# How to balance power and control when using Dataflow with an OLTP SQL Database ?

By Florian Bastin
& Léo Babonnaud

BEAM SUMMIT
NYC 2023

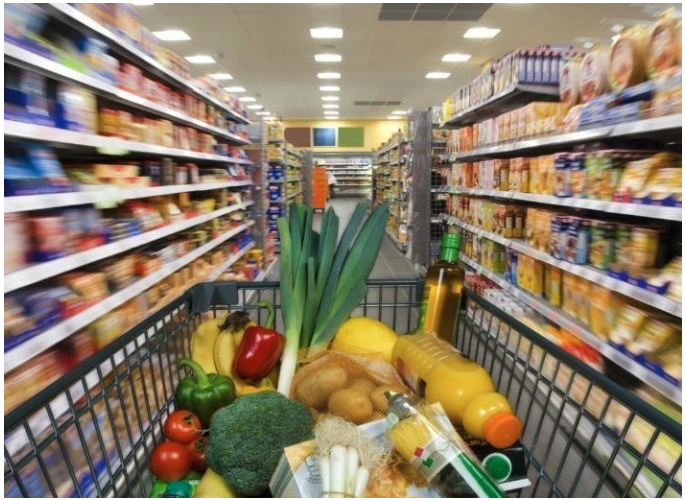
# About us

Florian Bastin
Data scientist @Octo

Léo Babonnaud
Data engineer @Octo

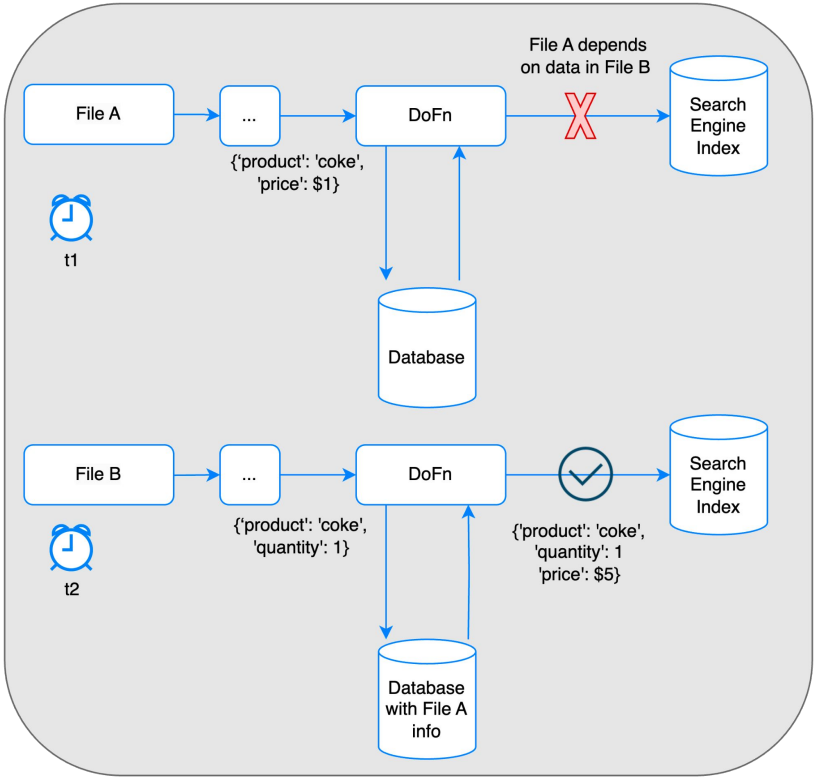
# The Use Case

- A Retail Customer use case
- Products information pipeline to serve a search engine index



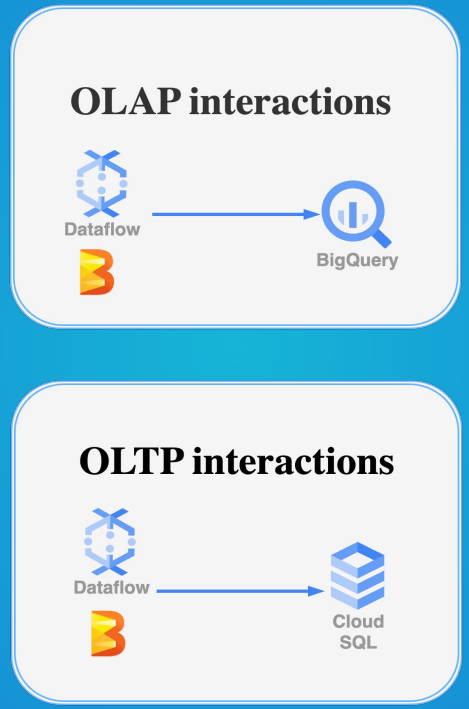
# Requirements.txt

- Dependency between file rows
- No order in files reception
- Random time interval between files reception
- Most updated data in the search engine
- File B may never arrive



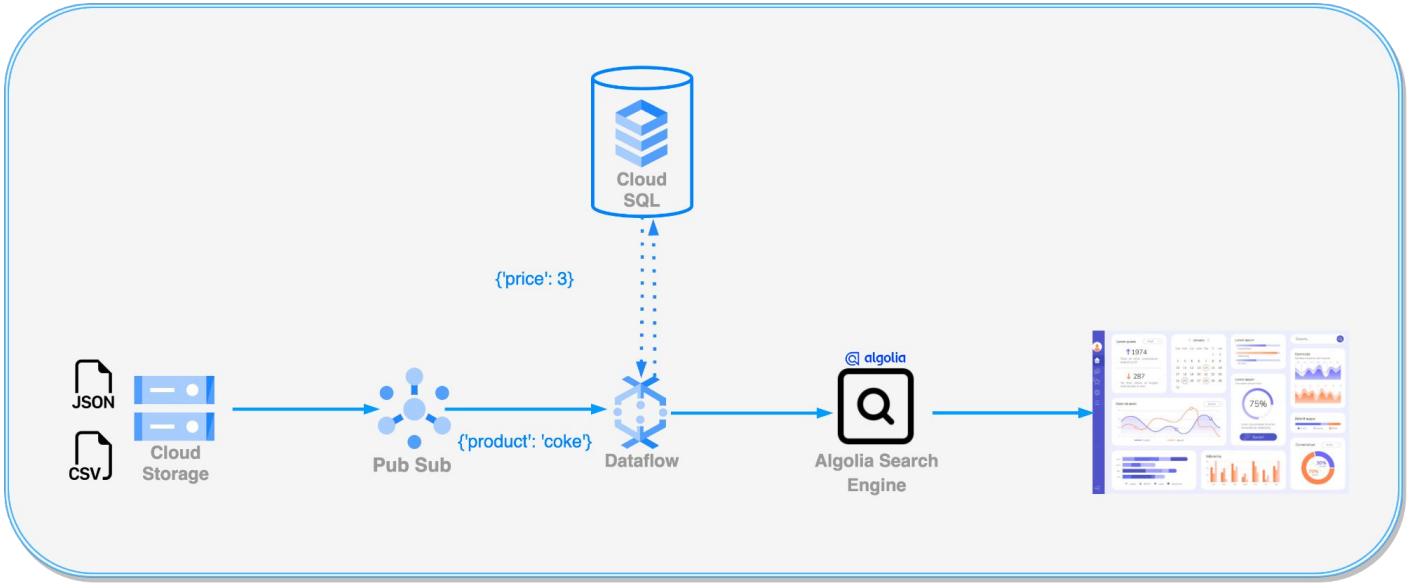
# What kind of storage ?

- BigQuery storage, as an OLAP database is used for large storage and fast analytics request
- Cloud SQL  is a transactional database good for fast interactions and modifications at the row level.

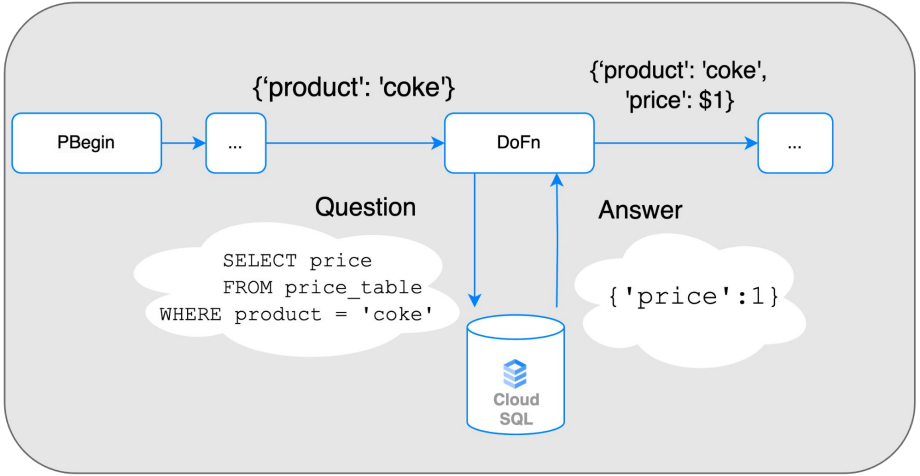


# Our Architecture

- Streaming pipeline
- Near real time
- Mini batches
- CRUD operations



# An I/O Cloud SQL connector ?




*class* **apache_beam.io.jdbc.ReadFromJdbc**(*driver_class_name, jdbc_url, username, password, query, output_parallelization=None, fetch_size=None, connection_properties=None, connection_init_sqls=None, expansion_service=None*) [source]

*class* **apache_beam.io.jdbc.WriteToJdbc**(*driver_class_name, jdbc_url, username, password, statement, connection_properties=None, connection_init_sqls=None, expansion_service=None*) [source]

# Developing our own connectors: No easy way

**How to develop your own I/O connectors using ParDo & GroupByKey operators?**

- Connector type
- Pool / Connections control
- Failed worker retry
- Dataflow autoscaling
- Idempotency

# The Cloud Sql connector

The *Cloud SQL Python Connector* provides the following benefits:

- Uses IAM permissions to control who/what can connect to your Cloud SQL instances
- Improved Security between the client connector and the server-side proxy.
- Removes the requirement to use and distribute SSL certificates, as well as manage firewalls or source/destination IP addresses.

```python
from google.cloud.sql.connector import Connector
import sqlalchemy

# initialize Connector object
connector = Connector()

# function to return the database connection
def getconn() -> pymysql.connections.Connection:
    conn: pymysql.connections.Connection = connector.connect(
        "project:region:instance",
        "pymysql",
        user="my-user",
        password="my-password",
        db="my-db-name"
    )
    return conn

# create connection pool
pool = sqlalchemy.create_engine(
    "mysql+pymysql://",
    creator=getconn,
)
```

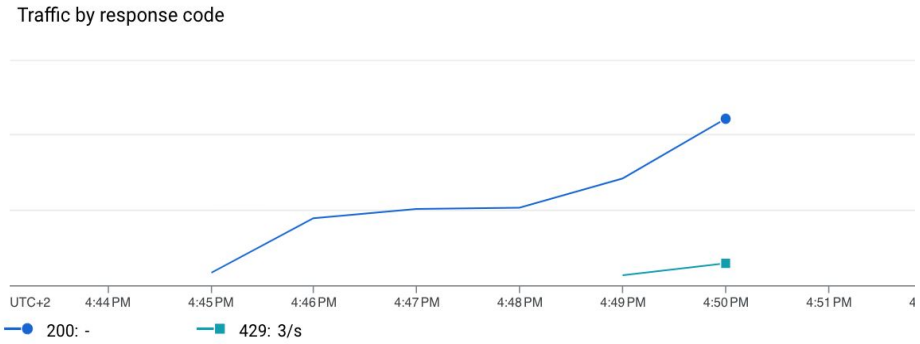
→ `DoFn.setup()` : Called whenever the `DoFn` instance is deserialized on the worker. This means it can be called more than once per worker because multiple instances of a given `DoFn` subclass may be created (e.g., due to parallelization, or due to garbage collection after a period of disuse). This is a good place to connect to database instances, open network connections or other resources.

# The Limit of the Cloud SQL Connector

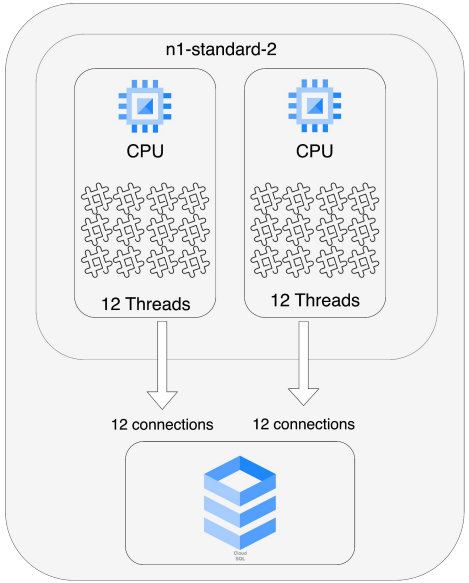

```
  File "/usr/local/lib/python3.9/site-packages/aiohttp/client_reqrep.py", line 1
    raise ClientResponseError(
aiohttp.client_exceptions.ClientResponseError: 429, message='Too Many Requests',
```

- 2 Cloud SQL Admin API calls/connection
- 600 Cloud SQL Admin API calls/minute

→ 300 new connections/minute maximum



# How many concurrent operations in Apache Beam ?




- Default 2 vCPUs
- Default 12 threads
- Max 1 DoFn per thread
- 24 Do Fns/n1-standard-2

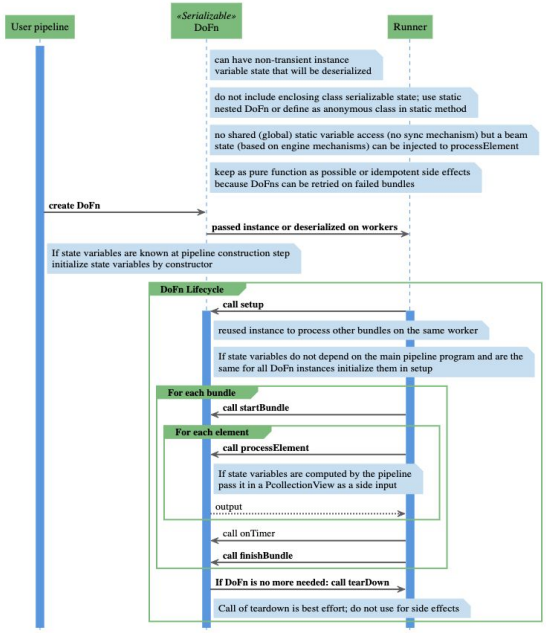
Beam Python SDK | Beam Java/Go SDK

| | Batch | Streaming without Streaming Engine | Streaming Engine |
|---|---|---|---|
| Parallelism | 1 process per vCPU<br>1 thread per process<br><br>1 thread per vCPU | 1 process per vCPU<br>12 threads per process<br><br>12 threads per vCPU | 1 process per vCPU<br>12 threads per process<br><br>12 threads per vCPU |
| Maximum number of concurrent DoFn instances (All of these numbers are subject to change at any time.) | 1 DoFn per thread<br><br>1 DoFn per vCPU | 1 DoFn per thread<br><br>12 DoFn per vCPU | 1 DoFn per thread<br><br>12 DoFn per vCPU |

***max(connections) = # workers x vCPUs/worker x # threads x # SQLDoFns steps***

# Leverage the beam.shared module

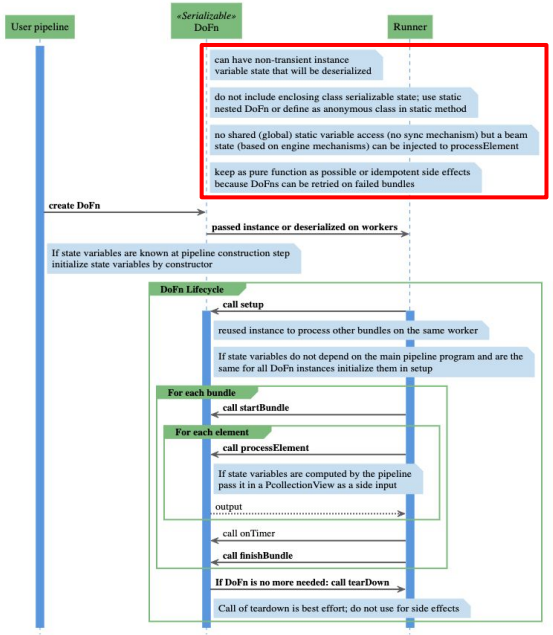
- Pool of connections shared at the worker level
- apache_beam.utils.Shared module




***max(connections) = # workers x # vCPUs/worker x pool size***

# Failure handling

- Side effects of the DoFn setup
- Exponential backoff algorithm




***max(connections) = # workers x # vCPUs/worker x pool size***

# A focus on idempotency

**How to deal with “the not exactly” once execution ?**

- **‘SQL’ Definition** : An operation that produces the same results no matter how many times it is performed
- INSERT and UPDATE statements have to be done carefully
- Check the state of a row before applying a statement

```
CREATE TABLE my_schema.product_table (
    id VARCHAR(32) NOT NULL,
    product VARCHAR(32) NOT NULL,
    quantity NUMERIC(9) NOT NULL
    PRIMARY KEY(id)
);

-- KO
INSERT INTO my_schema.product_table ('id', 'product')
VALUES ('coke_id', 'coke', 1000)
-- ERROR:  duplicate key value violates unique constraint...

-- OK
INSERT INTO my_schema.product_table ('id', 'product')
VALUES ('coke_id', 'coke', 1000)
ON CONFLICT DO NOTHING
```

# Our solution

```python
def pipeline(pipeline_options, data_size: int, max_num_workers: int, pool_size: int):
    with beam.Pipeline(options=pipeline_options) as p:
        shared_handle = shared.Shared()

        data = p | beam.Create(range(data_size)) | beam.Reshuffle()
        prepared_data = data | beam.ParDo(PreprocessData())

        enriched_data = (prepared_data
                         | "Reduce parallelism" >> beam.Reshuffle(num_buckets=max_num_workers)
                         | "Cloud SQL Do Fn" >> beam.ParDo(EnrichSQLDoFn(shared_handle=shared_handle, pool_size=pool_size))
                         | "Maximize parallelism" >> beam.Reshuffle()
                         )

        enriched_data | beam.ParDo(AnotherPreprocessData())
```

# Key takeaways

- The available Apache Beam Cloud SQL connectors are useful as an input or output of a pipeline
- Configuring your own Cloud SQL connector using DoFns requires:
  - The use of the Setup method for instantiation
  - The use of beam Shared module to share a connection pool
  - A well defined connector object if the workload is heavy
  - A retry mechanism in case of failed requests (Exponential Backoff Algorithm for instance)
  - Carefully chosen idempotent SQL statements

How to balance power and control when using Dataflow with an OLTP SQL Database ?

# QUESTIONS?

Florian Bastin: Linkedin
Léo Babonnaud: Linkedin

Octo Technology: https://octo.com/