



Dataflow Templates

Akvelon

<https://akvelon.com/>





Agenda

Dataflow Templates

Use Case - Data Protection Using Tokenization

Solution Architecture & Technical Highlights

Demo

Contributing to open source [DataflowTemplates](#)



AKVELON
We make BIG software happen



Google Cloud
Partner



Beam Pipeline





Dataflow Templates

Template - a way to package and stage pipelines

Template types: Classic and Flex

Flex template benefits

- Dynamic DAG
- Eliminated need for ValueProvider
- Expanded templates flexibility

A screenshot of the Google Cloud Platform Dataflow interface. The top navigation bar shows "Google Cloud Platform" and "DataTokenization". On the left, there's a sidebar with "Dataflow" selected, showing "Jobs", "Snapshots", and "Notebooks". The main area is titled "Create job from template". It has fields for "Job name" (set to "Demo"), "Regional endpoint" (set to "us-central1"), and "Dataflow template" (set to "Custom Template"). Below these, there's a section for "Execute a custom template that you've uploaded to Cloud Storage" with a checked checkbox for "Template path" pointing to "gs://tokenization_test/templates/datatokenization_test.json". There's also a "BROWSE" button. Further down, there's a "Required parameters" section with fields for "GCS location of data schema", "GCS file pattern for input files", "File format of input files" (set to "JSON, CSV or Avro input file format"), and "CSV file(s) contain headers" (set to "true").

Google Cloud Platform DataTokenization

Dataflow

Job name * Demo

Must be unique among running jobs

Regional endpoint * us-central1

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 * Custom Template

Execute a custom template that you've uploaded to Cloud Storage

Template path * gs://tokenization_test/templates/datatokenization_test.json [BROWSE](#)

Path to your template file stored in Cloud Storage

Tokenizes structured plain text data from GCS or Pub/Sub input source using an external API calls to Protegoity DSG and ingests date to GCS, Bigtable or BigQuery sink

Required parameters

GCS location of data schema *

Path to data schema file located on GCS. BigQuery compatible JSON format data schema required

GCS file pattern for input files

GCS file pattern for files in the source bucket

File format of input files

JSON, CSV or Avro input file format

CSV file(s) contain headers: true or false

'true' if CSV file(s) in the input bucket contain headers, and 'false' otherwise

Release Notes



Create Template

1. Implement pipeline
2. Create metadata
3. Build template

Job name *

Must be unique among running jobs

Regional endpoint *

us-central1

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 *

Word Count

Batch pipeline. Reads text from Cloud Storage, tokenizes text lines into individual words, and performs frequency count on each of the words. [OPEN TUTORIAL](#)

Required parameters

Input file(s) in Cloud Storage *

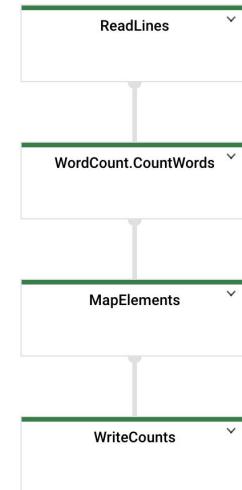
The input file pattern Dataflow reads from. Use the example file (gs://dataflow-samples/shakespeare/kinglear.txt) or enter the path to your own using the same format: gs://your-bucket/your-file.txt

Output Cloud Storage file prefix *

Path and filename prefix for writing output files. Ex: gs://your-bucket/counts

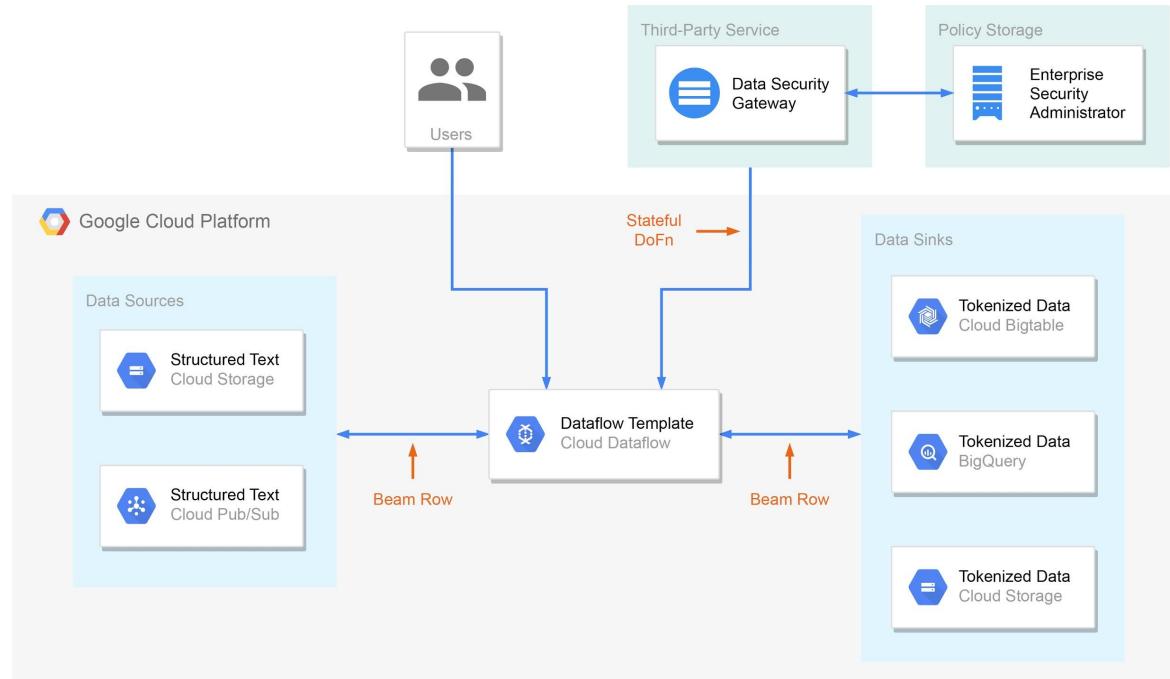
Temporary location *

Path and filename prefix for writing temporary files. Ex: gs://your-bucket/temp



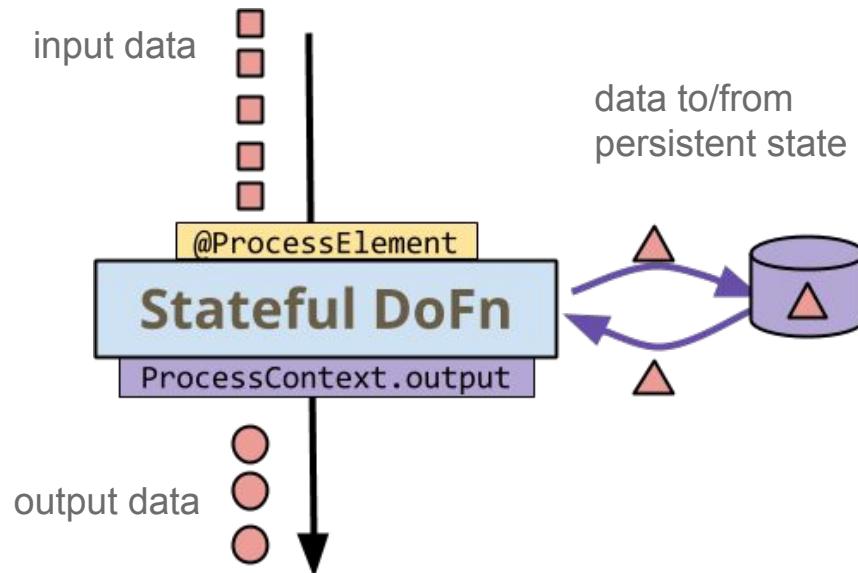


Data Protection Using Tokenization





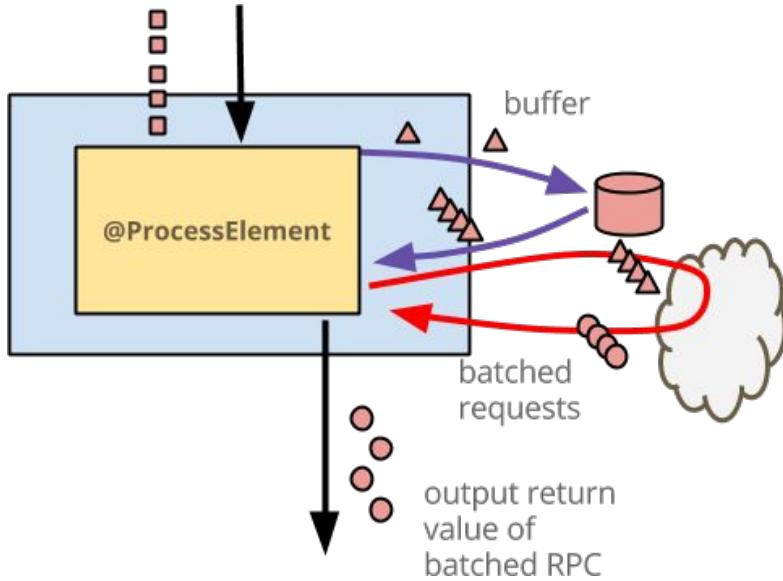
Stateful Processing



More info <https://beam.apache.org/blog/stateful-processing/>



Stateful DoFn

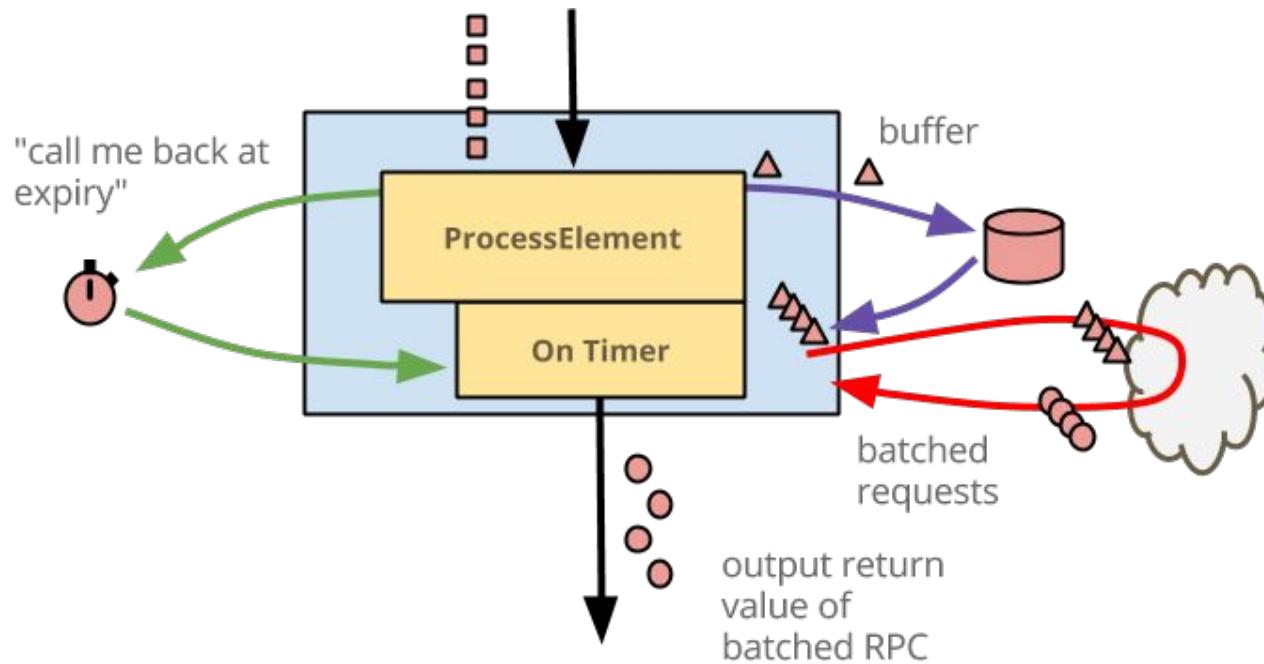


More info <https://beam.apache.org/blog/timely-processing/>

```
new DoFn<Event, EnrichedEvent>() {  
  
    private static final int MAX_BUFFER_SIZE = 500;  
  
    @StateId("buffer")  
    private final StateSpec<BagState<Event>> bufferedEvents = StateSpecs.bag();  
  
    @StateId("count")  
    private final StateSpec<ValueState<Integer>> countState = StateSpecs.value();  
  
    @ProcessElement  
    public void process(  
        ProcessContext context,  
        @StateId("buffer") BagState<Event> bufferState,  
        @StateId("count") ValueState<Integer> countState) {  
  
        int count = firstNonNull(countState.read(), 0);  
        count = count + 1;  
        countState.write(count);  
        bufferState.add(context.element());  
  
        if (count >= MAX_BUFFER_SIZE) {  
            for (EnrichedEvent enrichedEvent : enrichEvents(bufferState.read())) {  
                context.output(enrichedEvent);  
            }  
            bufferState.clear();  
            countState.clear();  
        }  
    }  
    //.... TBD ...  
}
```



Timely Stateful Processing



More info <https://beam.apache.org/blog/timely-processing/>



Stateful DoFn with Timer - Implementation

```
new DoFn<Event, EnrichedEvent>() {
    //...

    @TimerId("expiry")
    private final TimerSpec expirySpec = TimerSpecs.timer(TimeDomain.EVENT_TIME);

    @ProcessElement
    public void process(
        ProcessContext context,
        BoundedWindow window,
        @StateId("buffer") BagState<Event> bufferState,
        @StateId("count") ValueState<Integer> countState,
        @TimerId("expiry") Timer expiryTimer) {

        expiryTimer.set(window.maxTimestamp().plus(allowedLateness));

        //... same logic as above ...
    }

    @OnTimer("expiry")
    public void onExpiry(
        OnTimerContext context,
        @StateId("buffer") BagState<Event> bufferState) {
        if (!bufferState.isEmpty().read()) {
            for (EnrichedEvent enrichedEvent : enrichEvents(bufferState.read())) {
                context.output(enrichedEvent);
            }
            bufferState.clear();
        }
    }
};
```



GroupIntoBatches

- Groups your data into batches
- Implemented using Stateful DoFn
- Has several optimizations
 - Prefetches data
 - Autoshrarding in Dataflow

```
import apache_beam as beam

with beam.Pipeline() as pipeline:
    batches_with_keys = (
        pipeline
        | 'Create produce' >> beam.Create([
            ('spring', '🍓'),
            ('spring', '🥕'),
            ('spring', '🍇'),
            ('spring', '🍅'),
            ('summer', '🥕'),
            ('summer', '🍅'),
            ('summer', '🫐'),
            ('fall', '🥕'),
            ('fall', '🍅'),
            ('winter', '🍇'),
        ])
        | 'Group into batches' >> beam.GroupIntoBatches(3)
        | beam.Map(print))
```

Output:

```
('spring', ['🍓', '🥕', '🍇'])
('summer', ['🥕', '🍅', '🫐'])
('spring', ['🍅'])
('fall', ['🥕', '🍅'])
('winter', ['🍇'])
```

More info

<https://beam.apache.org/documentation/transforms/python/aggregation/groupintobatches>



Processing with GroupIntoBatches

```
public PCollectionTuple expand(PCollection<KV<Integer, Row>> inputRows) {
    FailsafeElementCoder<Row, Row> coder =
        FailsafeElementCoder.of(RowCoder.of(schema()), RowCoder.of(schema()));

    Duration maxBuffering = Duration.millis(MAX_BUFFERING);
    PCollectionTuple pCollectionTuple =
        inputRows
            .apply(
                name: "GroupRowsIntoBatches",
                GroupIntoBatches.<Integer, Row>.ofSize(batchSize())
                    .withMaxBufferingDuration(maxBuffering))
            .apply(
                name: "Tokenize",
                ParDo.of(new TokenizationFn(schema(), rpcURI(), failureTag()))
                    .withOutputTags(successTag(), TupleTagList.of(failureTag())));
}

return PCollectionTuple.of(
    successTag(), pCollectionTuple.get(successTag()).setRowSchema(schema()))
    .and(failureTag(), pCollectionTuple.get(failureTag()).setCoder(coder));
}
```



Use Case Recommendations

Stateful DoFn

Customization of stateful processing

- Deduplication
- Arbitrary-but-consistent indexing

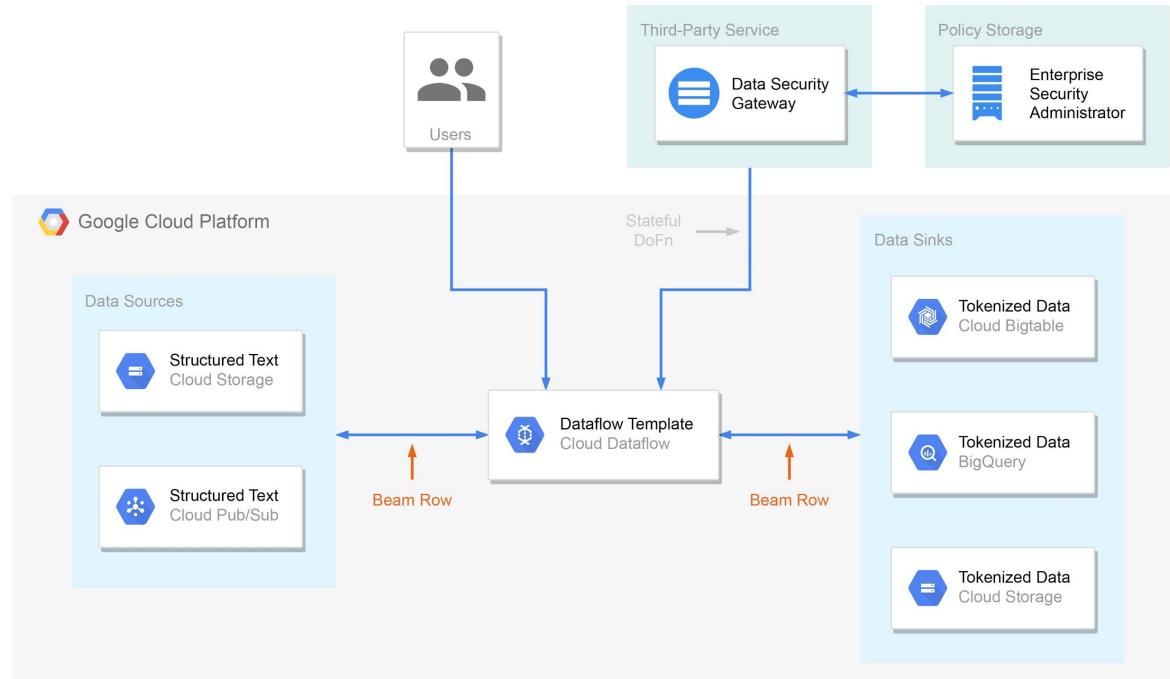
GroupIntoBatches

Optimized out-of-the-box transform

- External API call
- Data preparation for ML model



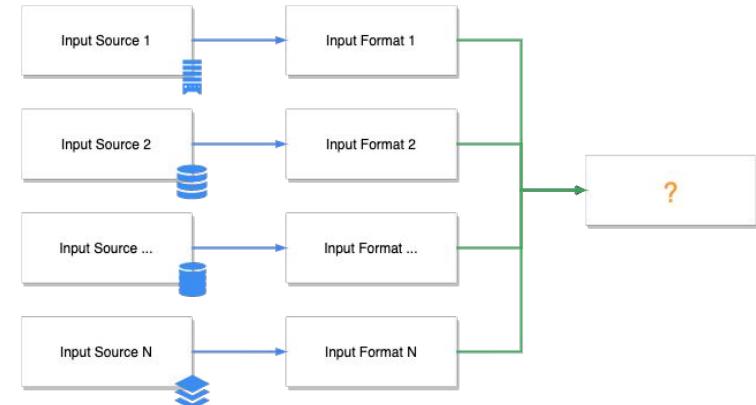
Data Protection Using Tokenization





How to Represent Data in Pipeline Code?

- Common abstractions for data representation
- Avoid writing transformation again and again
- Common interface to all IO transforms
- Easy and effective to serialize





Common Approaches

Text based formats

- JSON
- CSV
- XML
- YAML

Binary based formats

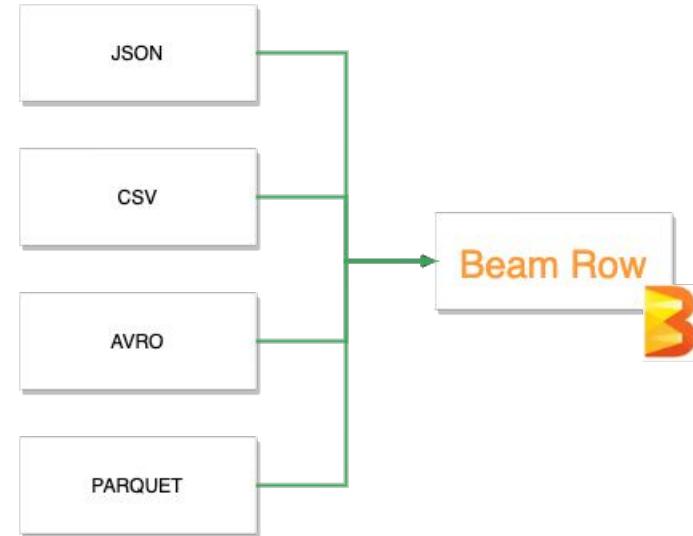
- Apache Avro
- Parquet
- Protobuf
- MessagePack

**BSON JSON
YAML ORC PROTOBUF
AVRO PARQUET XML
CSV THRIFT
MESSAGEPACK**



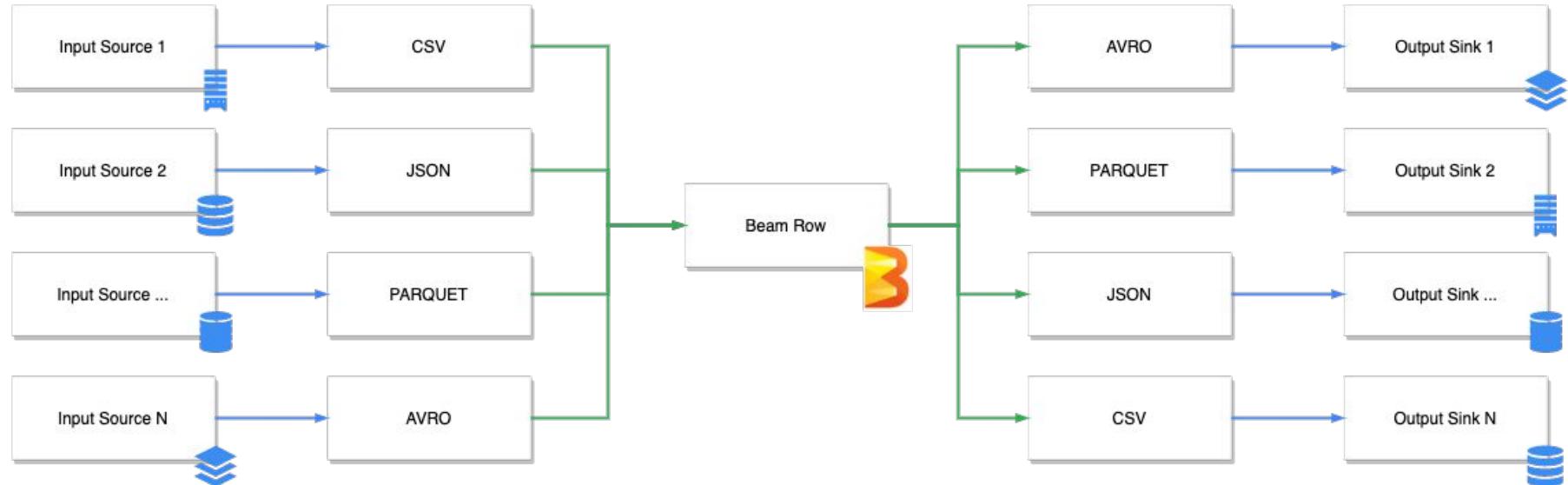
Beam Row

- A schema which supports primitives
- Ordering same with schema
- Quite effective serialization using RowCoder
- Generic for json specified schema
- Available for Beam SQL





How We Use Beam Row?





Demo

- Template overview
- Add a new input source Parquet IO
- Implement Transform Parquet to Row
- Build and Run template





Demo

1

—

2

—

3

—

4

Template Overview



- Java Template structure
- Metadata file
- Schemas
- IO transforms

Add a New Input Source Parquet IO



- Read .parquet files

Implement Transform Parquet to Row



- Transform from Parquet to Beam Row
- Work with schemas

Build and Run Template



- Template building
- Ways to run template

Visit codelab at <https://github.com/griscz/beam-college/tree/main/day3/dataflow-templates>



Contributing to Dataflow Templates

[https://github.com/GoogleCloudPlatform/
DataflowTemplates](https://github.com/GoogleCloudPlatform/DataflowTemplates)

1. Fork the repository to develop your template
2. Follow [style guides](#) and [best practices](#)
3. Sign CLA
4. Create a PR
5. LGTM code review!

The screenshot shows a GitHub repository page for `DataflowTemplates`. The repository has 62 forks and 590 stars. A pull request titled "prathapreddy123 and cloud-teleport PR #176: Kafka to PubSub template" has been merged. The commit message is "Kafka to PubSub template". The PR was created by `prathapreddy123` and `cloud-teleport` on Dec 3, 2020. The commit hash is `f6ebf98`. The repository contains a `src` directory, a `README.md` file, and a `pom.xml` file, all updated 3 months ago. The `README.md` file describes a Dataflow Flex Template to ingest data from Apache Kafka to Google Cloud Pub/Sub. It lists supported data formats (Serializable plaintext formats like JSON, PubSubMessage), input source configurations (Single or multiple Apache Kafka bootstrap servers, SASL/SCRAM authentication over plaintext or SSL connection, Secrets vault service HashiCorp Vault), destination configurations (Single Google Pub/Sub topic), and SSL certificate locations (Bucket in Google Cloud Storage). The template will create an Apache Beam pipeline reading from a source Kafka topic and writing to a specified Pub/Sub destination topic.

GoogleCloudPlatform / DataflowTemplates

Code Issues 74 Pull requests 12 Actions Projects 2 Security Insights

master DataflowTemplates / v2 / kafka-to-pubsub /

prathapreddy123 and cloud-teleport PR #176: Kafka to PubSub template f6ebf98 on Dec 3, 2020 History

src README.md pom.xml

PR #176: Kafka to PubSub template 3 months ago
PR #176: Kafka to PubSub template 3 months ago
PR #176: Kafka to PubSub template 3 months ago

README.md

Dataflow Flex Template to ingest data from Apache Kafka to Google Cloud Pub/Sub

This directory contains a Dataflow Flex Template that creates a pipeline to read data from a single or multiple topics from [Apache Kafka](#) and write data into a single topic in [Google Pub/Sub](#).

Supported data formats:

- Serializable plaintext formats, such as JSON
- `PubSubMessage`

Supported input source configurations:

- Single or multiple Apache Kafka bootstrap servers
- Apache Kafka SASL/SCRAM authentication over plaintext or SSL connection
- Secrets vault service [HashiCorp Vault](#)

Supported destination configuration:

- Single Google Pub/Sub topic

Supported SSL certificate location:

- Bucket in [Google Cloud Storage](#)

In a simple scenario, the template will create an Apache Beam pipeline that will read messages from a source Kafka server with a source topic, and stream the text messages into specified Pub/Sub destination topic. Other scenarios may need Kafka SASL/SCRAM authentication, that can be performed over plain text or SSL encrypted connection. The template supports using a single Kafka user account to authenticate in the provided source Kafka servers and topics. To support SASL authentication over SSL, the template will need access to a secrets vault service with Kafka username and password, and with SSL certificate location in Google Cloud Storage Bucket, currently supporting HashiCorp Vault.



Summary

- Dataflow Flex templates package pipelines into containers
- Stateful processing in Apache Beam
- BeamRow provides flexible abstraction for data representation
- [DataflowTemplates](#) repository - Google and community contributed templates and utilities





Thank You!

<https://akvelon.com/>

[@AkvelonInc](https://twitter.com/AkvelonInc)

