### Introduction

[Andrew Nguonly](https://www.linkedin.com/in/andrewnguonly/) has spent the last several years building data infrastructure and pipelines at companies ranging from early-stage startups to tech giants. Most recently, Andrew was the first data engineer at CartaX, where he bootstrapped the company's data platform from scratch. Before that, Andrew worked as an engineer on Netflix's streaming data infrastructure team, building tools and services used by Netflix's army of data engineers. In this blog post, Andrew shares his experience and advice.

### Andrew's Experience at CartaX

CartaX is a secondary market trading platform for startup equity. It serves as a partner product to Carta, which is software to manage startups’ cap tables. However, since CartaX is technically a separate company, it couldn’t use the same existing data infrastructure as Carta. As the first data engineer at CartaX, Andrew was tasked with building the company's data infrastructure from the ground up. He implemented the company's batch and streaming data pipelines, loading data into a Redshift data warehouse. According to Andrew, "The main challenge was learning the DevOps function of data engineering," like using Terraform. Once the infrastructure was in place, the work consisted mainly of data modeling and SQL.

The primary consumers of data at CartaX were Looker dashboards and reporting. As Andrew describes, "The executives at CartaX were most interested in business metrics (as opposed to product metrics). This included various metrics around the volume and nature of transactions on the trading platform."

#### Data SLAs at CartaX

The data quality issues that Andrew had to deal with at CartaX were less technical issues around ELT or source data, and more the consequence of mismatched expectations between data producers and consumers. For example, some of the consumers of the data just assumed that all of the data was always up to date, even though it was being updated in batch. “There were a lot of discussions where it was like, okay, we put something out there, the consumer thought it was something else, and then we had to come to a compromise of what the expectation or SLA is. Once that was kind of resolved, the issue was kind of over,” Andrew says. This process of setting data SLAs was fairly ad hoc and reactive. “We would wait for the consumers to come to us with their comments or complaints or requests, and that's when we would discover that there was a mismatch. We did not formalize an SLA by programmatically enforcing it. We just formalized it in the documentation. We had our internal monitors and metrics to make sure that our pipelines were always producing data within an expected time range. But it wasn't sophisticated by any means.”

### Andrew's Experience at Netflix

At Netflix, Andrew worked on the Keystone Data Pipeline, Netflix's real-time data infrastructure ecosystem. He helped build products like Data Mesh, a "next-generation data pipeline for change-data-capture use cases," and handled operational responsibilities for Netflix's Apache Kafka and Apache Flink clusters. Andrew's role at Netflix differed from CartaX in several key ways:

1) Scale - Netflix's data and infrastructure were orders of magnitude larger than CartaX's.

2) Focus on streaming - Andrew worked mainly with streaming tools and pipelines rather than batch data.

3) Building infrastructure rather than using it - At Netflix, Andrew built the tools and services used by data engineers. At CartaX, he was the data engineer using those tools.

The main needs Andrew's team addressed were abstraction and ease of use. They built an internal web UI allowing data engineers to construct streaming data pipelines by dragging and dropping, handling all the underlying infrastructure provisioning and data routing automatically. The data sources, Kafka topics, were created and maintained by application teams, while Andrew's team provided documentation and client libraries.

#### Data quality for real-time data

Ensuring data quality for real-time data was an ongoing challenge. “It was extremely difficult and I think the expectation with streaming data is more lax,” Andrew says. Netflix primarily used a batch reconciliation process that adhered to the Lambda architecture. This involved maintaining two parallel data pipelines: a real-time pipeline and a batch dataset. The batch dataset would then override the real-time data at scheduled intervals to confirm the accuracy and completeness of the data.

Internally, for their product, Netflix established "at-least-once" delivery semantics, meaning consumers had to perform deduplication and account for the possibility of duplication. In rare cases where a bug caused the real-time data producer to emit an entirely incorrect dataset, Netflix typically discarded that dataset entirely. They restarted the process to reproduce a new correct dataset from scratch instead of trying to overwrite or fix the erroneous real-time data.

### Andrew's Experience at Honey

Earlier in his career, Andrew was the first data engineer at Honey. Honey is a coupon code Chrome extension that tries to automatically apply coupon codes for you when you're on a checkout page on certain e-commerce sites.

While at Honey, Andrew worked on real-time data pipelines that powered auxiliary features like price drop notifications and product search. This involved taking data that had been scraped from product pages that customers browsed on, running it through an ETL pipeline, and constructing an e-commerce product catalog from it.

Unlike CartaX, which was an AWS shop, Honey’s data stack ran on GCP. Andrew used BigQuery, Pub/Sub and Apache Beam.

#### Data quality issues at Honey

A common data quality issue at Honey was incorrect price drop alerts. Honey relied on users visiting product pages to trigger the collection of pricing data. If Honey did not receive an update for a product's price within a given time window, it would consider the last seen price to be the current actual price for that product. This often led to incorrect price drop alerts being fired.

For example, Honey's scrapers might scrape a product page and find a price of $100 for a product, which matched the price seen over the last 30 days. Then, with no interim updates, the scrapers might suddenly see a price of $2 for that same product. Even though this was clearly an error, without receiving another update within the time window, Honey's system would consider $2 to be the actual current price and incorrectly alert users to a major price drop.

Honey's technique for detecting outliers at the time was fairly naive. They simply checked whether a new price fell within a specified standard deviation of the average price for that product over the last 30 days. More sophisticated anomaly detection could have improved their price drop alert accuracy. For example, tools like Apache Beam and Apache Flink offer more advanced windowing features to control how far back to consider old data and when to "close" a window and consider only new data.

### Key Learnings

Andrew's career has given him a uniquely broad perspective on data engineering and infrastructure. Some of his biggest learnings have been:

• **Empathy for infrastructure operators.** Experience building and running data infrastructure at Netflix gave Andrew an appreciation for the pressures of operating managed services. “To give a concrete example, when I use a managed service now, and I see the managed service go down, I know exactly what those guys are going through. They're probably stressed out as h\*\*\*. They're probably running around trying to figure out, oh, how do I get this thing back up and running for my customer? That's basically what we were doing at Netflix when the data pipeline was stuck. Our consumers or our users just expect it to be running. As a data engineer, sometimes you don't know what it takes to operate one of these managed services at scale.”

• **The product management aspect of data engineering.** “You have to be the product manager of your data set and you’ve got to own the lifecycle of it. You’ve got to manage the migrations for it. You’ve got to communicate to your consumers when you're breaking something or changing something.”

• **The difficulty balancing abstraction and customizability.** There is no formula for designing the perfect level of abstraction. It requires navigating the complex needs and priorities of users and the limitations of the underlying technology. “That was one very tricky thing to kind of balance at Netflix. There were use cases where we simply had to deny the user. We said, hey, we cannot do what you want, even though the underlying technology can do it. Partly because of resources and priorities. Partly because of a lack of demand. But then partly because it conflicted with our view of what the abstraction should be. You do have to kind of be opinionated when you're building a higher-level abstraction. Otherwise, you end up a jack of all trades product but master of none.”

• **The importance of operational excellence.** “Moving forward to Carta, going back into the traditional data engineer function, I wanted to bring the principles of operational excellence that I got from Netflix into the data engineering function at Carta. I wanted to make sure that we had all of the monitoring and observability in place to ensure that our data pipelines are running all the time and that we could be independent from the SRE team. We didn't have to rely on them at all. Maybe I would not have done that if I hadn't had that experience at Netflix.“