Pipeline I/O

ng: Dataflow SDK 1.x for Java is unsupported as of October 16, 2018. After August 12, 2020, Dataflow will no
sing Dataflow 1.x and below. See Migrating from Dataflow SDK 1.x for Java
aflow/docs/guides/migrate-java-1-to-2) for migration guidance.

ocumentation on this page applies only to the Dataflow SDK 1.x for Java.

ataflow SDK 2.x for Java and the Dataflow SDK for Python are based on Apache Beam. See the documentatio
aflow/model/programming-model-beam) for those SDKs.

When you create a pipeline, you'll often need to read data from some external source, such as a file in Google Cloud Storage (/storage/docs) or a BigQuery (/bigquery/docs) table. Likewise, you'll want your pipeline to output its result data to a similar external data sink, such as output files in Cloud Storage or BigQuery. The Dataflow SDKs provide transforms that can read data from an external source or write data to an external sink.

The Dataflow SDKs provide `Read` and `Write` transforms for a number of common data storage types. In addition, the `Read` and `Write` APIs are extensible; if you want your pipeline to read from or write to a data storage format that isn't supported by the built-in transforms, you can build extensions to provide your own `Read` and `Write` operations.

Extensibility is currently not supported in Python, but will be in the future.

Reading Input Data

`Read` transforms read data from an external source and return a `PCollection` representation of the data for use by your pipeline. You can use a `Read` transform at any point while constructing your pipeline to create a new `PCollection`, though it will be most common at the start of your program.

Java

Note: Since **`Read`** transforms do not have input **`PCollection`**s, they are applied to the **`Pipeline`** directly. As usual, the call to **`apply`** returns a **`PCollection`** of the appropriate type, whose elements represent the data. See Constructing Your Pipeline (/dataflow/pipelines/constructing-your-pipeline) for more information.

Reading From Multiple Locations

Many `Read` transforms, such as Text (#TextIO), support reading from multiple input files matching a glob operator you provide. Consider the following use of the `Read` transform, which uses a glob operator (*) to read all matching input files in the given location in Google Cloud Storage:

Java

```
p.apply(TextIO.Read.named("ReadFromText")
                   .from("gs://my_bucket/path/to/input-*.csv");
```

The above `Read` will read all files at the given location in Cloud Storage with the prefix "input-" and the suffix ".csv".

To read data from disparate sources into a single `PCollection`, read each one independently and then use the Flatten (/dataflow/model/multiple-pcollections#flatten) transform to create a single `PCollection`.

Writing Output Data

`Write` transforms write the data in a `PCollection` to an external data source. You'll most often use `Write` transforms at the end of your program to output your pipeline's final results. However, you can use `Write` to output a `PCollection`'s data at any point in your pipeline.

To use a `Write` transform, you call the `apply` method on the `PCollection` that you want to write, and pass the appropriate `Write` transform as an argument.

Java

When you **`apply`** a **`Write`** transform to a **`PCollection`**, the return value is an object of type **`PDone`**. The **`PDone`** object is a trivial result object and can be safely ignored.

Writing To Multiple Output Files

For file-based input and output data, such as Text (#TextIO), **`Write`** transforms write to multiple output files **by default.** The Cloud Dataflow service always produces sharded output files automatically. When you pass an output file name to a **`Write`** transform, the file name is used as the **prefix** for all output files that the **`Write`** transform produces.

You can append a suffix to each output file by specifying a suffix to your **`Write`** transform.

Consider the following use of the **`Write`** transform, which writes multiple output files to a location in Cloud Storage. Each file has the prefix "numbers", a numeric tag, and the suffix ".csv".

Java

```
records.apply(TextIO.Write.named("WriteToText")
                          .to("gs://my_bucket/path/to/numbers")
                          .withSuffix(".csv"));
```

The above **`Write`** will write multiple output files to the given location in Cloud Storage with the prefix "numbers" and the suffix ".csv".

I/O APIs Included in the Dataflow SDKs

Some Source and Sink APIs are included in the Dataflow SDKs.

Java

The Dataflow SDK for Java provides **`Read`** and **`Write`** transforms for a number of common data formats (/dataflow/java-sdk/JavaDoc) including:

- Avro (/dataflow/model/avro-io) files

- BigQuery (/dataflow/model/bigquery-io) tables
- Bigtable (/dataflow/model/bigtable-io)
- Datastore (/dataflow/model/datastore-io)
- Pub/Sub (/dataflow/model/pubsub-io)
- Text (/dataflow/model/text-io) files

Additional I/O APIs

In addition to the I/O APIs, the Dataflow SDKs provides an extensible API that you can use to create your own custom data sources and sinks.

Java

You can create your own custom input sources and output sinks (/dataflow/model/custom-io-java) using Dataflow's Source and Sink APIs.

Except as otherwise noted, the content of this page is licensed under the Creative Commons Attribution 4.0 License (https://creativecommons.org/licenses/by/4.0/), and code samples are licensed under the Apache 2.0 License (https://www.apache.org/licenses/LICENSE-2.0). For details, see the Google Developers Site Policies (https://developers.google.com/site-policies). Java is a registered trademark of Oracle and/or its affiliates.

Last updated 2020-06-22 UTC.