Beam at Talend:
the long road together

Alexey Romanenko
Qlik/Talend
Open Source Engineer
Beam PMC

BEAM SUMMIT
NYC 2023

A bit of history...

- **2006** Founded in **2006**, Talend was the first company to market open-source data integration software;
- **2006** Released in **October 2006**, Talend Open Studio is the company's first product;
- **2007** **July 2007**, Talend launched its first commercial version, Talend Data Integration;
- **2015** **March 2015**, the company launched Talend Integration Cloud to enable developers to simplify and accelerate cloud and hybrid integration projects;
- **2016** **January 2016**, Talend joins Cloudera, Data Artisans, Google, Cask and Paypal on the Apache Foundation's Google's Cloud Dataflow project - Apache Beam;
- **2018** **May 2018**, Talend launched Talend Data Streams for AWS - a new free offering for self-service integration;
- **2019** **April 2019**, the company launched Talend Pipeline Designer (formerly Talend Data Streams), a next generation data integration design environment included in Talend Cloud.
- **2023** **May 2023, Qlik** acquires **Talend**

Talend and Open Source

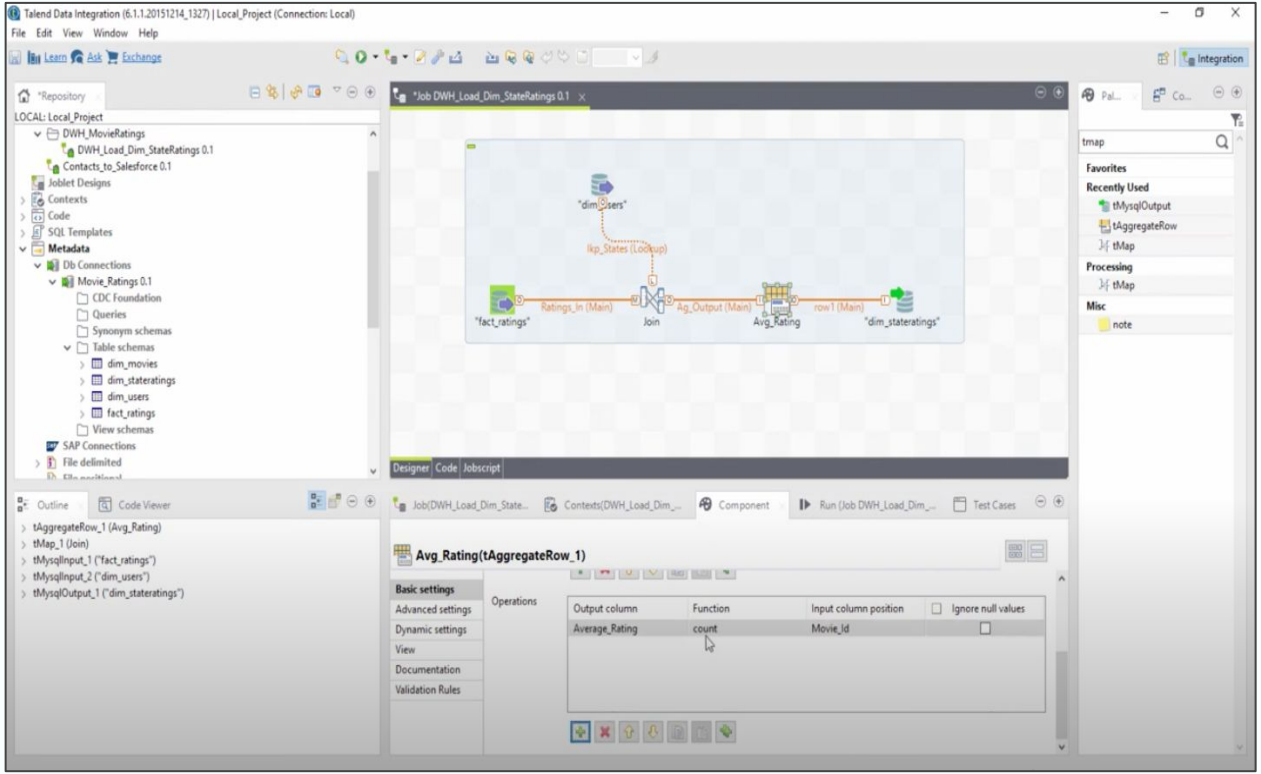
- Talend has a rich Open Source culture from the very beginning ;
- Talend is a long-time partner of the **ASF** ;
- Open Source team at Talend is **ASF** contributor for many projects:
  - notably in the *Apache CXF, Camel, Karaf, ActiveMQ, Beam, Spark, Flink, Avro* and other projects;
- Help to mentor numerous projects through the **ASF Incubator** ;
  - Beam is a good example
- The company is also a member of other open source foundations:
  - *Java Community Process (JCP), Eclipse Foundation, OW226* and *the Open Source School*.

Beam at Talend

The long and winding road...

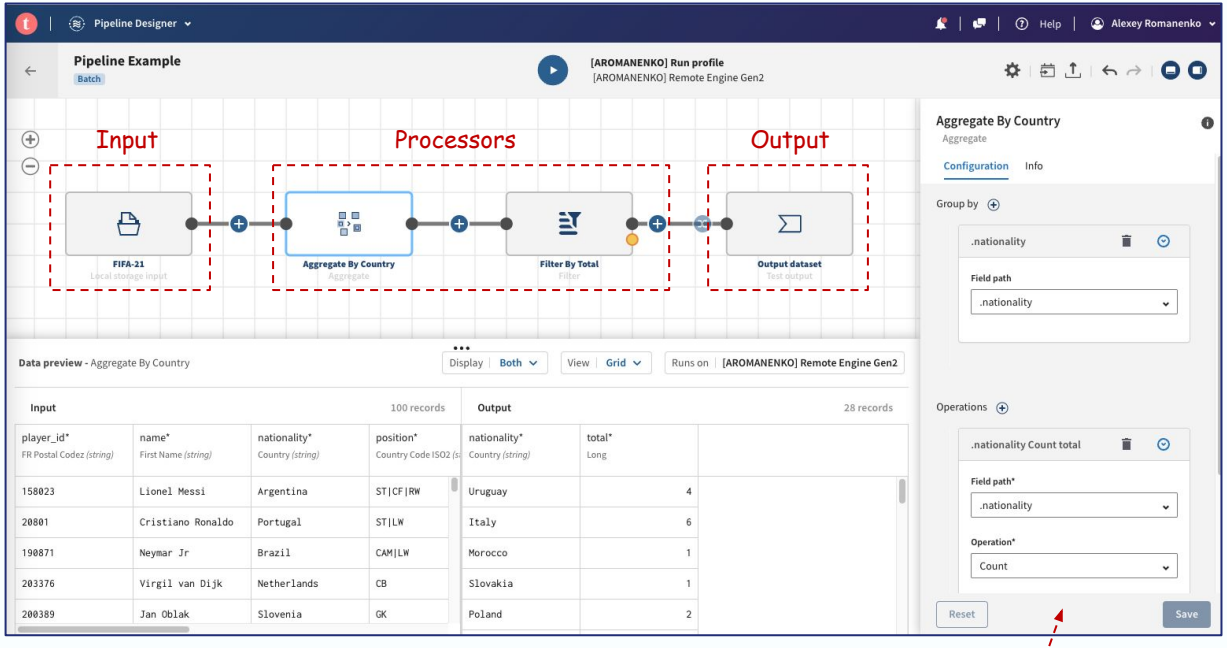
Talend Open Studio / Data Integration

- Talend Open Studio is a free open source ETL tool for Data Integration and Big Data;
- Eclipse based developer tool and job designer;
- Drag&drop components and connect them to create and run ETL/ELT jobs;
- No need to write a single line of code.



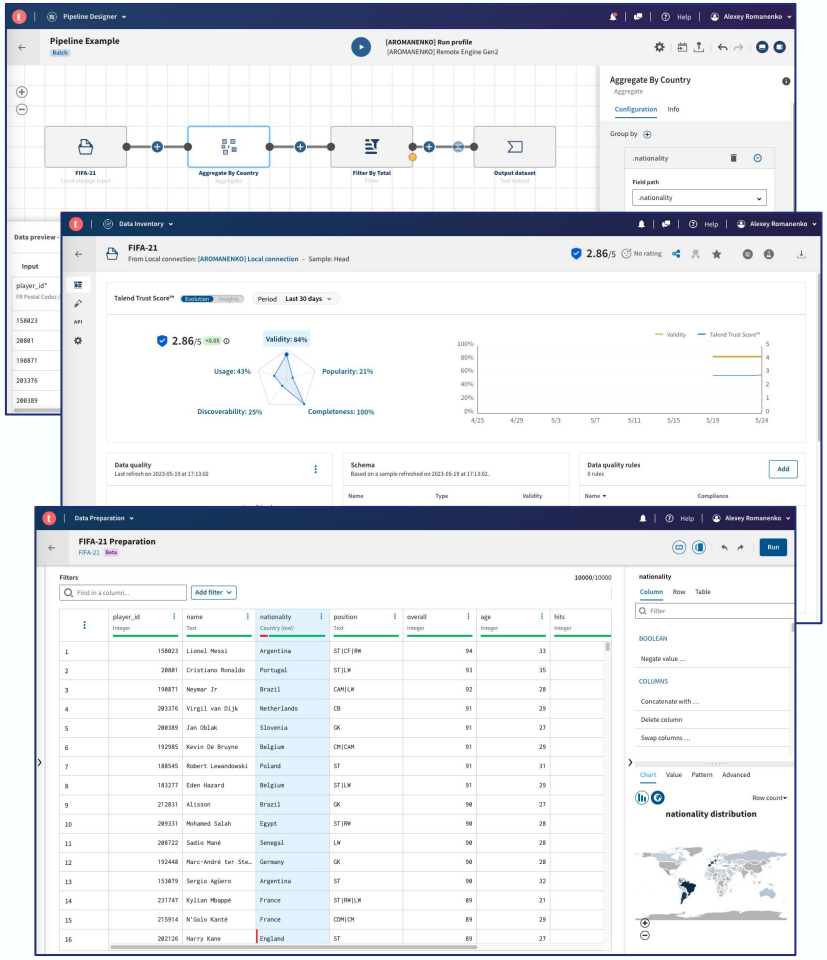
Talend Pipeline Designer (TPD)

- Modern flexible integration tool to process data in easy and powerful manner;
- Provides a graphical interactive Web UI to create complex pipelines;
- Live preview of data changes;
- Schema-based data collections;
- Batch & Streaming;
- Portable & Scalable;
- Uses Beam under the hood!



Using Beam at Talend

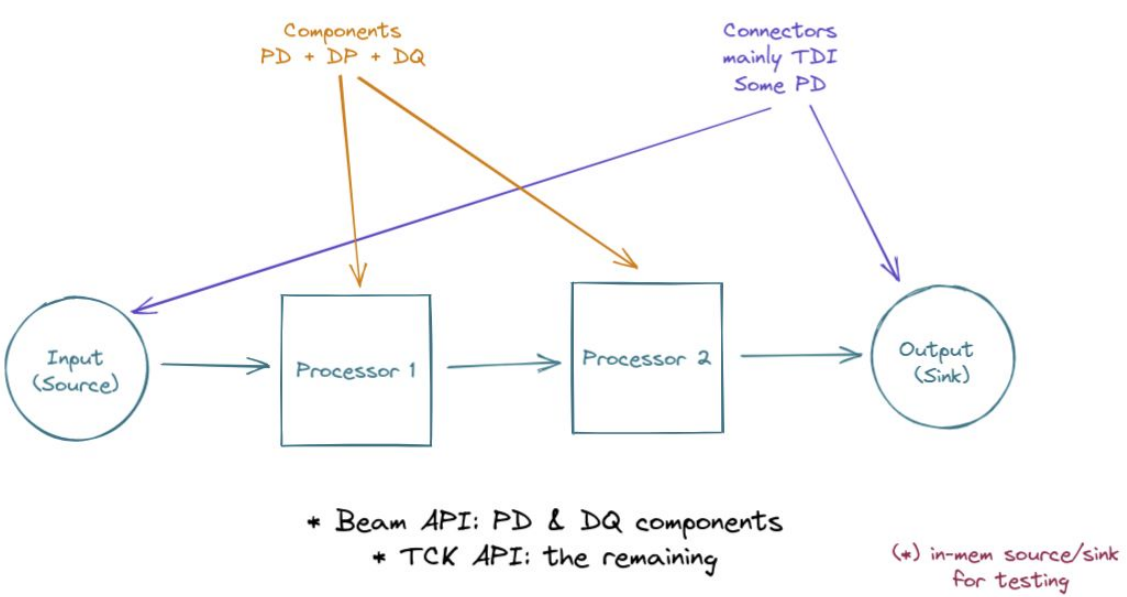
- Started to use Beam in 2016 as ASF Incubator project for **Talend DataStreams**, then **Talend Pipeline Designer** ;
- Talend Open Source team helped Beam to become a **top-level ASF project** ;
- Beam is used in the **Data Processing Platform** for several Talend products :
  - **Pipeline Designer** : Batch & Streaming pipelines
  - **Data Inventory** : Sampling sources
  - **Data Preparation** : Running data pre-processing jobs



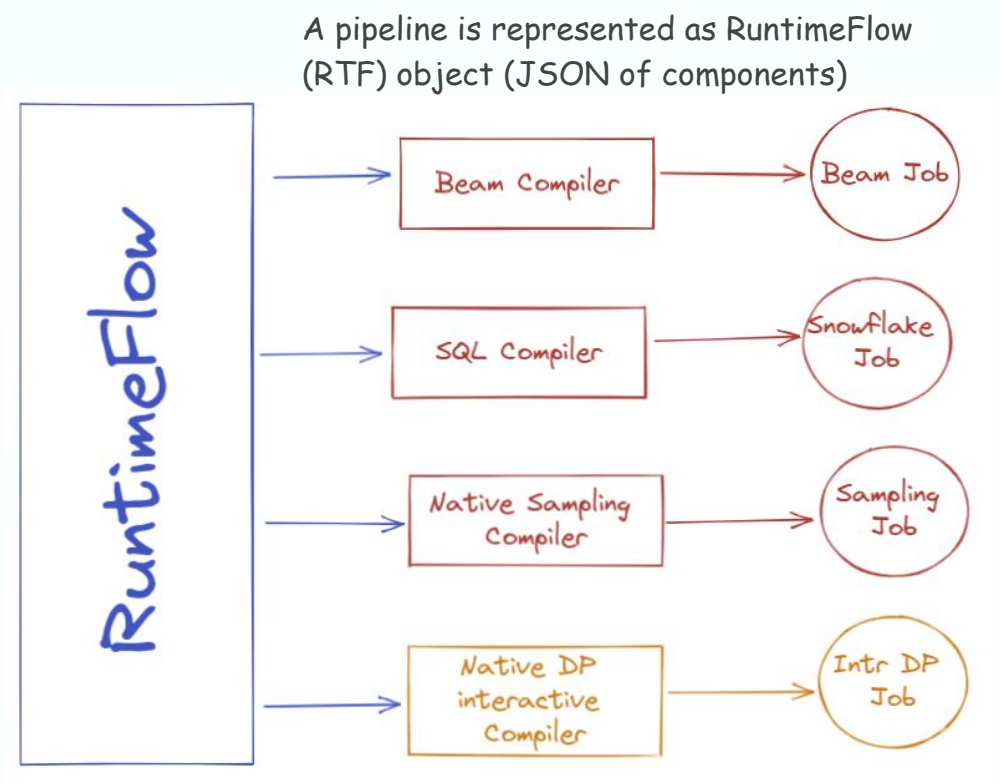
Engine Runtime: pipeline

Connectors and components:

- A pipeline is essentially a **DAG** of components:
  - IO components: a.k.a. **Connectors**.
  - Intermediate components: a.k.a. **Processors**.
- To be used in a pipeline, connector or component have to be either:
  - Beam-based: implement Beam API (e.g, PTransform for processors)
  - TCK-based: internal components framework



Engine Runtime: compiler

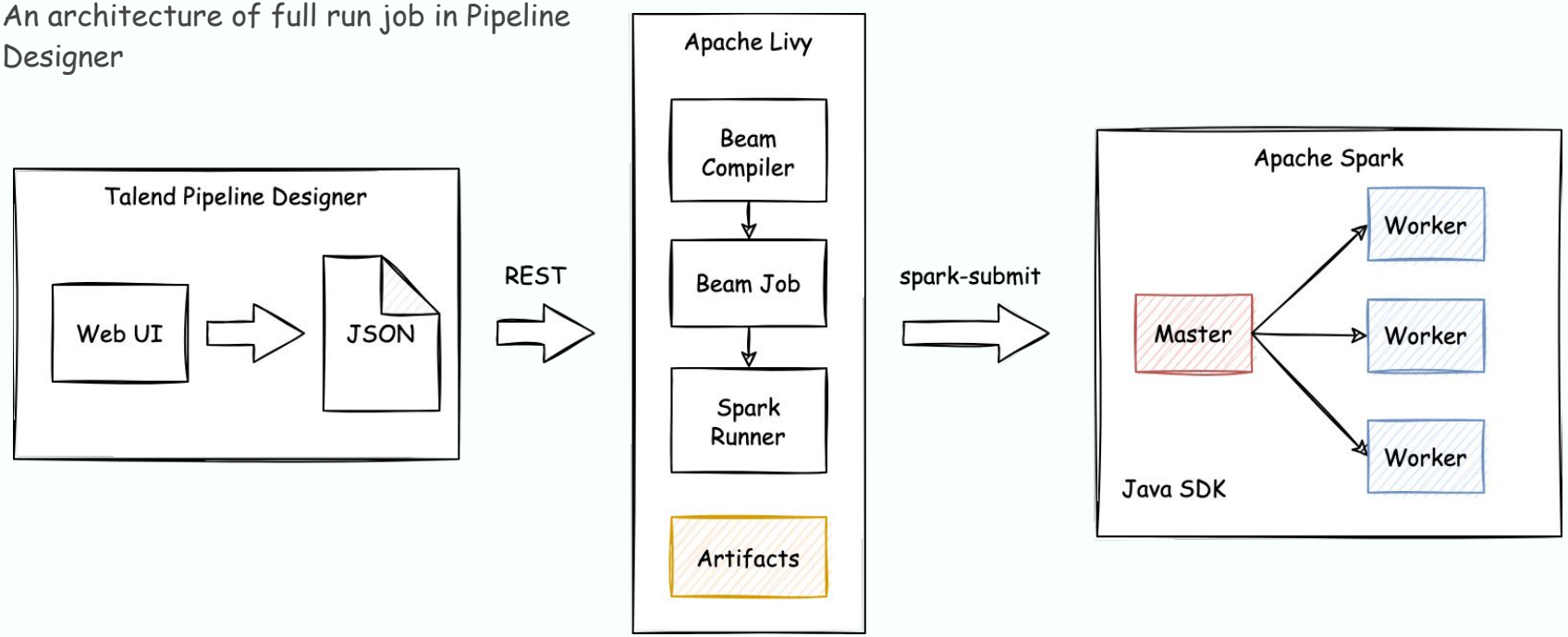


Beam Compiler (Translator):

- The first compiler that has been implemented;
- It translates an RTF to Beam pipeline;
- Then Beam pipeline is executed using either:
  - **SparkRunner** (Livy/FullRun job)
  - **FlinkRunner** (Interactive mode)
  - **DirectRunner** (Preview mode)

Full run Beam/Spark architecture

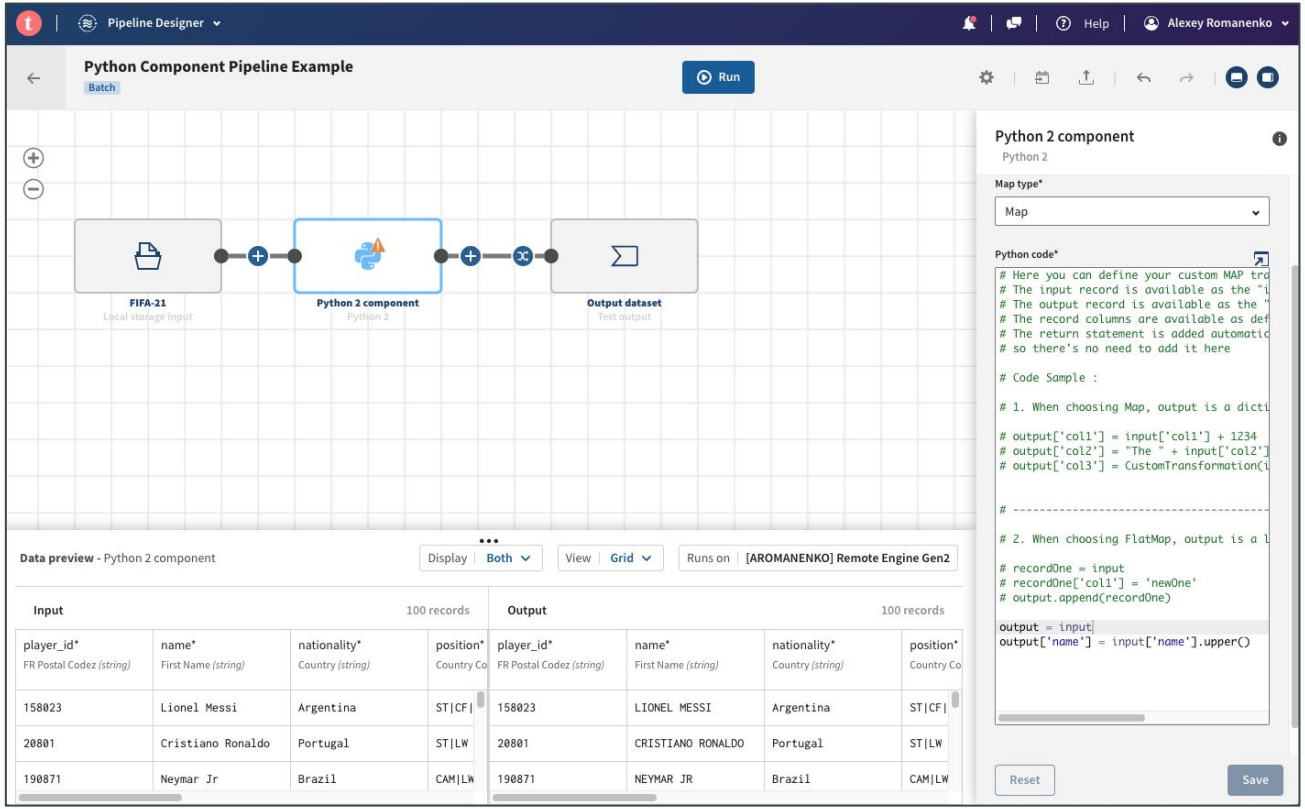
Example:
An architecture of full run job in Pipeline Designer



Use cases:
Python processor

Python processor

- TPD processor
- The Python processor executes user Python code to perform custom processing on user records.
- Originally, Python processor used Jython 2.7 as Python engine to process Python2 code



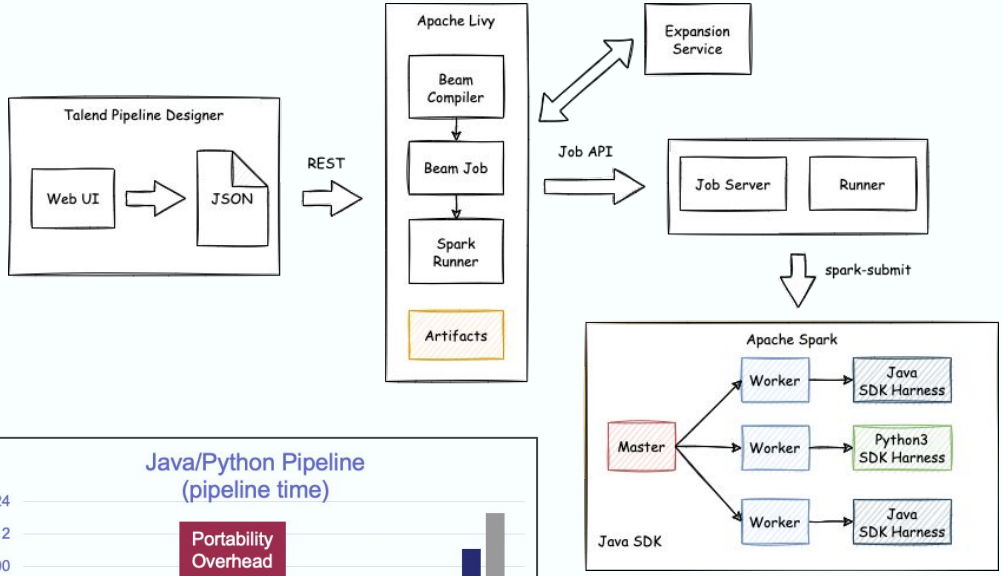
Problem:

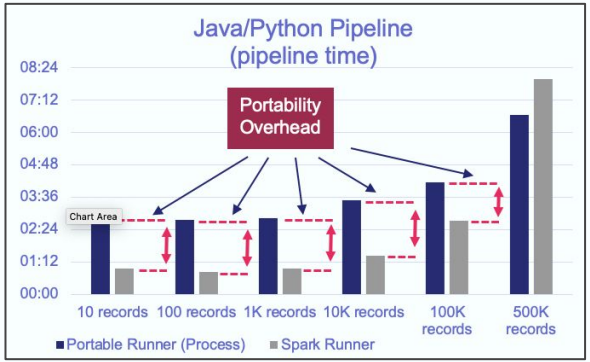
- Python 2.7 reached EOL on 12/31/2019
- Pipeline Designer Python processor used Jython 2.7 as Python engine
- Jython didn't support Python3, no plans to support it in the future
- No easy way to install 3rd-party Python libraries

Potential solutions:

- Beam portability framework:
  - Run Python 3 code as a Beam cross-language transform with Beam Portable Runner
  - See my talk *"Using Cross-Language pipeline to run Python 3 code with Java SDK"* at Beam Summit 2020
- Python-as-Service:
  - Use a custom Python server and dedicated PTransform to execute Python code
  - Thanks to **Ryan Scraba** (*@ryanskraba*) who worked on this

Cross-language Beam/Spark





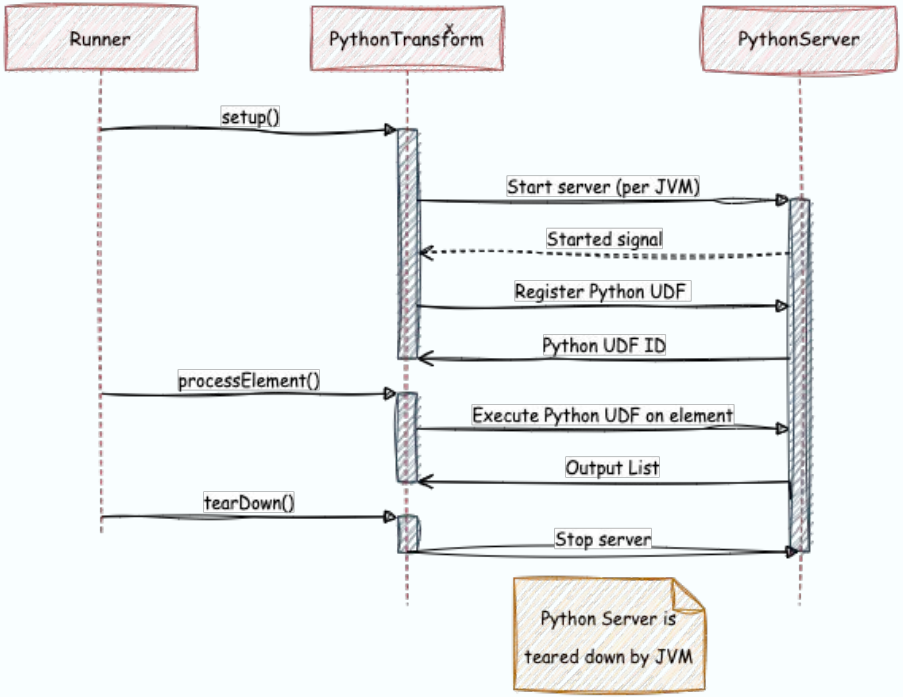
Advantages:

- Full support of Beam model and its features out-of-box;
- Tested and maintained by Beam community;
- Good performance for large data sets.

Drawbacks:

- Several times worse performance for small data;
- Required a complicated re-architecture of the TPD Runtime part
- High maintenance costs

Own Python server to execute python



Advantages:

- Simpler and configurable for our use case;
- No extra overhead/dependencies;
- Better performance for small data.

Drawbacks:

- Implementation/maintenance of the Python server;
- Only useful for specific use cases (no advanced Beam features - e.g. metrics, triggers, state, timers, etc);
- Requires a robust implementation of the Python server because of potential issues on startup/shutdown and resource leaks;
- Not tested/supported by a large community.

Use Cases:
Small Data Performance

Problem:

- One pipeline (DAG/schema) –> three sizes of input dataset
  - **Small dataset** (50-100 rows) for preview and interactive use;
  - **Average dataset** (~10K rows) for data sampling;
  - **Large dataset** (+10M rows) for full run pipeline.
- Fast (instant) results are critical for interactive mode
- Beam is supposed to run with large datasets and on distributed environments

Potential solutions:

- Use different runners for different use cases (current solution);
- Use native Java code compilation (PoC);
- Create Fast (In-Memory) runner for small/average datasets (PoC, WIP).

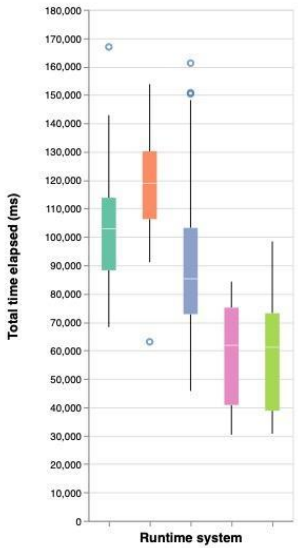
Native compilation

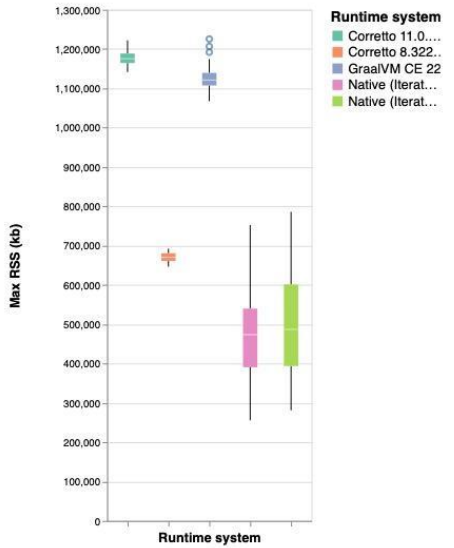
- Run a Beam pipeline (*MinimalWordCount*) locally as **GraalVM** native image
  - *GraalVM* is a high-performance JDK distribution designed to accelerate the execution of applications written in Java and other JVM languages along with support for a number of other popular languages.
- Use *DirectRunner* to simplify experiments
  - Other runners (*SparkRunner* & *FlinkRunner*) are in our ToDo list
- Our expectations:
  - Much lower memory usage for native images,
  - Faster startup times.

Native compilation

Benchmark results (*MinimalWordCount*):

- **Memory usage** improved ~ 29% (median) compared to the best performing JVM
- **Performance** also improved ~ 27% (median) compared to the best performing JVM.





Next steps:

- Run with more performance-oriented runners, like Spark/Flink or new *Fast Local* runner

More details at Moritz Mack's blog post:
https://github.com/mosche/blogposts/blob/main/beamnative/README.md

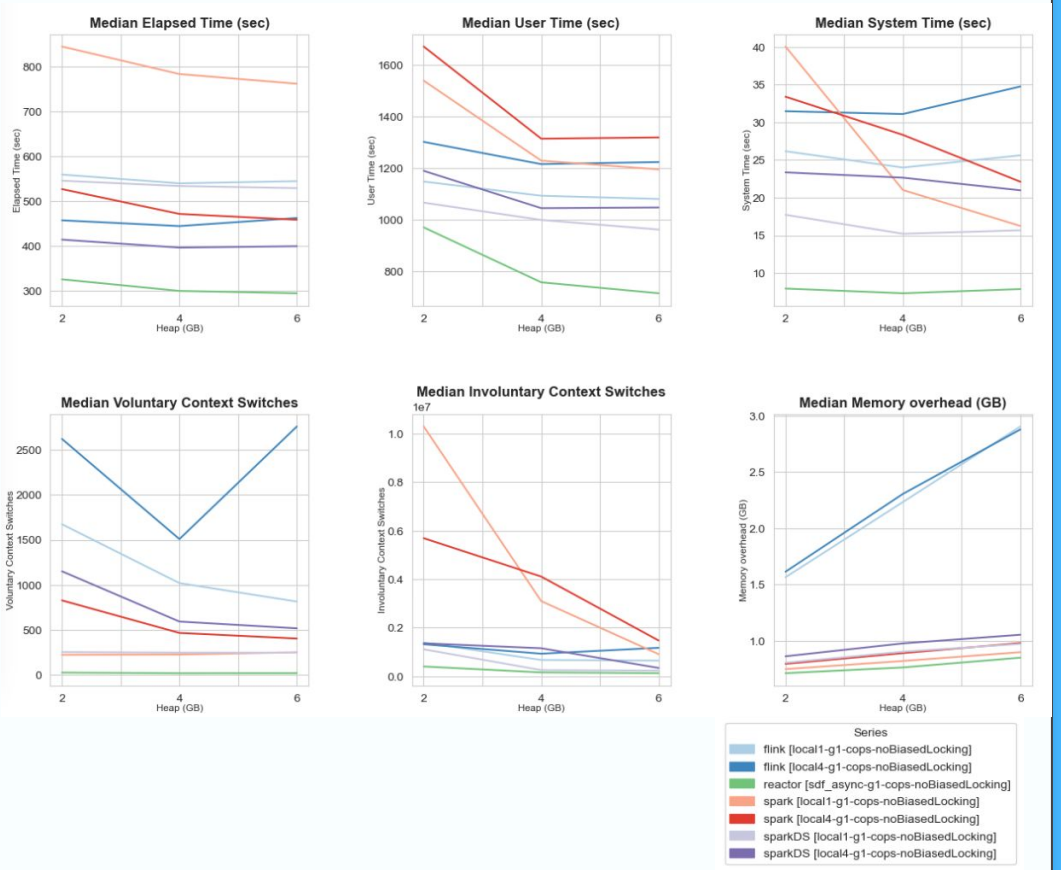
Fast Local Runner

- Develop a local in-memory Beam runner from scratch;
- Replace *DirectRunner*, *FlinkRunner* and *SparkRunner* used in local mode;
- Limited Beam model implementation (at least, for PoC):
  - Batch only
  - No state / timer support
  - Global Windows only
- Use Reactive Streams (Project Reactor)
  - One JVM, keep all data in memory
  - Map Java stream operations to Beam transforms
- PoC implemented by **Moritz Mack**, early stage:
  - WIP: https://github.com/mosche/beam/tree/reactor

Reactor is a fourth-generation reactive library, based on the Reactive Streams specification, for building non-blocking applications on the JVM

Intermediate results

- Used Beam TPC-DS benchmarks, 10 runs for every configuration;
- No *DirectRunner*, it constantly fails with OOM errors for the TPC-DS dataset of 1GB;
- Significant performance improvements with *ReactorRunner*;
- Next steps:
  - Add Windowing support
  - Run ValidateRunner tests
  - Add Streaming support
  - Contribute back to Beam



Talend contributions to Beam

Our Beam code contributions

- Java IO connectors
  - AWS, Hadoop, Kafka, Elasticsearch, Hbase, Jdbc, Avro, Parquet, ...
- Nexmark benchmark improvements
- TPC-DS benchmark integration
- Spark Runner
  - RDD runner improvements
  - Dataset runner from scratch
- Security fixes

Other contributions

- Releases testing
- PRs reviews
- Documentation updates
- Project mailing lists discussions
- Beam users support
- Blogging and talks at conferences
  - Beam Summit, ApacheCon, OpenSource Summit, etc

Some takeaways

- It's very important to contribute back to the OS project that is a key component of your product;
- Knowledge sharing saves time and money;
- Be part of project community;
- Sometimes it's challenging to find a balance between your specific and common users requests;
- Don't wait until someone do what you need - do it yourself!

Beam contributors at Talend

Many-many-many thanks to Talend all-time Beam contributors:

- **JB Onofré** (@jbonofre)
- **Ismaël Mejía** (@iemejia)
- **Etienne Chauchot** (@echauchot)
- **Daniel Kulp** (@dkulp)
- **Ryan Skraba** (@ryanskraba)
- **Colm O'Heigeartaigh** (@coheigea)
- **Moritz Mack** (@mosche)
- **Romain Manni-Bucau** (@rmannibucau)
- **Alexey Romanenko** (@aromanenko-dev)

Thank you!

QUESTIONS?

Twitter : @AlexRomDev
Github : @aromanenko-dev


BEAM
SUMMIT