish();

}

}

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:

mvn compile exec:java \

-Dexec.mainClass=com.examples.pubsub.streaming.PubSubToGcs \

-Dexec.cleanupDaemonThreads=false \

-Dexec.args=" \

--project=$PROJECT\_ID \

--region=$REGION \

--inputTopic=projects/$PROJECT\_ID/topics/$TOPIC\_ID \

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

--runner=DataflowRunner \

--windowSize=2"

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

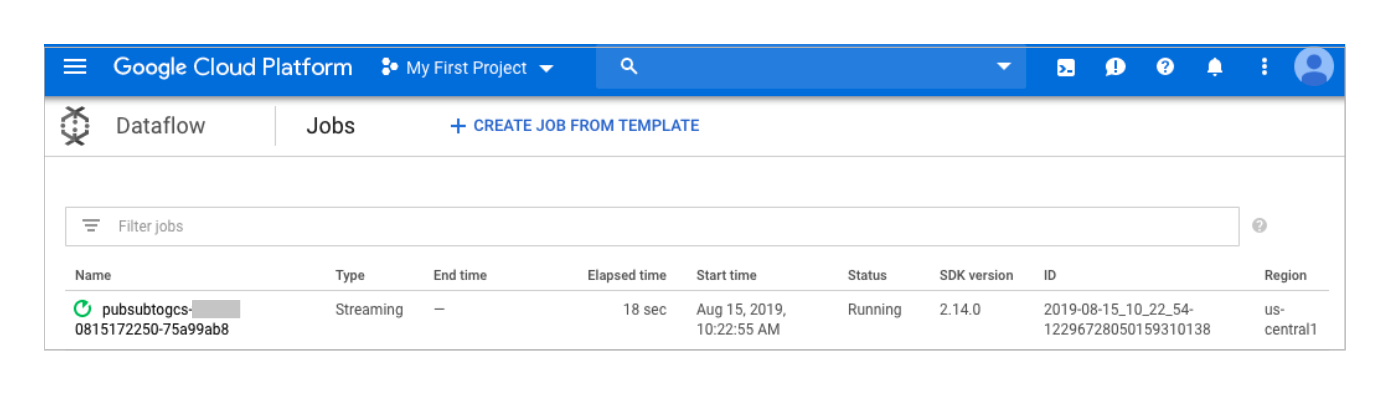
The preceding command runs locally and launches a Dataflow job that runs in the cloud. When the command returns JOB\_MESSAGE\_DETAILED: Workers have started successfully, exit the local program using Ctrl+C.

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

**Observe Job and Pipeline Progress**

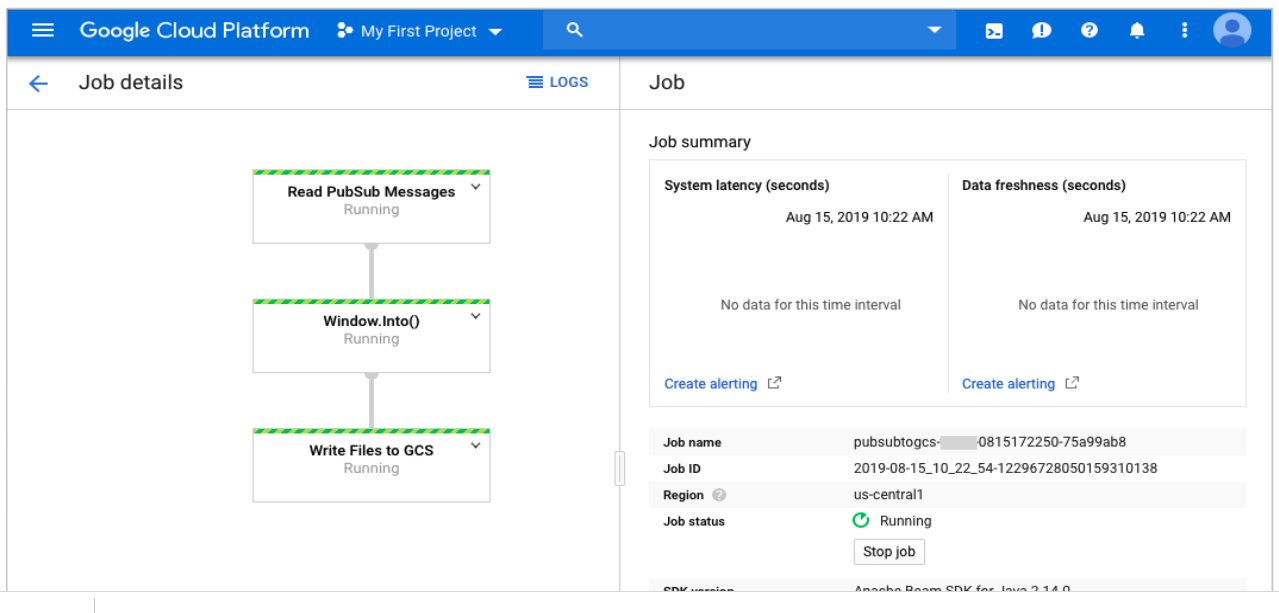
You can observe the job's progress in the Dataflow console.

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



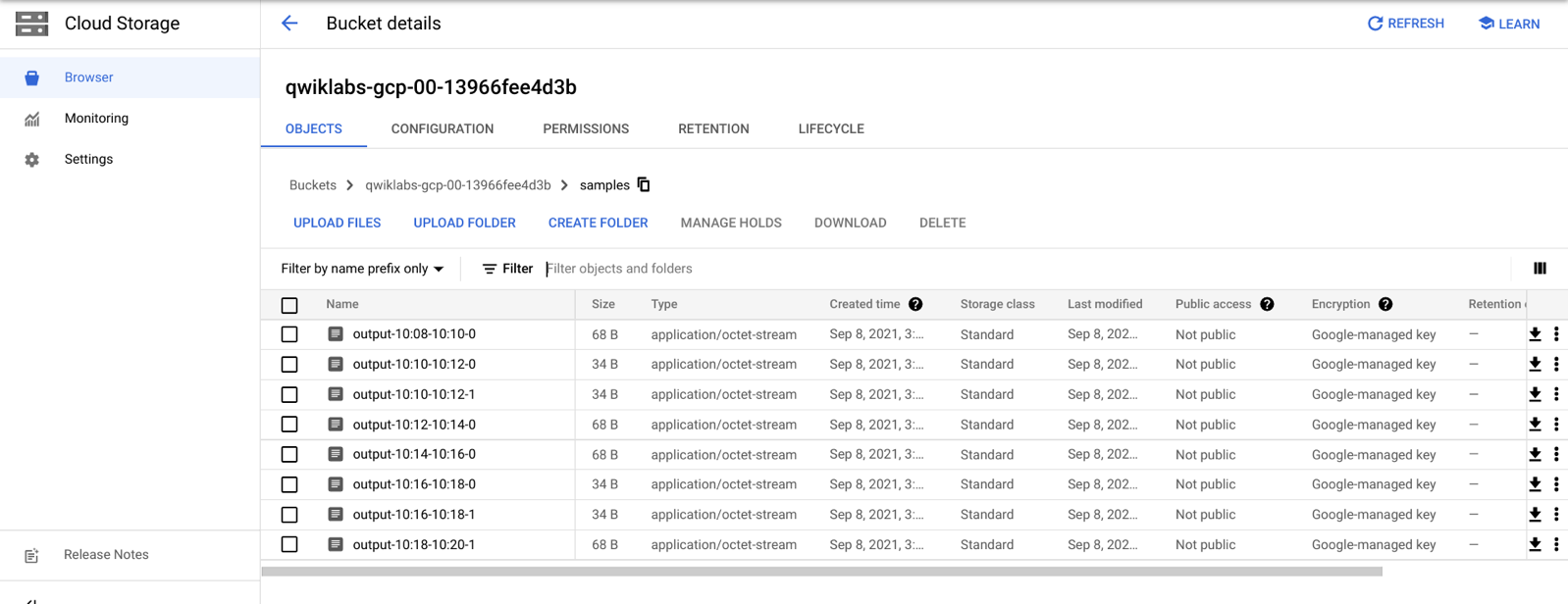
Open the job details view to see:

* 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}