Netflix Technology Blog Follow
Learn more about how Netflix designs, builds, and operates our systems and engineering organizations
Feb 15, 2016 · 4 min read

Evolution of the Netflix Data Pipeline

Our new Keystone data pipeline went live in December of 2015. In this article, we talk about the evolution of Netflix's data pipeline over the years. This is the first of a series of articles about the new Keystone data pipeline.

Netflix is a data-driven company. Many business and product decisions are based on insights derived from data analysis. The charter of the data pipeline is to collect, aggregate, process and move data at cloud scale. Almost every application at Netflix uses the data pipeline.

Here are some statistics about our data pipeline:

- ~500 billion events and ~1.3 PB per day
- ~8 million events and ~24 GB per second during peak hours

There are several hundred event streams flowing through the pipeline. For example:

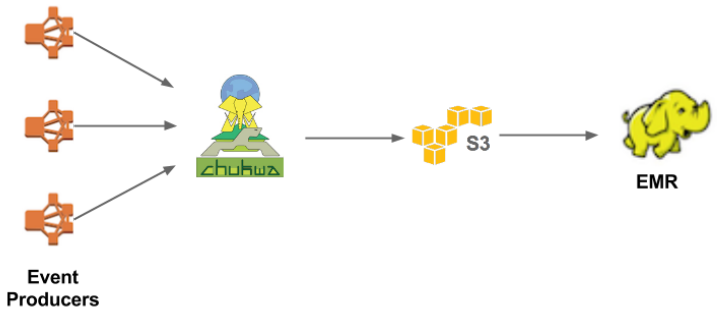
- Video viewing activities
- UI activities
- Error logs
- Performance events
- Troubleshooting & diagnostic events

Note that operational metrics don't flow through this data pipeline. We have a separate telemetry system Atlas, which we open-sourced just like many other Netflix technologies.

Over the last a few years, our data pipeline has experienced major transformations due to evolving requirements and technological developments.

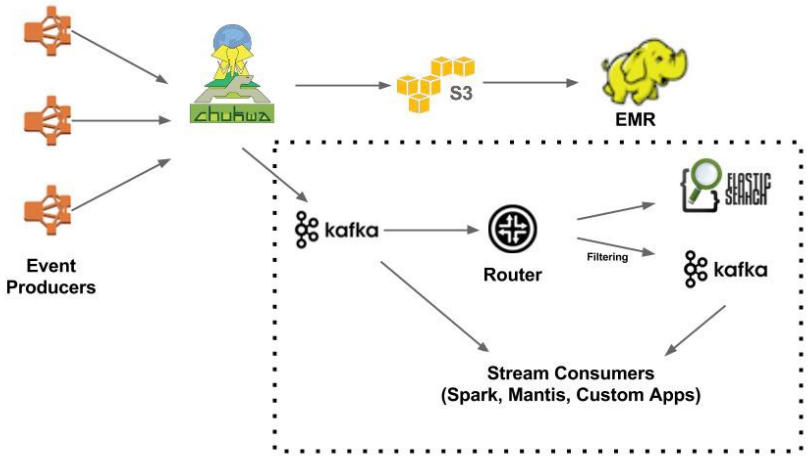
V1.0 Chukwa pipeline

The sole purpose of the original data pipeline was to aggregate and upload events to Hadoop/Hive for batch processing. As you can see, the architecture is rather simple. Chukwa collects events and writes them to S3 in Hadoop sequence file format. The Big Data Platform team further processes those S3 files and writes to Hive in Parquet format. End-to-end latency is up to 10 minutes. That is sufficient for batch jobs which usually scan data at daily or hourly frequency.



V1.5 Chukwa pipeline with real-time branch

With the emergence of Kafka and Elasticsearch over the last couple of years, there has been a growing demand for real-time analytics in Netflix. By real-time, we mean sub-minute latency.



In addition to uploading events to S3/EMR, Chukwa can also tee traffic to Kafka (the front gate of real-time branch). In V1.5, approximately 30% of the events are branched to the real-time pipeline. The centerpiece of the real-time branch is the router. It is responsible for routing data from Kafka to the various sinks: Elasticsearch or secondary Kafka.

We have seen explosive growth in Elasticsearch adoption within Netflix for the last two years. There are ~150 clusters totaling ~3,500 instances hosting ~1.3 PB of data. The vast majority of the data is injected via our data pipeline.

When Chukwa tees traffic to Kafka, it can deliver full or filtered streams. Sometimes, we need to apply further filtering on the Kafka streams written from Chukwa. That is why we have the router to consume from one Kafka topic and produce to a different Kafka topic.

Once we deliver data to Kafka, it empowers users with real-time stream processing: Mantis, Spark, or custom applications. "Freedom and Responsibility" is the DNA of Netflix culture. It's up to users to choose the right tool for the task at hand.

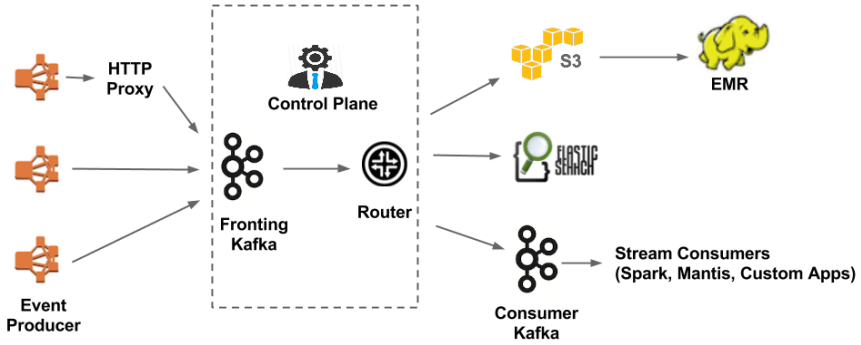
Because moving data at scale is our expertise, our team maintains the router as a managed service. But there are a few lessons we learned while operating the routing service:

- The Kafka high-level consumer can lose partition ownership and stop consuming some partitions after running stable for a while. This requires us to bounce the processes.
- When we push out new code, sometimes the high-level consumer can get stuck in a bad state during rebalance.
- We group hundreds of routing jobs into a dozen of clusters. The operational overhead of managing those jobs and clusters is an increasing burden. We need a better platform to manage the routing jobs.

V2.0 Keystone pipeline (Kafka fronted)

In addition to the issues related to routing service, there are other motivations for us to revamp our data pipeline:

- Simplify the architecture.
- Kafka implements replication that improves durability, while Chukwa doesn't support replication.
- Kafka has a vibrant community with strong momentum.



There are three major components:

Data Ingestion—
There are two ways for applications to ingest data:

1. use our Java library and write to Kafka directly.
2. send to an HTTP proxy which then writes to Kafka.

Data Buffering—
Kafka serves as the replicated persistent message queue. It also helps absorb temporary outages from downstream sinks.

Data Routing—
The routing service is responsible for moving data from fronting Kafka to various sinks: S3, Elasticsearch, and secondary Kafka.

We have been running Keystone pipeline in production for the past few months. We are still evolving Keystone with a focus on QoS, scalability, availability, operability, and self-service.

In follow-up posts, we'll cover more details regarding:

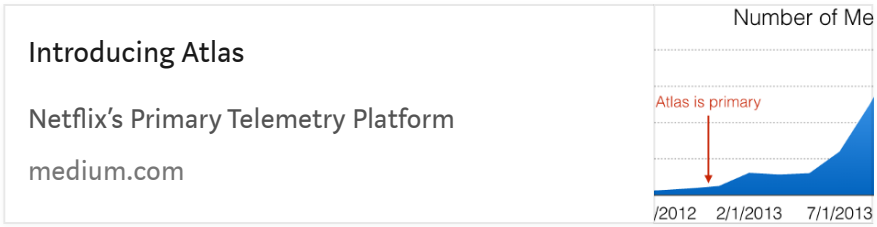
- How do we run Kafka in cloud at scale?
- How do we implement routing service using Samza?

- How do we manage and deploy Docker containers for routing service?

If building large-scale infrastructure excites you, we are hiring!

— Real-Time Data Infrastructure Team
Steven Wu, Allen Wang, Monal Daxini, Manas Alekar, Zhenzhong Xu, Jigish Patel, Nagarjun Guraja, Jonathan Bond, Matt Zimmer, Peter Bakas

See Also:



. . .

Originally published at techblog.netflix.com on February 15, 2016.