NUBANK

Breaking Free from Vendor Lock-In

Nubank DIY Observability Success

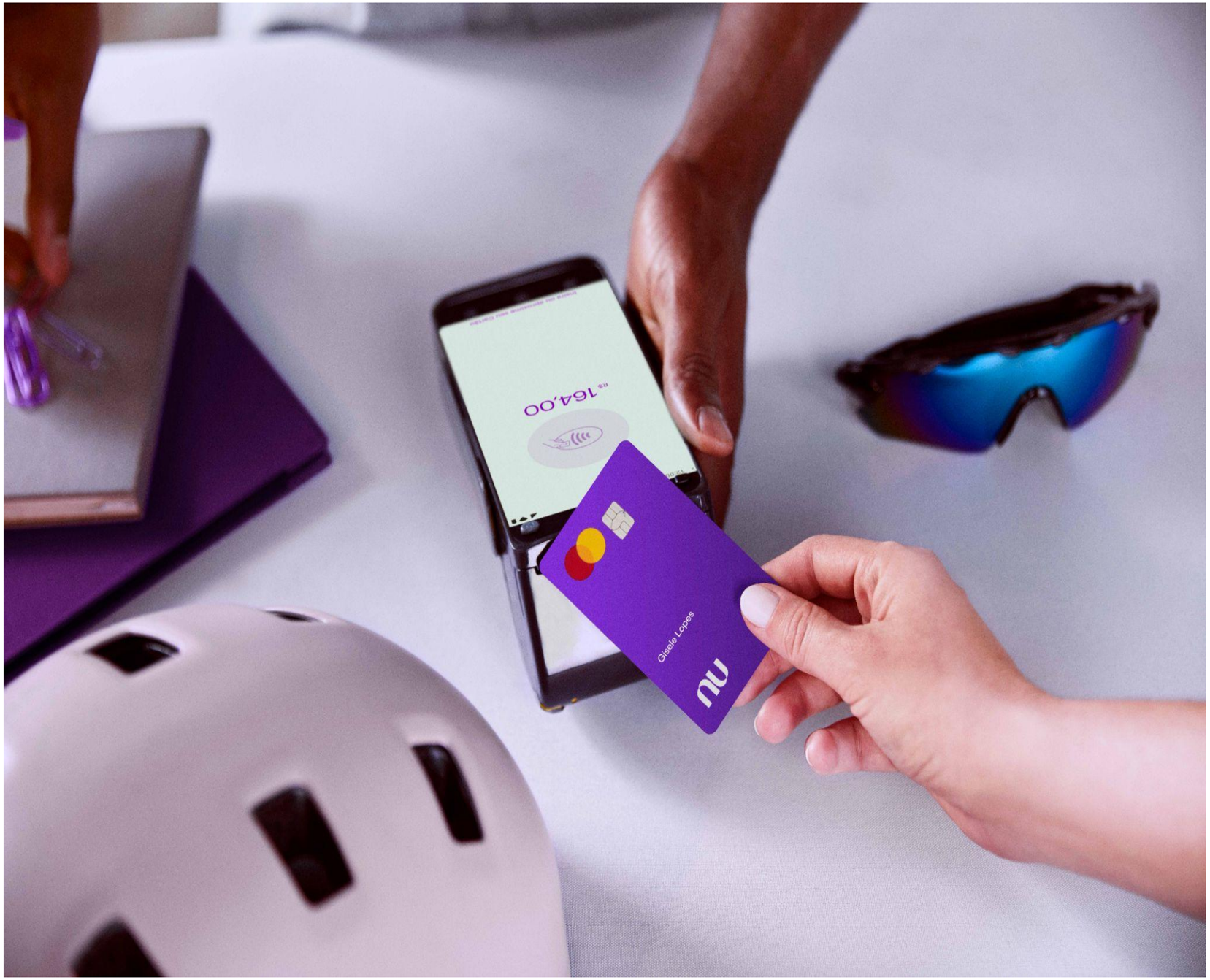
Reliability

April 2026

Meet the speakers

Otavio Valadares
Lead Software Engineer

Diego Rocha
Solutions Architect



What is Nubank?

Nubank was founded in 2013 and today is one of the largest platforms of digital financial services in the world. Across all countries, Nubank has more than 130 million customers.

Some data to understand our scale

+130 million

Customers Worldwide.

3 countries

Brazil, Mexico and Colombia. (Entering the USA)

US$16.3 Billion

Revenue in 2025.

US$2.9 Billion

Profit in 2025.

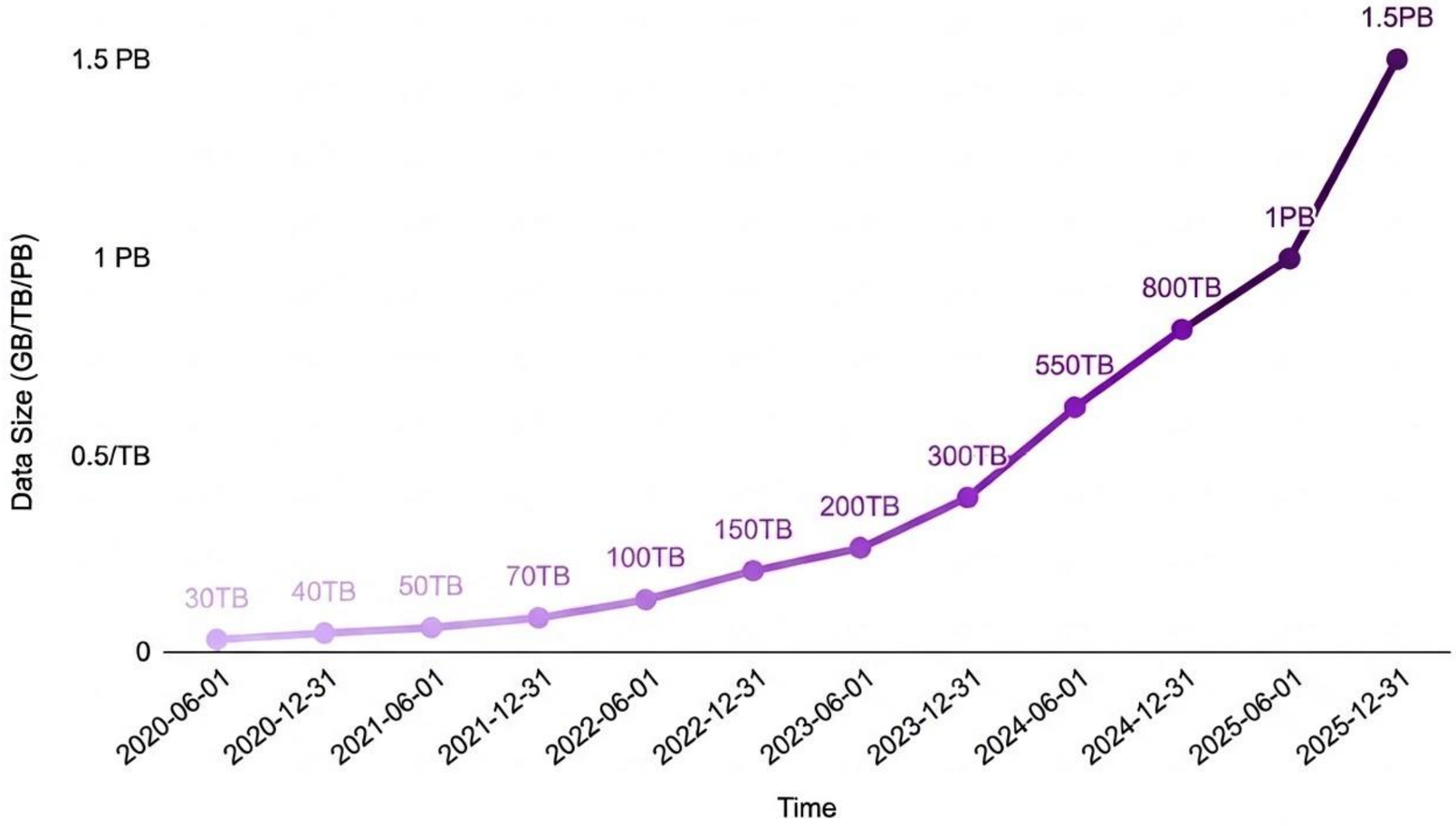
+5000 microservices

For our entire operation.

US$0.80

Cost per active customer.

With Many Customers Come Many Logs



From 50TB to more than 1.5PB in 5 years. 1900% total growth.


With this growth...
Why change the platform?

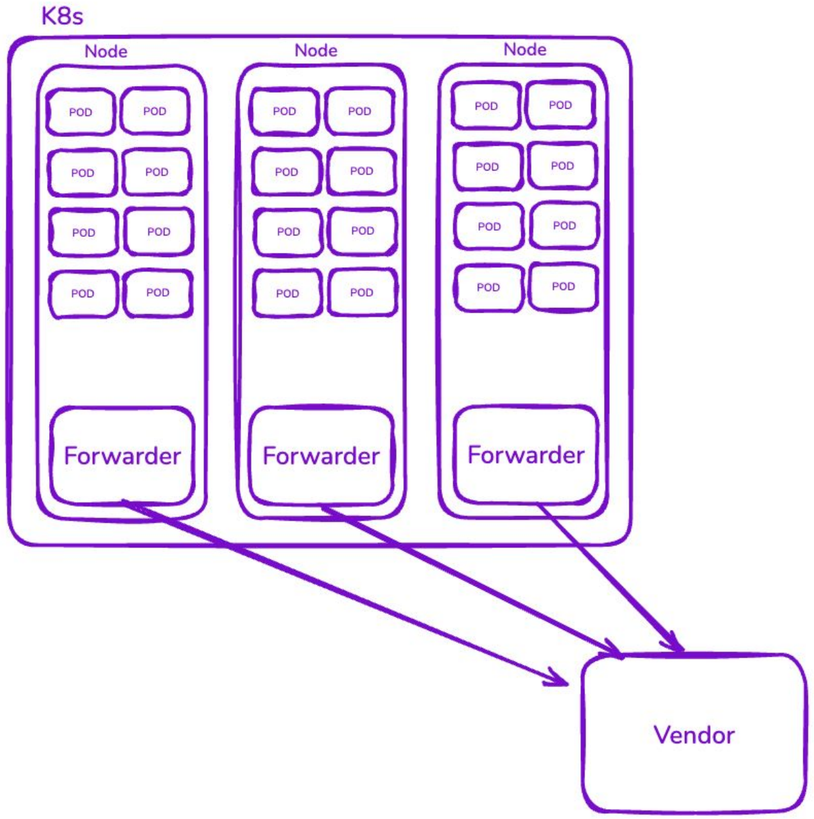
Why change?

Our external vendor started to have difficulties to handle our growth and scale at this time, together with unpredictable cost growth. Among the problems:

- No clear monitoring about their platform.
- Poor support.
- Hard cost model.
- Unpredictable cost growth.
- Losing telemetry data at peak times.
- High Vendor Lock-In

We needed a platform capable of supporting the hyper growth and future massive scale of Nubank reliably and in a cost-effective way, with clear, sustainable, and predictable cost-effectiveness while granting all necessary support and reliability to our engineers.

Vendor Lock-in Was a Major Challenge



Nubank developed a high dependency on the external vendor, at this time all our ingestion was handled by the vendor, on each node of our clusters, a forwarder of the vendor was responsible for collecting all logs and sending directly to its cloud.

A new vendor or build our own platform from scratch?

High Pricing

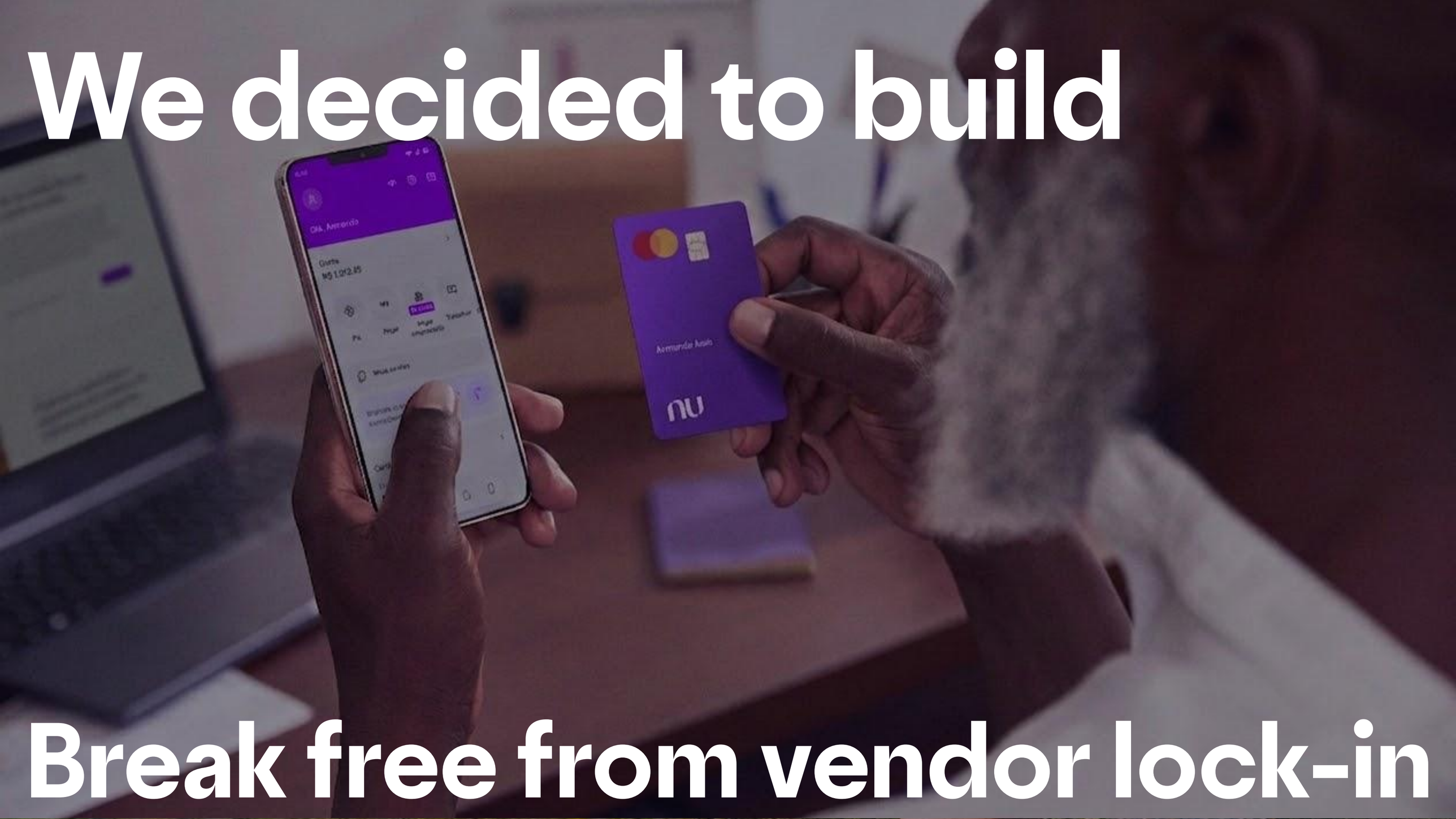
All vendors had a higher pricing than expected for our scale, aligned with long-term unpredictable pricing. Our long-term commitment to cost efficiency justified not moving forward.

Most of the problems looked the same

Other vendors had the same problems with unclear price models and support that didn't meet our expectations.

Next steps

We decided to build our own platform from scratch, focusing on having control over our data and costs, aligned with a focused developer experience for our context.


We decided to build
Break free from vendor lock-in

Platform Requirements

01

Scalable and Elastic

Nubank is in a constant growth, the final platform needs to be scalable on a long-term vision and also being able to handle any peak on usage.

02

Fault Tolerant

Every component of the platform should be fault tolerant to prevent a full outage or leaving our operations without support.

03

Transport and Store Data

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

04

Cost Efficient

The platform must be cost-effective over the long term, anticipating exponential growth in data ingestion. Furthermore, its operational cost needs to remain below vendor benchmarks.

Project Breakdown

We decided to divide this major project into two separate projects, based on the data flow. This approach allows us to continually deliver and validate our changes, avoiding a "big bang" migration away from the previous vendor.

1

Ingestion Platform

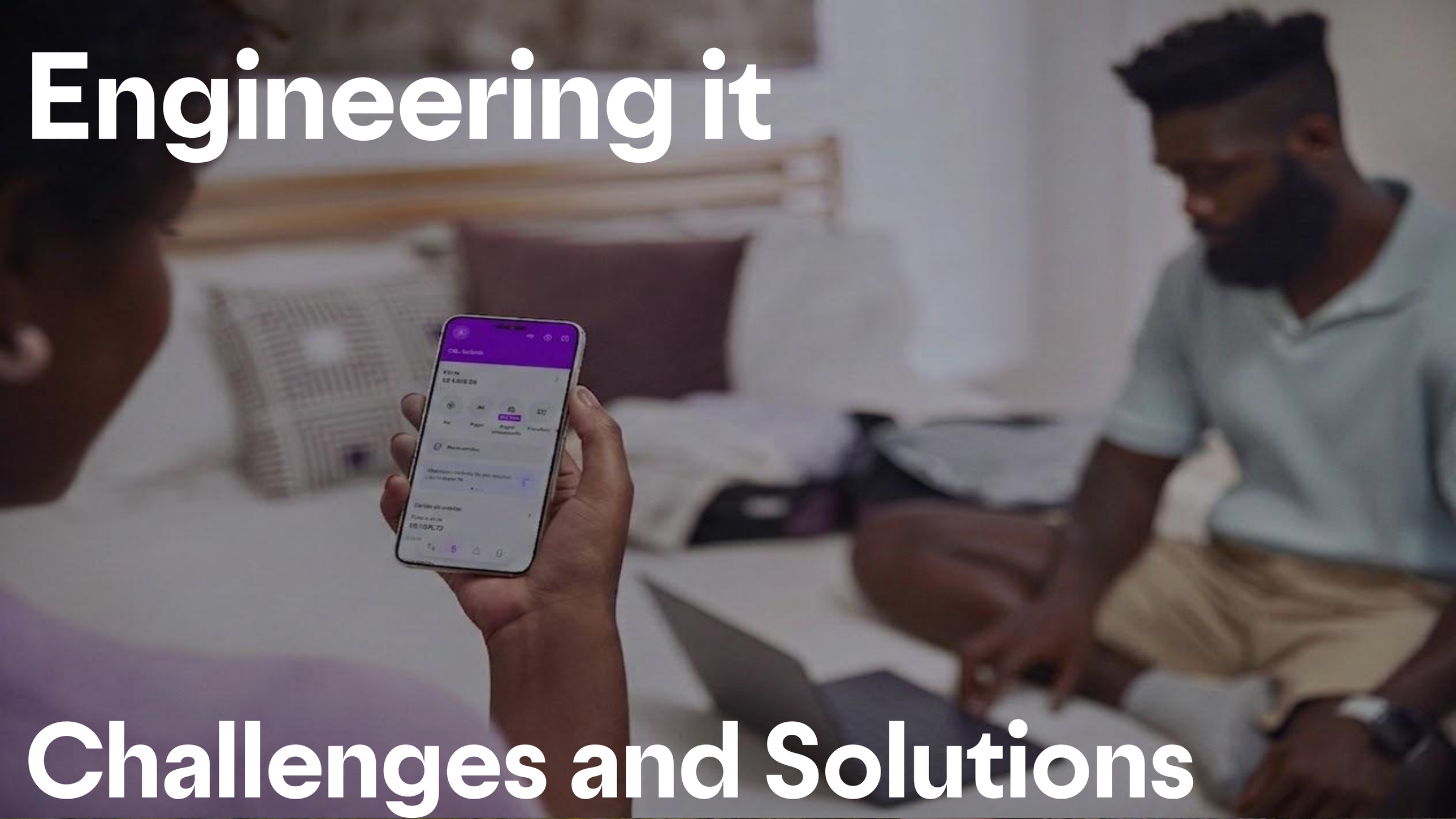
This platform would allow us to have control over our data ingestion, collecting logs, transforming them and sending to any storage platform, vendor agnostic.

2

Querying Platform

The central platform, responsible for store logs and query them.

Engineering it



Challenges and Solutions

Ingestion Platform

Challenges on Ingestion Platform

How to collect every log

Which application to use to collect every application log and send to the new platform.

How to transport so much data

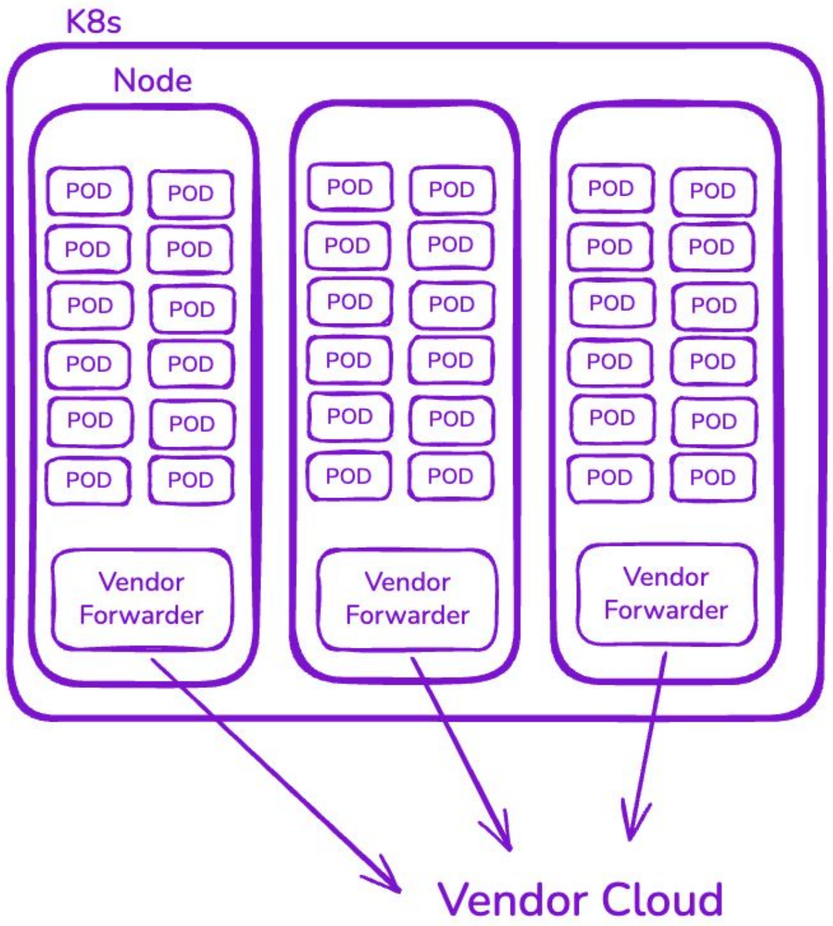
Transport petabytes of data in a cloud environment can be expensive, we need to do it in a smart way.

How to scale the platform

How to scale the platform efficiently to handle peak usage and long-term growth.

First challenge: Collect Every Log

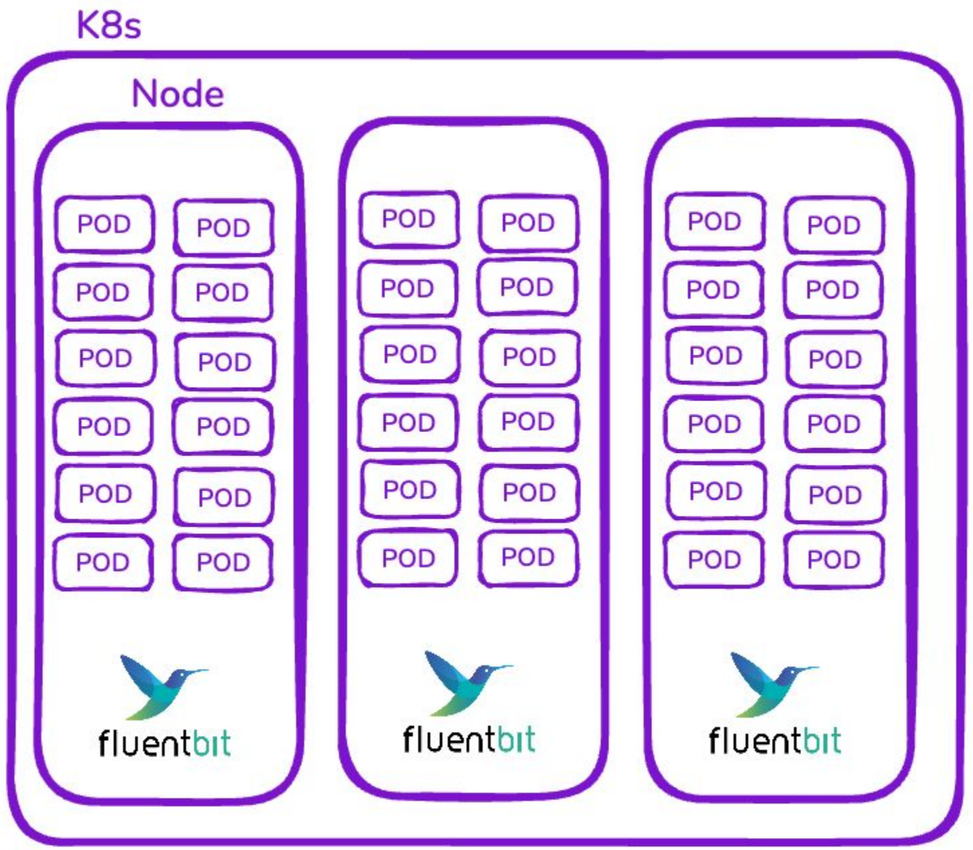
First challenge: Collect Every Log



For many years, Nubank relied on a vendor cloud for log storage. At that time, the only infrastructure we had for log shipping was a vendor forwarder deployed on each Kubernetes node, sending logs directly to the vendor's cloud.

First challenge: Collect Every Log

To replace the old vendor forwarder, we decided to use Fluent Bit to collect logs from all our applications running on Kubernetes.



Why Fluent Bit

fluentbit

Simple

Very simple setup, with an amazing set of plugins.

Lightweight

Fluent Bit is lightweight and runs with few resources, perfect for daemonsets.

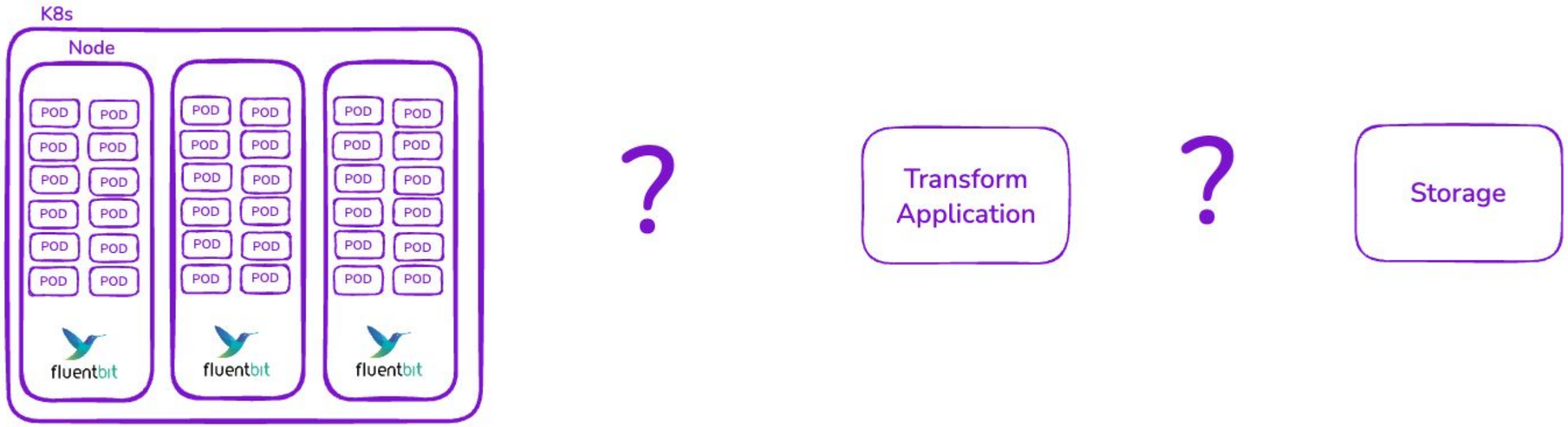
CNCF

Cloud-native, vendor-neutral, widely adopted.

Second challenge: Transport so much data

Second challenge: Transport so much data

All data was generated in a standardized format and collected by Fluent Bit. Given our goal to filter and collect metrics from these logs, the core challenge became: **How do we reliably and cost-effectively transport such a massive volume of data?**



Second challenge: Transport so much data

To address this challenge, Nubank collaborated with our cloud provider, AWS, to analyze all available options for efficiently and effectively transporting this data.

The requirements at this time were: Be able to handle more than 100GB/s and trillions of lines of log each day.

Two options were considered: Batch Processing and Stream processing.

Second challenge: Transport so much data

Batch Processing

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

Stream processing

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

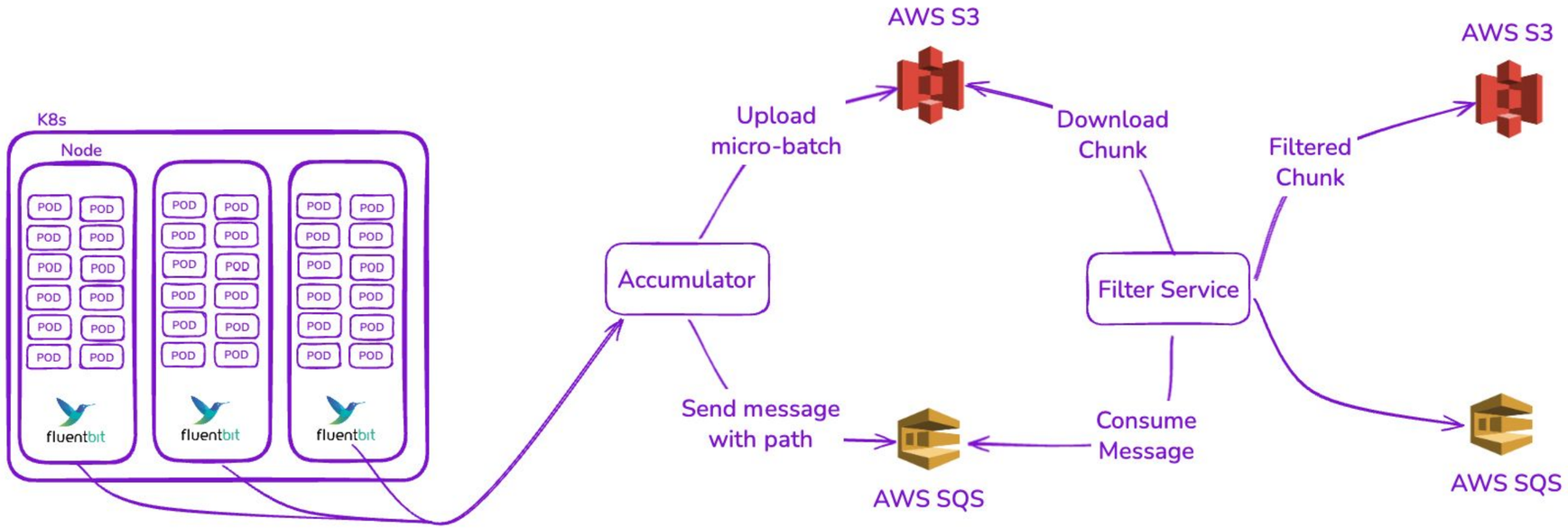
The Solution

The solution was to rely on a 1-2 minutes window to do the ingestion and apply a micro-batching architecture, to still have up-to-date data being ingested while being cost effective.

Second challenge: Transport so much data

The developed transport protocol has on its core two AWS Services: S3 and SQS. Relying on this services, we have a very cheap and efficient way of transporting data on AWS.

- Each SQS message contains the address of a micro-batch uploaded on S3 as GZIP.
- We accumulate micro-batches of megabytes of logs.
- All services from our platform use this protocol to communicate with each other.



Protocol Results

Over the last years with this platform, we were able to achieve great results relying on this protocol to communicate.

Cost Efficient

With this protocol, we achieved a cost of *$0.000159* per GB transported between systems on US-EAST-1. This was more efficient than any tool/protocol benchmarked by our team that has our requirements.

Reliability

This protocol and set of technologies prove to be extremely reliable, in these years in production the cost of maintaining and operating it was near zero, while also providing no critical incident during this period.

Failproof

With the usage of queues, the platform can easily recover from any problem without data loss.

Third challenge: Scale the platform

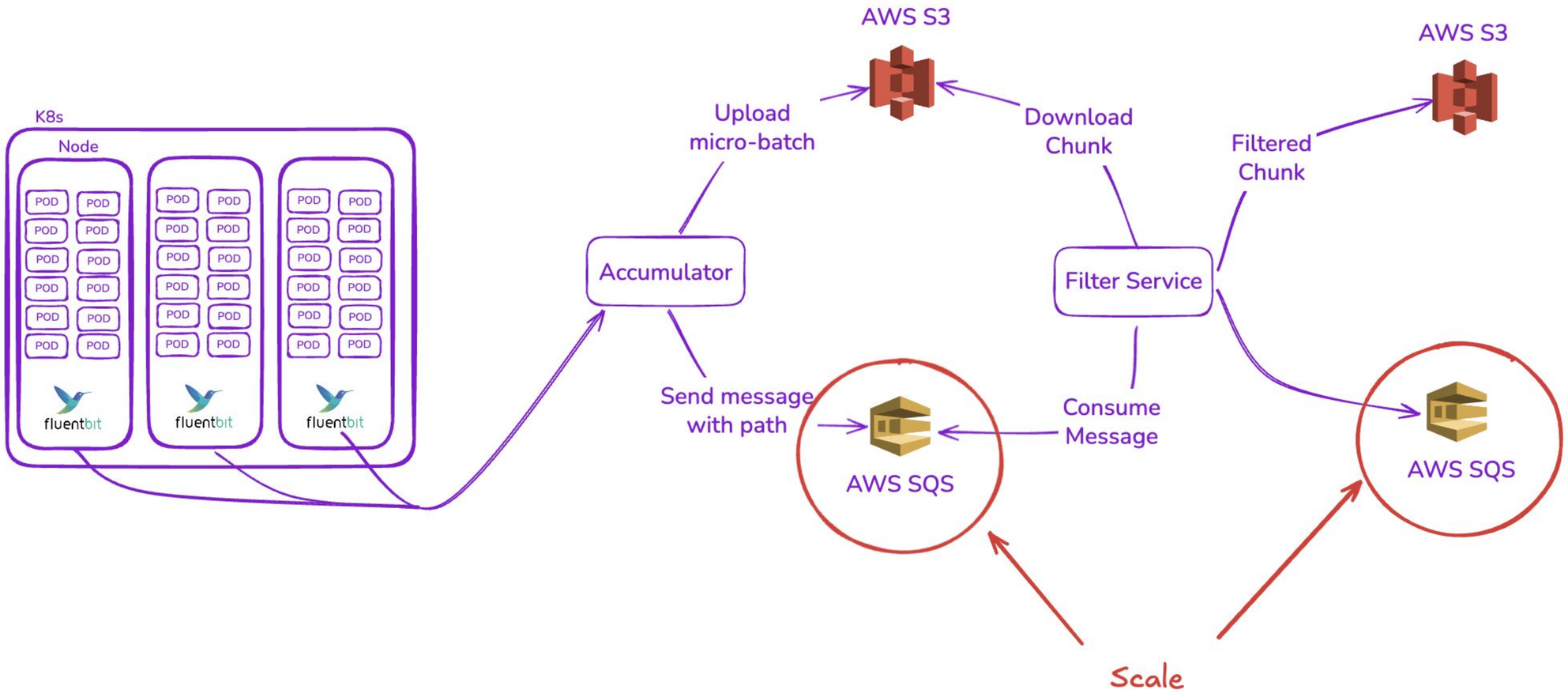
Third challenge: Scale the Platform

With everything in place for the ingestion platform, the last requirement was to find a way to scale the platform quickly and effectively when needed.

As being a Bank, we receive a spike in usage during specific events, like Black Friday, or specific hours during the day like the lunch hours. We wanted to have an automated way to scale the platform as much as needed based on how many logs are being generated.

Third challenge: Scale the Platform

The idea was simple: Scale every application based on the number of messages in the queues, if there are many messages being generated, there are many data being consumed, therefore, more applications to consume it are needed.



Third challenge: Scale the Platform

To handle this task, we used Keda, to do autoscaling based on SQS events, the tool was an excellent choice, and today is one of the core applications to our platform.

Why Keda

Event-driven Autoscaling

Scale workloads dynamically based on real-time SQS demand, improving responsiveness while optimizing resource consumption.

Native Kubernetes Integration

Seamlessly integrates with Kubernetes autoscaling primitives, simplifying operations and platform management.

CNCF

Backed by the CNCF ecosystem, ensuring strong community support, reliability, and long-term sustainability.

Storage & Querying Platform

Challenges on Storage & Querying Platform

Where to store all logs

Where can we store petabytes of logs in a reliable way?

How to store logs

How can we store petabytes of logs in a cost effective way?

How to search for it

How can we search on this amount of information reliable and cost-effective?

User Experience

How to provide the best experience to our customers?

The first challenge: Where to store all logs?

First challenge: Store Logs

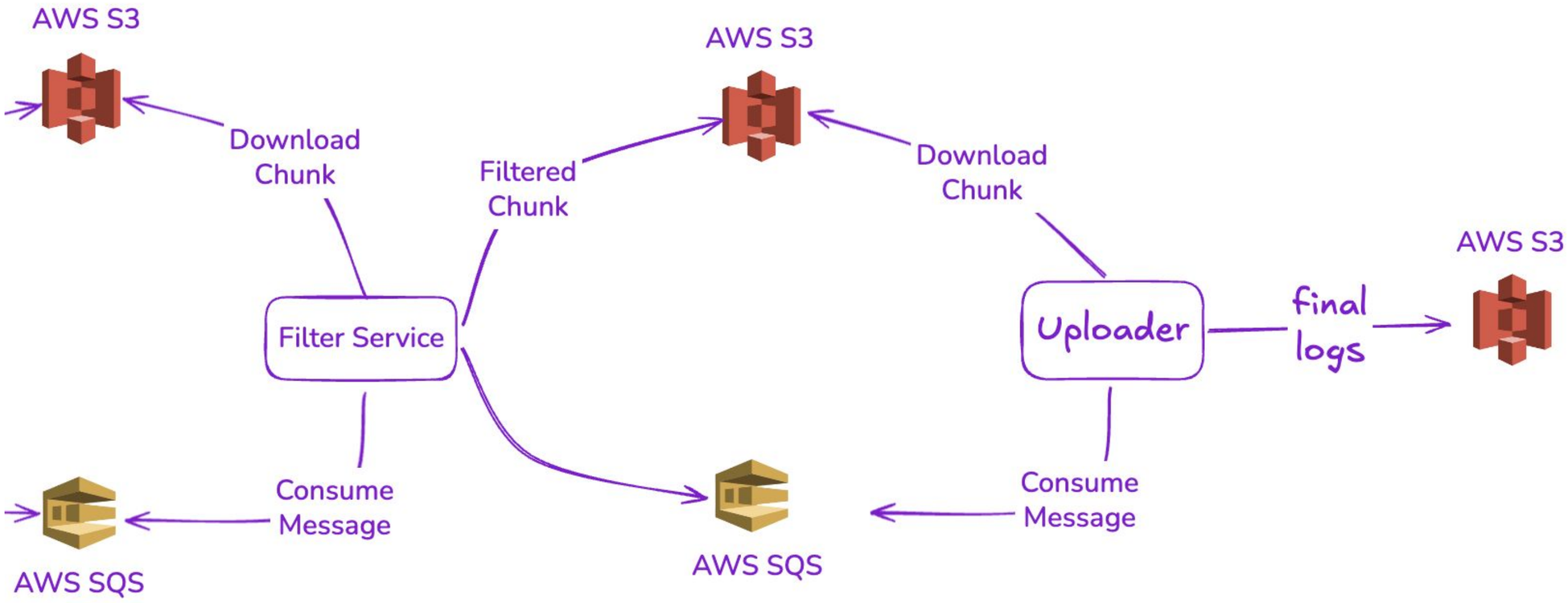
Given the immense volume, we required a storage solution capable of reliably and cost-effectively storing petabytes of data for the long term.

We chose to continue utilizing AWS S3. However, instead of solely using it for data transport, we transitioned to using an S3 bucket as the final destination for our logs, where they can also be queried.

S3 provides us with secure storage, competitive pricing, and zero operational overhead.

First challenge: Store Logs

The team started to send the logs to the final S3 bucket after the filter & processing application.



S3 Usage Results

Massive Scale

S3 is handling more than 800k uploads per second of files across all regions.

Zero Operation

Trusting our logs to S3 generated zero operation overhead in the team over the last few years, with S3 being able to handle every peak on usage.

Cost Efficient

S3 proved to be an excellent cost efficient way to store our logs over our 45 days of retention.

Second challenge: How to store logs?

Second challenge: How to store logs?

With the final decision made (To store our logs on S3), the next decision was about how to store all this data in S3?

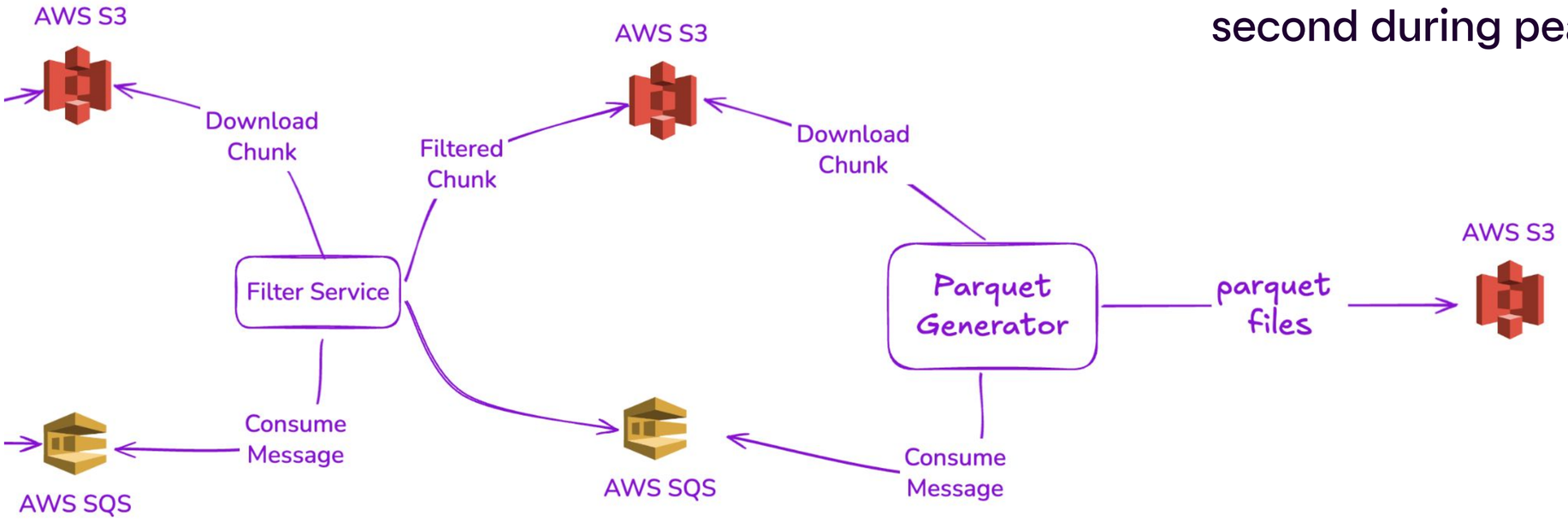
Many options were evaluated, keep using GZIP as our communication protocol uses would be simpler, but the team found Parquet files being the ideal solution.

- Apache Parquet is an open format, without any vendor lock-in and many tools support it.
- Columnar storage for faster queries.
- >90% compression rate for efficient storage.

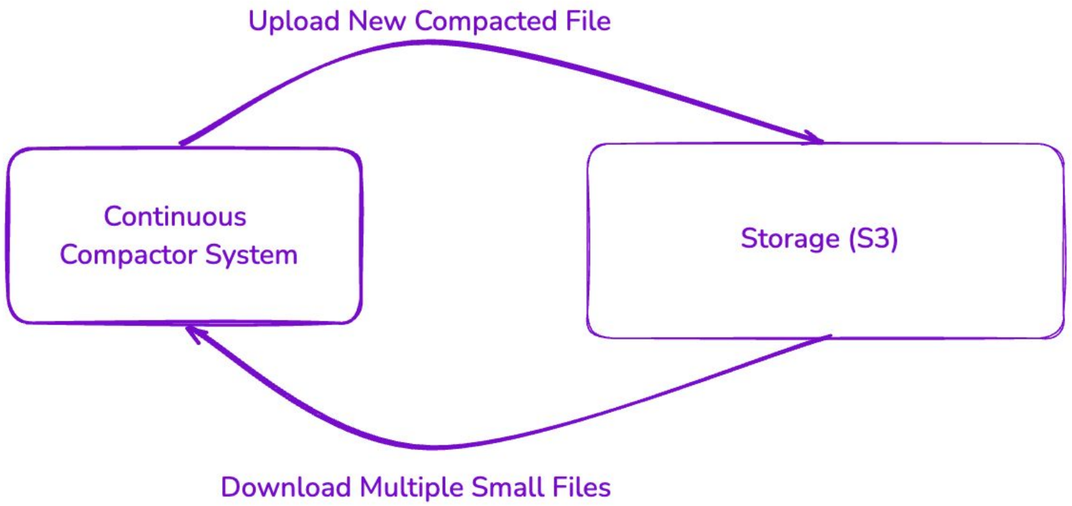
Second challenge: How to store logs?

The team created an application to generate parquet files based on our standard log format, and placed at the end of the ingestion stream, after the filter service. This service downloads filtered chunks from S3, converts them to parquet and uploads in the final bucket to be queried.

- This service creates over 800k files per second during peak times.



Second challenge: How to store logs?

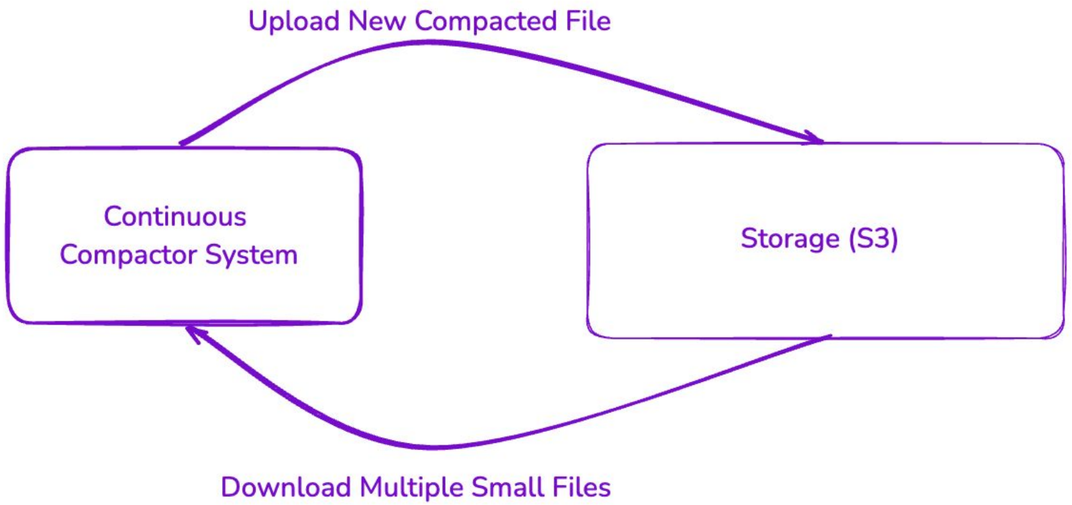


The combination of high volume of data being ingested with micro-batching made the parquet generator create a high number of small parquet files at the end of the day. **Over half a million files were being created per second.**

Since parquet files are more efficient when they contain more data, larger files would improve compression and overall cost efficiency.

To address this, the team developed an internal continuous compaction system, in addition to the parquet creator service, which merges thousands of small parquet files into larger ones, significantly enhancing storage efficiency.

Second challenge: How to store logs?



After the continuous compaction process runs at the end of each day, this system achieves an average compaction rate of approximately 95% using Parquet files.

For example, a service producing 1TB of data daily is compressed down to 50GB by day's end. This is accomplished without impacting users, significantly improving the platform's cost efficiency.

Third challenge: How to search it?

Third challenge: Query Logs

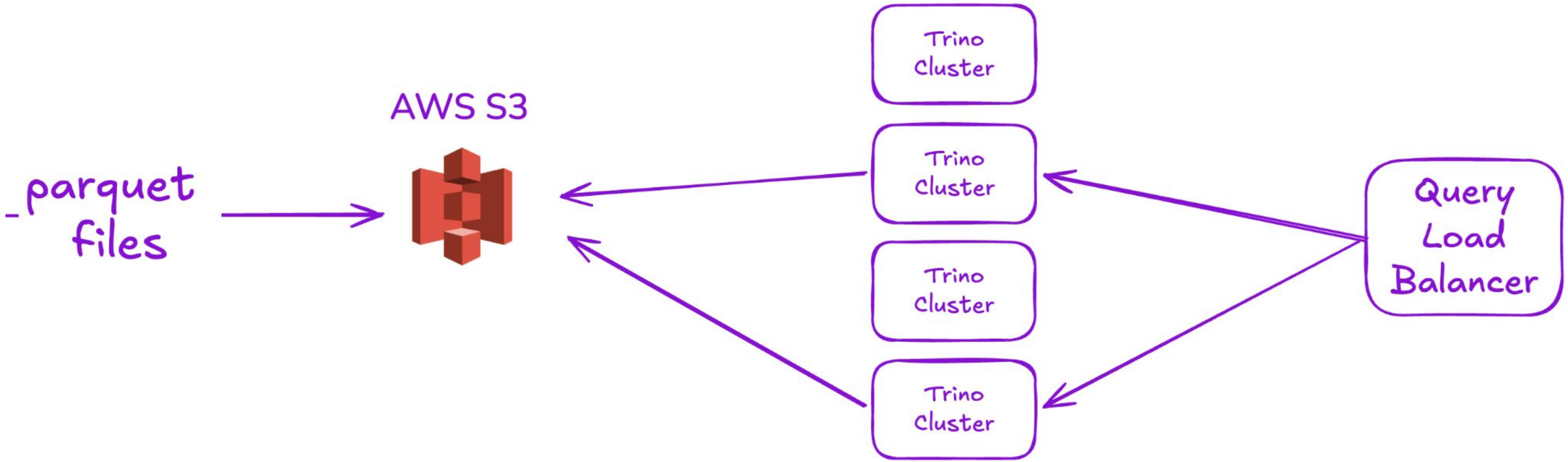
With every log already stored in a storage (AWS S3) in a known format (Apache Parquet), the last piece needed to have our platform was to choose or build a tool to query it.

For this job, **we chose Trino**. Trino is an open-source distributed SQL query engine, capable of searching on different sources.

- Open-Source
- Offers native AWS S3 Connection
- Support for querying on Parquets
- Distributed Deployment
- Cost Efficient
- Decoupled Ingestion
- Partitioning for fast queries

Third challenge: Query Logs

The team placed a cluster of trino clusters to query parquets on S3, and in front of it, developed an internal load balancer to distribute queries among clusters based on pre-defined rules.



User Experience

Fourth challenge: User Experience

With everything working, the last challenge was related to the user experience of our platform, how could we provide the best experience to our users?

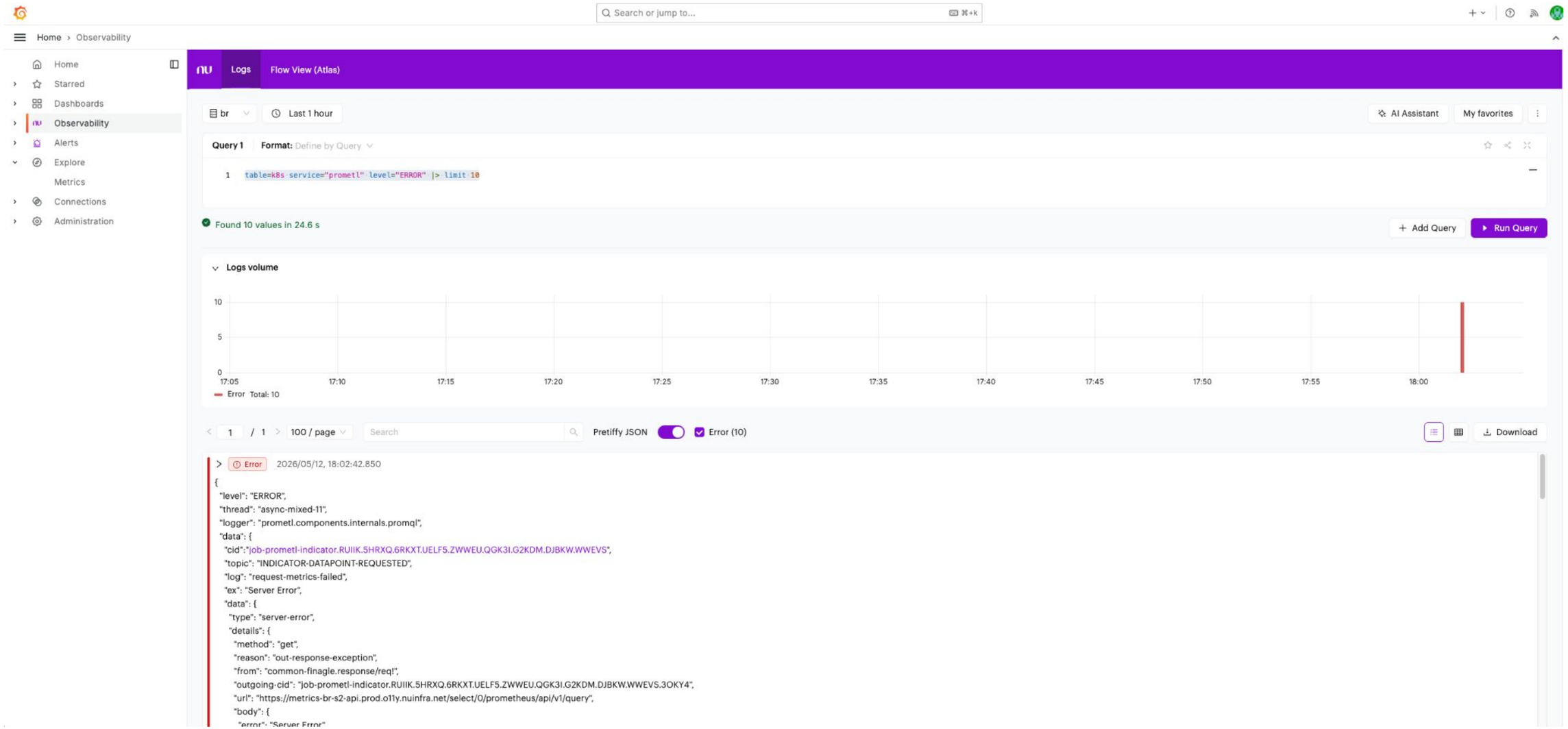
- Trino has an API, but we needed a way to simplify how our users query the data.
- Everybody knows SQL, but it was too verbose to search for logs.

Fourth challenge: User Experience

To do it, we decided to build our UI on Grafana, the company was already using it for our metrics platform, and the team had the objective of centralizing the entire observability platform on just one tool.

We developed our own Grafana custom app plugin, with everything built focusing on our engineers experience and special internal flows.

Fourth challenge: User Experience



Fourth challenge: User Experience

In addition to the app, the team saw an opportunity to replace Trino SQL with a custom Domain-Specific Language (DSL) for querying logs. This new DSL is less verbose, and most users preferred not having to use SQL for log queries.

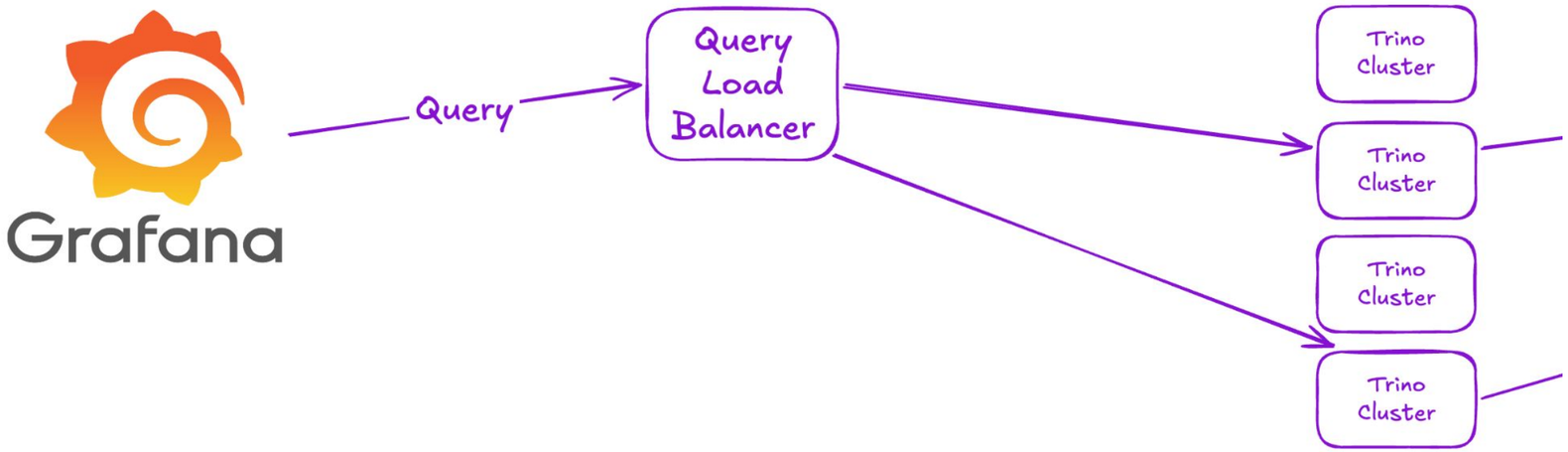
```
SELECT
 *
FROM
 nu.logs.k8s
WHERE
 $__timeRange()
 AND service = 'example'
 AND prototype = 'global'
ORDER BY
 time DESC
LIMIT
 100
```



```
table=k8s service="example" prototype="global"
|> limit 100
```

Fourth challenge: User Experience

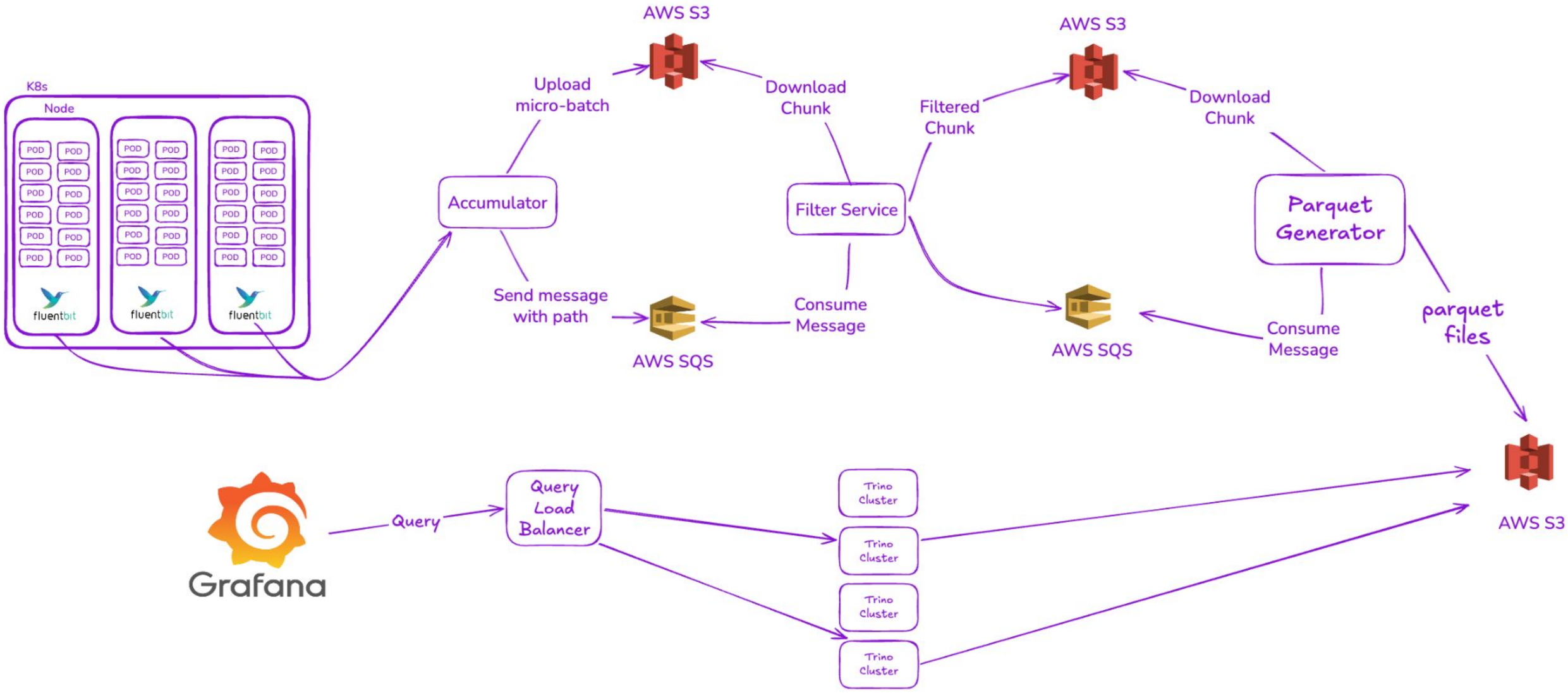
The Grafana app sends queries directly to our back-end, being the last piece of our architecture.



Final Architecture

(Simplified)

Final Architecture (Simplified)



Results

Project Results

1.5 Trillion

Lines processed every day.

45 GB/s

Ingested during daily peak times.

25TB

Of logs filtered every day.

1.5 PB

Logs ingested every day.

26 000

Searches every day.

100PB

Of data searched every day.

50% Cost Reduction

Compared to benchmarked vendors.

5 Engineers

Working in the project.

Thank you.

Reliability

Reliability