

01 Dataflow Intro - KirkYagami



Dataflow

<https://cloud.google.com/dataflow/docs/horizontal-autoscaling>

Course: # Serverless Data Processing with Dataflow: Operations:

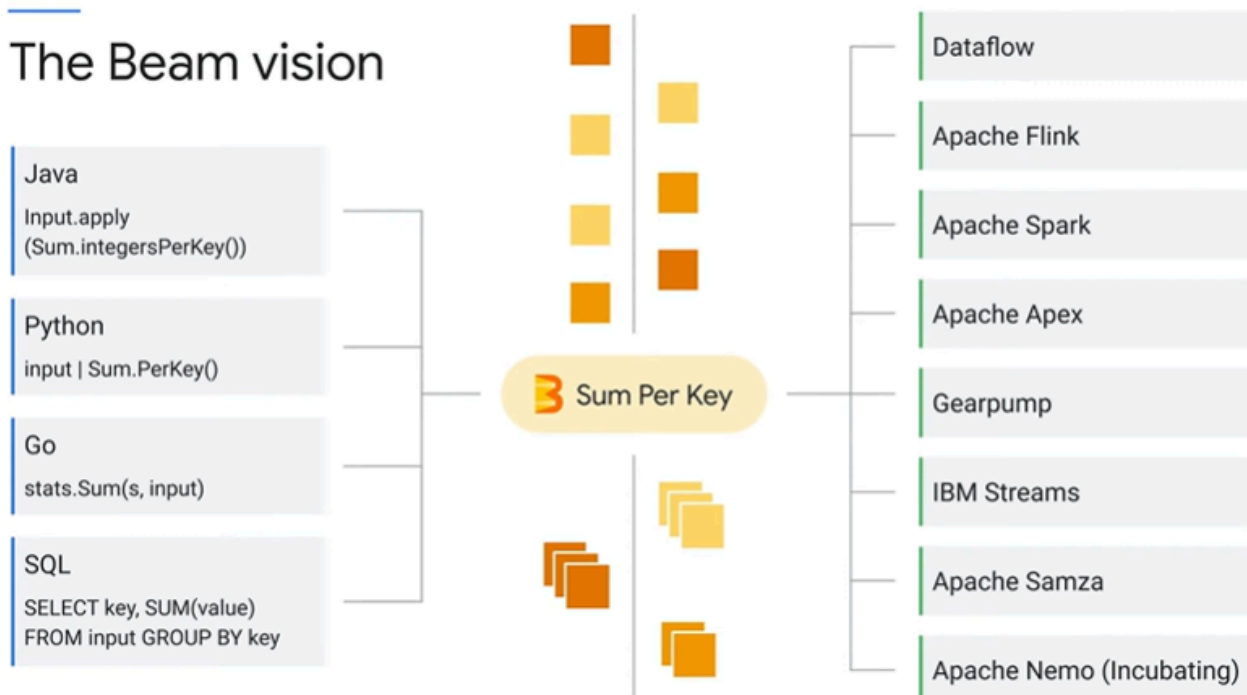
https://www.cloudskillsboost.google/course_templates/264

Refer to this article <https://medium.com/@sasidharan01/streaming-data-with-dataflow-understanding-watermarks-custom-triggers-and-data-accumulation-f4275fe4c883>

You can refer to [this](#) for docs.

- ◆ Cloud version for *Apache Beam*.
- ◆ Unified API for Batch and Streaming data and run same code on execution frameworks like Spark, Flink, Apex, Cloud dataflow, etc.
- ◆ Preferred for new Hadoop or Spark infrastructure development
- ◆ Process batch or stream data.
- ◆ Serverless, fast, scalable, fault-tolerant.
- ◆ Multi-Step processing data. Eg: Wordcount.
- ◆ Cloud dataflow executed as jobs where one or more worker carry out specific tasks
- ◆ Supports SQL, Java, Python
- ◆ When you run a job on Cloud Dataflow, it spins up a cluster of virtual machines, distributes the tasks in your job to the VMs, and dynamically scales the cluster based on how the job is performing.
- ◆ Stackdriver integration for logging and monitoring
- ◆ Used for data processing, ELT(Extract Transform Load), filter, group for data sets.
- ◆ Can read data from multiple sources, can kick off multiple cloud functions in parallel and can also writer to multiple sinks (like BigQuery, BigTable etc.)
- ◆ When you update a job on the Dataflow service, you replace the existing job with a new job that runs your updated pipeline code
- ◆ Jobs can be created with inbuilt templates, or notebook instances(write jobs in Java/Python/SQL)

- ◆ 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 structure to create a branching pipeline.
- ◆ Can apply [windowing](#) to streams for rolling average for the window, max in a window etc.
- ◆ You can stop a Dataflow job in the following two ways:
 - ◆ Canceling a job. This method applies to both streaming and batch pipelines. Canceling a job stops the Dataflow service from processing any data, including buffered data
 - ◆ **Draining a job.** This method applies only to streaming pipelines. Draining a job enables the Dataflow service to finish processing the buffered data while simultaneously ceasing the ingestion of new data.
- ◆ Updating the pipeline:
 - ◆ If any major change to windowing transformation (like completely changing window fn from fixed to sliding) in Beam/Dataflow/you want to stop pipeline but want inflight data -> use Drain option.
 - ◆ For all other use cases and Minor changing to windowing fn (like just changing window time of sliding window) --> Use Update with Json mapping.
- ◆ Important IAM roles -
 - ◆ **dataflow.developer** role enable the developer interacting with the Cloud Dataflow job , with data privacy.
 - ◆ **dataflow.worker** role provides the permissions necessary for a *Compute Engine service account* to execute work units for a Dataflow pipeline



Common Concepts in Dataflow

- ◆ **Pipeline:** encapsulates series of computations that accepts input data from external sources, transforms data to provide some useful intelligence, and produce output
- ◆ **PCollections:** abstraction that represents a potentially distributed, multi-element data set, that acts as the pipeline's data. PCollection objects represent input, intermediate, and output data. The edges of the pipeline.
- ◆ **Transforms:** operations in pipeline. A transform takes a PCollection(s) as input, performs an operation that you specify on each element in that collection, and produces a new output PCollection. Composite transforms are multiple transforms: combining, mapping, shuffling, reducing, or statistical analysis.
- ◆ **Pipeline I/O:** the source/sink, where the data flows in and out. Supports read and write transforms for a number of common data storage types, as well as custom.
- ◆ **Windowing:** Windowing a PCollection divides the elements into windows based on the associated event time for each element.
- ◆ **Triggers:** Allows specifying a trigger to control when (in processing time) results for the given window can be produced. Triggers determines when a Window's contents should be output based on certain criteria being met. Types of triggers are :
 - ◆ Time based triggers
 - ◆ Data Driven triggers
 - ◆ Composite triggers
- ◆ **Watermark:** It is a threshold that indicates when Dataflow expects all of the data in a window to have arrived. If new data arrives with a timestamp that's in the window but older than the watermark, the data is considered **late data**.
 - ◆ Watermark is a heuristic that tracks how far behind the system is in processing data from the event time. Where in event time does processing occur?
- ◆ **ParDo:** It is a parallel processing function which can transform elements of an input PCollection to an output PCollection.
 - ◆ ParDo is a common intermediate step in a pipeline
 - ◆ You might use it to extract certain fields from a set of raw input records or convert raw input into a different format.
 - ◆ You might also use ParDo to convert process data into an output format, like table rows for BigQuery or strings for printing.
 - ◆ You can use ParDo to consider each element in a PCollection and either output that element to a new collection or discard it.
 - ◆ If your input PCollection contains elements that are of a different type or format than you want, you can use ParDo to perform a conversion on each element and output the

result to a new PCollection.

- ◆ **DoFn:** It is a template which is used to create user defined functions that are referenced by ParDo. A DoFn is a Beam SDK class that defines a distributed processing function.

Quotas

Persistent Disks

Python: Use the `--worker_disk_type` flag

```
$ python3 -m apache_beam.examples.wordcount \
  --input gs://dataflow-samples/shakespeare/kinglear.txt \
  --output gs://$BUCKET/results/outputs --runner DataflowRunner \
  --project $PROJECT --temp_location gs://$BUCKET/tmp/ --region $REGION \
  --worker_disk_type compute.googleapis.com/projects/$PROJECT/zones/$ZONE/diskTypes/pd-ssd
```

Java: Use the `--workerDiskType` flag

```
$ gradle clean execute -DmainClass=org.apache.beam.examples.WordCount -Dexec.args="\
  --inputFile=gs://apache-beam-samples/shakespeare/kinglear.txt \
  --output=gs://$BUCKET/results/outputs --runner=DataflowRunner \
  --project=$PROJECT --tempLocation=gs://$BUCKET/tmp/ --region=$REGION \
  --workerDiskType=compute.googleapis.com/projects/$PROJECT/zones/$ZONE/diskTypes/pd-ssd"
```

Quotas

Persistent Disks - Batch Pipeline

- VM to PD ratio is 1:1 for Batch
- Size if Shuffle on VM: 250 GB
- Size if Shuffle Service: 25 GB
- Flag to override default:

Python: `--disk_size_gb`

Java: `--diskSizeGb`



Quotas

Persistent Disks - Streaming Pipeline

- Fixed number of PDs
- Default size if shuffle on VM: 400 GB
- Default size if Streaming Engine: 30 GB
- Flag to override default:
Python: `--disk_size_gb`
Java: `--diskSizeGb`



Quotas

Persistent Disks - Streaming Pipeline

- Amount of disk allocated == Maximum number of workers
- Flag to set maximum number of workers:
Python: `--max_num_workers`
Java: `--maxNumWorkers`
- Flag required for streaming with shuffle on VMs
- Maximum number of workers that can be launched is 1000



1. Your project's current SSD usage is 100 TB. You want to launch a streaming pipeline with shuffle done on the VM. You set the initial number of workers to 5 and the maximum number of workers to 100. What will be your project's SSD usage when the job launches?

102 TB

500 TB

103 TB

140 TB

*The number of disks allocated equals the maximum number of workers in streaming pipelines. When shuffle is done on the VM, the default PD size is 400 GB. Doing $100 + (0.4 \text{ TB} * 100 \text{ workers})$ gives you 140 TB.*

- ✓ 2. You want to run the following command:

```
gcloud dataflow jobs cancel 2021-01-31_14_30_00-9098096469011826084--region=$REGION
```

Which of these roles can be assigned to you for the command to work?

☐ Composer Worker

✓ ☒ Dataflow Admin

This role provides access for creating and managing Dataflow jobs.

☐ Dataflow Viewer

✓ ☒ Dataflow Developer

This role provides access to view, update, and cancel Dataflow jobs.



Why Dataflow is Preferred

- ◆ **Serverless:** Dataflow eliminates the need to manage clusters, simplifying setup and maintenance.
- ◆ **Auto-scaling:** Dataflow automatically scales resources up or down based on workload, ensuring efficient resource utilization.
- ◆ **Unified Batch and Stream Processing:** Dataflow allows using the same code for both batch and streaming data pipelines.

Dataproc as an Alternative

- ◆ **Existing Hadoop Workloads:** Dataproc is suitable for migrating existing Hadoop or Spark-based pipelines to Google Cloud.
- ◆ **DevOps Approach:** Dataproc caters to teams comfortable with a traditional DevOps approach involving machine provisioning.
- ◆ **Focus on Batch Processing:** Dataproc is a good choice if the primary goal is moving existing batch workloads without a need for streaming capabilities.

Choosing the Right Tool

The decision depends on various factors:

- ◆ Project requirements (batch vs. streaming)
- ◆ Existing codebase (Hadoop/Spark vs. Beam)
- ◆ Team expertise and preferences

Key Concepts of Beam Programming Model

- ◆ **PTransform:** Represents an operation that processes data within a pipeline.
- ◆ **PCollection:** Represents a distributed data set that can be either bounded (batch) or unbounded (stream).
- ◆ **Pipeline:** Defines the sequence of PTransforms to be applied on the data.
- ◆ **Pipeline Runner:** Executes the pipeline on a specific platform (local, cloud service, etc.).

Benefits of Beam and PCollections



- ◆ **Immutability:** PCollections are immutable, simplifying distributed processing and eliminating the need for complex access control.
- ◆ **Flexibility:** Beam pipelines can be visualized as directed acyclic graphs (DAGs) for complex data processing workflows.
- ◆ **Unified Data Representation:** PCollections can handle both batch and streaming data, offering a unified approach.

Data Processing with PCollections

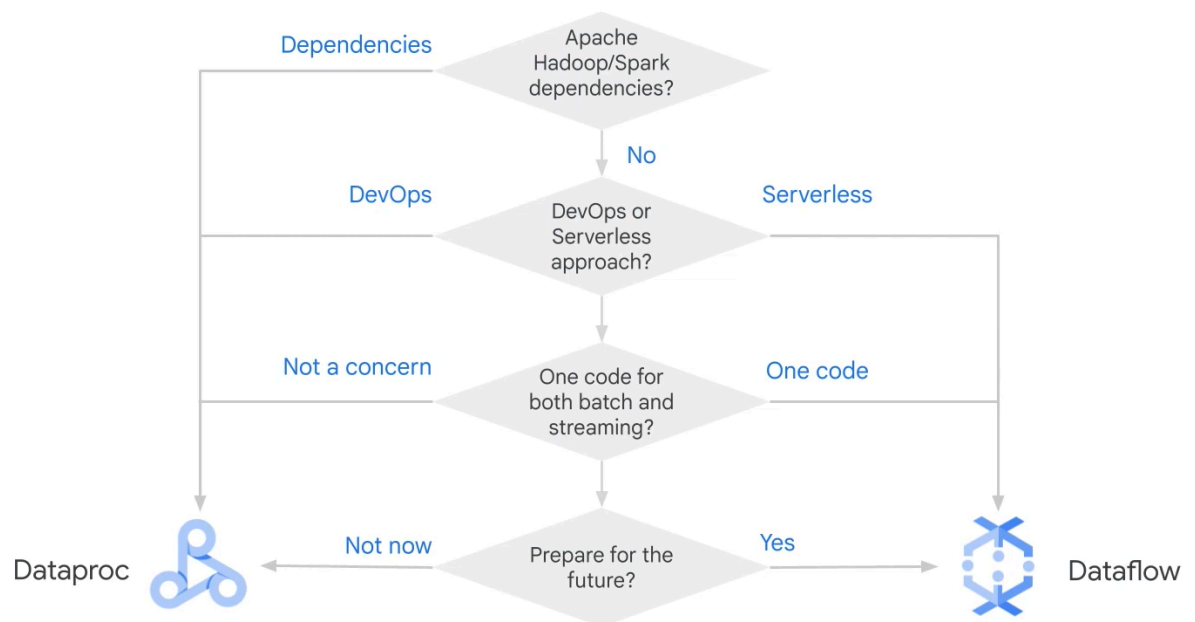
- ◆ PCollections store data as serialized byte strings, enabling efficient network transfers and minimizing processing overhead.
- ◆ Individual elements within a PCollection can be accessed and processed independently, facilitating distributed execution.

In summary, Dataflow with Beam is the recommended approach for building new data pipelines on Google Cloud due to its serverless nature, scalability, and unified batch/stream processing capabilities. Dataproc remains a viable option for migrating existing Hadoop workloads or when a traditional DevOps approach is preferred.

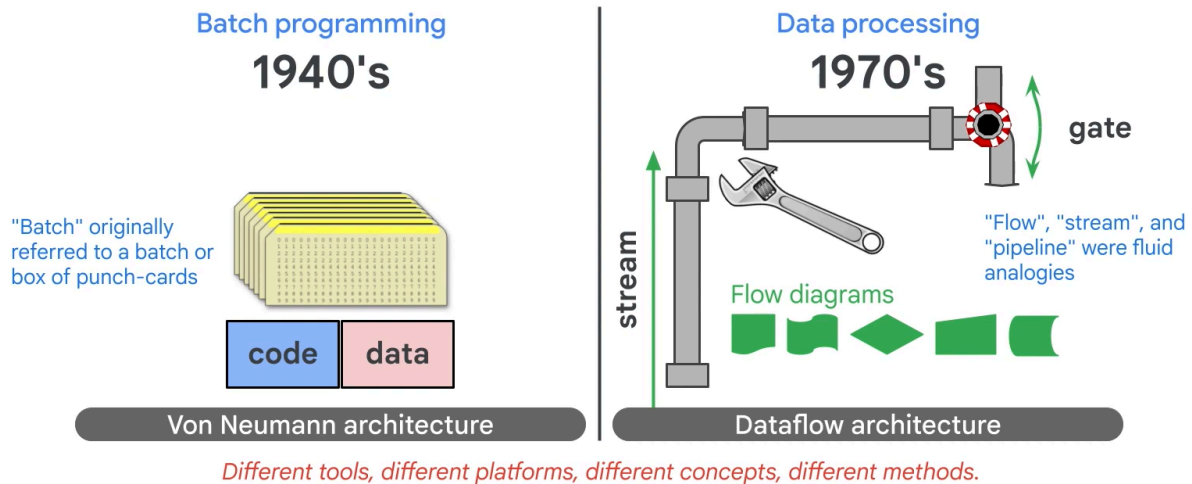
Dataflow versus Dataproc

	 Dataflow	 Dataproc
Recommended for:	New data processing pipelines, unified batch and streaming	Existing Hadoop/Spark applications, machine learning/data science ecosystem, large-batch jobs, preemptible VMs
Serverless:	Yes	No
Auto-scaling:	Yes, transform-by-transform (adaptive)	Yes, based on cluster utilization (reactive)
Expertise:	Apache Beam	Hadoop, Hive, Pig, Apache Big Data ecosystem, Spark, Flink, Presto, Druid

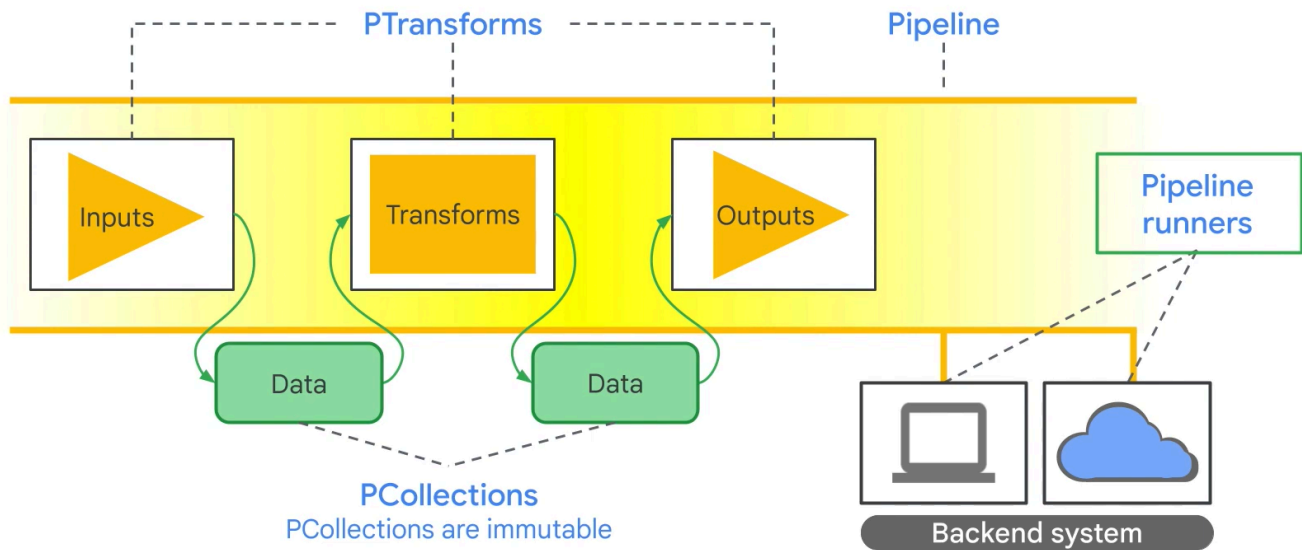
Choosing between Dataflow and Dataproc



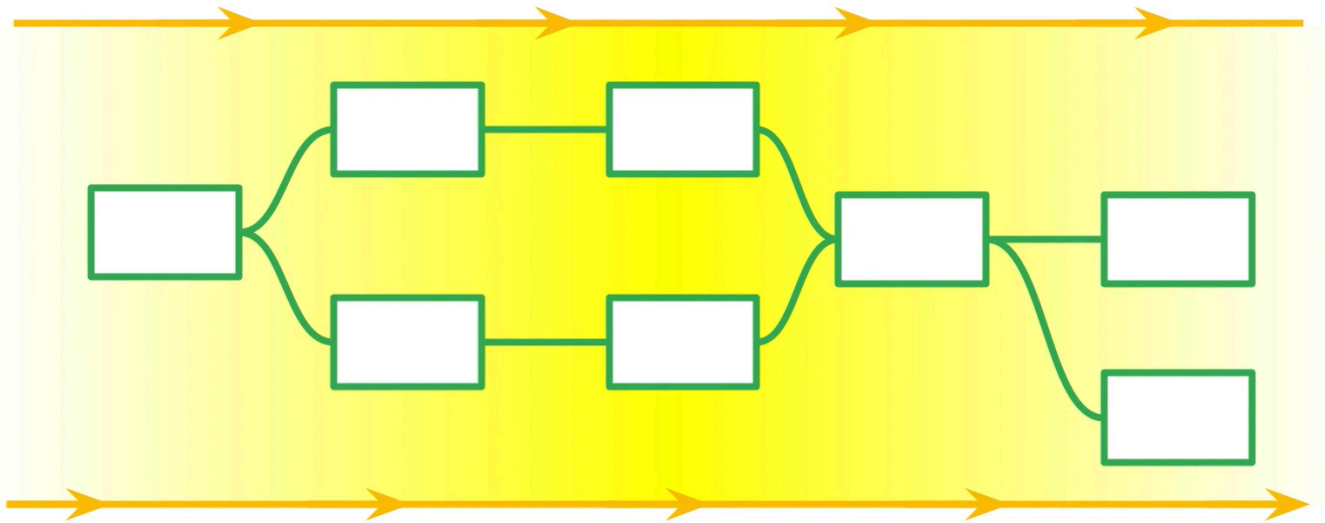
Batch programming and data processing used to be two very separate and different things



Apache BEAM = Batch + strEAM

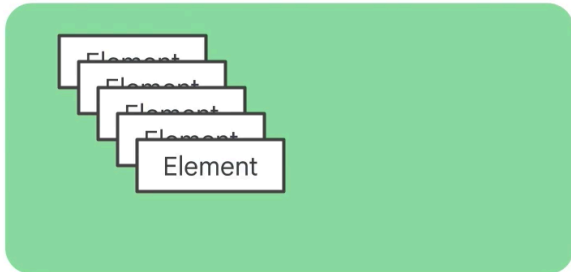


A Dataflow pipeline is a directed graph of steps



Each PCollection element can be distributed for parallel processing

Bounded PCollection



All data types are stored as serialized byte strings

Unbounded PCollection



Note: Bounded means the data has a fixed size not that the PCollection size is limited. A PCollection can be any size and be distributed across many workers.