Real-Time Streaming With Kafka

Tom Stepp

Table of Contents

Streaming Kafka Intro

Streaming Systems

Streaming systems are built for many reasons:

- Payment and financial transactions
- Logistics and automotive fleet tracking
- Sensor data from IoT systems
- Collect and react to customers in retail, travel, mobile apps
- Log extraction and fraud detection
- Gaming analytics

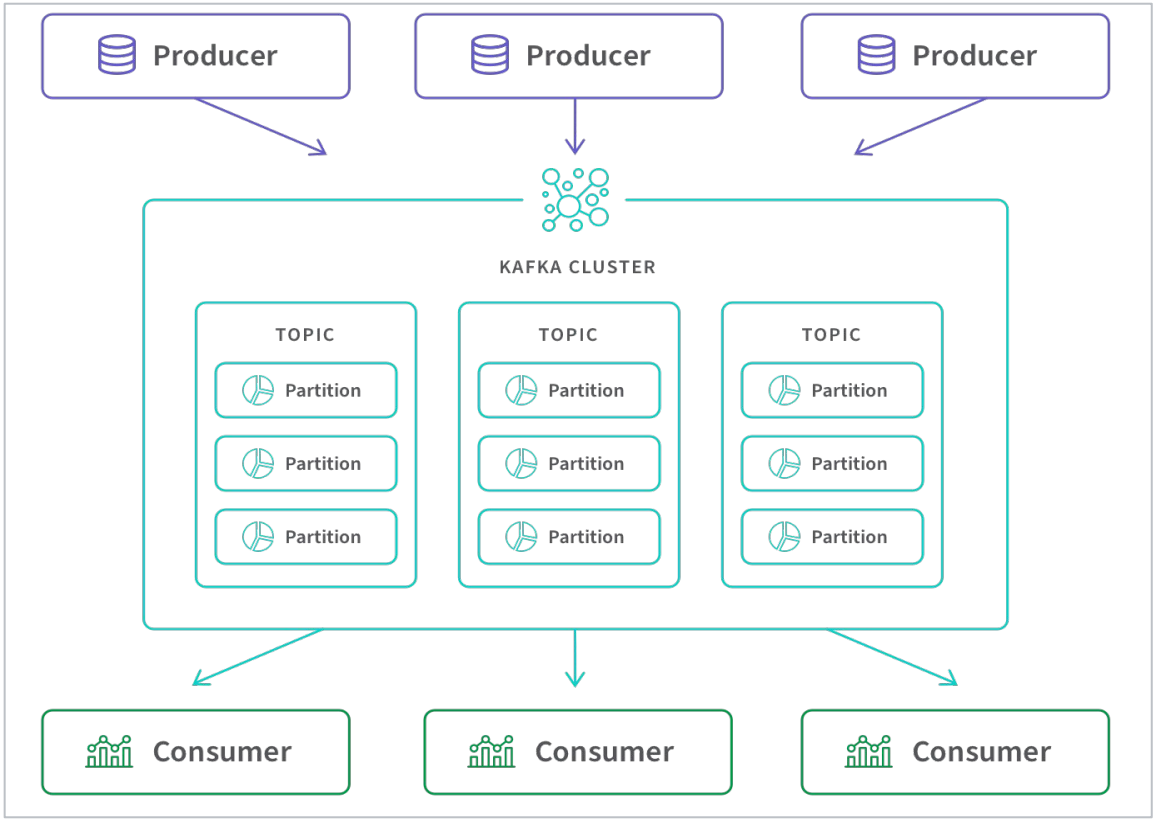
Kafka’s Role

- Reliably collects, stores, and distributes data streams.
- Scalable, fault-tolerant, and durable.
- Intermediary which connects producers to consumers.
- Most popular open-source stream-processing software.

Key Concepts

- Event – Piece of data with key, value, timestamp, metadata.
- Topic – Durably stores events, may have many producers and subscribers.
- Partition – Slice of a topic, allows for scalability, provides event ordering.
- Producer – Clients which publish events to Kafka.
- Consumer – Clients which subscribe, read, and process events from Kafka.
- Replication – Storing extra copies of data for fault tolerance.

Kafka Architecture



Kafka Client APIs

- Admin API – manage and inspect topics, brokers, and more.
- Producer API – publish a stream of events.
- Consumer API – subscribe, read and process events.

Apache Beam Kafka IO

Read From Kafka

```
1.  KafkaIO.<Key, Value>read()
2.      // Required settings
3.      .withBootstrapServers("broker_1:9092,broker_2:9092")
4.      .withTopic("my_topic")
5.      .withKeyDeserializer(KeyDeserializer.class)
6.      .withValueDeserializer(ValueDeserializer.class)
```

Optional Config

```
1.  // Consumer Config
2.  .withConsumerConfigUpdates(ImmutableMap.of("group.id", "my_beam_pipeline"))
3.
4.  // Event Timestamps, processing time is one of a few options.
5.  .withProcessingTime()
6.
7.  // Restrict reader to committed messages on Kafka, for exactly-once semantics.
8.  .withReadCommitted()
9.
10. // Commit offsets: preferred over 'auto.commit' in Kafka consumer config.
11. .commitOffsetsInFinalize()
12.
13. // Extract KV out of Kafka record.
14. .withoutMetadata() // PCollection<KV<Key, Value>>
```

Write To Kafka

```
1.  KafkaIO.<Long, String>write()
2.      // Required Config
3.      .withBootstrapServers("my_broker:9092")
4.      .withTopic("my_topic")
5.      .withKeySerializer(LongSerializer.class)
6.      .withValueSerializer(StringSerializer.class)
7.
8.      // Optional Producer Config
9.      .updateProducerProperties(ImmutableMap.of("compression.type", "gzip"))
```

Managed IO

Managed IO

Simplifies pipeline management for supported sources and sinks.

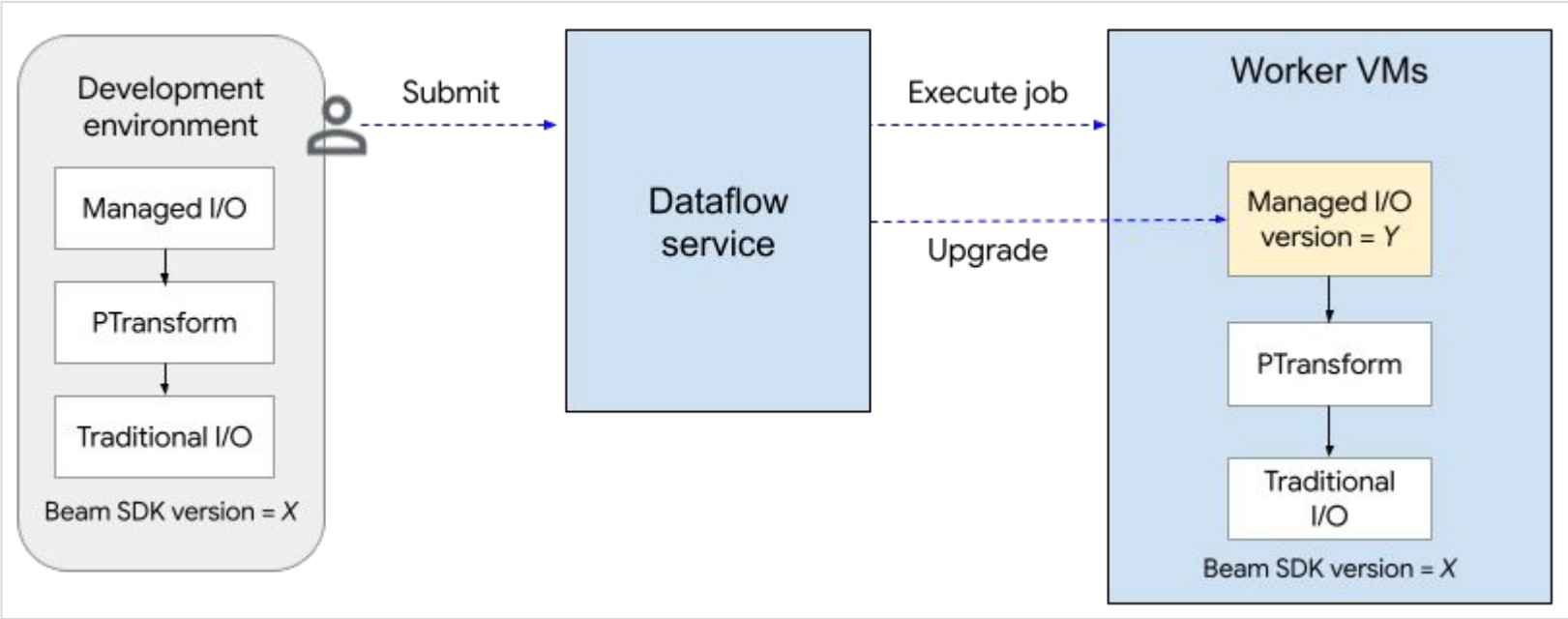
Consists of:

- A Beam transform that provides a common API for creating I/O connectors.
- A service that provides IO upgrades independent of the Beam version.

Advantages of Managed IO

- Dataflow automatically upgrades the managed I/O connectors in your pipeline.
- A single configuration API, resulting in simple and consistent pipeline code.

Managed IO


Development environment
Managed I/O
PTransform
Traditional I/O
Beam SDK version = X
Submit
Dataflow service
Execute job
Upgrade
Worker VMs
Managed I/O version = Y
PTransform
Traditional I/O
Beam SDK version = X

Managed IO for Kafka

It’s simple!

- Managed.read(Managed.KAFKA).withConfig(readConfigMap);

- Managed.write(Managed.KAFKA).withConfig(writeConfigMap);

Managed IO for Kafka

```
1.  ImmutableMap<String, Object> config = ImmutableMap.<String, Object>builder()
2.          .put("bootstrap_servers", options.getBootstrapServer())
3.          .put("topic", options.getTopic())
4.          .put("max_read_time_seconds", “1”)
5.          .build();
6.
7.  Managed.read(Managed.KAFKA).withConfig(readConfig);
```

Redistribute Transform

Partition-Limited Parallelism

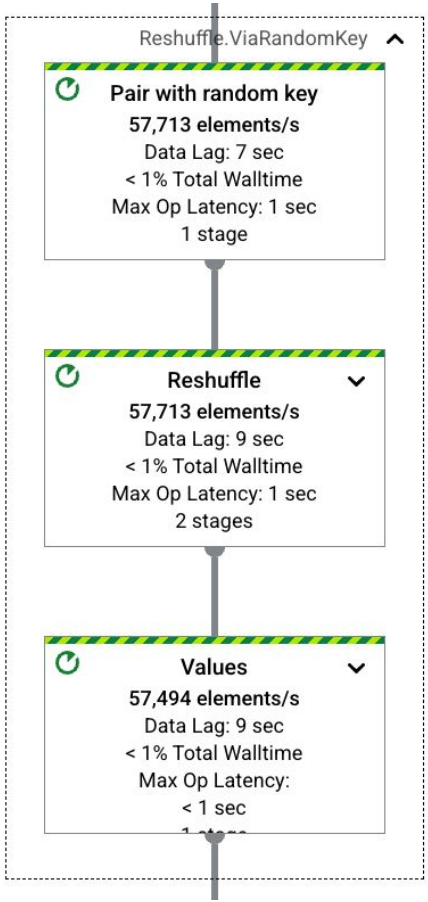
- Problem: Limited Kafka partitions cause limited parallelism (keys)

- Solution: Add parallelism by rekeying inputs via `read.withRedistribute().withRedistributeNumKeys(N)`

Pipeline Translation

Translates to:

- Add key: KV<Key, Record>
- Reshuffle (GroupByKey)
- Remove key, extract values: Record



At Least Once Optimization

- For at-least-once processing, rather than exactly-once.
- Specify: `kafkaRead.withRedistribute().withAllowDuplicates()`
- Why? Runner optimization is much cheaper than regular redistribute.

Pipeline (Kafka To BigQuery)	Throughput (elements/sec)	Hourly cost	Throughput / Cost
Standard	70 K	$1.29	54.3
Redistribute	270 K	$5.89	45.8
Redistribute + Allow Duplicates	280 K	$1.86	150.5

Offset Deduplication

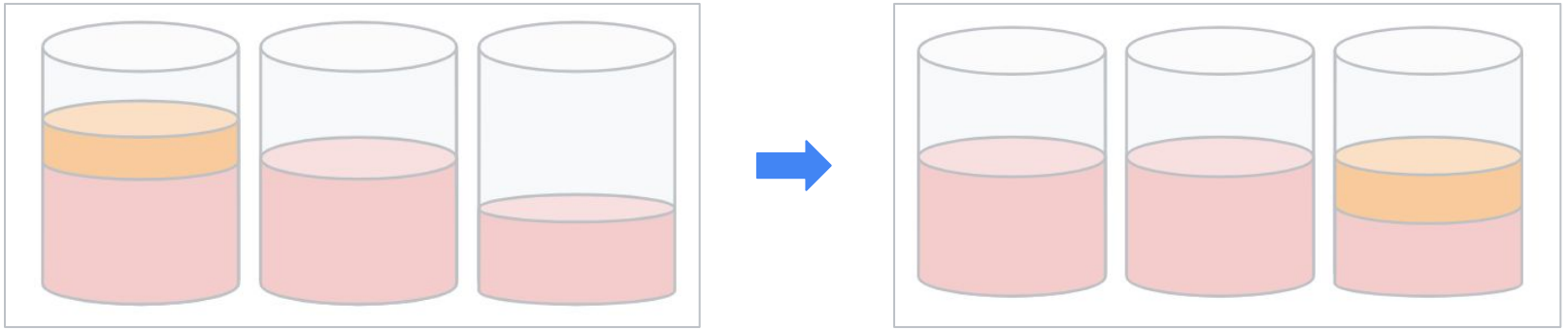
Offset Deduplication

- Problem:
  - Redistribute is expensive due to exactly-once cost and latency.
  - Can’t use allow duplicates optimization because you need exactly-once.
- Solution:
  - Utilize source metadata, such as message offsets, to make shuffle cheaper.
  - Add `withOffsetDeduplication()` to your Kafka read step with Redistribute.
  - Not yet available, coming soon…

Active Load Balancing

Active Load Balancing

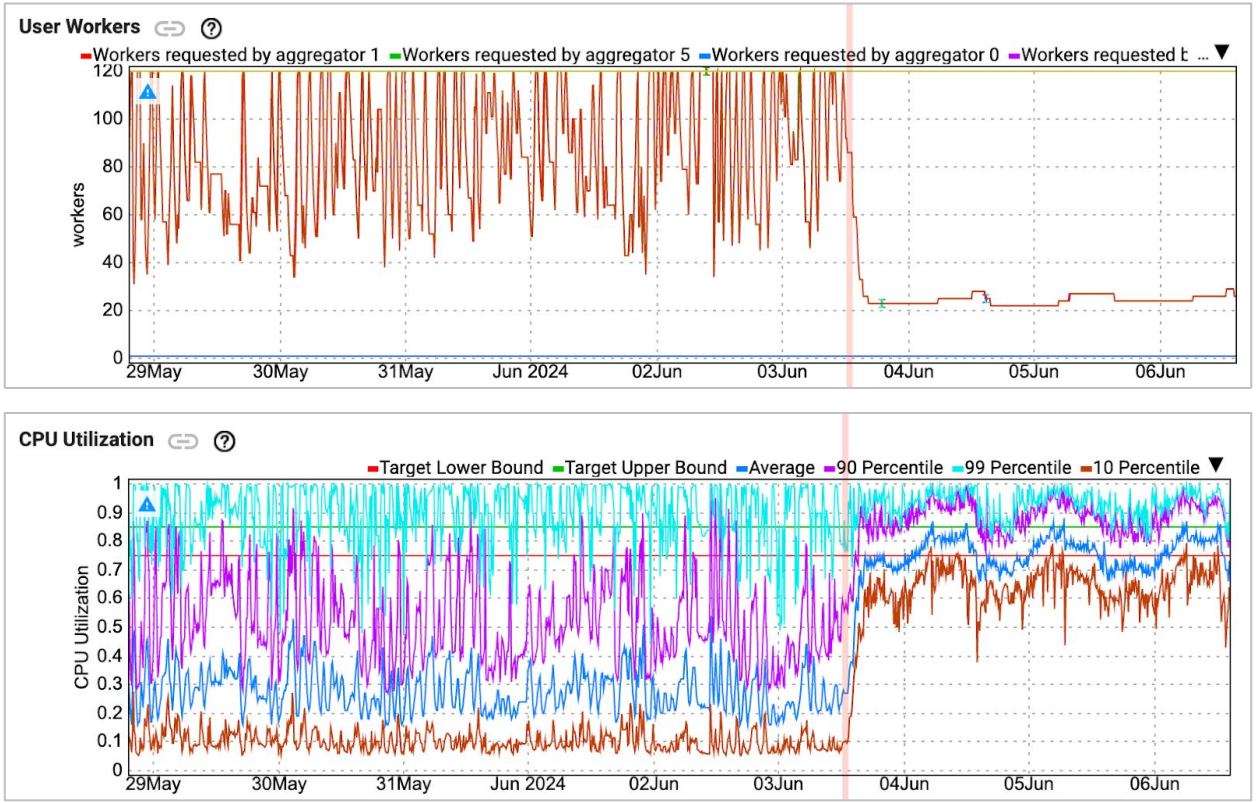
- Distributes load by moving work across workers to improve utilization and performance.
- Without Load Balancing a single worker could become the bottleneck for the entire pipeline.



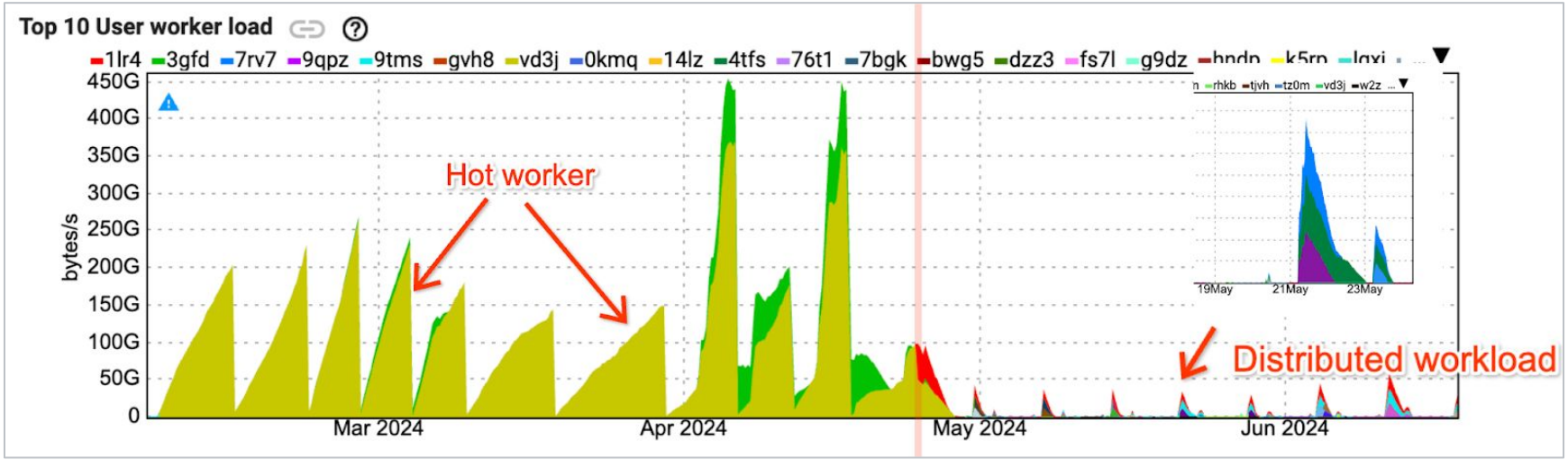
Without Load Balancing

With Load Balancing

Active Load Balancing



Active Load Balancing



Google Managed Kafka

Google Managed Kafka

- How do you setup authentication?
- There are a few configs to get right, example:
  - `Map<String, Object> consumerConfigs = new HashMap<>();`
  - `consumerConfigs.put("sasl.mechanism", "PLAIN");`
  - `consumerConfigs.put("security.protocol", "SASL_SSL");`
  - `consumerConfigs.put("sasl.jaas.config", "org.apache.kafka….PlainLoginModule….");`
  - `read.updateConsumerProperties(consumerConfigs)`

Google Managed Kafka

- Simplified!
  - `read.withGCPApplicationDefaultCredentials();`

- Authenticates with a Google Kafka Server using OAuth

Kafka Best Practices

Parallelism

- Limited by parallelism of the workers and keys (partitions for Kafka IO)
- Increasing partitions is an easy way to do this, but not always possible.
- Another approach is to redistribute the inputs into a larger keyspace with: `.withRedistribute().withRedistributeNumKeys(N)`.
- Specifying a number of keys is recommended.

Multiple Topics

- **Single step**: Create a single instance of the KafkaIO connector and configure it to read multiple topics. Then filter by topic name to apply different logic per topic.
- **Multiple steps**: To read from topics located in different clusters, your pipeline can include several KafkaIO instances.

Committing

- By default, the KafkaIO connector doesn't use Kafka offsets to track progress and doesn't commit back to Kafka.
- Setting `enable.auto.commit=True` commits offsets as soon as they are read from Kafka without any processing by Dataflow, using this option isn't recommended. The recommendation is to set `enable.auto.commit=False` and `commitOffsetsInFinalize=True`.

Watermarks

- By default, KafkaIO uses the current processing time to assign the watermark.
- Beam also provides: `withLogAppendTime` and `withCreateTime`
- Alternatively, set custom behavior: `withTimestampPolicyFactory`.

Client Tuning

- `unboundedReaderMaxReadTimeMs`. Defaults to 10K. Lower values can be used for low latency processing.
- `max.poll.records`. Defaults to 500. A higher value might perform better by retrieving more incoming records together.
- `fetch.max.bytes`. Defaults to 1MB. A higher value might improve throughput by reducing the number of requests.
- `max.partition.fetch.bytes`. Defaults to 1MB. This parameter sets maximum amount of data per partition that the server returns. Increasing the value can improve throughput by reducing the number of requests.

Reference

cloud.google.com/dataflow/docs/guides/read-from-kafka

Bonus Tip

- Use a Kafka schema registry for efficient schema encoding
- Full schema info per record → schema ID with lookup/caching
- A Dataflow customer shared they achieved 40x performance
- Reference: Youtube: How Shopify and Palo Alto Networks use Dataflow...

Thank you!

Questions? Feel free to reach out!

linkedin.com/in/tomstepp

References

- beam.apache.org
- beam.apache.org/…/KafkaIO.html
- github.com/apache/beam
- kafka.apache.org