# BEAM SUMMIT

# Dataflow Streaming

What's New & What's Next?

Iñigo San Jose, Tom Stepp

Google

BEAM SUMMIT

NYC 2023

## Agenda

- Overview
- Autotuning
- GCP PubSub Integration
- Observability
- Other Projects

## Overview

## Overview: Streaming @ Google

- History of Streaming @ Google
- Streaming Appliance vs Streaming Engine
- Streaming Basics

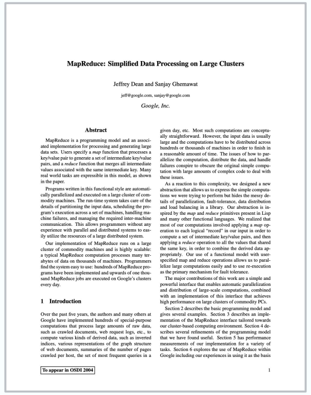
## History of Streaming @ Google

- Everything was batch
- MapReduce
- First streaming systems were designed for Ads
- Streaming MapReduce
- MillWheel
- Streaming Flume
- Windmill (Dataflow)

# History of Streaming @ Google

MapReduce

**2004**



MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat

jeff@google.com, sanjay@google.com

*Google, Inc.*

Abstract

[illegible]

**Simple distributed data processing**

Flume

**2010**

FlumeJava: Easy, Efficient Data-Parallel Pipelines

Craig Chambers, Ashish Raniwala, Frances Perry,
Stephen Adams, Robert R. Henry,
Robert Bradshaw, Nathan Weizenbaum

Google, Inc.

{chambers,raniwala,fjp,sra,rrh,robertwb,nweiz}@google.com

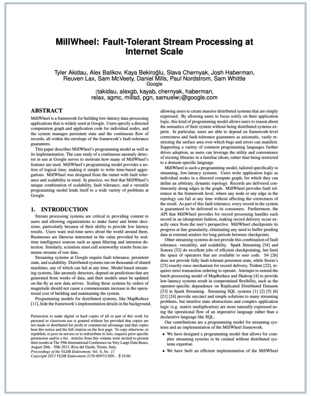
Abstract

[illegible]

**Logical pipelines & optimization**

Millwheel

**2013**



MillWheel: Fault-Tolerant Stream Processing at Internet Scale

Tyler Akidau, Alex Balikov, Kaya Bekiroğlu, Slava Chernyak, Josh Haberman,
Reuven Lax, Sam McVeety, Daniel Mills, Paul Nordstrom, Sam Whittle

Google

{takidau, alexgb, kayab, chernyak, haberman,
relax, sgmc, millsd, pgn, samuelw}@google.com

ABSTRACT

[illegible]

**Low-latency streaming**

Cloud Dataflow
+
Apache Beam

**2015**

The Dataflow Model: A Practical Approach to Balancing Correctness, Latency, and Cost in Massive-Scale, Unbounded, Out-of-Order Data Processing

Tyler Akidau, Robert Bradshaw, Craig Chambers, Slava Chernyak,
Rafael J. Fernández-Moctezuma, Reuven Lax, Sam McVeety, Daniel Mills,
Frances Perry, Eric Schmidt, Sam Whittle

Google

{takidau, robertwb, chambers, chernyak, rfernand,
relax, sgmc, millsd, fjp, cloude, samuelw}@google.com

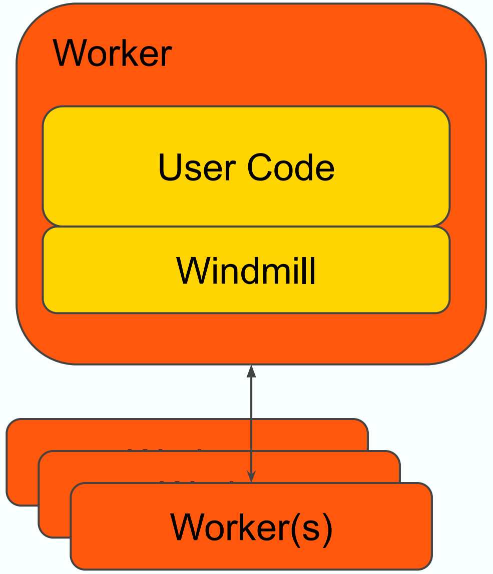
ABSTRACT

[illegible]

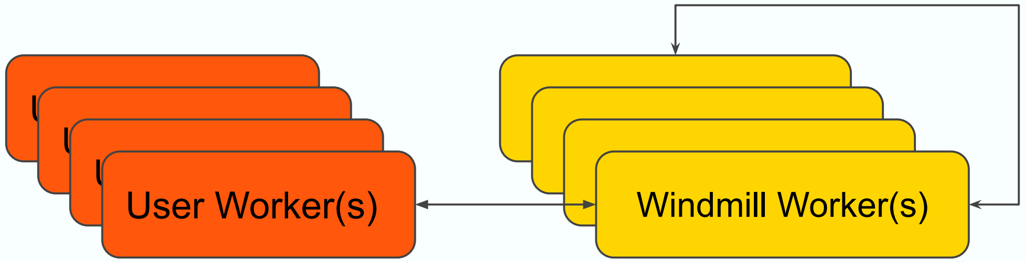
**Batch + Streaming Serverless Cloud**

# Streaming Engine vs Streaming Appliance

## Streaming Appliance

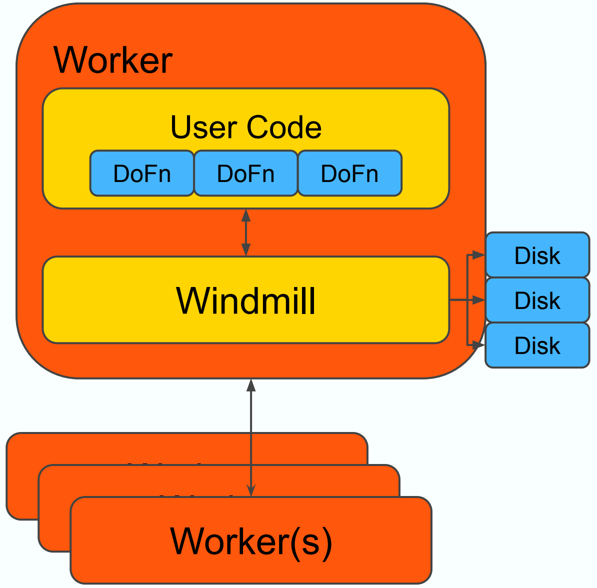


## Streaming Engine

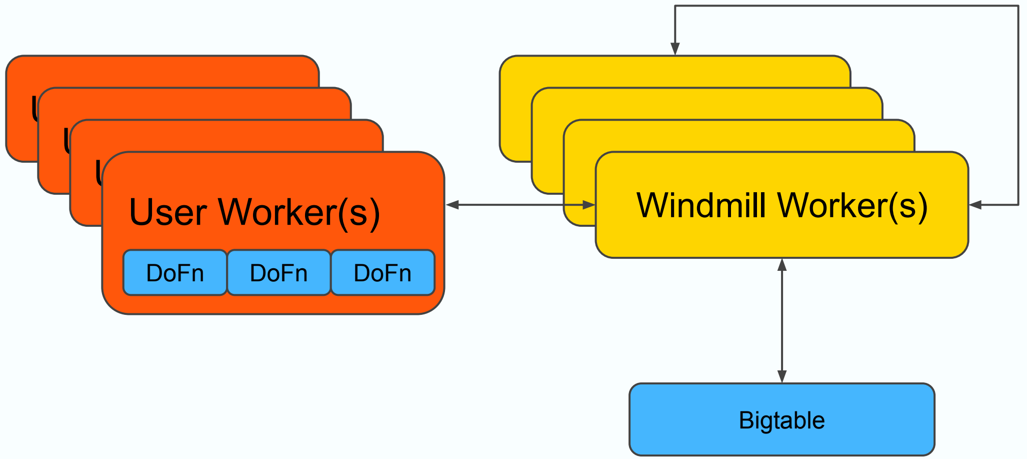


# Streaming Engine vs Streaming Appliance

## Streaming Appliance



## Streaming Engine



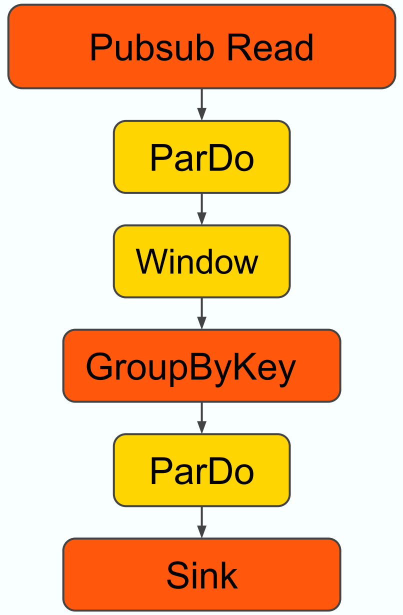
## Streaming Engine vs Streaming Appliance

Benefits of Streaming Engine:

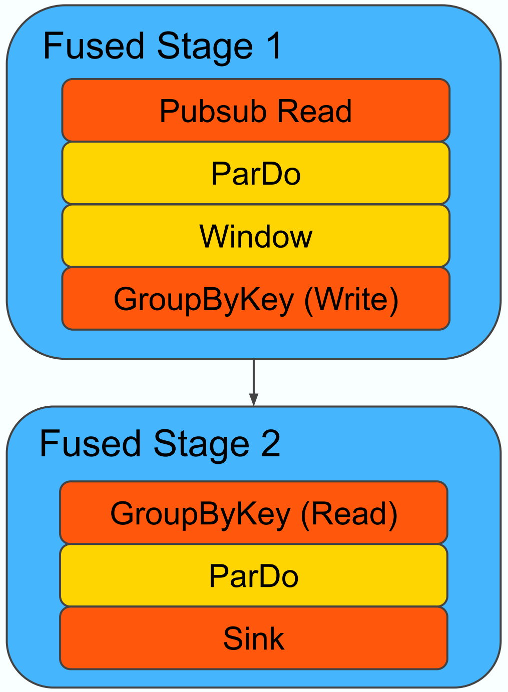
- More efficient use of User Workers
- No need for Persistent Disks
- More responsive Horizontal Autoscaling
- Improved supportability and visibility

# Streaming Basics

## Pipeline example

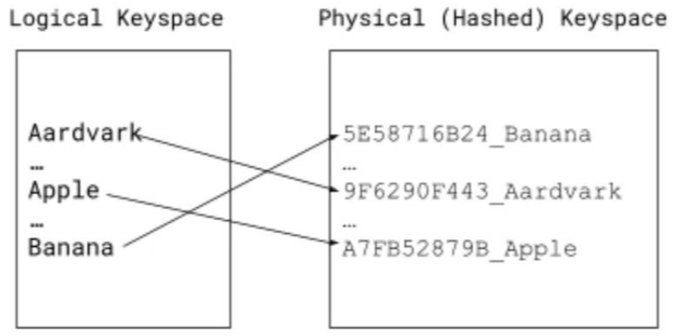


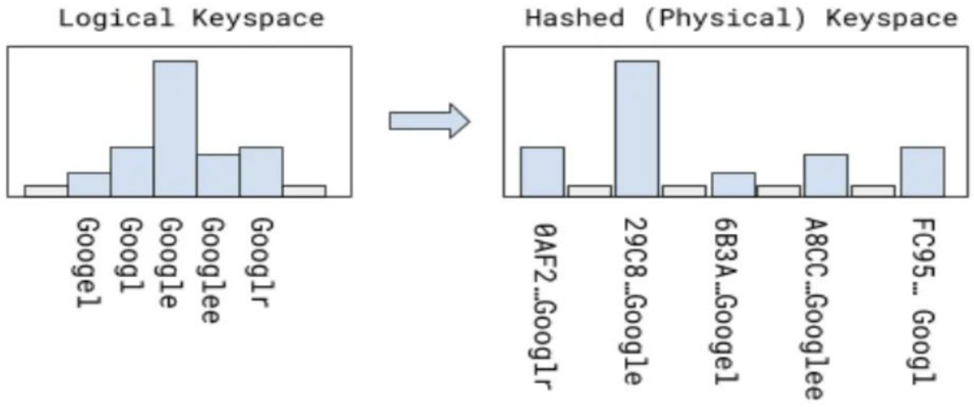
## What Dataflow Streaming Sees



## Streaming Basics

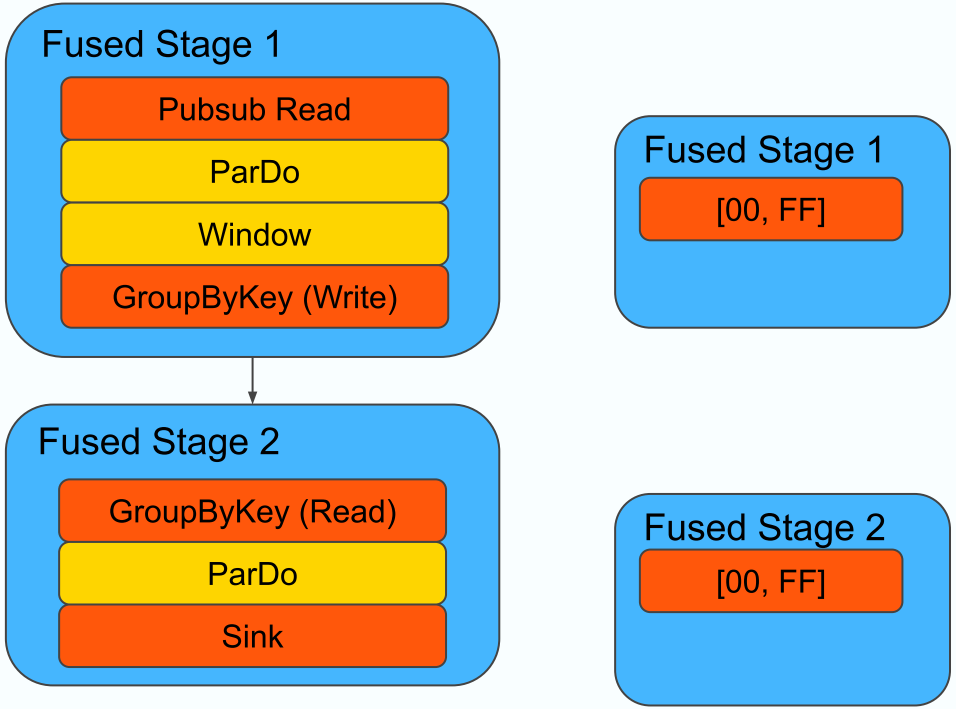
- Every message has a key assigned to it
- Keys can be user defined or system defined
- Keys are hashed
- Elements are processed in the context of a key
- Keys are the basic unit of parallelism





## Streaming Basics

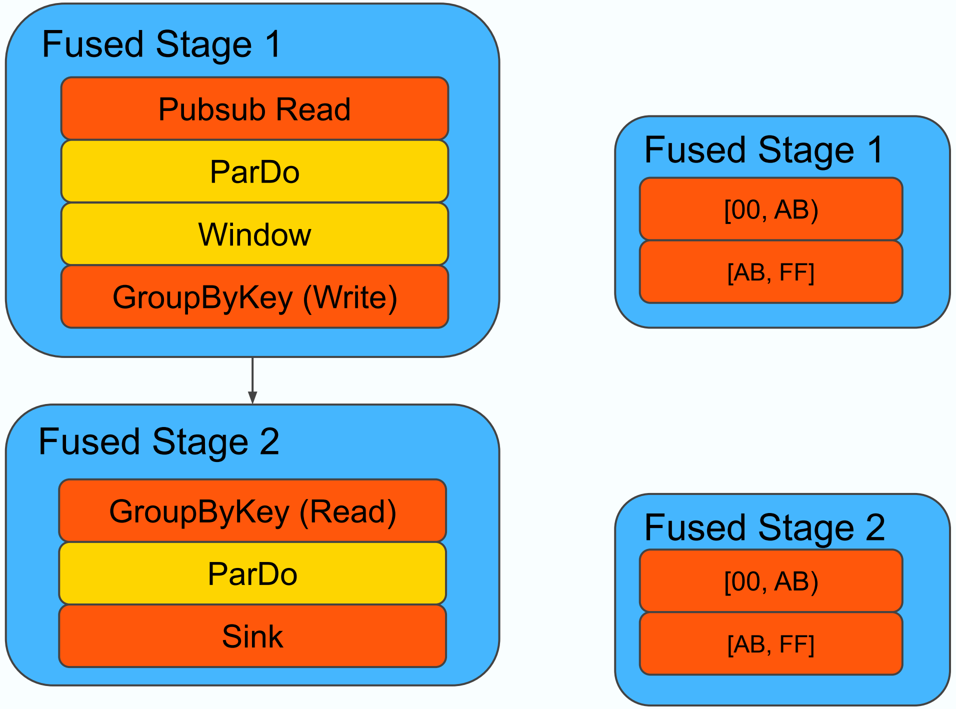
- Keys belong to key-ranges
- Key ranges are assigned to workers
- Key ranges can be split and sent to different workers



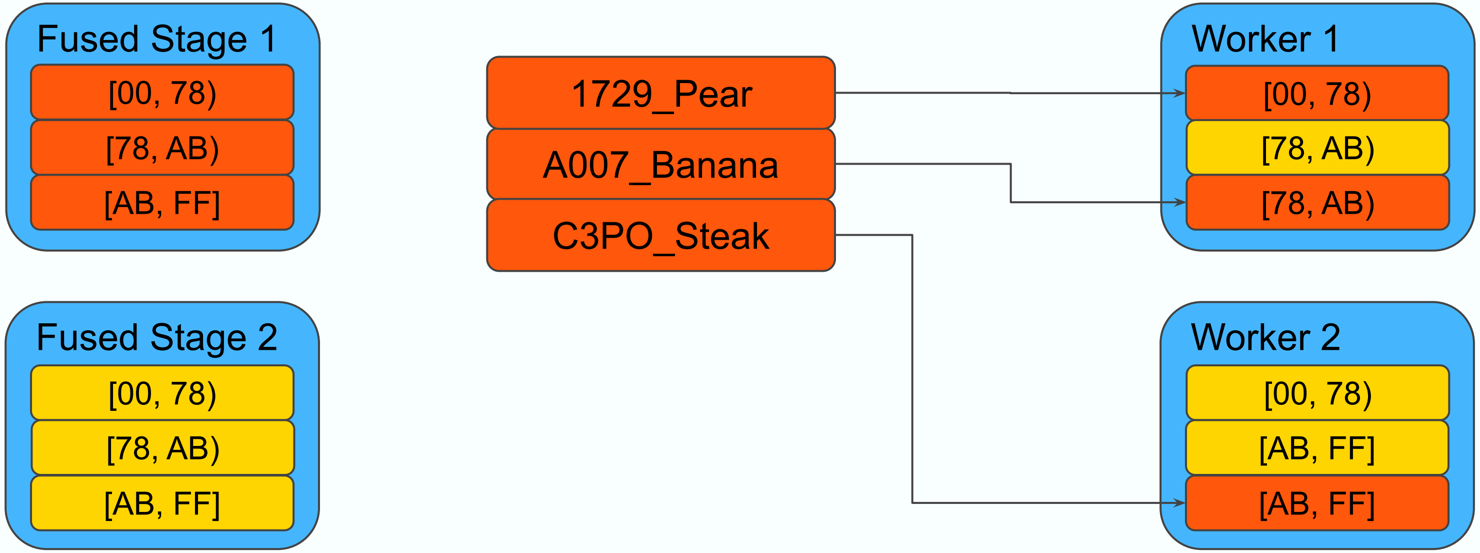
*NOTE: all range boundaries are hexadecimal values.*

## Streaming Basics

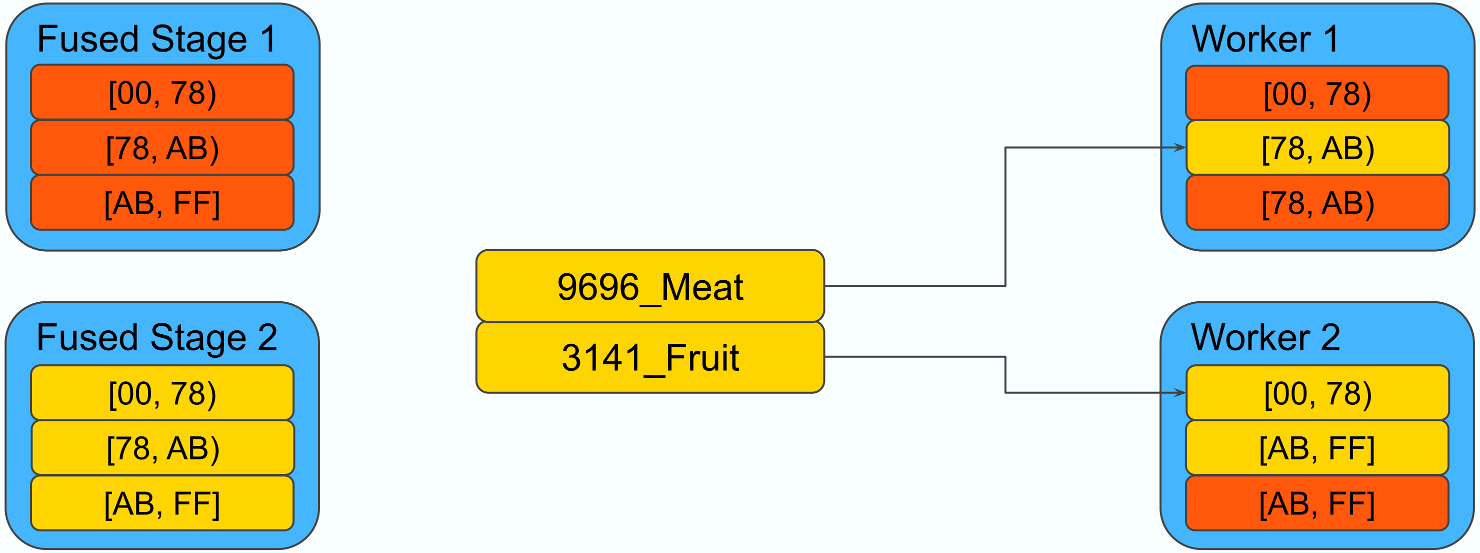
- Keys belong to key-ranges
- Key ranges are assigned to workers
- Key ranges can be split and sent to different workers



## Streaming Basics



## Streaming Basics



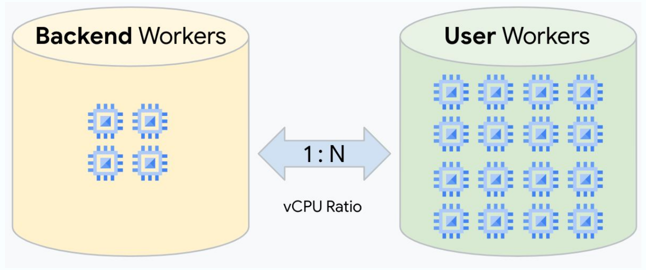
## Autotuning

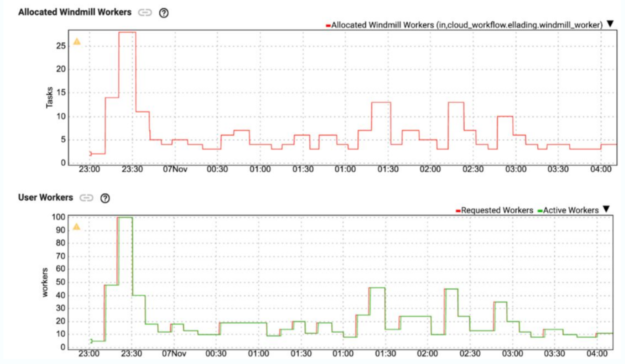
Launched

## Autotuning: Asymmetric Autoscaling

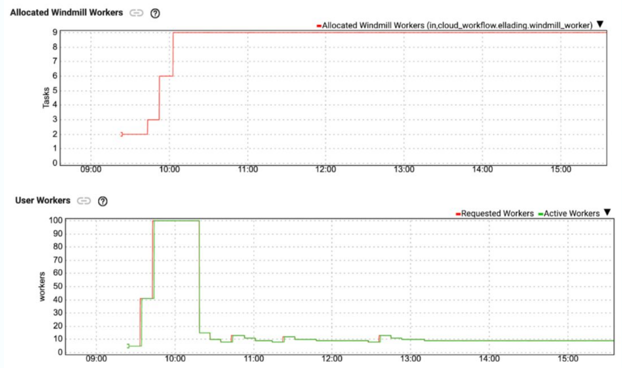
**Past:** Scaling backend workers linearly with user workers.

**Present:** Scaling each worker pool independently.





Baseline

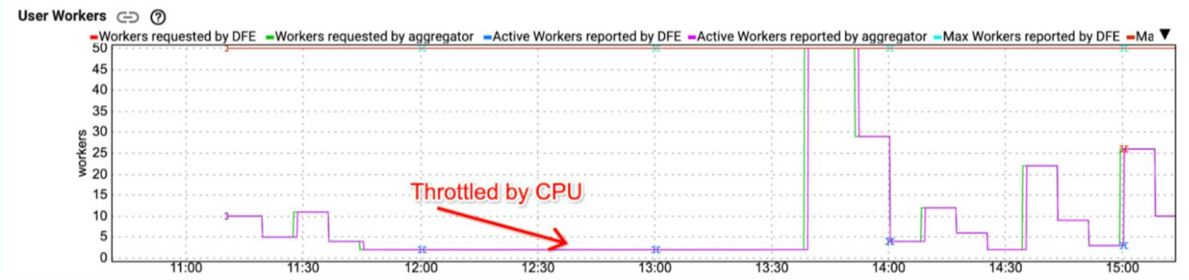


Asymmetric

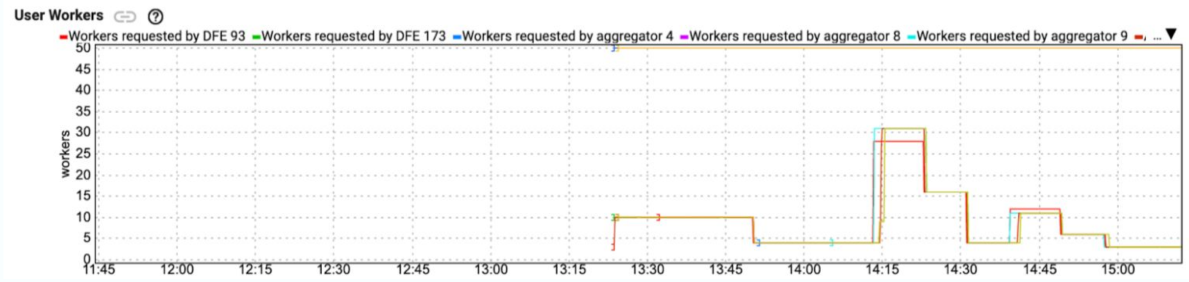
Launched

## Autotuning: Key-Based Throttling

**Past**: Unconditionally throttling user worker upscale if < 20% CPU utilization.



**Present**: Throttle user worker upscale on key parallelism limits (number of keys).

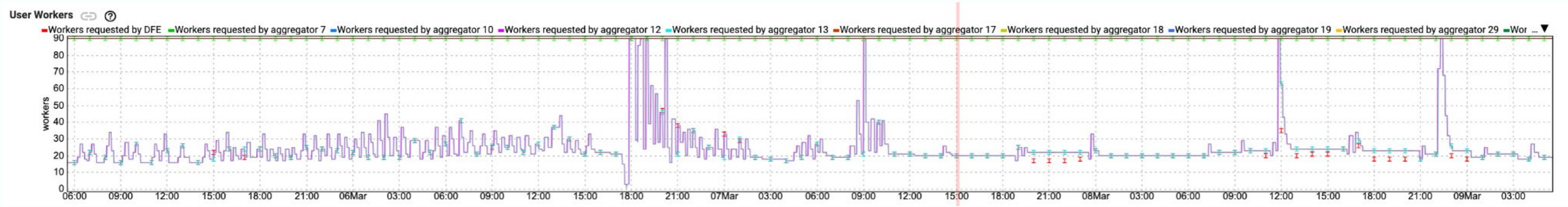


In Progress

# Autotuning: Downscale Dampening

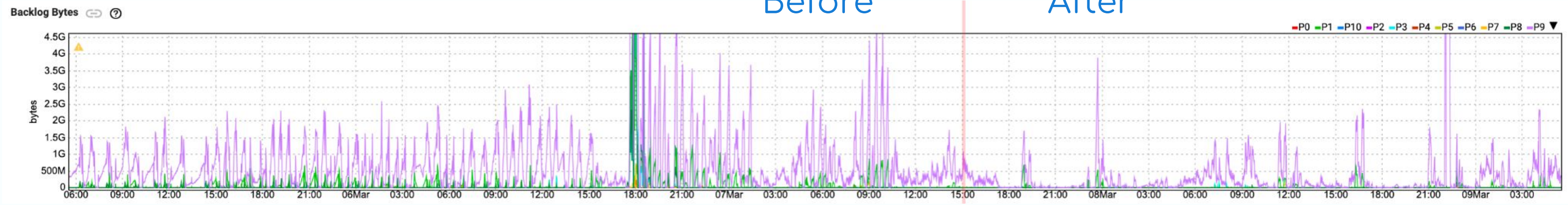
**Past:** Only consider the current state (backlog, throughput, etc.)

**Present:** Track scaling frequencies, downscale slower when yo-yoing detected (frequent up/down scaling in short time frame).



Before

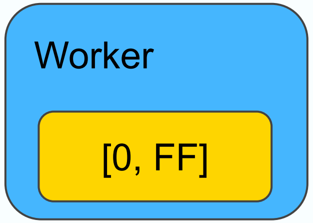
After



## Autotuning: Scaling Actuation Latencies

**Past:** When autoscale events happen, new workers need to load the pipeline state from persistence. This can take time and lead to backlog and latency.

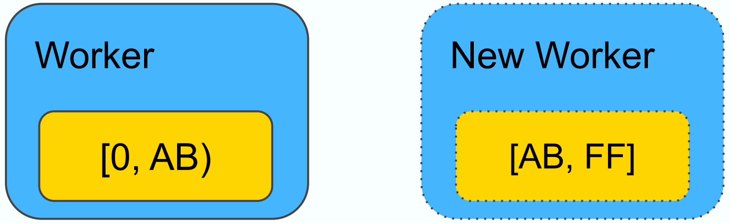
**Present:** Transfer info directly from workers, reducing latency



## Autotuning: Scaling Actuation Latencies

**Past**: When autoscale events happen, new workers need to load the pipeline state from persistence. This can take time and lead to backlog and latency.

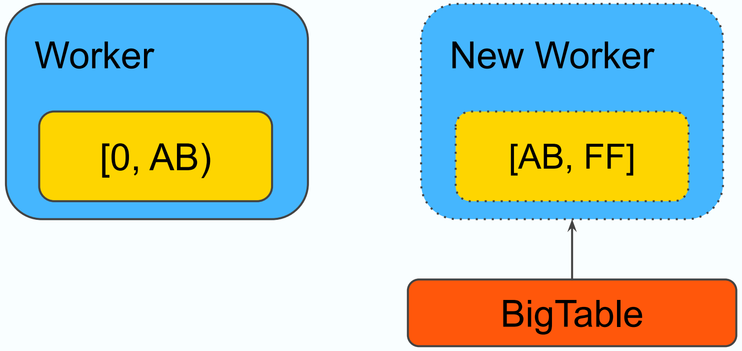
**Present**: Transfer info directly from workers, reducing latency



## Autotuning: Scaling Actuation Latencies

**Past**: When autoscale events happen, new workers need to load the pipeline state from persistence. This can take time and lead to backlog and latency.

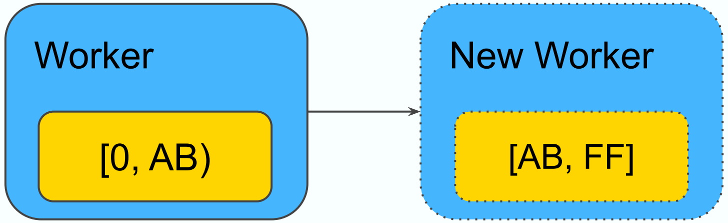
**Present**: Transfer info directly from workers, reducing latency



## Autotuning: Scaling Actuation Latencies

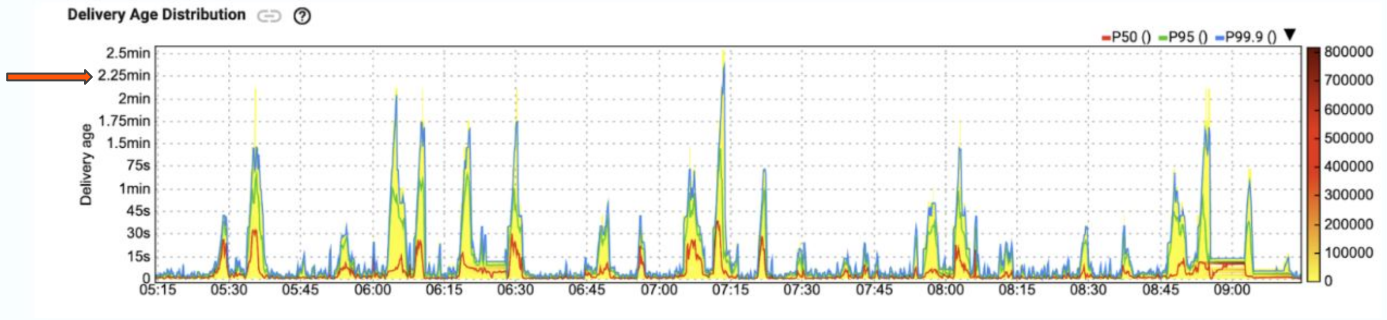
**Past:** When autoscale events happen, new workers need to load the pipeline state from persistence. This can take time and lead to backlog and latency.

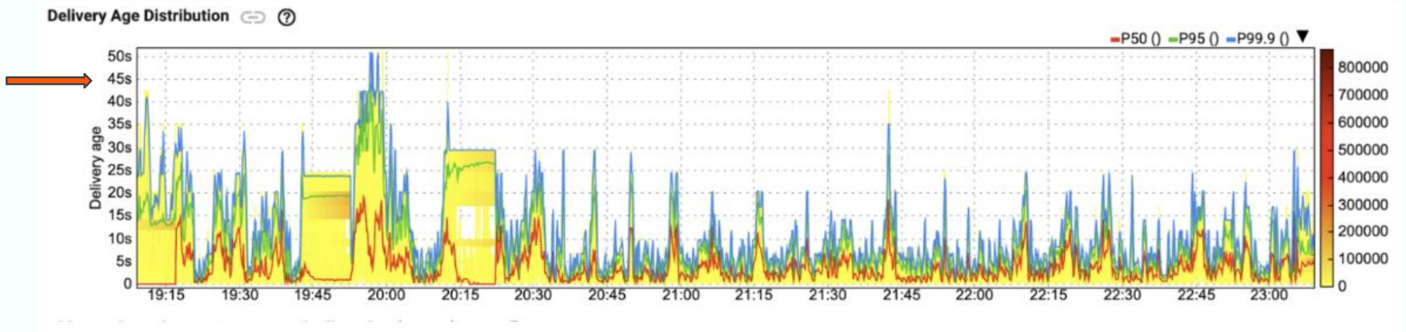
**Present:** Transfer info directly from workers, reducing latency



# Autotuning: Scaling Actuation Latencies

Latency Disabled (top) vs Enabled (bottom)

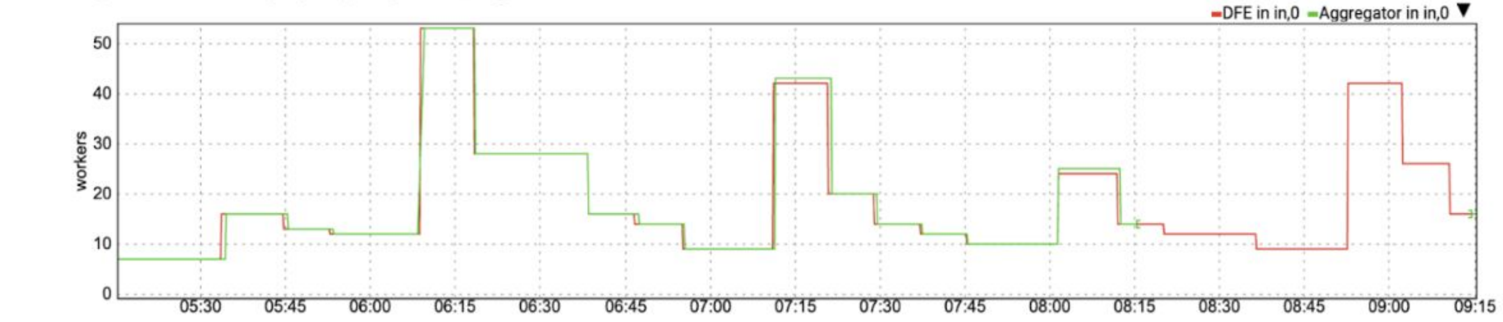




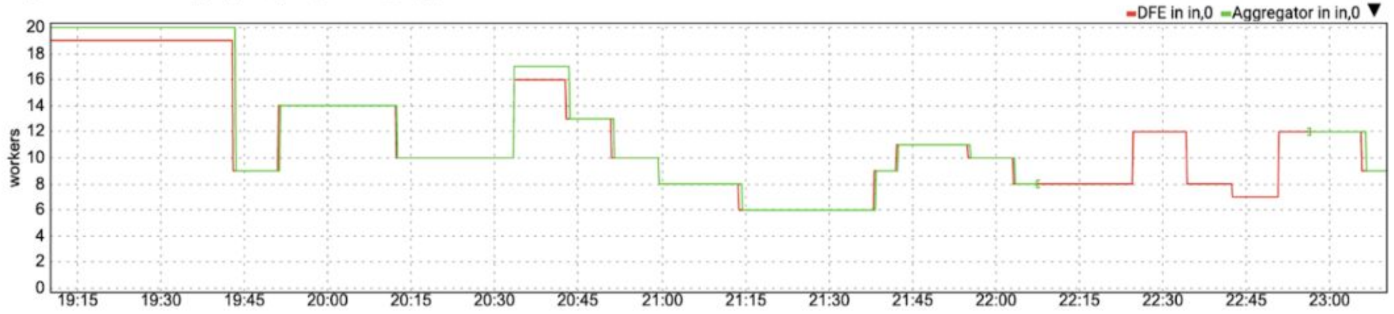
# Autotuning: Scaling Actuation Latencies

User Workers Disabled vs Enabled

Autoscaling recommendations grouped by Borg task



Autoscaling recommendations grouped by Borg task

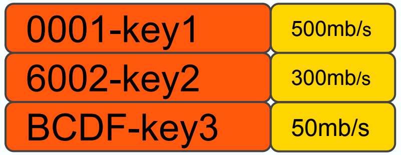


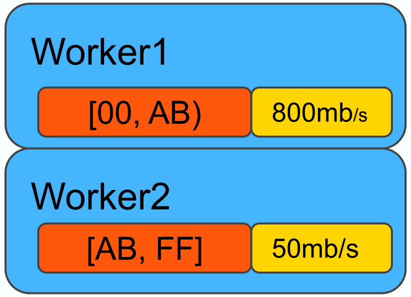
In Progress

## Autotuning: Range Rebalancing

**Past:** If a key range has a disproportionate amount of input rate, its worker would have more load than others, potentially accumulating backlog and wasting resources on other workers.

**Present:** We can split key ranges dynamically and rebalance them across workers based on their throughput

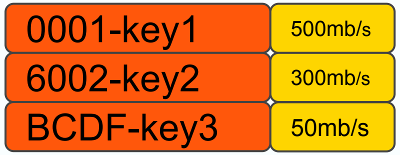


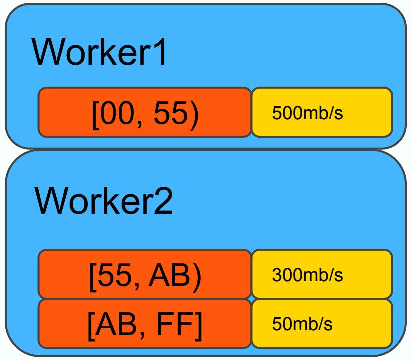


## Autotuning: Range Rebalancing

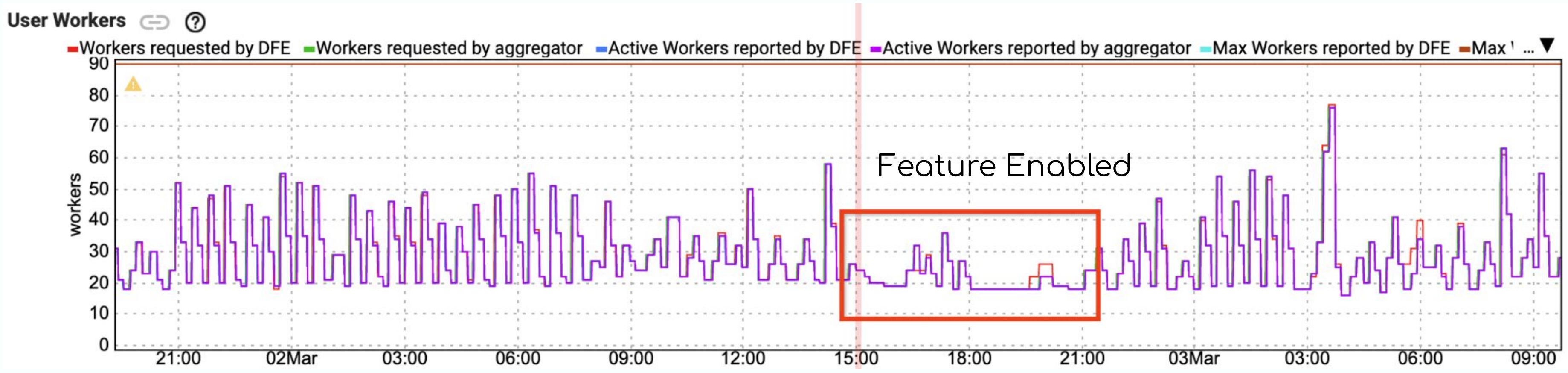
**Past:** If a key range has a disproportionate amount of input rate, its worker would have more load than others, potentially accumulating backlog and wasting resources on other workers.

**Present:** We can split key ranges dynamically and rebalance them across workers based on their throughput





# Autotuning: Range Rebalancing



In Progress

## Autotuning: BigQuery Autosharding

**Past:** Autosharding was only available for Streaming Inserts / File Loads and was load agnostic, which could lead to wasted resources in case of dynamic destinations

**Present:** StorageAPI gets autosharding option, using backlog and throughput as metric.

| | | |
|---|---|---|
| Table 1 | 200mb/s | 1000 shards |
| Table 2 | 100mb/s | 1000 shards |
| Table 3 | 1mb/s | 1000 shards |

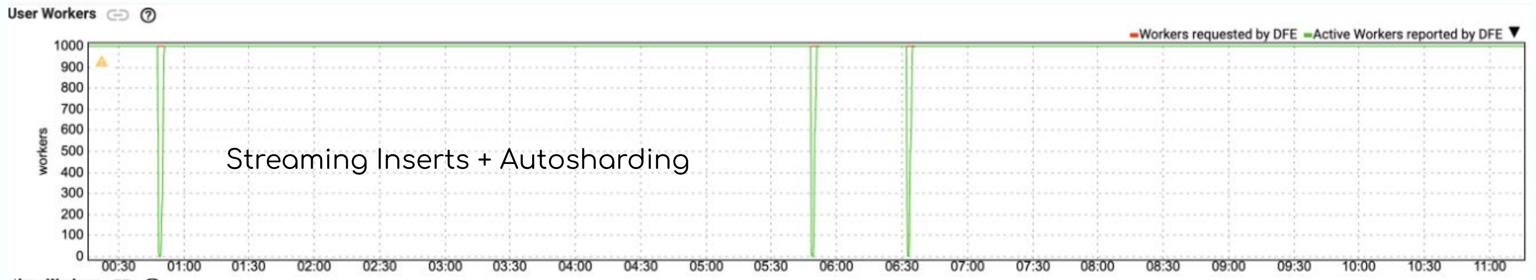
## Autotuning: BigQuery Autosharding

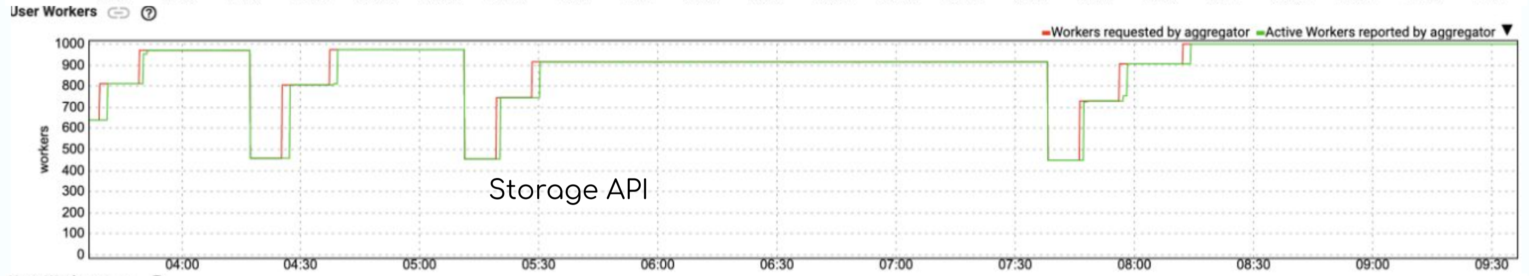
**Past**: Autosharding was only available for Streaming Inserts / File Loads and was load agnostic, which could lead to wasted resources in case of dynamic destinations

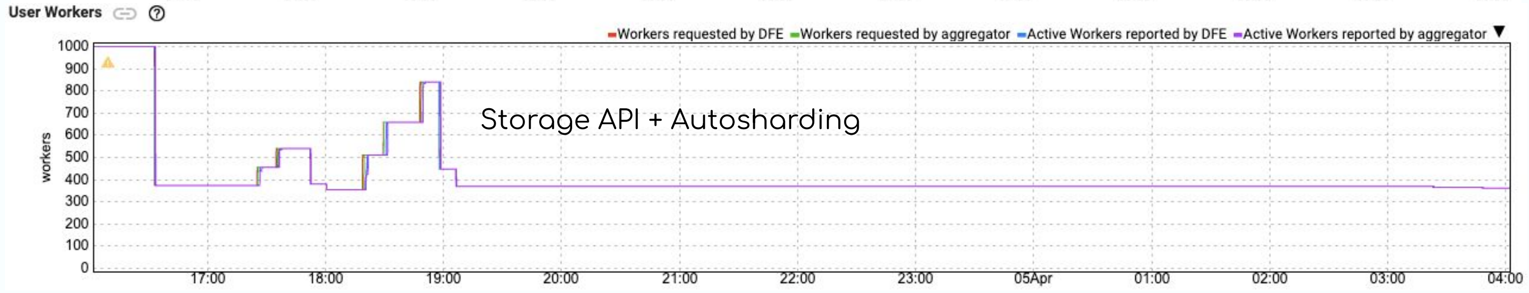
**Present**: StorageAPI gets autosharding option, using backlog and throughput as metric.

| | | |
|---|---|---|
| Table 1 | 200mb/s | 800 shards |
| Table 2 | 100mb/s | 400 shards |
| Table 3 | 1mb/s | 4 shards |

# Autotuning: BigQuery Autosharding







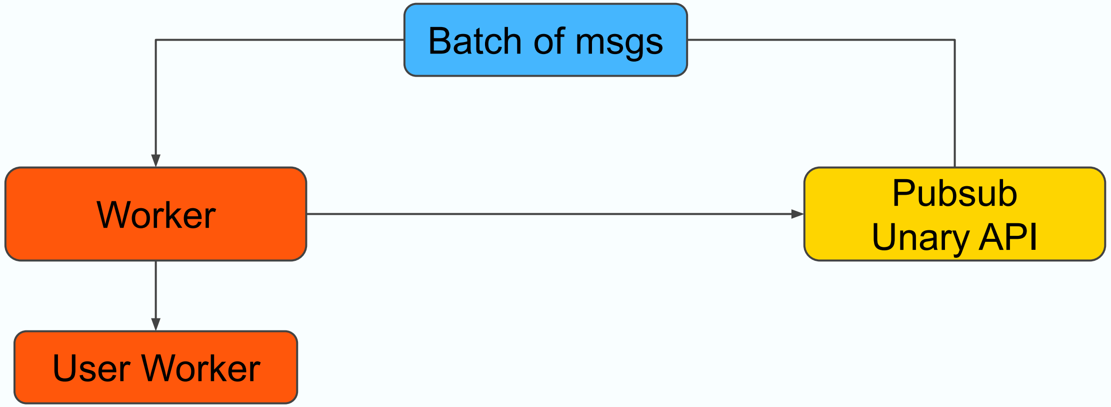
## GCP PubSub Integration

Launched

## PubSub Streaming Pull

**Past**: Pipelines used old Pubsub API Unary Pull

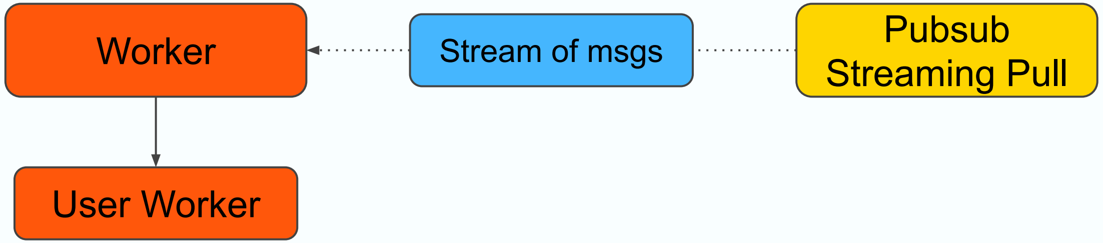
**Present**: Pipelines use newer Pubsub API Streaming Pull, improving throughput and latency



## PubSub Streaming Pull

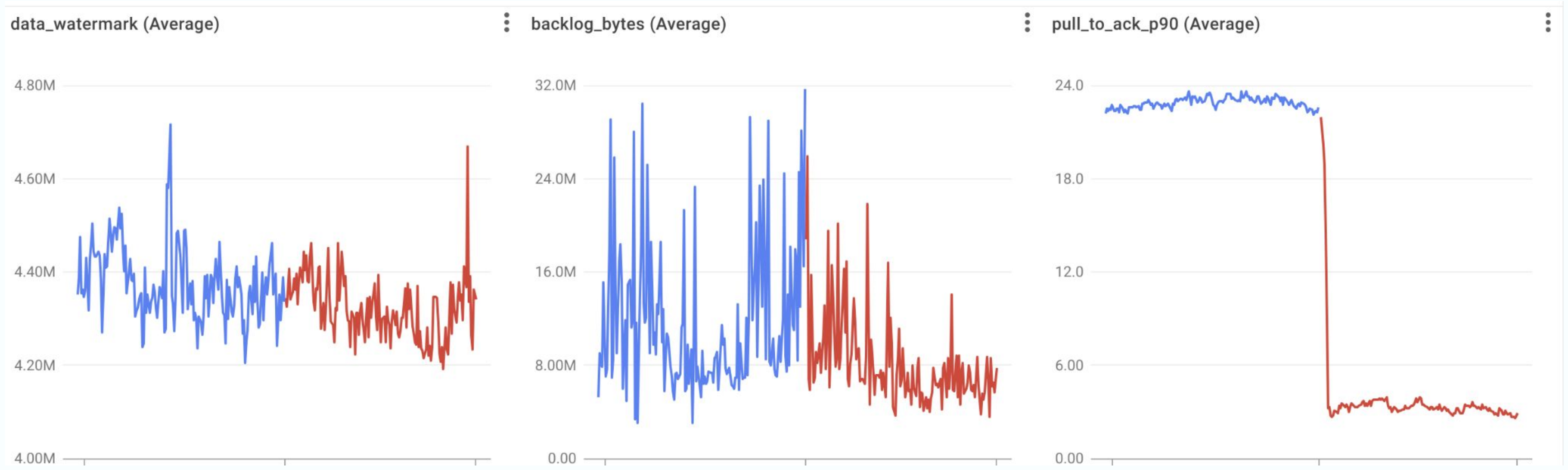
**Past:** Pipelines used old Pubsub API Unary Pull

**Present:** Pipelines use newer Pubsub API Streaming Pull, improving throughput and latency



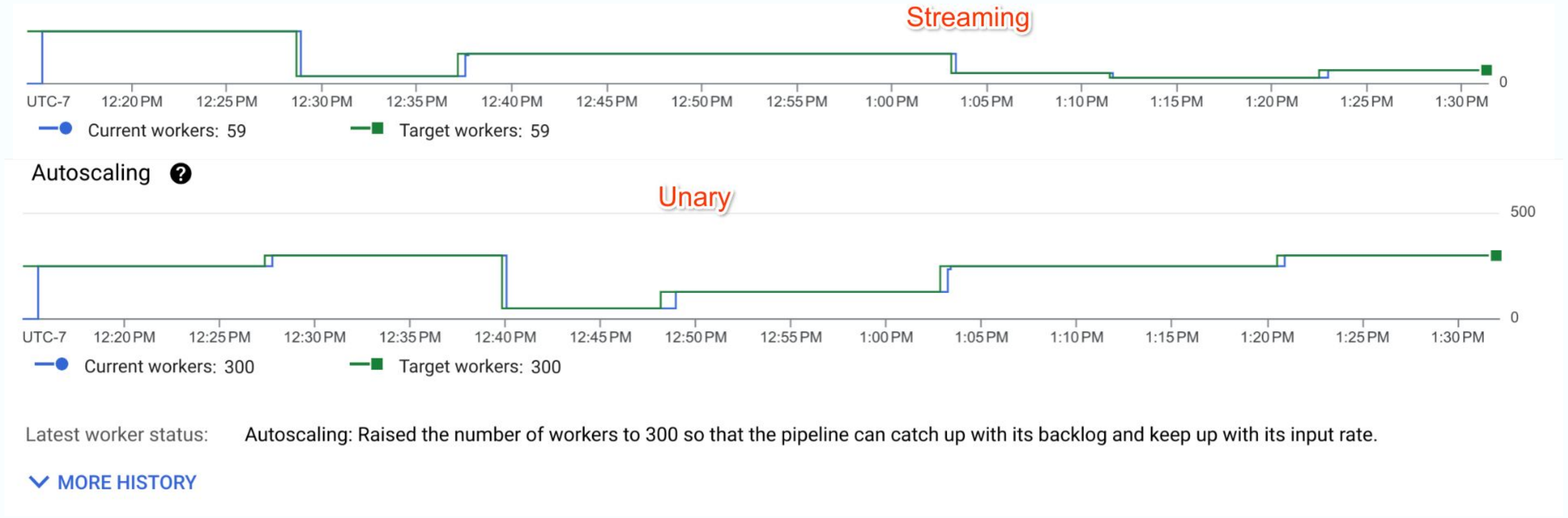
## PubSub Streaming Pull

### Latency and Backlog Improvements



# PubSub Streaming Pull

## Usage improvements



Latest worker status: Autoscaling: Raised the number of workers to 300 so that the pipeline can catch up with its backlog and keep up with its input rate.

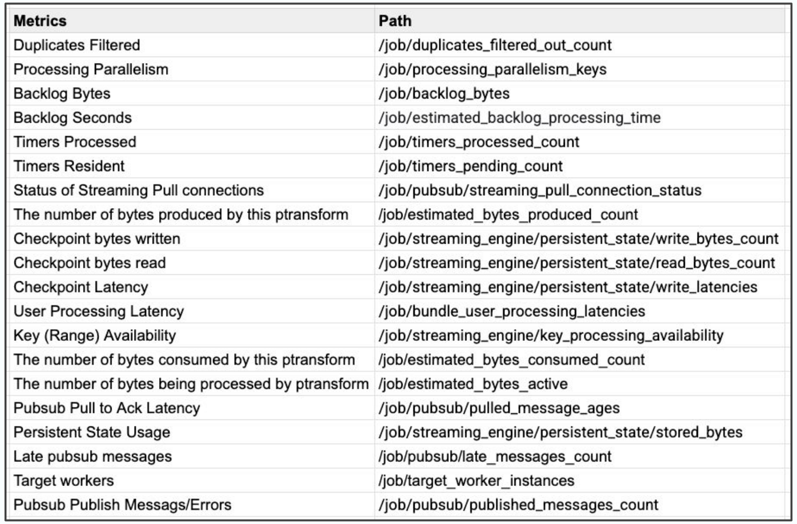
## Observability

Launched

## Observability: New Metrics

- Collecting many new Streaming Engine metrics
- Some integrated into Dataflow UI
- All available in Monitoring UI
- Available dashboard template for easy detailed job performance monitoring

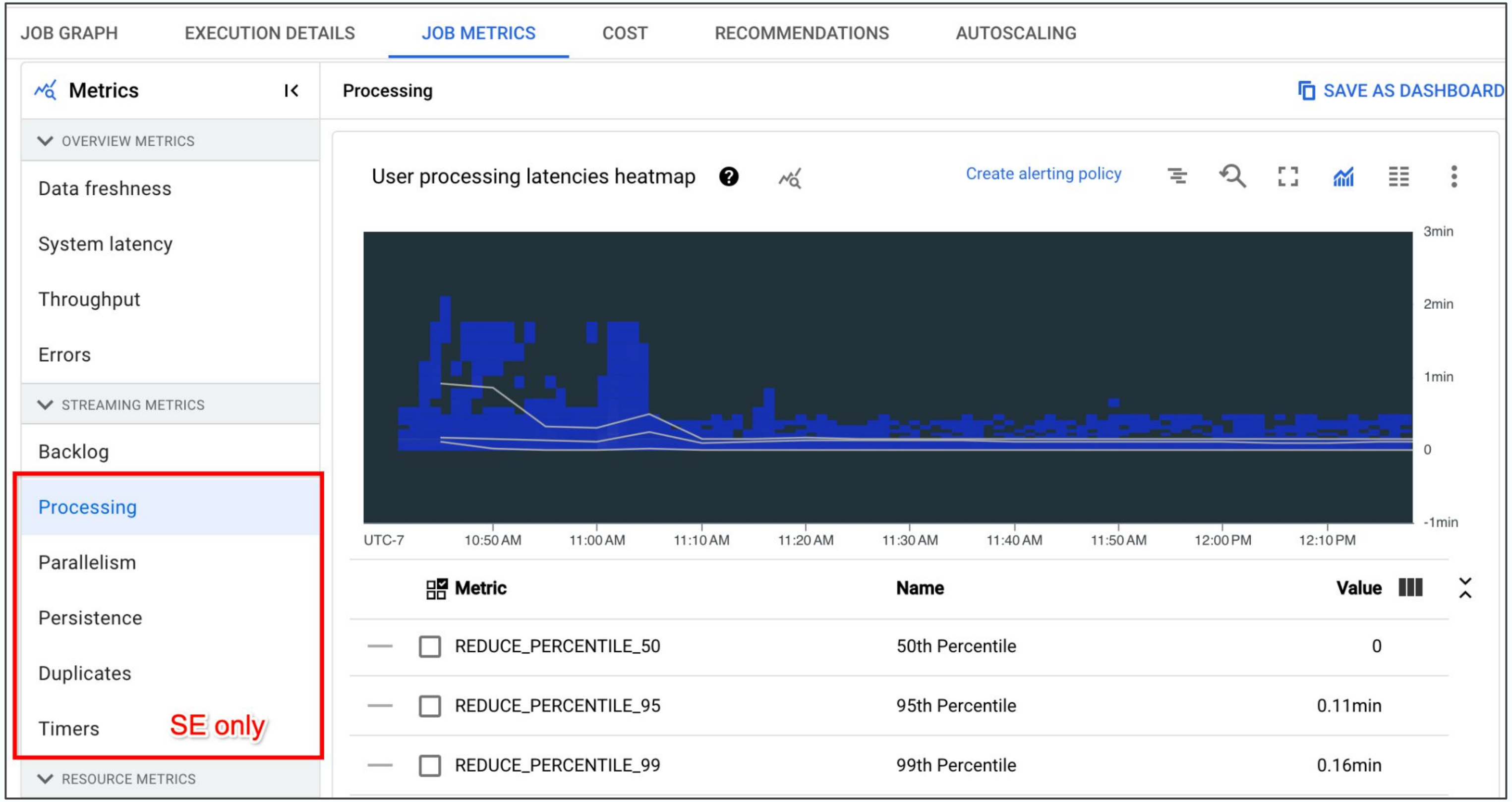
New Metrics



| Metrics | Path |
|---|---|
| Duplicates Filtered | /job/duplicates_filtered_out_count |
| Processing Parallelism | /job/processing_parallelism_keys |
| Backlog Bytes | /job/backlog_bytes |
| Backlog Seconds | /job/estimated_backlog_processing_time |
| Timers Processed | /job/timers_processed_count |
| Timers Resident | /job/timers_pending_count |
| Status of Streaming Pull connections | /job/pubsub/streaming_pull_connection_status |
| The number of bytes produced by this ptransform | /job/estimated_bytes_produced_count |
| Checkpoint bytes written | /job/streaming_engine/persistent_state/write_bytes_count |
| Checkpoint bytes read | /job/streaming_engine/persistent_state/read_bytes_count |
| Checkpoint Latency | /job/streaming_engine/persistent_state/write_latencies |
| User Processing Latency | /job/bundle_user_processing_latencies |
| Key (Range) Availability | /job/streaming_engine/key_processing_availability |
| The number of bytes consumed by this ptransform | /job/estimated_bytes_consumed_count |
| The number of bytes being processed by ptransform | /job/estimated_bytes_active |
| Pubsub Pull to Ack Latency | /job/pubsub/pulled_message_ages |
| Persistent State Usage | /job/streaming_engine/persistent_state/stored_bytes |
| Late pubsub messages | /job/pubsub/late_messages_count |
| Target workers | /job/target_worker_instances |
| Pubsub Publish Messags/Errors | /job/pubsub/published_messages_count |

Launched

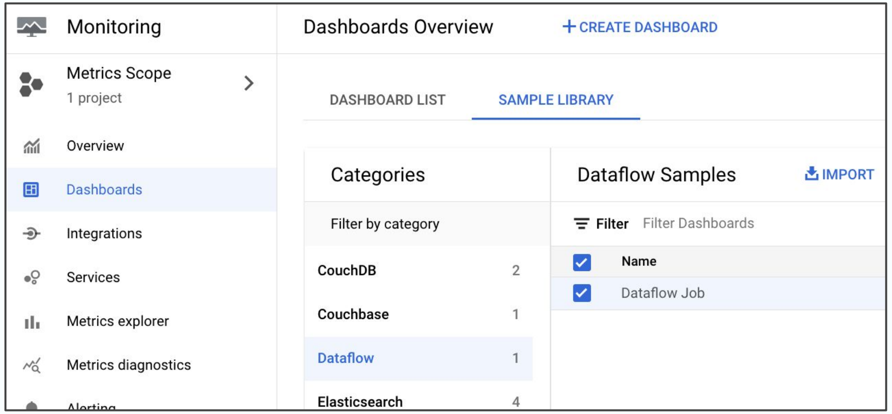
# Observability: Dataflow UI



| Metric | Name | Value |
|---|---|---|
| REDUCE_PERCENTILE_50 | 50th Percentile | 0 |
| REDUCE_PERCENTILE_95 | 95th Percentile | 0.11min |
| REDUCE_PERCENTILE_99 | 99th Percentile | 0.16min |

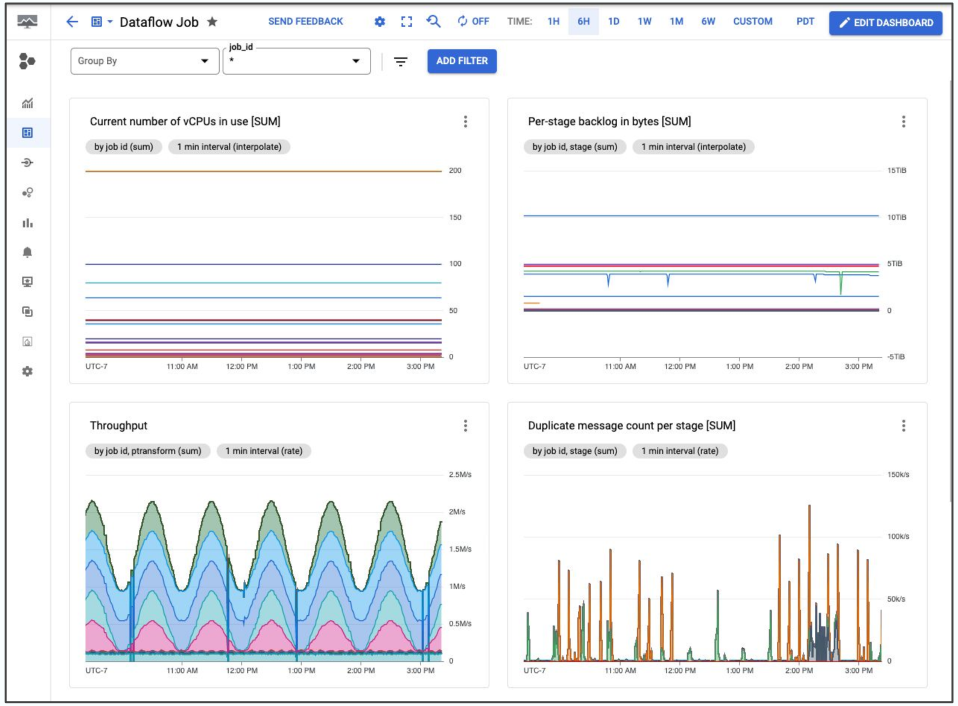
Launched

# Observability: Dashboard Template



| Categories | |
|---|---|
| CouchDB | 2 |
| Couchbase | 1 |
| Dataflow | 1 |
| Elasticsearch | 4 |

Importing template


Dataflow Job
SEND FEEDBACK
OFF
TIME: 1H 6H 1D 1W 1M 6W CUSTOM PDT
EDIT DASHBOARD
Group By
job_id
ADD FILTER
Current number of vCPUs in use [SUM]
by job id (sum)
1 min interval (interpolate)
Per-stage backlog in bytes [SUM]
by job id, stage (sum)
1 min interval (interpolate)
Throughput
by job id, ptransform (sum)
1 min interval (rate)
Duplicate message count per stage [SUM]
by job id, stage (sum)
1 min interval (rate)
UTC-7
11:00 AM
12:00 PM
1:00 PM
2:00 PM
3:00 PM

Preview of first few graphs

## Other Projects

Launched

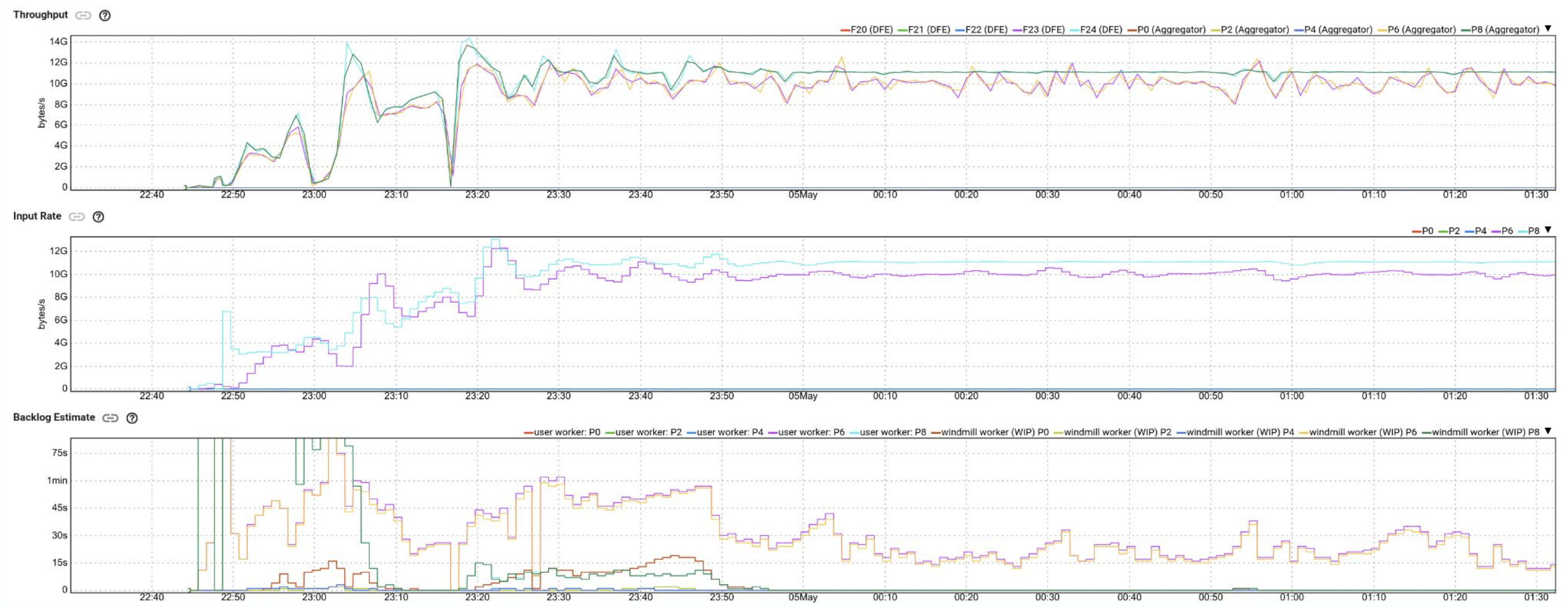
## Out of the box

We wanted to test the throughput of sources and sinks without any special settings. We got to 10 GB/s for these I/O combos:

- Pubsub to BQ
- Pubsub to Pubsub
- Pubsub to GCS*
- Kafka to GCS*
- Kafka to BQ

# Out of the box

Pubsub to GCS example



Launched

## Dataflow Cookbook

Collection of +190 self-contained Dataflow pipelines ready to use, including most common sources, sinks, and use cases.

https://github.com/GoogleCloudPlatform/dataflow-cookbook

Iñigo San Jose, Tom Stepp

## QUESTIONS?

BEAM
SUMMIT