BEAM SUMMIT

How to write an IO for Beam

John Casey

BEAM
SUMMIT

Beam IO: CDAP and SparkReceiver IO Connectors Overview

Alex Kosolapov
& Elizaveta Lomteva

BEAM SUMMIT
NYC 2023

Agenda

- → Introduction
- → Developing an IO
- → CDAP IO Overview
- → Streaming Source IO - SparkReceiver
- → Testing IO
- → Akvelon Data Analytics and ML Accelerators demo

AKVELON

1200+ technology experts

23+ years of expertise

150+ clients

15 offices in 11 countries

24/7 operations support

beam

AKVELON

Google Cloud
Partner

Developing Beam IO (Java)

- Starting point: [Developing a new I/O connector]
- Design:
  - Define the input/output format
  - Read – Splittable DoFn (SDF), Write – ParDo
  - Determine target pipeline configuration parameters
- Develop:
  - DoFn to process an element
  - Read/Write PTransforms
- Test IO:
  - Unit testing, Integration, Performance testing
- Release: IO Documentation and examples

CDAP cdap.io

Google Cloud Data Fusion

An open-source platform for data applications in hybrid and multi-cloud environments

Visual point-and-click interface enabling code-free deployment of ETL/ELT data pipelines

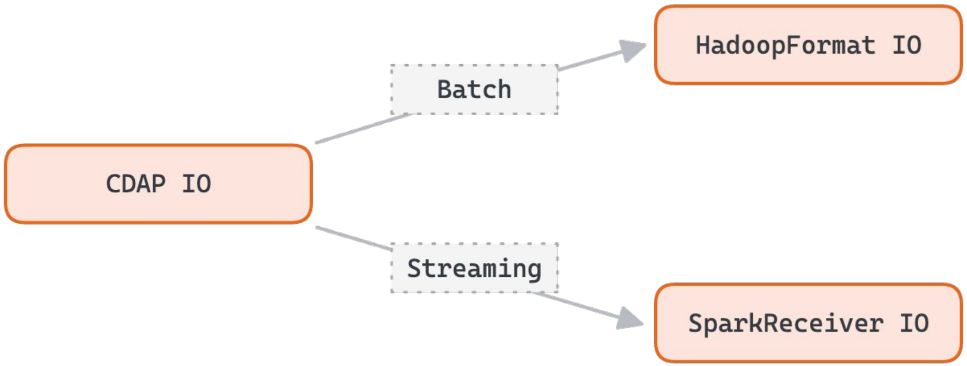
Ecosystem of plugins, including business applications connectors

CDAP IO

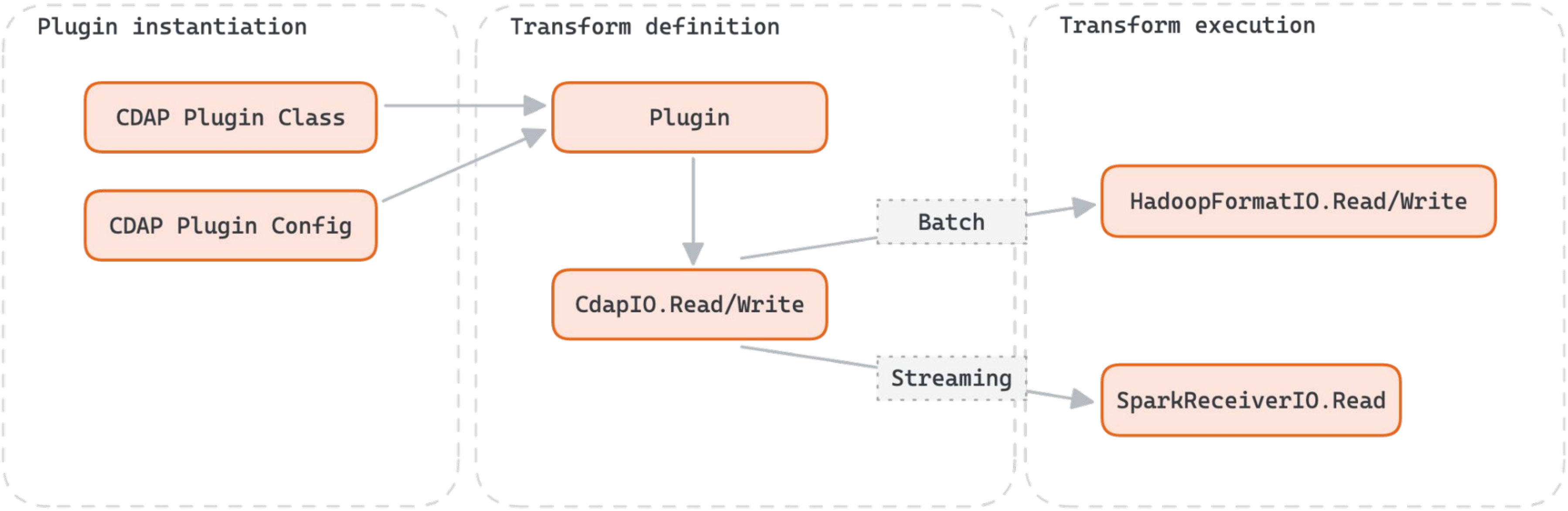
Provides transforms for reading and writing data via CDAP plugins

Connects Apache Beam with a variety of business applications like Salesforce, Hubspot, ServiceNow and Zendesk

Uses CDAP plugin definition



CDAP IO Workflow



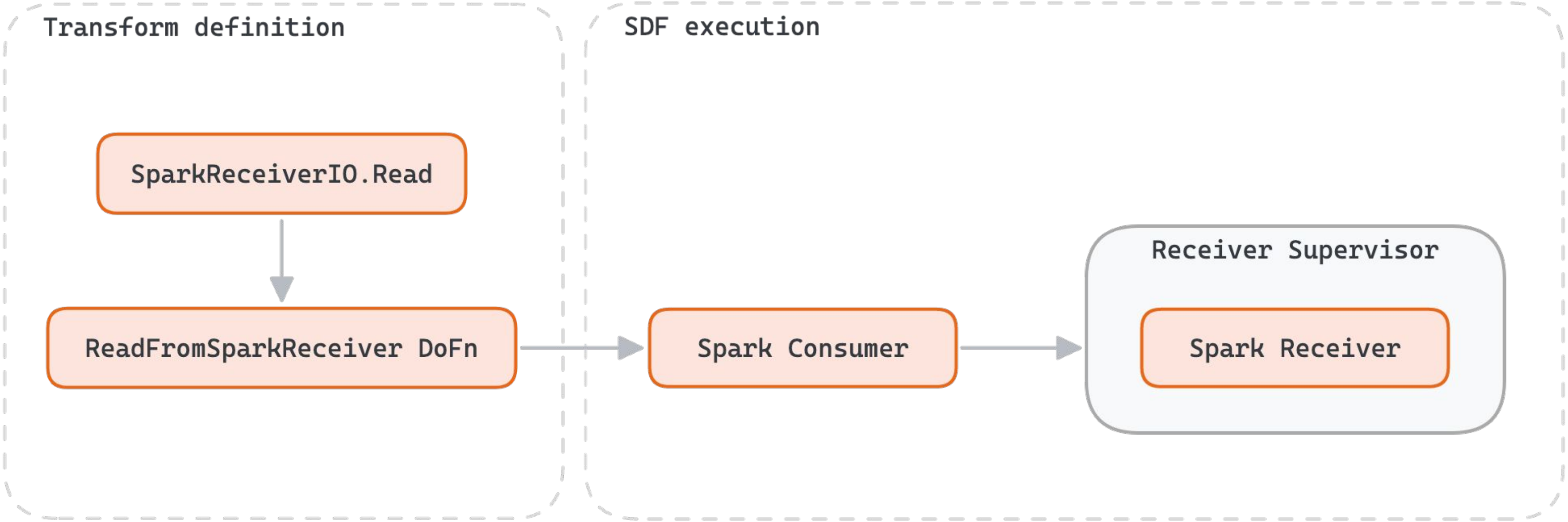
SparkReceiver IO

SparkReceiverIO provides transforms to read data via Apache Spark Receiver

Prerequisites:

- Spark Receiver provides HasOffset interface.
- Records have a numeric field that represents record offset.

SparkReceiver IO Workflow



Beam Parallelism & IO

Input parallelism - reading from bounded and unbounded sources, i.e. data source parallelism

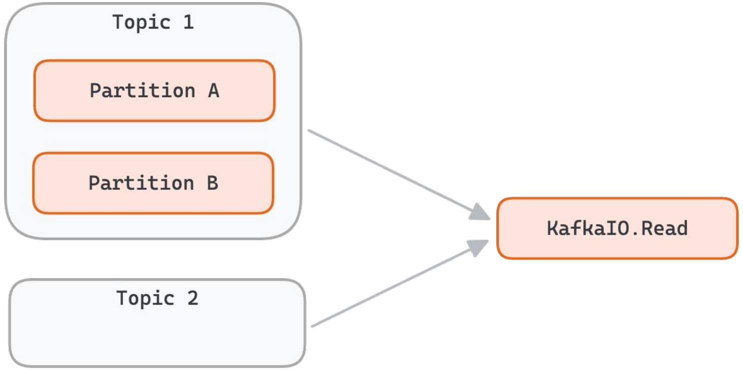
Inter-stage parallelism - splitting processing across workers, e.g. key-based data partitioning

Intra-stage parallelism - splitting element processing within transforms, e.g. Splittable DoFns, bundle processing

Data Source Parallelism

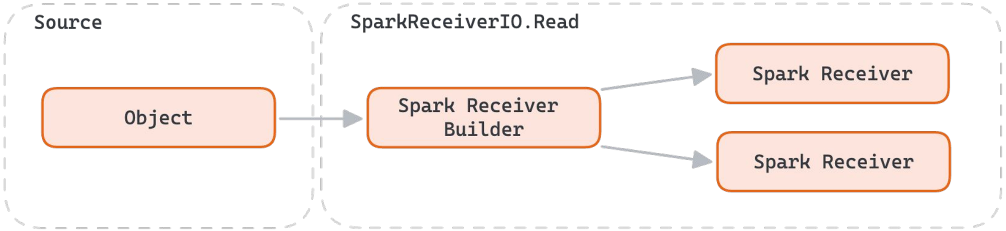
Refers to the parallelism achieved by reading data from multiple sources or partitions of a single source concurrently.

(E.g. Kafka topic partitions)



SparkReceiverIO

Each receiver builder can be associated with single source object and create multiple receivers during processing



Inter-stage parallelism

Refers to the parallelism between different transforms (or stages) within a Beam pipeline.

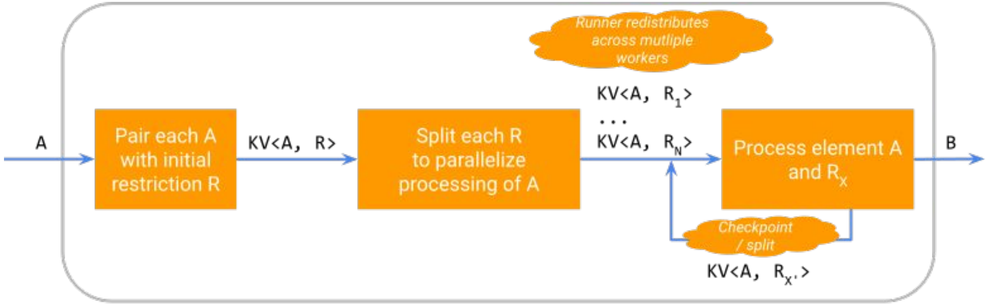
Achieved by runner implementation

(E.g. key-based operations in Beam)

SparkReceiverIO

Achieved by supported runners – Direct runner and
Dataflow runner v1 and v2

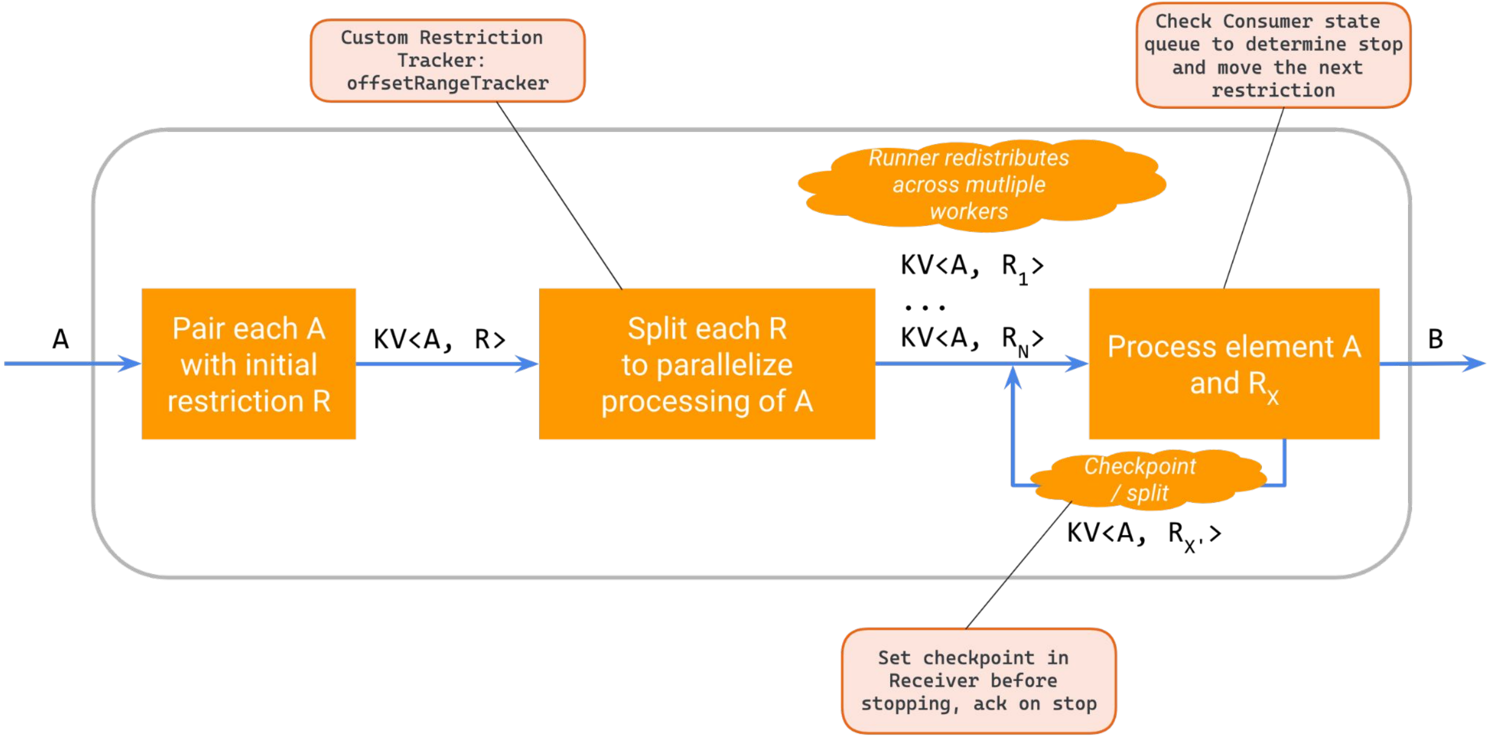
Intra-stage: Splittable DoFn (SDF)

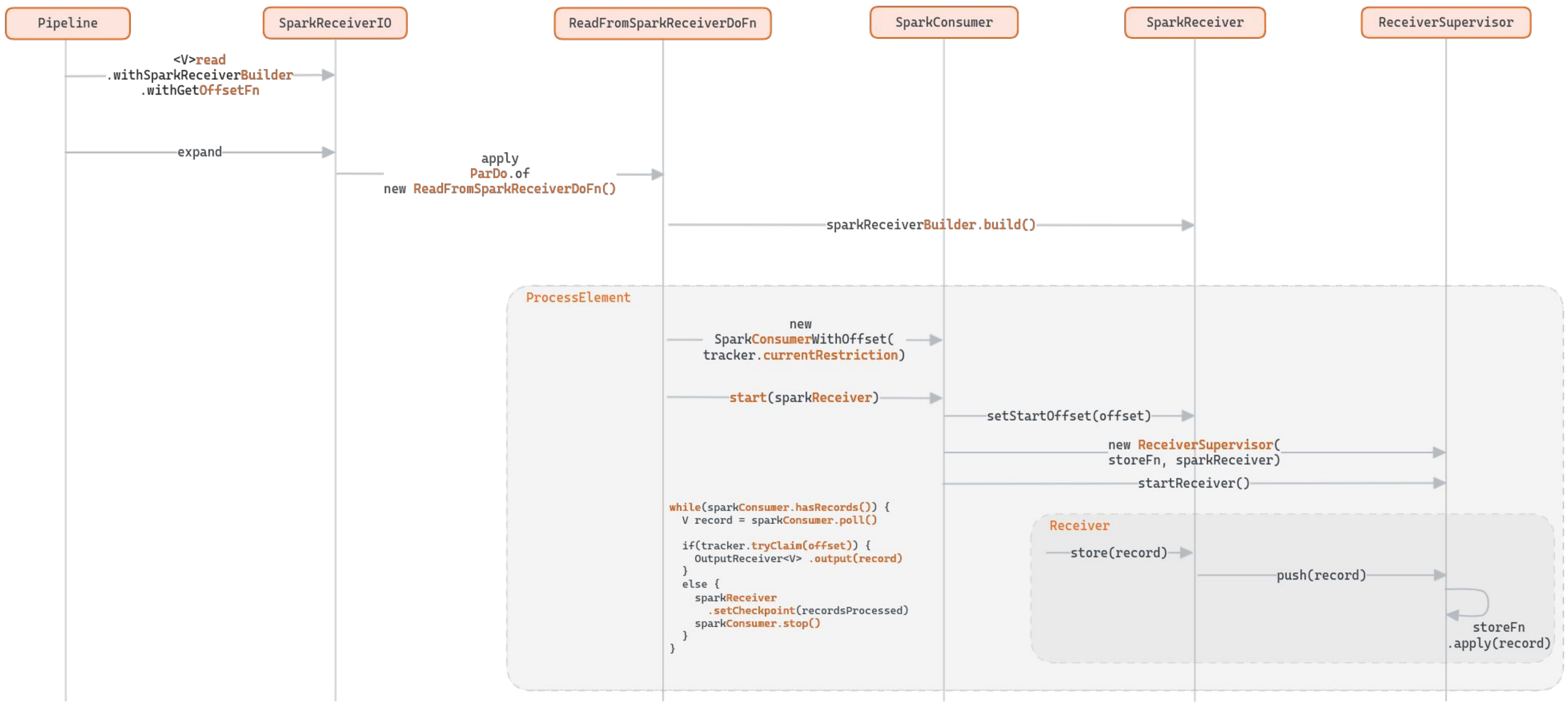


Executing an SDF follows the following steps:

1. Each **element** is paired with a **restriction** (e.g. filename is paired with offset range representing the whole file).
2. Each element and restriction pair is **split** (e.g. **offset** ranges are broken up into smaller pieces).
3. The runner redistributes the element and restriction pairs to several workers.
4. Element and restriction pairs are processed in parallel (e.g. the file is read). Within this last step, the element and restriction pair can pause its own processing and/or be split into further element and restriction pairs.

SparkReceiverIO




Pipeline
SparkReceiverIO
ReadFromSparkReceiverDoFn
SparkConsumer
SparkReceiver
ReceiverSupervisor
<V>read
.withSparkReceiverBuilder
.withGetOffsetFn
expand
apply
ParDo.of
new ReadFromSparkReceiverDoFn()
sparkReceiverBuilder.build()
ProcessElement
new
SparkConsumerWithOffset(
tracker.currentRestriction)
start(sparkReceiver)
setStartOffset(offset)
new ReceiverSupervisor(
storeFn, sparkReceiver)
startReceiver()
while(sparkConsumer.hasRecords()) {
V record = sparkConsumer.poll()
if(tracker.tryClaim(offset)) {
OutputReceiver<V> .output(record)
}
else {
sparkReceiver
.setCheckpoint(recordsProcessed)
sparkConsumer.stop()
}
}
Receiver
store(record)
push(record)
storeFn
.apply(record)

Agenda

- Introduction
- Developing an IO
- CDAP IO Overview
- Streaming Source IO - SparkReceiver
- **Testing IO**
- Akvelon Data Analytics and ML Accelerators demo

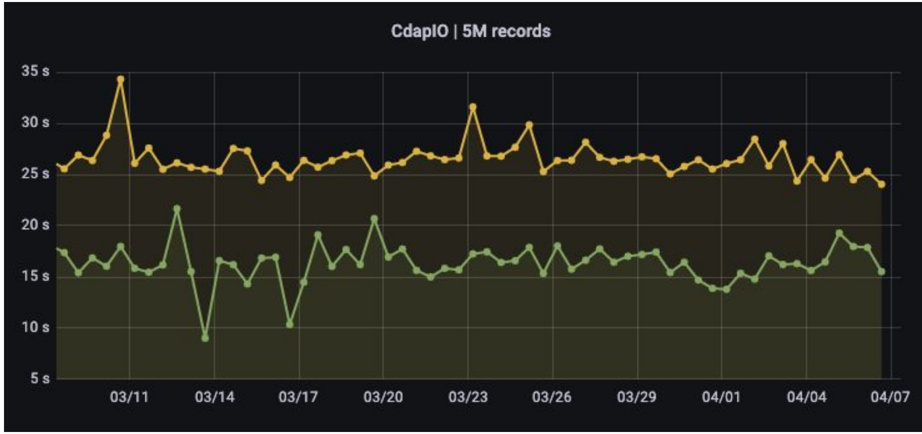
Testing IO and Release

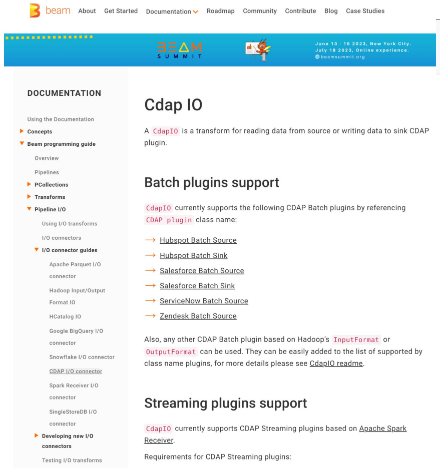
IO Testing

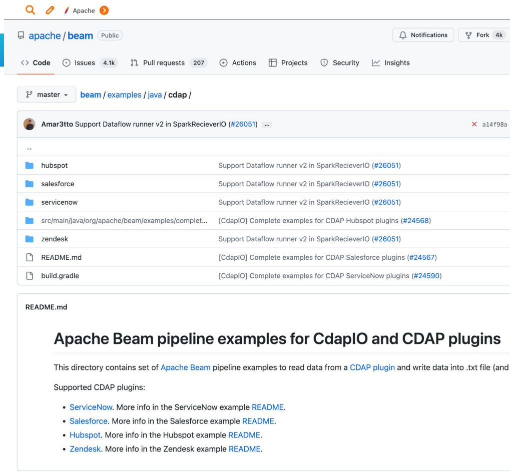
- testing guide, IO transforms testing
- Unit, integration and performance test
- Created RabbitMQ SparkReceiver on-demand source in Apache Beam that generates streaming data according to provided profile

Release

- Beam website IO Connectors
- Documentation & Readmes
- Complete examples







Demo

Data and Analytics Accelerators

https://github.com/akvelon/DnA_accelerators

https://github.com/akvelon/DnA_accelerators/

README.md

Akvelon Data and Analytics Accelerators

Akvelon is a digital product and software engineering company that empowers strategic advantage and accelerates your path to value in Data and Analytics, AI/ML, MLOps, Application development, and more with innovation and predictable delivery. Akvelon is providing this collection of accelerators as a reference and easy customizations for developers looking to build data, machine learning, and visualizations.

- Get in touch about Data and Analytics and Data Migrations projects.
- Get in touch about ML projects.
- Get in touch about Google Cloud projects.

Learn more about all our ML and software engineering services at our website akvelon.com.

Accelerators

ML, Streaming and Batch Data Processing

Apache Beam and Google Cloud Dataflow

Apache Beam provide unified streaming and batch processing to power ML and streaming analytics use cases. Google Cloud Dataflow is a managed to run Apache Beam in coud with minimal latency and costs, and integrations with other Google Cloud products like Vertex AI and Tensorflow TFX. Akvelon, a Google Cloud Service Partner, and an active Apache Beam contributor and Beam Summit partner, presents several of our favorite accelerators for Dataflow.

- Salesforce to Txt - Flex templates for batch and streaming Salesforce data processing with Google Cloud Dataflow, using Apache Beam CDAP IO.
- Salesforce to BigQuery - Flex templates for batch and streaming Salesforce data processing with Google Cloud Dataflow and BigQuery, using Apache Beam CDAP IO. Flex templates provide a comprehensive example of using Machine Learning (ML) to process streaming data in Dataflow, using Java multilanguage pipeline with Python transforms to run custom TFX and PyTorch ML models. This complete Flex template example also demonstrates creating and setting up Expansion Service in Dataflow to enable running custom Python transforms within a Java pipeline.
- Tensorflow TFX model training with Apache Beam - a Python notebook and Python Beam pipeline that demonstrates both Jupyter notebook to train a Tensorflow TFX ML model and the converted Python pipeline ready for Expansion Service use
- PyTorch ML model training and Expansion Service for multilanguage pipelines with Apache Beam - a complete example to train a PyTorch ML model using Apache Beam, convert the notebook to the Python pipeline, create custom Python Transforms and deploy as Apache Beam Expansion Service for Google Cloud Dataflow.

Custom Visualizations

Akvelon has accumulated vast experience with data analytics, custom visualizations, dashboards, and reports for a wide range of industries and use cases. Here are some of our favorite visualization accelerators.

Looker Visuals

github.com/akvelon/DnA_accelerators/tree/main/dataflow

main DnA_accelerators / dataflow /

Google Cloud Dataflow Accelerators

Apache Beam provides unified streaming and batch processing to power ML and streaming analytics use cases. Google Cloud Dataflow is managed to run Apache Beam in the cloud with minimal latency and costs, and integrates with other Google Cloud products like Vertex AI and Tensorflow TFX. Akvelon, a Google Cloud Service Partner, and an active Apache Beam contributor and Beam Summit partner, presents several of our favorite accelerators for Dataflow.

Akvelon, a Google Cloud Partner, is providing this open-source collection of Dataflow Flex templates as a reference and easy customizations for developers looking to build streaming, batch, multilanguage data pipelines with ML processing in Google Cloud Dataflow.

Flex Templates for Google Cloud Dataflow

Google Cloud Dataflow Flex Templates are a powerful way to build and run data pipelines on Google Cloud Platform. With Flex Templates, you can package your pipeline code and dependencies as a Docker image, and then run it on Dataflow with just a few clicks. This makes it easy to build and deploy complex pipelines quickly and reliably.

- Salesforce to Txt - Flex templates for batch and streaming Salesforce data processing with Google Cloud Dataflow, usin CDAP IO.
- Salesforce to BigQuery - Flex templates for batch and streaming Salesforce data processing with Google Cloud Dataflov using Apache Beam CDAP IO. Flex templates provide a comprehensive example of using Machine Learning (ML) to pro data in Dataflow, using Java multilanguage pipeline with Python transforms to run custom TFX and PyTorch ML models. Flex template example also demonstrates creating and setting up Expansion Service in Dataflow to enable running cust transforms within a Java pipeline.

Machine Learning with Google Cloud Dataflow

- Tensorflow TFX model training with Apache Beam - a Python notebook and Python Beam pipeline that demonstrate bot

Summary

AKVELON

Developing Beam IOs

Machine Learning

Multilanguage pipelines

https://github.com/akvelon/DnA_accelerators

https://github.com/akvelon/DnA_accelerators/

README.md

Akvelon Data and Analytics Accelerators

Akvelon is a digital product and software engineering company that empowers strategic advantage and accelerates your path to value in Data and Analytics, AI/ML, MLOps, Application development, and more with innovation and predictable delivery. Akvelon is providing this collection of accelerators as a reference and easy customizations for developers looking to build data, machine learning, and visualizations.

- Get in touch about Data and Analytics and Data Migrations projects.
- Get in touch about ML projects.
- Get in touch about Google Cloud projects.

Learn more about all our ML and software engineering services at our website akvelon.com.

Accelerators

ML, Streaming and Batch Data Processing

Apache Beam and Google Cloud Dataflow

Apache Beam provide unified streaming and batch processing to power ML and streaming analytics use cases. Google Cloud Dataflow is a managed to run Apache Beam in coud with minimal latency and costs, and integrations with other Google Cloud products like Vertex AI and Tensorflow TFX. Akvelon, a Google Cloud Service Partner, and an active Apache Beam contributor and Beam Summit partner, presents several of our favorite accelerators for Dataflow.

- Salesforce to Txt - Flex templates for batch and streaming Salesforce data processing with Google Cloud Dataflow, using Apache Beam CDAP IO.
- Salesforce to BigQuery - Flex templates for batch and streaming Salesforce data processing with Google Cloud Dataflow and BigQuery, using Apache Beam CDAP IO. Flex templates provide a comprehensive example of using Machine Learning (ML) to process streaming data in Dataflow, using Java multilanguage pipeline with Python transforms to run custom TFX and PyTorch ML models. This complete Flex template example also demonstrates creating and setting up Expansion Service in Dataflow to enable running custom Python transforms within a Java pipeline.
- Tensorflow TFX model training with Apache Beam - a Python notebook and Python Beam pipeline that demonstrates both Jupyter notebook to train a Tensorflow TFX ML model and the converted Python pipeline ready for Expansion Service use
- PyTorch ML model training and Expansion Service for multilanguage pipelines with Apache Beam - a complete example to train a PyTorch ML model using Apache Beam, convert the notebook to the Python pipeline, create custom Python Transforms and deploy as Apache Beam Expansion Service for Google Cloud Dataflow.

Custom Visualizations

Akvelon has accumulated vast experience with data analytics, custom visualizations, dashboards, and reports for a wide range of industries and use cases. Here are some of our favorite visualization accelerators.

Looker Visuals

AKVELON

https://github.com/akvelon/DnA_accelerators

https://akvelon.com

Questions?

https://www.linkedin.com/in/akosolapov
https://www.linkedin.com/in/elizaveta-lomteva

BEAM SUMMIT

BEAM

SUMMIT

Meeting Security Requirements for Apache Beam Pipelines on Google Cloud

Lorenzo Caggioni
Google
linkedin.com/in/lcaggio/

BEAM SUMMIT
NYC 2023

Agenda

Securing a Beam Pipelines on Google Cloud

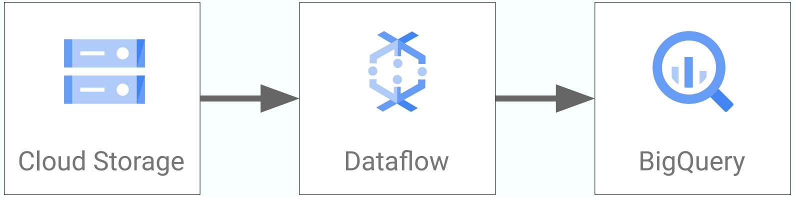
- Private resources
- Role separation and least privileges
- Data Encryption at rest

Customer requirements

1. Internal addressment of tenants must be private.

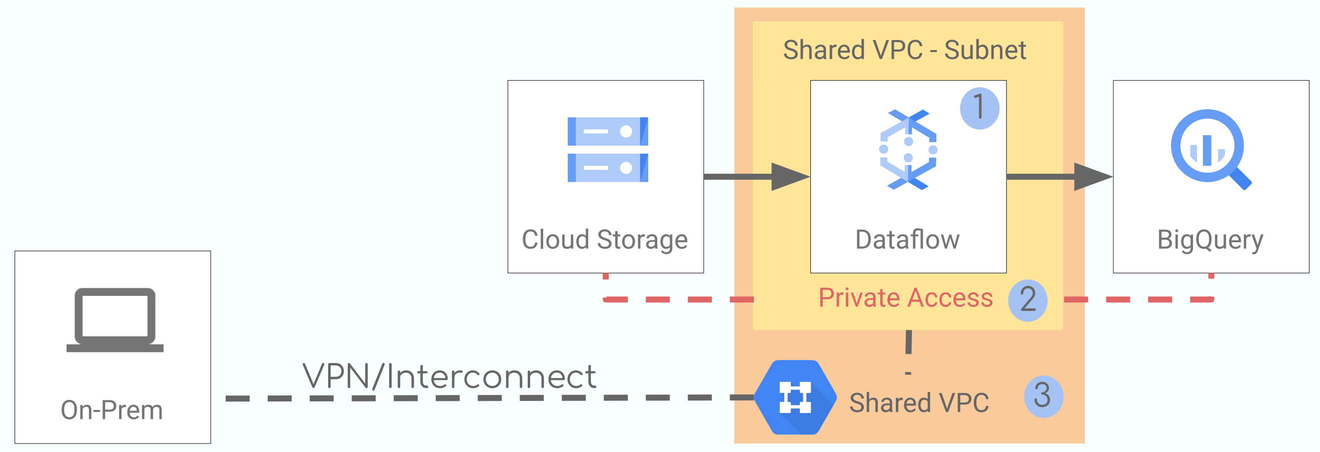
2. Every tenants must be isolated and dedicated to a specific system of services.

3. All data must have encryption at-rest with keys managed by ACME's security team.



1. Internal addressment of tenants must be private.

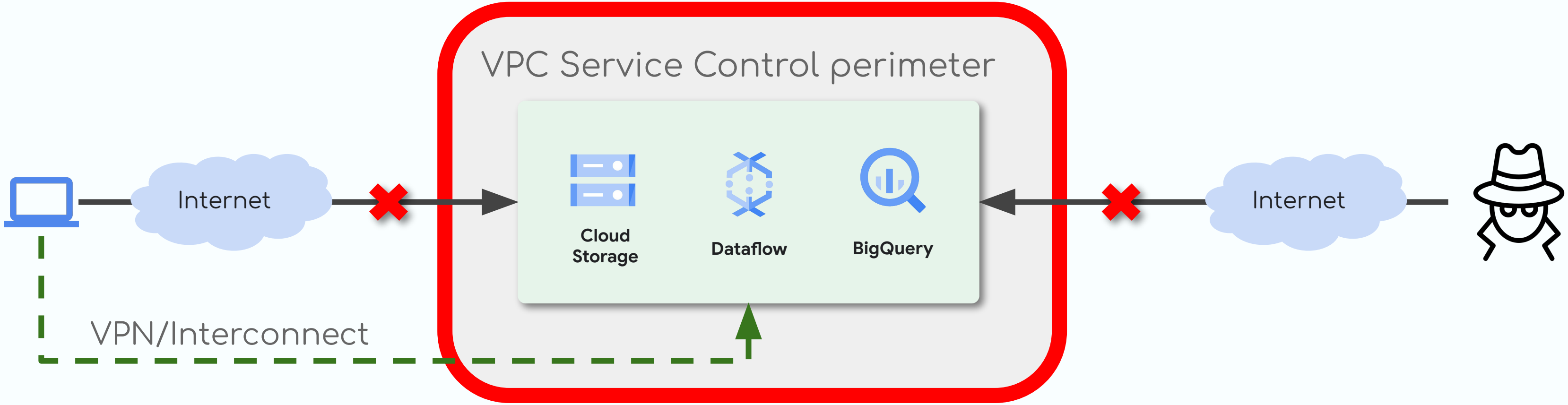
1. Set `disable-public-ips` when deploying the pipeline
2. Enable `Private Access` on the subnet to access GoogleAPIs
3. Network: shared-VPC



1. Mitigate Data Exfiltration

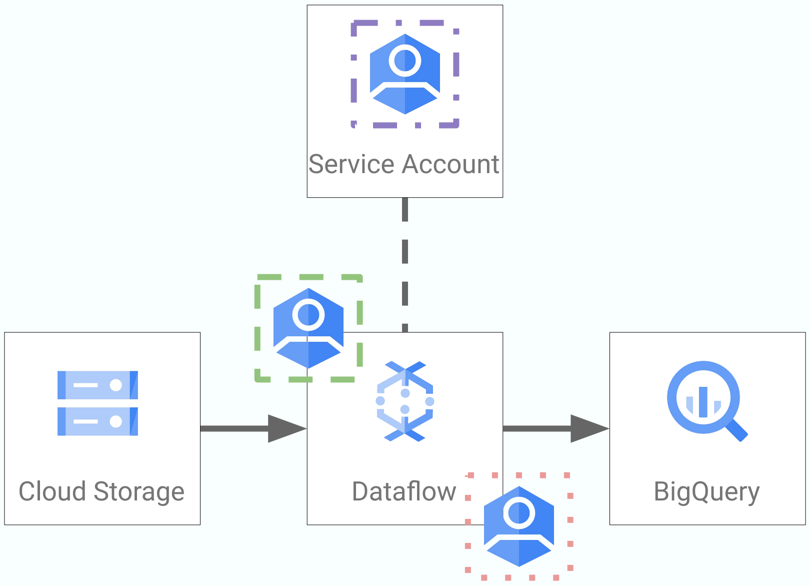
VPC Service Controls helps preventing **data exfiltration** and controlling access to **Google APIs**.

Isolate resources of **multi-tenant Google Cloud** services to mitigate data exfiltration risks.



2. Tenants must be isolated

IAM and Service Accounts



Dataflow Service Agent

```
roles/dataflow.serviceAgent
roles/compute.networkUser
```

Worker Service Account

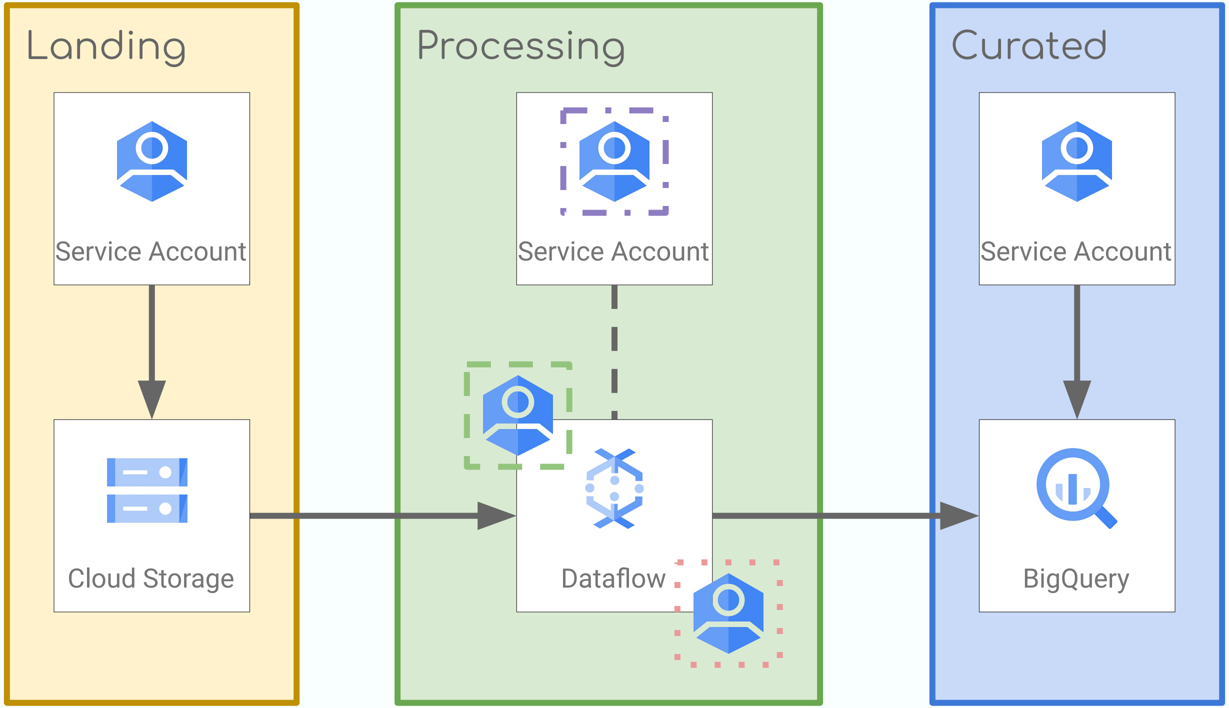
```
roles/storage.objectAdmin
roles/dataflow.worker
roles/bigquery.dataEditor
```

Job orchestrator

```
role/iam.serviceAccountUser
role/dataflow.admin
```

2. Tenants must be isolated

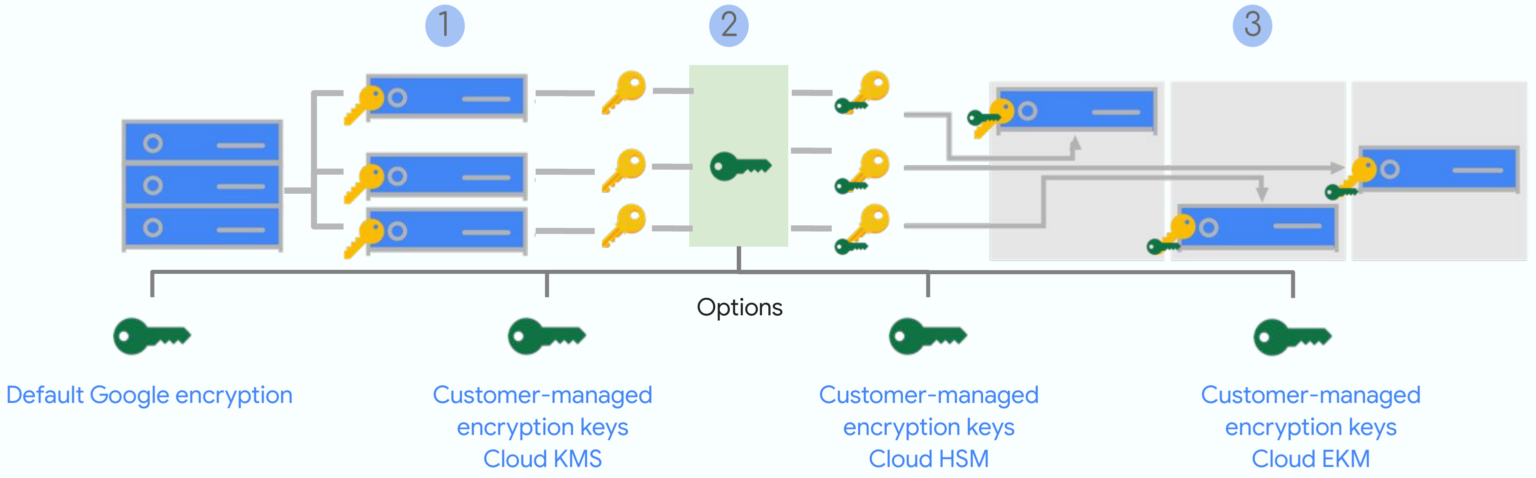
Project separation



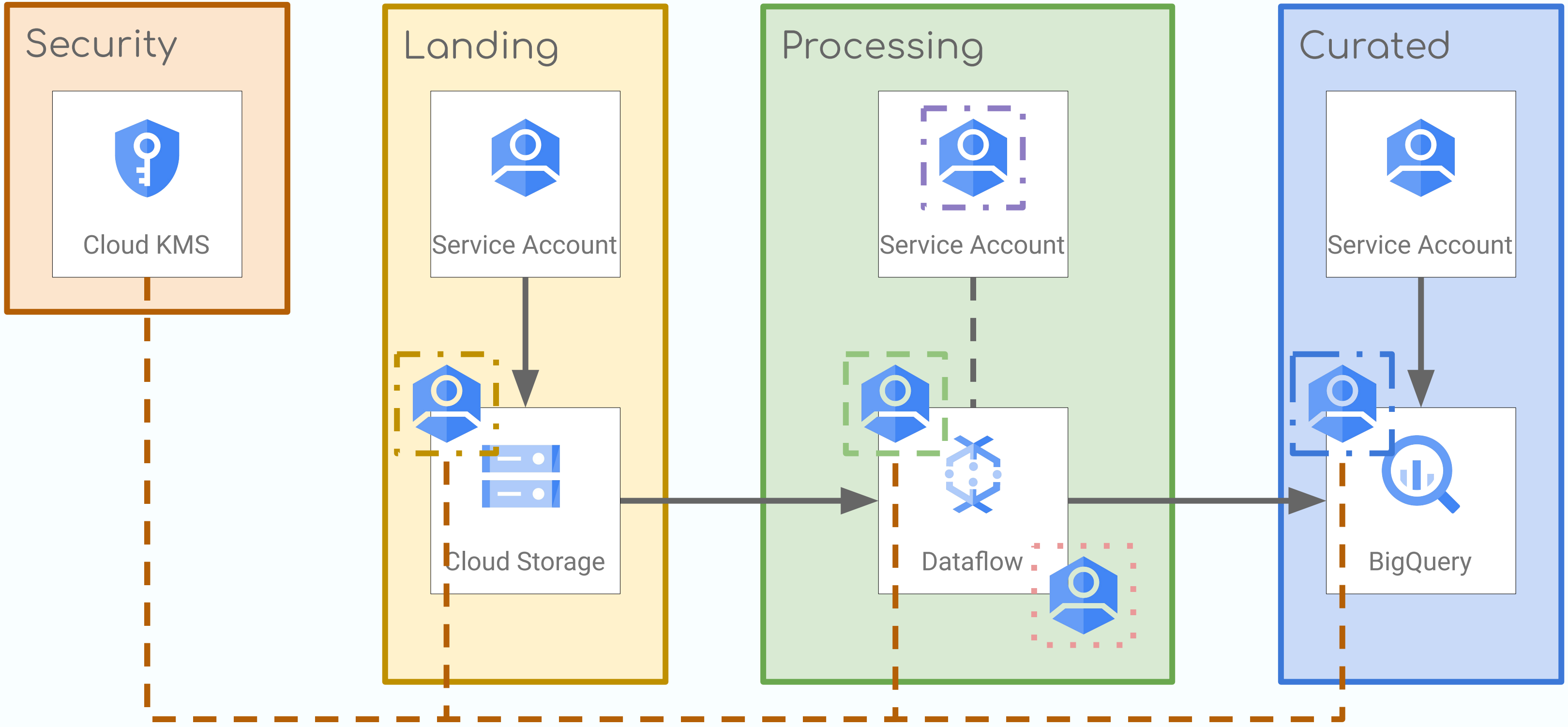
3. At rest encryption

Data at rest are encrypted on GCP:

1. Data split in chunk and encrypted with a key: Data Encryption Key (DEK)
2. DEK encrypted with Key Encryption Key (KEK)
3. Chunk stored with encrypted DEK



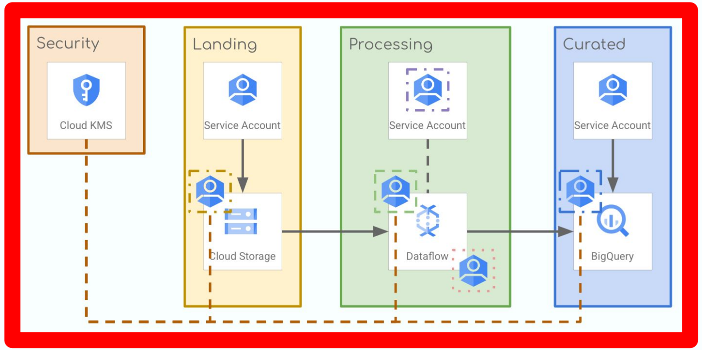
3. At rest encryption



`roles/cloudkms.cryptoKeyEncrypterDecrypter`

Recap

1. Every tenants must be isolated and dedicated to a specific system of services.
2. Internal addressment of tenants must be private.
3. All data must have encryption at-rest with keys managed by ACEME's security team.



SCAN ME

End to end example

Lorenzo Caggioni

QUESTIONS?

Contact info
https://twitter.com/lcaggio
https://www.linkedin.com/in/lcaggio
https://github.com/lcaggio

BEAM SUMMIT

Simplifying Speech-to-Text Processing with Apache Beam and Redis

Pramod Rao
& Prateek Sheel

BEAM SUMMIT
NYC 2023

Simplifying Speech-to-Text Processing with Apache Beam and Redis

Pramod Rao

Cloud Data Engineer

Google Cloud Consulting

Prateek Sheel

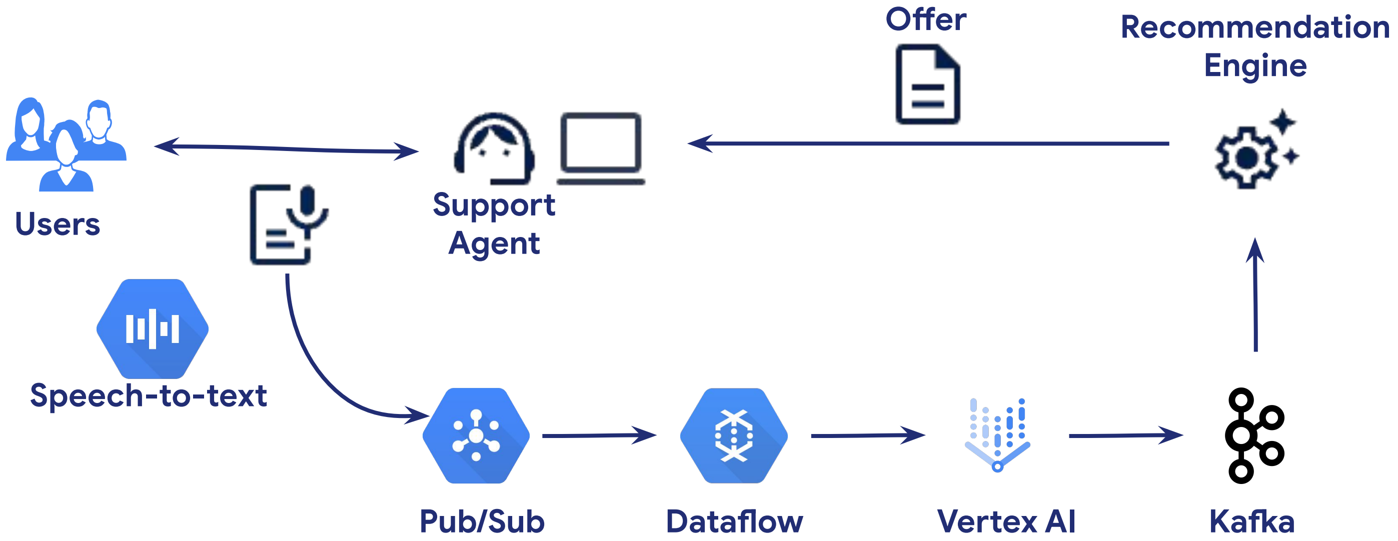
Data & Analytics Consultant

Google Cloud Consulting

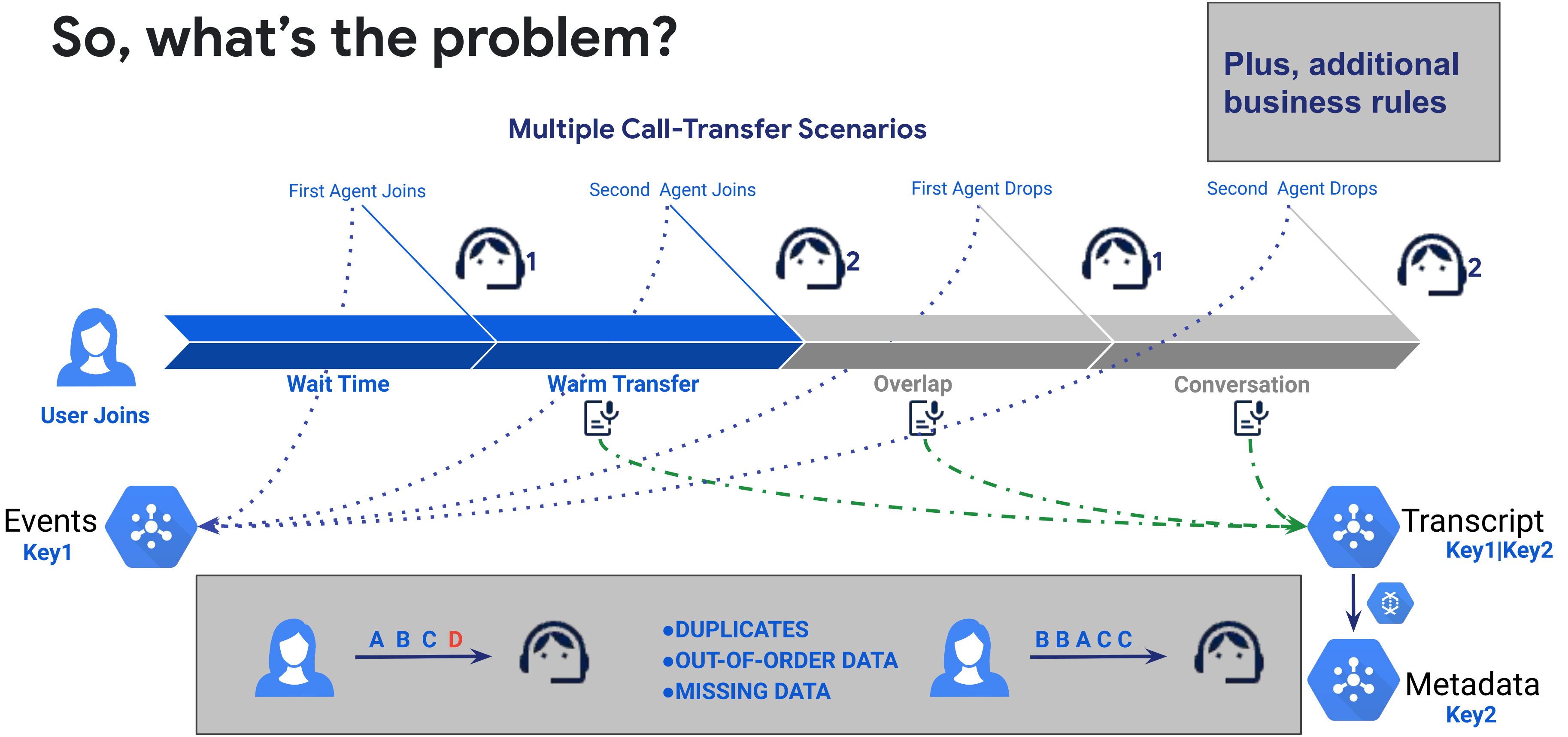
01

Overview

Business Process



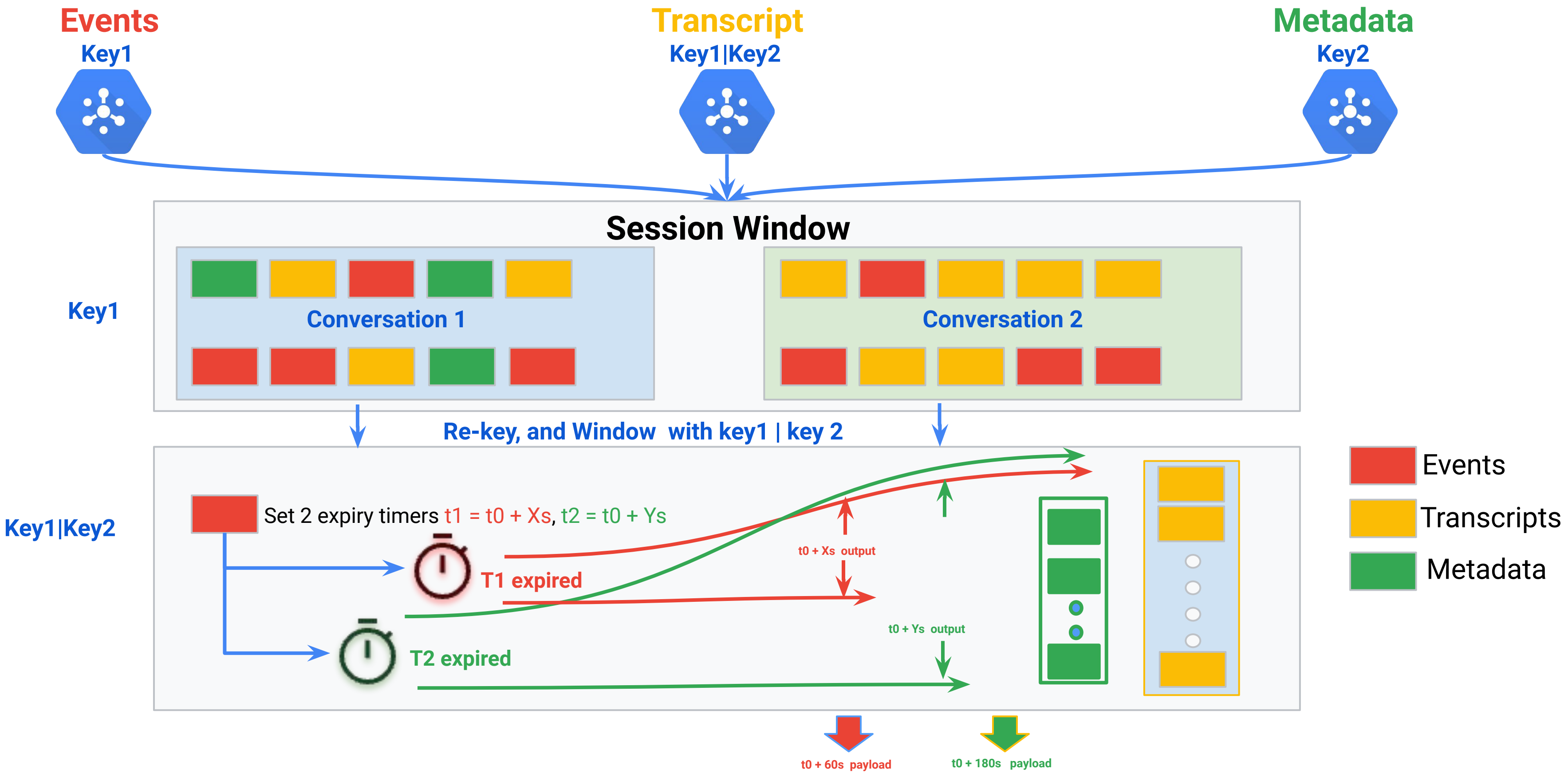
So, what's the problem?



02

Design Journey

Design Approach # 1



Design 1 Trade Offs

Dependencies

No state external to Dataflow. No external service dependencies.

Latency

Need to wait for the session to end and the timers to expire before the output payloads can be produced. Not ideal based on the business SLO.

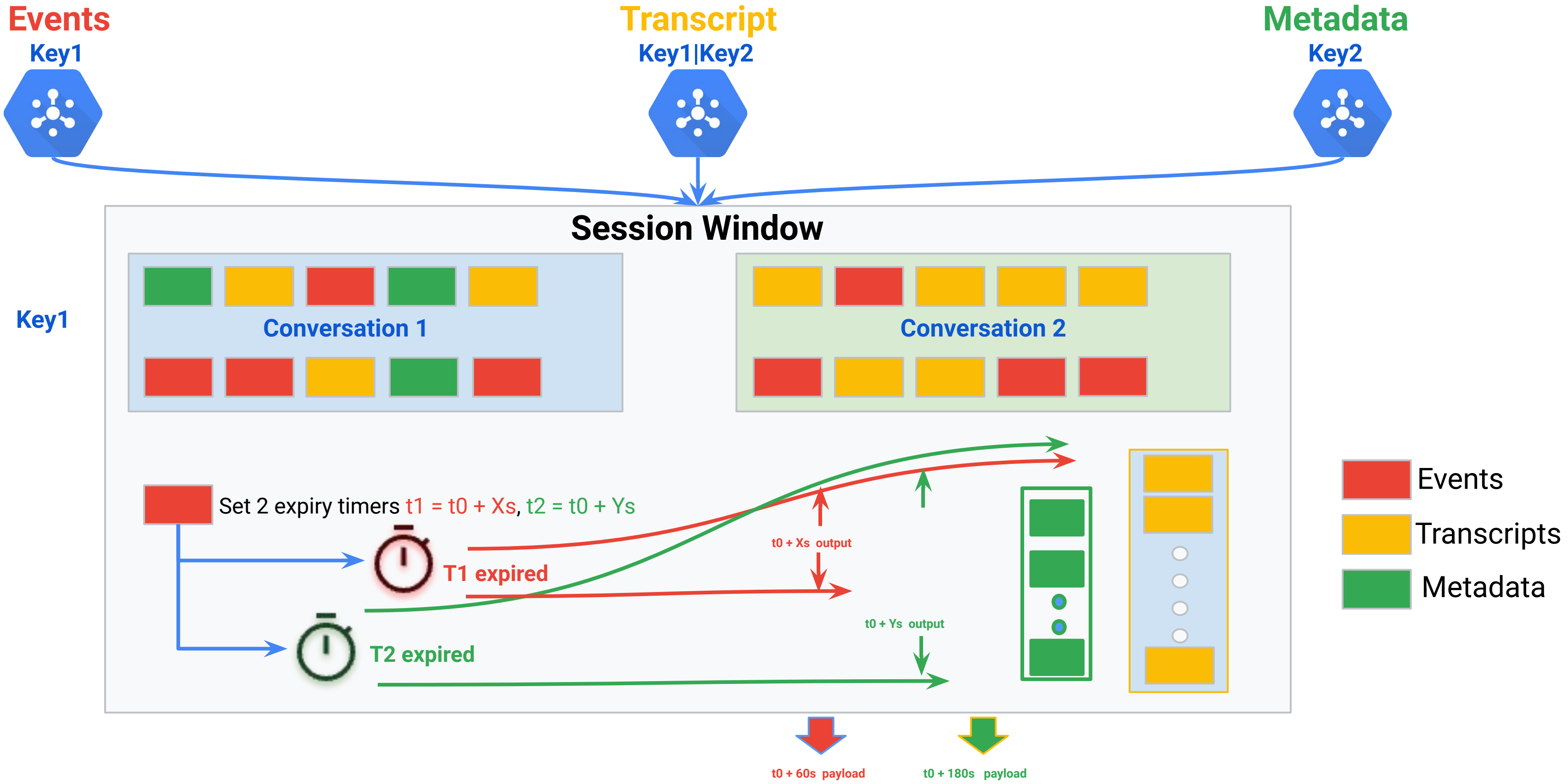
Completeness

In some cases all of the information required to creating the output payloads may not be available when the timers expire. This is due to the uncertain **ordering** of events.

Code Complexity

Windowing allows for ***relatively*** simpler business logic implementation for creating the output payloads since re-keying produces outputs at the required ***granularity***

Design Approach # 2



Design 2 Trade Offs

Dependencies

No state external to Dataflow. No external service dependencies.

Latency

Need to wait for the session to end and the timers to expire before the output payloads can be produced. Not ideal based on the business SLO.

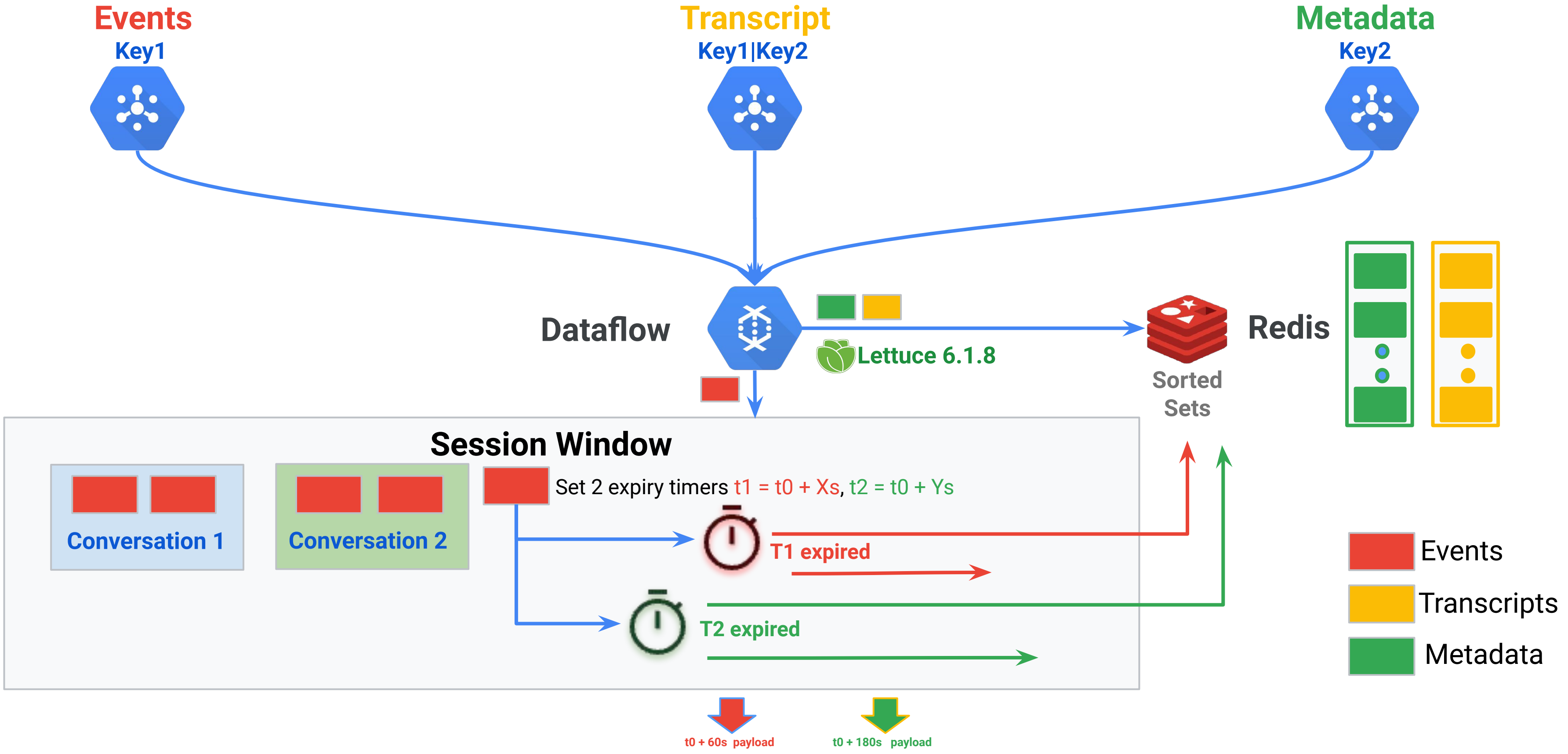
Completeness

In some cases all of the information required to creating the output payloads may not be available when the timers expire. This is due to the uncertain **ordering** of events.

Code Complexity

Granularity of outputs doesn't match the inputs thereby increasing the business logic **complexity** required to produce the output payloads

Design Approach # 3



Redis

Latency

Low latency data store that dovetails well with streaming use cases

Order

We rely on Redis sorted sets for accumulating the speech transcripts, we are able to maintain the **order** of the conversation as well as **deduplicating** the transcripts **automagically**

Data Lifecycle

Redis offers a simple approach to manage **cleanup** of stale data

Design 3 Trade Offs

Dependencies

Dependency on a managed Redis instance. This also results in additional **costs** to host a Redis instance in the Cloud environment.

Latency

No need for any additional wait time over and above the required timers.

Subsecond end-to-end latency for ML predictions.

Completeness

Least chance of incomplete outputs due to the **ordering** provided by Redis

Code Complexity

Much **simpler processing** because complicated scenarios related to cross-referencing the three data sources are eliminated. Only need to "act" on events.

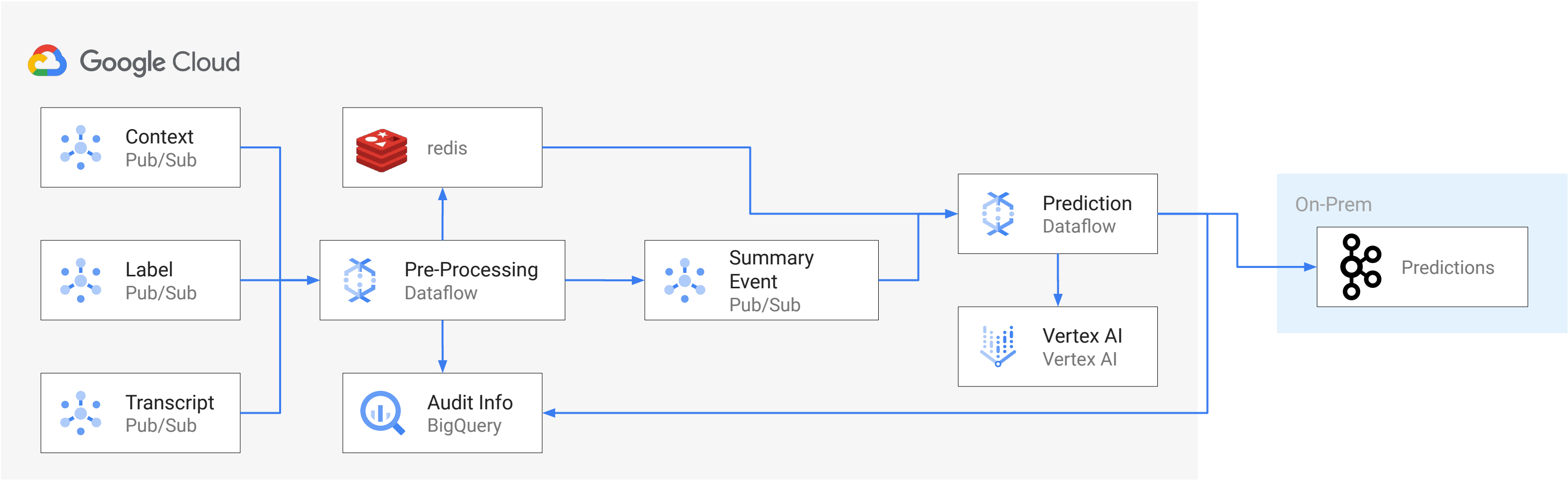
Latency Metrics*

Dataflow Machine Type	PreProcessing Avg. (ms)	Redis Avg. (ms)	Predictions Avg. (ms)	End-To-End Avg. (ms)
n1-standard-2 t0+60s	1210.90	20.84	204.83	1441.75
n1-standard-2 t0+180s	1155.52	18.62	260.33	1441.72
n2d-standard-4 t0+60s	580.38	9.84	198.68	796.10
n2d-standard-4 t0+180s	596.54	9.98	260.54	874.35

*Excluding the wait time to accumulate data for each event type

Final Solution

Speech-to-text Processing with Apache Beam and Redis



03

Lessons Learned

Lessons Learned

Functional

Order of data

Real world scenarios include out-of-order data, duplicates, and missing elements

Granularity of inputs

Business logic is greatly simplified if all inputs are at the same level of "granularity"

Latency

Latency requirements dictate the nature of the final solution

Operational

Observability

Non functional requirements such as operational metrics and dead-letter queues are essential to gain insights into the processing state at any time

Configurability

Levers should be provided to change the processing characteristics without changing any code

Representative test data

"Good" test data is imperative to shorten the development lifecycle and can be tricky to generate or acquire

Thank you!

https://cloud.google.com/consulting

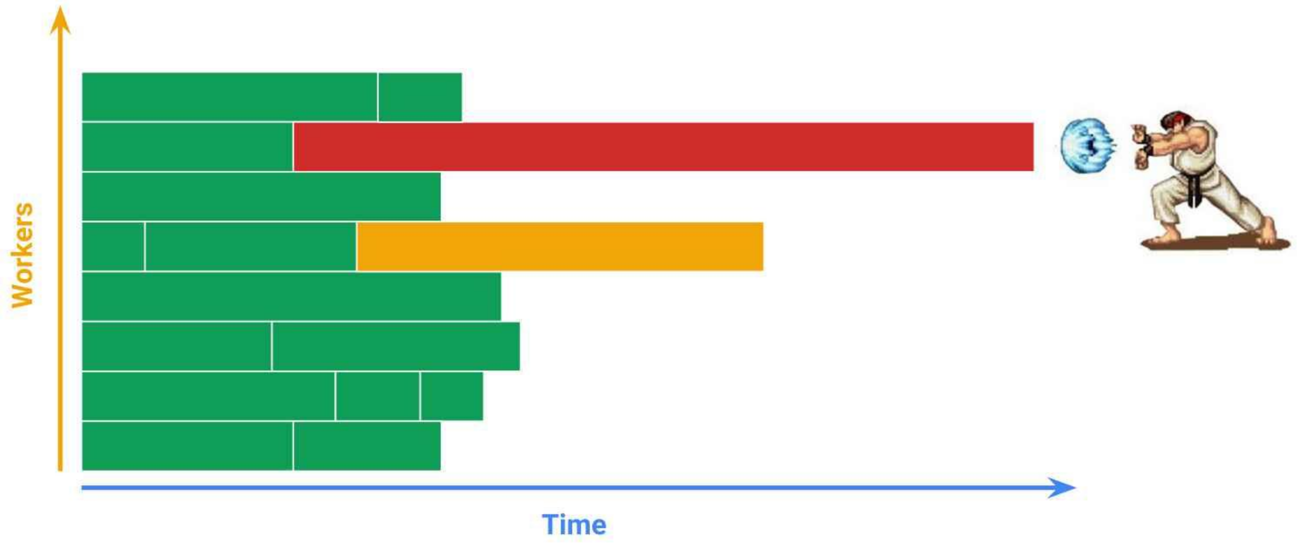
BEAM

SUMMIT

Hot Key Detection and Handling in Apache Beam Pipelines

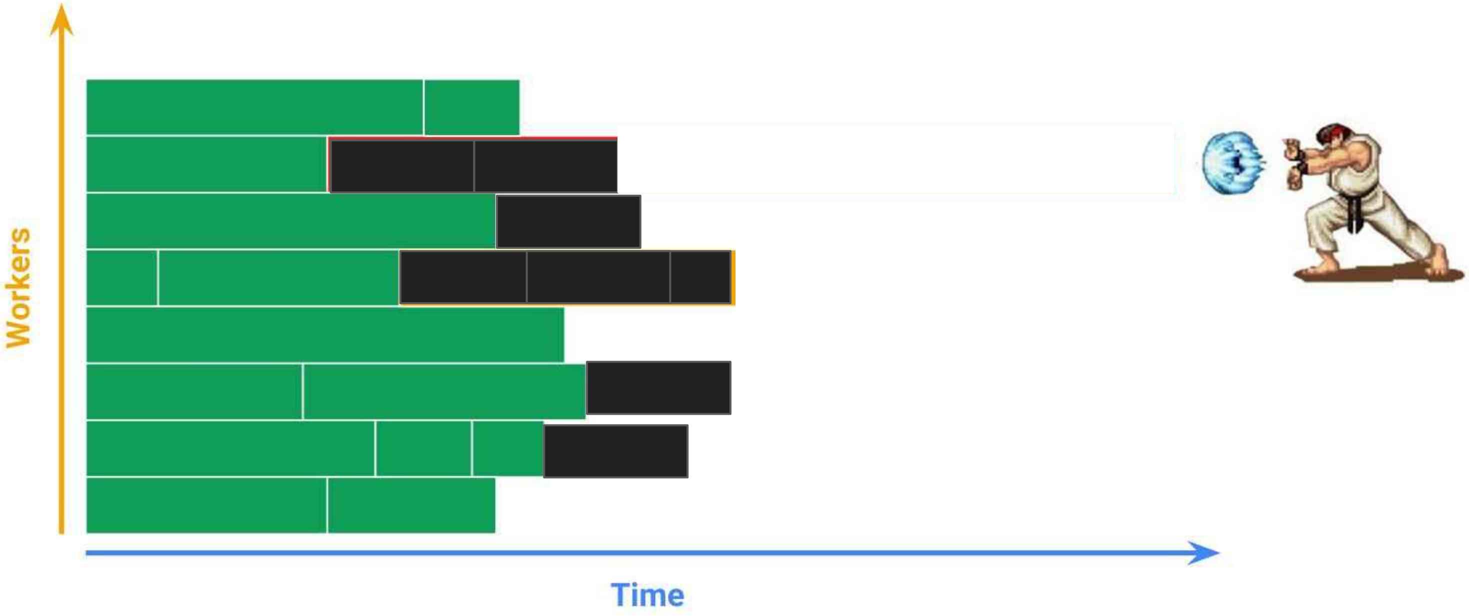
Shafiqa Iqbal
& Ikenna Okolo

BEAM SUMMIT
NYC 2023

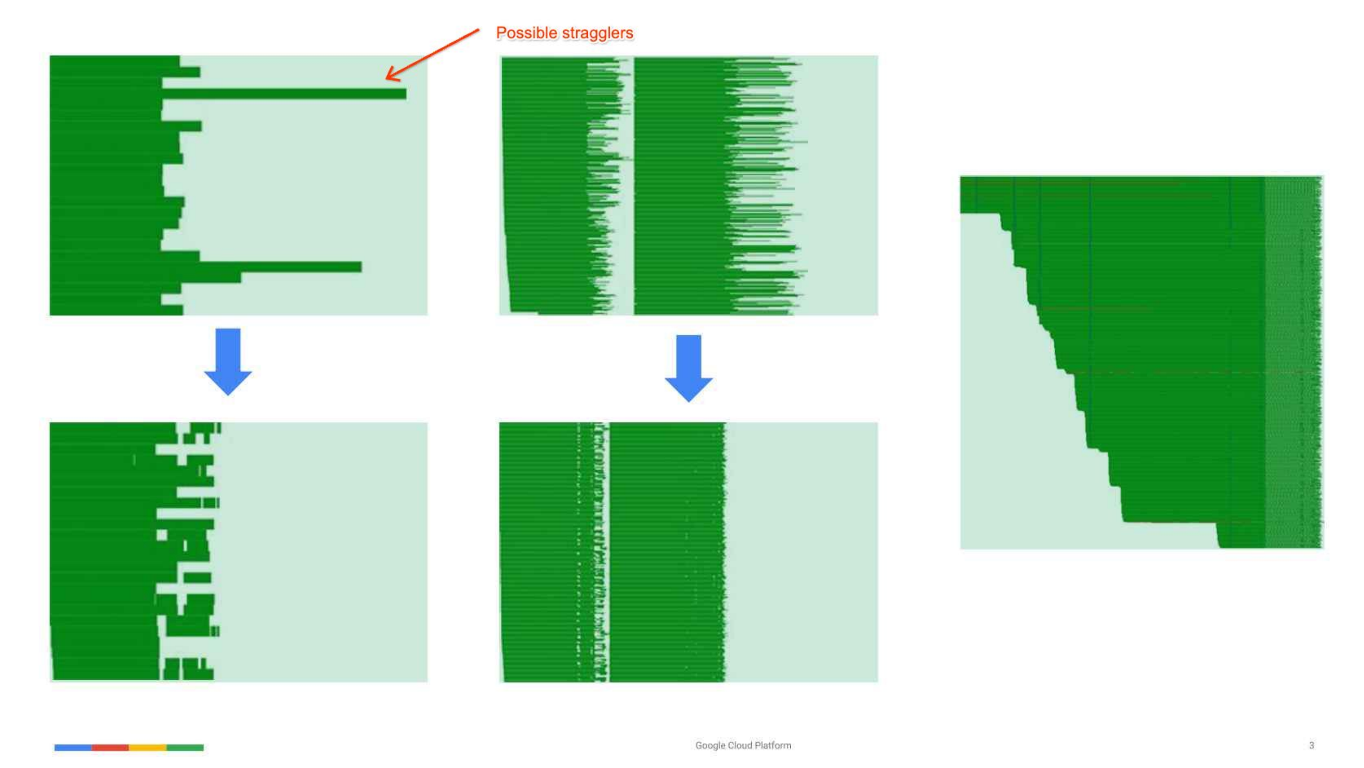


Google Cloud Platform 2

To this



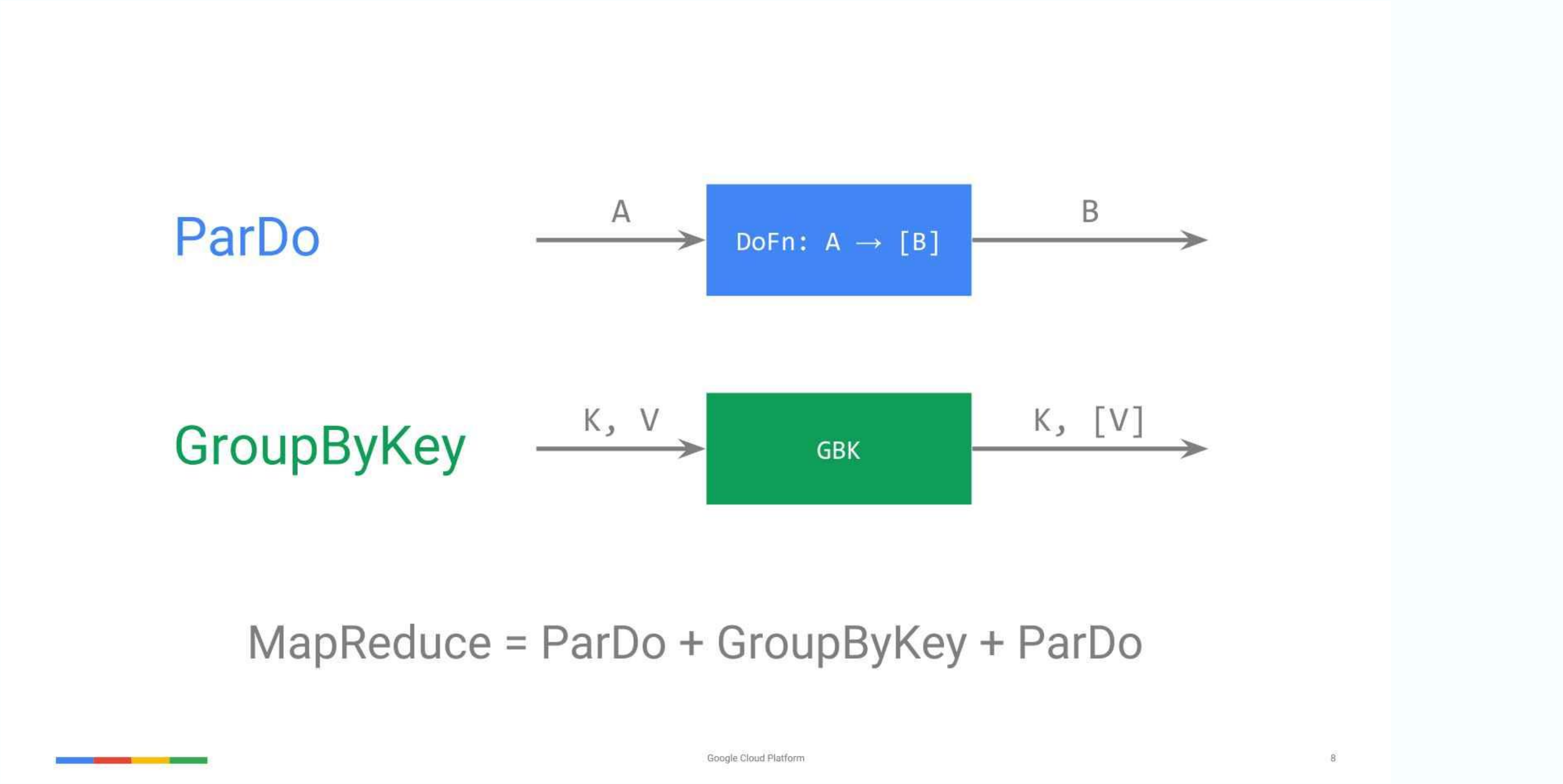
How stragglers can look like



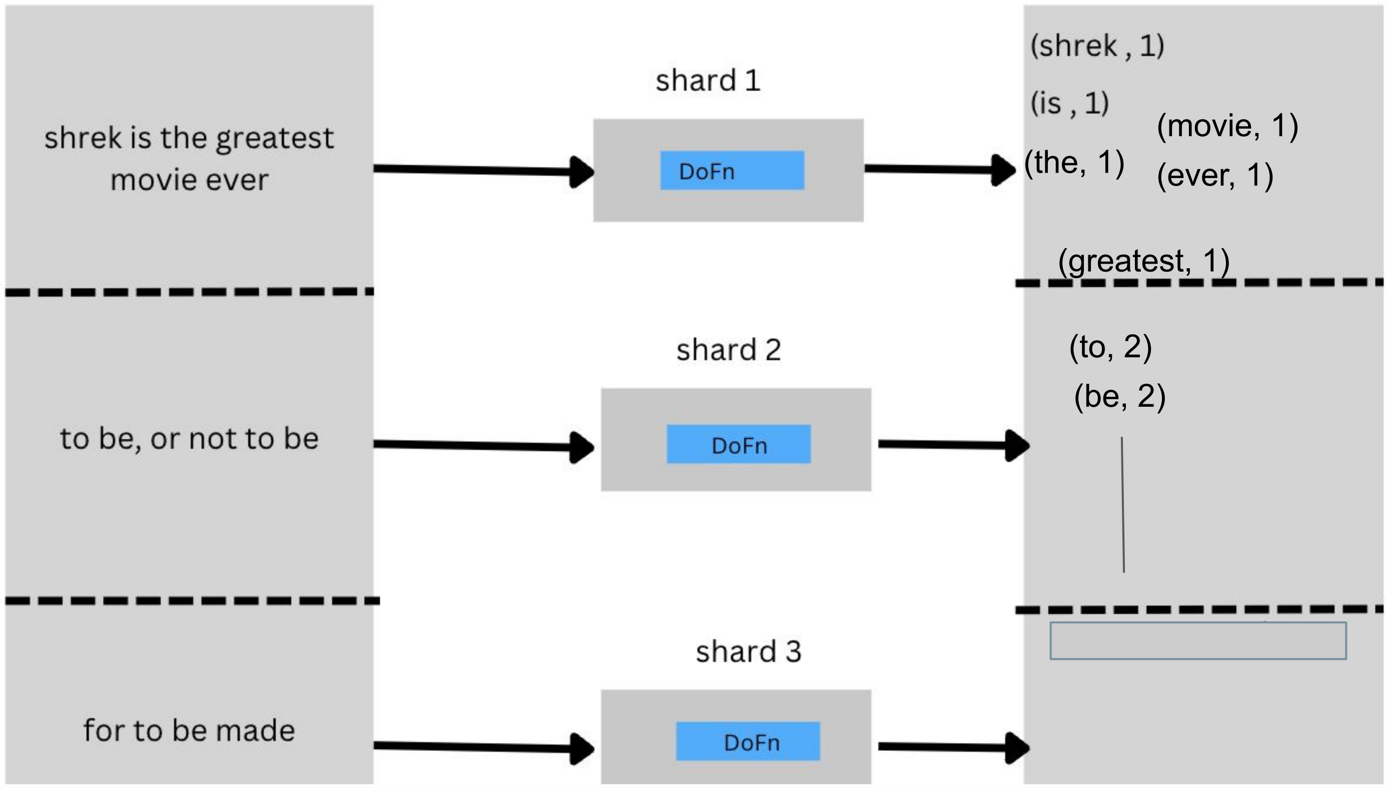
WordCount

```
Pipeline p = Pipeline.create(options);
p.apply(TextIO.Read.from("gs://dataflow-samples/shakespeare/*"))
 .apply(FlatMapElements.via(
     word → Arrays.asList(word.split("[^a-zA-Z']+"))))
 .apply(Filter.byPredicate(word → !word.isEmpty()))
 .apply(Count.perElement())
 .apply(MapElements.via(
     count → count.getKey() + ": " + count.getValue())
 .apply(TextIO.Write.to("gs://.../..."));
p.run();
```

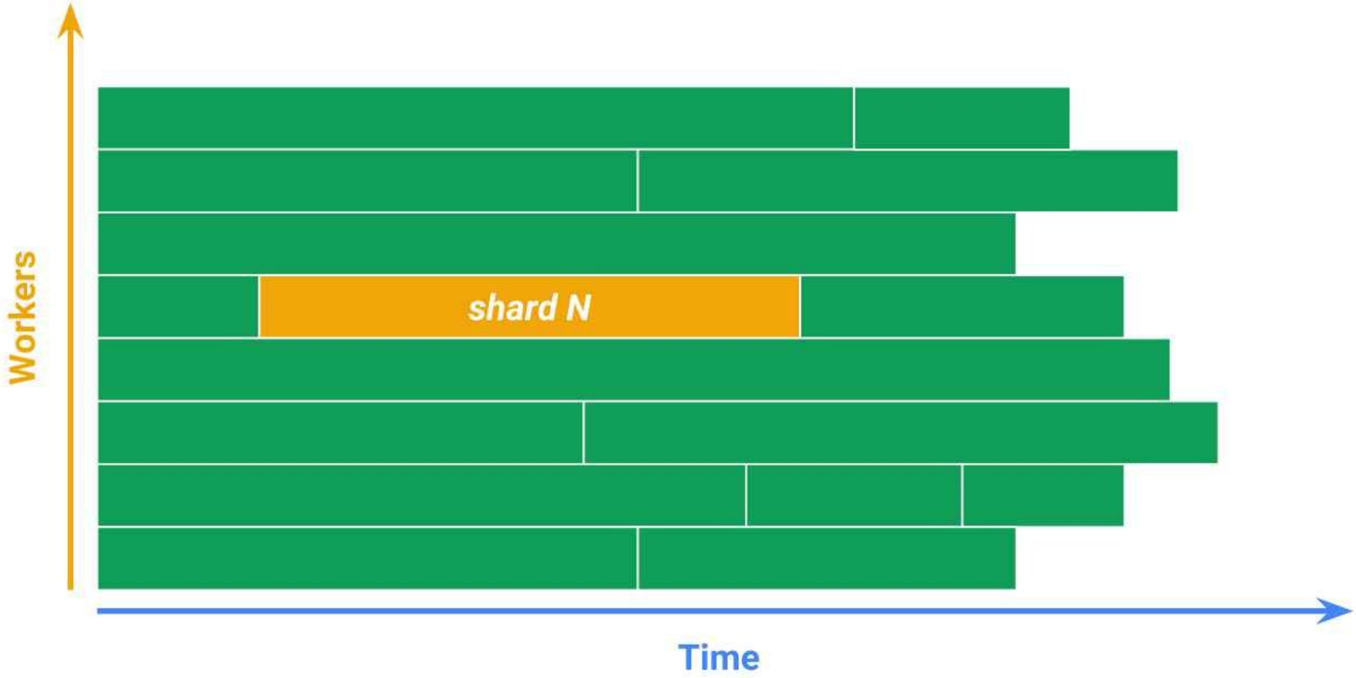
Primitives to keep in mind



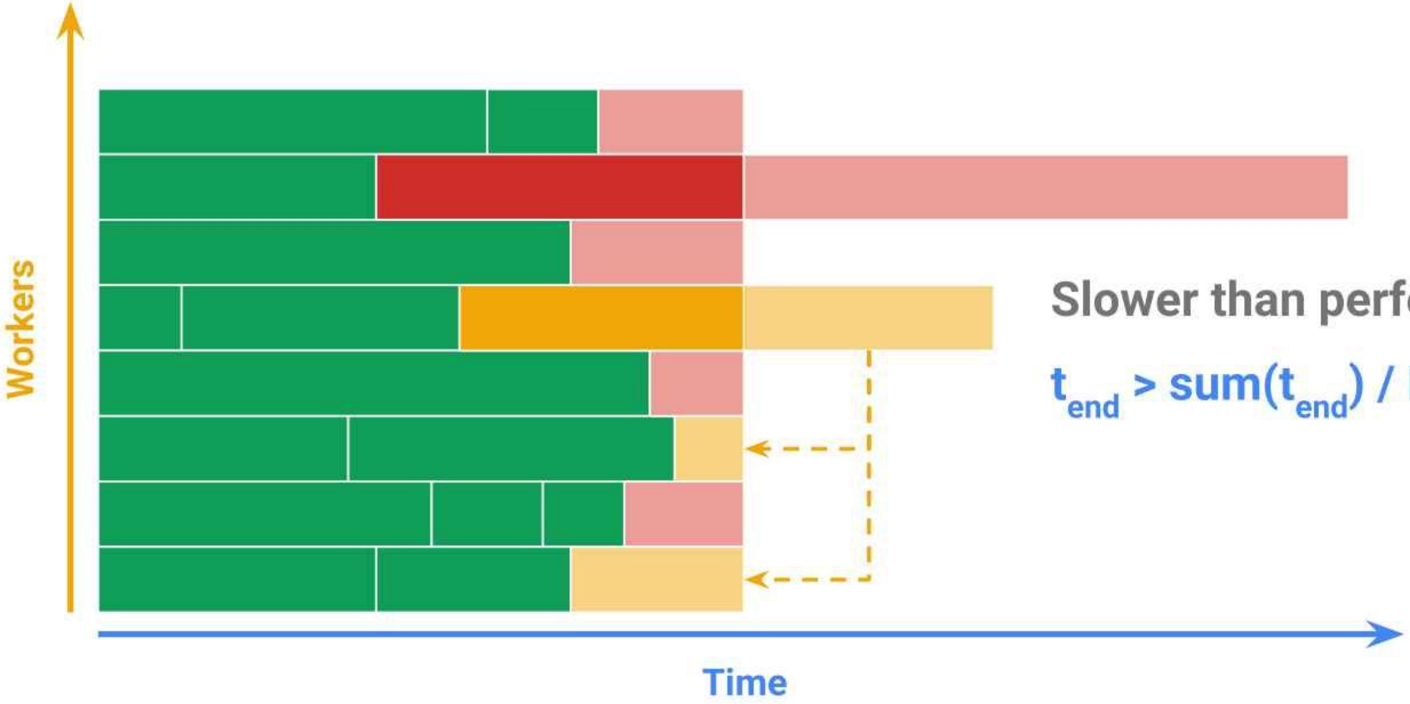
How a ParDo would work



Gantt charts



What is a straggler, really?



Slower than perfectly-parallel:

$t_{end} > \text{sum}(t_{end}) / N$

Amdahl's law: it gets worse at scale

#workers

serial fraction

$$Speedup = \frac{N}{1 + (N-1) \cdot S}$$

Higher scale ⇒ More bottlenecked by serial parts.

Reasons for Stragglers

Uneven partitioning

- Process dictionary in parallel by first letter -> 6x speedup only by ahmdahl's law

Uneven Complexity

- Join keys with some external input values

Uneven resources

- Bad machines, network or resource contention

Bugs

- Slow RPCs or bugs

Reasons for Stragglers

Hot keys

Uneven partitioning

- Process dictionary in parallel by first letter -> 6x speedup only by ahmdahl's law

Uneven Complexity

- Join keys with some external input values

Uneven resources

- Bad machines, network or resource contention

Bugs

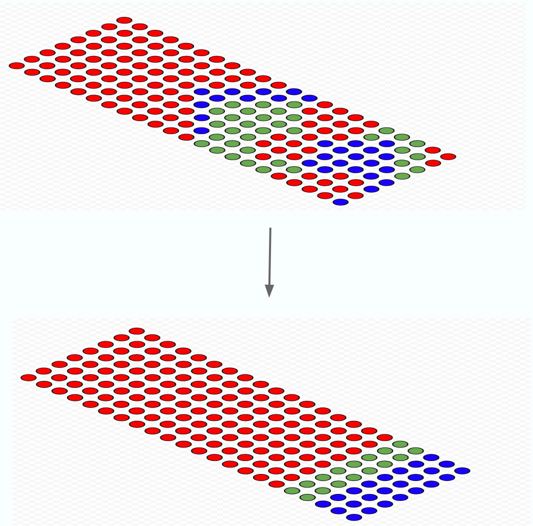
- Slow RPCs or bugs

What are hotkeys

A hot key is a key with enough elements to negatively impact pipeline performance. These keys limit a Pipeline’s ability to process elements in parallel, which increases execution time.

Think about hotkeys in this way. Let's imagine there's a room filled with 150 Red, 30 Blue and 20 Green unsorted plates and there are 3 students who are to arrange those plates in sorted orders (as seen here to the right).

Let's assume that student 1 will sort the Red plates, student 2 will sort the blue and the last student will sort the green plates.



What are hotkeys contd...

From the illustration in the previous slide, students 2 and 3 will finish before student 1. Though the second and third students had already completed sorting their respective colored plates, they have to wait for the first student to complete theirs before the task can be termed as completed. This delay by student 1 is due to the larger number of plates they need to sort. In parallel processing, this is referred to as hotkeys.

If we replace the students with workers and the unsorted-plates with work-items to be processed, we can apply the same thinking to Dataflow pipelines. If the work-items are not evenly distributed, then there's bound to be an issue of hotkeys which obviously would impact the performance of the Pipeline.

In subsequent slides, we will explain this using a Key Value pair to represent individual work-items.

How do Hotkeys cause problems?

Source Data (KeyValues)	GroupByKey	Transformation

<K1, V>
...
<K1, V>
1,000,000 records
<K2, V>
<K2, V>
<K3, V>
...
<Kn, V>

Note that the dataset is heavily imbalanced.

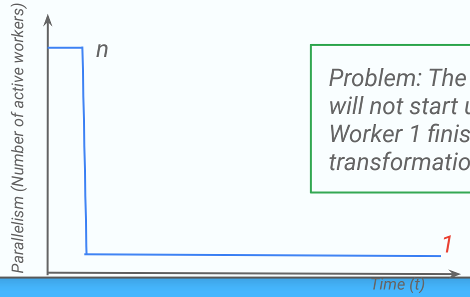
<K1, (V, V, V, ..., V)> → Worker 1 — *100% CPU utilization rate.*

<K2, (V, V)> → Worker 2 — *Idle. Done processing data.*

⋮

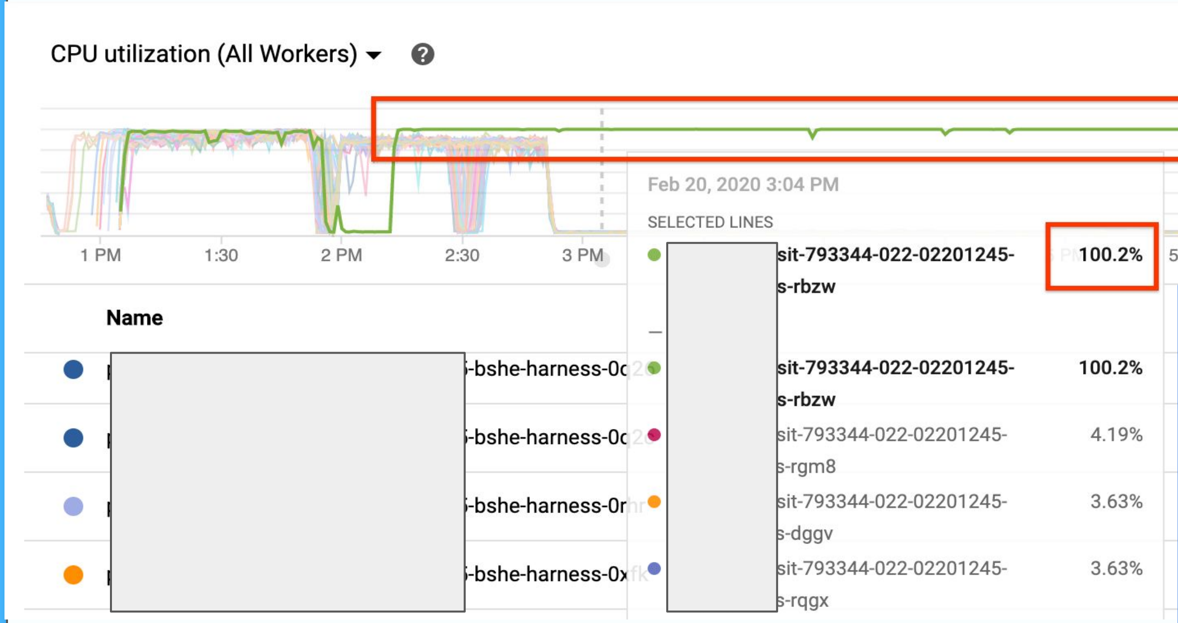
<Kn, (V, V, V)> → Worker n — *Idle. Done processing data.*

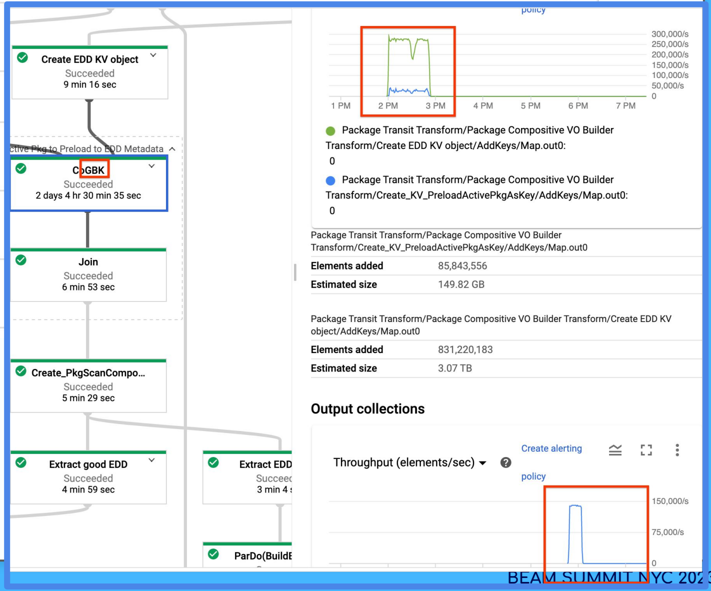
*K1 has broken the uniformity and thus is called the "**hotkey**."*



Problem: The next job will not start until Worker 1 finishes its transformations.

How to identify hotkeys





One of the quickest ways to identify a Job that is impacted by hotkeys is by taking a quick look at the worker CPU utilisation. While some workers are maxing out at about 90% utilisation, some are idle at about <5% utilisation. This truly indicates that there is a possibility that the Job is stuck due to hotkeys.

Fxg-dna-gsi-edd-npe-dev-1, 2020-02-27_04_30_10-5380840952249221324

What can you do?

Uneven partitioning **Uneven complexity** **Uneven resources** **Noise**

Oversplit

Hand-tune

Use data statistics

Backups

Restarts

Predictive

Weak

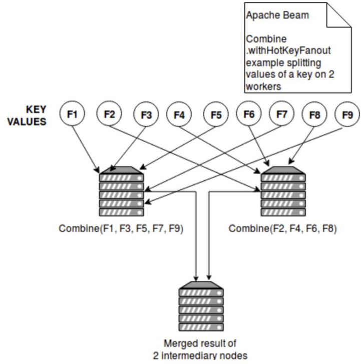
Using statistical analysis to pre-detect the hot key

Data Monitoring, key partitioning, iterative optimization

How do you fix hotkeys?

To resolve this issue, you may have to check that your data is evenly distributed. If a key has disproportionately many values, consider the following courses of action:

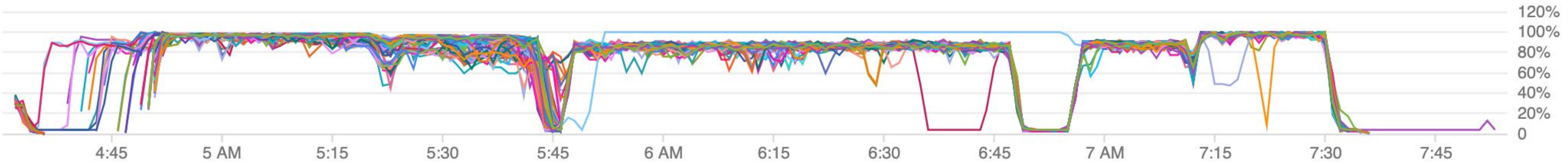
- Rekey their data. Apply a ParDo transform to output new key-value pairs.
- Autosharding
- `Combine.Globally #withFanout(int fanout)`
- Java jobs should consider using the Combine.PerKey.withHotKeyFanout transform.
- Python jobs should consider using the CombinePerKey.with_hot_key_fanout transform.
- Finally, consider enabling Dataflow Shuffle (if using dataflow).



Job not impacted by hotkeys anymore!

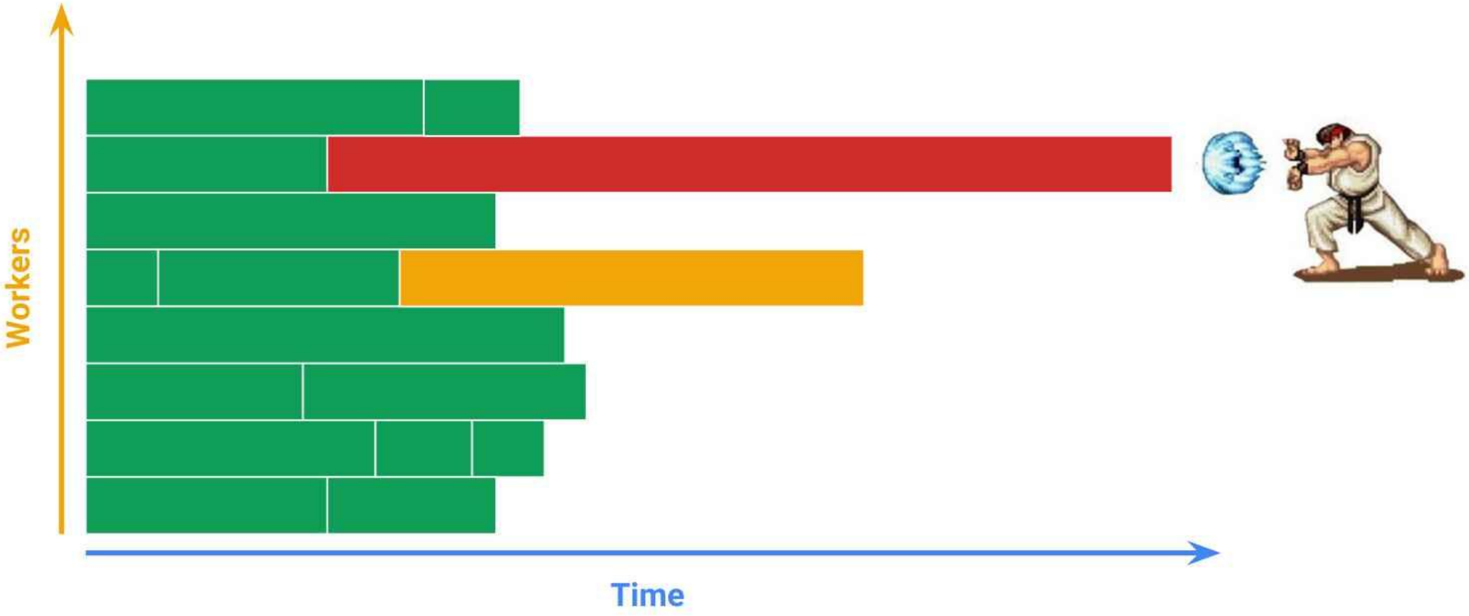
CPU utilization (All Workers)

Create alerting policy

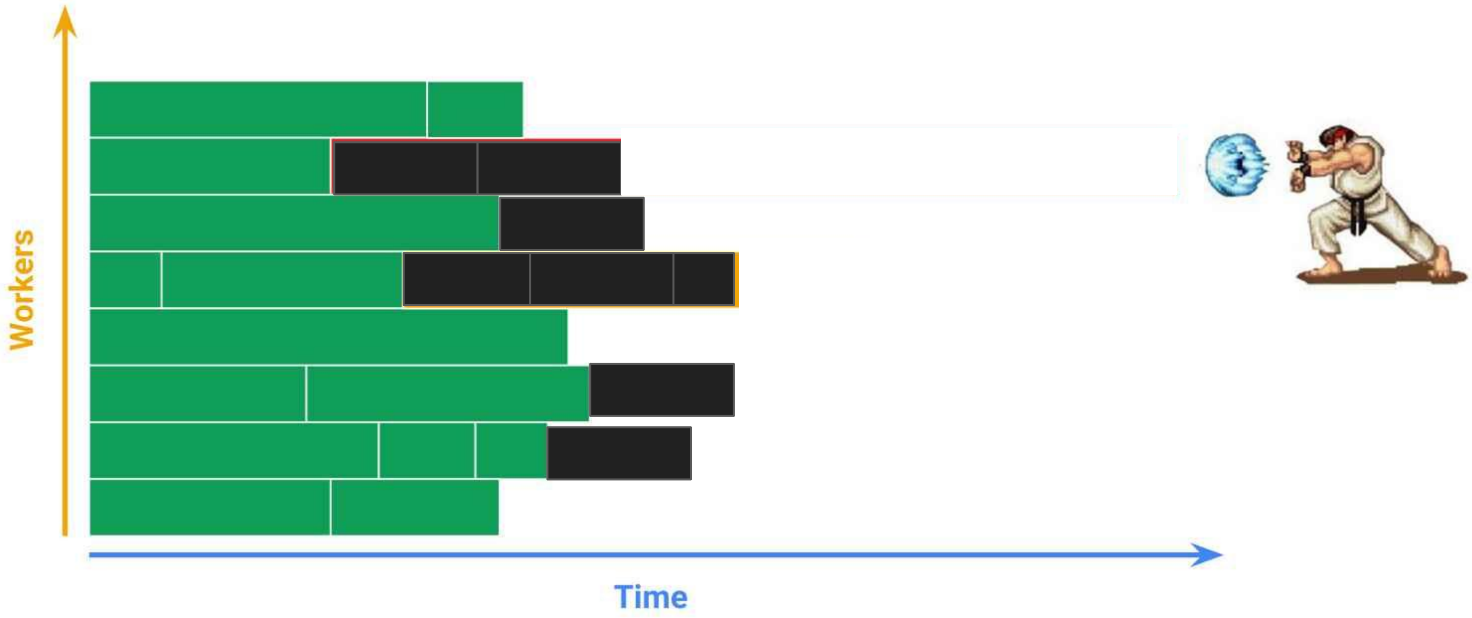


Name	Value
dna-integration-gsi-edd-p-02270430-b79j-harness-0100	1.66%
dna-integration-gsi-edd-p-02270430-b79j-harness-05z9	1.61%
dna-integration-gsi-edd-p-02270430-b79j-harness-0g4x	2.08%
dna-integration-gsi-edd-p-02270430-b79j-harness-0kjp	1.81%

From this



To this



Hotkeys FAQ

Can we assign a more powerful machine to the worker that is processing the hotkey (i.e. Worker 1)?

> **>>** Unfortunately, you cannot. Dataflow, by design, assigns the same machine to *all* of its workers.

In that case, if all workers run with powerful machines, the pipeline will finish quicker.
+ It will be cheap, since most of them will be idle anyways.

> **>>** This will not speed up the process. A powerful machine will still use up only one of its cores. Imagine a giant for-loop to better understand -- cores do not split the work of a for loop.

I enabled autoscale, but my job doesn't finish any faster. Why?

> **>>** You will see in monitoring that the average CPU utilization rate is far below 20%; therefore, Dataflow will not bring in more workers. Even if it does, it won't help -- remember that you already have n-1 idle workers. Surely n idle workers won't make a difference.

Root cause: dataset is imbalanced.
Fix the root cause: balance the dataset.

Solution: Classify the imbalanced key and break them down into smaller pieces.

<K1, V>		<K1.1, V>
…		<K1.1, V>
<K1, V>		<K1.2, V>
<K2, V>		<K1.2, V>
<K2, V>	→	…
<K3, V>		<K2, V>
…		…
<Kn, V>		<Kn, V>

BEAM

SUMMIT

Troubleshooting Slow Running Beam Pipelines

By Mehak Gupta
Google Cloud, Canada

BEAM SUMMIT
NYC 2023

About Me

Hello!

I’m Mehak

Technical Solutions Specialist at Google Cloud

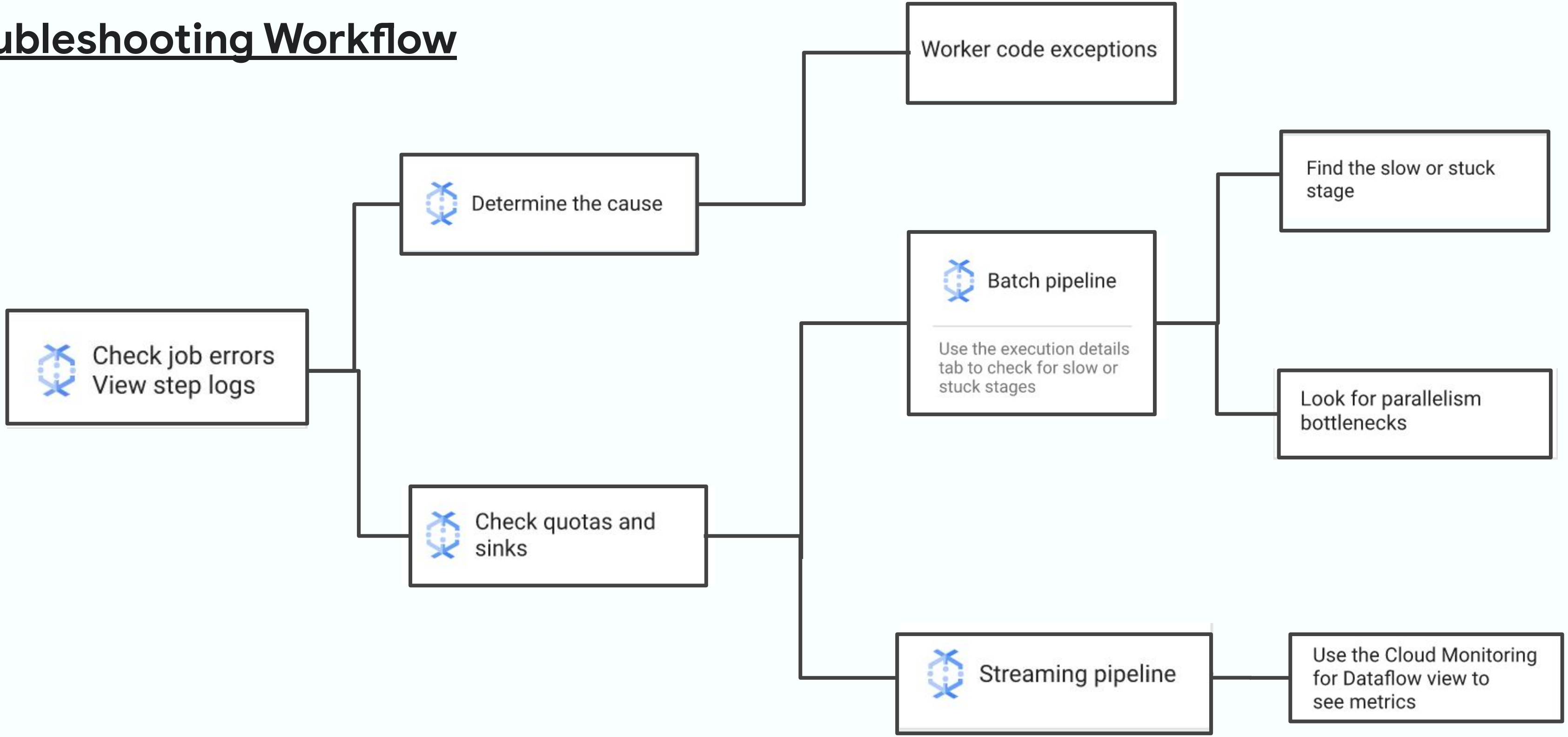
Goals

- Apache Beam pipeline troubleshooting techniques that would empower professionals to research and resolve Beam issues
- Self service skills would reduce MTTR (Mean Time To Recover) from a job failure significantly
- Share some tricks and samples of troubleshooting slow running beam pipelines using Dataflow as an example

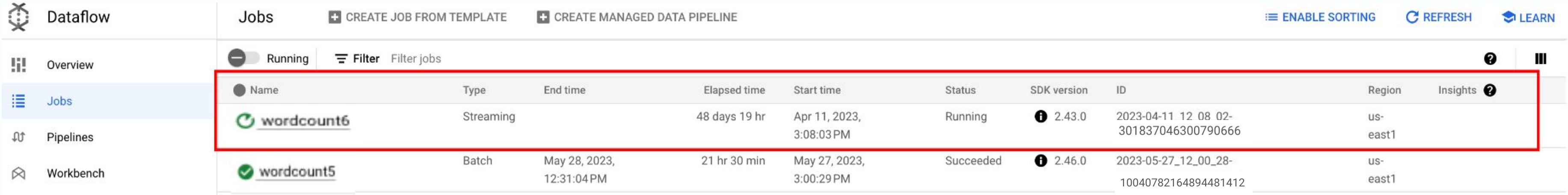
How to identify if the beam pipeline is slow/stuck

- Pipeline is running from a long time without reporting results
- Increased data watermark or system latency
- Pipeline is not consuming input

Troubleshooting Workflow



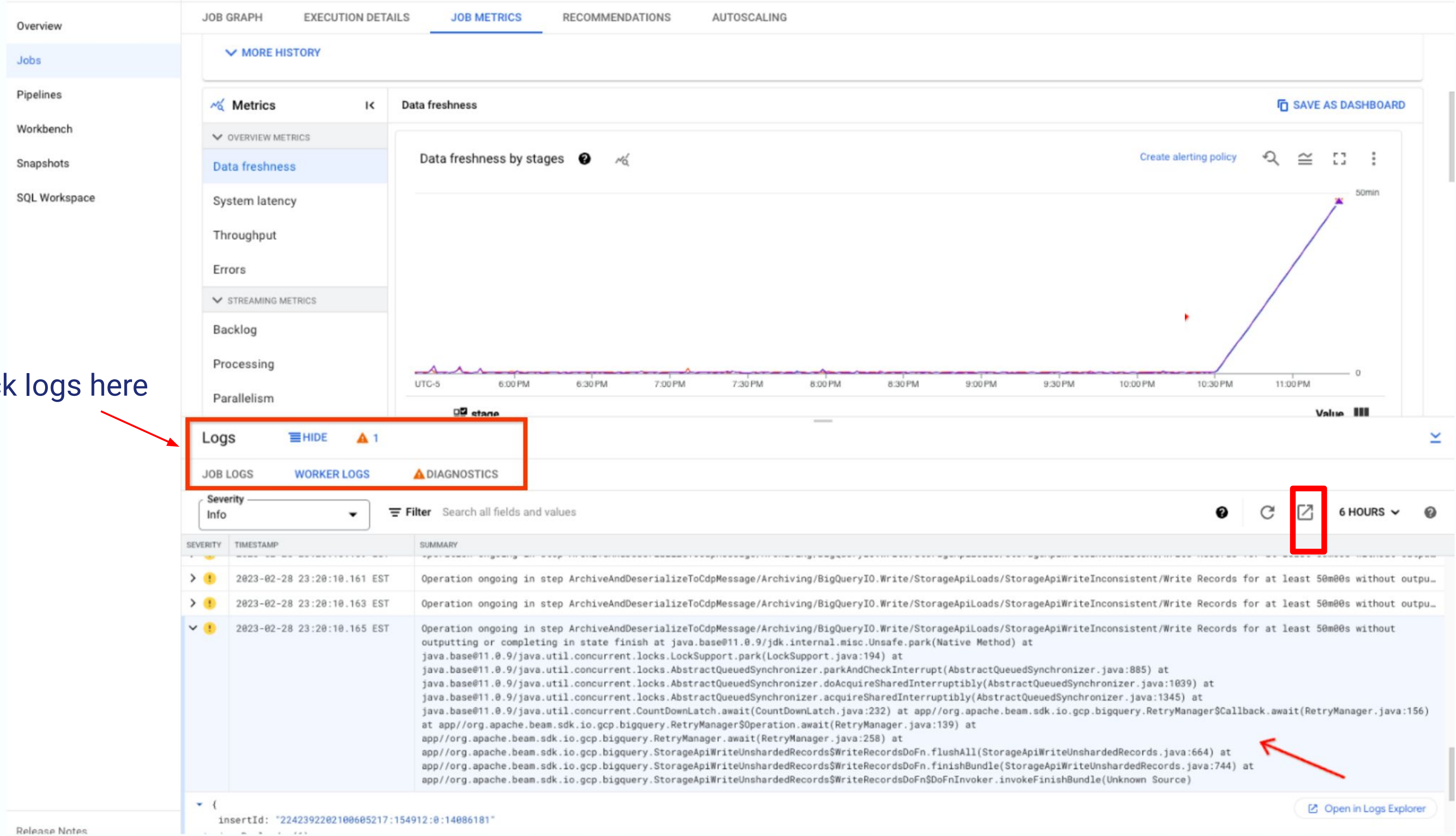
Troubleshoot slow/stuck dataflow jobs



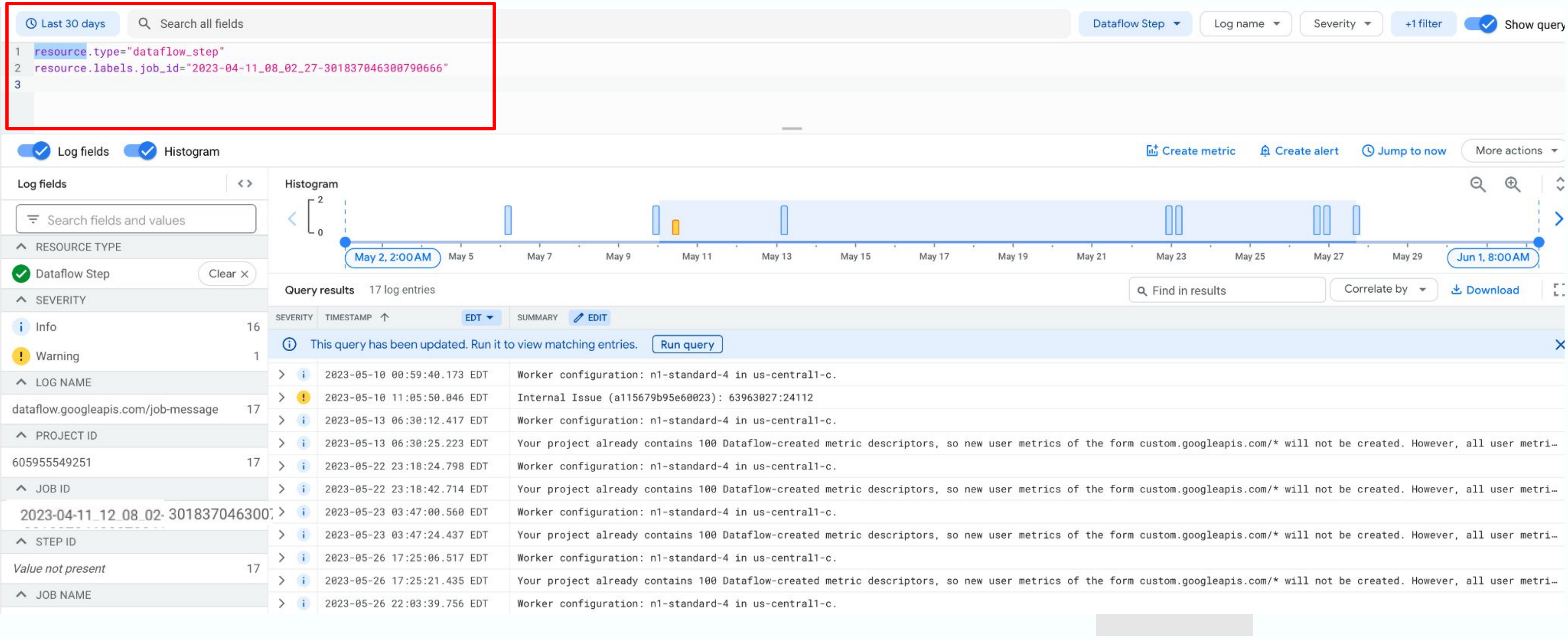
Troubleshooting using Logs Explorer View

Troubleshoot slow/stuck dataflow jobs

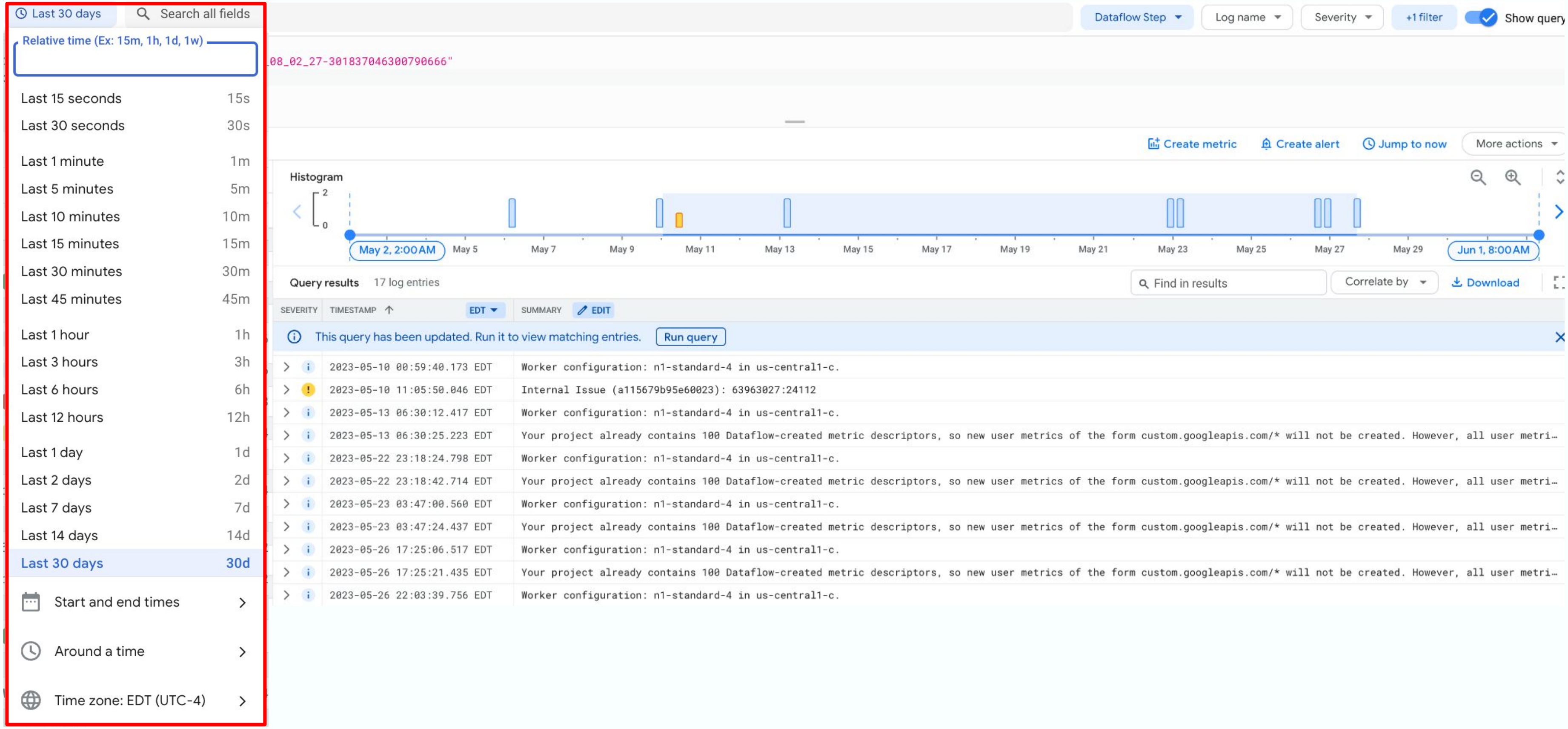
Check logs here



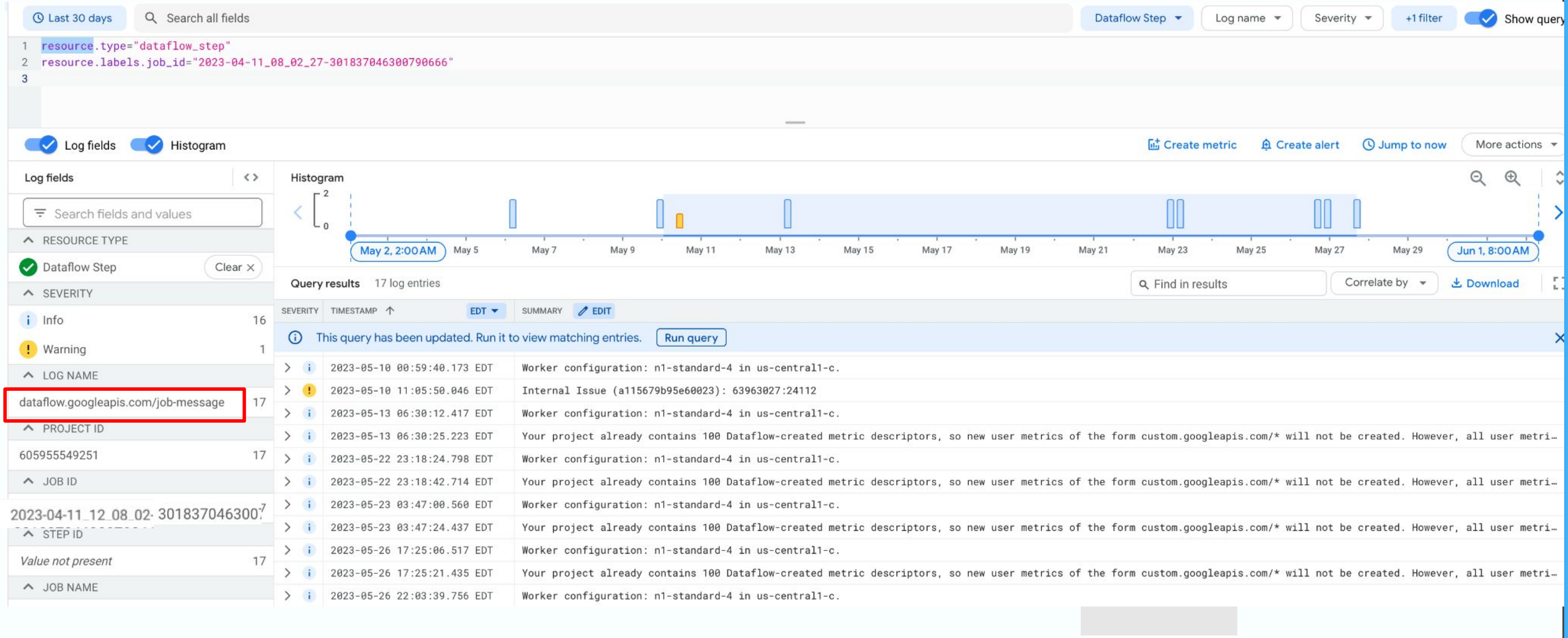
Troubleshoot slow/stuck dataflow jobs



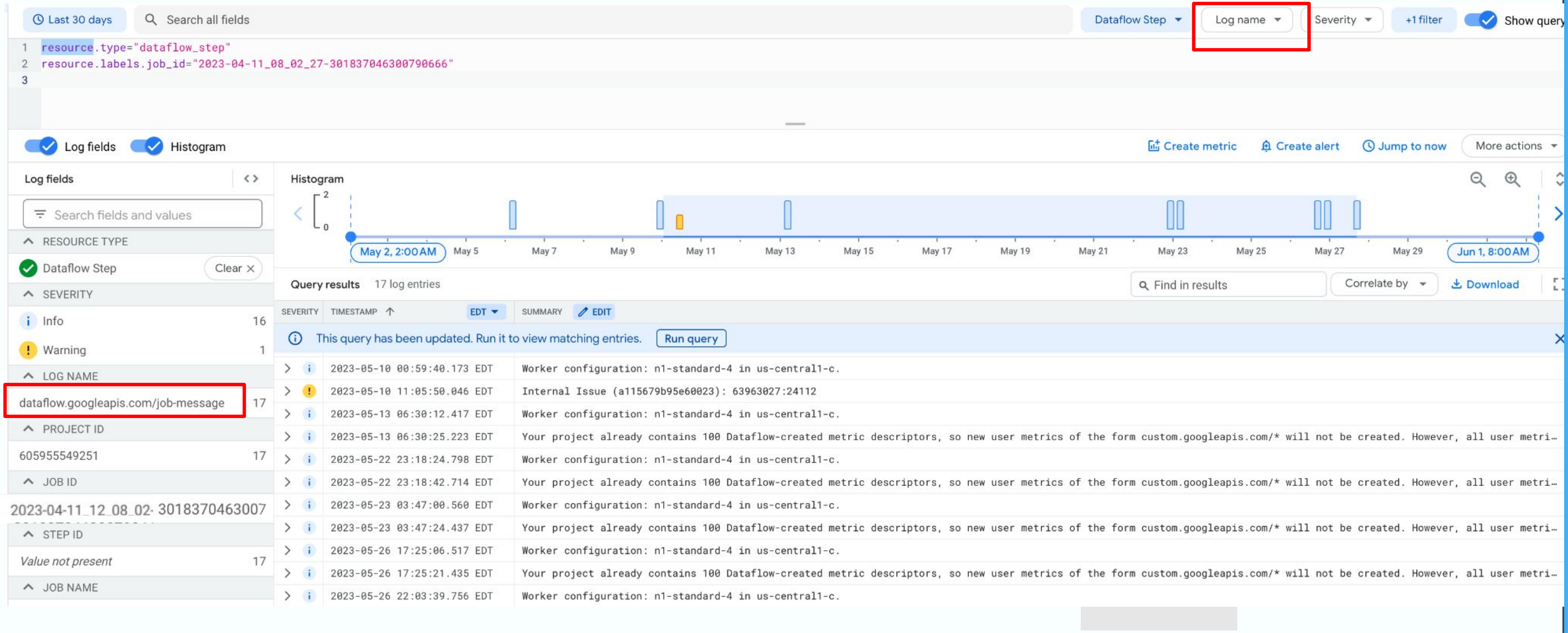
Troubleshoot slow/stuck dataflow jobs



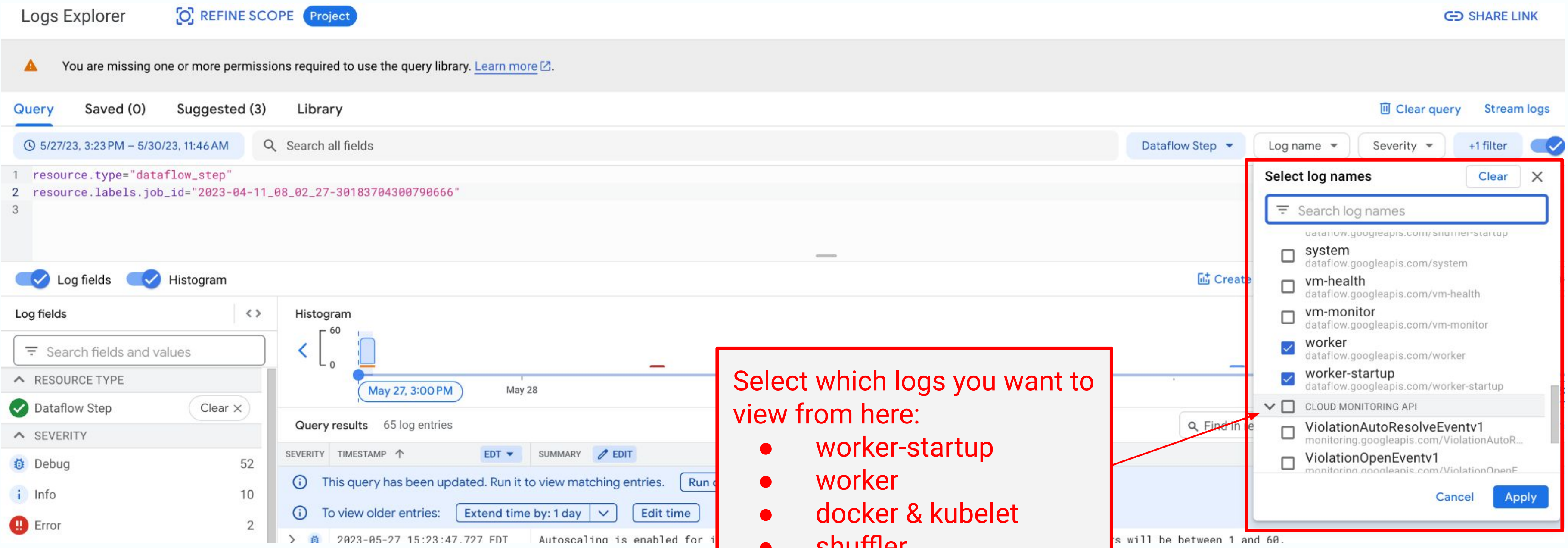
Troubleshoot slow/stuck dataflow jobs



Troubleshoot slow/stuck dataflow jobs



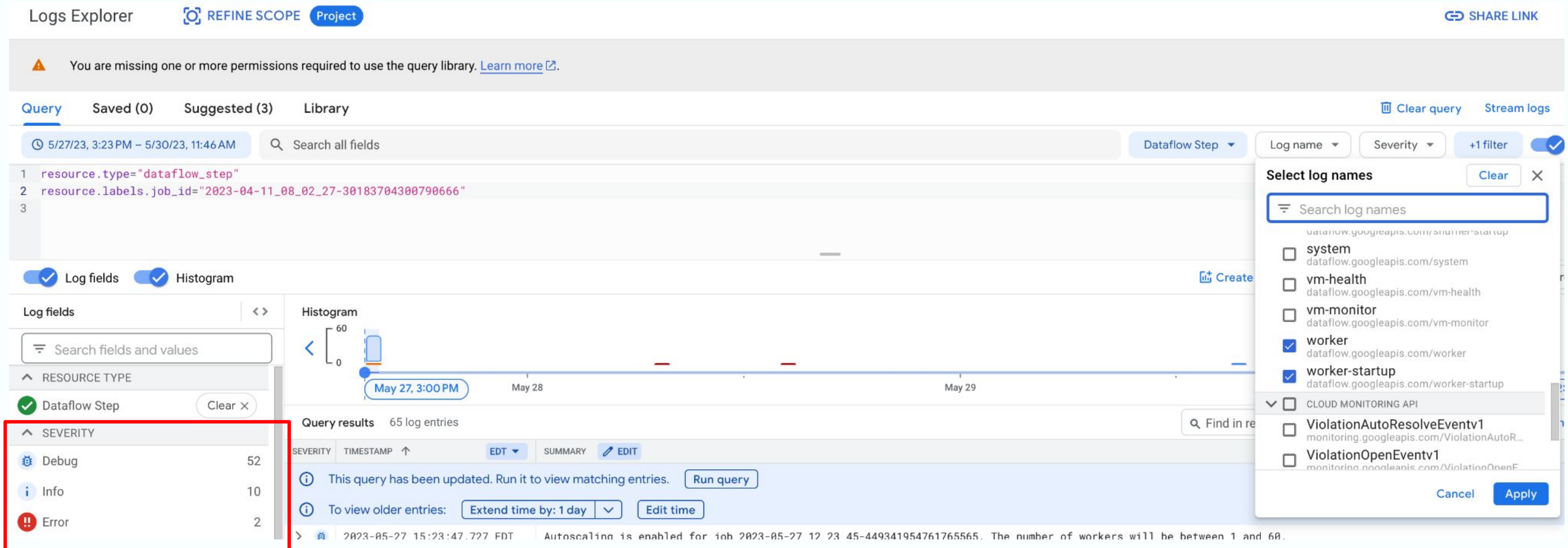
Troubleshoot slow/stuck dataflow jobs



Select which logs you want to view from here:

- worker-startup
- worker
- docker & kubelet
- shuffler

Troubleshoot slow/stuck dataflow jobs



Troubleshooting using Job Metrics Tab

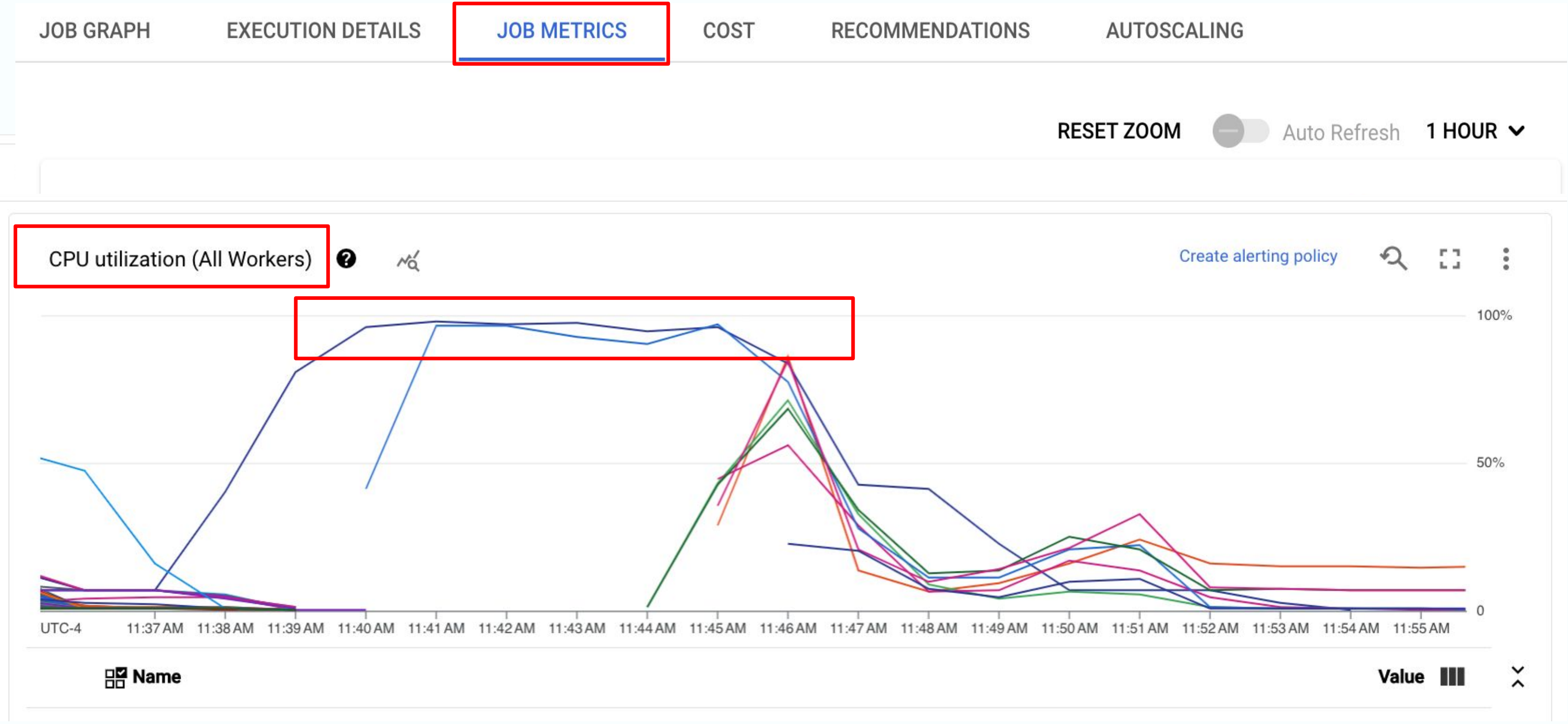
Troubleshoot slow/stuck dataflow jobs

Throughput dropping to zero

Check under "Job Metrics" tab for various metrics

Troubleshoot slow/stuck dataflow jobs

High CPU Utilization

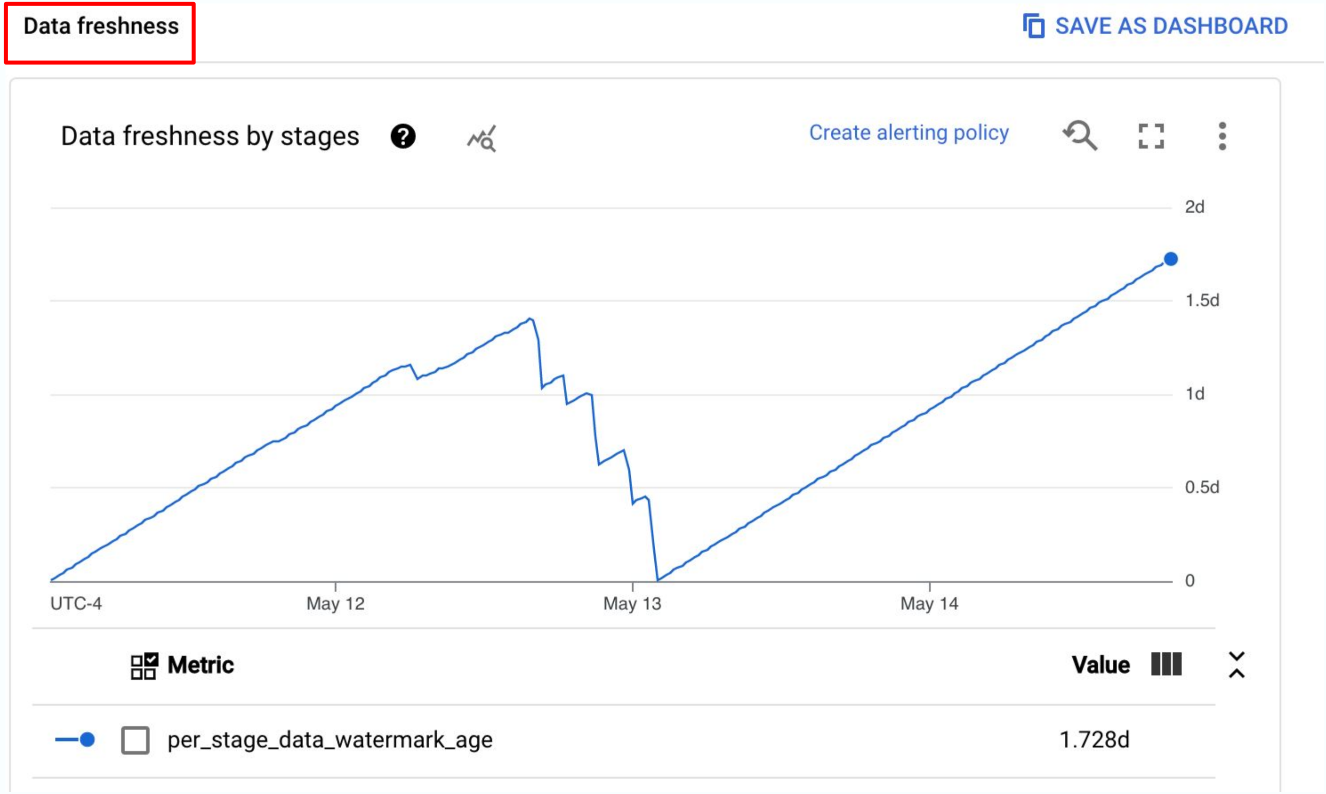


Troubleshoot slow/stuck dataflow jobs

High CPU Utilization

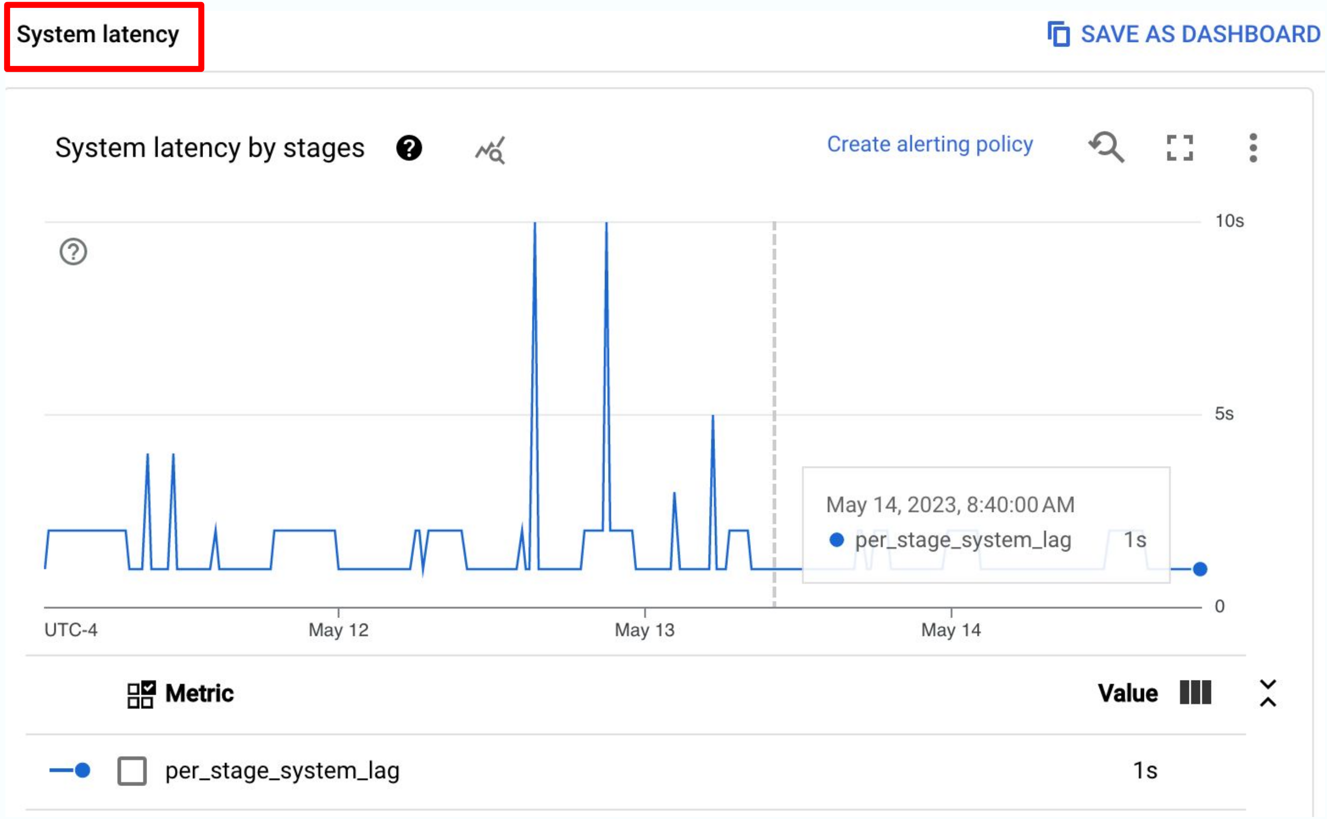
Troubleshoot slow/stuck dataflow jobs

Data Freshness



Troubleshoot slow/stuck dataflow jobs

System Latency



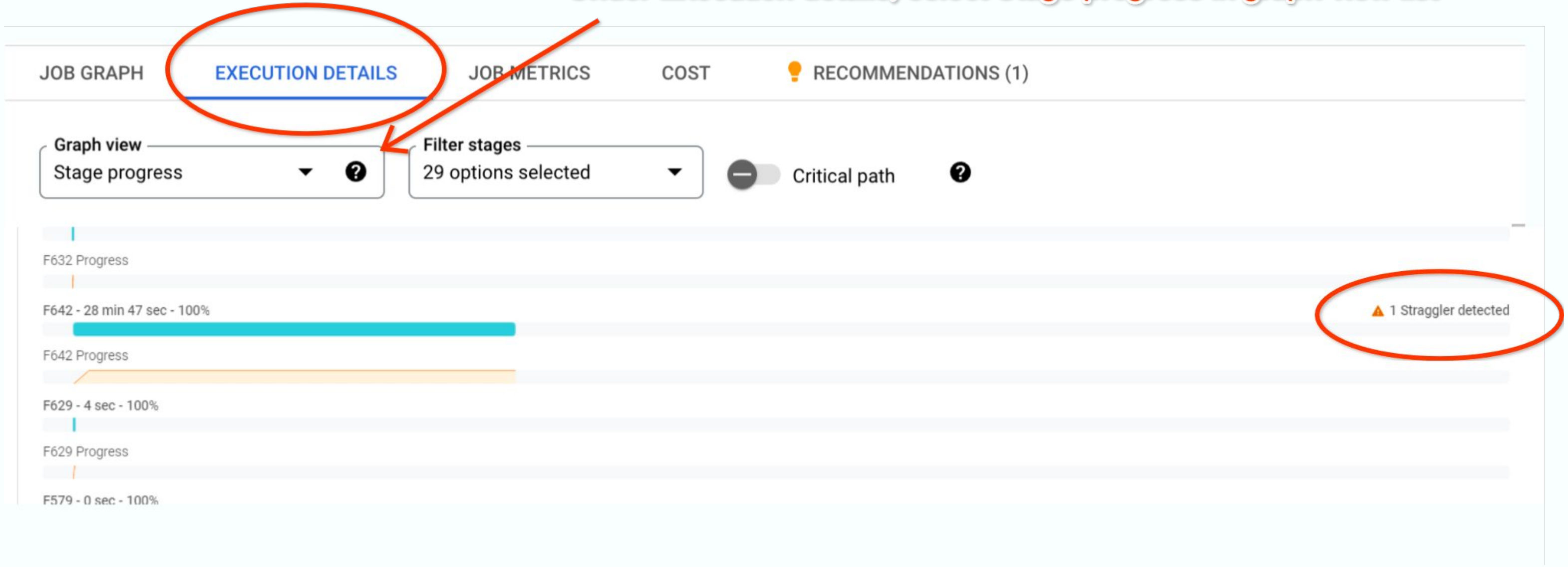
Stragglers in batch job

Troubleshoot slow/stuck dataflow jobs

When a batch job takes a long time to process data, it would be best to check on the Straggler Workers

How to check it?

Under Execution details, select Stage progress in graph view list



Troubleshoot slow/stuck dataflow jobs

There can be various causes of stragglers:

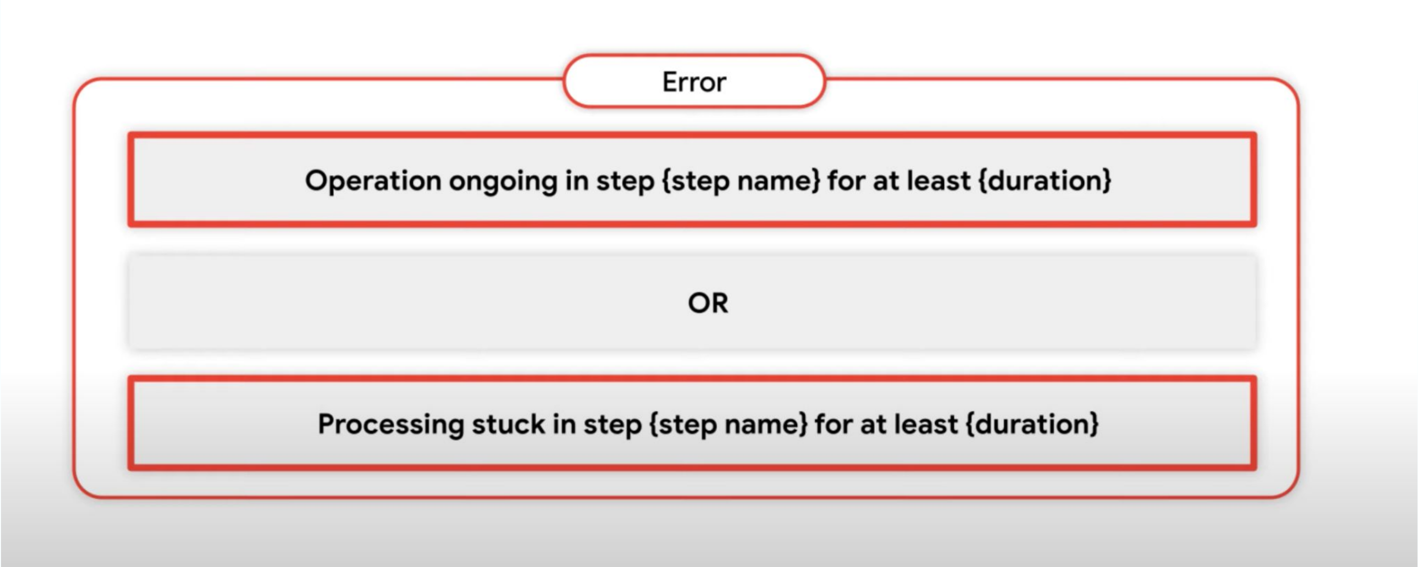
- **Hot Keys:** Hot keys can create stragglers because they limit ability of Dataflow to process elements in parallel.
  a. Re-key your data. Apply a ParDo transform to output new key-value pairs.

- **Re-shuffle your data** to avoid a single worker having extra load

Scenario 1: Long active user operation

Troubleshoot slow/stuck dataflow jobs

Processing Stuck/ Operation ongoing



Troubleshoot slow/stuck dataflow jobs

Processing Stuck/ Operation ongoing

From Logs Explorer

Query:

Query | Saved (0) | Suggested (2) | Library

Last 14 days | Search all fields

```
resource.type="dataflow_step"
resource.labels.job_id=$JOB_ID
logName:"/logs/dataflow.googleapis.com%2Fworker"
```

Results:

```
Operation ongoing in step Write to BQ/BatchLoads/SinglePartitionWriteTables/ParMultiDo(WriteTables) for at least 02h20m00s without outputting or completing
in state finish
  at java.base@11.0.9/java.lang.Thread.sleep(Native Method)
  at app//com.google.api.client.util.Sleeper$1.sleep(Sleeper.java:42)
  at app//com.google.api.client.util.BackOffUtils.next(BackOffUtils.java:48)
  at app//org.apache.beam.sdk.io.gcp.bigquery.BigQueryHelpers$PendingJobManager.nextBackOff(BigQueryHelpers.java:162)
  at app//org.apache.beam.sdk.io.gcp.bigquery.BigQueryHelpers$PendingJobManager.waitForDone(BigQueryHelpers.java:148)
  at app//org.apache.beam.sdk.io.gcp.bigquery.WriteTables$WriteTablesDoFn.finishBundle(WriteTables.java:380)
  at app//org.apache.beam.sdk.io.gcp.bigquery.WriteTables$WriteTablesDoFn$DoFnInvoker.invokeFinishBundle(Unknown Source)
```

Troubleshoot slow/stuck dataflow jobs

Processing Stuck/ Operation ongoing

From Logs Explorer

```
Operation ongoing in step Write to BQ/BatchLoads/SinglePartitionWriteTables/ParMultiDo(WriteTables) for at least 02h20m00s without outputting or completing
in state finish
  at java.base@11.0.9/java.lang.Thread.sleep(Native Method)
  at app//com.google.api.client.util.Sleeper$1.sleep(Sleeper.java:42)
  at app//com.google.api.client.util.BackOffUtils.next(BackOffUtils.java:48)
  at app//org.apache.beam.sdk.io.gcp.bigquery.BigQueryHelpers$PendingJobManager.nextBackOff(BigQueryHelpers.java:162)
  at app//org.apache.beam.sdk.io.gcp.bigquery.BigQueryHelpers$PendingJobManager.waitForDone(BigQueryHelpers.java:148)
  at app//org.apache.beam.sdk.io.gcp.bigquery.WriteTables$WriteTablesDoFn.finishBundle(WriteTables.java:380)
  at app//org.apache.beam.sdk.io.gcp.bigquery.WriteTables$WriteTablesDoFn$DoFnInvoker.invokeFinishBundle(Unknown Source)
```

https://github.com/apache/beam/blob/master/sdks/java/io/google-cloud-platform/src/main/java/org/apache/beam/sdk/io/gcp/bigquery/BigQueryHelpers.java

Troubleshoot slow/stuck dataflow jobs

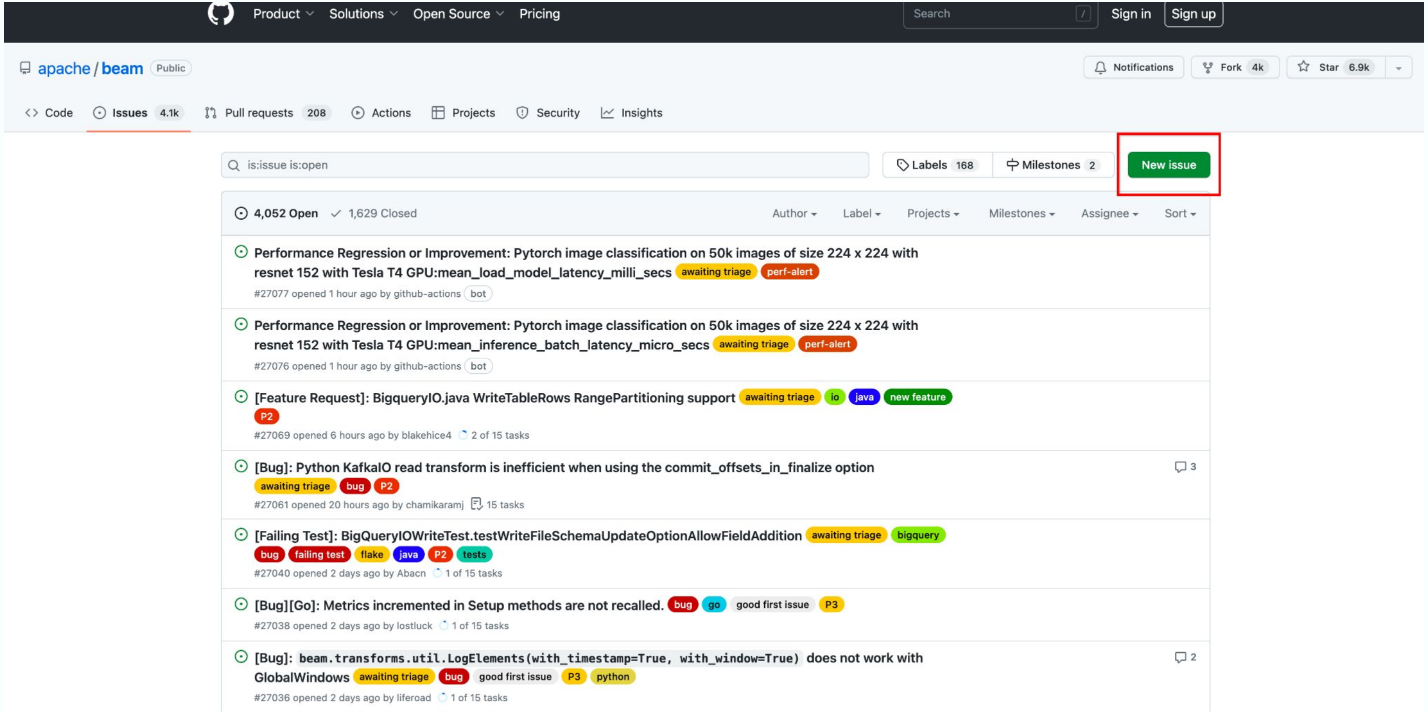
Processing Stuck/ Operation ongoing

49% of results are similar and can be hidden. Hide similar entries Preview

SEVERITY	TIMESTAMP EDT	SUMMARY EDIT
i	2023-05-23 19:04:08.886 EDT	Detected missing event columns in [redacted] BigQuery schema. Schema must be updated manually, if required. Dropping/Missing attributes from Event payload. Details
i	2023-05-23 19:04:08.886 EDT	Detected missing event columns in [redacted] BigQuery schema. Schema must be updated manually, if required. Dropping/Missing attributes from Event payload. Details
i	2023-05-23 19:04:08.886 EDT	No BigQuery job with job id beam_bq_job_LOAD_[redacted]00001_0000
i	2023-05-23 19:04:08.886 EDT	job id beam_bq_job_LOAD_[redacted]00001_00000-72 not found, so …
!	2023-05-23 19:04:08.886 EDT	Load job beam_bq_job_LOAD[redacted]_00001_00000-71 failed, will…
i	2023-05-23 19:04:08.886 EDT	Job beam_bq_job_LOAD_[redacted]00001_00000-72 pending. retrying.

Troubleshoot slow/stuck dataflow jobs

Apache Beam Issues/Feature Request



Scenario 2: GC Thrashing/OOM

Troubleshoot slow/stuck dataflow jobs

GC Thrashing/OOM: Diagnostics Tab

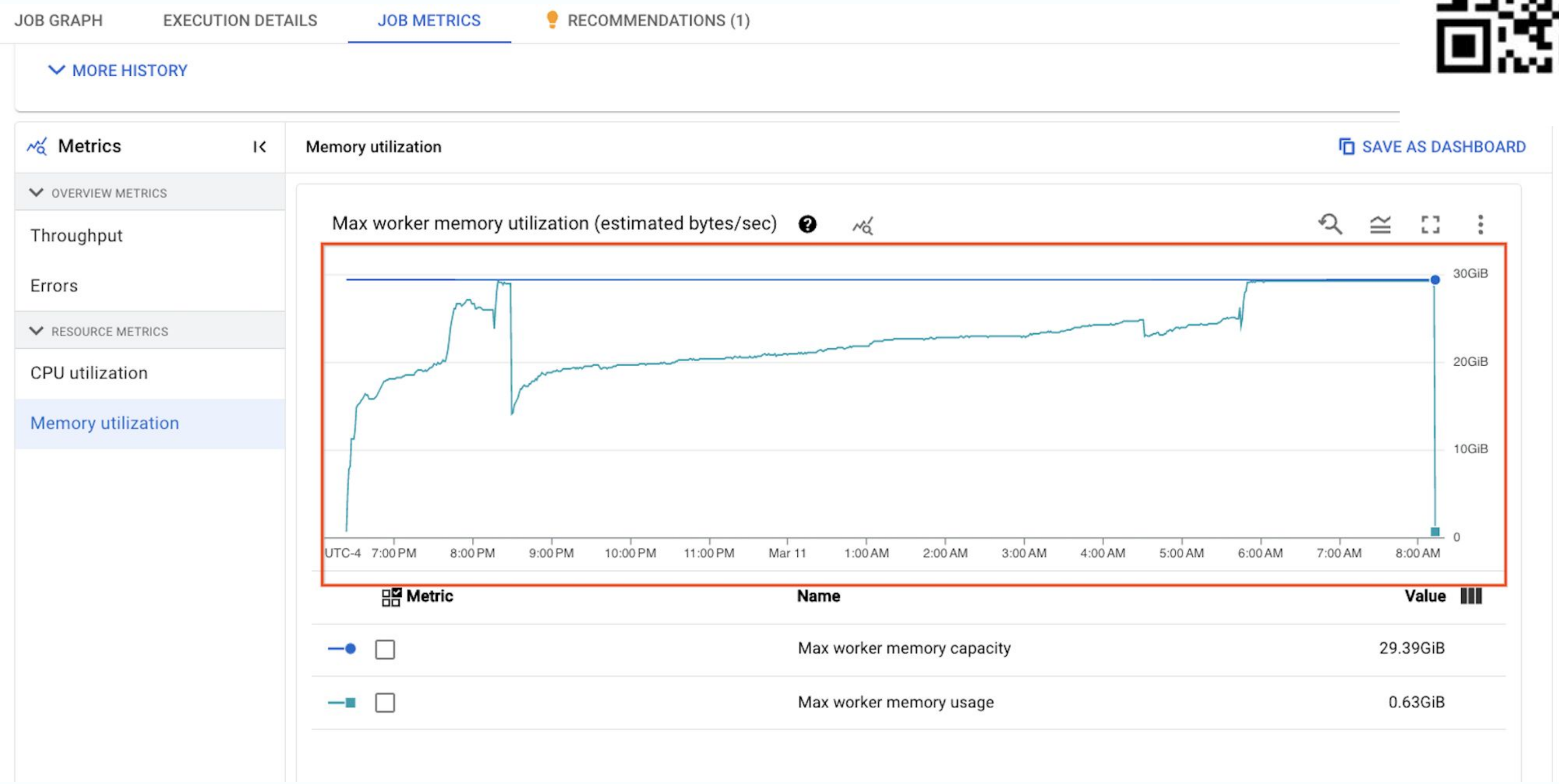
Logs HIDE 30

JOB LOGS | WORKER LOGS | DIAGNOSTICS

Occurrences	Count	Error	First
	8	Shutting down JVM after 8 consecutive periods of measured GC thrashing. Memory is used/total/max = 7904/20103/37513 MB, GC last/max = 90.03/95.7... The worker was shut down after a long period of high memory pressure.	Dec 2022
	1	StatusRuntimeException: UNAVAILABLE: keepalive watchdog timeout	Jan 2023

Troubleshoot slow/stuck dataflow jobs

GC Thrashing/OOM

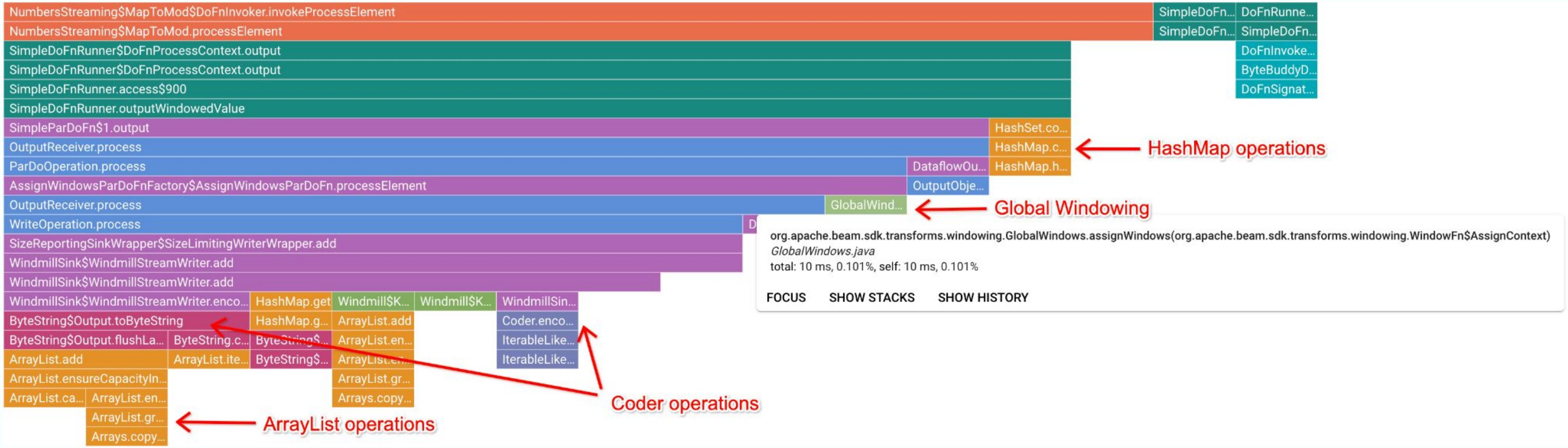


General Recommendations

- Use machine types with higher memory
  - **Link: goo.gle/45USWe3**
- Decrease the parallelism of processing by reducing the number of worker harness threads
  - **Link: goo.gle/45RM6WT**
- Do vertical autoscaling (Enable Dataflow Prime)
  - **Link: goo.gle/3r3KZjv**

Performance Optimization using Dataflow profiling

- Cloud Profiler is available for Dataflow pipelines written in Apache Beam SDK for Java and Python, version 2.33.0 or later.
- It can be enabled at pipeline start time
- E.g. For Java SDK, to enable CPU profiling, start the pipeline with the following option:
  --dataflowServiceOptions=enable_google_cloud_profiler



QUESTIONS?

mhkgupta@google.com

linkedin.com/in/mhkgupta

BEAM

SUMMIT

Resolving out of memory issues in Beam Pipelines

Zeeshan Khan

BEAM SUMMIT
NYC 2023

BEAM
SUMMIT

Benchmarking Beam pipelines on Dataflow

Pranav Bhandari

BEAM SUMMIT
NYC 2023

BEAM
SUMMIT