Managed Transforms

Power of Beam without maintenance overheads

Chamikara Jayalath

BEAM SUMMIT NYC 2025

Beam Transforms

- A computation that can be applied on your data
- Can be executed in multiple workers for parallelization
- Can be combined into composite transforms
- Executed in a well defined environment

Beam Transforms - examples

- Core transforms - ParDo, Combine, Flatten
- I/O connectors - Text I/O Read/Write, Iceberg I/O Read/Write, Kafka I/O Read/Write
- Utility transforms - Sum, Top, Count
- Turnkey transforms - RunInference, MLTransform, Enrichment, AnomalyDetection,
- Multi-language transforms - Python SDK Kafka Read/Write, Java SDK RunInference

Some downsides

- Transforms APIs vary widely. Moving from one I/O connector to another usually involves
  - Changing the transforms that produce input
  - Changing the transforms that consume output
  - Changing the transform configuration.
- Runner cannot easily re-configure transforms since transform configuration is language specific and not well defined.

Schema-aware transforms

- Uses standard input/output types: *PCollection<Row>*
- Uses a standard constructor for configuration: *Row*
- Existing transforms can be supported by implementing the interfaces *SchemaTransformProvider* and *SchemaTransform*.

Managed Transforms

- A new API that encapsulates schema-aware transforms.
- Primarily focuses on I/O connectors but can be generalized in the future.
- Standard API to construct transforms
- Standard API to use transforms
- Multi-language compatible

API - Java

```
Managed.read(SOURCE).withConfig(sourceConfig) -> PCollection<Row>
```

```
PCollection<Row> -> Managed.write(SINK).withConfig(sinkConfig)
```

API - Python

```
managed.Read(SOURCE, config=transform_config) -> PCollection<Row>
```

```
PCollection<Row> -> managed.Write(KEY, config=transform_config)
```

Examples

Java BigQuery I/O source

```
Map<String, Object> bqReadConfig = ImmutableMap.of("query", "<query>", ...);
Managed.read(Managed.BIGQUERY).withConfig(bqReadConfig)
```

Java Kafka I/O source

```
Map<String, Object> kafkaReadConfig = ImmutableMap.of("bootstrap_servers", "<server>", "topic", "<topic>",
...);
Managed.read(Managed.KAFKA).withConfig(kafkaReadConfig)
```

Java Kafka I/O source but with a YAML config

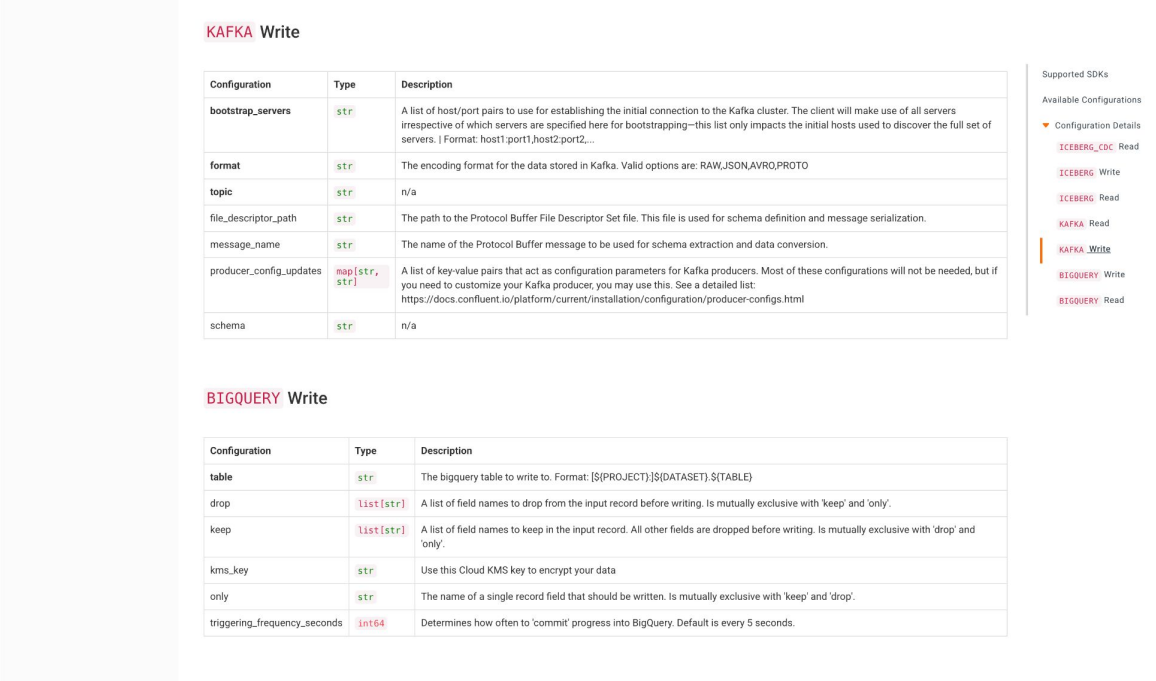
```
String kafkaReadYAMLConfig = "gs://path/to/config.yaml"
Managed.read(Managed.KAFKA).withConfigUrl(kafkaReadYAMLConfig)
```

Python Iceberg I/O source

```
iceberg_config = {"table": "<table>", ...}
managed.Read(managed.ICEBERG, config=iceberg_config)
```

Configuration documentation is auto-generated

https://beam.apache.org/documentation/io/managed-io/



KAFKA Write

Configuration	Type	Description
bootstrap_servers	`str`	A list of host/port pairs to use for establishing the initial connection to the Kafka cluster. The client will make use of all servers irrespective of which servers are specified here for bootstrapping—this list only impacts the initial hosts used to discover the full set of servers. | Format: host1:port1,host2:port2,...
format	`str`	The encoding format for the data stored in Kafka. Valid options are: RAW,JSON,AVRO,PROTO
topic	`str`	n/a
file_descriptor_path	`str`	The path to the Protocol Buffer File Descriptor Set file. This file is used for schema definition and message serialization.
message_name	`str`	The name of the Protocol Buffer message to be used for schema extraction and data conversion.
producer_config_updates	`map[str, str]`	A list of key-value pairs that act as configuration parameters for Kafka producers. Most of these configurations will not be needed, but if you need to customize your Kafka producer, you may use this. See a detailed list: https://docs.confluent.io/platform/current/installation/configuration/producer-configs.html
schema	`str`	n/a

Supported SDKs

Available Configurations

Configuration Details

- `ICEBERG_CDC` Read
- `ICEBERG` Write
- `ICEBERG` Read
- `KAFKA` Read
- `KAFKA` Write
- `BIGQUERY` Write
- `BIGQUERY` Read

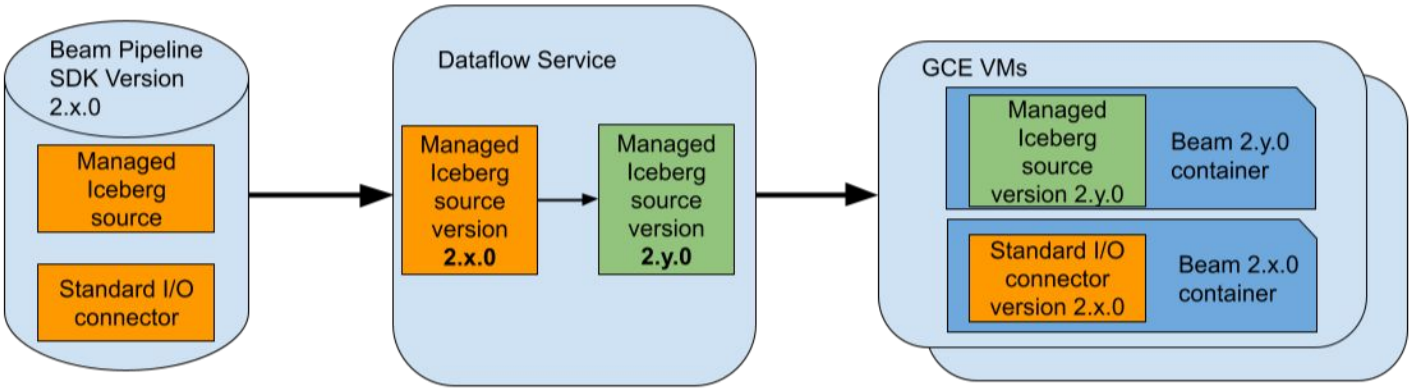
BIGQUERY Write

Configuration	Type	Description
table	`str`	The bigquery table to write to. Format: [${PROJECT}:]${DATASET}.${TABLE}
drop	`list[str]`	A list of field names to drop from the input record before writing. Is mutually exclusive with 'keep' and 'only'.
keep	`list[str]`	A list of field names to keep in the input record. All other fields are dropped before writing. Is mutually exclusive with 'drop' and 'only'.
kms_key	`str`	Use this Cloud KMS key to encrypt your data
only	`str`	The name of a single record field that should be written. Is mutually exclusive with 'keep' and 'drop'.
triggering_frequency_seconds	`int64`	Determines how often to 'commit' progress into BigQuery. Default is every 5 seconds.

Runner side benefits

- Runners can reconfigure transforms using the standard constructor.
- Runners can upgrade transforms using the standard constructor and the input/output types.
- Managed transforms are guaranteed to be upgrade compatible.
- Managed transforms are guaranteed to be update compatible for streaming.
- Runner side features are currently only supported by Dataflow
    - Managed transforms work for all runners

Dataflow Implementation



Dataflow - auto upgrades

- Dataflow service automatically upgrades the supported transforms to the latest version.
- Also upgrades the dependencies used by the transform.
- This means that critical bug fixes and vulnerability updates will be automatically applied by the service.
- Upgrading is performed during
  - Initial job submission
  - Update via replacement (streaming jobs only)

Dataflow - transform auto re-configuration

- Dataflow may automatically re-configure the transforms to better suit the current configuration of the overall pipeline.

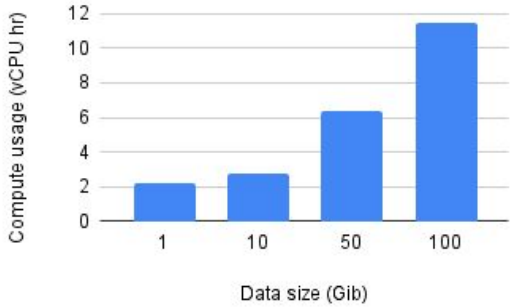
- For example for BigQuery I/O sink transforms delivery semantics is automatically configured to map the Dataflow streaming mode
  - Dataflow streaming at-least once -> BigQuery storage write API at-least-once delivery semantics, which is less expensive and results in lower latencies.

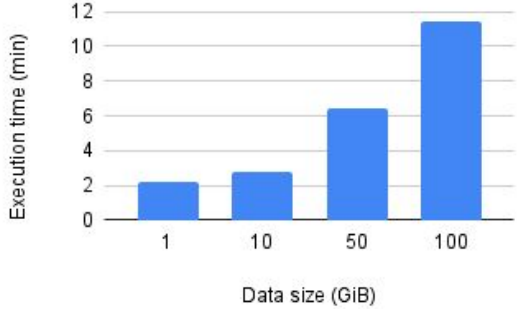
Overall benefit

You can stay in the same Beam version and improve your pipeline and hand over responsibility of upgrading and optimizing your transform to the Dataflow runner.

Performance - Iceberg I/O

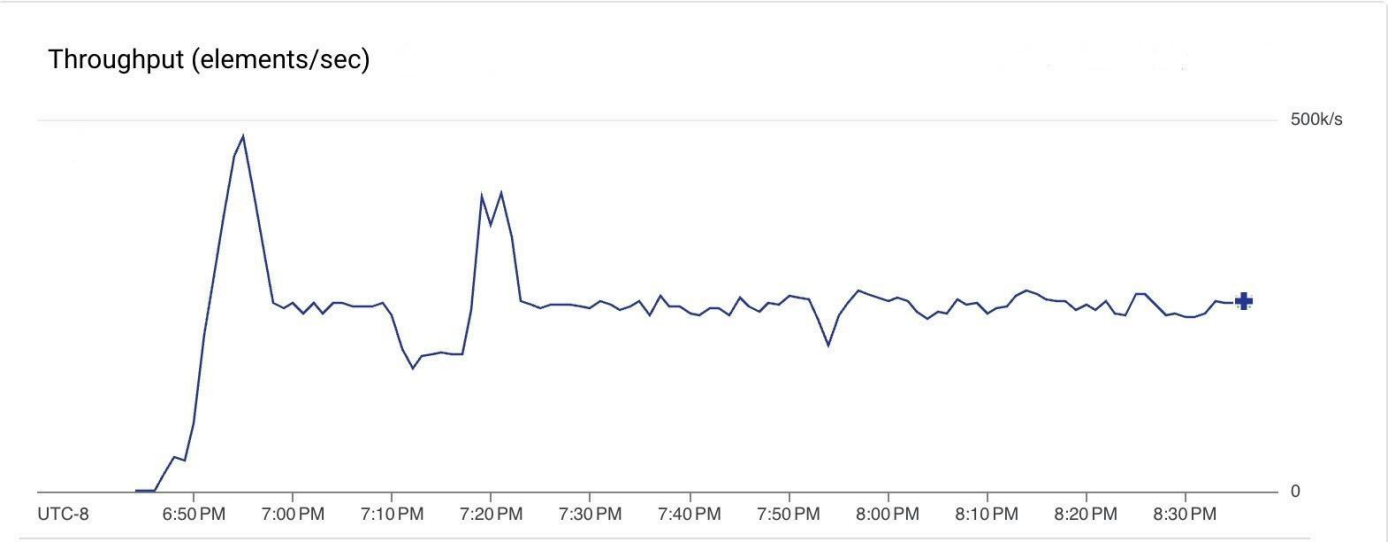
- Managed Iceberg I/O sink backed by a Hadoop catalog deployed in GCS.
- submitted using Beam 2.61.0 and the Managed I/O sink was automatically upgraded by Dataflow to the latest supported version
- 100 n1-standard-4 worker VMs.



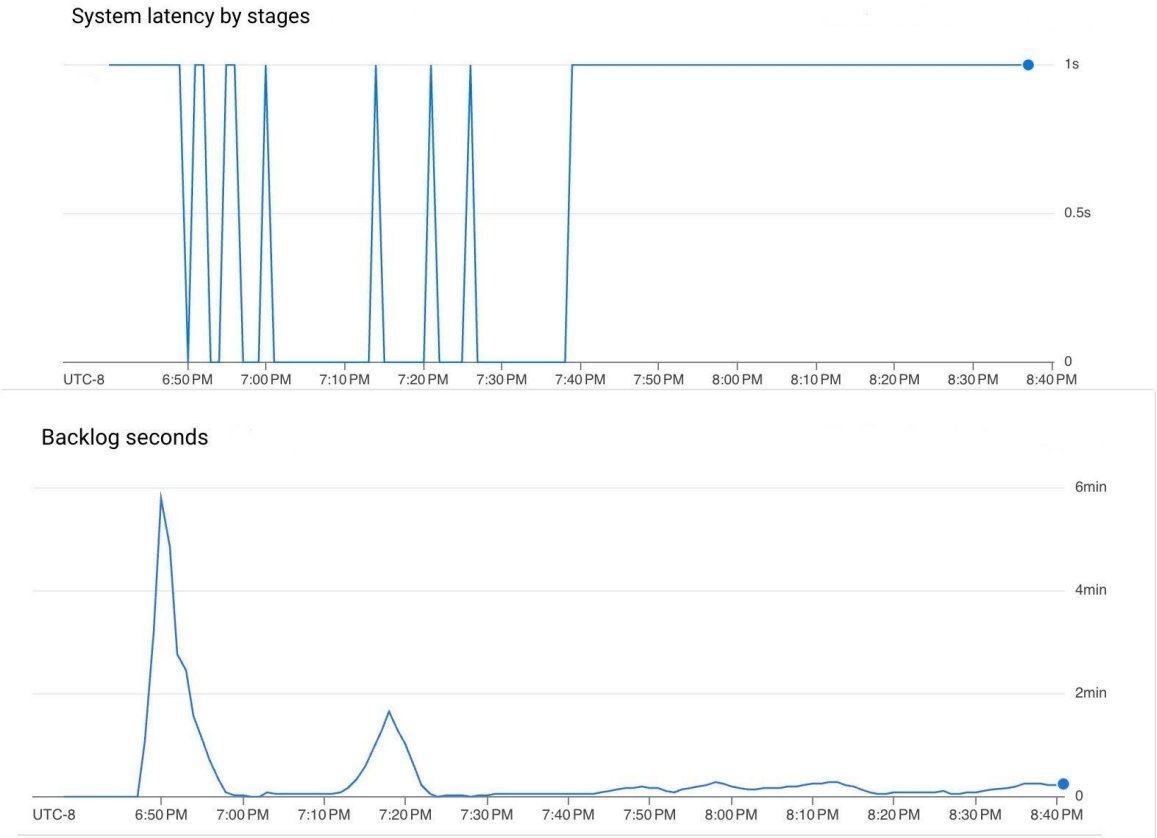


Performance Kafka I/O

- A streaming pipeline that read from Google Pub/Sub and used the Managed Kafka sink transform to push messages to a Kafka cluster hosted in GCP
- Uses 10 n1-standard-4 wokers at steady state (max 20)



Performance Kafka I/O



Resources

Transform configuration (auto-generated): https://beam.apache.org/documentation/io/managed-io/

Dataflow support: https://cloud.google.com/dataflow/docs/guides/managed-io

Java API:
https://beam.apache.org/releases/javadoc/current/org/apache/beam/sdk/managed/Managed.html

Python API:
https://beam.apache.org/releases/pydoc/current/apache_beam.transforms.managed.html#module-apache_beam.transforms.managed

Chamikara Jayalath

Thank you

BEAM SUMMIT NYC 2025