Constructing Your Pipeline

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/pipelines/creating-a-pipeline-beam) for those SDKs.

Your Dataflow program expresses a data processing pipeline, from start to finish. This section explains the mechanics of using the classes in the Dataflow SDKs to build a pipeline. To construct a pipeline using the classes in the Dataflow SDKs, your program will need to perform the following general steps:

- Create a `Pipeline` object.
- Use a **Read** or **Create** transform to create one or more `PCollection`s for your pipeline data.
- Apply **transforms** to each `PCollection`. Transforms can change, filter, group, analyze, or otherwise process the elements in a `PCollection`. Each transform creates a new output `PCollection`, to which you can apply additional transforms until processing is complete.
- **Write** or otherwise output the final, transformed `PCollection`s.
- **Run** the pipeline.

he Simple Example Pipeline (/dataflow/model/pipelines#Simple-Example-Pipeline) below for a complet
ple that demonstrates each general step.

Creating Your Pipeline Object

A Dataflow program often starts by creating a `Pipeline` object.

In the Dataflow SDKs, each pipeline is represented by an explicit object of type `Pipeline`. Each `Pipeline` object is an independent entity that encapsulates both the data the pipeline operates over and the transforms that get applied to that data.

Java

To create a pipeline, declare a `Pipeline` object, and pass it some configuration options (/dataflow/pipelines/specifying-exec-params#Options). You pass the configuration options by creating an object of type `PipelineOptions`, which you can build by using the static method `PipelineOptionsFactory.create()`.

```
// Start by defining the options for the pipeline.
PipelineOptions options = PipelineOptionsFactory.create();

// Then create the pipeline.
Pipeline p = Pipeline.create(options);
```

Configuring Pipeline Options

Use the pipeline options to configure different aspects of your pipeline. These can include:

- Where your pipeline runs
- Where your pipeline job stages files
- Which Cloud Platform project your pipeline is associated with
- How many Compute Engine instances your pipeline uses as workers

The pipeline options properties include information about your Cloud Platform project required by the Cloud Dataflow service, such as your project ID and Cloud Storage staging locations. The pipeline options also let you control how many workers the Dataflow service should assign to your pipeline job, and where to direct your pipeline job's status messages.

A key property in the pipeline options that determines where your pipeline executes (either on the Cloud Dataflow Service, or locally) is the pipeline runner. The pipeline runner property also specifies whether your pipeline's execution is to be asynchronous or blocking.

Java

While you can set the properties of the `PipelineOptions` object directly within your pipeline program using setter methods (`PipelineOptions.set[OptionName]`), a best practice is to pass in the values using command line options. The Dataflow SDK for Java provides a `PipelineOptionsFactory` class that parses and validates command line options passed in to your pipeline. By using command line options to determine the `PipelineRunner` and other fields in `PipelineOptions` at runtime, you can use the same code to construct and run your pipeline both locally and in the cloud.

See Specifying Execution Parameters (/dataflow/pipelines/specifying-exec-params) for more information about how to set pipeline options programatically for either cloud or local mode execution. The WordCount example pipeline (/dataflow/examples/wordcount-example) also demonstrates how to set pipeline options at runtime by using command-line options.

Reading Data Into Your Pipeline

To create your pipeline's initial `PCollection`, you apply a root transform to your pipeline object. A root transform creates a `PCollection` from either an external data source or some local data you specify.

Java

There are two kinds of root transforms in the Dataflow Java SDK: `Read` and `Create`. `Read` transforms read data from an external source, such as BigQuery or a text file in Google Cloud Storage. `Create` transforms create a `PCollection` from an in-memory `java.util.Collection`.

The following example code shows how to `apply` a `TextIO.Read` root transform to read data from a text file in Google Cloud Storage. The transform is applied to a `Pipeline` object `p`, and returns a pipeline data set in the form of a `PCollection<String>`:

```
PCollection<String> lines = p.apply(
  TextIO.Read.named("ReadMyFile").from("gs://some/inputData.txt"));
```

Applying Transforms to Process Pipeline Data

To use transforms in your pipeline, you **apply** them to the `PCollection` that you want to transform.

Java

To apply a transform, you call the `apply` method on each `PCollection` that you want to process, passing the desired transform object as an argument.

The Dataflow SDKs contain a number of different transforms that you can apply to your pipeline's `PCollection`s. These include general-purpose core transforms, such as ParDo (/dataflow/model/par-do) or Combine (/dataflow/model/combine). There are also pre-written composite transforms (/dataflow/model/library-transforms) included in the SDK, which combine one or more of the core transforms in a useful processing pattern, such as counting or combining elements in a collection. You can also define your own more complex composite transforms (/dataflow/model/composite-transforms) to fit your pipeline's exact use case.

Java

In the Dataflow Java SDK, each transform is a subclass of the base class `PTransform`. When you call `apply` on a `PCollection`, you pass the `PTransform` you want to use as an argument.

The following code shows how to `apply` a transform to a `PCollection` of strings. The transform is a user-defined custom transform that reverses the contents of each string and outputs a new `PCollection` containing the reversed strings.

The input is a `PCollection<String>` called `words`; the code passes an instance of a `PTransform` object called `ReverseWords` to `apply`, and saves the return value as the `PCollection<String>` called `reversedWords`.

```
PCollection<String> words = ...;

PCollection<String> reversedWords = words.apply(new ReverseWords());
```

Writing or Outputting Your Final Pipeline Data

Once your pipeline has applied all of its transforms, you'll usually need to output the results. To output your pipeline's final `PCollection`s, you apply a `Write` transform to that `PCollection`. `Write` transforms can output the elements of a `PCollection` to an external data sink, such as a file in Google Cloud Storage or a BigQuery table. You can use `Write` to output a `PCollection` at any time in your pipeline, although you'll typically write out data at the end of your pipeline.

Java

The following example code shows how to `apply` a `TextIO.Write` transform to write a `PCollection` of `String` to a text file in Google Cloud Storage:

```
PCollection<String> filteredWords = ...;
filteredWords.apply(TextIO.Write.named("WriteMyFile").to("gs://some/outputData.txt")
```

Running Your Pipeline

Once you have constructed your pipeline, you use the `run` method to execute the pipeline. Pipelines are executed asynchronously: the program you create sends a specification for your pipeline to a **pipeline runner**, which then constructs and runs the actual series of pipeline operations. You can specify where your pipeline runs: either locally for testing and debugging purposes, or on the Cloud Dataflow managed service (/dataflow/service/dataflow-service-desc). See Specifying Execution Parameters (/dataflow/pipelines/specifying-exec-params) for more information on pipeline runners, configuring pipeline options, and local vs. cloud execution.

In the Dataflow SDKs, you specify a `PipelineRunner` in your pipeline options when you create your `Pipeline` object. When you've finished constructing your pipeline, you invoke `run` on your pipeline object as follows:

Java

```
p.run();
```