**Apache Beam basics**

The following concepts are used throughout the notebooks:

* **Element**: minimal unit of data.
* **PCollection**: represents a distribute data set; it can be *bounded* or *unbounded*. Made of element(s).
  + ***Bounded* PCollection** is data that has a fixed size. For example, text files, BigQuery tables, Avro files, and so on.
  + ***Unbounded* PCollections** are potentially of infinite size, coming from a data stream. Examples of this are Pub/Sub topic/subscription and Kafka.

Before running into code, examine the basic structure for creating a pipeline:

* At the beginning, to define your pipeline, use p = beam.Pipeline().
* The pipe | separates steps within the pipeline. Every time you want to add a new step, you need a new pipe.
* At the right of the pipe, add the step you want to execute, | <STEP>. You can optionally name the step using >> between the step and the pipe  | "NAME" >> <STEP>. Two steps cannot have the same name.
* At the left of the pipe, there has to be a reference to a pipeline p | <STEP>, p | <STEP1> | <STEP2>... or squares | <STEP> (where squares is a pipeline variable).

## 

## Basic Operations

**Create** is used to create elements.

**Map** does an operation at the element level. Applies a simple one-to-one mapping function over each element in the collection.

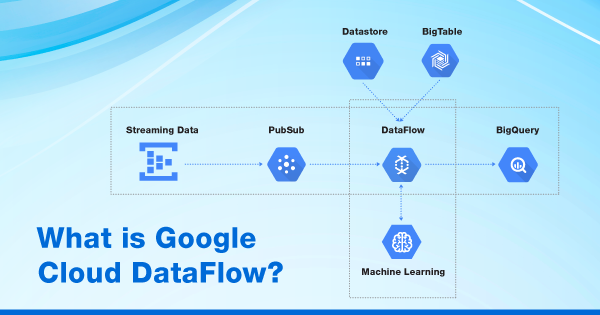
During most of the following examples we are going to use Create as the source in these examples, since it's more intuitive than using CSVs or other sources.

The following pipeline returns the squares of the N first non-negative integers. For this first pipeline, we are going to use the default runner (DirectRunner):

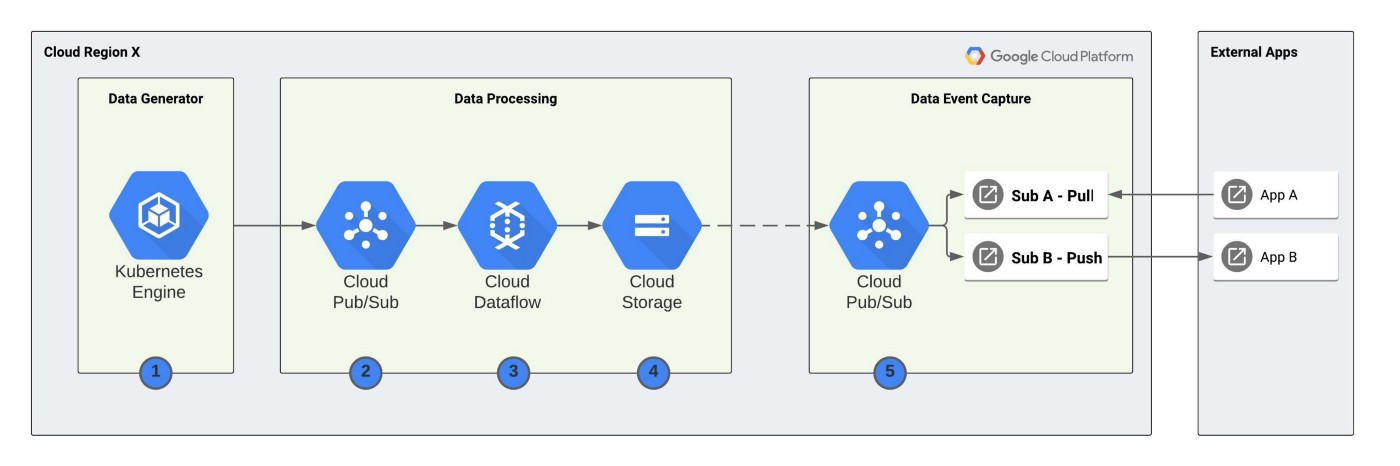
**What can we do with DataFlow:**

* Creating a pipeline
* Deploying a pipeline
* Specifying pipeline execution parameters
* Developing with notebooks
* Get started with Google-provided templates
* Stopping a running pipeline
* Troubleshooting and debugging

**Data Flow Architecture Flow:**



**Dataflow Pipeline:**



### Basic concepts

**Pipelines**

A pipeline encapsulates the entire series of computations involved in reading input data, transforming that data, and writing output data. The input source and output sink can be the same or of different types, allowing you to convert data from one format to another. Apache Beam programs start by constructing a Pipeline object, and then using that object as the basis for creating the pipeline's datasets. Each pipeline represents a single, repeatable job.

**PCollection**

A PCollection represents a potentially distributed, multi-element dataset that acts as the pipeline's data. Apache Beam transforms use PCollection objects as inputs and outputs for each step in your pipeline. A PCollection can hold a dataset of a fixed size or an unbounded dataset from a continuously updating data source.

**Transforms**

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.

**ParDo**

ParDo is the core parallel processing operation in the Apache Beam SDKs, invoking a user-specified function on each of the elements of the input PCollection.

**ParDo** collects the zero or more output elements into an output PCollection. The ParDo transform processes elements independently and possibly in parallel.

**Pipeline I/O**

Apache Beam I/O connectors let you read data into your pipeline and write output data from your pipeline. An I/O connector consists of a source and a sink. All Apache Beam sources and sinks are transforms that let your pipeline work with data from several different data storage formats. You can also write a custom I/O connector.

**Aggregation**

Aggregation is the process of computing some value from multiple input elements. The primary computational pattern for aggregation in Apache Beam is to group all elements with a common key and window. Then, it combines each group of elements using an associative and commutative operation.

**User-defined functions (UDFs)**

Some operations within Apache Beam allow executing user-defined code as a way of configuring the transform. For ParDo, user-defined code specifies the operation to apply to every element, and for Combine, it specifies how values should be combined. A pipeline might contain UDFs written in a different language than the language of your runner. A pipeline might also contain UDFs written in multiple languages.

**Runner**

Runners are the software that accepts a pipeline and executes it. Most runners are translators or adapters to massively parallel big-data processing systems. Other runners exist for local testing and debugging.

**Source**

A transform that reads from an external storage system. A pipeline typically reads input data from a source. The source has a type, which may be different from the sink type, so you can change the format of data as it moves through the pipeline.

**Sink**

A transform that writes to an external data storage system, like a file or a database.

### Advanced concepts

**Event time**

The time a data event occurs, determined by the timestamp on the data element itself. This contrasts with the time the actual data element gets processed at any stage in the pipeline.

**Windowing**

Windowing enables grouping operations over unbounded collections by dividing the collection into windows of finite collections according to the timestamps of the individual elements. A windowing function tells the runner how to assign elements to an initial window, and how to merge windows of grouped elements. Apache Beam lets you define different kinds of windows or use the predefined windowing functions.

**Watermarks**

Apache Beam tracks a watermark, which is the system's notion of when all data in a certain window can be expected to have arrived in the pipeline. Apache Beam tracks a watermark because data is not guaranteed to arrive in a pipeline in time order or at predictable intervals. In addition, there are no guarantees that data events will appear in the pipeline in the same order that they were generated.

**Trigger**

Triggers determine when to emit aggregated results as data arrives. For bounded data, results are emitted after all of the input has been processed. For unbounded data, results are emitted when the watermark passes the end of the window, indicating that the system believes all input data for that window has been processed. Apache Beam provides several predefined triggers and lets you combine them.