Improving stability for running Python SDK with flink runner

Lydian Lee @ Affirm

BEAM SUMMIT

September 4-5, 2024

Sunnyvale, CA. USA

Lydian Lee

- Staff Engineer @ Affirm Streaming Infra Team
- My Journey
  - ML Engineer
  - ML Ops
  - Data Engineer (focus on streaming)
- Cat Lover
- Contact:
  - LinkedIn: https://www.linkedin.com/in/lydianlee/
  - Github: https://github.com/lydian

affirm by the numbers

As of FQ4'2024

18.6M

Active Consumers

14% Growth

93% of transactions from repeat users

24.7M

FQ4'24 Transactions

42% Growth

26% Increased transactions per active consumer

303K

Active Merchants

19% Growth

28% Growth in
Merchants with > $1k TTM GMV

Agenda

- Original Setup
- Stability Issues we were seeing
- What we discovered
- The Updated Architecture
- Other Improvements
- Summary

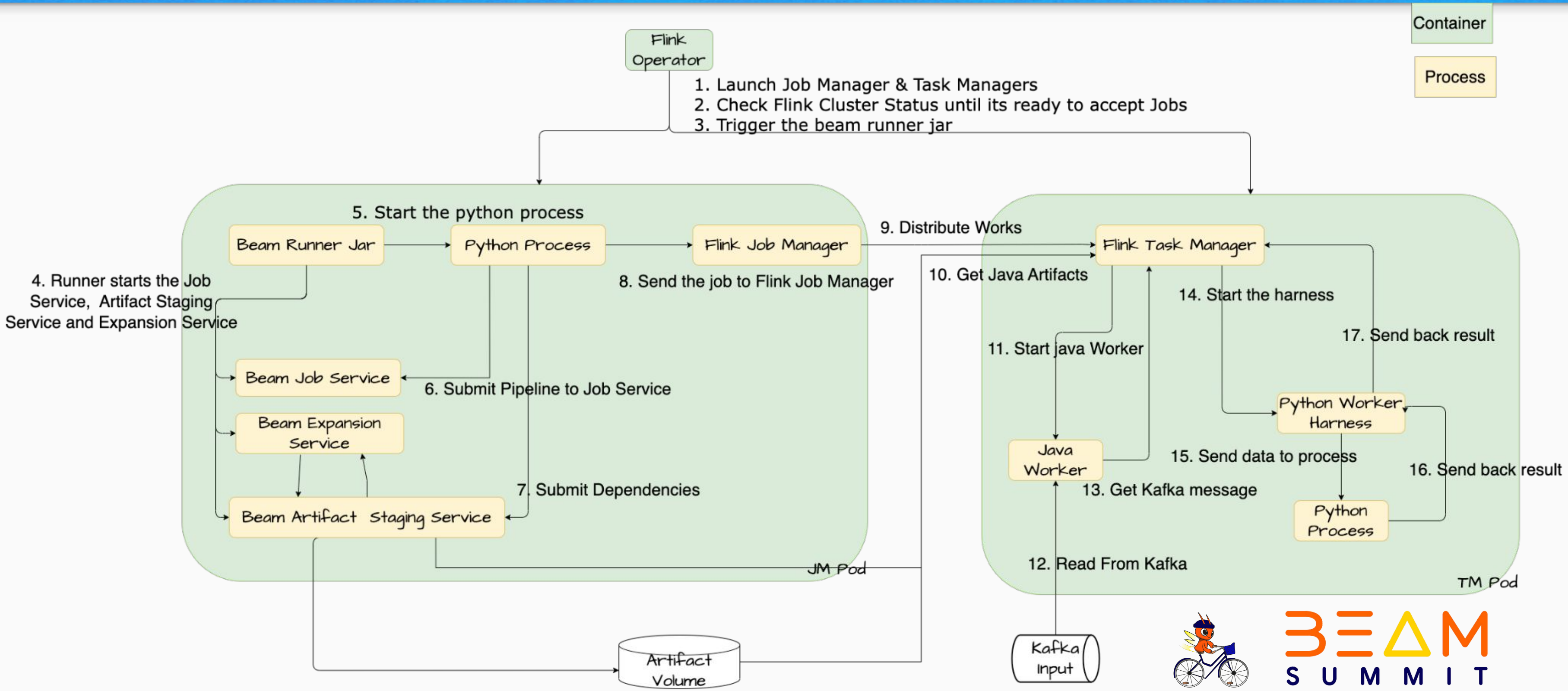
Affirm Original Beam/Flink Runner Setup

- Lyft Flink K8s Operator
  - Back in 2022, when we started the project, there was no Apache Flink Operator available
  - Lyft Operator is the default choice and was widely adopted in production

- We were running the Beam app with environment as “Process”
  - The Flink task manager & Python Harness runner was under the same container

Previous Talk - Beam Summit 2023
Running Beam Multi Language Pipeline on Flink Cluster on Kubernetes
https://youtu.be/XUz90LpGAgc?si=HhScgOJdzskKcae6

Affirm's Original Beam/Flink Runner Setup



1. Launch Job Manager & Task Managers
2. Check Flink Cluster Status until its ready to accept Jobs
3. Trigger the beam runner jar
4. Runner starts the Job Service, Artifact Staging Service and Expansion Service
5. Start the python process
6. Submit Pipeline to Job Service
7. Submit Dependencies
8. Send the job to Flink Job Manager
9. Distribute Works
10. Get Java Artifacts
11. Start java Worker
12. Read From Kafka
13. Get Kafka message
14. Start the harness
15. Send data to process
16. Send back result
17. Send back result

BEAM SUMMIT

What are the Status of Affirm’s Beam Usage

- We have total ~1000 Flink app runs in standalone mode
- The Lyft operator has issues and is no longer actively maintained:
  - The operator can only successfully update ~30 app states at a time, which makes our deployment have to constantly retry on the failed app to get the new code deployed
  - The operator also doesn’t have good support for High Availability, and leaves unused configmaps after the deployment
- We decided to switch to Apache Flink Operator

The issues

- After migration, we noticed more Out of Memory errors than before

```
Caused by: java.lang.RuntimeException: SDK Harness connection lost.
```

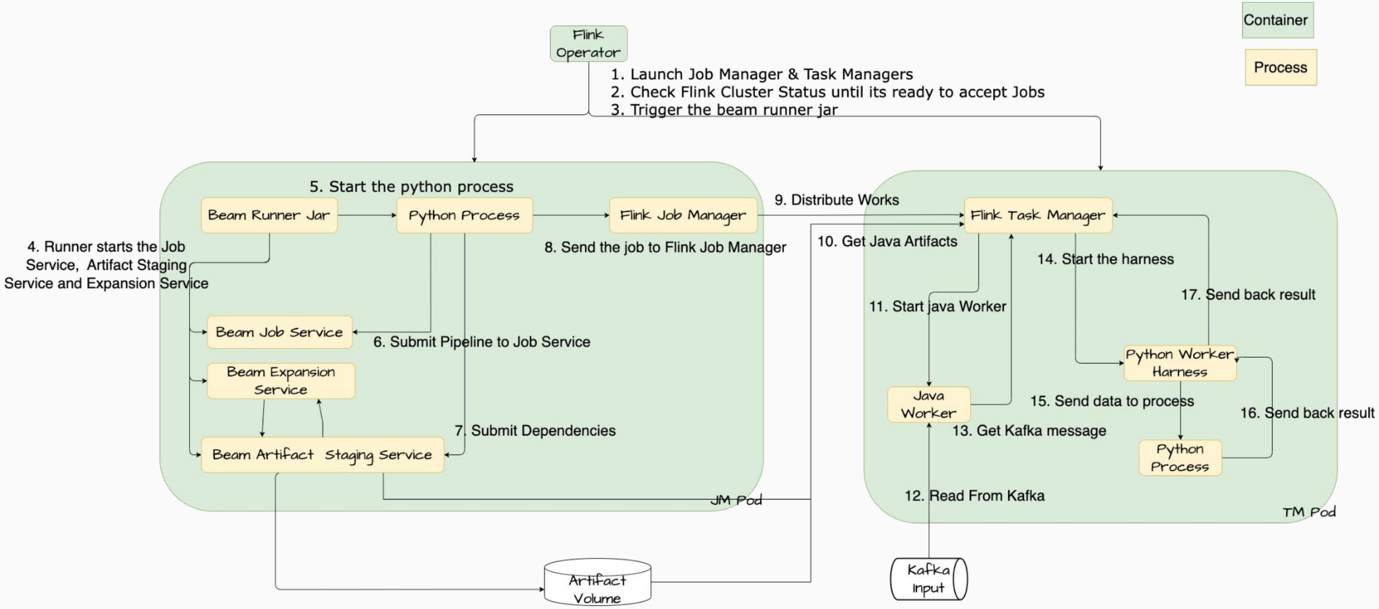
```
org.apache.flink.runtime.io.network.netty.exception.RemoteTransportException: Connection
unexpectedly closed by remote task manager '<ip> [ <task manager name>] '. This might indicate
that the remote task manager was lost
```

The Cause

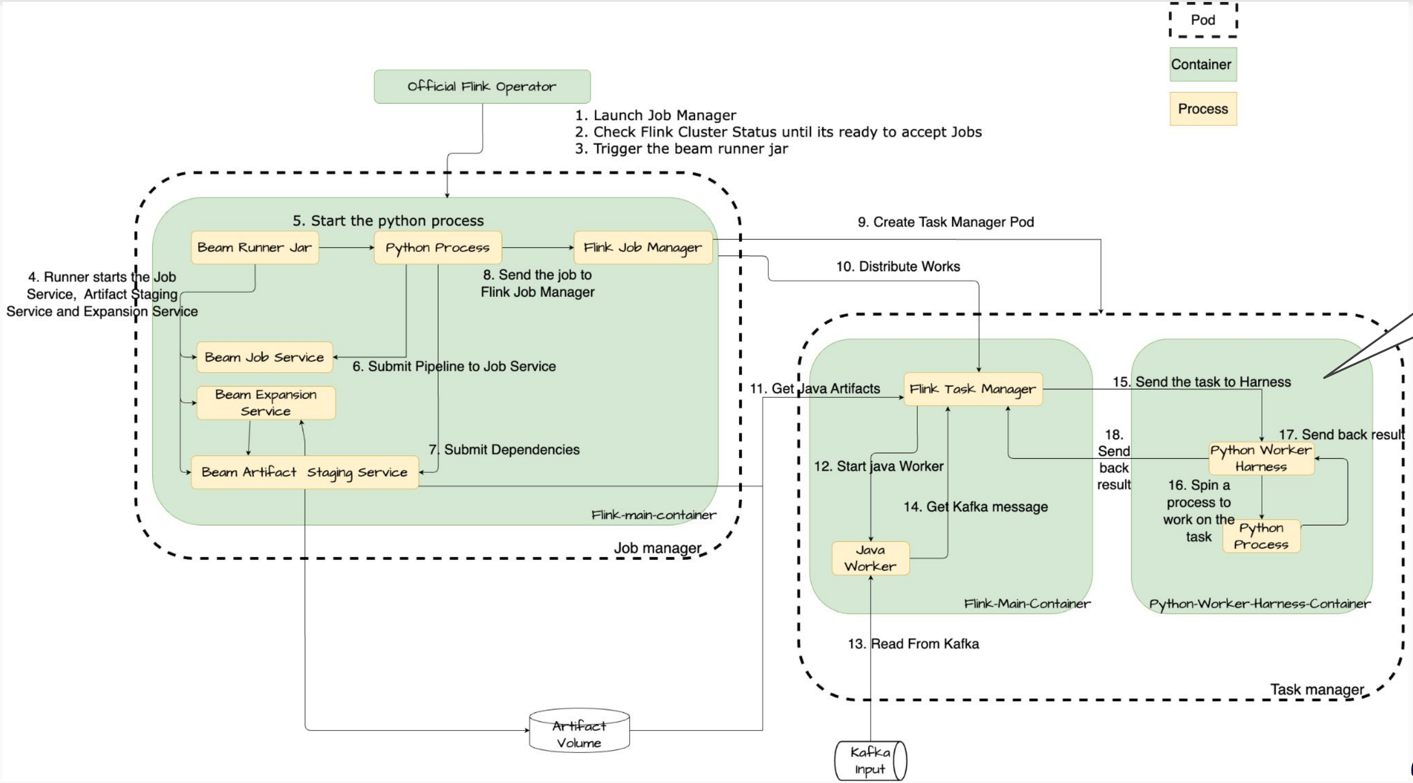
- Lyft Flink Operator
  - `systemMemoryFraction`: default 20%, which is reserved for non Java memory usage

- Apache Flink Operator
  - Almost all scheduled memory is reserved for Java usage
  - Barely any memory left for the python harness job to use

The Solution (Before)



The Solution (After)



1. Launch Job Manager
2. Check Flink Cluster Status until its ready to accept Jobs
3. Trigger the beam runner jar

4. Runner starts the Job Service, Artifact Staging Service and Expansion Service

5. Start the python process

6. Submit Pipeline to Job Service

7. Submit Dependencies

8. Send the job to Flink Job Manager

9. Create Task Manager Pod

10. Distribute Works

11. Get Java Artifacts

12. Start java Worker

13. Read From Kafka

14. Get Kafka message

15. Send the task to Harness

16. Spin a process to work on the task

17. Send back result

18. Send back result

Separate Harness to a different container

The Solution (Cont.)

- Separate Out the Harness container
  - The Apache Flink Operator allow us to configure `podTemplate` and launch sidecar container
  - Use `EXTERNAL` instead of `PROCESS`
- Benefits
  - We can be flexible on the resource usage, i.e. setting up `REQUESTS/LIMITS` for the python container
  - Much easier to monitor for the resource usage as it is a separate container

Launch Sidecar Container

```
args:
- /opt/apache/beam/boot
- --worker_pool
```

Beam Args

```
--runner=portableRunner
--environment_type=EXTERNAL
--environment_config=localhost:50000
```

Other Improvement We did

- Ensure the checkpoint is using s3p (presto) than s3a (Hadoop)
  - Based on our observation, the checkpointing time for one of the longest job is lower from 5 mins to 2 mins
- Ensure the Flink Task Manager to run in the same AZ to minimize the network communication cost
- Have a dedicated node pool in K8s to better control the resource
- Changed the instance type from generic to memory intensive and does save us some cost as well

Summary

- Different from `pyspark` / `pyflink`, Beam Python SDK is not a thin wrapper over the existing library, and thus we should ensure we assigned enough resource to the SDK harness process

- Using the `podTemplate` & the Apache Flink Operator to launch a sidecar container as well as using EXTERNAL environment for portable runner helps us achieve the goal.

Resources

- Demo Repo for the settings: https://github.com/lydian/beam-python-flink-runner-examples
  - docker-compose
  - k8s

Thank you!

Questions?

Contact

- LinkedIn: https://www.linkedin.com/in/lydianlee/
- Github: https://github.com/lydian