Avro 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.

The built-in `Read` and `Write` transforms for Avro files are included in `AvroIO` (/dataflow/java-sdk/JavaDoc/com/google/cloud/dataflow/sdk/io/AvroIO). You can use `AvroIO` to read/write both local files (meaning files on the system where your Java program runs) and remote files in Google Cloud Storage (/storage).

Java

Note: If you want your pipeline to read or write local files, you'll need to use the `DirectPipelineRunner` to run your pipeline locally (/dataflow/pipelines/specifying-exec-params#LocalExecution). This is because the Google Compute Engine instances that the Dataflow service (/dataflow/service/dataflow-service-desc) uses to run your pipeline won't be able to access files on your local machine for reading and writing.

Specifying an Avro Schema

To use `AvroIO`, you'll need to specify an Avro schema that describes the records to read or write. Avro relies on schemas to describe how data is serialized. See the Avro documentation (http://avro.apache.org/docs/current) to learn how Avro schemas work.

You can read specific kinds of Avro records by providing an Avro-generated class type, or you can read `GenericRecords` by providing an `org.apache.avro.Schema` object. Usually, you'll read

the `Schema` object from a schema file (`.avsc`). You can also specify a `Schema` in JSON-encoded string form.

To provide a schema, you use the `.withSchema` method with the `AvroIO` transform. You must call `.withSchema` any time you use `AvroIO.Read` or `AvroIO.Write`.

Reading with AvroIO

The `AvroIO.Read` transform reads records from one or more Avro files and creates a `PCollection` in which each element represents a record. `AvroIO.Read` can produce a `PCollection` of automatically-generated Avro class objects or of `GenericRecord` objects. The kind of `PCollection` produced depends on the schema type that you choose.

Using an automatically-generated Avro class will result in a `PCollection` whose elements are objects of that Avro class type, as shown:

Java

```
PipelineOptions options = PipelineOptionsFactory.create();
Pipeline p = Pipeline.create(options);

PCollection<AvroAutoGenClass> records = p.apply(
  AvroIO.Read.named("ReadFromAvro")
        .from("gs://some/inputData.avro")
        .withSchema(AvroAutoGenClass.class));
```

To read your Avro file(s) into a `PCollection<GenericRecord>`, you can pass an `org.apache.avro.Schema` object or a schema written as a JSON-encoded string. The following code sample obtains a `org.apache.avro.Schema` object by parsing an `.avsc` file, then uses the resulting `Schema` to read sharded input Avro files from Google Cloud Storage:

Java

```
PipelineOptions options = PipelineOptionsFactory.create();
Pipeline p = Pipeline.create(options);

Schema schema = new Schema.Parser().parse(new File("schema.avsc"));
```

```
PCollection<GenericRecord> records =
  p.apply(AvroIO.Read.named("ReadFromAvro")
                     .from("gs://my_bucket/path/records-*.avro")
                     .withSchema(schema));
```

As with other file-based Dataflow sources, the `AvroIO.Read` transform can read multiple input files. See Reading Input Data (/dataflow/model/pipeline-io#using-reads) for more information on how to handle multiple files when reading from file-based sources.

Writing with AvroIO

The `AvroIO.Write` transform writes a `PCollection` of Avro records to one or more Avro files. To use `AvroIO.Write`, you'll need to represent your final output data as a `PCollection` of either automatically-generated Avro class objects or a `PCollection` of `GenericRecords`. You can use a ParDo (/dataflow/model/par-do) to transform your data appropriately.

To write specific records, use an automatically-generated Avro class as the Avro schema:

Java

```
PCollection<AvroAutoGenClass> filteredRecords = ...;
filteredRecords.apply(AvroIO.Write.named("WriteToAvro")
                                  .to("gs://some/outputData.avro")
                                  .withSchema(AvroAutoGenClass.class)
                                  .withSuffix(".avro"));
```

To write `GenericRecord` objects, you can pass an `org.apache.avro.Schema` (often by parsing an `.avsc` file) or a schema written as a JSON-encoded string. The following code sample parses an `.avsc` file to obtain a `Schema` object, and uses it to write sharded output Avro files to Google Cloud Storage:

Java

```
Schema schema = new Schema.Parser().parse(new File("schema.avsc"));
```

```
PCollection<GenericRecord> records = ...;
records.apply(AvroIO.Write.named("WriteToAvro")
                          .to("gs://my_bucket/path/numbers")
                          .withSchema(schema)
                          .withSuffix(".avro"));
```

Note that `AvroIO.Write` writes to multiple output files by default. See Writing Output Data (/dataflow/model/pipeline-io#using-writes) for additional information.

Last updated 2020-06-22 UTC.