Serving Near Real-Time Features at Scale

Feng Xu
Software Engineer | Uber Marketplace
May, 2022

Uber

Agenda

- Common Challenges
- Streaming Processing
- Streaming Platform
- Framework of Tuning
  - How to tune a pipeline with examples
- Key Takeaways
- Q&A

At scale - data as a product

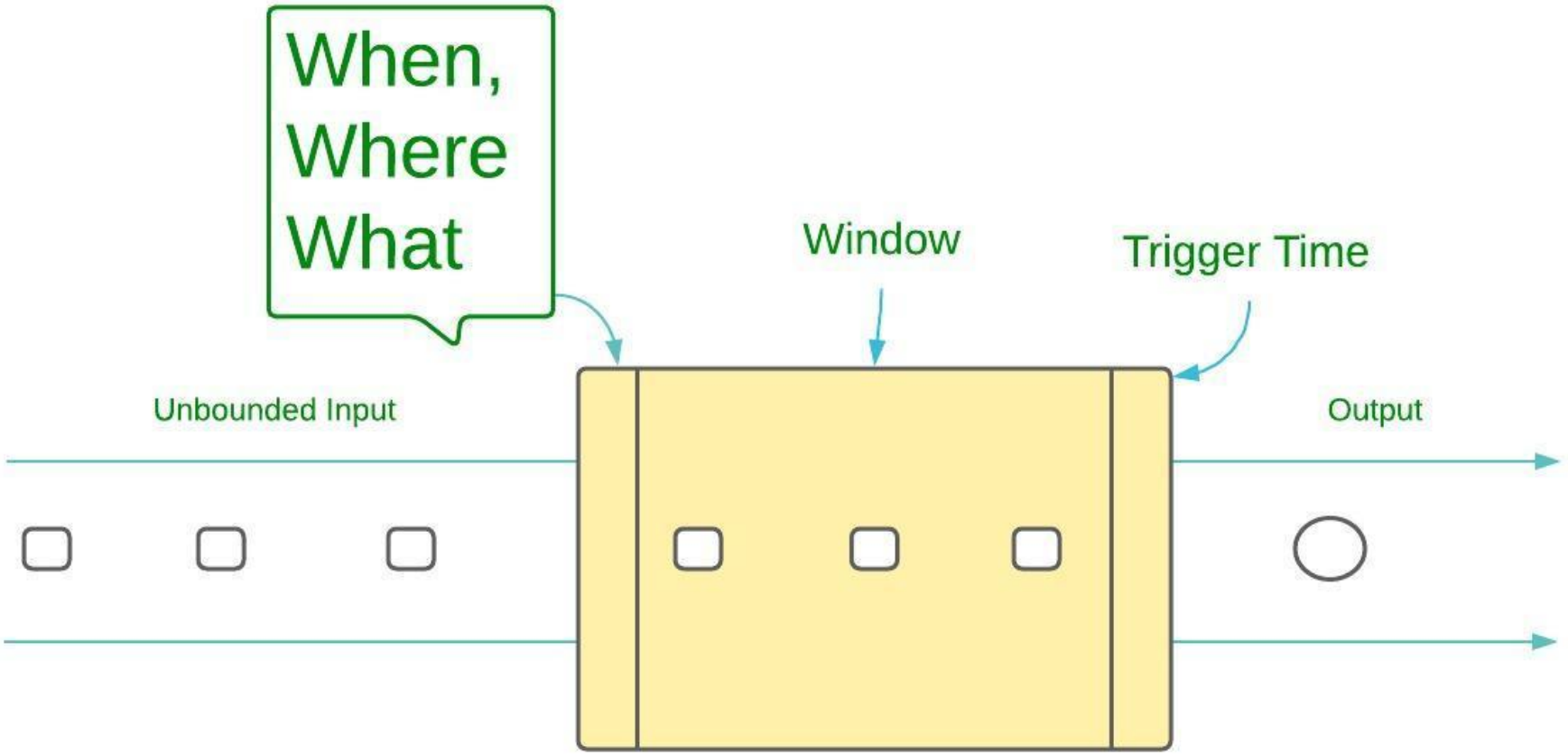
- Data onboarding and extension
- Data quality
- Data discovery
- ACL

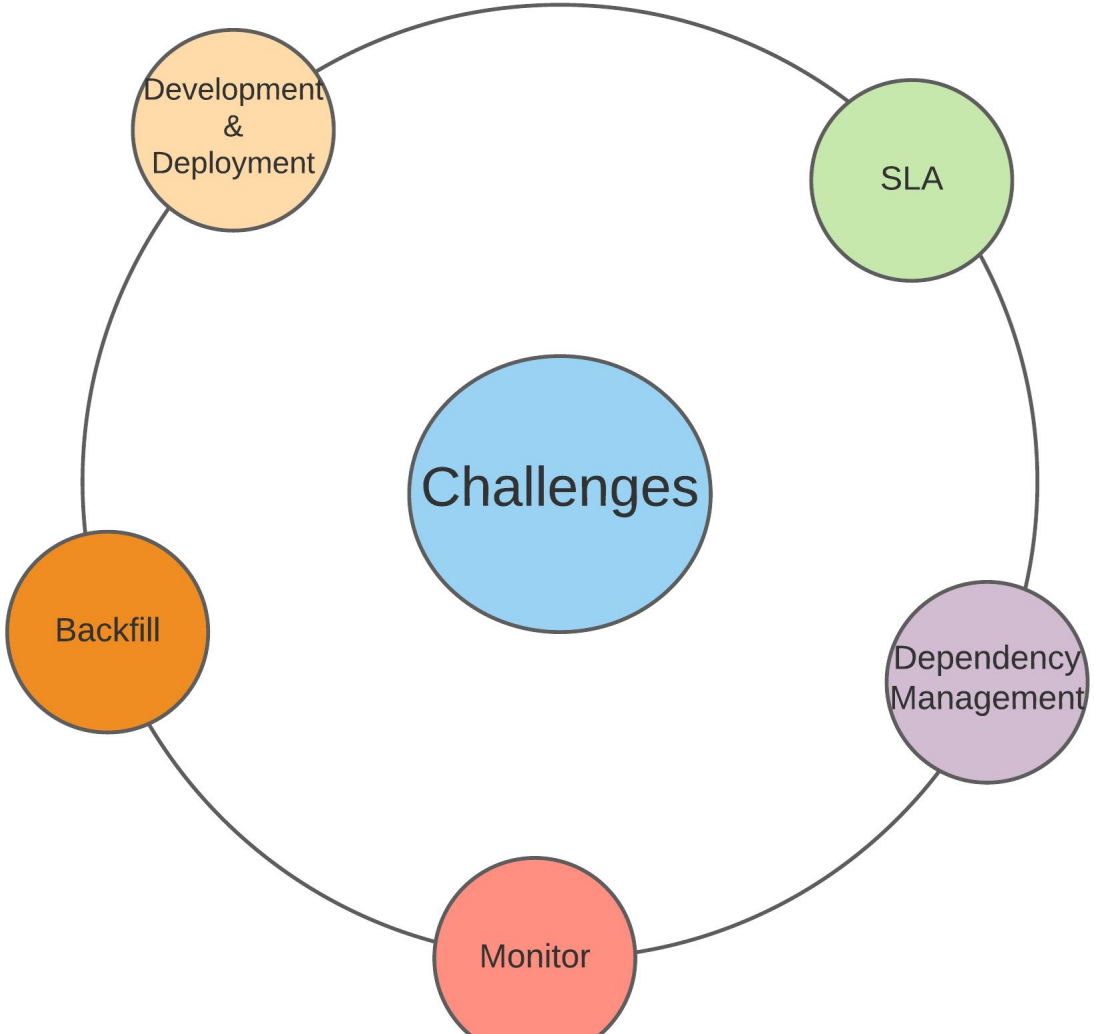
At scale - define the serving contract

Use Case	**Data Freshness**	**Latency of Access**	**Data Range**	**…**
Backend Service 1	Most recent data	At Second level	Past 30 minutes	
Backend Service 2	Most recent data	At Millisecond level	Past 1 hour	
Trend Dashboard	Up to hours	At Second Level	Could be more than 1 year	
Ad-hoc Query	Up to hours	At Minute Level	Could be more than 1 year	
…				

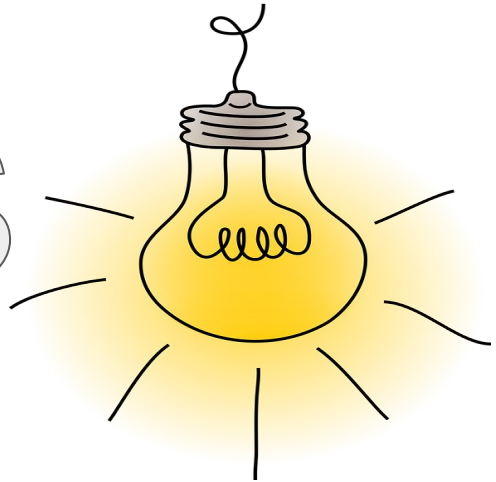
Since Uber operates in the transportation domain, we need to have the visibility into the state of market as close to REAL-TIME as possible, and make informed decisions in a timely manner.

Next we focus on serving near real-time features with the streaming processing.


When,
Where
What
Window
Trigger Time
Unbounded Input
Output


Development
&
Deployment
SLA
Challenges
Backfill
Dependency
Management
Monitor

Solutions



Gairos (Java Based)

uMetric Streaming
(Configuration Driven)

Which solution, Java-based or uMetric Streaming?

	Gairos	uMetric Streaming
Typical Use Cases	• Enriched *raw-data* or non-metric related • *Customized window* operation	• Feature computation on a time window Formula is *SQL based* and common UDFs
Onboarding Effort	• Customized Java code • One day to couple months	• Feature definition with YAML • Fulfill with workflow One day for 80% use case

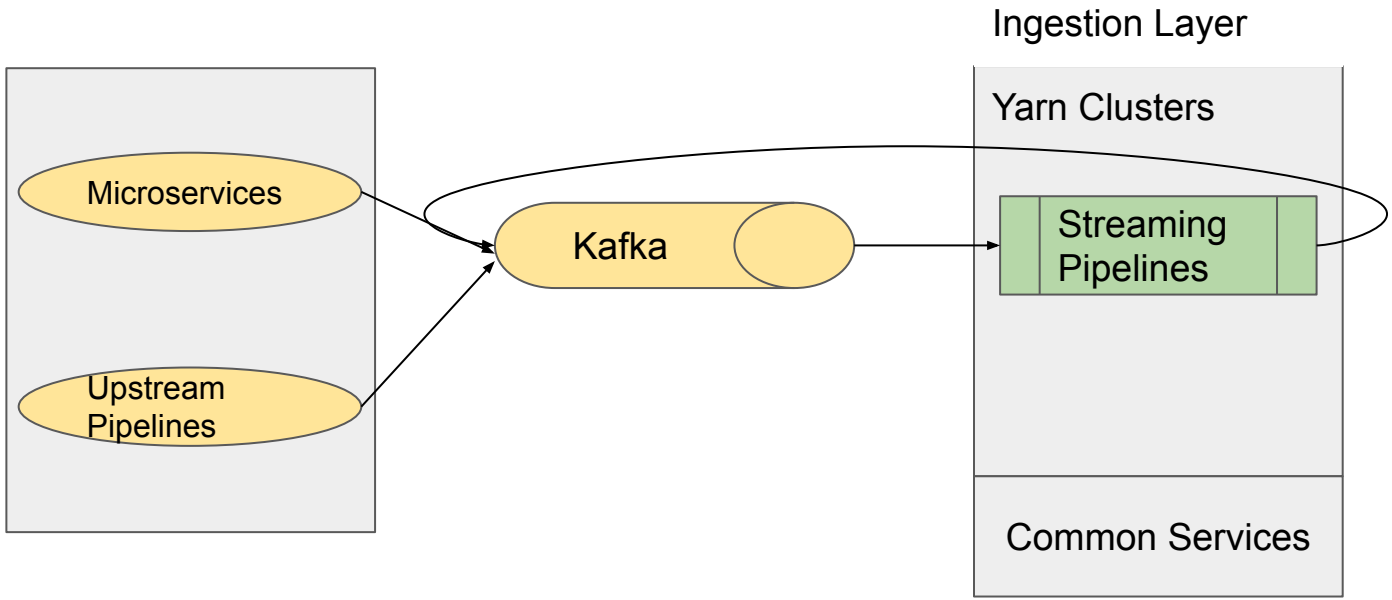
What does the real-time data analytics platform look like?

Three major layers:

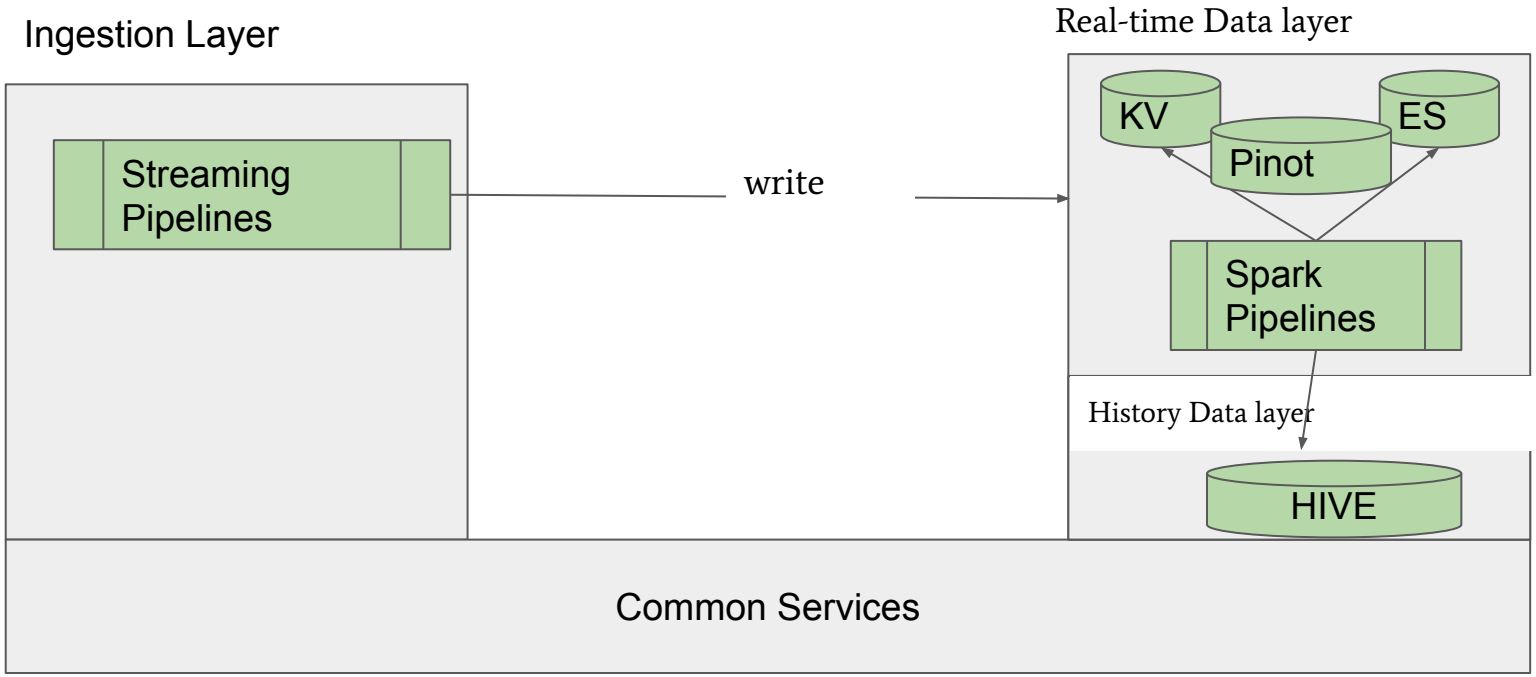
- Ingestion Layer
- Storage Layer
- Query Layer

Let’s go through each layer...

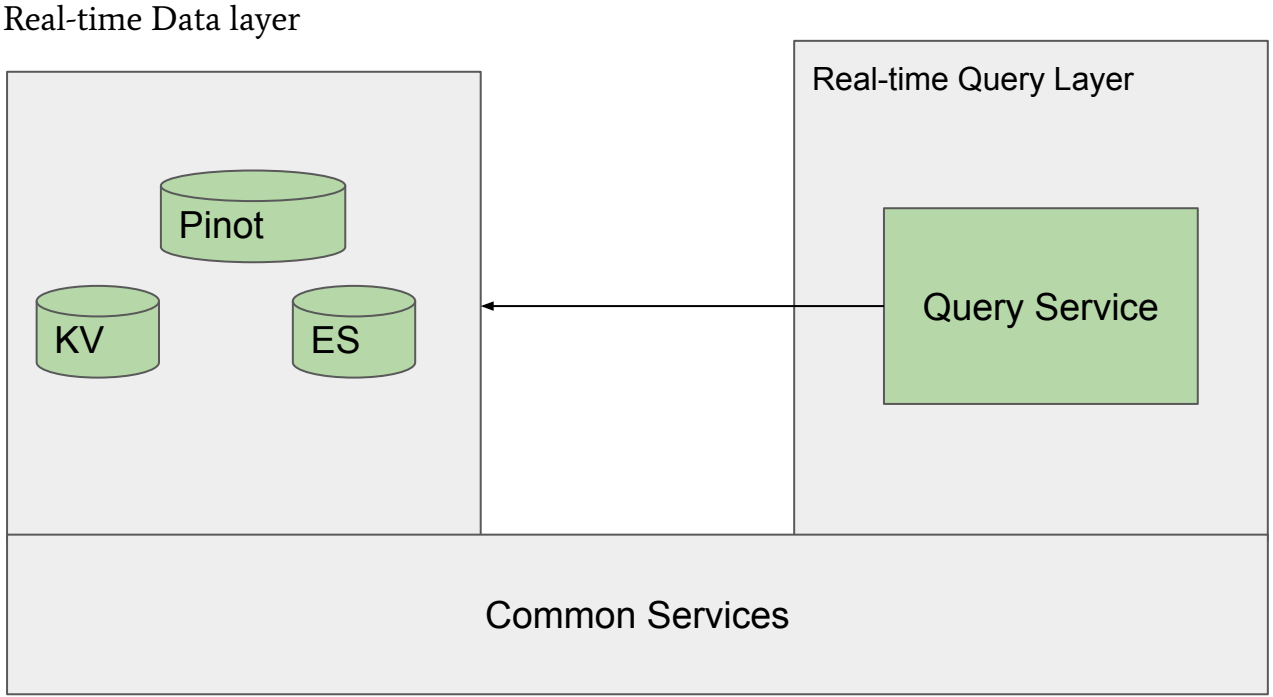
Where does the streaming data come from?



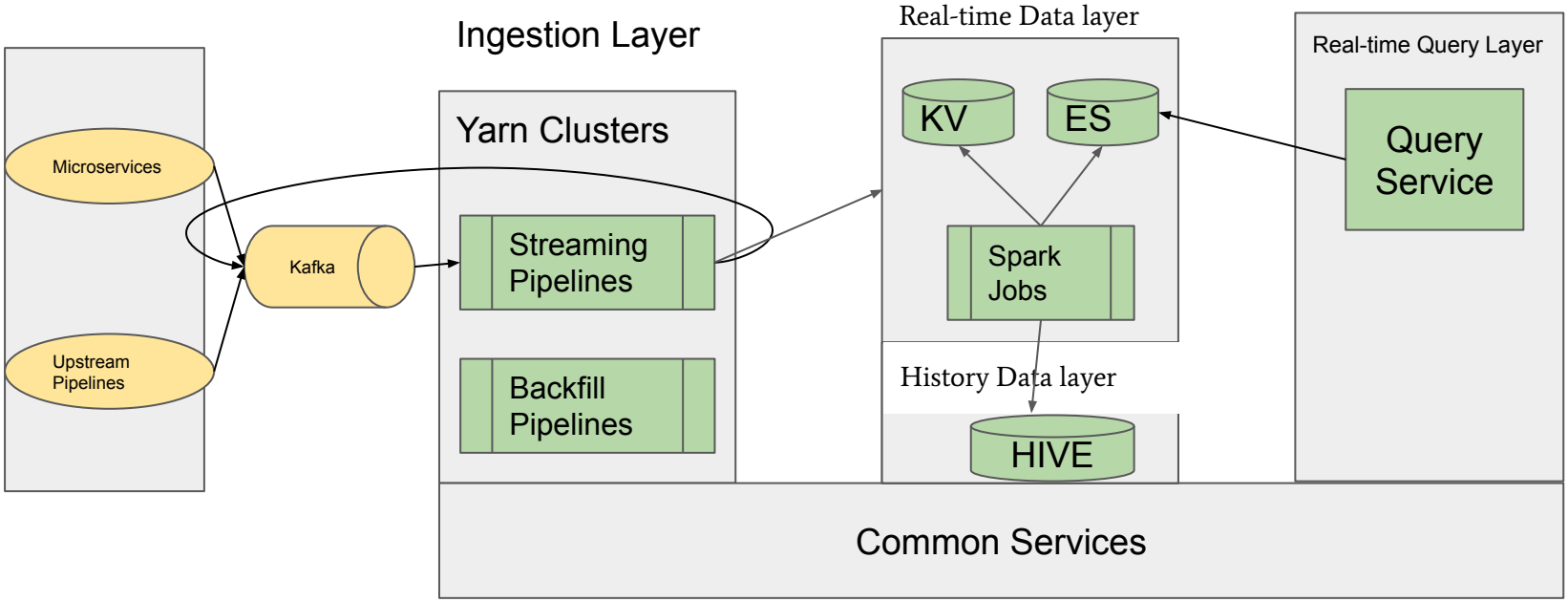
Where is the data stored?



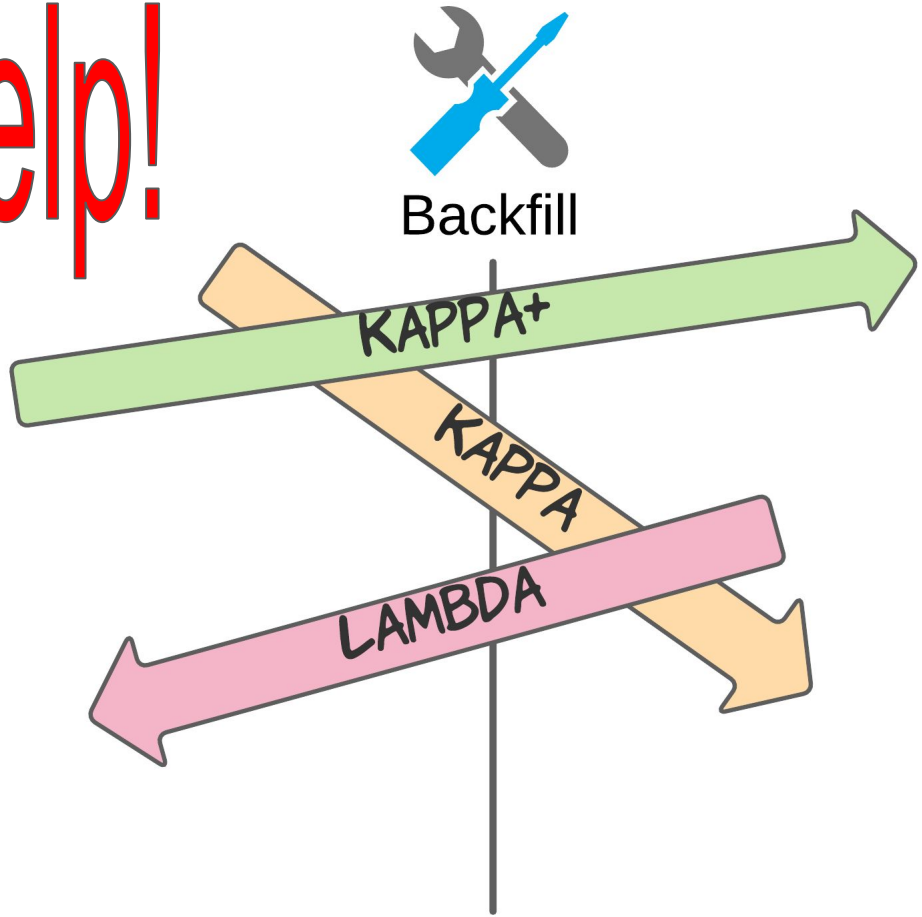
How to access the data?



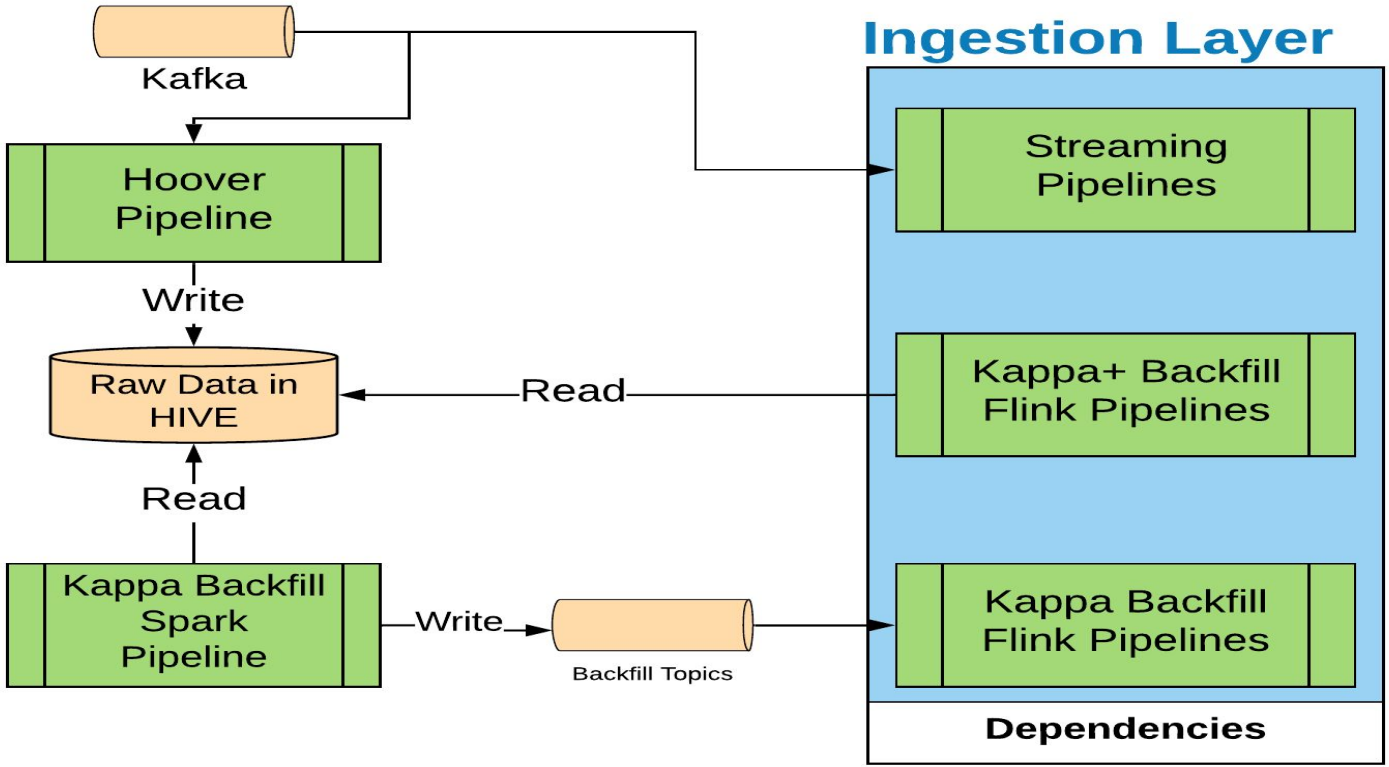
To put together



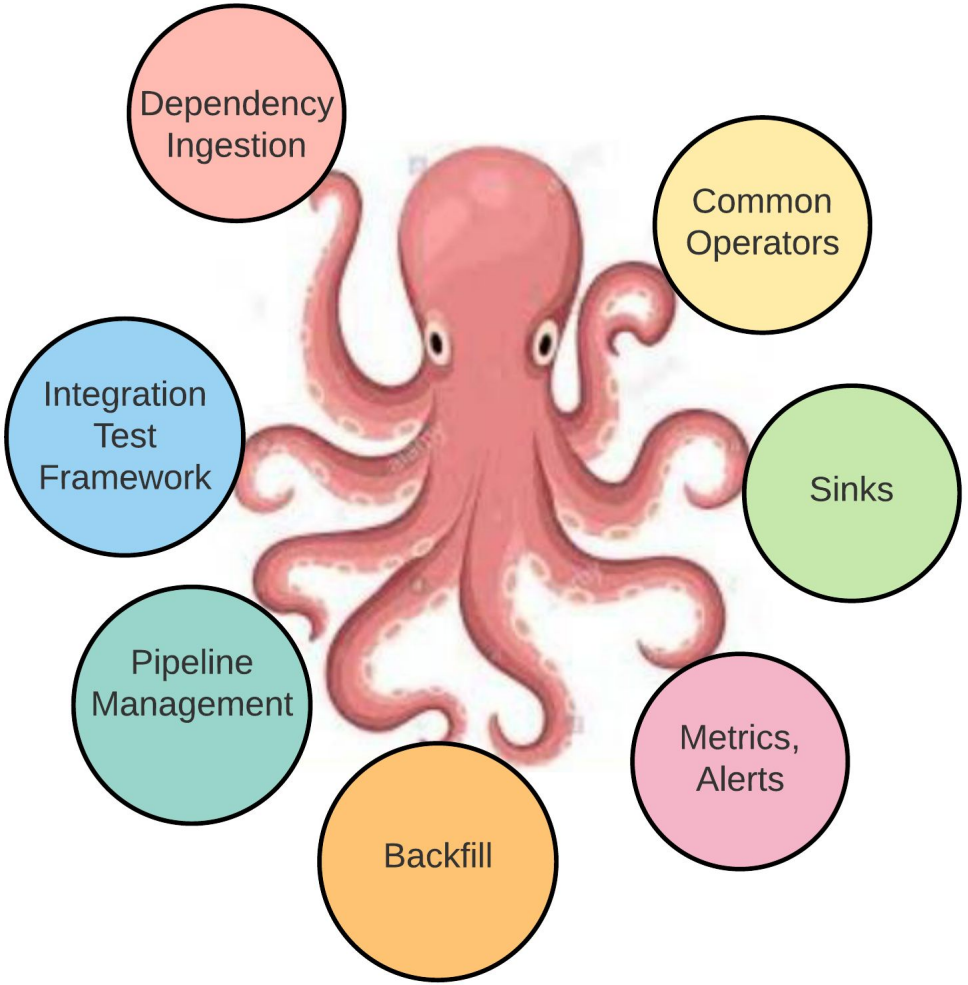
Data Loss, Help!



How to backfill?



How could the platform help developers if to have pipelines in Java?



Some Numbers

100+

Streaming Pipelines in Flink

2M

Messages per second

1T

Data ingestion to Hive

80+

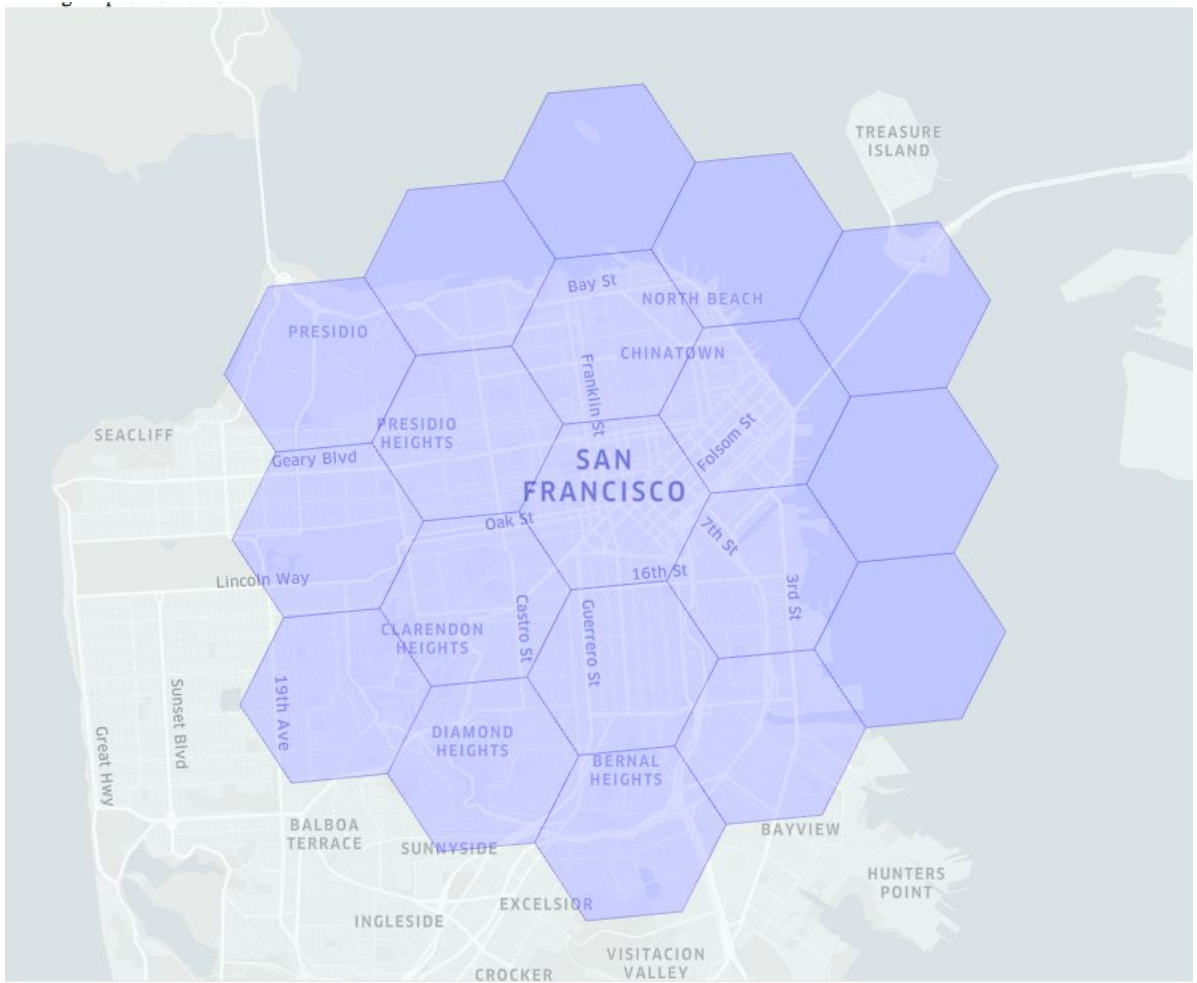
Real time tables

Time to explore some pipelines with scale concern

Demand & Supply Pipelines:

- Eyeball minute & supply minute
- Time granularities: 1, 2, 4, 8, 16, 32 min, sliding by 1-minute
- Spatial Index: kring 0, 1, 2, 3, 4, 5, 10, 15, 20

Uber's H3 Spatial Index



Scales of Eyeballs

70K

Count(Hexagon with Eyeball in a city)

120K

Ingestion rate of Eyeball per second

6.5K

Eyeball count of busiest Hexagon per minute

1200+

Number of neighbours

54

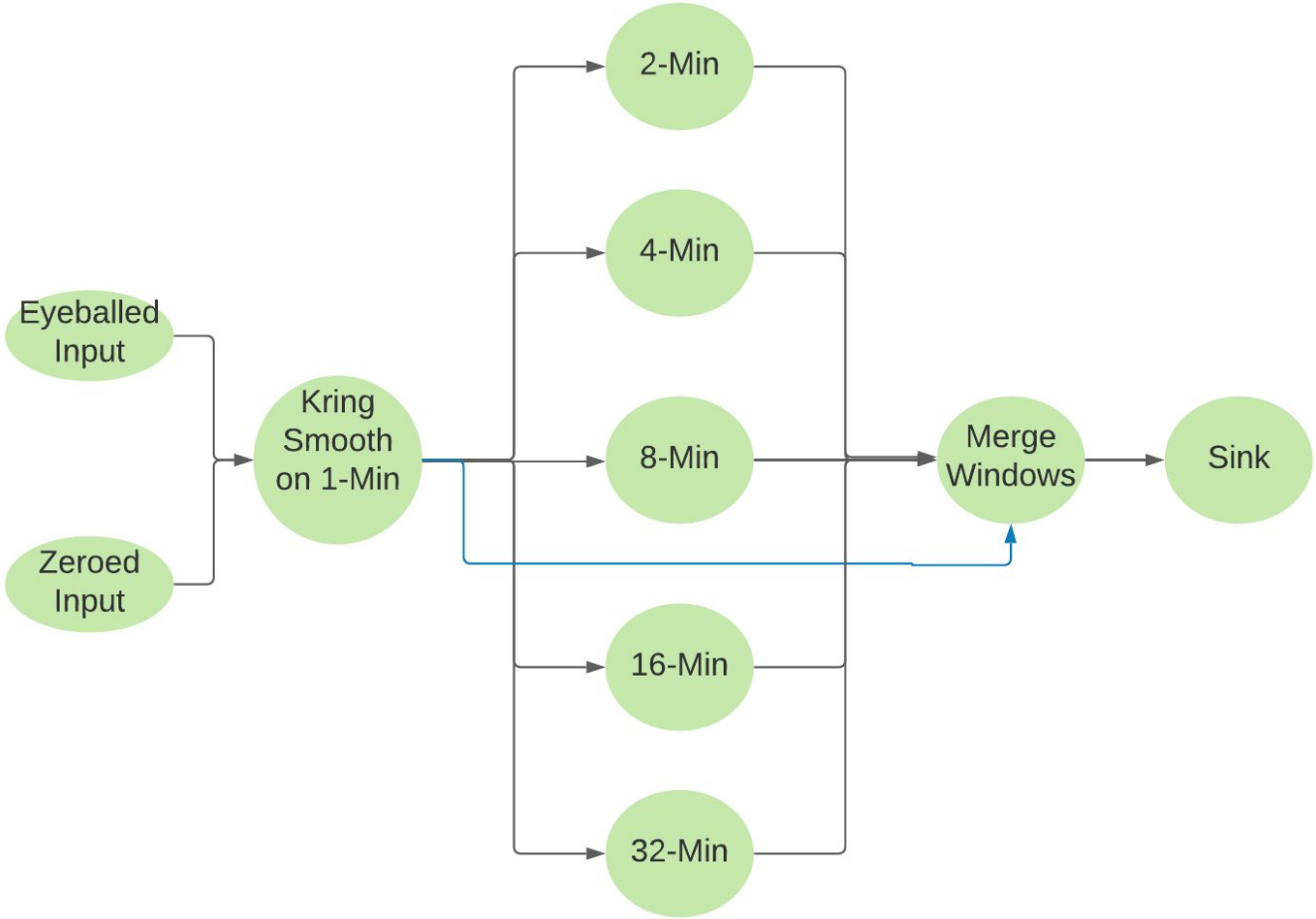
Features per hexagon per minute

63

The count of windows where an event is computed

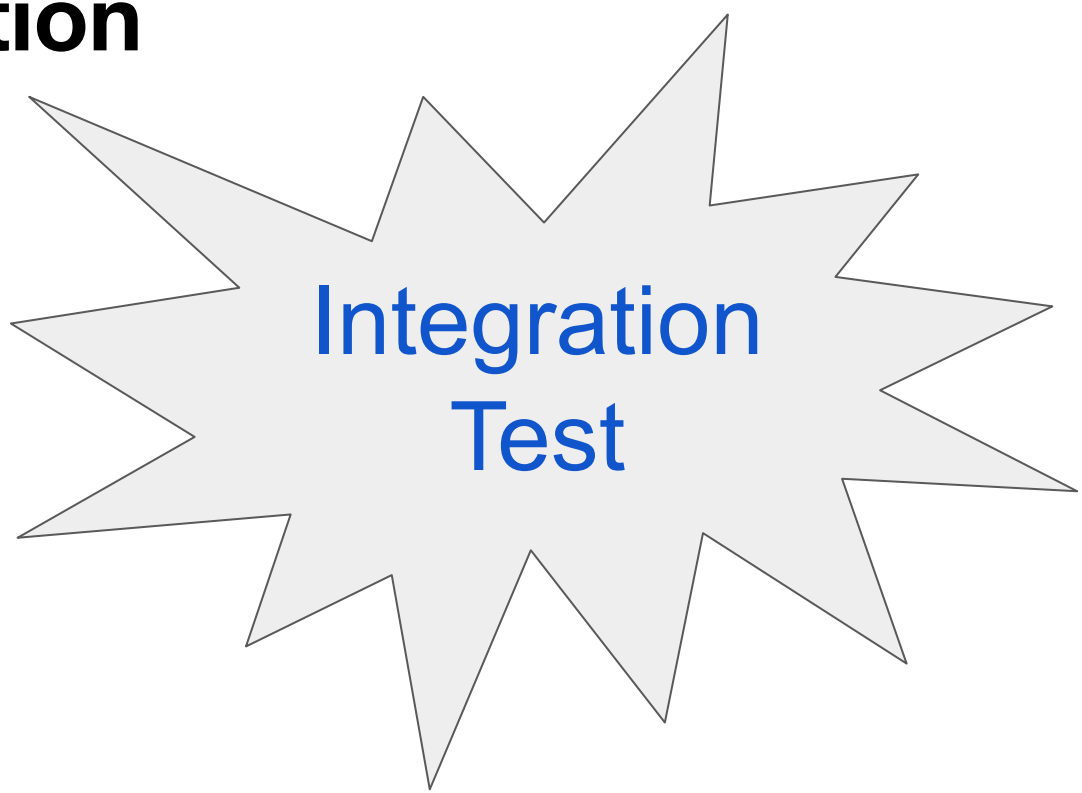
Logic Job DAG

[Optimized DAG]

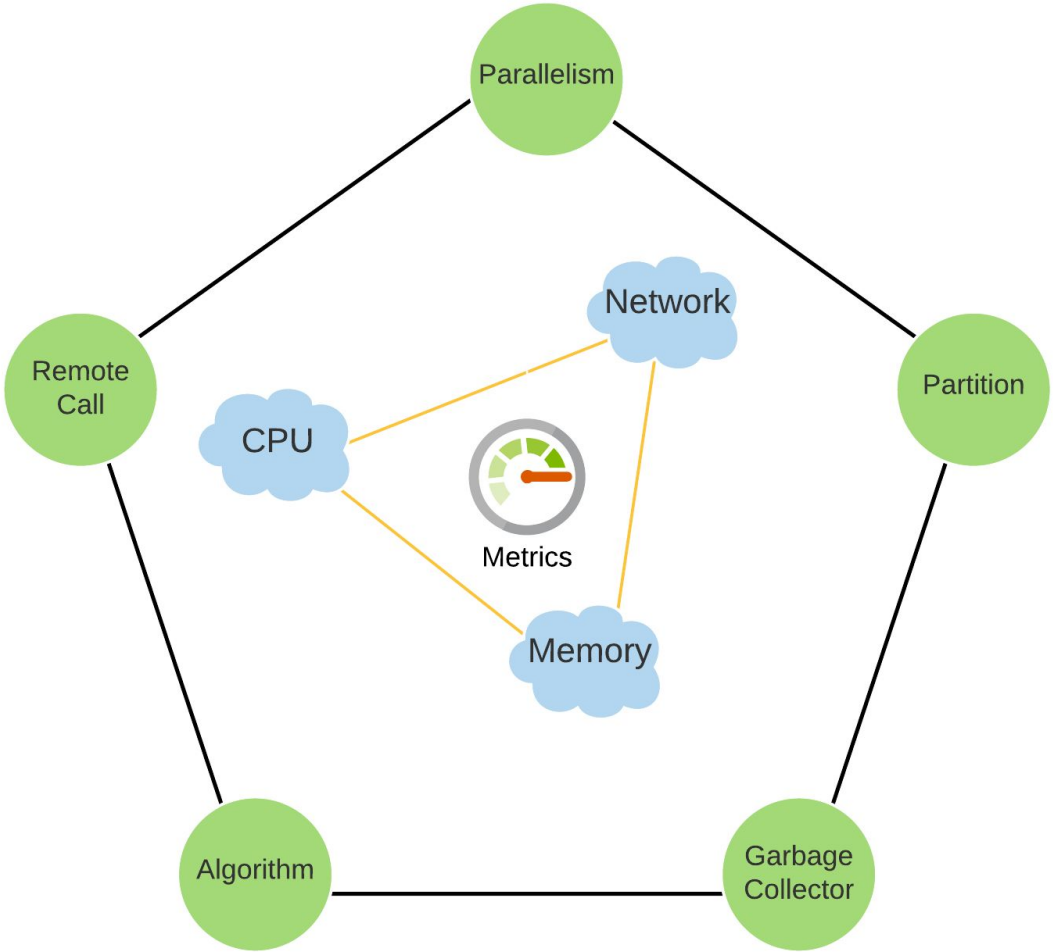


Oooh, Not Good

Before Optimization



How to optimize streaming pipeline?



Tuning on network

- Only pass necessary fields from upstream to downstream
- Encode the parent product type uuid as a byte via an internal coding
- Apply filters and dedup operators as early as possible
- **Replace the sliding window operators** (for 2, 4, 8, 16, 32 min) with a customized FlatMap operator

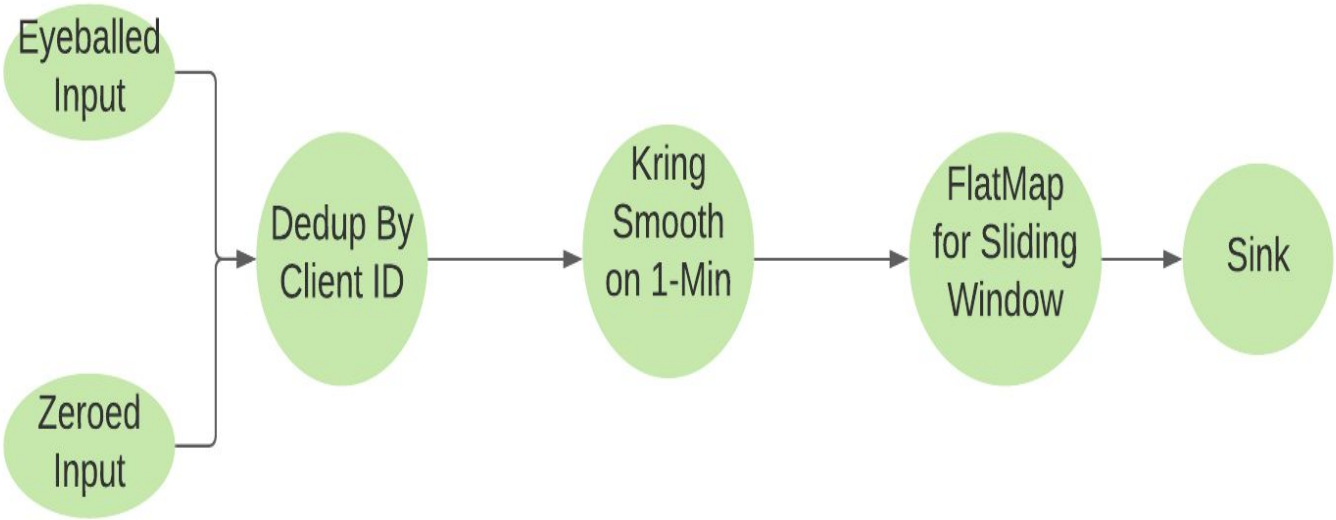
Tuning on memory

- Use Tuple rather than POJO
- Integer is used at intermediate stage, reduced the message size from 450+ bytes to less than 240 bytes
- Enable Object Reuse

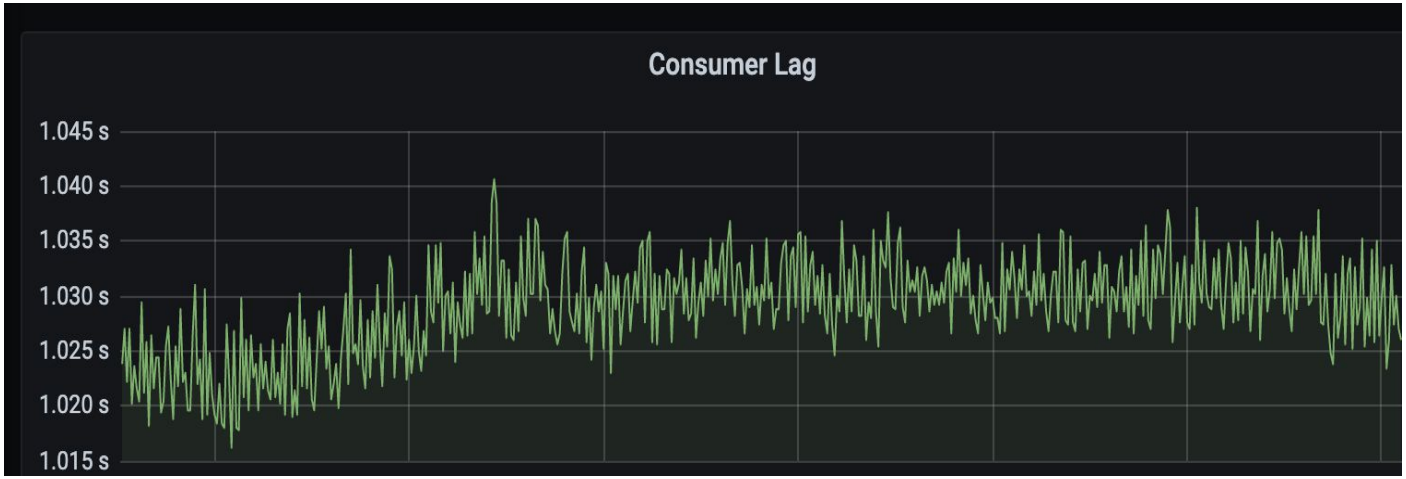
Tuning on CPU

- Applied dedup before further aggregation
- Customized operator to replace sliding window, avoided the overhead for the window management and related de/ser costs
- Avoid the cost of boxing/unboxing

Optimized Job DAG



Before Optimization


Consumer Lag
1.045 s
1.040 s
1.035 s
1.030 s
1.025 s
1.020 s
1.015 s


TaskManager Memory Usage
7 GiB
7 GiB
6 GiB
5 GiB
4 GiB
3 GiB
2 GiB
954 MiB
0 B
2020-12-31 11:15:00
gairos-athena-flink-agg_demand_supply-processor-demand-agg jvm_memory_total_used: 3.75 GiB
gairos-athena-flink-agg_demand_supply-processor-demand-agg jvm_heap_used: 3.60 GiB
gairos-athena-flink-agg_demand_supply-processor-demand-agg jvm_non_heap_used: 150 MiB
gairos-athena-flink-agg_demand_supply-processor-demand-agg jvm_heap_max: 6.75 GiB
gairos-athena-flink-agg_demand_supply-processor-demand-agg jvm_memory_total_max: 6.75 GiB
gairos-athena-flink-agg_demand_supply-processor-demand-agg jvm_non_heap_max: -1 B

Key Takeaways

- To be scalable: build a platform which enable engineers to focus on the business logic
- To have the scalable pipeline, you need to have a Framework of performance tuning

Further Readings

- H3: Uber’s Hexagonal Hierarchical Spatial Index
- The journey towards metric standardization
- Uber's Real-time Data Intelligence Platform At Scale: Improving Gairos Scalability/Reliability
- Building Scalable Streaming Pipelines for Near Real-Time Features

Thank you!
Q&A