Usage Billing with BEAM

Narayanan Venkiteswaran
Engineering Manager

Jinjing Bi
Senior Software Engineer

Agenda

1. What is Usage Billing
2. Challenges
3. New System
4. Why Beam

LinkedIn helps companies around the world hire, learn, market and sell

96%

of the Fortune 500 use multiple LinkedIn enterprise products

#1

platform for professional hiring, corporate learning, & sales intelligence

Top 3

B2B digital advertising platform

Usage Billing

- Pay-as-you-go model: Customers are charged based on their actual consumption of a service or resource.
- Threshold Billing: A charge is triggered when usage reaches a predetermined threshold, rather than at fixed time intervals. This can help manage cash flow for both providers and customers.
- Variable costs: Bills fluctuate month-to-month depending on the level of usage, rather than a fixed recurring fee.
- Transparent pricing: Customers typically have access to detailed breakdowns of their usage and associated costs.

Usage Billing

Jobs

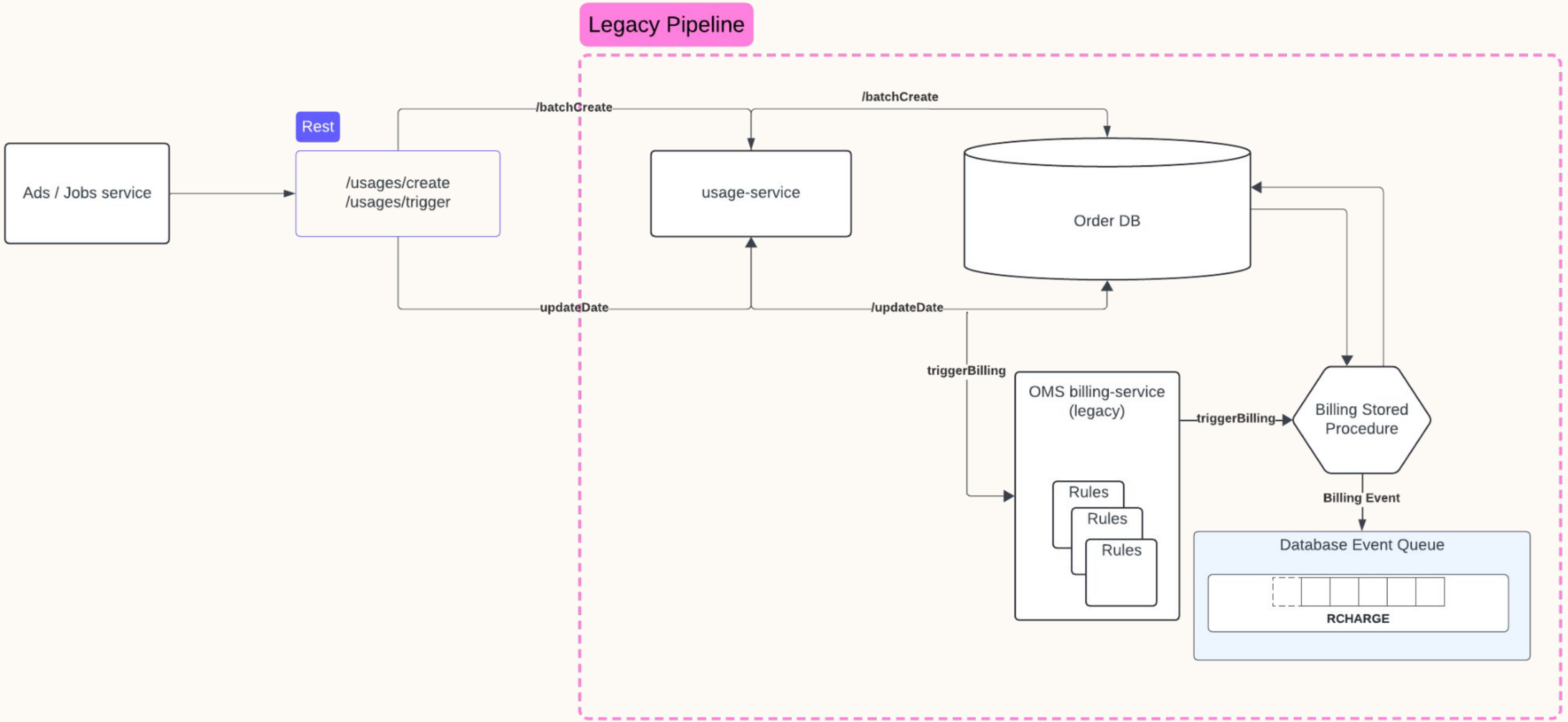
Your charge amount depends on your job posts' budget and the number of views from candidates.

Ads

Your charge amount depends on dynamic, auction-based system associated with your ad.

Usage Billing

Legacy Usage Billing



Challenges

Batch

- The current system was built to support only batched workloads
- The whole process was in the hands of the callers to trigger the complete batch
- Over time, daily batch processing becomes increasingly long stretching execution windows.

Legacy Stack

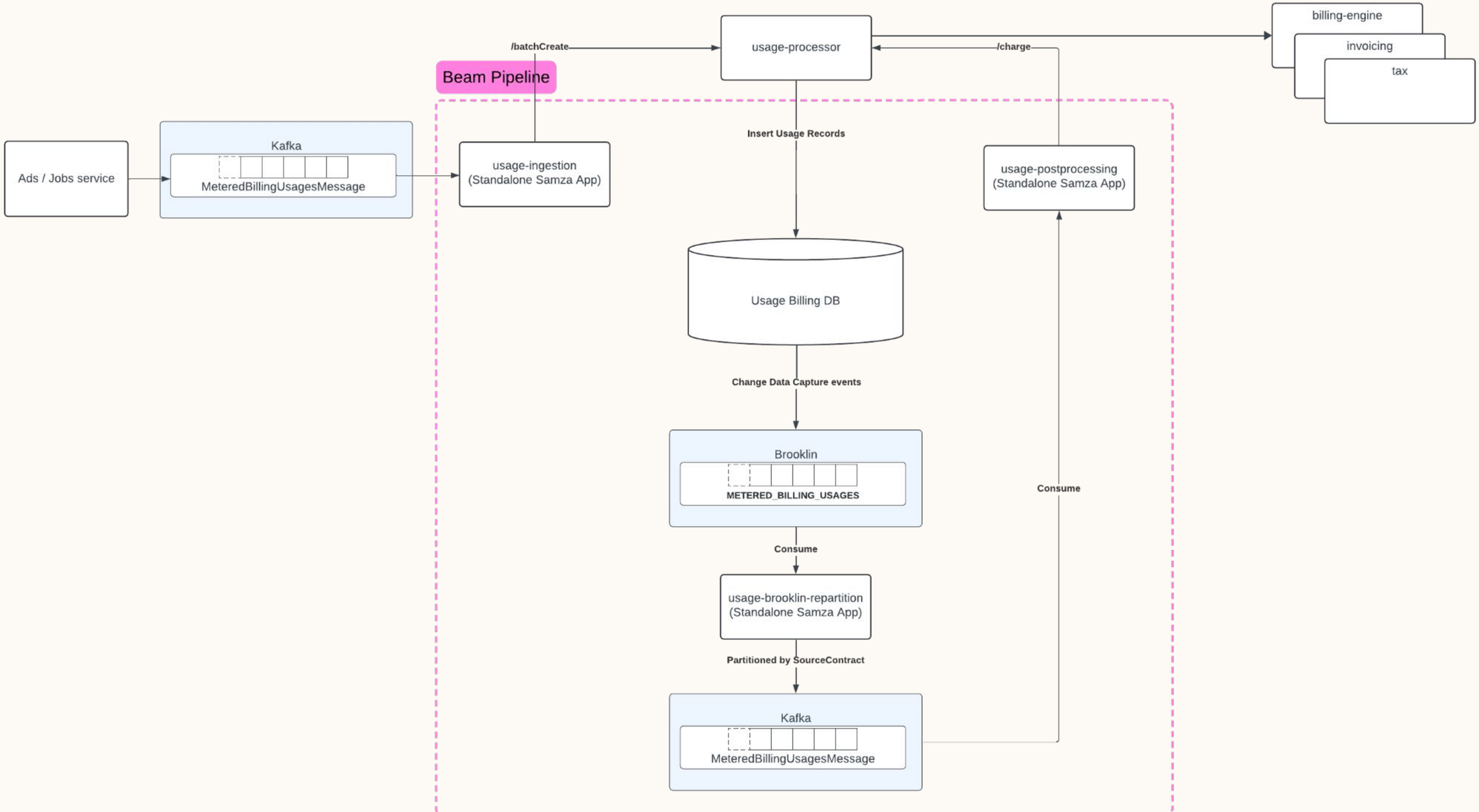
- Used combination of Stored Procedures and Oracle AQ
- Code was written very consumer specific and not a platform friendly to support additional use case
- Scaling problems to support higher loads

Fault Tolerance

- Hard to rectify usages once sent
- Can fail the batch if there were problems with the usages like encoding, bad passed past by upstream.

Usage Billing

New



Three Samza Jobs

Usage-Ingestion

- Reads messages from Kafka, unpacks individual usages, applies various filters, and performs idempotency checks.
- De-dupes and aggregates intraday usages.
- Dead Letter Queue Management

Usage-brooklin-repartition

- Reads usage data from a Kafka topic and repartitions it based on specific criteria to optimize downstream processing.
- Applies various filters and transformations to the data, ensuring that only valid and necessary records are processed.
- Output to Kafka for consumption and processing.

Usage-Postprocessing

- Processes usages with specific states such as CREATED, HOLD, and DELIVERED
- Discards usages with negative gross or discount amounts.
- Dead Letter Queue Management

How Beam is Useful

- Supports both batch and streaming
- Supports multiple processing tiers - Samza, Flink etc
- Usage billing needs sophisticated windows which Beam can support like
  - Time based
  - Amount based
  - Action based
- Exactly-Once Processing support which needed for a monetary system

Sample Code

```
final PCollection<MeteredBillingUsagesMessage> input = readFromKafka(pipeline);
final PCollection<KV<com.linkedin.messages.lbp.billing.records.MeteredBillingUsage, ProductLineUrn>>
    flattenedUsages =
        input
            .apply(ParDo.of(new UsageFlattenTask()))
            .setCoder(
                KvCoder.of(
                    AvroCoder.of(com.linkedin.messages.lbp.billing.records.MeteredBillingUsage.class),
                    ProductLineUrnCoder.of()))
            .apply("Filter the usages based on Source Contract", ParDo.of(new SourceContractFilterTask()));

// Get pointer to Idempotency Rocks db table
final PSeekableCollection<UsageEventIdempotencyKey, UsageEventIdempotencyValue> idempotencyTable =
    getIdempotencyTable(pipeline);

// Write the Flattened Usages into the rocks db.
writeToIdempotenceStore(flattenedUsages, idempotencyTable);

// Check for each usage inside MeteredBillingUsagesMessage event if it has been processed
// already or not.
final PCollectionTuple idempotencyCheckOutput =
    flattenedUsages.apply(
        ParDo.of(new IdempotencyCheckTask(idempotencyTable)).withOutputTags(UNIQUE, TupleTagList.of(DUPLICATE)));

// All failures that happen in any stage of this pipeline are accumulated here and eventually
// written to a DLQ for retries
final List<PCollection<PipelineFailure>> failurePCollection = new ArrayList<>();

// Any failures that have to do with invalid messages will be accumulated here
// and written to the invalid message queue for LOBs to consume.
final List<PCollection<PipelineFailure>> invalidMessagePCollection = new ArrayList<>();

// Unpack the processed usages and convert them to MeteredBillingUsage PDL messages, keyed by
// SourceBillable
final PCollectionTuple usageUnpackTaskOutput =
    idempotencyCheckOutput
        .get(UNIQUE)
        .apply(
            ParDo.of(new UsageUnpackTask())
                .withOutputTags(USAGE_UNPACK_SUCCESS_TAG, TupleTagList.of(USAGE_UNPACK_FAILURE_OUTPUT_TAG)));
```

Questions?