Apache Beam -KirkYagami🧑‍💻🕵️

https://kamilwu.medium.com/apache-beam-tutorial-and-beginners-guide-8f19623be923🔗

What is Apache Beam?

Apache Beam is a unified programming model that defines both batch and streaming data processing pipelines. It abstracts the underlying execution engines, meaning that you can write a data processing pipeline once and run it on different engines like Google Cloud Dataflow, Apache Spark, Apache Flink, and others.

Apache Beam can be expressed as a programming model for distributed data processing

> It separates the pipeline definition from its execution.

Advantages of Apache Beam:

- **Unified Model**: Can handle both streaming and batch data processing with the same code.
- **Portability**: Apache Beam pipelines can run on multiple backends.
- Windowing and Watermarks First-class support for event-time processing, handling late data and out-of-order events.

Concepts

`Pipeline`
A `Pipeline` encapsulates your entire data processing task, from start to finish. This includes reading input data, transforming that data, and writing output data. All Beam driver programs must create a `Pipeline`. When you create the `Pipeline`, you must also specify the execution options that tell the `Pipeline` where and how to run.

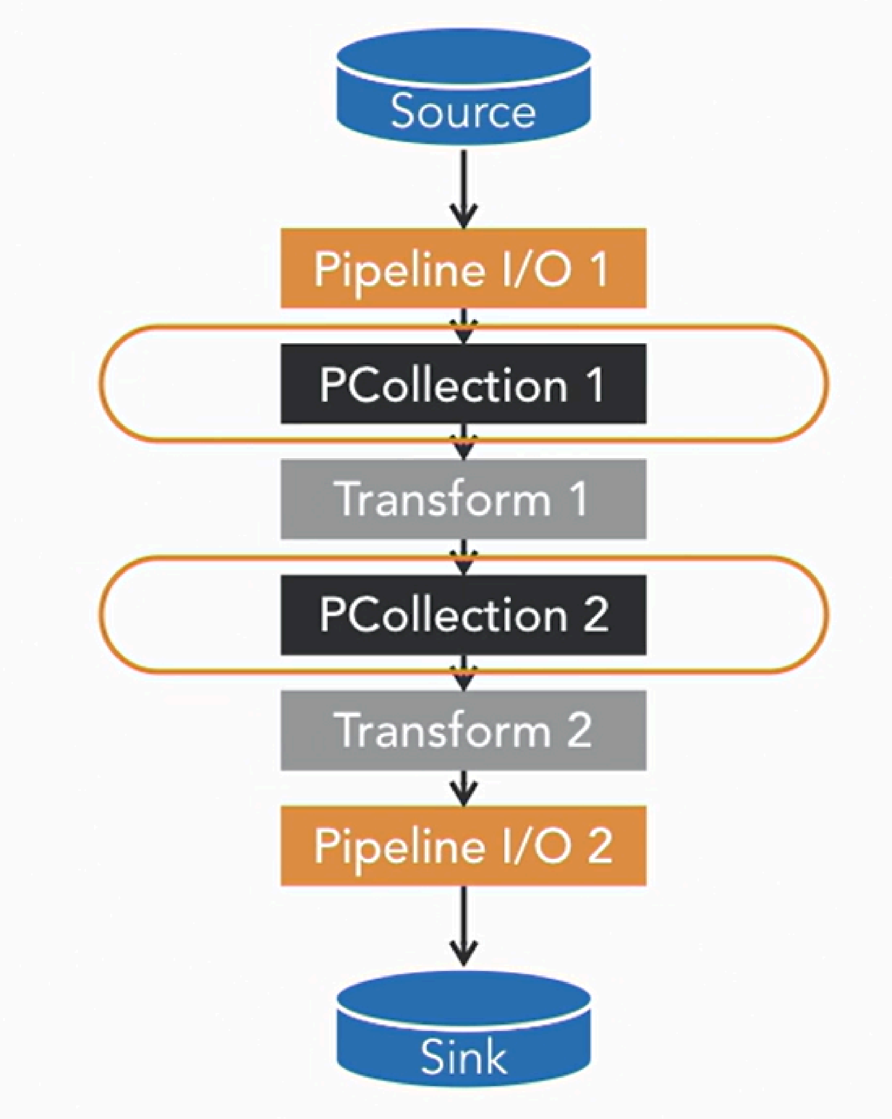
`Runner`
It determines where this pipeline will operate

`PCollection`

> It is equivalent to RDD or DataFrames in Spark. The pipeline creates a `PCollection` by reading data from a data source, and after that, more PCollections keep on developing as PTransforms are applied to it

A `PCollection` represents a distributed data set that your Beam pipeline operates on. The data set can be *bounded*, meaning it comes from a fixed source like a file, or *unbounded*, meaning it comes from a continuously updating source via a subscription or other mechanism.

Your pipeline typically creates an initial `PCollection` by reading data from an external data source, but you can also create a `PCollection` from in-memory data within your driver program. From there, `PCollections` are the inputs and outputs for each step in your pipeline.



- Data in the pipeline
- A distributed, multi-element dataset
- Data moving in the pipeline
- Serve as inputs and/or outputs of processing
- Intermediate PCollections (data holders) between processing steps

`PTransform`

A `PTransform` represents a data processing operation, or a step, in your pipeline. Every `PTransform` takes one or more `PCollection` objects as input, performs a processing function that you provide on the elements of that `PCollection`, and produces zero or more output `PCollection` objects.

`I/O transforms`

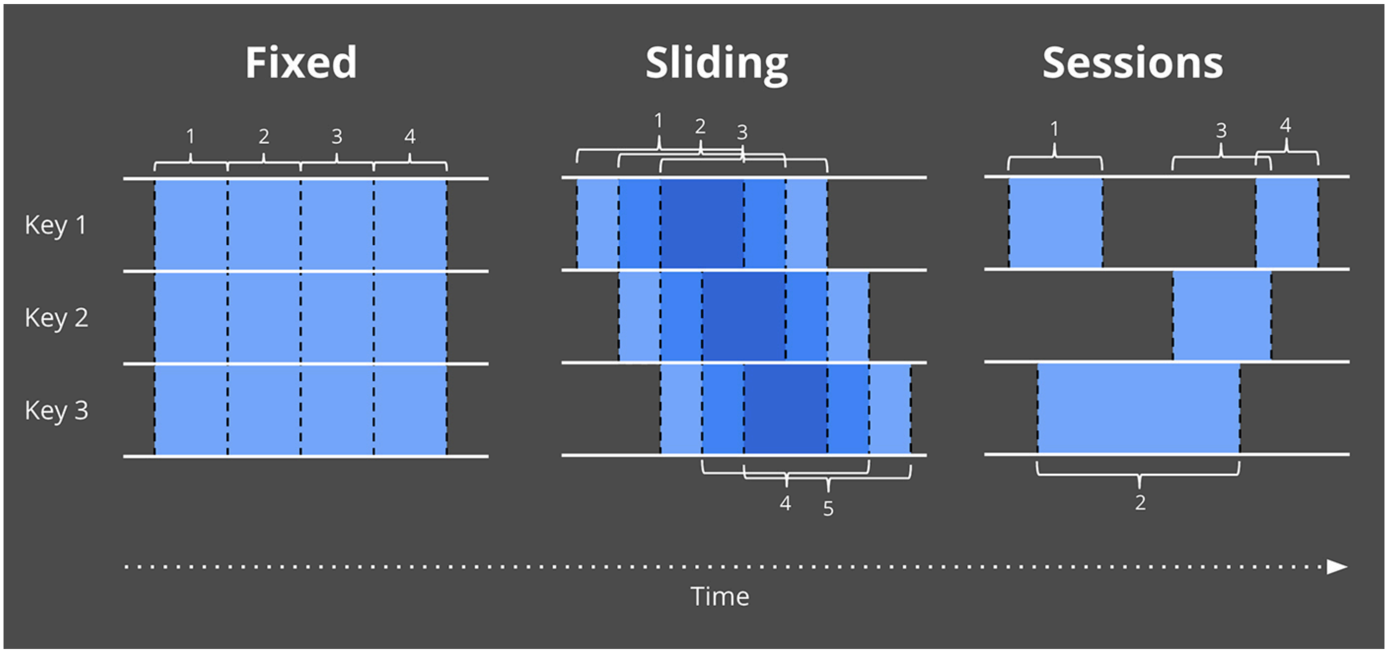
Beam comes with a number of "IOs" - library `PTransforms` that read or write data to various external storage systems.

`Windowing`
Windowing a PCollection divides the elements into windows based on the associated event time for each element.

`Windowing`

- Windowing is the process of taking a small subset of a larger dataset, for processing and analysis.
- A naïve approach, the rectangular window, involves simply truncating the dataset before and after the window, while not modifying the contents of the window at all.
- Types:
  1. **Fixed Window**:
     - Windows of fixed interval duration, uniform across all the keys, no overlaps between two consecutive widows.
     - Also Known as Tumbling Window
     - Use cases — any aggregation use cases, any batch analysis of data, relatively simple use cases.
  2. **Sliding Window**:
     - Windows of fixed interval duration, uniform across all the keys, overlap between two windows (same element can be present in multiple windows)
     - Also Known as Hopping Window
     - Use cases — Moving averages of data
  3. **Session Window**:
     - Windows of dynamically set intervals, non-uniform across keys (different windows for different keys, different window sizes for each key), no overlap between two windows
     - Use cases — user session data, click data, real time gaming data analysis

!



`Triggers`
Allows specifying a trigger to control when (in processing time) results for the given window can be produced. Triggers determines when a Window's contents should be output based on certain criteria being met. Types of triggers are :
- Time based triggers
- Data Driven triggers
- Composite triggers

`Watermark`
It is a threshold that indicates when Dataflow expects all of the data in a window to have arrived. If new data arrives with a timestamp that's in the window but older than the watermark, the data is considered **late data**.

> Watermark is a heuristic that tracks how far behind the system is in processing data from the event time. Where in event time does processing occur?

$$Max(EventTime) - Watermark = WatermarkBoundary$$

`ParDo`
It is a parallel processing function which can transform elements of an input PCollection to an output PCollection.
- ParDo is a common intermediate step in a pipeline
- You might use it to extract certain fields from a set of raw input records or convert raw input into a different format.
- You might also use ParDo to convert process data into an output format, like table rows for

BigQuery or strings for printing.
- You can use ParDo to consider each element in a PCollection and either output that element to a new collection or discard it.
- If your input PCollection contains elements that are of a different type or format than you want, you can use ParDo to perform a conversion on each element and output the result to a new PCollection.

`DoFn`
It is a template which is used to create user defined functions that are referenced by ParDo. A DoFn is a Beam SDK class that defines a distributed processing function.

The key difference between `ParDo` and `DoFn` in Apache Beam lies in their roles within a pipeline:

1. `DoFn` **(Distributed Function)**:
   - **What it is**: `DoFn` is a user-defined function that specifies the processing logic for each element in a PCollection. It's a low-level API where you define what happens to individual elements.
   - **Role**: It represents the processing logic (the "how" of the transformation) applied to the elements in the pipeline.
   - **Example**: A function that takes a string and splits it into words.

   ```
   class WordExtractingDoFn(beam.DoFn):
       def process(self, element):
           # Split the line into words and return them
           return element.split()
   ```

2. `ParDo` **(Parallel Do)**:
   - **What it is**: `ParDo` is the Beam transform that applies a `DoFn` (or a function) to each element in a PCollection. It's the high-level operation that runs the `DoFn` in parallel across the elements in a dataset.
   - **Role**: It is responsible for distributing the work of a `DoFn` across the elements of the input PCollection (the "what" of the transformation).
   - **Example**: Using the `WordExtractingDoFn` in a pipeline to process each line of text.

   ```
   lines | 'ExtractWords' >> beam.ParDo(WordExtractingDoFn())
   ```

How they work together:

- `DoFn` is the function that does the actual processing (e.g., splitting a line of text into words).

- `ParDo` is the operation that applies that function to all elements in parallel across a PCollection.

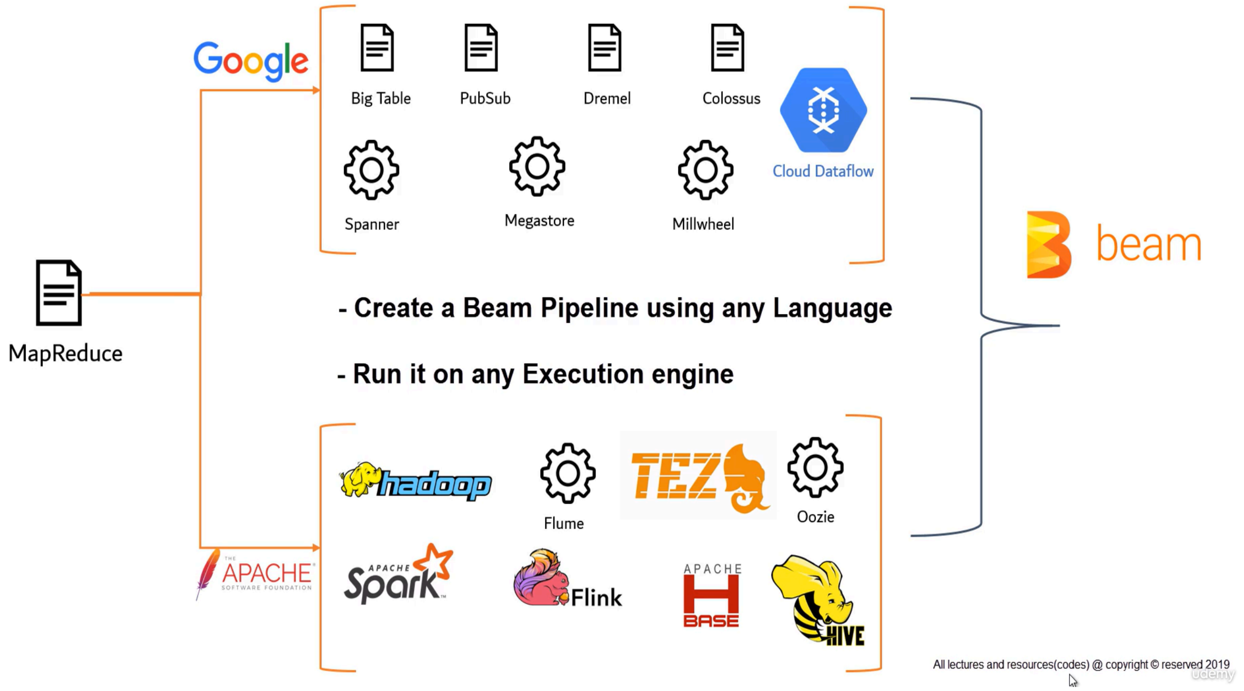
Example:

```
class SplitWords(beam.DoFn):
    def process(self, element):
        return element.split()

# Applying the DoFn using ParDo
lines = p | 'Read' >> beam.io.ReadFromText('sample.txt')
words = lines | 'SplitWords' >> beam.ParDo(SplitWords())
```

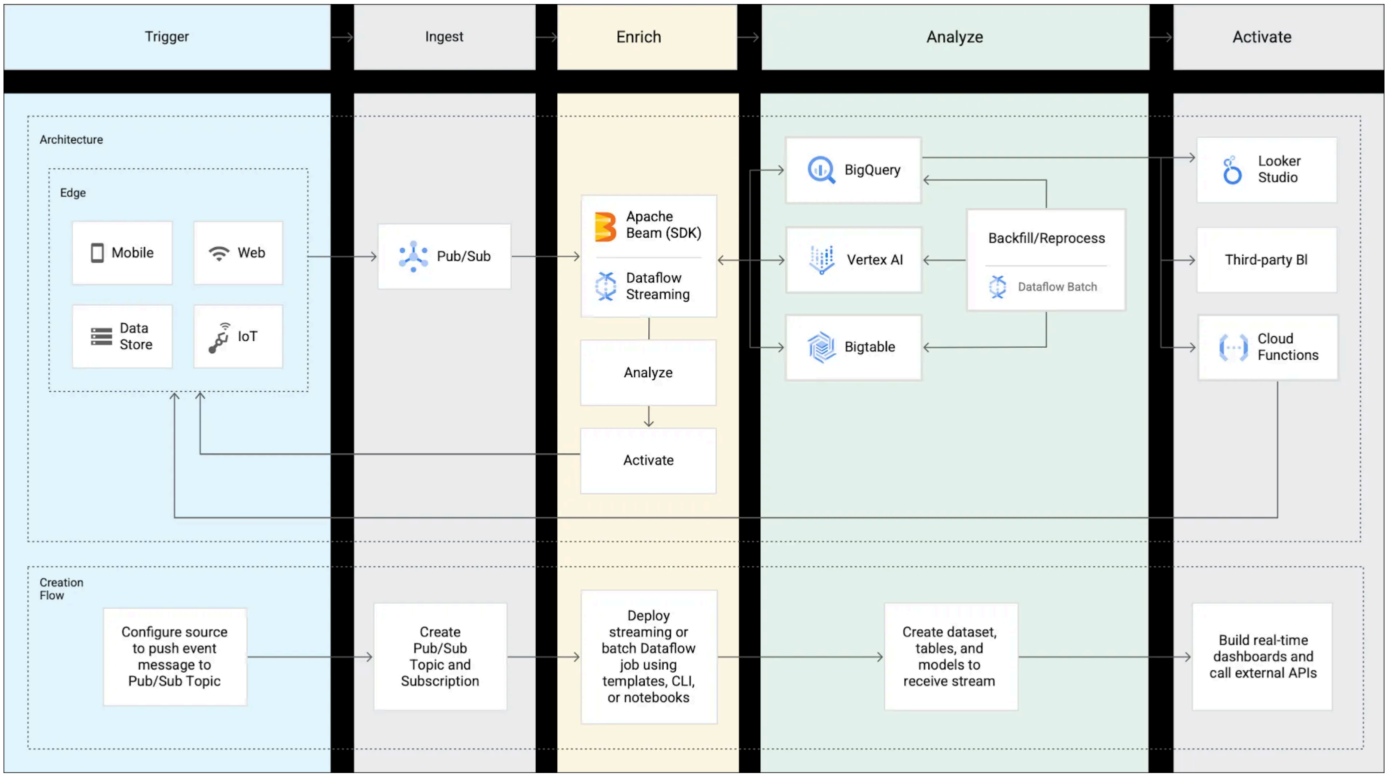
In this example:

- `SplitWords` is a `DoFn` that defines how to split each line into words.
- `ParDo(SplitWords())` applies the function to every element in the `lines` PCollection in parallel.



- Create a Beam Pipeline using any Language

- Run it on any Execution engine


Trigger
Ingest
Enrich
Analyze
Activate
Architecture
Edge
Mobile
Web
Data Store
IoT
Pub/Sub
Apache Beam (SDK)
Dataflow Streaming
Analyze
Activate
BigQuery
Vertex AI
Bigtable
Backfill/Reprocess
Dataflow Batch
Looker Studio
Third-party BI
Cloud Functions
Creation Flow
Configure source to push event message to Pub/Sub Topic
Create Pub/Sub Topic and Subscription
Deploy streaming or batch Dataflow job using templates, CLI, or notebooks
Create dataset, tables, and models to receive stream
Build real-time dashboards and call external APIs

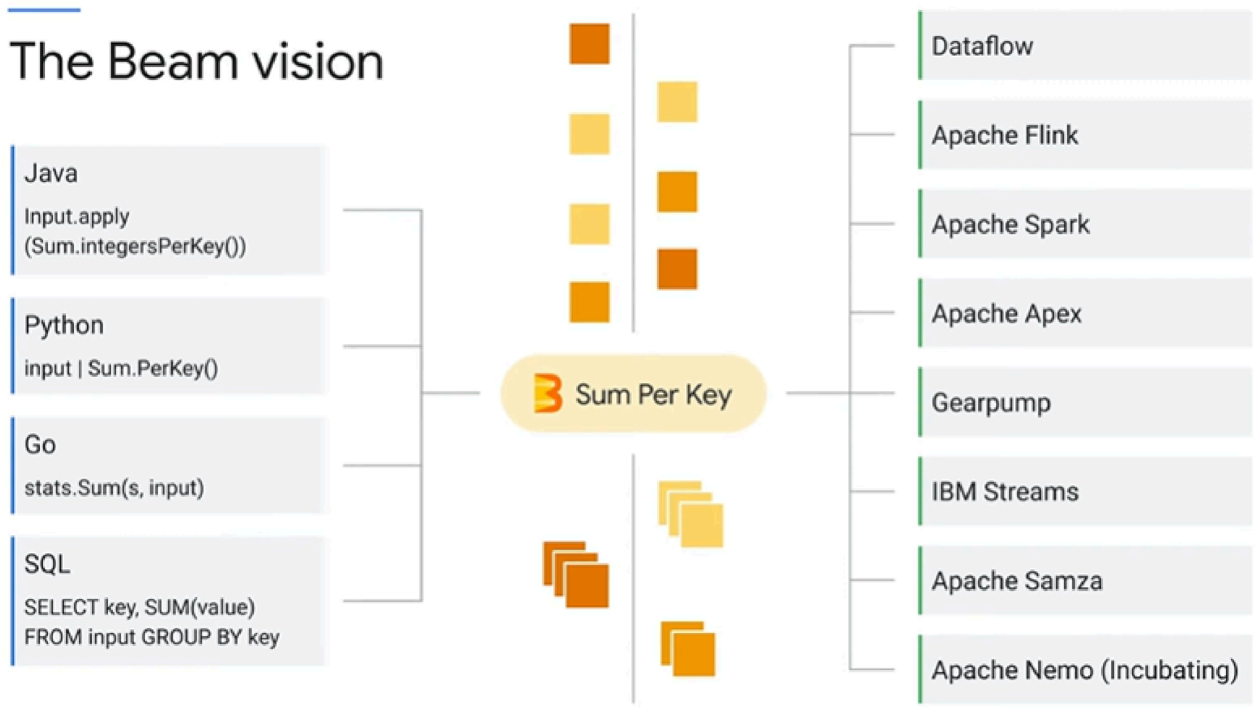
Which is faster - Apache Beam or Spark?

- Invalid Comparison

> Beam's performance depends upon the performance of Execution Engine

Is Apache Beam replacement for Spark, Flink etc.?

- No, it is just a programming model. Needs Spark, Flink etc. for execution



The Beam vision

Java

Input.apply
(Sum.integersPerKey())

Python

input | Sum.PerKey()

Go

stats.Sum(s, input)

SQL

SELECT key, SUM(value)
FROM input GROUP BY key

Sum Per Key

Dataflow

Apache Flink

Apache Spark

Apache Apex

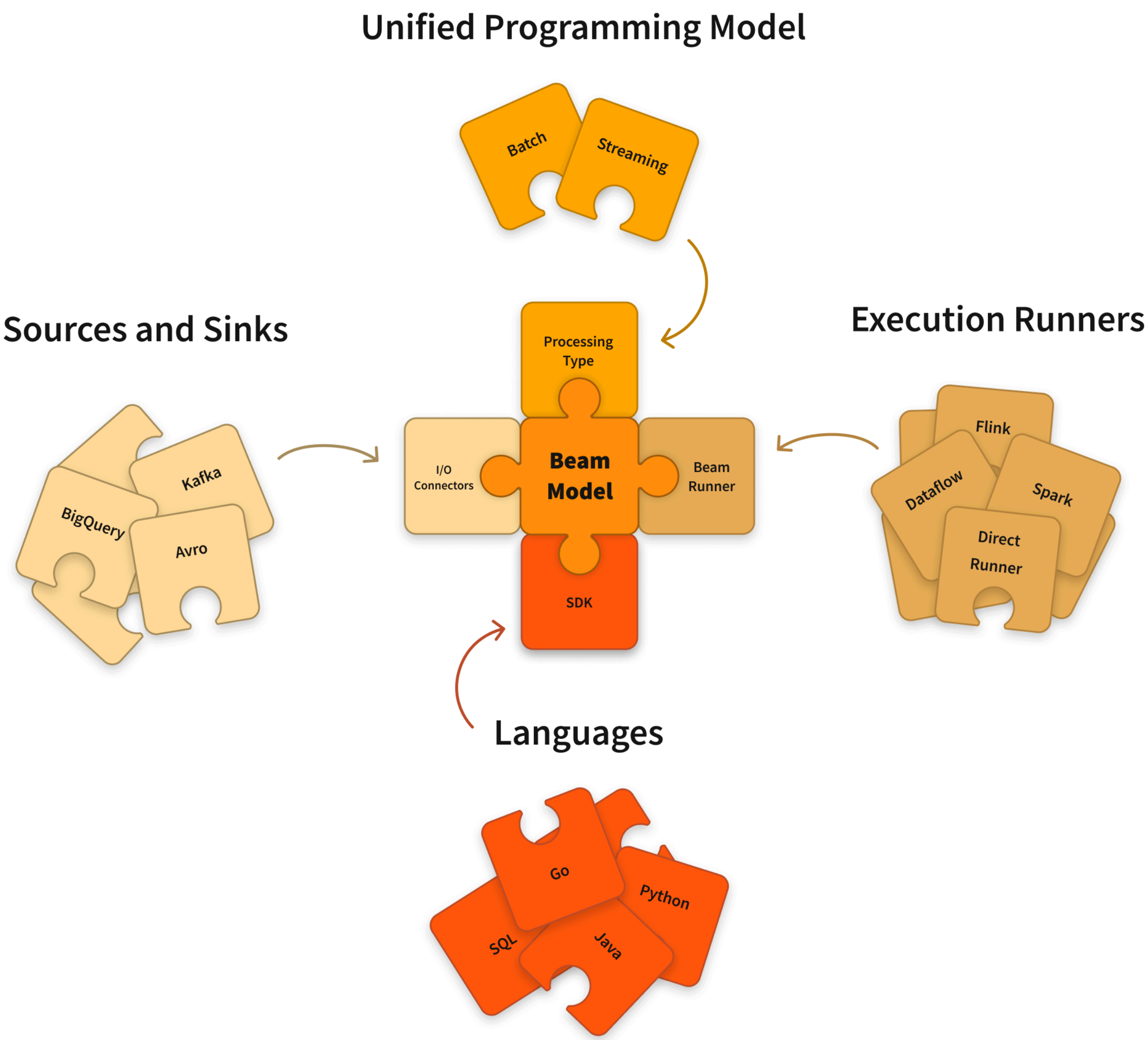
Gearpump

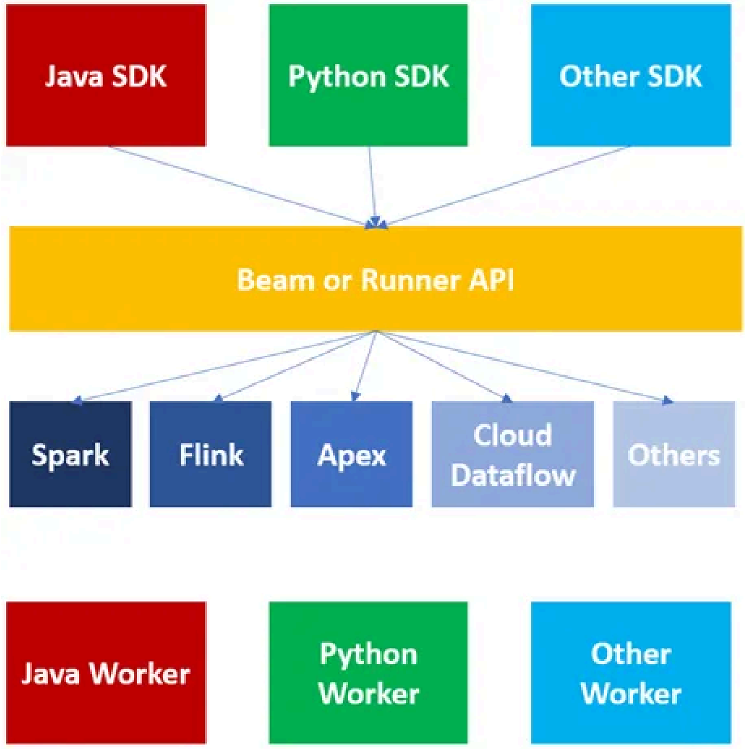
IBM Streams

Apache Samza

Apache Nemo (Incubating)

Beam Architecture


Unified Programming Model
Batch
Streaming
Sources and Sinks
Kafka
BigQuery
Avro
Execution Runners
Flink
Dataflow
Spark
Direct Runner
Processing Type
I/O Connectors
Beam Model
Beam Runner
SDK
Languages
Go
Python
SQL
Java



Apache Beam Pipeline Runners

The Beam Pipeline Runners translate the data processing pipeline you define with your Beam program into the API compatible with the distributed processing back-end of your choice. When you run your Beam program, you'll need to specify an appropriate runner for the back-end where you want to execute your pipeline.

Beam currently supports the following runners:

- Direct Runner
- Apache Flink Runner
- Apache Nemo Runner
- Apache Samza Runner
- Apache Spark Runner

- Google Cloud Dataflow Runner
- Hazelcast Jet Runner
- Twister2 Runner

```
gcloud auth login
gcloud services enable dataflow compute_component logging storage_component
storage_api bigquery pubsub datastore.googleapis.com
cloudresourcemanager.googleapis.com

# Create local authentication credentials for your user account:
# Instead of generating and managing service account keys manually, you can
use this method during development.
gcloud auth application-default login



# Add Dataflow Admin role
gcloud projects add-iam-policy-binding bigdata3844 \
  --member="serviceAccount:22777180107-compute@developer.gserviceaccount.com"
\
  --role="roles/dataflow.admin"


# Add Dataflow Worker role
gcloud projects add-iam-policy-binding bigdata3844 \
  --member="serviceAccount:22777180107-compute@developer.gserviceaccount.com"
\
  --role="roles/dataflow.worker"


# Add Storage Object Admin role
gcloud projects add-iam-policy-binding bigdata3844 \
  --member="serviceAccount:22777180107-compute@developer.gserviceaccount.com"
\
  --role="roles/storage.objectAdmin"



# storage bucket
gcloud storage buckets create gs://rev_dataflow --default-storage-class
```

```
STANDARD --location US

# Check version
# python --version # 3.10
# python -m pip --version


pip install 'apache-beam[gcp]'

# locally
python -m apache_beam.examples.wordcount \
  --output outputs

cat outputs*


# Use Dataflow
python -m apache_beam.examples.wordcount \
    --region DATAFLOW_REGION \
    --input gs://dataflow-samples/shakespeare/kinglear.txt \
    --output gs://BUCKET_NAME/results/outputs \
    --runner DataflowRunner \
    --project PROJECT_ID \
    --temp_location gs://BUCKET_NAME/tmp/


python -m apache_beam.examples.wordcount \
    --region us-central1 \
    --input gs://dataflow-samples/shakespeare/kinglear.txt \
    --output gs://rev_dataflow/results/outputs \
    --runner DataflowRunner \
    --project bigdata3844 \
    --temp_location gs://rev_dataflow/tmp/


gcloud projects add-iam-policy-binding bigdata3844 \
  --member="serviceAccount:sa-local2gcs@bigdata3844.iam.gserviceaccount.com"
\
  --role="roles/storage.objectAdmin"

gcloud projects add-iam-policy-binding bigdata3844 \
    --member="serviceAccount:sa-
local2gcs@bigdata3844.iam.gserviceaccount.com" \
    --role="roles/dataflow.admin"


# The above service account is a user variable.
```

https://github.com/apache/beam/blob/master/sdks/python/apache_beam/examples/wordcount.py

`wordcount.py`

```
"""A word-counting workflow."""
import argparse
import logging
import re

import apache_beam as beam
from apache_beam.io import ReadFromText
from apache_beam.io import WriteToText
from apache_beam.options.pipeline_options import PipelineOptions
from apache_beam.options.pipeline_options import SetupOptions



class WordExtractingDoFn(beam.DoFn):
  """Parse each line of input text into words."""
  def process(self, element):
    return re.findall(r'[\w\']+', element, re.UNICODE)

def run(argv=None, save_main_session=True):
  """Main entry point; defines and runs the wordcount pipeline."""
  parser = argparse.ArgumentParser()
  parser.add_argument(
      '--input',
      dest='input',
      default='gs://dataflow-samples/shakespeare/kinglear.txt',
      help='Input file to process.')
  parser.add_argument(
      '--output',
      dest='output',
      required=True,
      help='Output file to write results to.')
  known_args, pipeline_args = parser.parse_known_args(argv)

  # We use the save_main_session option because one or more DoFn's in this
  # workflow rely on global context (e.g., a module imported at module
level).

  pipeline_options = PipelineOptions(pipeline_args)
  pipeline_options.view_as(SetupOptions).save_main_session =
save_main_session

  # The pipeline will be run on exiting the with block.
  with beam.Pipeline(options=pipeline_options) as p:

    # Read the text file[pattern] into a PCollection.
    lines = p | 'Read' >> ReadFromText(known_args.input)

    counts = (
        lines
        | 'Split' >>
(beam.ParDo(WordExtractingDoFn()).with_output_types(str))
        | 'PairWithOne' >> beam.Map(lambda x: (x, 1))
        | 'GroupAndSum' >> beam.CombinePerKey(sum))

    # Format the counts into a PCollection of strings.
    def format_result(word, count):
      return '%s: %d' % (word, count)

    output = counts | 'Format' >> beam.MapTuple(format_result)

    # Write the output using a "Write" transform that has side effects.
    # pylint: disable=expression-not-assigned
    output | 'Write' >> WriteToText(known_args.output)

if __name__ == '__main__':
  logging.getLogger().setLevel(logging.INFO)
  run()
```

```
python wordcount.py --input path/to/the/file.txt  --output ./output.txt
```

```
cat output.txt-* > output.txt
```

Nikhil Sharma
Anil Kumar (Unverified)
Aravindan B (Unverified)
Babita Bulasara (Unverified)
c.sangeetha (U nverified)
Chandu (Unverified)
Daripalli Bhavana ... (Unverified)
Gabriel Kumar
Organizer
Ganesh (Unverified)
Hanish Kumar (Unverified)

Karthikeyi (Unverified)
krishna chaithanaya (Unverified)
Lepsita (Unverified)
Lokesh (Unverified)
Mamilla Bhavana (Unverified)
Manoj R (Unverified)
Manya Dharshini S (Unverified)
Mukesh (Unverified)
NAGENDRAN M (Unverified)
Nivis Miriam
Pasupuleti Bhargav (Unverified)
PATHIPATI CHIRR.. (Unverified)
Pavan Teja (Unverified)
Poornima S
Praveen kumar (Unverified)
Priya K
RAGHAVANANDU ... (Unverified)
Rahul B (Unverified)
Ram (Unverified)
Rennet T
SAI KIRAN (Unverified)
Shaik Akram Huss... (Unverified)
SHAIK JAFAR VAU (Unverified)
Srilekha Yeruva (Unverified)
Sudharshan.V (U nverified)
Supriya Pasupuleti (Unverified)
swapna (Unverified)
SYED UMMAR BA.. (Unverified)
Vivek (Unverified)