

Dataflow for Beginners

Presented by:

Chamikara Jayalath, Derrick Williams

Software Engineers at Google
Working on Apache Beam and Dataflow



BEAM
SUMMIT

July 8-10, 2025

New York, NY. USA

Agenda

- 01 Beam Yaml Introduction, Positioning, and Advantages
- 02 Beam Yaml Pipeline Structure and Examples
- 03 Basic Transforms Examples
- 04 Advanced Topics (Providers, Inlining, Jinja, ML, Testing)
- 05 Job Builder Introduction and Relationship to Beam YAML
- 06 Job Builder Ease-of-Use Features
- 07 Customizing Data Movement Jobs
- 08 Data Processing Patterns (Filter, Map, SQL, Error Handling)
- 09 Working with YAML in the Job Builder (Export, Import, Yaml Editor)



BEAM
SUMMIT

01

Beam Yaml Introduction, Positioning, and Advantages

Derrick Williams

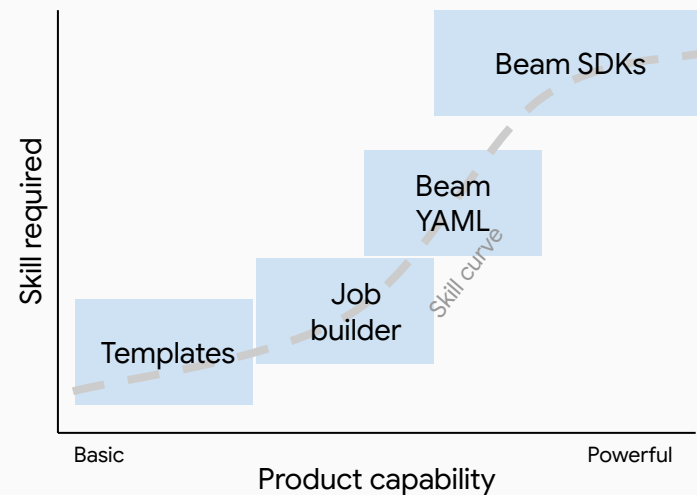
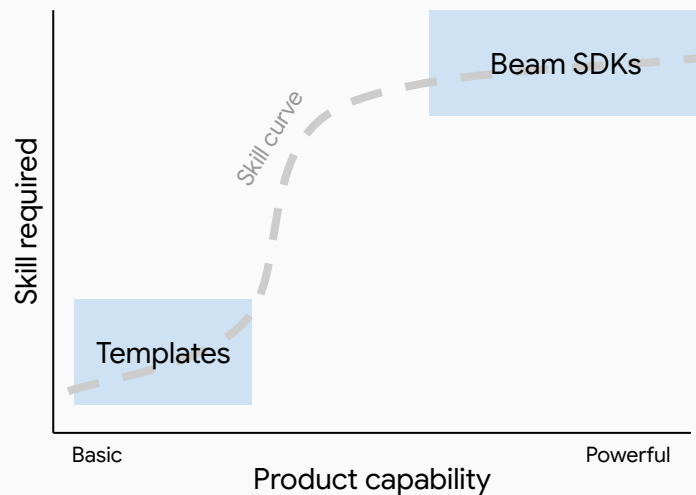


BEAM
SUMMIT

Current Beam/Dataflow State

- Python, Golang, Java, etc. SDK's to give users a choice of language
 - Still requires programming language knowledge and Beam model experience
- Dataflow Templates
 - Only works if someone has written a pipeline that exactly matches your use case
 - Even the smallest tweaks typically require as much knowledge as writing a pipeline from scratch.

YAML helps fill a missing gap between Templates and Beam SDKs



BEAM
SUMMIT

Core Goals of Beam YAML Design

- Schema-first design (i.e. structured data via Beam Row)
 - But allow for schemaless
- Deliver main Beam functionality
 - IO's, Windowing, Turnkey transforms, etc.
- Robust error handling on a per-transform basis
- Easy syntax with syntactic sugar where possible
- Built-in transforms and IO's can be executed using Java or Python interchangeably
 - Affinity heuristic will optimize pipeline for specific SDK
- Provide Google-provided Dataflow Template for running YAML pipelines
 - Users will be able to submit YAML jobs directly to Dataflow
- Allow for code translation for getting started with Beam



02

Beam Yaml Pipeline Structure and Examples

Derrick Williams



BEAM
SUMMIT

Basic Read-Write YAML Pipeline

```
pipeline:  
  type: chain  
  
  source:  
    type: ReadFromPubSub  
    config:  
      subscription: ...  
      schema: ...  
  
  sink:  
    type: WriteToBigQuery  
    config:  
      table: ...
```

ReadFromPubSub



WriteToBigQuery



BEAM
SUMMIT

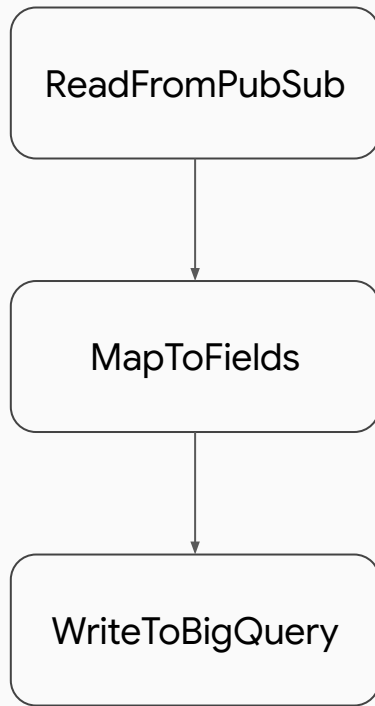
Add a Transformation

```
pipeline:
  type: chain

  source:
    type: ReadFromPubSub
    config:
      subscription: ...
      schema: ...

  transforms:
  - type: MapToFields
    config:
      language: python
      fields:
        name: "name.upper()"
        age: "age + 20"

  sink:
    type: WriteToBigQuery
    config:
      table: ...
```



BEAM
SUMMIT

Many transforms can be chained

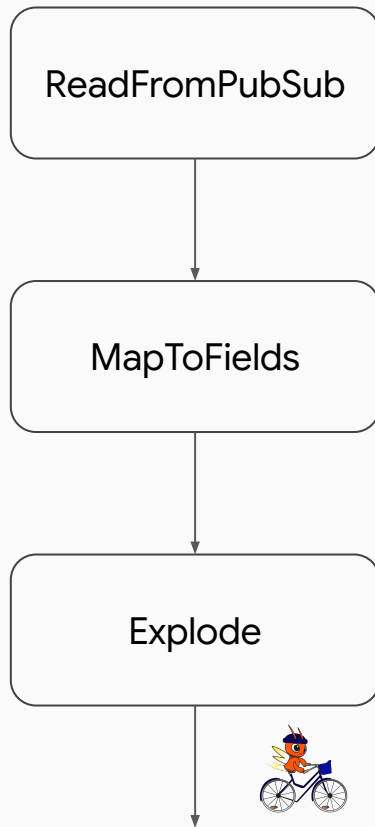
```
pipeline:
  type: chain

  transforms:
    - type: ReadFromPubSub
      config:
        subscription: ...
        schema: ...

    - type: MapToFields
      config:
        language: python
        fields:
          name: "name.upper()"
          age: "age + 20"

    - type: Explode
      config:
        fields: [pets]

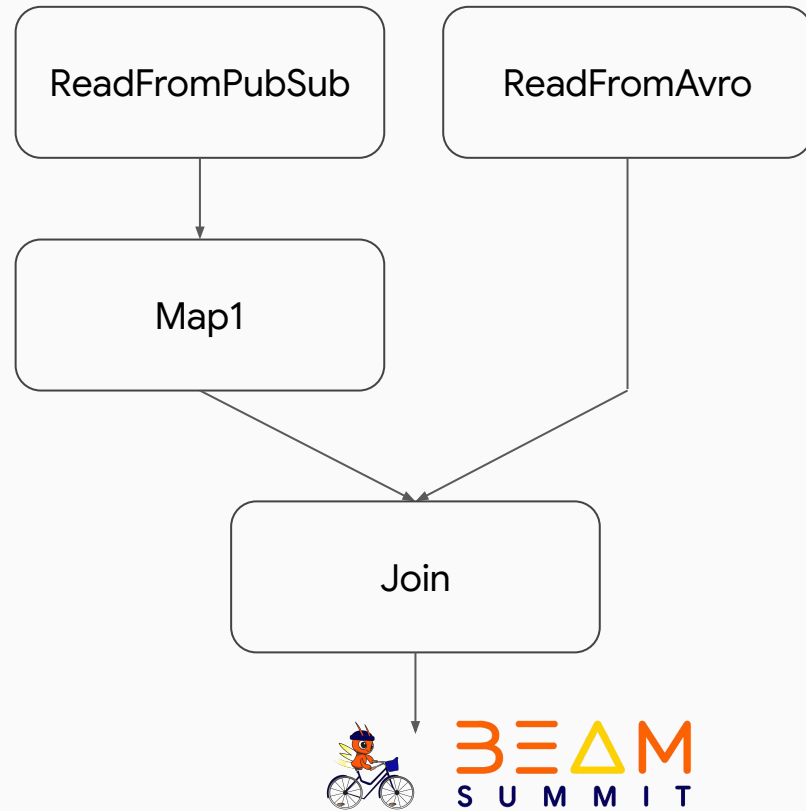
    ...
```



BEAM
SUMMIT

Pipelines need not be linear

```
pipeline:  
  transforms:  
    - type: ReadFromPubSub  
      config: ...  
  
    - type: MapToFields  
      name: Map1  
      input: ReadFromPubSub  
      config: ...  
  
    - type: ReadFromAvro  
      config:  
        path: "gs://..."  
  
    - type: Join  
      input:  
        left: MapToFields  
        right: ReadFromAvro  
      config:  
        ...  
    ...
```



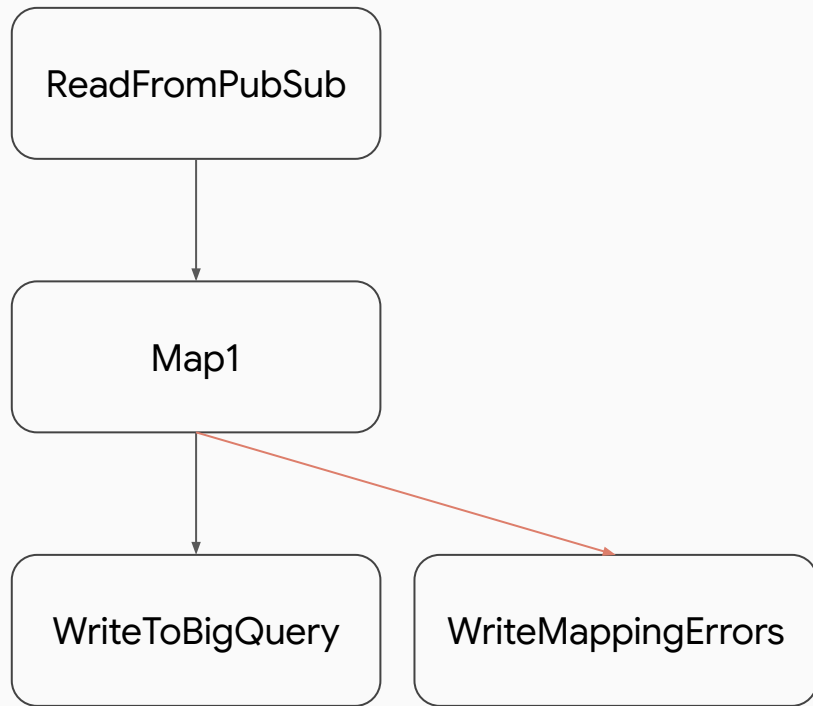
Sophisticated error handling

```
pipeline:
  transforms:
    - type: ReadFromPubSub
      config: ...

    - type: MapToFields
      name: Map1
      input: ReadFromPubSub
      config:
        ...
        error_handling:
          output: errors

    - type: WriteToBigQuery
      input: Map1
      config: ...

    - type: WriteToJson
      name: WriteMappingErrors
      input: MapToFields.errors
      config:
        path: "/path/to/errors.json"
```



BEAM
SUMMIT

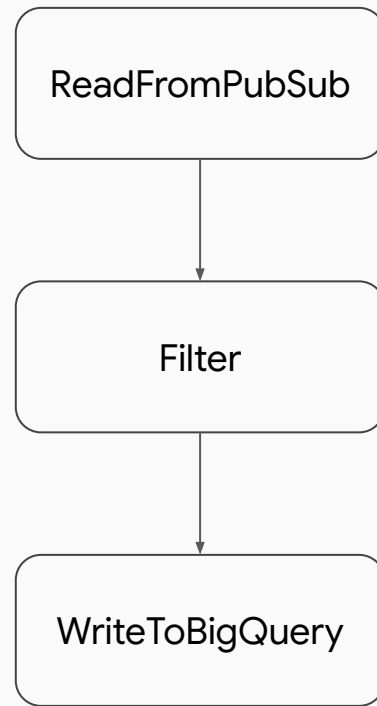
Built-in Jinja2 templatzation

```
pipeline:
  type: chain

  source:
    type: ReadFromPubSub
    config:
      subscription: ...
      schema: ...

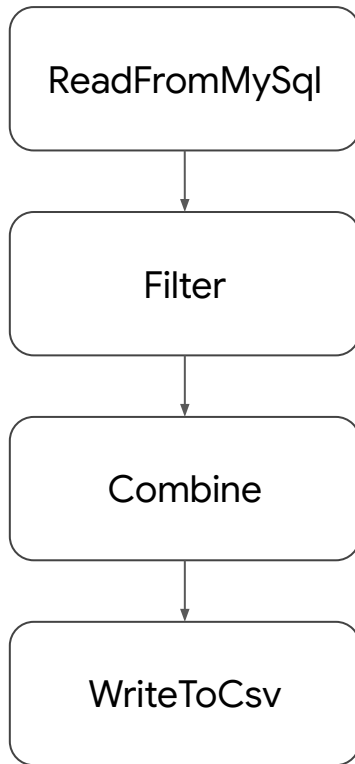
  transforms:
  - type: Filter
    config:
      language: python
      keep: "age > {{threshold}}"

  sink:
    type: WriteToBigQuery
    config:
      table: "{{table_prefix}}-{{threshold}}"
```



Example pipeline

```
pipeline:
  type: chain
  transforms:
    - type: ReadFromMySQL
      config:
        url: jdbc:mysql://host:port/database
        table: transactions
        username: 'username'
        password: 'password'
    - type: Filter
      config:
        language: python
        keep: category == "Electronics"
    - type: Combine
      name: CountNumberSold
      config:
        group_by: product_name
        combine:
          num_sold:
            value: product_name
            fn: count
    - type: WriteToCsv
      config:
        path: electronics.csv
```



Beam YAML Transforms - IO and others

- ReadFrom/WriteToAvro
- ReadFrom/WriteToCsv
- ReadFrom/WriteToJson
- ReadFrom/WriteToParquet
- ReadFrom/WriteToMySQL
- ReadFrom/WriteToBigQuery
- ReadFrom/WriteToPubSub
- ReadFrom/WriteToKafka
- ...
- Full list at <https://beam.apache.org/releases/yamldoc/current/>



BEAM
SUMMIT

Beam YAML Transforms - IO and others

- Utility
 - Create
 - Flatten
 - WindowInto
 - LogForTesting
 - AssertEqual
- Mapping
 - MapToFields
 - Explode
 - Filter
 - Partition
- Aggregation
 - Combine
- ML
 - MLTransform
 - Enrichment
 - RunInference
- Other
 - Sql
 - Join
- ...
- Full list at <https://beam.apache.org/releases/yamldoc/current/>



BEAM
SUMMIT

Running Beam YAML

Beam YAML jobs can be launched

- From the template UI
- Via the gcloud tool via Dataflow

```
$ gcloud dataflow yaml run /path/to/my.yaml
```

- Locally

```
$ python -m apache_beam.yaml.main --yaml_pipeline_file=/path/to/my.yaml
```

Can set runner using --runner or in YAML options block

Dataflow

Create job from template

Overview

Monitoring

Jobs

Pipelines

Workbench

Snapshots

SQL Workspace

Dataflow templates

Launch jobs from Google-provided or custom templates

Job builder

Build a job with sources, transforms, and sinks

Job name *

my-yaml-job

Must be unique among running jobs

Some locations have been restricted due to a policy set by your organization.

Regional endpoint *

us-central1 (Iowa)

Choose a Dataflow regional endpoint to deploy worker instances and store job metadata. You can optionally deploy worker instances to any available Google Cloud region or zone by using the worker region or worker zone parameters. Job metadata is always stored in the Dataflow regional endpoint. [Learn more](#)

Dataflow template *

YAML

The YAML Template is used to run Dataflow pipelines written in Beam YAML. The YAML pipeline can be passed to the template directly as a raw string or the location of a Beam YAML pipeline file stored in Google Cloud Storage can optionally be passed. For launching a Beam YAML pipeline directly from the gcloud command line, see <https://cloud.google.com/sdk/gcloud/reference/dataflow/yaml> For more information on Beam YAML, see <https://beam.apache.org/documentation/sdks/yaml/>.

Optional Parameters

Input YAML pipeline spec.

A yaml description of the pipeline to run.

gs:// Input YAML pipeline spec file in Cloud Storage. **BROWSE**

A file in Cloud Storage containing a yaml description of the pipeline to run.

Input Jinja preprocessing variables.

g the Jinja preprocessor on the provided yaml



03

Basic Transforms Examples

Derrick Williams



BEAM
SUMMIT

Mapping Transforms

- MapToFields
 - Map the input collection to a schema where each field can be defined by a UDF (user-defined function)
 - Expressions (Generic/Python/Java/SQL/JS*)
 - Callables (Python/Java/JS*)
 - File (Python/Java/JS*)
- Filter
 - Filter records in a collection given a predicate
 - Same expression, callable, file capabilities as MapToFields
- Explode
 - Produce elements for each iterable field specified
 - `{name='a', iter=[1, 2, 3]}` → `{name='a', iter=1}`, `{name='a', iter=2}`, and `{name='a', iter=3}`
- Partition
 - Split input collection into multiple output collections based on condition
- AssignTimestamps
 - Mark field in collection as Timestamp - useful for streaming pipeline with embedded timestamps

* JavaScript support is experimental

Generic MapToFields

```
- type: MapToFields
name: RenameAndMapCustomFields
input: ReadFromCsv
config:
  fields:
    new_col: col1
    int_literal: 389
    float_literal: 1.90216
    str_literal: "example"
```

col1
1
2
3



new_col	int_literal	float_literal	str_literal
1	389	1.90216	"example"
2	389	1.90216	"example"
3	389	1.90216	"example"

MapToFields

```
- type: MapToFields
name: RenameAndMapCustomFields
input: ReadFromCsv
config:
  language: python
  fields:
    myNewStr:
      expression: "myOldStr"
    myNewNum:
      callable: "lambda row: row.myOldNum * 2"
    myNewName:
      path: "udf.py"
      name: "to_uppercase"
```

myOldNum	myOldStr	myOldName
1	"a"	"John"
2	"b"	"Jane"
3	"c"	"Apache Beam"



myNewNum	myNewStr	myNewName
2	"a"	"JOHN"
4	"b"	"JANE"
6	"c"	"APACHE BEAM"

MapToFields Callable

```
- type: MapToFields
name: RenameAndMapCustomFields
input: ReadFromCsv
config:
  language: python
  fields:
    json_str:
      callable: |
        import json
        def process(row):
            json_str = json.dumps(row._asdict())
            return json_str
```

col1	col2	col3
1	"a"	"John"
2	"b"	"Jane"
3	"c"	"Apache Beam"



json_str
'{"col1": 1, "col2": "a", "col3": "John"}'
'{"col1": 2, "col2": "b", "col3": "Jane"}'
'{"col1": 3, "col2": "c", "col3": "Apache Beam"}'

MapToFields Output Types

```
- type: MapToFields
  name: RenameAndMapCustomFields
  input: ReadFromCsv
  config:
    language: python
    fields:
      json_str: # -----> Any
      callable: |
        import json
        def process(row):
            json_str = json.dumps(row._asdict())
            return json_str
- type: SomeJavaTransform
  config:
    ...
```

java.lang.IllegalArgumentException:
Failed to decode Schema due to an error
decoding Field proto:

```
name: "json_str"
type {
  nullable: true
  logical_type {
    urn: "beam:logical:pythonsdk_any:v1"
  }
}
```

MapToFields Output Types

```
- type: MapToFields
  name: RenameAndMapCustomFields
  input: ReadFromCsv
  config:
    language: python
    fields:
      json_str:
        callable: |
          import json
          def process(row):
            json_str = json.dumps(row._asdict())
            return json_str
        output_type: string # -----> String
- type: SomeJavaTransform
  config:
    ...
```



MapToFields Create Schema

```
- type: MapToFields
name: RenameAndMapCustomFields
input: ReadFromCsv
config:
  language: python
  fields:
    col1:
      callable: 'lambda row: row.col1'
      output_type: integer
    col2:
      callable: 'lambda row: row.col2'
      output_type: string
    col3:
      callable: 'lambda row: row.col3'
      output_type: string
```

col1	col2	col3
1	"a"	"John"
2	"b"	"Jane"
3	"c"	"Apache Beam"



col1	col2	col3
1	"a"	"John"
2	"b"	"Jane"
3	"c"	"Apache Beam"

MapToFields Java

```
- type: MapToFields
name: RenameAndMapCustomFields
input: ReadFromCsv
config:
  language: java
  fields:
    myNewStr:
      expression: "myOldStr"
    myNewNum:
      callable: |
        import org.apache.beam.sdk.values.Row;
        import java.util.function.Function;
        public class MyFunction implements Function<Row, String> {
          public String apply(Row row) {
            return row.getString("myOldNum") * 2;
          }
        }
    myNewName:
      path: "udf.java"
      name: "to_uppercase"
```

myOldNum	myOldStr	myOldName
1	"a"	"John"
2	"b"	"Jane"
3	"c"	"Apache Beam"



myNewNum	myNewStr	myNewName
2	"a"	"JOHN"
4	"b"	"JANE"
6	"c"	"APACHE BEAM"

MapToFields SQL

```
SELECT
  `timestamp`,
  UPPER(myOldName) AS myNewName,
  "myOldNum + 1" AS myNewNum
FROM PCOLLECTION;
```

```
- type: MapToFields
  name: RenameAndMapCustomFields
  input: ReadFromCsv
  config:
    language: sql
    fields:
      timestamp:
        expression: "`timestamp`"
      myNewNum:
        expression: "myOldNum + 1"
      myNewName:
        expression: "UPPER(myOldName)"
```

timestamp	myOldNum	myOldName
1	1	"John"
2	2	"Jane"
3	3	"Apache Beam"

timestamp	myNewNum	myNewName
1	2	"JOHN"
2	3	"JANE"
3	4	"APACHE BEAM"

Aggregation Transforms

- Combine
 - Aggregate the input collection according to a given aggregation method
 - Built-in
 - sum, max, min, all, any, mean, count, group, concat
 - Custom transform
 - Some function that implements `core.CombineFn`
 - Supports multiple languages - Python, SQL

Basic Combine

```
- type: Combine
  config:
    group_by: col1
    combine:
      col2:
        value: col2
        fn:
          type: sum
    count:
      value: col1
      fn:
        type: count
```

col1	col2
'a'	1
'b'	2
'a'	3



col1	col2	count
'a'	4	2
'b'	2	1

Basic Combine

```
- type: Combine
  config:
    group_by: col1
    combine:
      col2:
        value: col2
        fn: sum
      count:
        value: col1
        fn: count
```

col1	col2
'a'	1
'b'	2
'a'	3



col1	col2	count
'a'	4	2
'b'	2	1

Basic Combine

```
- type: Combine
  config:
    group_by: col1
    combine:
      col2: sum
    count:
      value: col1
      fn: count
```

col1	col2
'a'	1
'b'	2
'a'	3



col1	col2	count
'a'	4	2
'b'	2	1

SQL Combine

```
- type: Combine
  config:
    language: sql
    group_by: id
    combine:
      num_values: "count(*)"
      total: "sum(col1)"
```

id	col1
1	1
2	2
1	3



id	num_values	total
1	2	4
2	1	2

Custom Combine Fn

```
- type: Combine
  config:
    language: python
    group_by: id
    combine:
      top_two:
        value: "col1 + col2"
        fn:
          type: 'apache_beam.transforms.combiners.TopCombineFn'
          config:
            n: 2
```

id	col1	col2
1	1	5
2	2	6
1	3	7
1	4	8



id	top_two
1	[12,10]
2	[8]

A set of Beam's built-in CombineFn's can be found at

https://beam.apache.org/releases/pydoc/current/apache_beam.transforms.combiners.html

04

Advanced Topics (Providers, Inlining, Jinja, ML, Testing)

Derrick Williams



BEAM
SUMMIT

Custom Transforms

Though we aim to provide a rich set of built-in transforms, invariable customers will want to provide their own custom transformations.

- We provide this extensibility via *Providers*
- Allows one to use the full expressivity of Beam SDKs
- Custom transforms can be authored and deployed in multiple ways
 - Inline
 - PyPi packages
 - Java jars
 - Maven/gradle targets
 - YAML
- Customers can use this to provide their own custom transforms, and their users can then reference and use them.

A tutorial on creating a custom Java provider can be found at <https://github.com/Polber/beam-yaml-xlang>

Provider Example (Java Jar)

```
pipeline:
  type: chain

  transforms:
    - type: ReadFromPubSub
      config: ...

    - type: MyCustomTransform
      config: ...

    - type: WriteToBigQuery
      config: ...

  providers:
    - type: javaJar
      config:
        jar: "/path/or/url/to/myExpansionService.jar"
      transforms:
        MyCustomTransform: "urn:for:my:service"
```

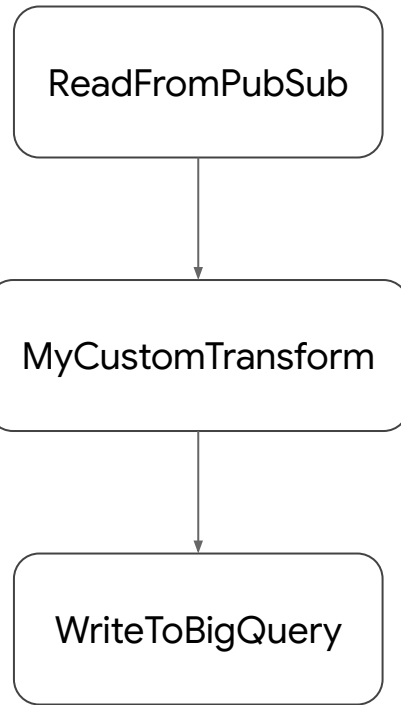
use

definition

ReadFromPubSub

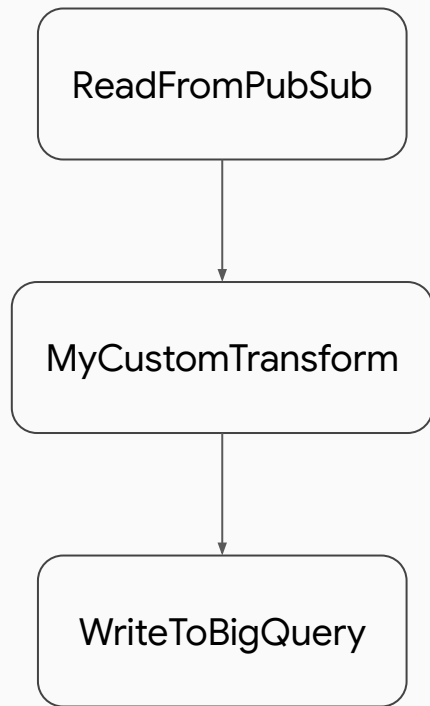
MyCustomTransform

WriteToBigQuery



Provider Example (PyPi)

```
pipeline:  
  type: chain  
  
  transforms:  
    - type: ReadFromPubSub  
      config:  
        subscription: ...  
    - type: MyCustomTransform  
      config:  
        param: value  
    - type: WriteToBigQuery  
      config:  
        table: ...  
  
  providers:  
    - type: pythonPackage  
      config:  
        packages:  
          - my_pypi_package>=version  
          - /path/to/local/package.zip  
      transforms:  
        MyCustomTransform: "pkg.subpkg.MyTransform"
```



BEAM
SUMMIT

Additional Custom Transforms

There are cases where the overhead of supplying a custom packaged provider is not worth the time investment or overhead. In those cases, Beam YAML provides a method for creating in-line Python transforms.

There are currently two ways to implement these in-line Python transforms

- Using PyTransform
- Leveraging the Providers framework (python Provider)

Best Practice:

- Output a Beam Row - outputting schema'd data helps integration with existing Beam YAML transforms

PyTransform

- Typically used to call some arbitrary Transform that is not specifically wrapped for Beam
- This could be a transform built into Beam, or one packaged with the pipeline

```
- type: PyTransform
  config:
    constructor: apache_beam.pkg.module.SomeTransform
    args: [1, 'foo']
    kwargs:
      baz: 3
```

PyTransform __constructor__

```
- type: PyTransform
config:
  constructor: __constructor__
  kwargs:
    source: |
      class MyPTransform(beam.PTransform):
        def __init__(self, inc):
          self._inc = inc
        def expand(self, pcoll):
          return pcoll | beam.Map(lambda x: beam.Row(out=x.col2 + self._inc))

    inc: 10
```

col1	col2
1	4
2	5
3	6



out
14
15
16

PyTransform __callable__

```
- type: PyTransform
config:
  constructor: __callable__
  kwargs:
    source: |
      def my_pttransform(pcoll, inc):
        return pcoll | beam.Map(lambda x: beam.Row(out=x.col2 + inc))
    inc: 10
```

col1	col2
1	4
2	5
3	6



out
14
15
16

Python Provider

```
pipeline:
  transforms:
    - ...
    - type: MyTransform
      input: ...
      config:
        inc: 10
    - ...

providers:
  - type: python
    config: {}
    transforms:
      MyTransform: |
        @beam.ptransform_fn
        def my_pttransform(pcoll, inc):
            return pcoll | beam.Map(lambda x: beam.Row(out=x.col2 + inc))
```

col1	col2
1	4
2	5
3	6



out
14
15
16

Jinja Templatization

There are many cases where a static YAML file, or components, may be cumbersome to share, negating the benefit of YAML being easy to share across teams.

- Sensitive information embedded (SPII)
- Different teams/users need minor variations (though possibly clearly defined subsets of use-cases)
- Copy/paste-ing YAML blocks can be disorganized and difficult to maintain across an organization
- etc.

Jinja Templatization

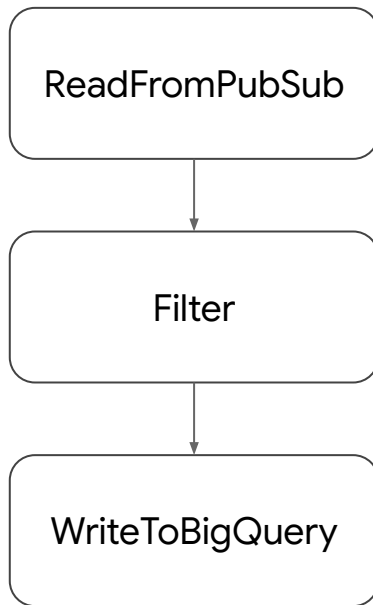
Jinja framework provides standardized method for creating template YAML pipelines

- Inject variables, such as SPII, when running the pipeline, rather than embedding
- Allows for dynamic construction of pipeline graphs based on runtime parameters
- Import central-hosted YAML blocks/files

More information on Jinja syntax can be found at <https://jinja.palletsprojects.com/en/3.1.x/templates/#>

Variable injection

```
pipeline:
  type: chain
  transforms:
    - type: ReadFromPubSub
      config:
        subscription: ...
        format: ...
        schema: ...
    - type: Filter
      config:
        language: python
        keep: "age > {{threshold}}"
    - type: WriteToBigQuery
      config:
        table: "my_project.my_dataset.my_table_staging"
```

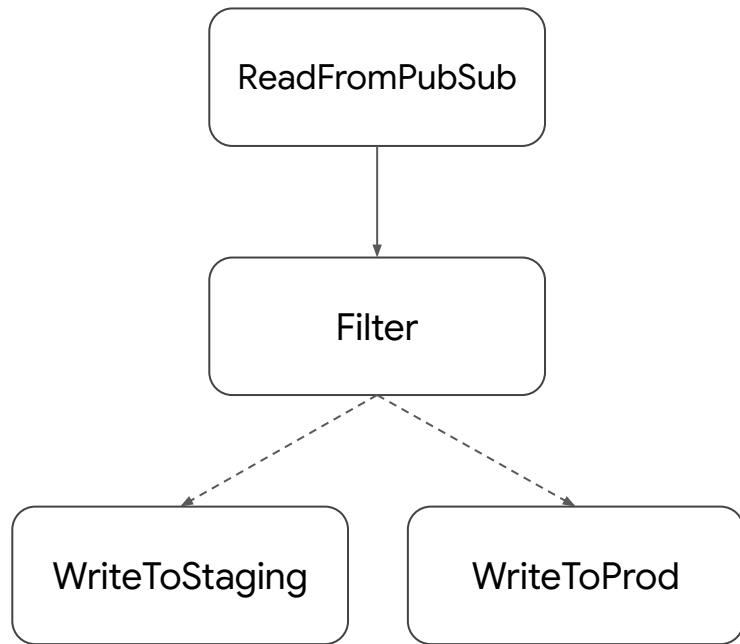


```
python -m apache_beam.yaml.main --yaml_pipeline_file=pipeline.yaml --jinja_variables='{"threshold": "5"}'
```

Dynamic graph construction

```
pipeline:
  type: chain
  transforms:
    - type: ReadFromPubSub
      config:
        subscription: ...
        format: ...
        schema: ...
    - type: Filter
      config:
        language: python
        keep: "age > {{threshold}}"

{% if use_staging == "true" %}
  - type: WriteToBigQuery
    name: WriteToStaging
    config:
      table: "my_project.my_dataset.my_table_staging"
{% else %}
  - type: WriteToBigQuery
    name: WriteToProd
    config:
      table: "my_project.my_dataset.my_table"
{% endif %}
```



```
python -m apache_beam.yaml.main --yaml_pipeline_file=pipeline.yaml --jinja_variables='{"threshold": "5", "use_staging": "true"}'
```

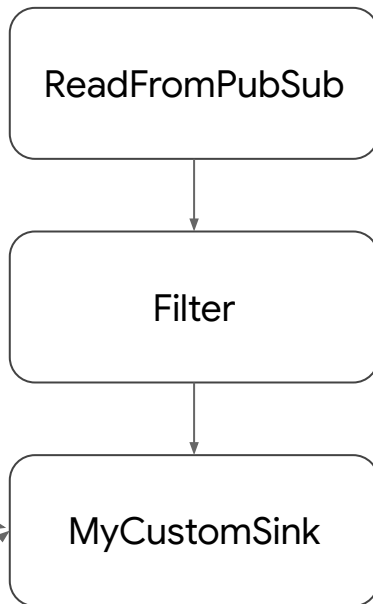
Easily share transforms catalogs

```
{% include 'gs://my-bucket/path/to/providers.yaml' %}

pipeline:
  type: chain
  transforms:
    - type: ReadFromPubSub
      config:
        subscription: ...
        format: ...
        schema: ...
    - type: Filter
      config:
        language: python
        keep: "age > {{threshold}}"
    - type: MyCustomSink
      config:
        ...
```

providers.yaml

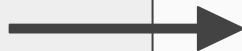
```
providers:
  - type: pythonPackage
    config:
      packages:
        - my_pypi_package>=version
        - /path/to/local/package.zip
  transforms:
    MyCustomSink: "pkg.subpkg.PTransformClassOrCallable"
```



ML – RunInference

A transform that takes the input rows, containing examples (or features), for use on an ML model. The transform then appends the inferences (or predictions) for those examples to the input row.

```
pipelines:  
- pipeline:  
  type: chain  
  transforms:  
    - type: Create  
    config:  
      elements:  
        - question: "What is a car?"  
        - question: "Where is the Eiffel Tower located?"  
    - type: RunInference  
  config:  
    model_handler:  
      type: "VertexAIModelHandlerJSON"  
    config:  
      endpoint_id: 9157860935048626176  
      project: "apache-beam-testing"  
      location: "us-central1"  
    preprocess:  
      callable: 'lambda x: {"prompt": x.question}'  
    private: false
```



```
[Row(inference=PredictionResult(example={'prompt': 'What is a car?'}, inference={'content': 'A  
car is a wheeled, self-propelled motor vehicle used for transportation. Cars are typically  
powered by an internal combustion engine, although some are powered by electric motors.  
Cars can be used for personal transportation, commercial transportation, or both.',  
'citationMetadata': {'citations': []}}...]
```



BEAM
SUMMIT

ML – MLTransform

Use MLTransform to apply common machine learning (ML) processing tasks on keyed data.

```
pipelines:
# MLTransform with write_artifact_location
- pipeline:
  type: chain
  transforms:
    - type: Create
      config:
        elements:
          - {num: 0, text: 'To be or not to be'}
          - {num: 2, text: 'I think, therefore I am'}
          - {num: 5, text: 'The only thing we have to fear is fear itself'}
          - {num: 8, text: 'Be the change you wish to see in the world'}
    - type: MLTransform
      config:
        write_artifact_location: "{TEMP_DIR}"
      transforms:
        - type: ScaleTo01
          config:
            columns: [num]
        - type: ScaleByMinMax
          config:
            columns: [num]
            min_value: 0
            max_value: 100
    - type: MapToFields
      config:
        language: python
        fields:
          num_scaled:
            callable: 'lambda x: x.num[0]'
    - type: AssertEqual
      config:
        elements:
          - {num_scaled: 0.0}
          - {num_scaled: 25.0}
          - {num_scaled: 62.5}
          - {num_scaled: 100.0}
  options:
    yml_experimental_features: ['ML']
```



Testing - example pipeline

```
pipeline:
  transforms:
    - type: ReadFromText
      name: Read from GCS
      config:
        path:
          gs://dataflow-samples/shakespeare/kinglear.txt
      - type: MapToFields
        name: Split words
        config:
          language: python
        fields:
          word:
            callable: |
              import re
              def all_words(row):
                return re.findall(r'[a-z]+' , row.line.lower())
            value: 1
          input: Read from GCS
      - type: Explode
        name: Explode word arrays
        config:
          fields: [word]
          input: Split words
  ...
```

```
...
  - type: Combine
    name: Count words
    config:
      group_by: [word]
      combine:
        value: sum
      input: Explode word arrays
  - type: MapToFields
    name: Format output
    config:
      language: python
      fields:
        output: "word + ' ' + str(value)"
      input: Count words
  - type: WriteToText
    name: Write to GCS
    config:
      path: gs://bucket/counts.txt
      input: Format output
```

```
tests: []
```



BEAM
SUMMIT

Testing - testing section and execution

Testing section:

tests:

- name: MyRegressionTest

mock_outputs:

- name: Read from GCS

elements:

- line: "Nothing can come of nothing"

expected_inputs:

- name: Write to GCS

elements:

- output: 'nothing: 2'

- output: 'can: 1'

- output: 'come: 1'

- output: 'of: 1'

Execution:

```
$ python -m apache_beam.yaml.main --yaml_pipeline_file=wordcount.yaml --tests
```



BEAM
SUMMIT

More Information

- Beam YAML docs:
 - <https://beam.apache.org/documentation/sdks/yaml/>
- Creating a custom Beam Java transform for Beam YAML:
 - <https://github.com/Polber/beam-yaml-xlang>
- Beam YAML blog:
 - <https://beam.apache.org/blog/beam-yaml-release/>
- Beam YAML examples catalog (including use-case from slides)
 - https://github.com/apache/beam/tree/master/sdks/python/apache_beam/yaml/examples
- Beam YAML Getting Started Notebook:
 - <https://colab.sandbox.google.com/github/apache/beam/blob/master/examples/notebooks/get-started/try-apache-beam-yaml.ipynb>

05

JobBuilder Introduction and Relationship to Beam YAML

Chamikara Jayalath



BEAM
SUMMIT

What is the Dataflow Job Builder?

- A visual UI for building and running Dataflow pipelines
- Backed by Beam YAML
- Allows saving your pipeline in GCS and loading them back.
- Includes a set of pre-defined pipelines (blueprints)

What is the Dataflow Job Builder?

Google Cloud Apache Beam Testing Search (/) for resources, docs, products, and more Q Search

Dataflow / Jobs

Jobs CREATE JOB FROM TEMPLATE CREATE JOB FROM BUILDER ENABLE SORTING REFRESH MANAGE LEARN

Running Archived Filter Filter jobs

Name	Type	End time	Elapsed time	Start time	Status	SDK version	ID	Region	Insights
combinetest0combinewithcontexttests0testsimplecombinewithcontextempty-runner-0627233443-b22f6dde	Streaming		13 sec	Jun 27, 2025, 4:34:51 PM	Starting...	2.67.0-SNAPSHOT	2025-06-27_16_34_51-14226377364057100805	us-central1	
pardotest0onwindowexpirationtests0testonwindowexpiationsimpleunbounded-runner-0627233420-bc563c51	Streaming		33 sec	Jun 27, 2025, 4:34:31 PM	Running	2.67.0-SNAPSHOT	2025-06-27_16_34_30-5071771841407414995	us-central1	
combinetest0windowingtests0testwindowedcombinegloballysingletonview-runner-0627233411-9c118209	Streaming		41 sec	Jun 27, 2025, 4:34:23 PM	Running	2.67.0-SNAPSHOT	2025-06-27_16_34_23-2303068588707070996	us-central1	
splittabledofntest0testwindowedsideinputwithcheckpointunbounded-runner-0627233401-4512c8b0	Streaming		55 sec	Jun 27, 2025, 4:34:09 PM	Running	2.67.0-SNAPSHOT	2025-06-27_16_34_08-8898928489406354523	us-central1	1 INSIGHT
reshuffletest0testreshuffleafterslidingwindowsandgroupbykey-runner-0627233400-98a98fd	Batch		55 sec	Jun 27, 2025, 4:34:09 PM	Running	2.67.0-SNAPSHOT	2025-06-27_16_34_08-14886492459681534760	us-central1	
viewtest0testmapsideinput-runner-0627233317-4f93da33	Batch		1 min 36 sec	Jun 27, 2025, 4:33:28 PM	Running	2.67.0-SNAPSHOT	2025-06-27_16_33_28-9578362189294579453	us-central1	
viewtest0testemptyiterablesideinput-runner-0627233319-a8ef99a	Streaming		1 min 36 sec	Jun 27, 2025, 4:33:28 PM	Running	2.67.0-SNAPSHOT	2025-06-27_16_33_27-6284998816097220507	us-central1	1 INSIGHT
splittabledofntest0testwindowedsideinputwithcheckpointunbounded-runner-0627233313-2745e902	Streaming		1 min 40 sec	Jun 27, 2025, 4:33:24 PM	Running	2.67.0-SNAPSHOT	2025-06-27_16_33_24-5136480102726828745	us-central1	1 INSIGHT
viewtest0testnonsingletonsideinput-runner-0627233315-f185535	Batch		1 min 40 sec	Jun 27, 2025, 4:33:24 PM	Running	2.67.0-SNAPSHOT	2025-06-27_16_33_24-2194828743137251619	us-central1	
beamapp-runner-0627233321-887568-zxz6xnth	Batch		1 min 41 sec	Jun 27, 2025, 4:33:23 PM	Running	2.67.0.dev	2025-06-27_16_33_22-12360376753569932992	us-central1	1 INSIGHT
beamapp-runner-0627233321-178047-zxz6xnth	Batch		1 min 42 sec	Jun 27, 2025, 4:33:22 PM	Running	2.67.0.dev	2025-06-27_16_33_22-9898663725212307475	us-central1	1 INSIGHT
beamapp-runner-0627233321-099793-ge34as1h	Batch		1 min 42 sec	Jun 27, 2025, 4:33:22 PM	Running	2.67.0.dev	2025-06-27_16_33_22-3886439867657155529	us-central1	1 INSIGHT
pcollectionrowuptest0testcomposecollectionrowtuple-runner-0627233312-a3cd8dbb	Batch		1 min 43 sec	Jun 27, 2025, 4:33:21 PM	Running	2.67.0-SNAPSHOT	2025-06-27_16_33_20-15392322707436043662	us-central1	
beamapp-runner-0627233319-406269-zxz6xnth	Batch		1 min 43 sec	Jun 27, 2025, 4:33:21 PM	Running	2.67.0.dev	2025-06-27_16_33_20-7411903724664174020	us-central1	1 INSIGHT

Show debug panel

What is the Dataflow Job Builder?

Create custom batch and streaming jobs using the Dataflow job builder and YAML editor. You can transform, filter, and combine data using Python and SQL. Use the builder form to compose your job steps or edit the YAML specification directly. [Learn more](#)

BUILDER FORM

YAML EDITOR

LOAD BLUEPRINTS

Job name *

Job names can contain lowercase letters, numbers, and dashes

Job type

☒ Batch

☐ Streaming

Sources

Read data from BigQuery, Pub/Sub, or Cloud Storage

New source

Source name *

Source

Source type *

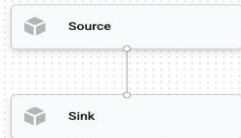
DONE

ADD A SOURCE

Transforms

Optionally manipulate, aggregate, and join data from sources and transforms

ADD A TRANSFORM



Why use Dataflow Job Builder ?

- No need to download or install a Beam SDK.
- Visual editing in the UI can be much simpler than building the pipeline programmatically.
- Transforms can be configured using the UI.
- Enough customization to work for many pipelines.
- Jobs can be validated and launched via the UI.

^ New source



Source name *

Source

Source type *

Pub/Sub Topic

Pub/Sub Subscription

BigQuery Table

CSV from Cloud Storage

JSON from Cloud Storage

Text files from Cloud Storage

YAML Source

JDBC

Trans

Options

[ADD A TRANSFORM](#)

^ New source



Source name *

Source

Source type *

CSV from Cloud Storage



 gs:// CSV location *

BROWSE

The location of your CSV file(s) in Cloud Storage. You can use wildcards to specify multiple files. [Learn more](#) 

PREVIEW SOURCE DATA

CSV delimiter character

Comma



Character used to parse CSV fields

DONE

^ New source



Source name *

Source

Source type *

BigQuery Table



BigQuery table *

☒ bigquery-public-data.samples.shakespeare

BROWSE

PREVIEW SOURCE DATA

Table fields *

☐ Read all table fields

☒ Specify table fields

Table fields *

1 of 4 selected

- ☒ word
Type: STRING
- ☐ word_count
Type: INTEGER
- ☐ corpus
Type: STRING
- ☐ corpus_date
Type: INTEGER

CANCEL

OK

ADD A TRANSFORM

[←](#) Create job from template[LOAD YAML](#)[SAVE YAML](#)[SEND FEEDBACK](#)

Dataflow templates

Launch jobs from Google-provided or custom templates



Job builder

Create custom jobs with the builder form and YAML editor

Source type *

[DONE](#)[ADD A SOURCE](#)

Transforms

Optionally manipulate, aggregate, and join data from sources and transforms

New transform

Transform name *

Transform

Transform type *

Filter (Python)

Filter records with a Python expression

SQL Transform

Manipulate records or join multiple inputs with a SQL statement

Map Fields (Python)

Add new fields or re-map entire records with Python expressions and functions

Map Fields (SQL)

Add or map record fields with SQL expressions

A

YAML transform

Use any transform from the Beam YAML SDK

Sinks

Write output to BigQuery, Pub/Sub, or Cloud Storage

New sink

Sink name *

Sink



Source



Transform



Sink

[Show debug panel](#)

^ New sink

Sink name *

Sink

Sink type *

Pub/Sub Topic

BigQuery Table

CSV files on Cloud Storage

JSON files on Cloud Storage

Text files on Cloud Storage

YAML Sink

Log Sink

06

Job Builder Ease-of-Use Features

Chamikara Jayalath



BEAM
SUMMIT

Blueprints

- Includes widely used pipelines that are already built for you
- Just load and run
- You can customize and save in GCS if needed

Blueprints

The screenshot shows the Google Cloud Dataflow console interface for creating a job from a template. The top navigation bar includes the Google Cloud logo, 'Apache Beam Testing', a search bar, and user profile icons. The left sidebar shows the 'Create job from template' path, with 'Job builder' selected. The main content area is titled 'Create job from template' and includes a 'LOAD BLUEPRINTS' dropdown menu that is currently open, displaying a list of data sources and transforms. The 'SEND FEEDBACK' button is highlighted with a red box. The interface also shows fields for 'Job name', 'Job type' (Batch/Streaming), 'Sources' (New source), and 'Transforms' (Add a transform). A 'Show debug panel' button is visible on the right side.

Google Cloud Apache Beam Testing

Search (/) for resources, docs, products, and more Search

Dataflow / Create job

Create job from template

LOAD YAML SAVE YAML SEND FEEDBACK

Dataflow templates
Launch jobs from Google-provided or custom templates

Job builder
Create custom jobs with the builder form and YAML editor

Create custom batch and streaming jobs using the Dataflow job builder and YAML editor. You can transform, filter, and combine data using Python and SQL. Use the builder form to compose your job steps or edit the YAML specification directly. [Learn more](#)

BUILDER FORM YAML EDITOR LOAD BLUEPRINTS

Job name *

Job names can contain lowercase letters, numbers, and hyphens.

Job type

☒ Batch

☐ Streaming

Sources

Read data from BigQuery, Pub/Sub, or Cloud Storage

New source

Source name *

Source

Source type *

ADD A SOURCE

Text files on Cloud Storage to BigQuery

CSV files on Cloud Storage to BigQuery

Pub/Sub topic to BigQuery

Pub/Sub subscription to BigQuery

Pub/Sub subscription to Pub/Sub topic

SQL Server to BigQuery

PostgreSQL to BigQuery

MySQL to BigQuery

Oracle to BigQuery

Word Count

Patterns

Stream Pub/Sub messages to BigQuery

Filter data with Python

Map data with Python dependencies

Join sources with SQL

Divert data with error handling

Log data to worker logs

DONE

Source

Sink

Show debug panel

Transforms

Optionally manipulate, aggregate, and join data from sources and transforms

ADD A TRANSFORM

Validate pipelines

The screenshot displays the Google Cloud Dataflow console interface for creating a job from a template. The 'Job builder' tab is selected, showing a configuration form for a pipeline. The 'Input step for the sink' is set to 'Format output'. The 'VALIDATE' button is highlighted with a red box, indicating the current action. The pipeline diagram on the right shows three steps: 'Count words' (Group by), 'Format output' (Map Fields (Python)), and 'Write to GCS' (Text files on Cloud Storage). The terminal at the bottom shows the command 'gcr.io/dataflow-templates/job-builder-server:latest' and the output 'Pipeline validation succeeded!'.

```
Using default tag: latest
latest: Pulling from dataflow-templates/job-builder-server
Digest: sha256:46d94357b90e297bda27576edc90a37a3da070717603d5f59a95d659b0e
Status: Image is up to date for gcr.io/dataflow-templates/job-builder-server:latest
gcr.io/dataflow-templates/job-builder-server:latest
Started Dataflow job builder validation server!
Building pipeline...
Running pipeline...
Pipeline validation succeeded!
```

Dynamic Job Graph

Google Cloud

Apache Beam Testing

Search (/) for resources, docs, products, and more

Search

◆ 📄 1 ? ⋮ 👤

Dataflow / Create job

← Create job from template

Dataflow templates

Launch jobs from Google-provided or custom templates

Job builder

Create custom jobs with the builder form and YAML editor

Sinks

Write output to BigQuery, Pub/Sub, or Cloud Storage

➤ New sink

Sink name *

Write to GCS

Sink type *

Text files on Cloud Storage

Text location *

☒ chamikara-beam-test/yaml_test2/output

BROWSE

The location to write text files in Cloud Storage. Ex: gs://your-bucket/your-file.txt.

Input step for the sink *

Format output

DONE

ADD A SINK

YAML Providers

Import additional transforms from YAML files to use them in your job. [Learn more](#)

+ ADD PROVIDER

Dataflow Options

RUN JOB

SCHEDULE RECURRING EXECUTION

VALIDATE

✓

LOAD YAML

SAVE YAML

SEND FEEDBACK

Read from GCS

Text files from Cloud Storage

Split words

Map Fields (Python)

Explode word arrays

Explode

Count words

Group by

Format output

Map Fields (Python)

Write to GCS

Text files on Cloud Storage

Show debug panel

Preview Source Data

Google Cloud Apache Beam Testing Search (/) for resources, docs, products, and more

Dataflow / Create job

Create job from template

Dataflow templates
Launch jobs from Google-provided or custom templates

Job builder
Create custom jobs with the builder form and YAML editor

Job type
☒ Batch
☐ Streaming

Sources
Read data from BigQuery, Pub/Sub, or Cloud Storage

New source

Source name *
Read from GCS

Source type *
Text files from Cloud Storage

Text location *
☒ gs://dataflow-samples/shakespeare/kinglear.txt BROWSE

The location of your text file(s) in Cloud Storage. You can use wildcards to specify multiple files. [Learn more](#)

PREVIEW SOURCE DATA

DONE

ADD A SOURCE

Transforms
Optionally manipulate, aggregate, and join data from sources and transforms

Split words
Map Fields (Python)

Explode word arrays
Explode

Source preview

KING LEAR

DRAMATIS PERSONAE

LEAR king of Britain (KING LEAR:)

KING OF FRANCE:

DUKE OF BURGUNDY (BURGUNDY:)

DUKE OF CORNWALL (CORNWALL:)

DUKE OF ALBANY (ALBANY:)

EARL OF KENT (KENT:)

EARL OF GLOUCESTER (GLOUCESTER:)

EDGAR son to Gloucester.

EDMUND bastard son to Gloucester.

CURAN a courtier.

Old Man tenant to Gloucester.

Doctor:

Fool:

OSWALD steward to Goneril.

A Captain employed by Edmund. (Captain:)

Gentleman attendant on Cordelia. (Gentleman:)

A Herald.

Servants to Cornwall.
(First Servant:)
(Second Servant:)
(Third Servant:)

OSWALD

CLOSE

07

Customizing Data Movement Jobs

Chamikara Jayalath



BEAM
SUMMIT

Traditional Dataflow Templates

- Great to move data from source A to sink B.
- Pipeline is fixed.
- Modifying the pipeline is high effort.
 - Requires defining a new template.
 - Requires defining a new container.
 - Requires configuring the new template.

Data Movement to ETL

- But ETL jobs usually require customizing the data (transform) before writing them back.
- Can be a custom transform and/or a custom config.
- So high effort to use traditional templates here.

Job Builder for ETL jobs

- Much easier to to customize existing blueprints.
- Much easier to define new pipelines.
- Much easier to define new transforms.
- Predefined transforms for mapping, filtering, combining, etc.

08

Data Processing Patterns (Filter, Map, SQL, Error Handling)

Chamikara Jayalath



BEAM
SUMMIT

^ New transform

Transform name *

Transform

Transform type *

Filter (Python)

Filter records with a Python expression

SQL Transform

Manipulate records or join multiple inputs with a SQL statement

Map Fields (Python)

Add new fields or re-map entire records with Python expressions and functions

Map Fields (SQL)

Add or map record fields with SQL expressions

YAML transform

Use any transform from the Beam YAML SDK

^ New transform

Transform name *

Transform

Transform type *

Filter (Python) ▼

Python filter expression *



Records that do not match the expression will be filtered out

☐ Handle errors

Route records that throw exceptions when processed to a separate step output. This output must be a transform or sink.

Input step for the transform *

Python filter expressions can filter records by performing checks against the record's fields. For example:

```
myField == 'value'  
myField > 0 and myField < 10  
myField != False
```

DONE

^ New transform



Transform name *

Transform

Transform type *

Map Fields (Python)



Preserve existing fields

Append new fields to the record. If unchecked, existing fields are dropped.

Mapped fields

ADD A FIELD

Dropped fields

+ ADD A FIELD

Python dependencies

Add PyPI packages as dependencies to make them available to import in callables.

+ ADD A DEPENDENCY



Handle errors

Route records that throw exceptions when processed to a separate step output. This output must be consumed by a downstream transform or sink.

Input step for the transform *

Source



DONE

^ New field



Field name *

year



Callable

If checked, the field value must be a Python function rather than a simple expression

Python callable ?

Python function that defines the field value

Press Option+F1 for Accessibility Options.

```
1 import roman
2 def convert(row):
3     return roman.toRoman(row.year)
```

DONE

ADD A FIELD

Python dependencies

Add PyPI packages as dependencies to make them available to import in callables.

Dependency 1 *

roman>=4.2

The requirement specifier with an optional version specifier

+ ADD A DEPENDENCY

^ Edit transform



Transform name *

Join

Transform type *

SQL Transform



Input steps for the transform *

Taxi ride data and Zone lookup data



Input aliases

If a single input is selected, it can be referenced in the query as PCOLLECTION. If multiple inputs are selected, aliases for each input will appear above the query editor.

Taxi ride data: input0

Zone lookup data: input1

Press Option+F1 for Accessibility Options.

```
1 SELECT * FROM input0
2 JOIN input1 ON input0.PULocationID = input1.LocationID
```

A [Beam Calcite SQL](#) query to be run over one or more inputs

DONE

[←](#) Create job from template[LOAD YAML](#)[SAVE YAML](#)[SEND FEEDBACK](#)**+ Dataflow templates**

Launch jobs from Google-provided or custom templates

Job builder

Create custom jobs with the builder form and YAML editor

^ New transform

Transform name *

Format output

Transform type *

Map Fields (Python)



Preserve existing fields

Append new fields to the record. If unchecked, existing fields are dropped.

Mapped fields

output

[ADD A FIELD](#)**Python dependencies**

Add PyPI packages as dependencies to make them available to import in callables.

[ADD A DEPENDENCY](#)

Handle errors

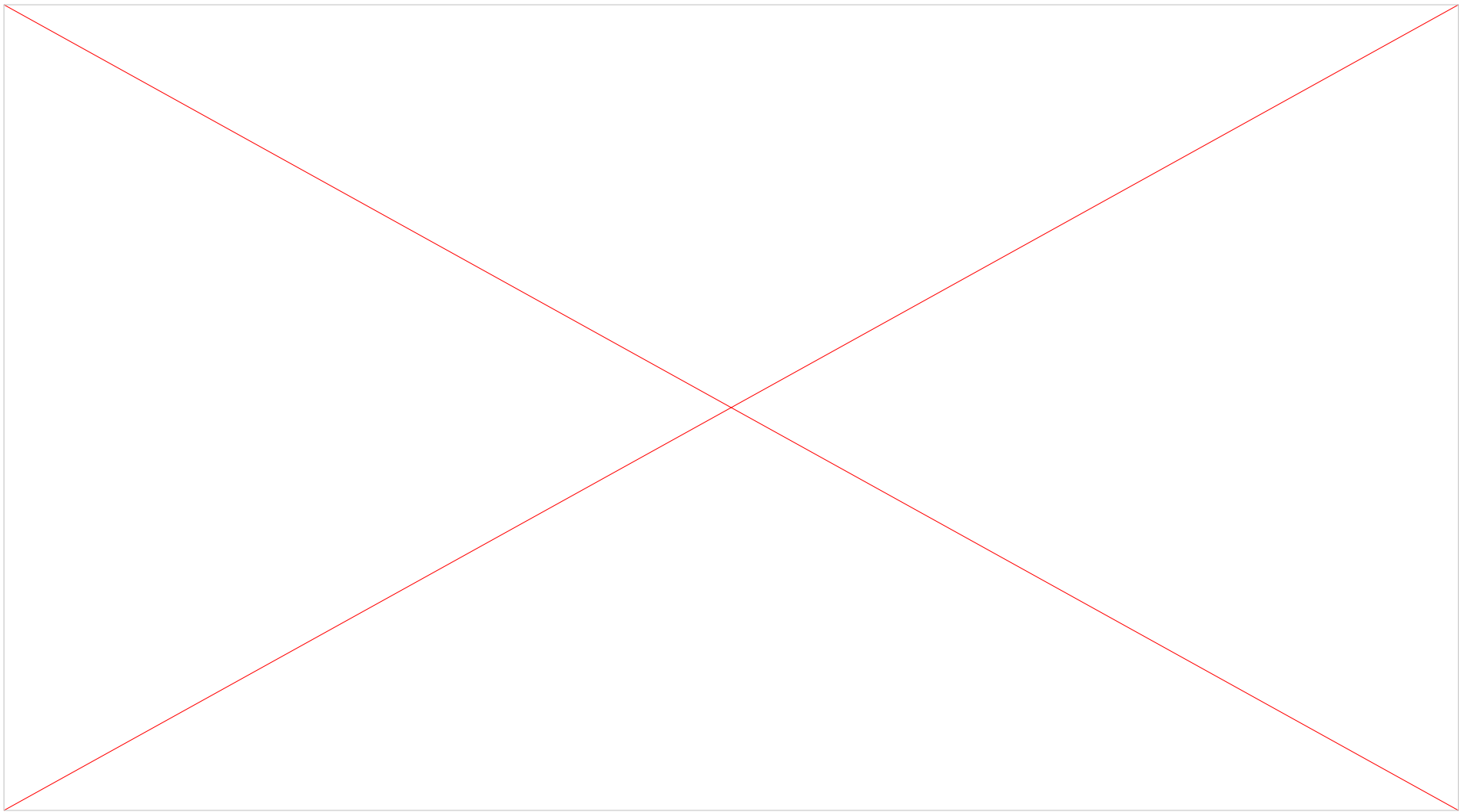
Route records that throw exceptions when processed to a separate step output. This output must be consumed by a downstream transform or sink.

Input step for the transform *

Count words

[DONE](#)[ADD A TRANSFORM](#)**Sinks****Read from GCS**
Text files from Cloud Storage**Split words**
Map Fields (Python)**Explode word arrays**
Explode**Count words**
Group by**Format output**
Map Fields (Python)**Format output.errors**
Error records**Write to GCS**
Text files**Write errors to GCS**

Show debug panel



09

Working with YAML in the Job Builder (Export, Import, Yaml Editor)

Chamikara Jayalath



BEAM
SUMMIT

Working with YAML

Save YAML and run with gcloud

Save YAML job definition

Saved YAML definitions can be run with gcloud or loaded back into the job builder.

Cloud Storage file path *

BROWSE

Path and file name for writing the YAML. For example, your-bucket/your-file.yaml

SAVE

DOWNLOAD

Run with gcloud

Once you save or download your job's YAML definition, you can run it with gcloud:

\$ gcloud dataflow yaml run word-count
--yaml-pipeline-file=gs://<path-to-yaml-file>

YAML job definition

```
pipeline:
  transforms:
    - type: ReadFromText
      name: Read from GCS
      config:
        path: gs://dataflow-samples/shakespeare/kinglear.txt
    - type: MapToFields
      name: Split words
      config:
        language: python
        fields:
          word:
            callable: |
              import re
              def my_mapping(row):
                return re.findall(r'[A-Za-z\']+', row.line.lower())
            value: "1"
input: Read from GCS
```

Working with YAML

The screenshot shows the Google Cloud Dataflow console. The top navigation bar includes the Google Cloud logo, 'Apache Beam Testing', and a search bar. The left sidebar shows the 'Dataflow / Create job' breadcrumb and a menu with 'Dataflow templates' and 'Job builder'. The main content area is titled 'Create job from template' and contains a description of the Dataflow job builder. Below this, there are three tabs: 'BUILDER FORM' (selected), 'YAML EDITOR', and 'LOAD BLUEPRINTS'. A red error message states: 'Make sure all fields are correct to continue'. The 'Job name' field contains 'word-count' with a note that job names can contain lowercase letters, numbers, and dashes. The 'Job type' section has 'Batch' selected. The 'Sources' section is expanded, showing a 'New source' form with fields for 'Source name' (Read from GCS), 'Source type' (Text files from Cloud Storage), and 'Text location' (gs://dataflow-samples/shakespeare/kinglear.txt). A 'BROWSE' button is next to the text location field. At the bottom right of the form is a 'DONE' button.

Load from YAML

YAML pipeline specification

Load a Beam YAML pipeline into the job builder.

Beam YAML is a declarative syntax for describing pipelines by using YAML files. You can use Beam YAML to author a Beam pipeline without writing any code. [Learn more](#)

YAML file location * BROWSE

The location of your YAML file in Cloud Storage. Ex: gs://your-bucket/your-file.yaml.

LOAD CANCEL

Working with YAML

Create custom batch and streaming jobs using the Dataflow job builder and YAML editor. You can transform, filter, and combine data using Python and SQL. Use the builder form to compose your job steps or edit the YAML specification directly. [Learn more](#)

BUILDER FORM

YAML EDITOR

LOAD BLUEPRINTS

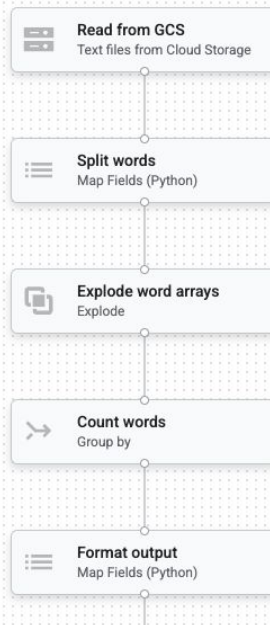
Job name *

word-count

Job names can contain lowercase letters, numbers, and dashes

Press Option+F1 for Accessibility Options.

```
1 pipeline:
2   transforms:
3     - type: ReadFromText
4       name: Read from GCS
5       config:
6         path: gs://dataflow-samples/shakespeare/kinglear.txt
7     - type: MapToFields
8       name: Split words
9       config:
10        language: python
11        fields:
12          word:
13            callable: |
14              import re
15              def my_mapping(row):
16                return re.findall(r'[A-Za-z\']+', row.line.lower())
17            value: "1"
18        input: Read from GCS
19     - type: Explode
20       name: Explode word arrays
21       config:
22         fields: [word]
23       input: Split words
24     - type: Combine
```



YAML Providers

YAML Providers

Import additional transforms from YAML files to use them in your job. [Learn more](#) 

YAML provider path *



BROWSE



The location of the YAML provider file in Cloud Storage

[+ ADD PROVIDER](#)

YAML Providers

YAML Providers

Import additional transforms from YAML files to use them in your job. [Learn more](#) 

Loaded transforms

Range  Power 

YAML provider path *



ryanmadden-corp-bucket/provider_listing.yaml

BROWSE

The location of the YAML provider file in Cloud Storage

 ADD PROVIDER

Dataflow Day Schedule (07/10)

		Speaker	Role	Topic
	Before 9:00			Onramp signup and getting settled
9:00	9:05	Wei Hsia	Developer Advocate	Agenda, objective, and kickoff
9:05	9:35	Mehran Nazir	Product Manager	Dataflow's current and future state
9:35	10:35	Riju Kallivalappil	Software Engineer	Dataflow: Life of a batch pipeline (and best practices)
10:35	10:50			Morning Break
10:50	11:50	Tom Stepp	Software Engineer	Dataflow: Life of a streaming pipeline (and best practices)
11:50	12:50	Danny McCormick	Software Engineer	Dataflow ML and Turnkey transforms
		Valentyn Tymofieiev	Software Engineer	
13:00	14:00			Lunch
14:00	14:30	Valeria Sanchez	Cloud Technical Solutions Engineer, Big Data	Dataflow and Google Cloud Customer Care
		Varnikaa Gupta	Technical Solutions Specialist, Google Cloud	
14:30	15:00	Ian Mburu	Solutions Consultant	Common I/Os and how to use them with Dataflow
15:00	16:00			Q&A

Venue: 75 9th Avenue, New York, NY 10011 (2nd floor of the Chelsea Market building)

Thank you!

Questions?

Please reach out with any questions!

Emails:

chamikara@google.com

derrickaw@google.com