Google Cloud

BEAM SUMMIT

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

Pramod Rao

Cloud Data Engineer

Google Cloud Consulting

Prateek Sheel

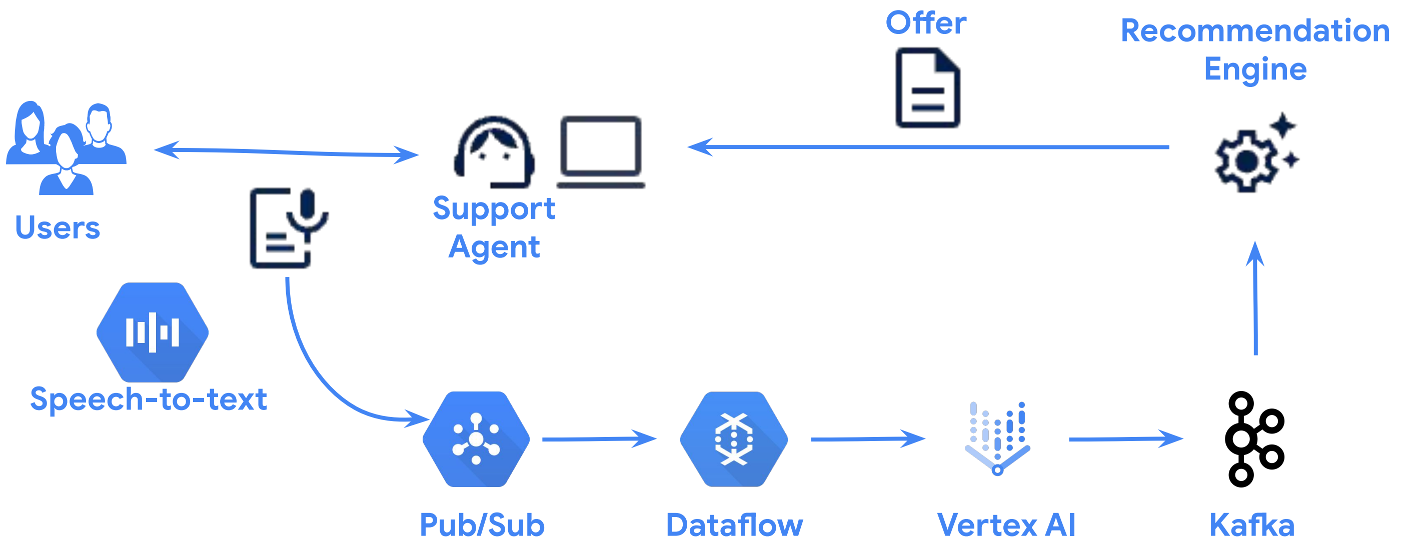
Data & Analytics Consultant

Google Cloud Consulting

01

Overview

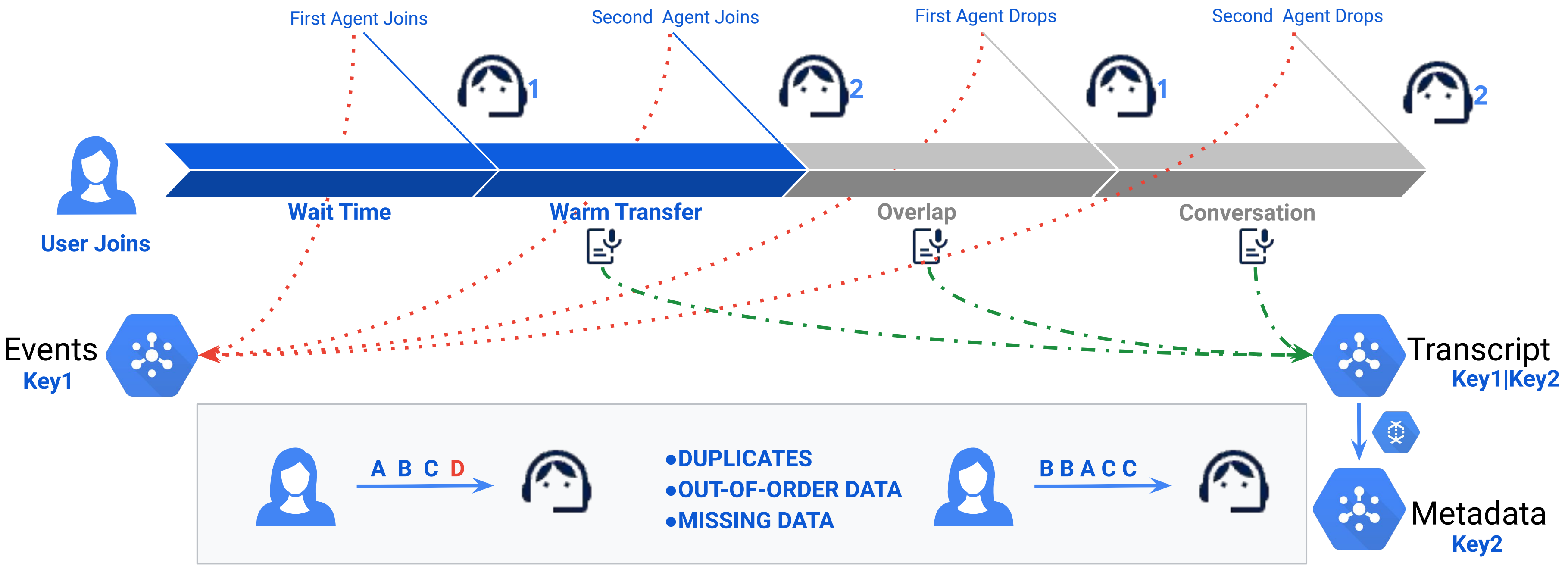
Business Process



So, what’s the problem?

Plus, additional business rules

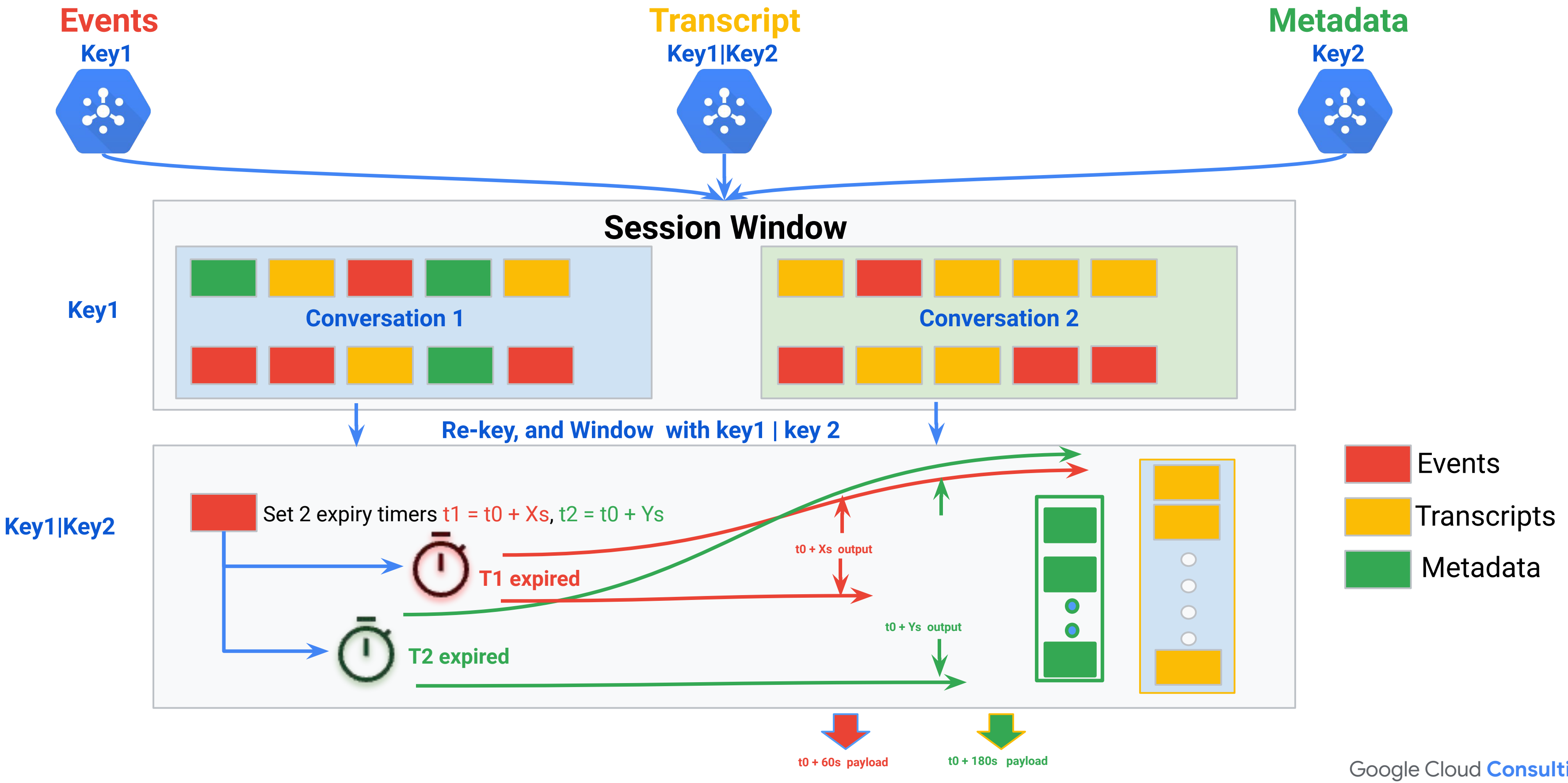
Multiple Call-Transfer Scenarios



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.

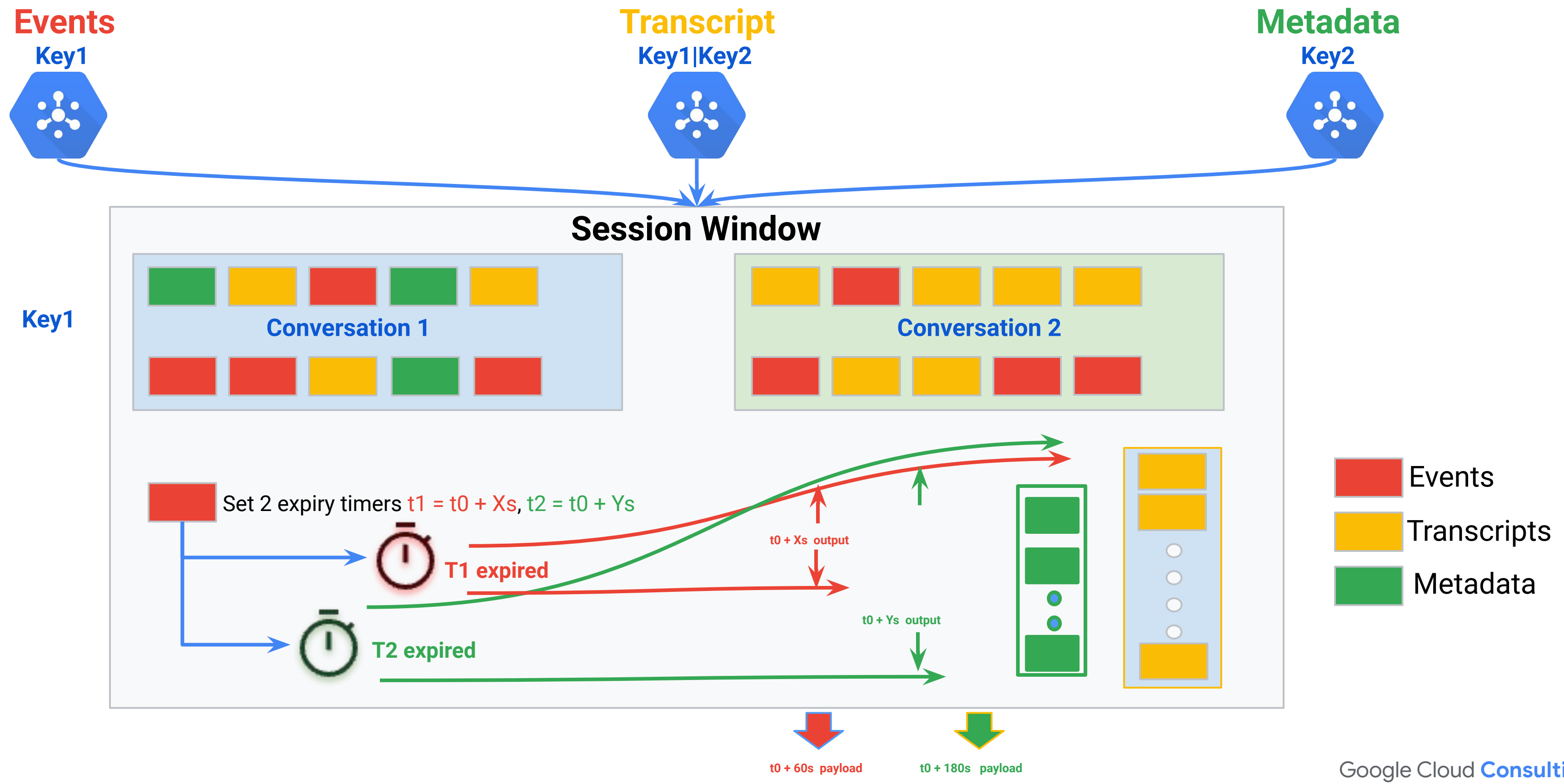
Completeness

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.

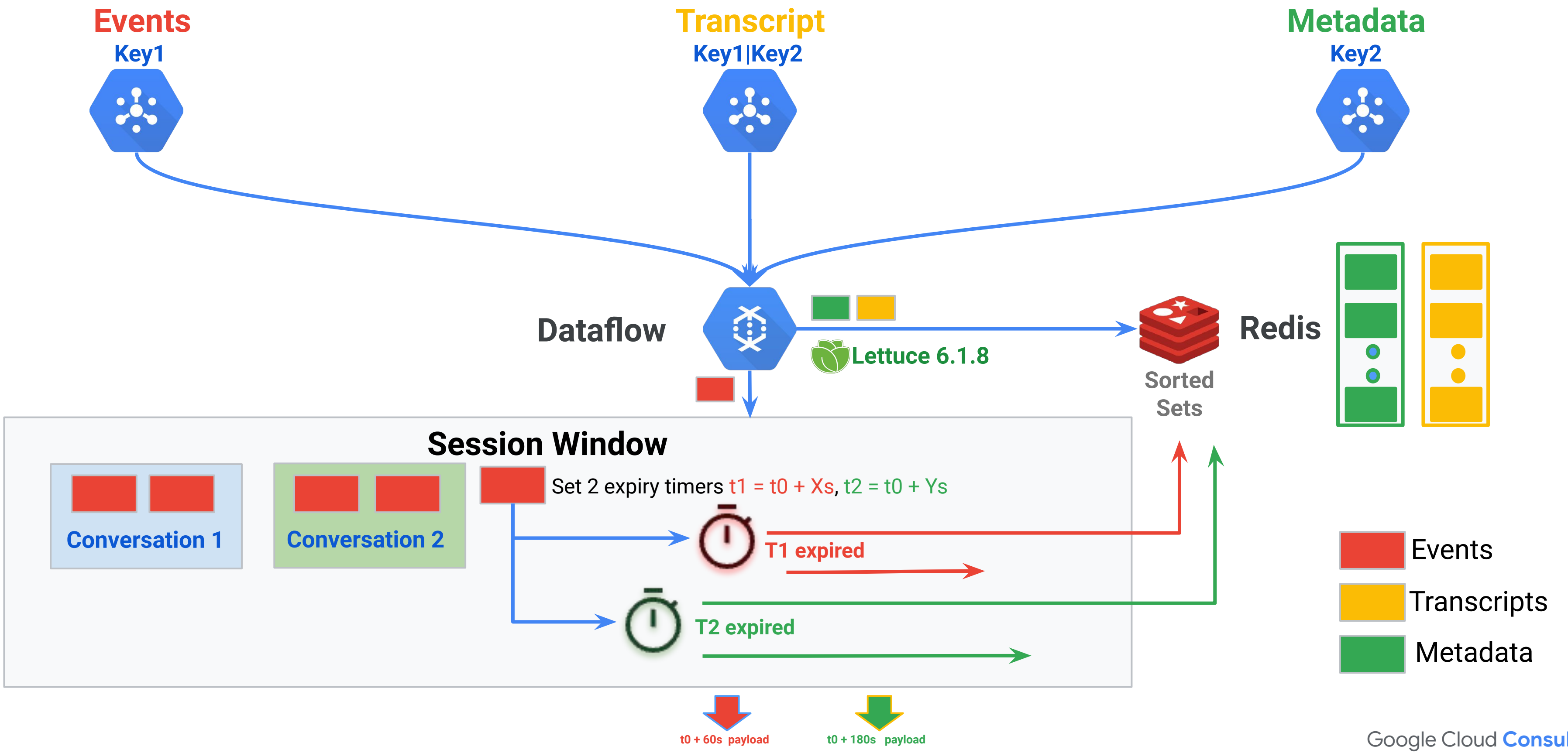
Completeness

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

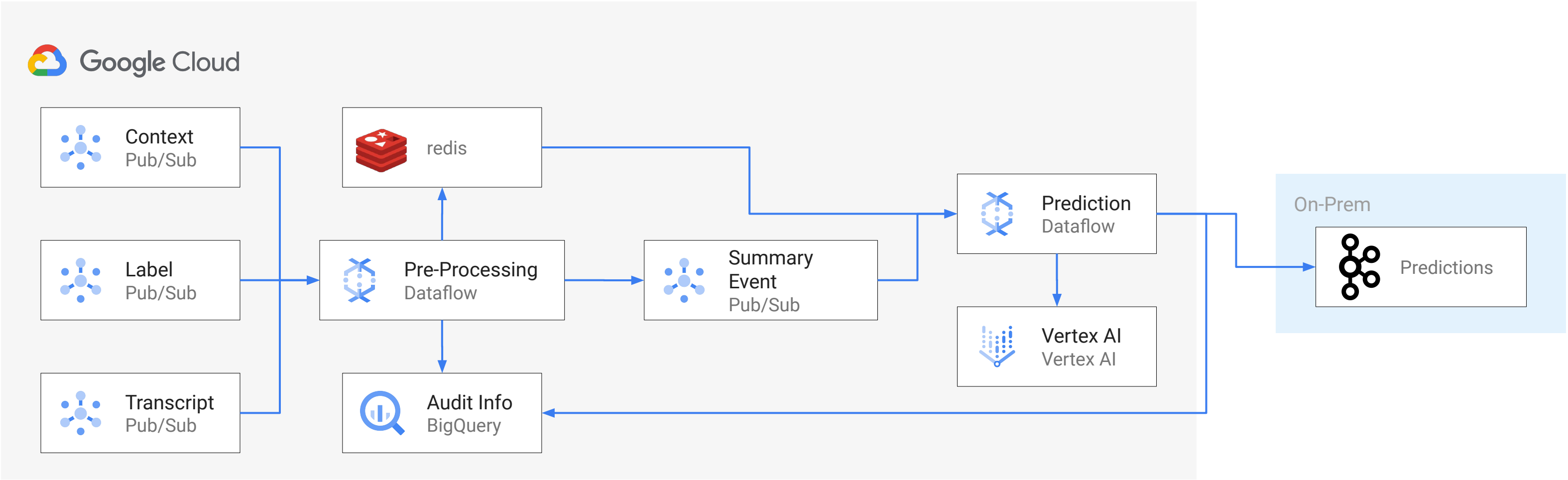
Much **simpler processing** because complicated scenarios related to cross-referencing the three data sources are eliminated. Only need to “act” on events.

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

Final Solution

Speech-to-text Processing with Apache Beam and Redis



03

Lessons Learned

Lessons Learned

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"

Representative test data

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

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

Latency

Latency requirements dictate the nature of the final solution

Thank you!

https://cloud.google.com/consulting