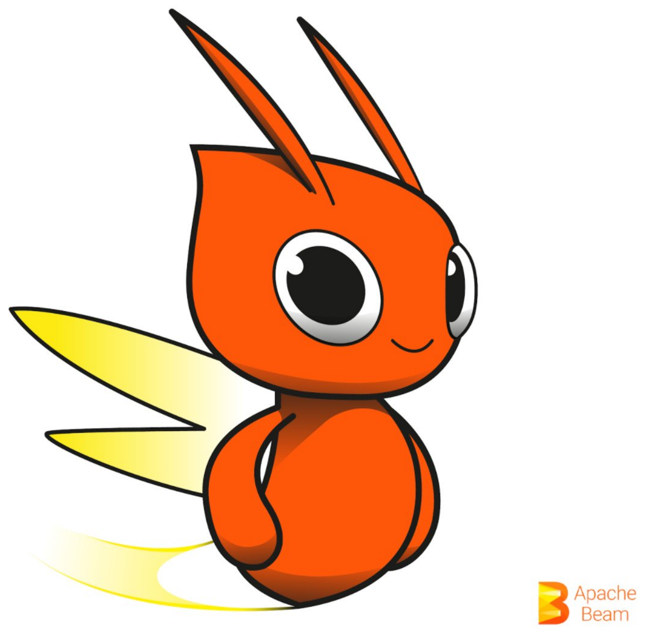
Building an streaming cloud production data pipeline with Apache Beam

Esteban Maya Cadavid



About me

- Software Engineer and consultant
- VC Investor.
- Co-Organizer of PyCon Colombia Conference https://pycon.co/
- Co-Organizer of Python Medellin / Colombia Community
- Born and base in Colombia, Medellin

Agenda

- Introduction.
  - Introduction to Beam.
  - Beam model and concepts.
  - Overview Beam io packages.
  - Beam supported languages.
- Architecture
  - A common ETL flow
  - Streaming data pipeline architecture overview.
  - Tips and tricks
  - Debugging streaming pipeline locally (Demo live).
- Deploy
  - Deploying the data pipeline in cloud runner (Demo Live).
- Q/A session.

Introduction to Apache beam

What is Apache Beam?

- An advanced unified programming model.
- Implement batch and streaming data jobs agnostic that run in cloud, locally and many others runners/execution engines.

Why Beam?

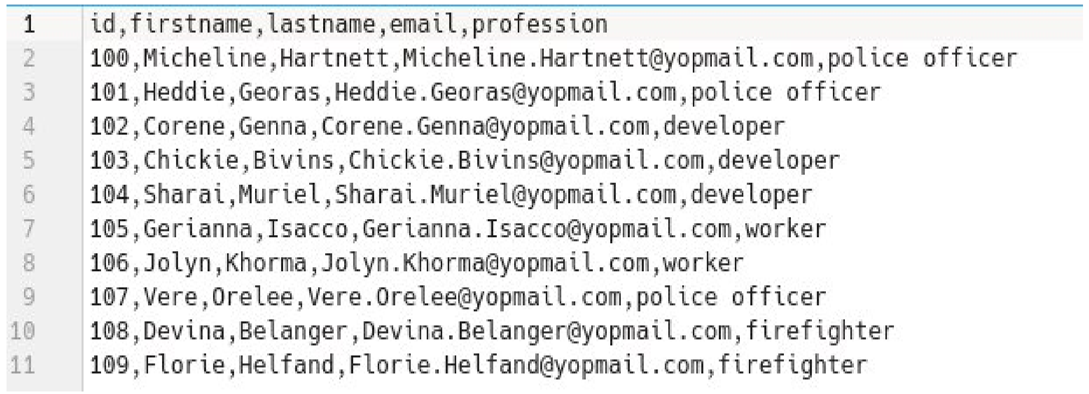
- Write the code one time in (Java, Python, Go) and run anywhere.
- It’s an agnostic technology which means you can run it in cloud runners like Google Dataflow. Or you can run it in your own infrastructure/machine using spark, flink, samza.
- It’s open source, portable, flexible and extensible.

Beam Model and Concepts

PCollections

The PCollection abstraction represents a potentially distributed, multi-element data set.

For example if you are reading a csv file like this.



```
id,firstname,lastname,email,profession
100,Micheline,Hartnett,Micheline.Hartnett@yopmail.com,police officer
101,Heddie,Georas,Heddie.Georas@yopmail.com,police officer
102,Corene,Genna,Corene.Genna@yopmail.com,developer
103,Chickie,Bivins,Chickie.Bivins@yopmail.com,developer
104,Sharai,Muriel,Sharai.Muriel@yopmail.com,developer
105,Gerianna,Isacco,Gerianna.Isacco@yopmail.com,worker
106,Jolyn,Khorma,Jolyn.Khorma@yopmail.com,worker
107,Vere,Orelee,Vere.Orelee@yopmail.com,police officer
108,Devina,Belanger,Devina.Belanger@yopmail.com,firefighter
109,Florie,Helfand,Florie.Helfand@yopmail.com,firefighter
```

The dataset above is our PCollection.

PTransforms

Transforms are the operations in your pipeline. You provide processing logic in the form of a function object (colloquially referred to as “user code”), and your user code is applied to each element of an input PCollection

For example

```
def get_columns(data):
    id, firstname, lastname, email, profession = data.split(“,”)

    return [{
        “id”: id,
        “firstname”: firstname,
        “lastname”: lastname,
        “email”: email,
        “profession”: profession
    }]
```

The function above is a PTransform which is applied to get the data over each element from our previos PCollection and parse them in a dictionary object.

Beam io packages

Apache beam support most of the know well industry sources and destinations technologies. Here is some of them

Cloud storage (Google storage GCS, AWS S3, Azure blobstorage...).
Queue brokers (Kafka, Google pub/sub, AWS SQS...)
Files (Plain text, csv, parquet...)
Databases (Mongo, JDBC, Snowflake, Bigquery, Cassandra...)

You can consult the entire list according the language here.

- https://beam.apache.org/releases/pydoc/2.32.0/apache_beam.io.html
- https://pkg.go.dev/github.com/apache/beam/sdks/go/pkg/beam?utm_source=godoc
- https://beam.apache.org/releases/javadoc/2.32.0/

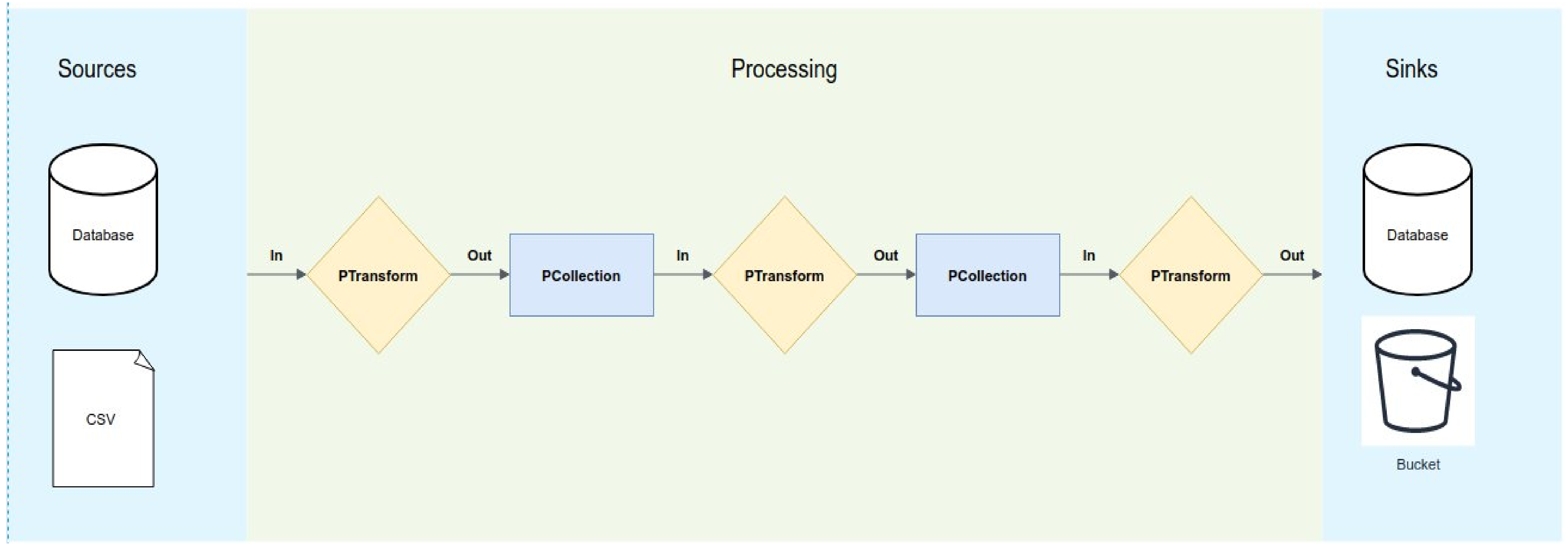
Beam sdk supported languages.

Java

beam

A common ETL flow

Understanding the things mentions before, most of the time our model pipeline looks like.



(Source)

- We need a queue broker like rabbitmq, kafka, AWS SQS or google pub/sub when we are building streaming pipelines.
  - For batch pipelines the source could be a file text, csv or database.

- Streaming common sources
- Batch common sources

APACHE kafka

Cloud Pub/Sub

amazon SQS

RabbitMQ™

CSV

SQL

(Processing)

- An ETL model we wrote before needs a runner. The supported runners are Google Dataflow, Flink, Spark, samza and more.

- Most known runners supported by Beam

beam

Google Dataflow

samza

Flink

APACHE Spark™

(Sink/Destination)

A common good destination sources are PostgreSQL, S3, cloud storage, MongoDB, Google Bigquery, Amazon RDS.

- Sinks/Destination

Amazon S3

Amazon RDS

PostgreSQL

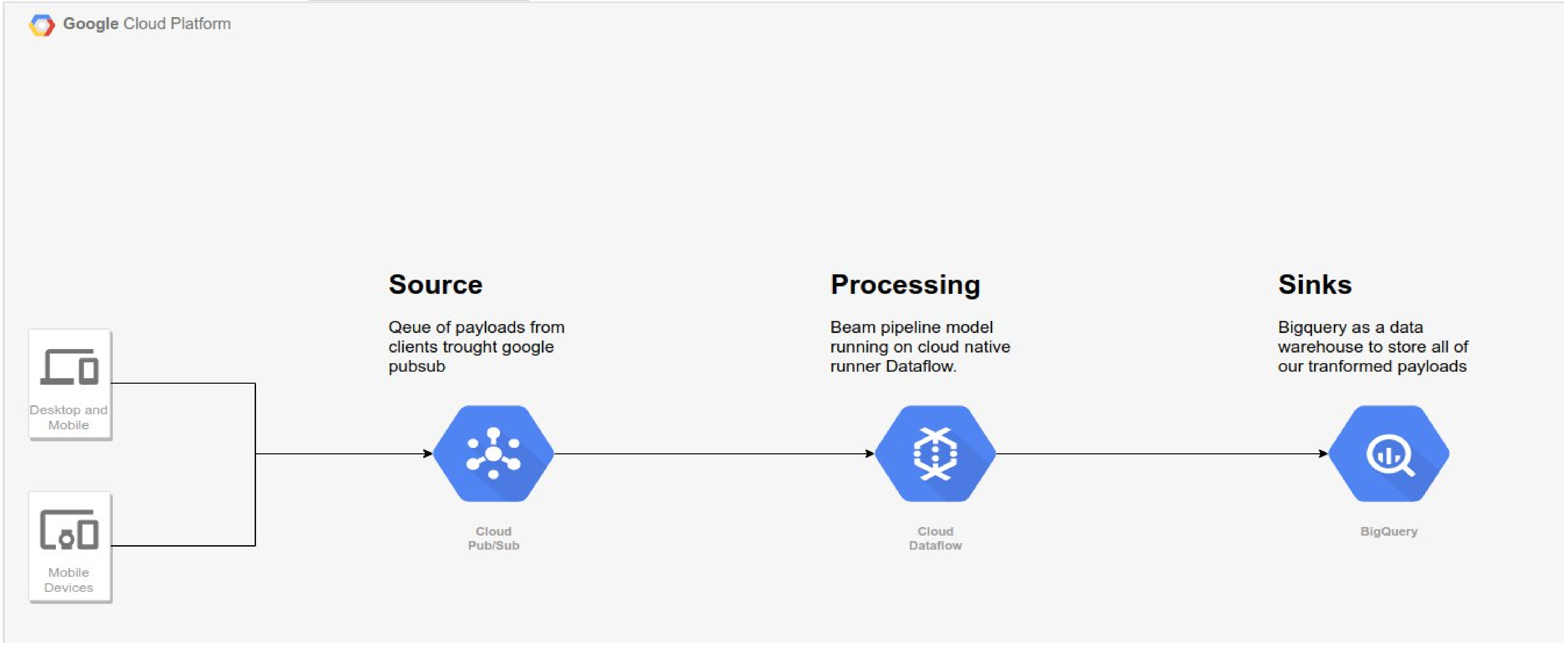
CSV

BigQuery

mongoDB®

Architecture for streaming data pipeline.

- Our architecture pipeline looks like


Google Cloud Platform
Source
Qeue of payloads from clients trought google pubsub
Processing
Beam pipeline model running on cloud native runner Dataflow.
Sinks
Bigquery as a data warehouse to store all of our tranformed payloads
Desktop and Mobile
Mobile Devices
Cloud Pub/Sub
Cloud Dataflow
BigQuery

Architecture for streaming data pipeline.

Stages:

- We are going to use google pub/sub because is serverless, auto scalable and not ops config needed (Source).

- We are going to use dataflow, because is auto scalable and we don’t need a lot of configuration or ops knowledge (Processing).

- We are goin to use Google bigquery beacause again here is serverless, auto scalable, without ops config and cheaper only pays for the query executed (Sink).

Architecture Performance tips for streaming data pipeline.

- Avoid to make request to another api or external network resource, this can delay or block your pipeline.

```
Events_payload = requests.get('https://api.github.com/events') # tooks 100ms-200ms for each requests

#If we have 1.000.000 of payloads so 100ms-200ms x 1.000.000, this could be broke the pipeline or have a delay issue
```

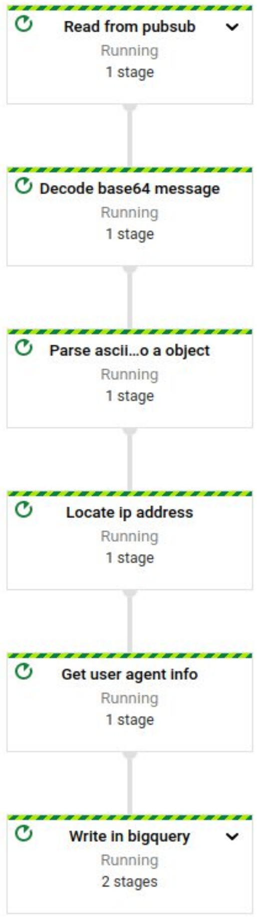
Architecture Performance tips for streaming data pipeline.

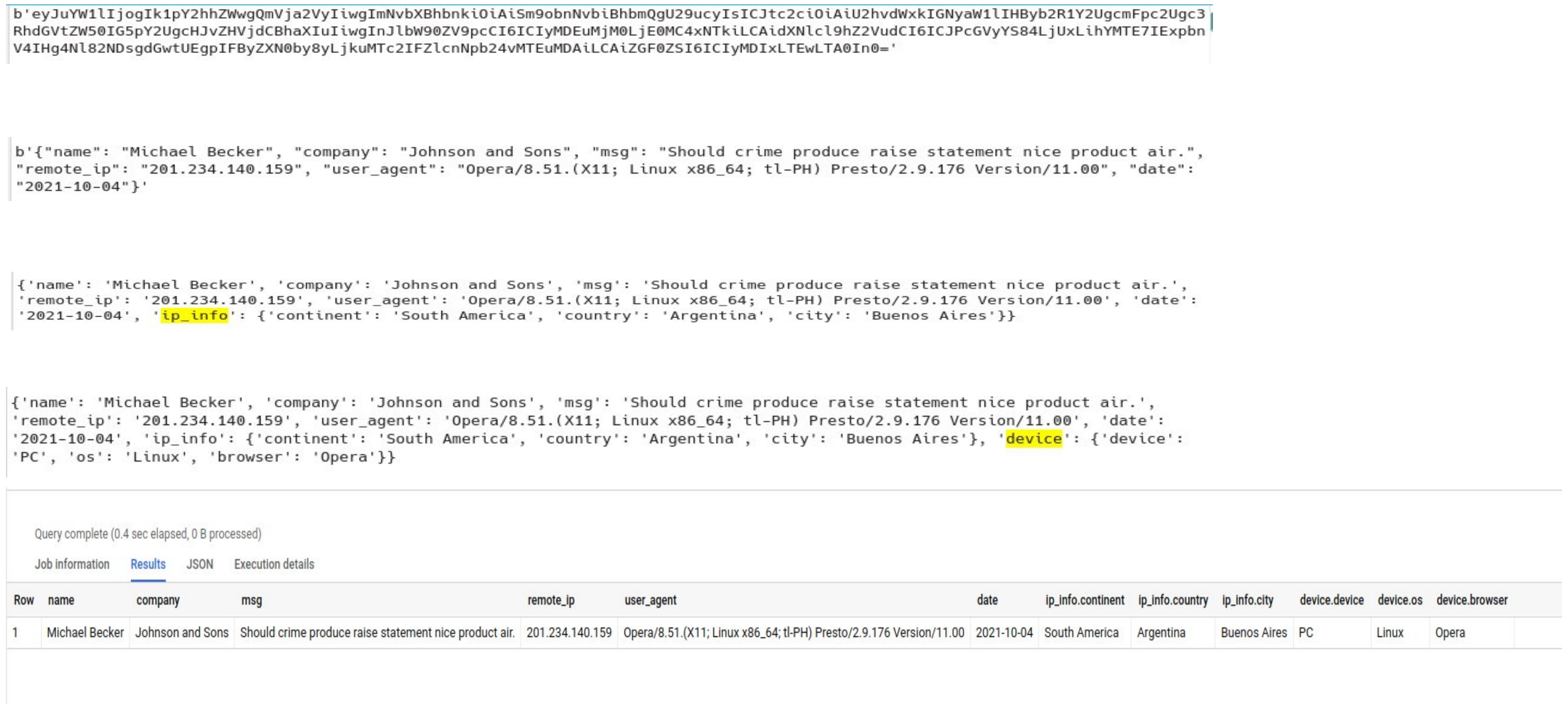
- Avoid to make validations againts database or any other resource outside of the logic pipe.

```
•   user = UserModel.find(id=1234)
•   If user.active:
•     #Do something here
•   Elif user.name == “Joe”:
•     # Another logic here
•
# request a database with queries and validate logic that depends of external resources like this
is not recommended for streaming pipelines. The database could have a heavy load and the pipeline
can be broke or stop due database response.
```

Architecture for streaming data pipeline.

- This is our beam processing model for demo





```
b'eyJuYW1lIjogIk1pY2hhZWwgQmVja2VyIiwgImNvbXBhbnkiOiAiSm9obnNvbiBhbmQgU29ucyIsICJtc2ciOiAiU2hvdWxkIGNyaW1lIHByb2R1Y2UgcmFpc2Ugc3
RhdGVtZW50IG5pY2UgcHJvZHVjdCBhaXIuIiwgInJlbW90ZV9pcCI6ICIyMDEuMjM0LjE0MC4xNTkiLCAidXNlcl9hZ2VudCI6ICJPcGVyYS84LjUxLihYMTE7IExpbn
V4IHg4Nl82NDsgdGwtUEgpIFByZXN0by8yLjkuMTc2IFZlcnNpb24vMTEuMDAiLCAiZGF0ZSI6ICIyMDIxLTEwLTA0In0='
```

```
b'{"name": "Michael Becker", "company": "Johnson and Sons", "msg": "Should crime produce raise statement nice product air.",
"remote_ip": "201.234.140.159", "user_agent": "Opera/8.51.(X11; Linux x86_64; tl-PH) Presto/2.9.176 Version/11.00", "date":
"2021-10-04"}'
```

```
{'name': 'Michael Becker', 'company': 'Johnson and Sons', 'msg': 'Should crime produce raise statement nice product air.',
'remote_ip': '201.234.140.159', 'user_agent': 'Opera/8.51.(X11; Linux x86_64; tl-PH) Presto/2.9.176 Version/11.00', 'date':
'2021-10-04', 'ip_info': {'continent': 'South America', 'country': 'Argentina', 'city': 'Buenos Aires'}}
```

```
{'name': 'Michael Becker', 'company': 'Johnson and Sons', 'msg': 'Should crime produce raise statement nice product air.',
'remote_ip': '201.234.140.159', 'user_agent': 'Opera/8.51.(X11; Linux x86_64; tl-PH) Presto/2.9.176 Version/11.00', 'date':
'2021-10-04', 'ip_info': {'continent': 'South America', 'country': 'Argentina', 'city': 'Buenos Aires'}, 'device': {'device':
'PC', 'os': 'Linux', 'browser': 'Opera'}}
```

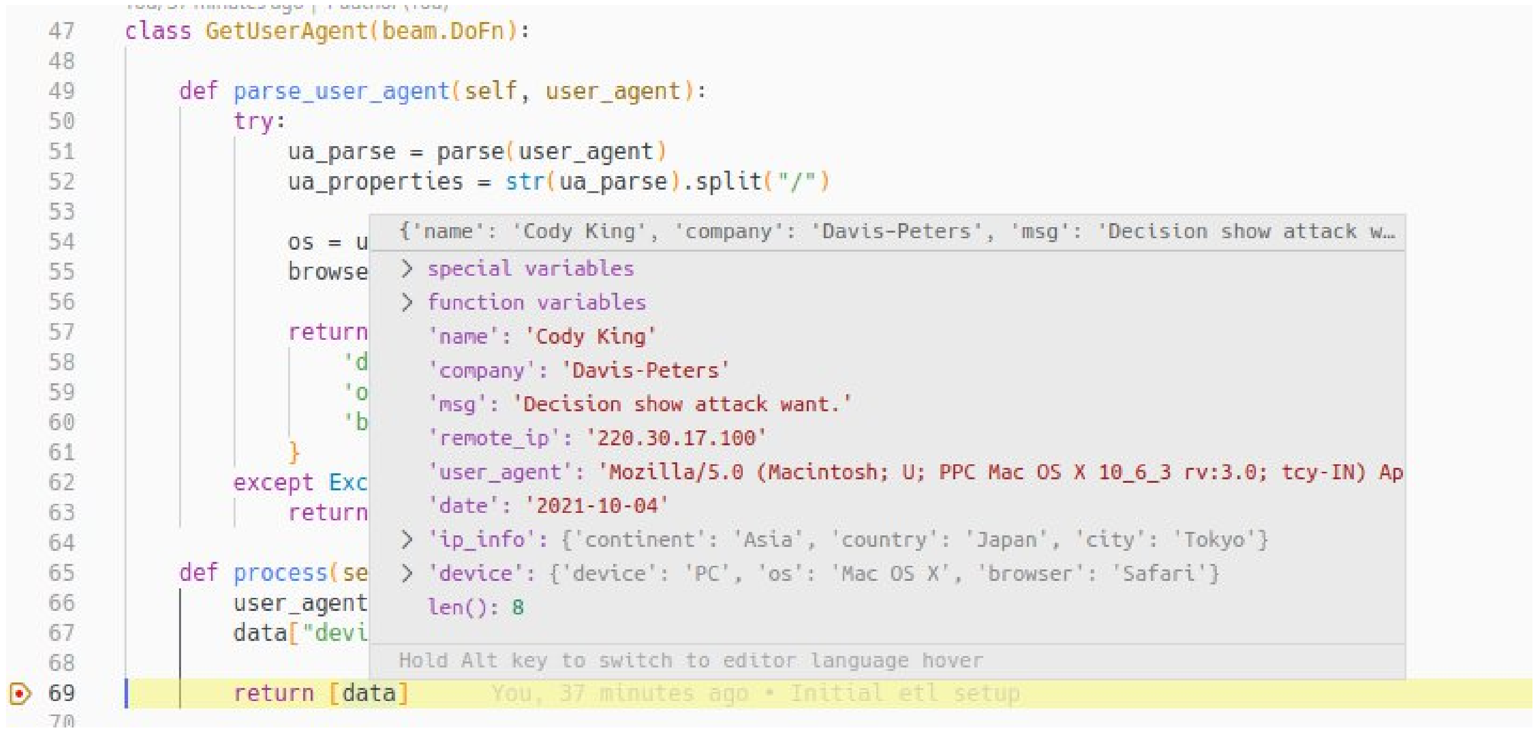
Query complete (0.4 sec elapsed, 0 B processed)

Job information Results JSON Execution details

Row	name	company	msg	remote_ip	user_agent	date	ip_info.continent	ip_info.country	ip_info.city	device.device	device.os	device.browser
1	Michael Becker	Johnson and Sons	Should crime produce raise statement nice product air.	201.234.140.159	Opera/8.51.(X11; Linux x86_64; tl-PH) Presto/2.9.176 Version/11.00	2021-10-04	South America	Argentina	Buenos Aires	PC	Linux	Opera

Tips and tricks debugging streaming pipeline locally

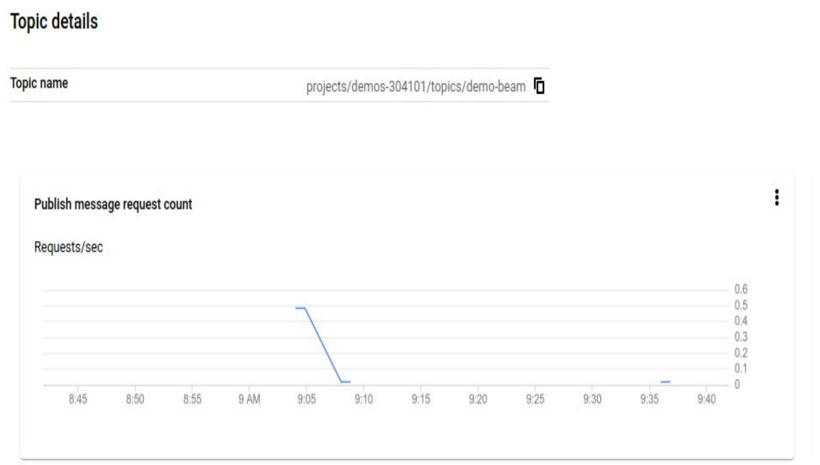
Apache Beam provide us a DirectRunner which we can use to debug the pipeline in our favorite IDE. (Demo Live)



```
class GetUserAgent(beam.DoFn):

    def parse_user_agent(self, user_agent):
        try:
            ua_parse = parse(user_agent)
            ua_properties = str(ua_parse).split("/")

            os = u
            browse

            return
                'd
                'o
                'b
            }
        except Exc
            return

    def process(se
        user_agent
        data["devi

        return [data]      You, 37 minutes ago • Initial etl setup
```

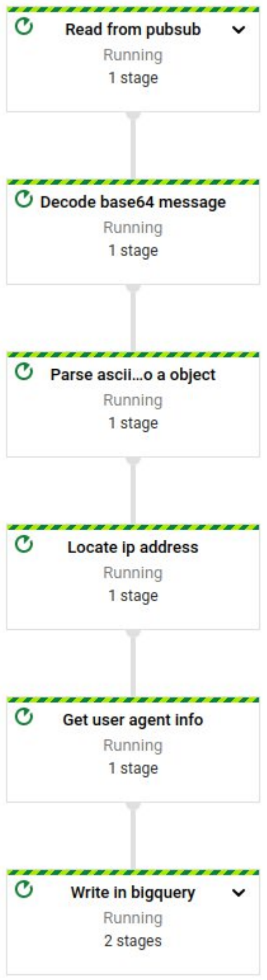
```
{'name': 'Cody King', 'company': 'Davis-Peters', 'msg': 'Decision show attack w…
> special variables
> function variables
  'name': 'Cody King'
  'company': 'Davis-Peters'
  'msg': 'Decision show attack want.'
  'remote_ip': '220.30.17.100'
  'user_agent': 'Mozilla/5.0 (Macintosh; U; PPC Mac OS X 10_6_3 rv:3.0; tcy-IN) Ap
  'date': '2021-10-04'
> 'ip_info': {'continent': 'Asia', 'country': 'Japan', 'city': 'Tokyo'}
> 'device': {'device': 'PC', 'os': 'Mac OS X', 'browser': 'Safari'}
  len(): 8
Hold Alt key to switch to editor language hover
```

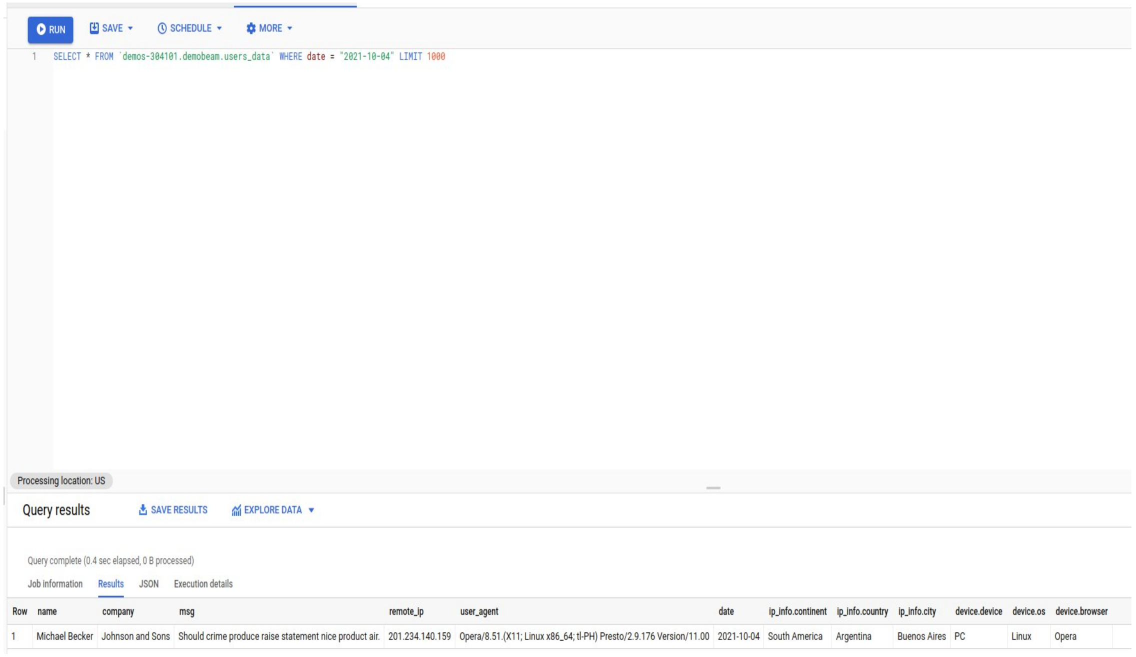
Deploying the data pipeline in cloud runner (Demo Live).



Topic details

Topic name	projects/demos-304101/topics/demo-beam





```
SELECT * FROM `demos-304101.demobeam.users_data` WHERE date = "2021-10-04" LIMIT 1000
```

Processing location: US

Query results

Query complete (0.4 sec elapsed, 0 B processed)

Row	name	company	msg	remote_ip	user_agent	date	ip_info.continent	ip_info.country	ip_info.city	device.device	device.os	device.browser
1	Michael Becker	Johnson and Sons	Should crime produce raise statement nice product air.	201.234.140.159	Opera/8.51.(X11; Linux x86_64; tl-PH) Presto/2.9.176 Version/11.00	2021-10-04	South America	Argentina	Buenos Aires	PC	Linux	Opera