
BEAM
SUMMIT

Testing Data Pipelines

Bipin Upadhyaya
Strategic Cloud Engineer
Google Cloud PSO

BEAM SUMMIT
NYC 2023

Agenda

Data Pipelines

BEAM Pipelines Testing Overview

Unit Testing

Integration Testing

Batch Pipeline Testing (Walkthrough)

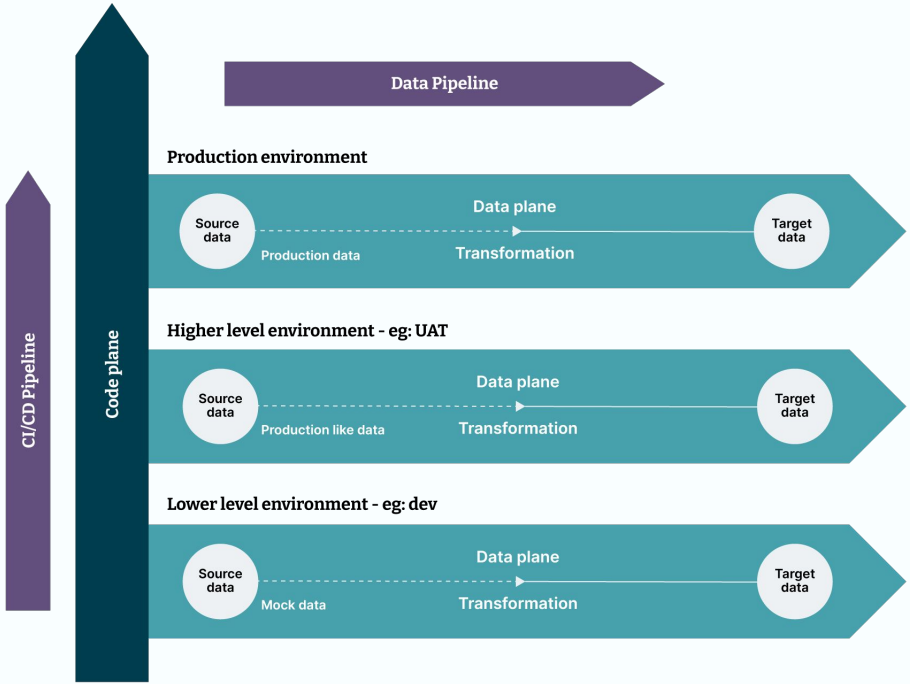
Streaming Pipeline Testing (Walkthrough)

Integration Testing - Deployment Options

Data pipelines

Systems that facilitate the flow and transformation of data from one stage to another

- In Code Plane code flows vertically up the Y-axis between environments.
- In Data Plane data flows horizontally along the X-axis in each of these environments where data is transformed from one form to another.

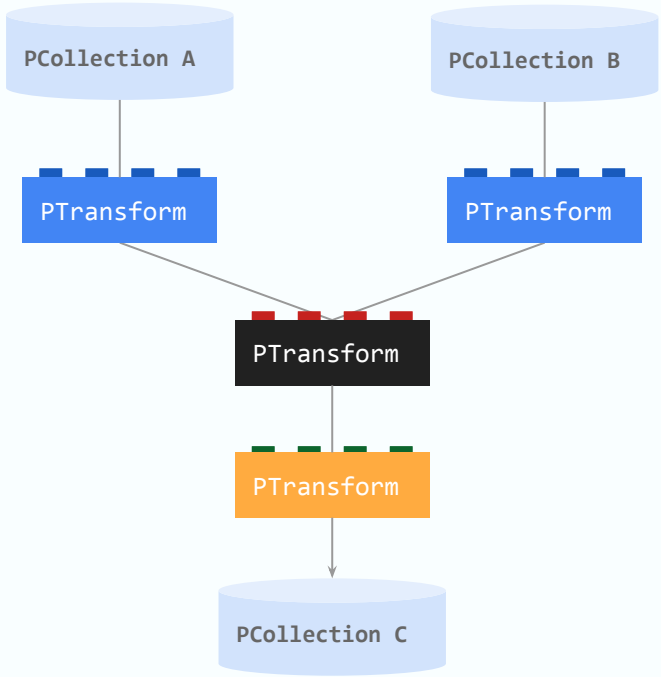


Ref: Get back to basics with testing data pipelines: two orthogonal planes

Pipeline in Apache Beam

A Pipeline is a Directed Acyclic Graph (DAG) of data transformations applied to one or more collections of data. It might include multiple input sources and output sinks and its operations (PTransforms) can both read and output PCollections.

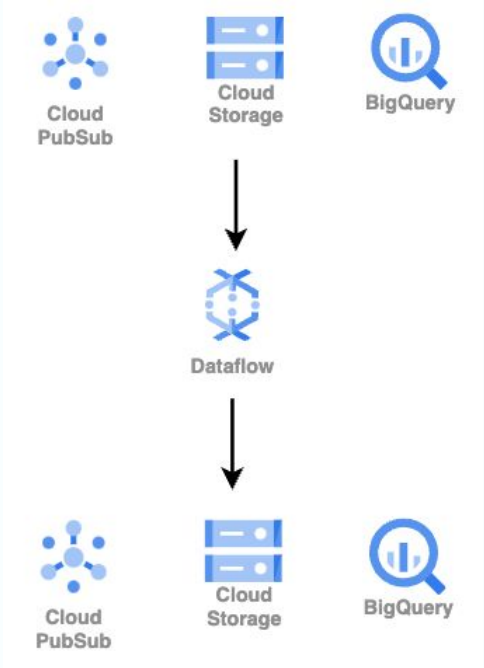
Apache beam support multiple SDKs. In this workshop we concentrate on Python. Example/Exercise and samples are all on python.



Batch Vs Streaming

- **Batch processing**
  - Fixed size, complete data
  - Bounded Data
    - E.g., bounded sources: Objects in bucket, Datastore, BigQuery
- **Stream processing**
  - Continuously new data
  - Unbounded data
    - E.g., unbounded sources: Pubsub subscription, Kafka topic

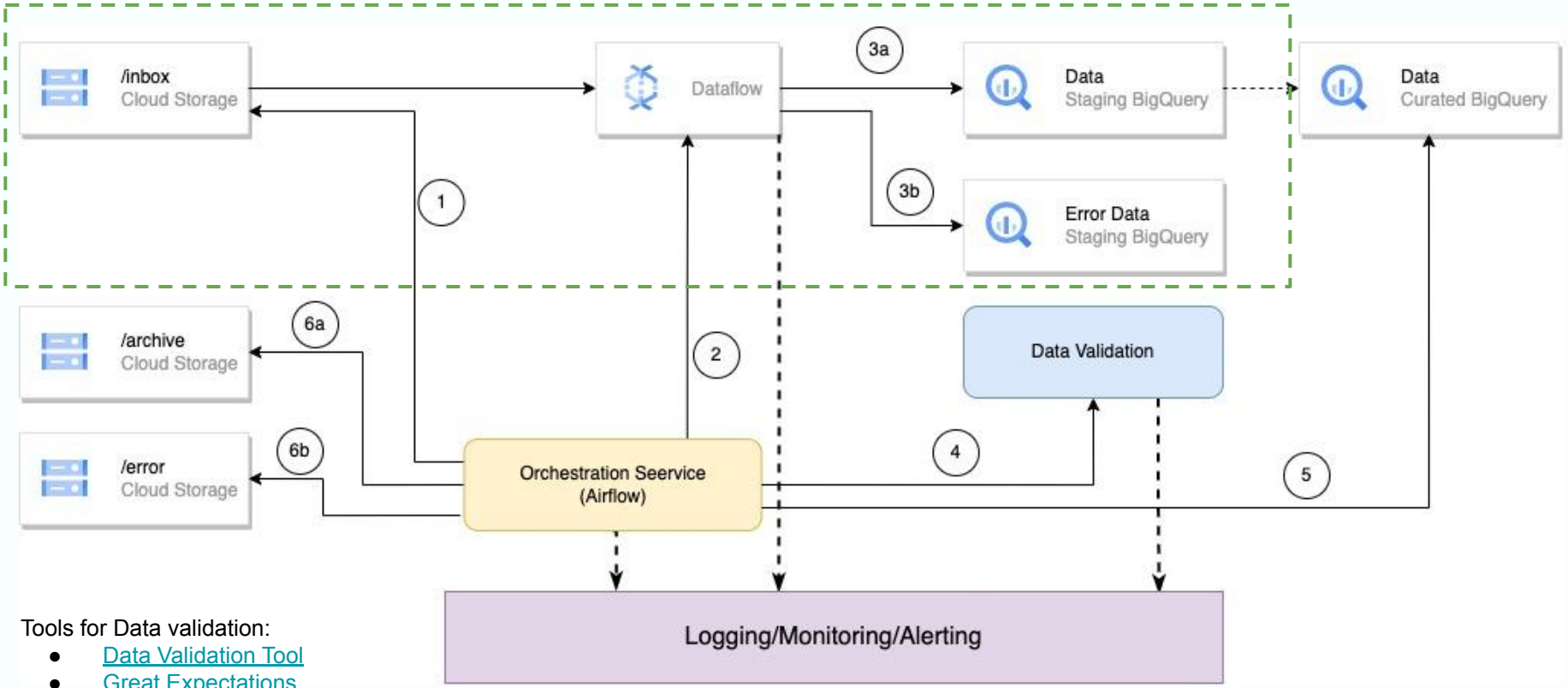
Batch Vs Streaming Code




```
Pipeline p = Pipeline.create();
    p.begin()
.apply(TextIO.Read.from("gs://...")
)


.apply(ParDo.of(new Filter1()))
.apply(new Group1())
.apply(ParDo.of(new Filter2())
.apply(new Transform1())

.apply(TextIO.Write.to("gs://..."))
  p.run();
```

```
Pipeline p = Pipeline.create();
    p.begin()
.apply(PubsubIO.Read.from("input_t
opic"))
.apply(SlidingWindows.of(60,
MINUTES))


.apply(ParDo.of(new Filter1()))
.apply(new Group1())
.apply(ParDo.of(new Filter2())
.apply(new Transform1())

.apply(PubsubIO.Write.to("output_t
opic"))
  p.run();
```

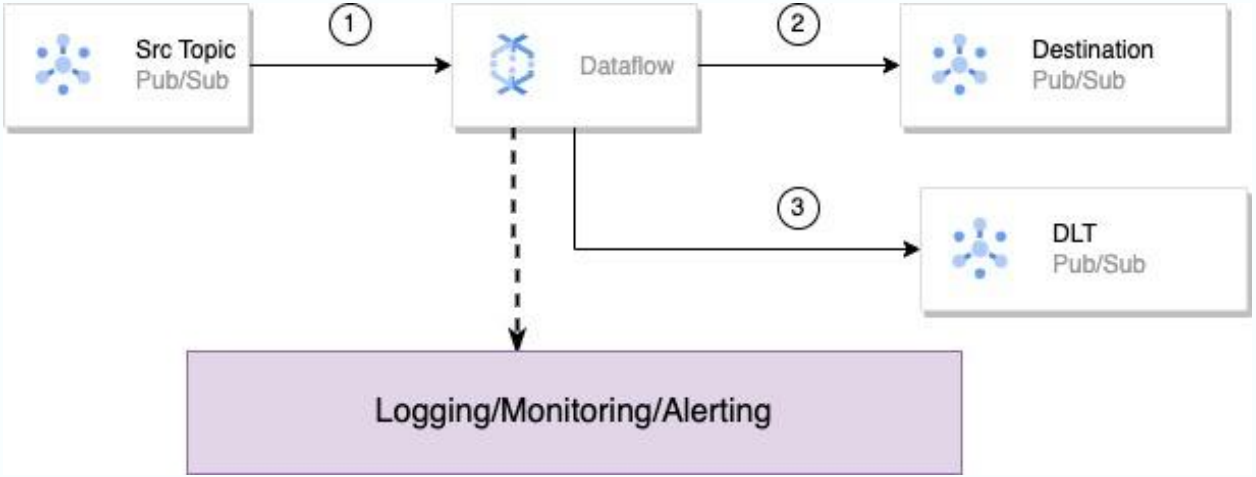
Batch Pipeline (Bounded Source)



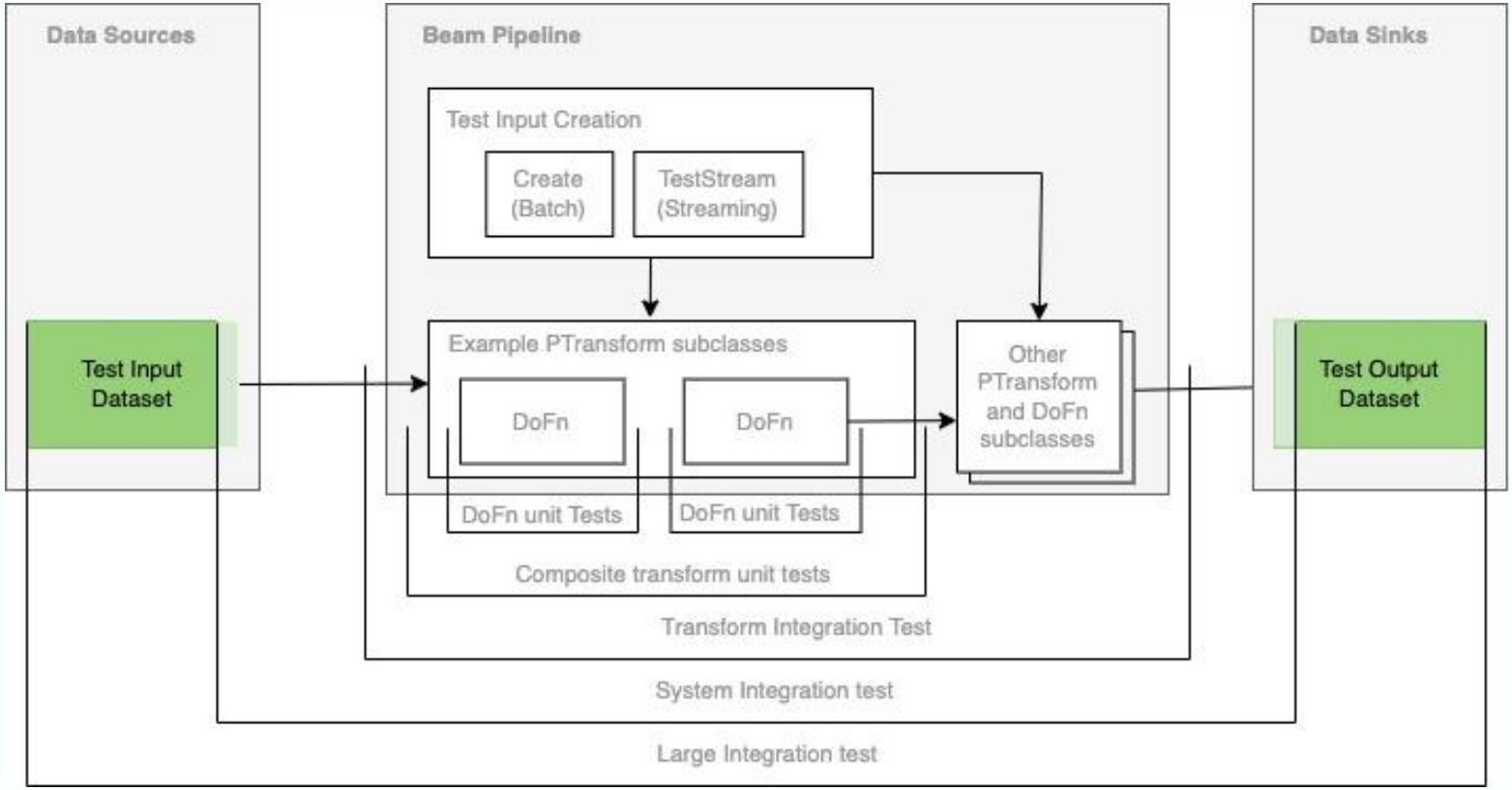
Tools for Data validation:

- Data Validation Tool
- Great Expectations

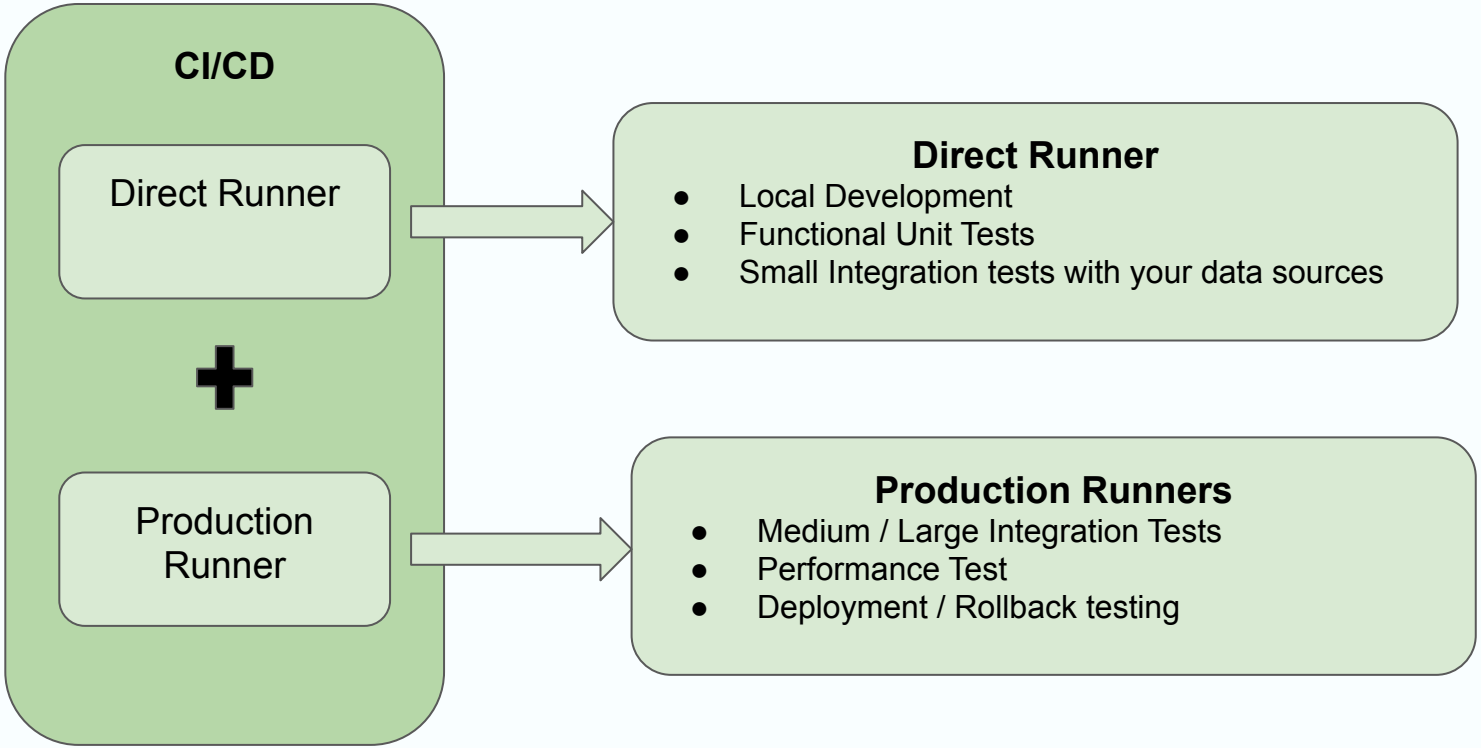
Streaming Pipeline (Unbounded Source)



Types of test



Testing Environment



Unit test

- Mini-pipelines that assert behavior of one small testable piece of your pipeline e.g. a single PTransform or DoFn.
- Runs quickly, locally, and without dependencies on external systems.

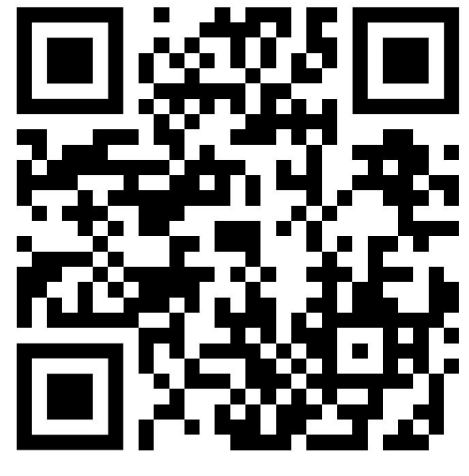
Test Classes

```
  WORDS = [ "hi", "there", "hi", "hi", "sue", "bob",
      "hi", "sue", "", "", "ZOW", "bob"]

# For tests, use TestPipeline in place of Pipeline

# when you create the pipeline object.

 with TestPipeline() as p:

     # Create an input PCollection.
      input = p | beam. Create(WORDS)

      # Apply the Count transform under test.
      output = input | beam.combiners.Count.PerElement()

      # Assert on the results.
      assert_that( output,
        equal_to([ ("hi", 4),("there", 1),
            ("sue", 2),("bob", 2),
            ("", 2),  ("ZOW", 1)]))
```

Example/Walkthrough: Unit Testing

Parse the csv data and generate valid and error output

- Example Code: [data-pipeline-testing/beam-testing-example]
- Run: python -m pytest --disable-warnings testing_pardo/data_validation_pardo_unittest.py



average_fxn uses the accumulator and compute the average

- Example Code: [data-pipeline-testing/beam-testing-example]
- Run: python -m pytest --disable-warnings testing_combinefxn/average_fxn_unittest.py

Try yourself: exercise contains divisible_by_pardo.py(DivisibleByDoFn). Write unittest.

Example/Walkthrough: Unit Testing

Apply fixed window and group the messages

- Code: testing_windows
- Run: python -m pytest --disable-warnings testing_windows/streaming_fixed_window_unittest.py

Try yourself: exercise contains a sliding window pipeline. Write a unit test for the pipeline.

Why Integration Tests?

Pipeline Functionality

Ensures that the entire pipeline is functioning correctly as a unified system.

Simulating real-world scenarios.

Input/Output Validation

Validate the interaction of the pipeline with external dependencies.

Performance and Scalability Testing

Identify bottlenecks, optimize resource utilization, and ensure the pipeline can handle the expected load

Error Handling and Fault Tolerance

Ensure that the pipeline can handle exceptions, recover from failures, and continue processing data reliably

Integration Tests

01. For every source of input data to your pipeline, create some known static test input data or data generators

02. Create output data/checksums/record counts that matches what you expect in your pipeline's final output

03. Create a TestPipeline

04. Create transform to create one or more PCollections from your static input data (if required)

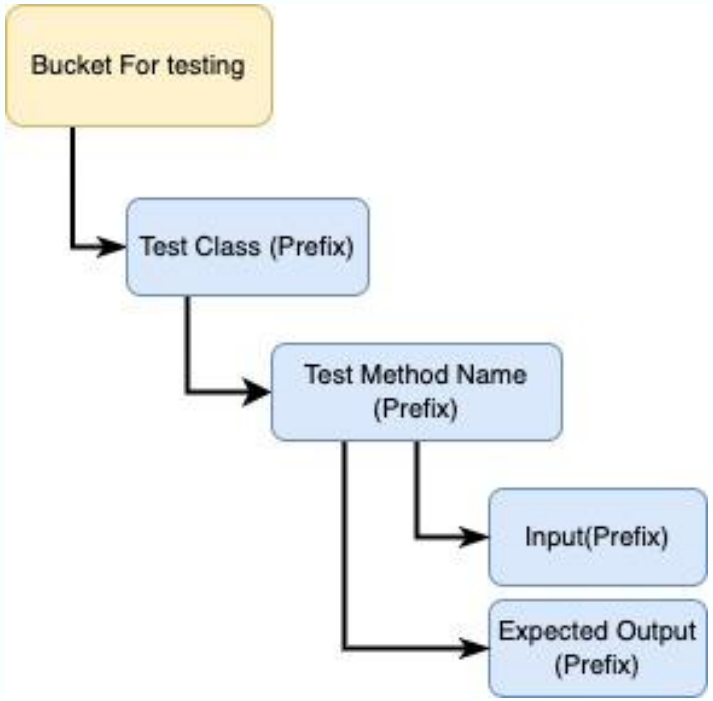
05. Apply your pipeline's transforms (if source/ sinks are not included)
Or call the pipeline with all required parameters

06. Write transform(s), use assert to verify that the contents of the final PCollections

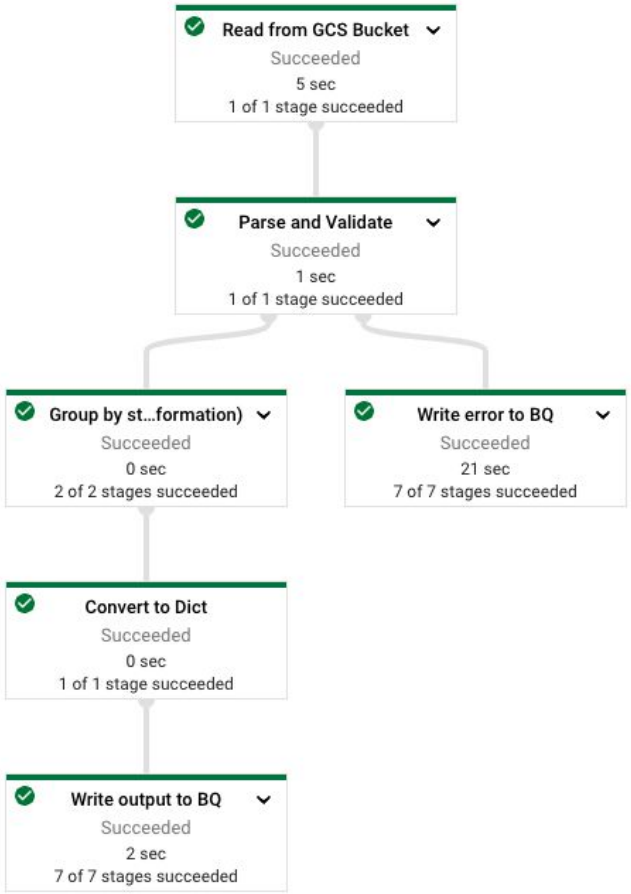
For input/output validation using PubSubMessageMatcher, BigqueryMatcher or custom comparison

Data for Integration Tests

- Naming conventions (GCS bucket for all input and expected output)
- Prepare the data and then upload to GCS bucket
  - For streaming, publish each line from the object to input_topic
  - For batch using BQ, load files from input prefix to BQ table
  - For batch using GCS, point to the location of input prefix



StoreInfo Pipeline (GCS to BQ)



GCS objects in CSV has data in the following format

txn_date, txn_id, store_id, product_name, qty, amount

Field name	Type	Mode
store_id	STRING	NULLABLE
sales	FLOAT	NULLABLE
start_date	DATETIME	NULLABLE
end_date	DATETIME	NULLABLE
products	RECORD	REPEATED
product_name	STRING	NULLABLE
qty	INTEGER	NULLABLE

Data table

Field name	Type	Mode
error	STRING	NULLABLE
payload	STRING	NULLABLE
error_step_id	STRING	NULLABLE

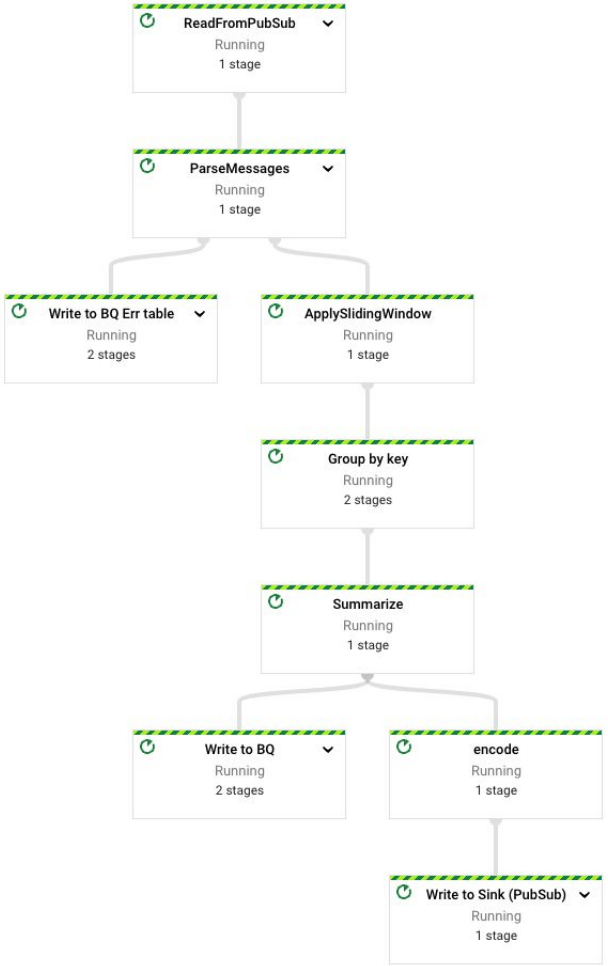
Error table

StoreInfo Pipeline (GCS to BQ)

- Code: /store-info-pipeline
- Data: Synthetic data for Store Info Pipeline
- Run test:

```
export TEMP_LOCATION= <<set the temp gcs location>>
export STAGING_LOCATION= <<set the staging location>>
export PROJECT_ID=<project_id>

python -m pytest --log-cli-level=INFO test/store_info_batch*.py

--test-pipeline-options="--project=${PROJECT_ID} --input_bucket=${INPUT_BUCKET}

--temp_location=${TEMP_LOCATION} --staging_location=${STAGING_LOCATION}

--setup=./setup.py"
```

Exercise: Convert it to streaming pipeline, data comes to pubsub topic and compute the total sales per store every 15 mins



Streaming Pipeline (taxirides)

- Code: [/taxi-rides-streaming-pipeline]
- Run:
  - Unittest

```
python -m pytest --disable-warnings
tests/taxirides_unittest.py
```

  - ITTest

```
pytest --log-cli-level=INFO test/taxirides_ittest.py
--test-pipeline-options="--runner=TestDataflowRunn
er --wait_until_finish_duration=100000
--temp_location=${TEMP_LOCATION}
--staging_location=${STAGING_LOCATION}
--project=${PROJECT_ID} --setup=./setup.py"
```

Integration Test - Deployment

Part of the integration testing role will be to establish the deployment path, which will become part of the final artifact.

Batch vs Stream

- With Batch we have choices around the use of templates vs running the pipeline directly.
- With Streaming we need to explore the method to replace the existing pipeline, unless this is the first time the pipeline is deployed.

Streaming Pipeline Deployment Options

Update

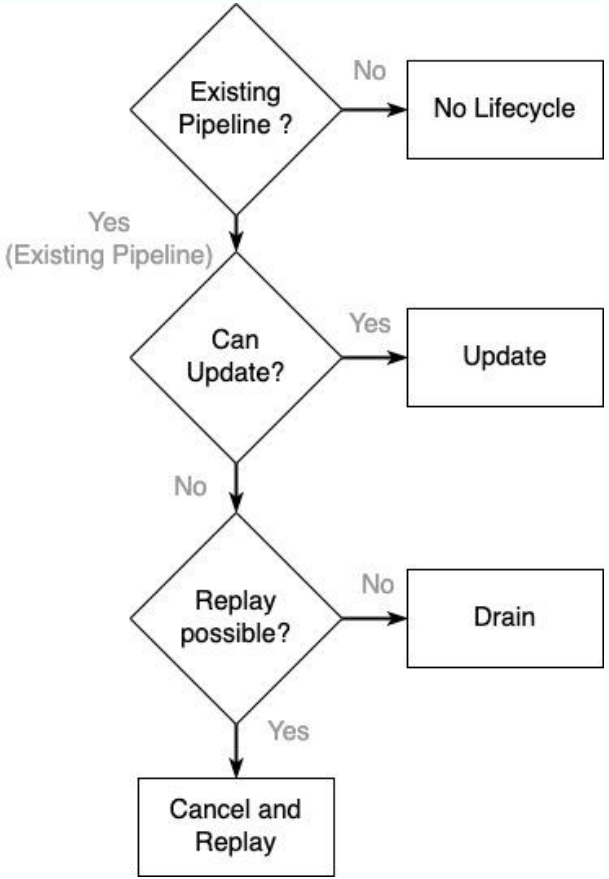
- When you update a job on the Cloud Dataflow service, you replace the existing job with a new job that runs your updated pipeline code. The Cloud Dataflow service retains the job name, but runs the replacement job with an updated job id.

Drain

- Using the Drain option to stop your job tells the Cloud Dataflow service to finish your job in its current state.

Cancel

- Using the Cancel option to stop your job tells the Cloud Dataflow service to cancel your job immediately.



Other avenues on testing

Capacity planning and optimization based on machine type (CPU/memory)

Expected service-level objectives (SLOs) in terms of daily volume, event throughput and/or end-to-end latency

Pipeline's total cost of ownership (TCO)

References:

- Benchmarking Beam pipelines on Dataflow
- Benchmarking your Dataflow jobs for performance, cost and capacity planning

Bipin Upadhyaya

QUESTIONS?

References:

- Get back to basics with testing data pipelines: two orthogonal planes
- Test Your Pipeline
- Benchmarking your Dataflow jobs for performance, cost and capacity planning
- Examples and Cookbook from Apache Beam Github Repo

Thanks for Help/Suggestion
-Hariprabhaa Murugesan
-Pramod Rao
-Prathap Kumar Parvathareddy
-Vince Gonzalez