Introduction to Beam YAML

Presented by:

Jeff Kinard

Software Engineer at Google
Working on Apache Beam and Dataflow

BEAM SUMMIT

September 4-5, 2024

Sunnyvale, CA. USA

Agenda

01
Introduction

How do we make Beam easier?

- Python, Golang, typescript, etc. SDK's to give users a choice of language
  - Still requires programming language knowledge and Beam model experience
- Beam SQL to convert data engineers familiar with SQL
  - Performance limitations (hotkeys, etc.)
  - Syntax limitations
  - Deprecated
- Dataflow Templates (Dataflow runner only)
  - 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 - An easier way to express pipelines

- A format many more users are familiar with
- Easier to author and deploy intermediate pipelines without the complexity of Beam (e.g. SDK/dependency install, set up dev environment, grok Beam programming model)
- Easily copy, modify, share existing YAML pipelines

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
- Allow for code translation for getting started with Beam

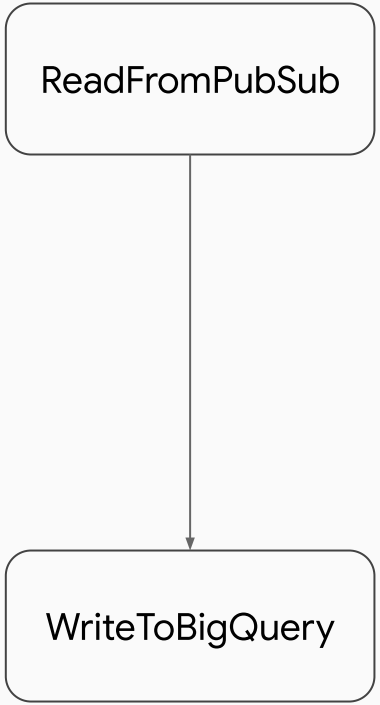
02

Basic Syntax

BEAM SUMMIT

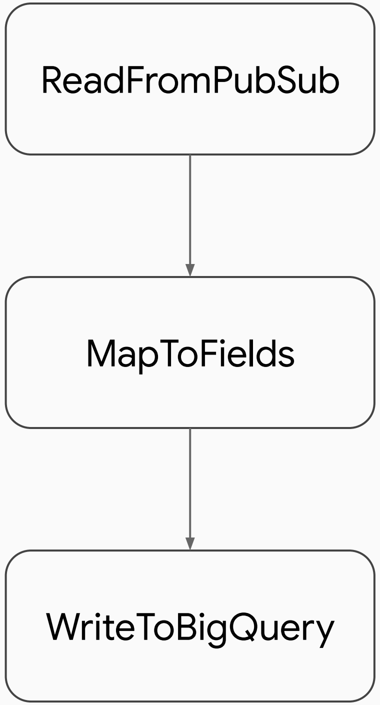
Basic Read-Write YAML Pipeline

```yaml
pipeline:
  type: chain

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







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



Add a Transformation

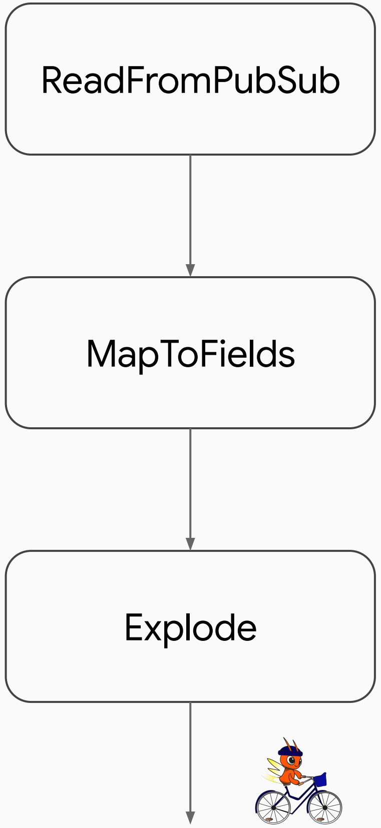
```
pipeline:
  type: chain

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

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

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



$$\begin{tikzcd}
\text{ReadFromPubSub} \arrow[d] \\
\text{MapToFields} \arrow[d] \\
\text{WriteToBigQuery}
\end{tikzcd}$$

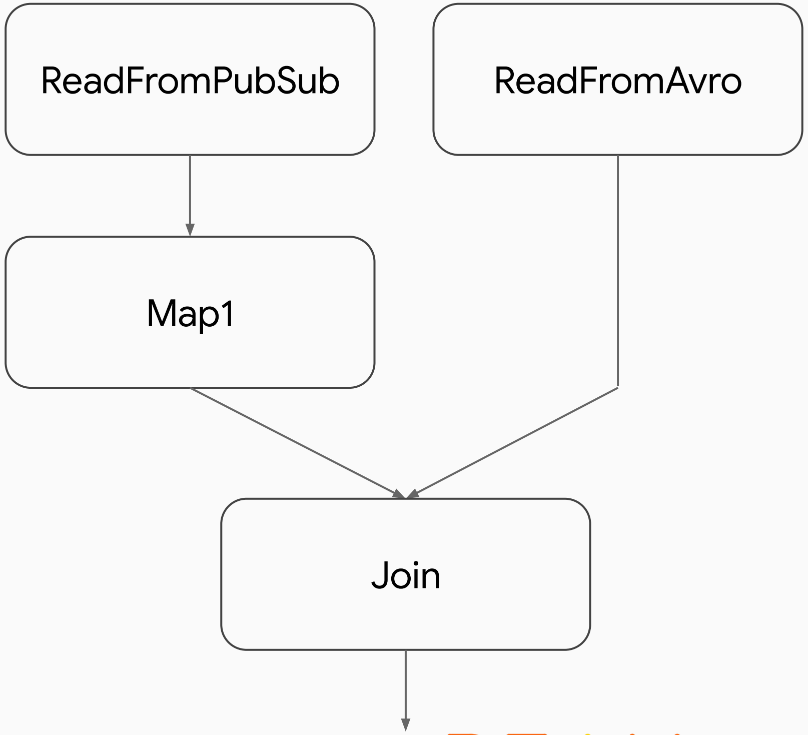
Many transforms can be chained

```
pipeline:
  type: chain

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

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

  - type: Explode
    config:
      fields: [pets]

  ...
```



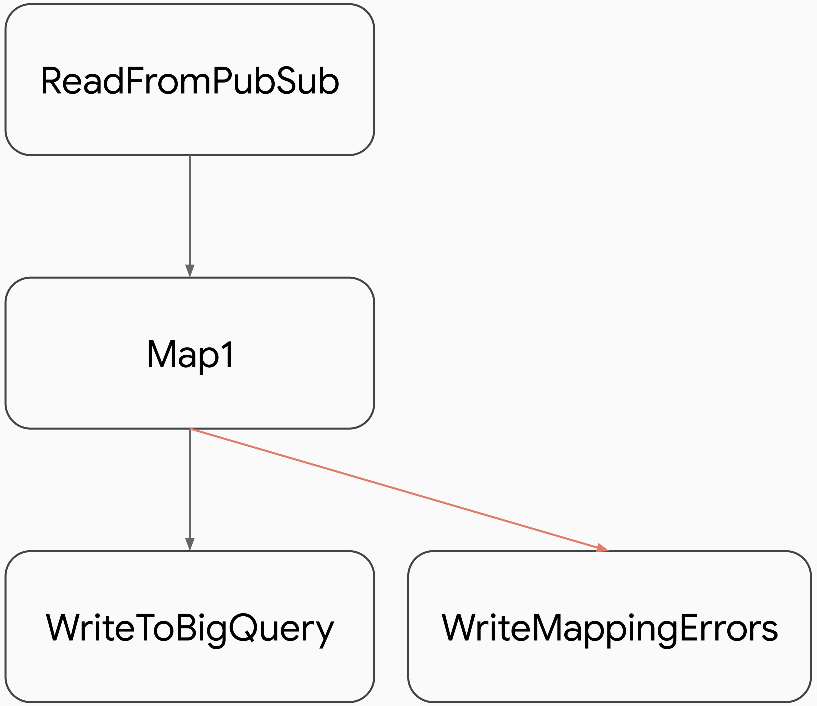
Pipelines need not be linear

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

  - type: MapToFields
    name: Map1
    input: ReadFromPubSub
    config: ...

  - type: ReadFromAvro
    config:
      path: "gs://..."

  - type: Join
    input:
      left: Map1
      right: ReadFromAvro
    config:
      ...

  ...
```



Sophisticated error handling

```yaml
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: Map1.errors
    config:
      path: "/path/to/errors.json"
```



03

Current Turnkey Transforms

Beam YAML supports a large number of IOs...

- 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/

…and other turn-key transforms

- Utility
  - Create
  - Flatten
  - WindowInto
  - LogForTesting
  - AssertEqual
- Mapping
  - MapToFields
  - Explode
  - Filter
  - Partition
- Aggregation
  - Combine
- ML
  - MLTransform (experimental)
    - Coming to template/gcloud in Beam 2.59
  - Enrichment (coming soon)
  - RunInference (coming soon)
- Other
  - Sql
  - Join
- . . .

Full list at https://beam.apache.org/releases/yamldoc/current/

Example: MapToFields

```yaml
- type: MapToFields
  name: RenameAndMapCustomFields
  input: ReadFromCsv
  config:
    language: python
    fields:
      myNewStr: "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”

Full docs at [https://beam.apache.org/documentation/sdks/yaml-udf/ /](https://beam.apache.org/documentation/sdks/yaml-udf/)

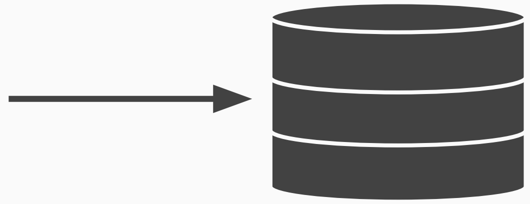
BEAM SUMMIT

04
Use-case

Use case: Department Store

- Department store records every transaction and stores in a MySQL database

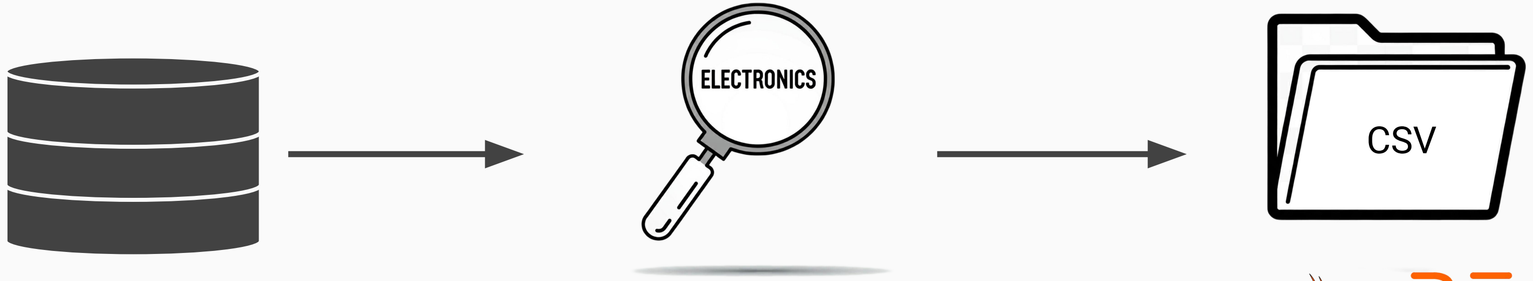
transaction_id	product_name	category	price
T0012	Headphones	Electronics	59.99
T5034	Leather Jacket	Apparel	109.99
T0024	Aluminum Mug	Kitchen	29.99
T0104	Headphones	Electronics	59.99
…	…	…	…
T0302	Monitor	Electronics	249.99



BEAM SUMMIT

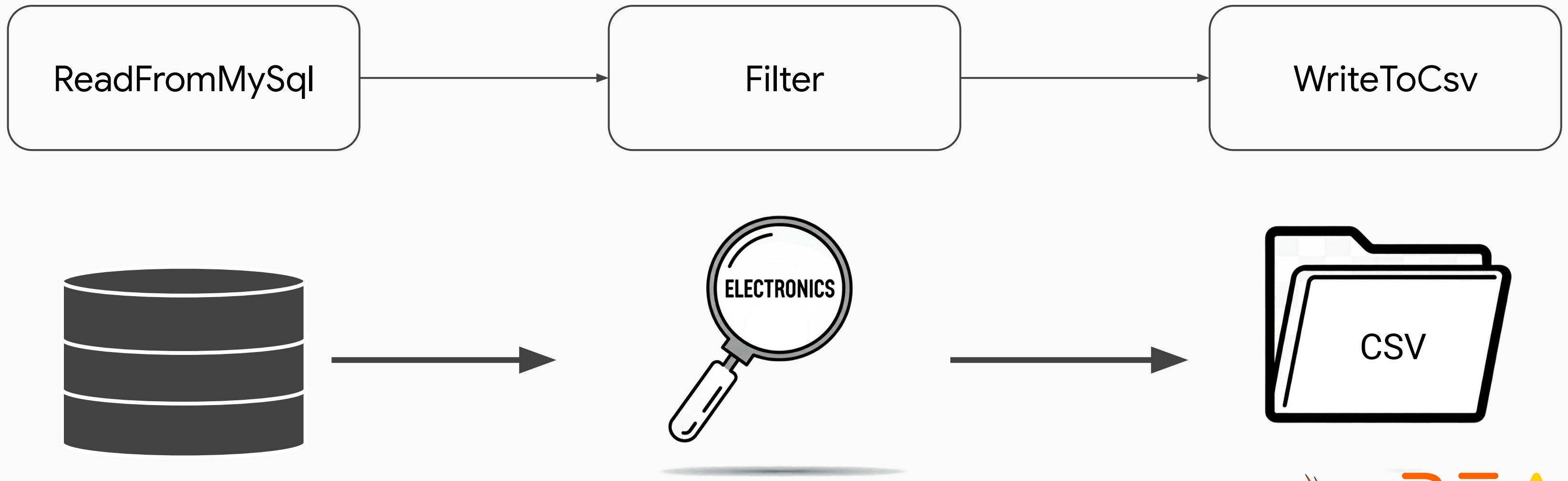
Use case: Department Store

- It is the end of the fiscal year, and the Electronics department needs to gather a report of transactions for auditing purposes



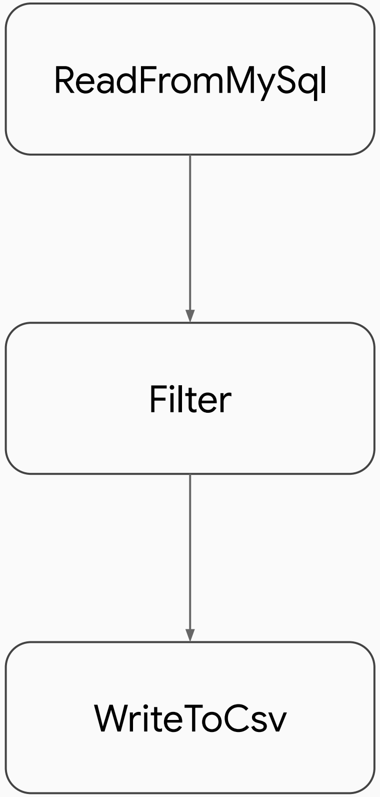
Use case: Department Store

- How would this look in Beam YAML?



Simple filter pipeline

```
pipeline:
  type: chain

  source:
    type: ReadFromMySql
    config:
      url: jdbc:mysql://host:port/database
      table: transactions
      username: 'username'
      password: 'password'

  transforms:
  - type: Filter
    config:
      language: python
      keep: category == "Electronics"

  sink:
    type: WriteToCsv
    config:
      path: electronics.csv
```



Use case: Department Store

- Results of the pipeline...

transaction_id	product_name	category	price
T0012	Headphones	Electronics	59.99
T5034	Leather Jacket	Apparel	109.99
T0024	Aluminum Mug	Kitchen	29.99
T0104	Headphones	Electronics	59.99
…	…	…	…
T0302	Monitor	Electronics	249.99

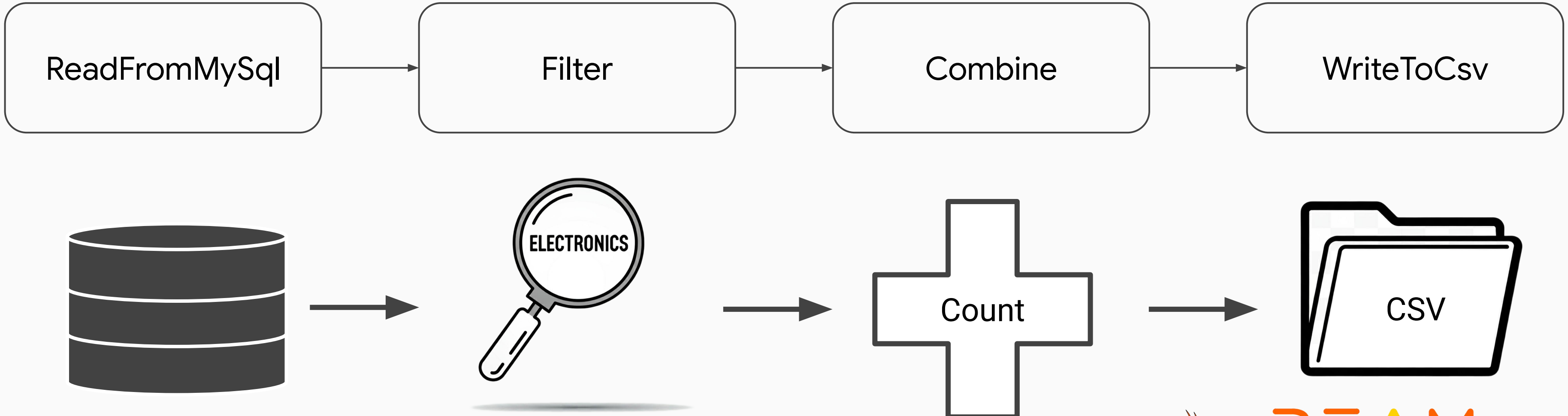
$\longrightarrow$

transaction_id	product_name	category	price
T0012	Headphones	Electronics	59.99
T0104	Headphones	Electronics	59.99
…	…	…	…
T0302	Monitor	Electronics	249.99

BEAM SUMMIT

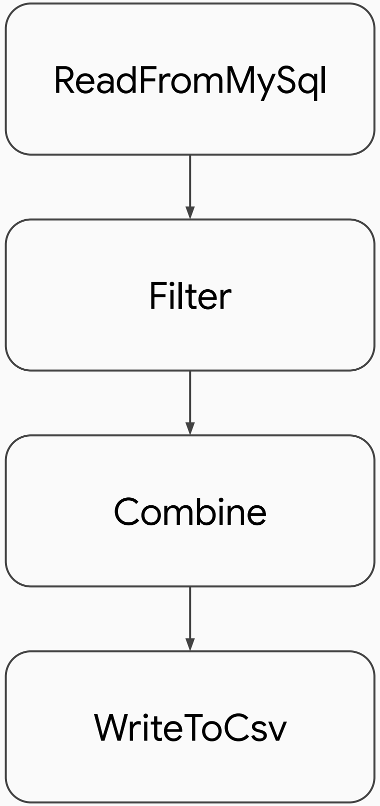
Use case: Department Store

- Fast forward… It is now the beginning of the next fiscal year, and the Electronics department needs to order more inventory to meet expected demand



Simple aggregation pipeline

```yaml
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
      input: FilterWithCategory
      config:
        group_by: product_name
        combine:
          num_sold:
            value: product_name
            fn: count
    - type: WriteToCsv
      config:
        path: electronics.csv
```



Use case: Department Store

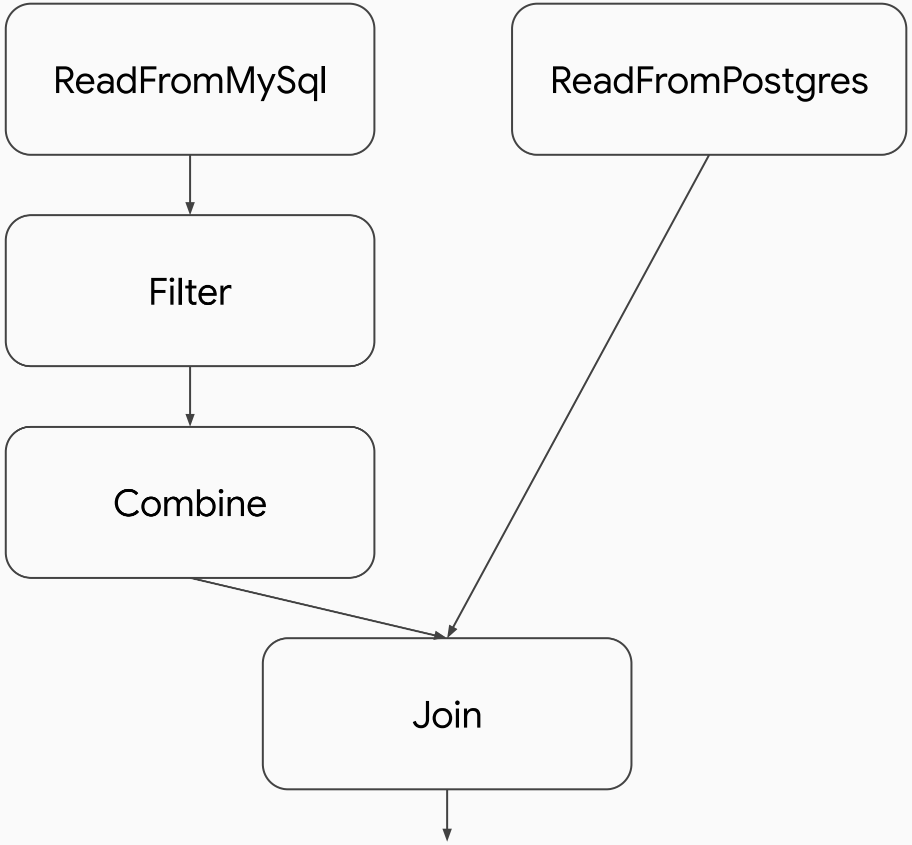
- Results of the pipeline…

transaction_id	product_name	category	price
T0012	Headphones	Electronics	59.99
T0104	Headphones	Electronics	59.99
…	…	…	…
T0302	Monitor	Electronics	249.99

→

transaction_id	product_name	category	price	num_sold
T0012	Headphones	Electronics	59.99	2
…	…	…	…	
T0302	Monitor	Electronics	249.99	1

Use case: Department Store



05

Running a Pipeline

Running Beam YAML

- On Dataflow

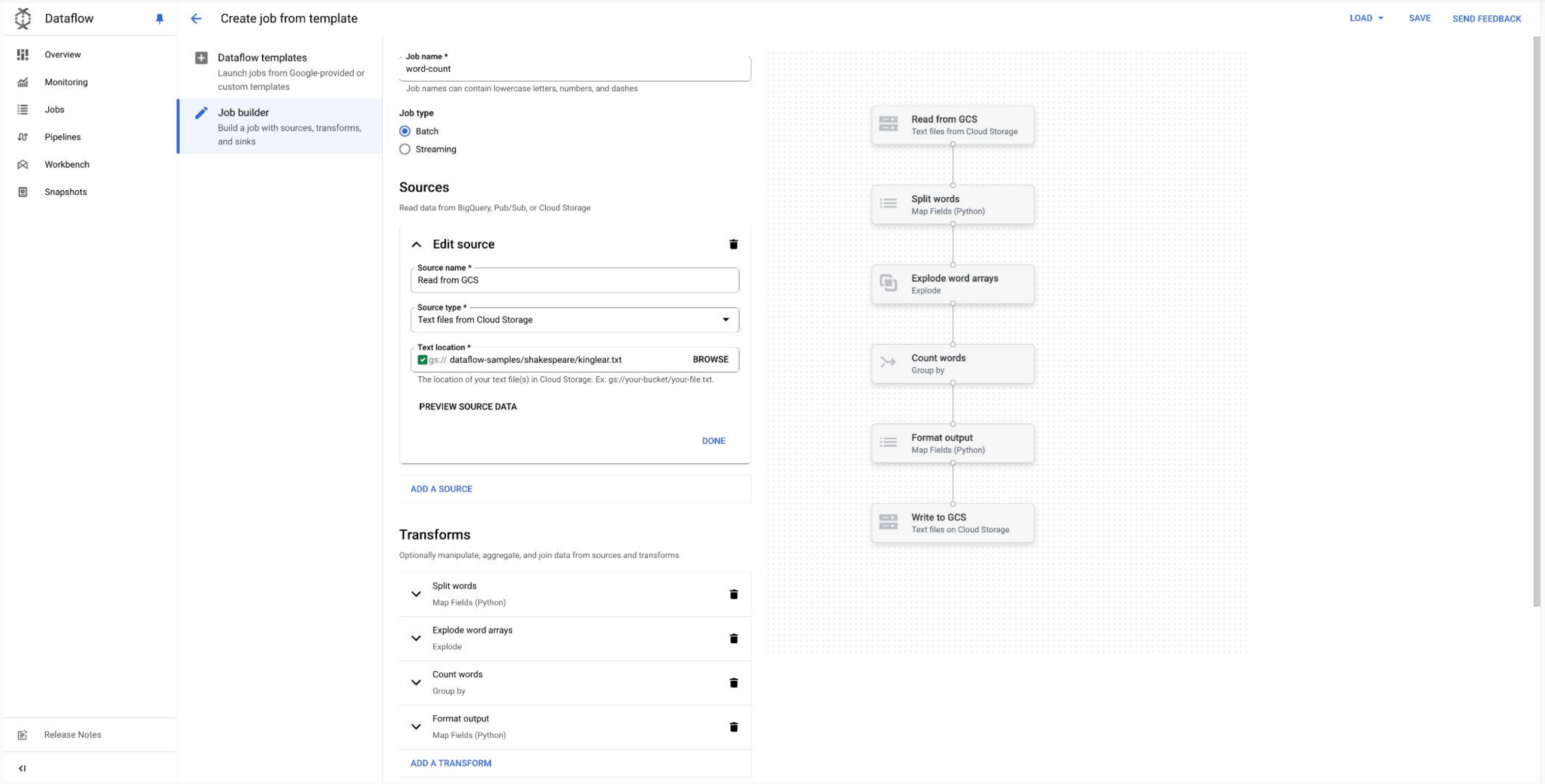
```
$ gcloud dataflow yaml run /path/to/my.yaml
```

- 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 Job Builder



Dataflow

Overview
Monitoring
Jobs
Pipelines
Workbench
Snapshots

Release Notes

Create job from template

LOAD SAVE SEND FEEDBACK

Dataflow templates
Launch jobs from Google-provided or custom templates

Job builder
Build a job with sources, transforms, and sinks

Job name *
word-count
Job names can contain lowercase letters, numbers, and dashes

Job type
[✓ Batch]
Streaming

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

Edit 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. Ex: gs://your-bucket/your-file.txt.

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

Count words
Group by

Format output
Map Fields (Python)

ADD A TRANSFORM

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

BEAM SUMMIT

More Information

- Beam YAML docs:
    - https://beam.apache.org/documentation/sdks/yaml/
- Beam YAML Getting Started Notebook:
    - https://colab.sandbox.google.com/github/apache/beam/blob/master/examples/notebooks/get-started/try-apache-beam-yaml.ipynb
- 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

Thank you!

Questions?

Please reach out with any questions!

Email:
jkinard@google.com

LinkedIn:
https://www.linkedin.com/in/jeffrey-kinard-92637214a/

BEAM SUMMIT