**What is it?**

* Serverless (apache spark requires more)
* Portable: Processing code is separate from execution environment
* Unified: Batch and streaming modes use the same programming model

**How does it work?**

* Use Beam’s SDK to define your pipeline, then use one a supporte distributed processing execution backends (such as Dataflow) to run the program.

[Doc: Constructing your Pipeline](https://cloud.google.com/dataflow/pipelines/constructing-your-pipeline)

* Pipeline’s can have multiple inputs
* Root transform = creating a Pcollection from a datasource.
  + Two java root transforms
    - Read = external data source (BigQuery or Cloud Storage)
    - Create = in-memory (java.util.Collection)
* Output = apply “Write” transform
* Run method for executing pipeline

[Doc: Pipeline I/O](https://cloud.google.com/dataflow/model/pipeline-io)

* I/O APIs included in Dataflow SDK
  + [Avro](https://cloud.google.com/dataflow/model/avro-io) files
  + [BigQuery](https://cloud.google.com/dataflow/model/bigquery-io) tables
  + [Bigtable](https://cloud.google.com/dataflow/model/bigtable-io)
  + [Datastore](https://cloud.google.com/dataflow/model/datastore-io)
  + [Pub/Sub](https://cloud.google.com/dataflow/model/pubsub-io)
  + [Text](https://cloud.google.com/dataflow/model/text-io) files

**Doc: transformations**

* Examples of transformations
  + Move data between data sources or storage units
  + Transforming data into a more desirable format
  + Loading data into a new system
* GroupbyKey -- use to group fitness data by member-ID
  + Transforms from a multi-map (multiple keys to individual values) to unimap (unique keys to collection of values)
* Combine PerKey -- use to create averages (averages of individuals, averages of group)
* DoFn is a Beam SDK class that defines a distributed processing function.

**Doc: Testing**

* Recommended: test locally
* SDK tests
  + Individual functions
  + Entire composite transform
  + Full pipeline

**Doc: Specifying execution parameters**

Asynchronous execution - pipeline executes asynchronously

Blocking execution - pipeline executes asynchronously but the second job doesn’t start until first job completes.

Streaming execution - if your datasource uses an unbounded source or sink, you must set the streaming option to “true” in PipelineOptions

CTL + C does not quit a job. to cancel the job, you'll need to use the [Dataflow Monitoring Interface](https://cloud.google.com/dataflow/pipelines/dataflow-monitoring-intf) or the [Dataflow Command-line Interface](https://cloud.google.com/dataflow/pipelines/dataflow-command-line-intf).

Doc: Deploying your pipeline

DMI & DCI are the two ways to monitor your job.

DMI = Dataflow Monitoring Interface

DCI = Dataflow Commandline Interface

Automatically manages aspects of distributed parallelized processing: such as autoscaling and dynamic work rebalancing.

Able to run up to 25 concurrent jobs per project

Jobs that are 10MB in size or smaller (size of job request is tied to size of JSON representation of pipeline). To estimate the size of your job run this command:

--dataflowJobFile=< path to output file >

Allows max 1000 compute engine instances per job // 4000 cores per job

15 persistent disks per worker instance

Must be 1:1 persistent disk to worker instance

Default size is **250 GB in batch mode**

**What software and packages do I need?**

* Python (or Java)
* Google Cloud SDK
* [Google Dataflow SDK 2.x](https://cloud.google.com/dataflow/docs/installing-dataflow-sdk)
  + [Documentation](https://beam.apache.org/get-started/quickstart-py/)
* Object/block storage (Cloud Storage)
* XML document in storage

**What information do I need to provide Dataflow? How do I pass it that info?**

* ProjectID = gcp-project-final
* Bucket name = df-codelabs-tutorial

**Where do all the pieces go?**

XML document → Cloud Storage

Dataflow Python app → Cloud Shell

**How do I run?**

[Command line example:](https://github.com/GoogleCloudPlatform/DataflowSDK-examples/blob/1e098181a3b99fe8581ee924056d4de2062113b6/python/dataflow_examples/complete/juliaset/juliaset_main.py)

python juliaset\_main.py \

--job\_name juliaset-$USER \

--project YOUR-PROJECT \

--runner DataflowRunner \

--setup\_file ./setup.py \

--staging\_location gs://YOUR-BUCKET/juliaset/staging \

--temp\_location gs://YOUR-BUCKET/juliaset/temp \

--coordinate\_output gs://YOUR-BUCKET/juliaset/out \

--grid\_size 20 \

|  |  |
| --- | --- |
| import logging |  |
|  |  |
|  |  |
|  | from apache\_beam.examples.complete.juliaset.juliaset import juliaset |
|  |  |
|  |  |
|  | if \_\_name\_\_ == '\_\_main\_\_': |
|  | logging.getLogger().setLevel(logging.INFO) |
|  | juliaset.run() |