Tracing at Scale

Running a pipeline of billions of spans per hour at Nubank

Kaynan Coelho · Felipe Araujo
KubeCon + CloudNativeCon North America 2026 · Salt Lake City



Meet the speakers



Felipe Araujo
Lead Software Engineer

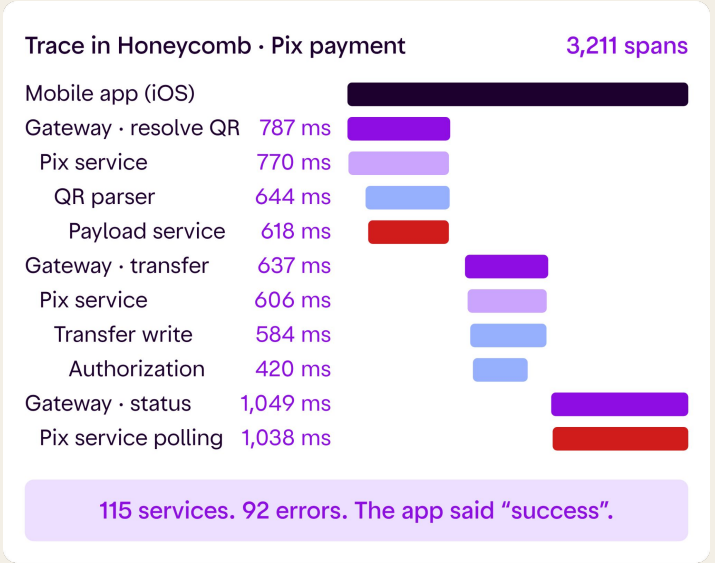


Kaynan Coelho
Engineering Manager

One tap. Many Services.

A customer opens the app and pays a bill. That single action produced 3,211 spans across 115 services before the confirmation shows up.

Now multiply that by million customers in Brazil, Mexico and Colombia, plus our new operation in the United States.



Tracing at scale

Our talk proposal said 16 billion spans per hour. We measured again in **October 2026.**

139M
Customers

4
Countries

+6k
Microservices

~23B
Spans per hour

~700TB
Span data per day

5
Engineers running it

Customers: Nu Holdings Q2 2026 results · Platform data: Internal measurement in October 2026

Nubank

Part 1

Getting the first trace

Kaynan Coelho

Three rules for the proof of concept

01
Mobile first

We started where customers feel latency: the app.

02
Instrumentation Once

One change covers the whole fleet.

03
Built to extend

If the first version works, the second one should not start from scratch.

Canonicity: solve the same problem the same way everywhere

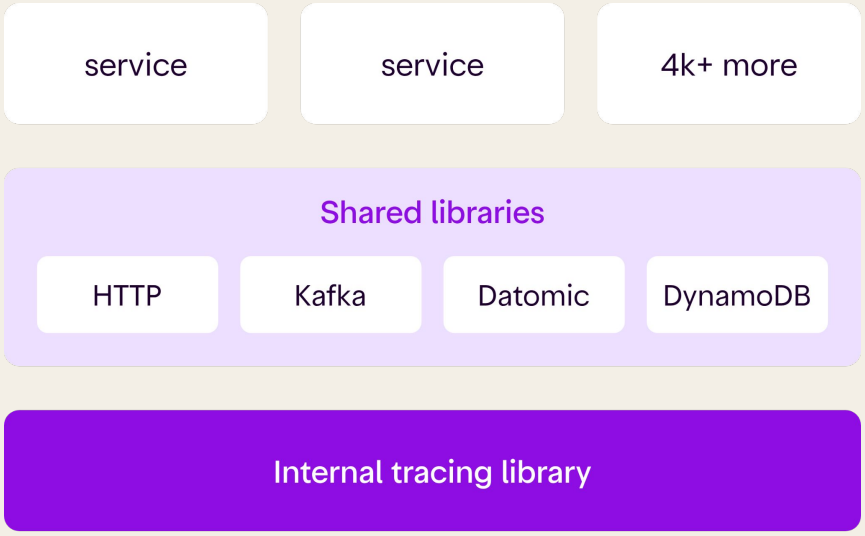
Most of our services run on Clojure, on the same service template and the same shared libraries. Any engineer can open another team's repo and find their way around.

A horizontal team ships one fix, and every service gets it.

That's the property tracing needed.

So we instrumented 6 libraries instead of 4k+ services

We built an internal tracing library that is used in two different ways, by the shared libraries responsible for HTTP, Kafka, Datomic and DynamoDB and also with possibility to be used by directly into the services.



We picked the OTel Collector over our own log aggregator



Build vs reuse

Our log aggregator was battle-tested, but it couldn't speak OTLP.

A collector tier sits between all services and the vendor. Changing backends means changing exporter config, and no service gets redeployed.

It's a CNCF project, maintained by the community in this room.

V1 proved the idea

0.5%

of Brazil transactions. Enough to prove the idea without a big bill.

Deterministic head sampling

Every collector's probabilistic sampler uses the same hash_seed. Each pod makes the same keep-or-drop call for a trace ID without talking to the others.

We left the full sampling strategy, cost per team and opt-out tooling for later phases. On purpose.

Nubank

Part 2

Building the ingestion platform

Felipe Araujo

Four requirements for the platform

Scalable & Elastic

Traffic on payday is 1.5x a normal day, and the baseline grows every year.

Fail in pieces

Every component of the platform should be fault tolerant to prevent a full outage.

Transport Data

The platform needs to be able to transport petabytes of data in an effective way.

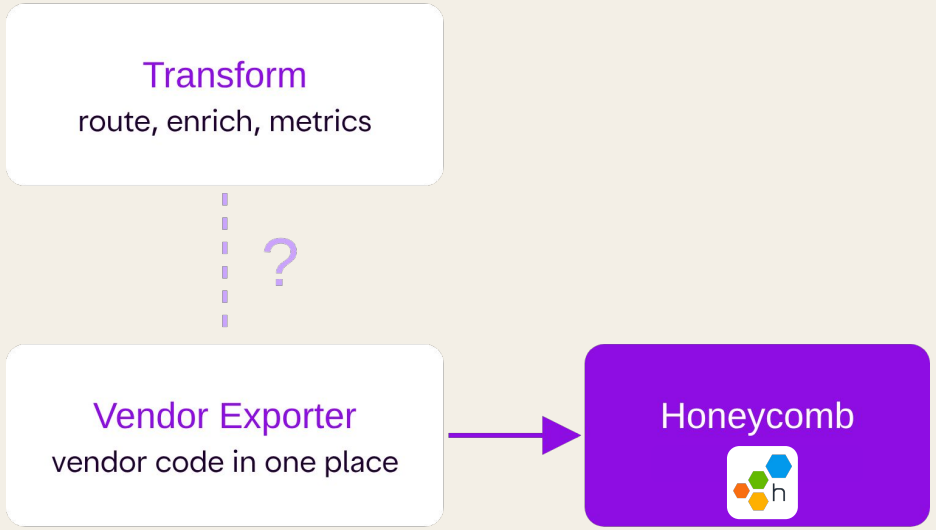
Own our data

Every span had to pass through code we run before it reached Honeycomb.

We started with the two ends we knew

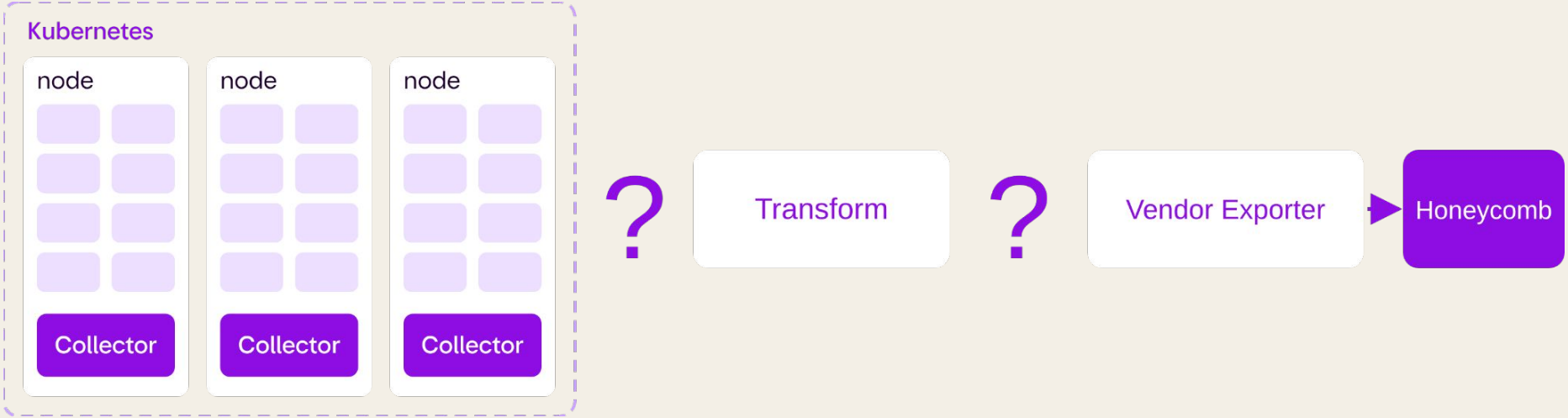
A Transform app, because we wanted route spans, enrich them and turn them into data we own.

A Vendor Exporter, so a vendor outage or API change stays in one place. Its destination has been Honeycomb since the first day.



The hard part was everything in between

All data was sent to otel collector. Given the goal of controlling our data being able to generate metrics from those traces, the core challenges became: How do we reliably and cost-effectively transport such a massive volume of data?



Option 1, batch: simple and cheap, but too slow

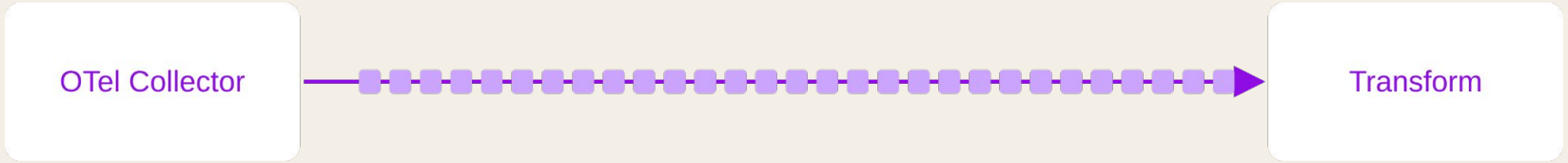
The Batch Processing solution was not feasible even being simpler and cheaper, spans data can't have its ingestion delayed too much.



Freshness ~5 minutes Relative cost Lowest

Option 2, stream: instant, but too expensive

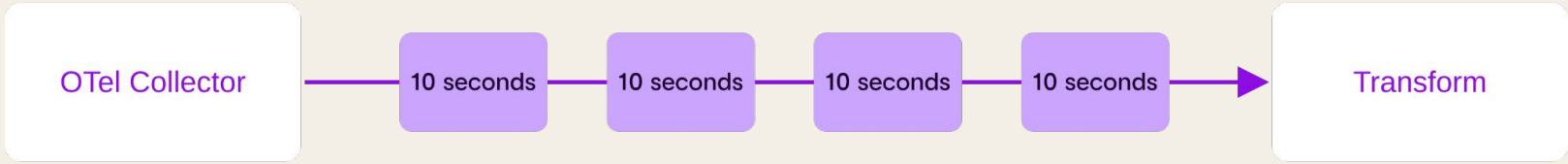
Stream processing was the ideal solution, it would deliver the spans almost instantaneously, but the cost of it was prohibitive.



Freshness Instant Relative cost Higher than batch

Option 3, micro-batching: fresh enough, cheap enough

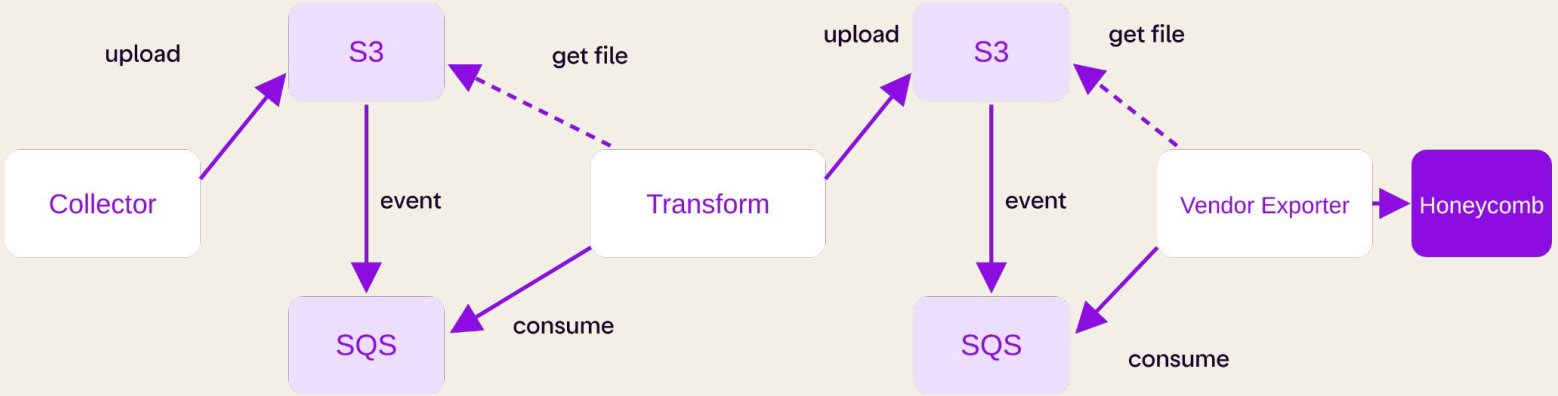
The solution was to rely on a **short window** to do the ingestion and apply a micro-batching architecture, to still have up-to-date data being ingested while being cost effective.



Freshness ~few seconds Relative cost Really cheaper than streaming

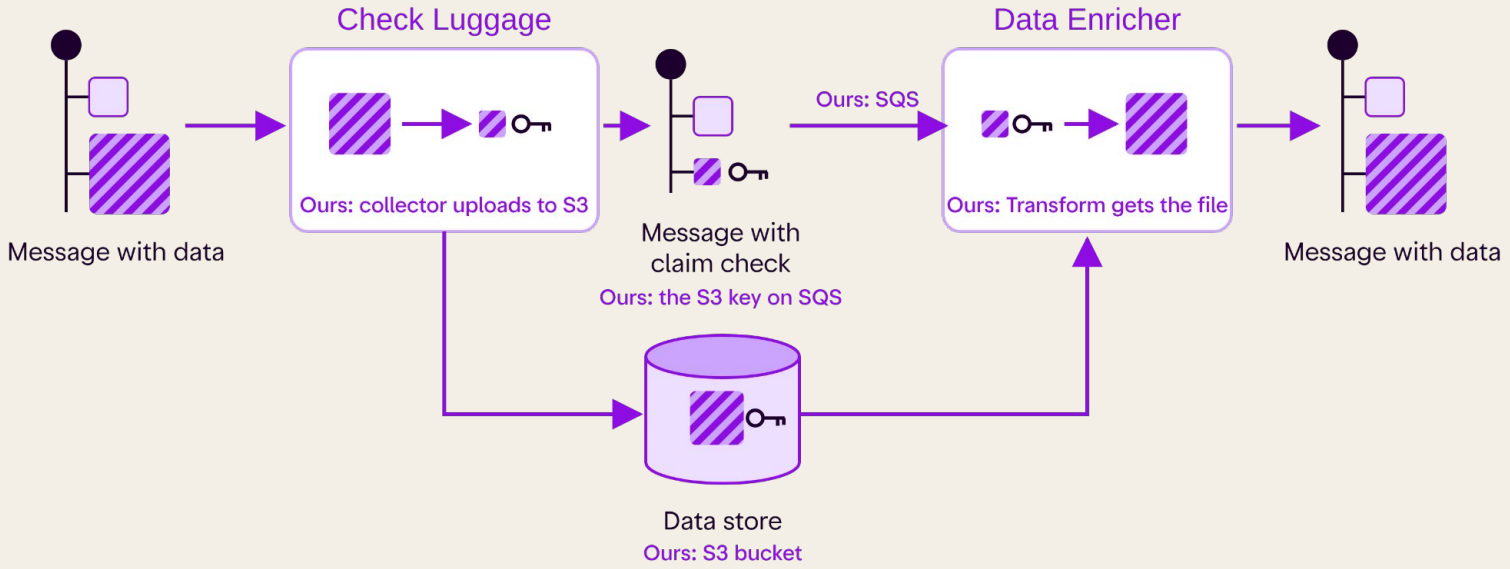
Two AWS services carry the whole stream: S3 and SQS

The collector uploads a gzip micro-batch of a few MB to S3, and S3 puts an event on SQS. The consumer reads the event, fetches the file, does its work and uploads the result. The next stage repeats the same steps.



If it looks familiar, it's Claim Check

Hohpe and Woolf described it in 2003. We use it to move ~700 TB of spans a day.



Hohpe and Woolf, Enterprise Integration Patterns (2003), "Store in Library"

Queue depth is our autoscaling signal

More messages in SQS means more data waiting. KEDA reads the queue and scales consumers up or down.

Event Driven

CPU says little about how much data is waiting. Queue depth says exactly that.

Kubernetes Native

KEDA feeds the HPA, so it scales like any other workload in the cluster.



Nubank

Part 3

Keeping what matters

Felipe Araujo

Tail sampling: decide after seeing the whole trace

Head, at the collector

- Decides on the first span
- Stateless and cheap
- Can't see errors or latency

Tail, with Refinery

- Decides when the trace is complete
- Keeps errors and slow traces on purpose
- Open source, last step of the stream

First rules: keep 1% of all traffic, 3% of Brazil error traces and 100% of mobile traces.

75% of what we stored was a random 1% slice



75.1% baseline, kept at 1%

24.0% errors, 3%

0.66% mobile · 0.24% other

An ETL that runs 10 million times a day got the same treatment as a payment dispute flow that runs 50 times a day.

Brazil production, 24 hours

Turning the dial up doesn't fix it

Endpoint	Traces a day	Kept at 1%	Kept at 3%
High-volume read path	100,000,000	1,000,000	3,000,000
Medium-volume business flow	100,000	1,000	3,000
Low-volume admin endpoint	1,000	10	30
Rare critical path, like a dispute flow	50	0–1	1–2

Flat-rate sampling keeps data in proportion to traffic, not to value. We needed a different rule, not a bigger dial.

New rules: sample by what a trace is

Each rule keeps a trace for what it is: slow, failing or rare. Fast, healthy traffic is kept lightly.

01
Keep slow traces

Every trace over 800 ms is kept.

02
Compress fast success

Fast, successful traces, under 200 ms, are sampled lightly.

03
Keep server errors

Every 5xx error is kept.

04
Sample the rest

A small share of everything the rules above didn't match.

05
Sample other errors

A separate small share of errors no other rule matched.

Rare errors are no longer lost to chance

+888

services now have error visibility(+84%). Almost half had never appeared in Honeycomb before.

19% → 0.6%

Fast 2xx traces were 19% of incoming traffic. The most common, least useful traces now take almost no space.

97%

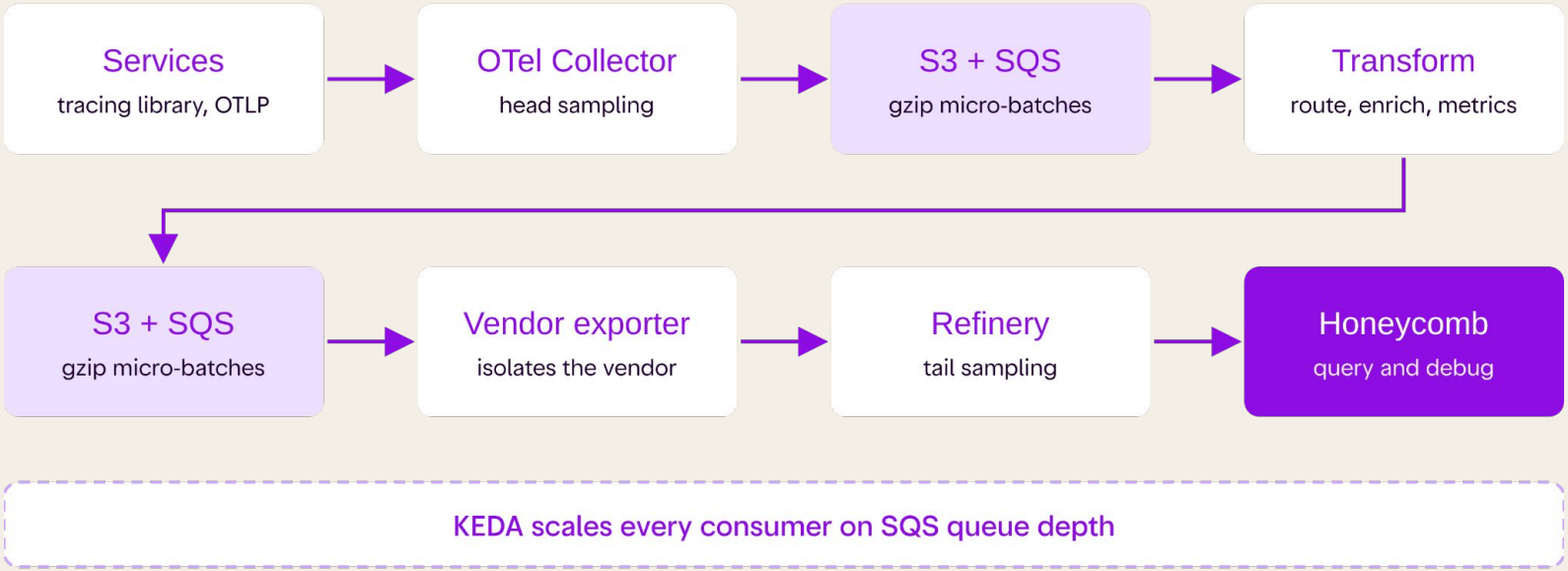
chance a one-off error was dropped before

Now the dynamic sampler sees it is rare and keeps it.

75% → 61%

Random baseline share of storage, still falling as the new rules settle.

Putting it together: the platform end to end



Part 4

Highlights & Takeaways

Kaynan Coelho

Home screen p99: from up to 15 s to 200 ms

Symptom

Traffic jumped 300% in two minutes, and the home screen p99 swung between 2 and 15 seconds.

The Trace

The home screen waits on ~30 content providers owned by different business units. The trace showed which one was the slowest.

The Fix

A cache on the overloaded service, interceptor fixes for spikes during deploys, and more headroom on the main services.

Result

p99 at 200 ms, against a 500 ms goal. CPU down ~8 pp, and server errors at peak cut by 25–35%.

The bottleneck is rarely where everyone is looking

Global Account

An Incident avoided before it happened

Ahead of our expansion, the team traced the account-movement flows. Everyone suspected the database path.

The real bottleneck was repeated calls to a third party's API. Fixed in time, it never became a large-scale incident.

Datomic

Three fixes, found and verified with traces

Indexing improvements, efficiency changes and removing audit datoms, rolled out in 2026.

Removing audit datoms eliminated ~60% of everything Datomic wrote. Compute use is projected to drop ~15%.

If you're building this, inspire on these

1. Instrument your shared libraries. With canonicity, that covers the whole fleet.
2. Move telemetry with Claim Check: payload to object storage, pointer on the queue.
3. Scale consumers on queue depth with KEDA.
4. Analyze your own user case, even what we did maybe doesn't fit what you need.

Thank you

Questions?

Felipe Araujo · https://www.linkedin.com/in/felipearaujos/
Kaynan Coelho · https://www.linkedin.com/in/kaynancl/

We're hiring. Scan to see open roles.

https://nu.com/en/careers

Feedback

Tracing at Scale: Running a 16 Billion Spans/Hour Pipeline at Nubank

Session feedback