## **Dataflow: Dataflow has autoscaling feature**

instance **n1-standard-1** is low configuration, and you can upgrade for improving the performance.

This topic was technical and if you don’t have experience working with Dataflow it may be a little bit tricky.

* How to discard erroneous data and for example sent it to Pub/Sub or Cloud Storage
* Transform, DoFn, Sideinputs, Sideoutputs.
* IAM Roles for Developers and how to secure the data.
* Windows, all kinds of them. There were 3 questions about Sliding time windows, Session time windows, and the best way to deal with late data.
* Bounded and unbounded data.
* How to connect with Pub/Sub, BigQuery, BigTable, etc.
* Integration with IAM roles, especially the developer role
* Differences between global, fixed, session, and sliding windows and when to use each type
* Best practices on handling pipeline errors, especially, try-catch-block errors
* Different types of transform methods, for example, Apache Beam ParDo
* Watermarks
* Apache Beam

Dataflow can connect with Kafka and sliding window is used for taking averages.

Google Cloud Dataflow provides the open Apache Beam programming model as a managed service for processing data in multiple ways, including batch operations, extract-transform-load (ETL) patterns, and continuous, streaming computation. Cloud Dataflow can be particularly useful for managing the high-volume data processing pipelines required for IoT scenarios. Cloud Dataflow is also designed to integrate seamlessly with the other Cloud Platform services you choose for your pipeline.

There is no Cloud Dataflow-specific cross pipeline communication mechanism for sharing data or **processing context between pipelines**. You can use durable storage like Cloud Storage or an in-memory cache like App Engine to share data between pipeline instances.

Dataflow does not have a cross pipeline communication mechanism for sharing data between pipelines.

Sharing data is not possible **unless using reliable data storage such as Google Storage**.

Dataflow is a unified processing model and can execute both streaming and batch data pipelines.

Dataflow pipelines can also run-on alternate runtimes like Spark and Flink, as they are built using the Apache Beam SDKs

Reference: <https://cloud.google.com/dataflow/>

**Dataflow developer role** would help provide the third-party consultant access to create and work on the Dataflow pipeline. However, it does not provide access to view the data, thus maintaining user’s privacy.

**PCollection** indicates it is a Cloud Dataflow pipeline. And the Cloud Runner will enable the pipeline to scale to production levels.

A **transform** represents a processing operation that transforms data. A transform takes one or more PCollections as input, performs an operation that you specify on each element in that collection, and produces one or more PCollections as output. A transform can perform nearly any kind of processing operation, including performing mathematical computations on data, converting data from one format to another, grouping data together, reading and writing data, filtering data to output only the elements you want, or combining data elements into single values.

Reference(s):  
<https://cloud.google.com/dataflow/docs/concepts/beam-programming-model>

When the requirement is not to lose the data; the Dataflow pipeline can be stopped using the **Drain** option. Drain options would cause Dataflow to stop any new processing, but would also allow the existing processing to complete

Using the Drain option to stop your job tells the Cloud Dataflow service to finish your job in its current state. Your job will immediately stop ingesting new data from input sources. However, the Cloud Dataflow service will preserve any existing resources, such as worker instances, to finish processing and writing any buffered data in your pipeline. When all pending processing and writing operations are complete, the Cloud Dataflow service will clean up the GCP resources associated with your job.

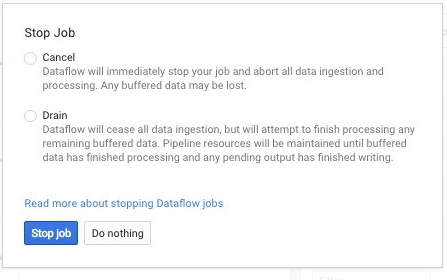
Dataflow stopping a pipeline: - <https://cloud.google.com/dataflow/docs/guides/stopping-a-pipeline>

**Note**: Your pipeline will continue to incur the cost of maintaining any associated GCP resources until all processing and writing has been completed.  
Use the Drain option to stop your job if you want to prevent data loss as you bring down your pipeline.

**Effects of draining a job**:  
When you issue the Drain command, Cloud Dataflow immediately closes any in-process windows and fires all triggers. The system does not wait for any outstanding time-based windows to finish. For example, if your pipeline is ten minutes into a two-hour window when you issue the Drain command, Cloud Dataflow won’t wait for the remainder of the window to finish. It will close the window immediately with partial results.

Drain command helps Dataflow process and complete in-flight messages and stops accepting any new ones.

If you need to stop running a Cloud Dataflow job, you can do so by issuing a command using either the Cloud Dataflow Monitoring Interface or the Cloud Dataflow Command-line Interface. There are two possible commands you can issue to stop your job: Cancel and Drain.

  
**Note**: The Drain command is supported for streaming pipelines only.

**Triggers** control when the elements for a specific key and window are output. As elements arrive, they are put into one or more windows by a Window transform and its associated WindowFn, and then passed to the associated Trigger to determine if the Windows contents should be output.  
Reference: <https://cloud.google.com/dataflow/java-sdk/JavaDoc/com/google/cloud/dataflow/sdk/transforms/windowing/Trigger>

There are **three** major kinds of triggers that Dataflow supports: 1. **Time-based triggers** 2. **Data-driven triggers.** You can set a trigger to emit results from a window when that window has received a certain number of data elements. 3. **Composite triggers**. These triggers combine multiple time-based or data-driven triggers in some logical way  
Reference: <https://cloud.google.com/dataflow/model/triggers>

When collecting and grouping data into windows, Beam uses triggers to determine when to emit the aggregated results of each window.  
Processing time triggers. These triggers operate on the processing time the time when the data element is processed at any given stage in the pipeline.  
Event time triggers. These triggers operate on the event time, as indicated by the timestamp on each data element. **Beams default trigger is event time-based**.  
Reference: <https://beam.apache.org/documentation/programming-guide/#triggers>

In Google Cloud dataflow SDK, you can use the **ParDo** to extract only a customer name of each element in your PCollection.

Which Java SDK class can you use to run your Dataflow programs locally?

DirectPipelineRunner allows you to execute operations in the pipeline directly, without any optimization. Useful for small local execution and tests  
Reference: <https://cloud.google.com/dataflow/java-sdk/JavaDoc/com/google/cloud/dataflow/sdk/runners/DirectPipelineRunner>

In Google Cloud, the Dataflow SDK provides a transform component. It is responsible for the data processing operation. You can use conditional, for loops, and other complex programming structures to create a branching pipeline.  
Reference: <https://cloud.google.com/dataflow/model/programming-model>

The dataflow.worker role provides the permissions necessary for a Compute Engine service account to execute work units for a Dataflow pipeline  
Reference: <https://cloud.google.com/dataflow/access-control>

The data and transforms in a pipeline are unique to, and owned by, that pipeline. While your program can create multiple pipelines, pipelines cannot share data or transforms  
Reference: <https://cloud.google.com/dataflow/model/pipelines>

Dataflow’s default windowing behavior is to assign all elements of a PCollection to a **single, global window, even for unbounded PCollections**  
Reference: <https://cloud.google.com/dataflow/model/pcollection>

A **sliding time window** uses time intervals in the data stream to define bundles of data. However, with sliding time windowing, the windows overlap. Each window might capture five minutes’ worth of data, but a new window starts every ten seconds. The frequency with which sliding windows begin is called the period. Therefore, our example would have a window size of five minutes and a period of ten seconds.  
The sliding-time window is the windowing function recommended for this scenario.

<https://cloud.google.com/dataflow/model/windowing#windowing-functions>

A session window function defines windows around areas of concentration in the data. Session windowing is useful for data that is irregularly distributed with respect to time; for example, a data stream representing user mouse activity may have long periods of idle time interspersed with high concentrations of clicks. Session windowing groups the high concentrations of data into separate windows and filters out the idle sections of the data stream. Note that session windowing applies on a per-key basis: That is, grouping into sessions only takes into account data that has the same key. Each key in your data collection will, therefore, be grouped into disjoint windows of differing sizes.  
For this scenario, the per-session window is the function to choose to build a Dataflow pipeline.

Dataflow does not help in solving the increasing API requests.

Dataflow cannot ingest event streams. It needs Pub/Sub service to do so

*use hopping windows to compute moving averages.*