Observability and Performance 

**PREPARED FOR**

Erik van der Schriek

Niko Kuijpers

Gertjan Schouten

**PREPARED BY**

Maarten Blömer

Nick Krijgsman

Faruk Aydin

Jursley Gonzalez

Vincent Andersen

# Table of Contents

[**Table of Contents**](#_z6lhuq9emniv) **1**

[**Case Study**](#_k6jx82pctwds) **2**

[Case goal:](#_bpxkra1gvikb) 2

[Case questions:](#_3vn6lbsuph60) 2

[Brainstorming:](#_llbf4q7qnu37) 3

[**Netflix study**](#_zc6h8lveooxw) **4**

[Scaling (Faruk)](#_eaou76j8xww7) 4

[Subject (Mäártén)](#_97ho52d59gzk) 6

[Monitoring (Vincent)](#_7tixsx57ubo2) 7

[Usage](#_wtxdqrbmei3e) 7

[Netflix Telltale](#_ctpswv4lfgdl) 8

[Netflix Intelligent Alerting](#_8hd6mg1djk9x) 9

[Netflix Deployment monitoring](#_nx79vmfx9k05) 9

[Tracing (Nick)](#_n6r32poarpp4) 10

[Trace instrumentation](#_k7yqy3ixb2lg) 10

[Stream processing](#_mrttp4mr1loo) 11

[Storage](#_d235hbh8hjd3) 11

[Observability (Jursley)](#_505gm0evad4t) 12

[Edgar](#_kfkx9d3d7c9b) 12

[Scaling log ingestion](#_y6iqc3psqu42) 13

[**Sources**](#_qwibiw7iqp2l) **14**

# Case Study

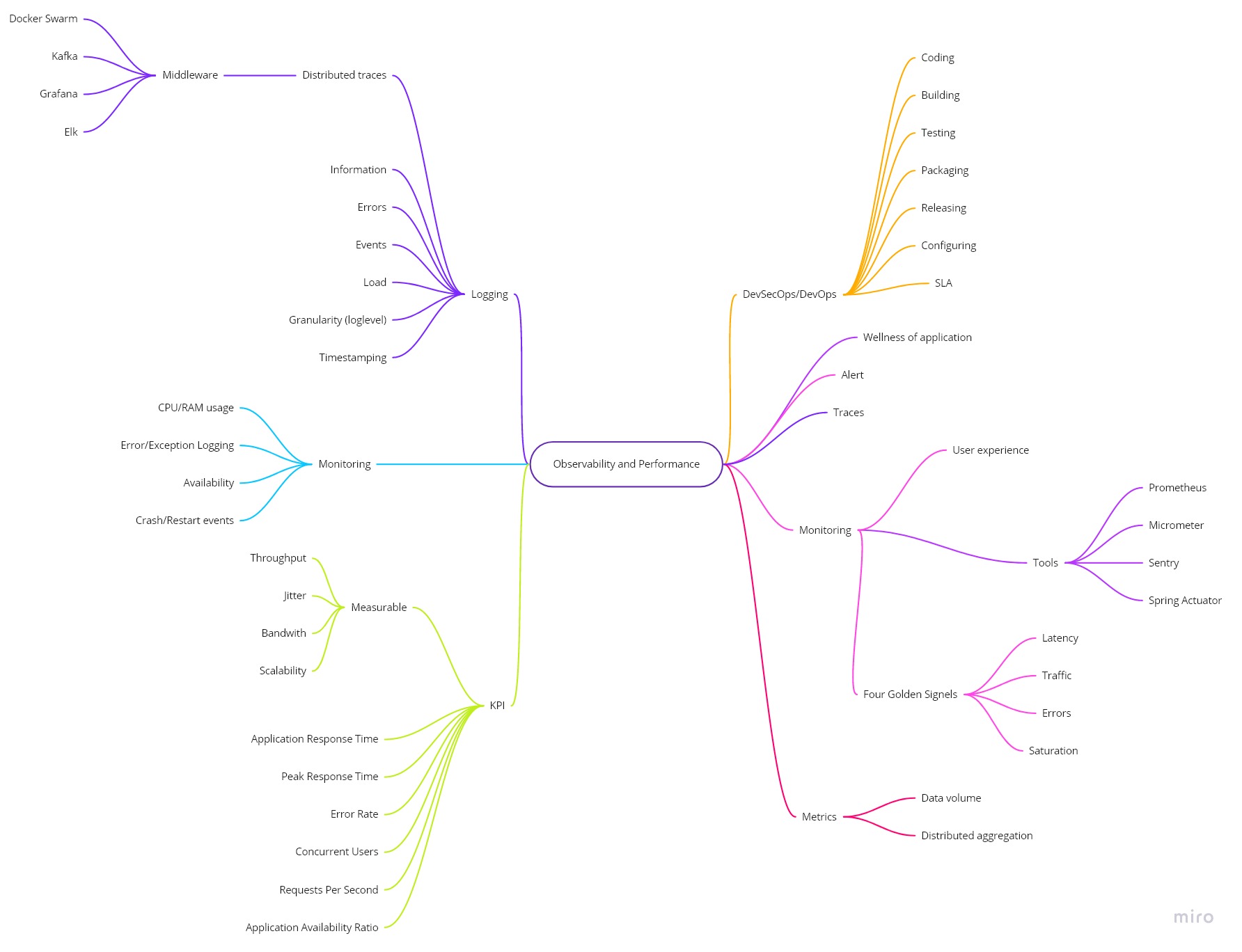
## Case goal:

Find out how Observability is implemented at big companies and what you can use for your own implementation within the group.

## Case questions:

* Find out what it is that Netflix does when it comes to observability.
* What are ways that Netflix implements logging? (Maarten)
* What are ways that Netflix implements monitoring? (Vincent)
* What are ways that Netflix handles scaling? (Faruk)
* What are ways that Netflix implements observability? (Jursley)
* What are ways that Netflix implements tracing? (Nick)

## Brainstorming:



# Netflix study

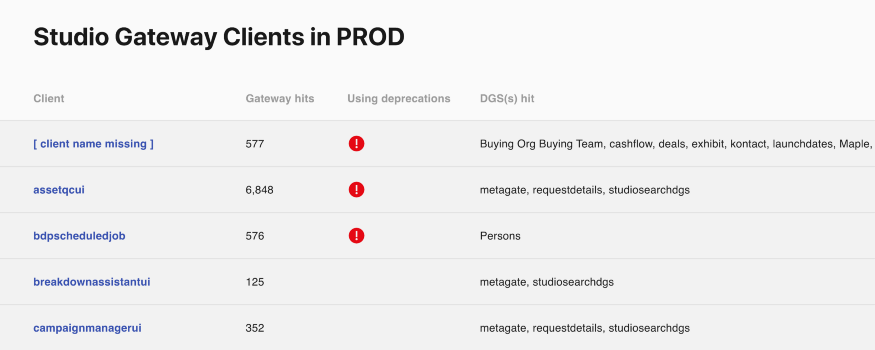
## Scaling (Faruk)

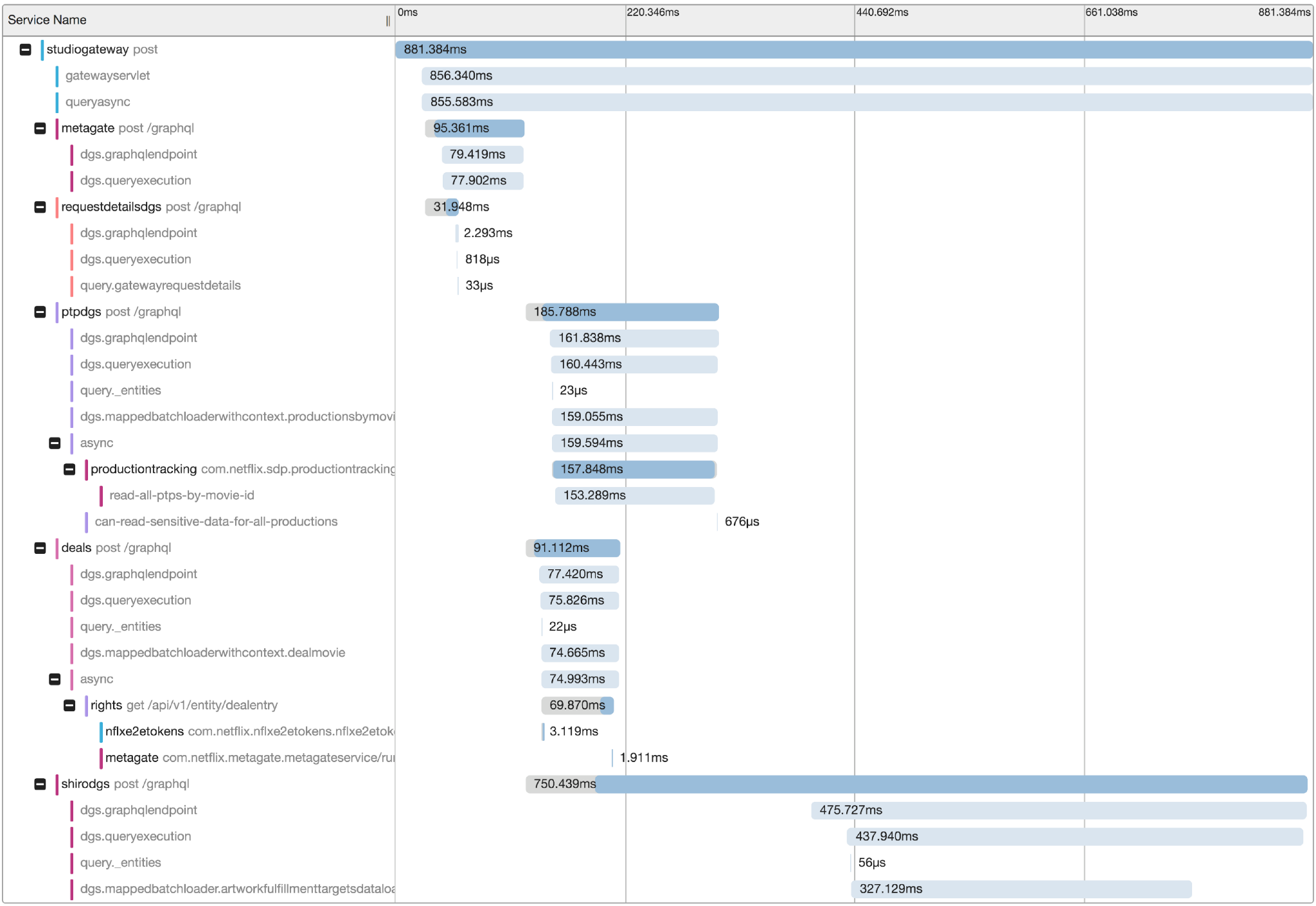
Netflix uses AWS for almost all of its computing and storage needs, including databases, analytics, recommendation engines, video transcoding, and more.

Netflix uses auto scaling. Many microservices are built to be horizontally scalable and use Amazon EC2 Auto Scaling to automatically add or remove EC2 instances when the workload changes. For example, when people return home from work and turn on Netflix, the services are automatically scaled up to meet this demand.

Netflix is ​​known for its loosely coupled and highly scalable microservice architecture. Independent services make it possible to evolve at different speeds and scale independently. Still, they add complexity for use cases spanning multiple services. Rather than exposing hundreds of microservices to UI developers, Netflix provides a unified API aggregation layer at the edge.

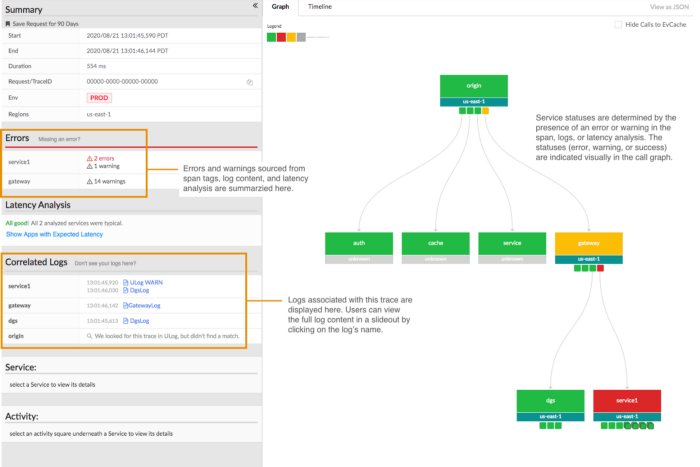
Observing with GraphQl





## Logging (Maarten)

Previously the netflix engineers had to go through a mountain of logs from all the different services in order to understand a single specific streaming failure. To reconstruct these errors it took way too long and involved tracing all the interactions between the netflix app and the content delivery network alongside the backend microservices.



## 

Having a main location where the logs are collected increases the amount of logs you have to read through to get to an error.

## 

## Monitoring (Vincent)

### Usage

Monitoring is a form of getting feedback out of the system’s health, performance and cost. By defining some measures and setting up some tools to log these measures and set up an alarm for when something is out of the norm. This way you can ensure the system is performing to the expected level.

The reason why monitoring is important is because it allows for detecting issues and troubles at earlier stages without them bleeding to a production level. This will support the help of changes and upgrades to a system as the whole environment is tracked.

Some monitoring metrics are:

* CPU/RAM usage per service
* Error/Exception logging
* Availability
* Crash/restart events
* Other technical KPIs

Usually some existing tools allow for a quick and easy way to get started with monitoring. Some examples are:

* Docker API
* Data Dog
* Prometheus
* Scout

### 

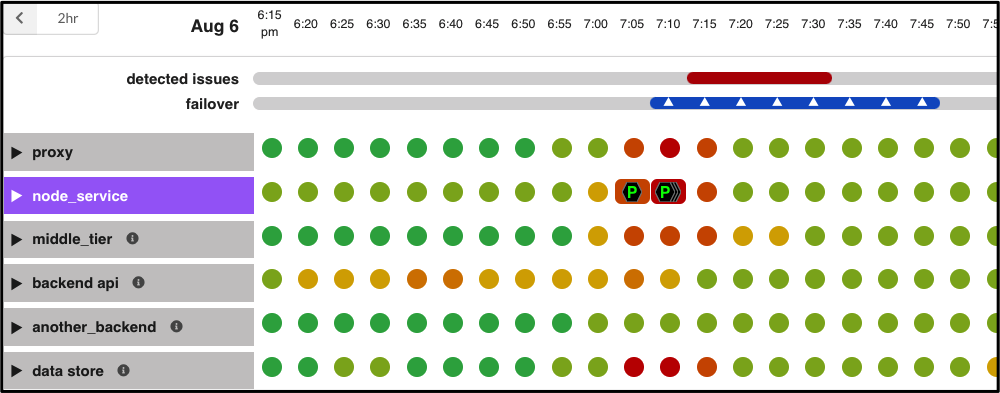
### 

### Netflix Telltale

In order to monitor their system Netflix built their own monitoring tool named Telltale. This application uses a variety of data sources to learn the typical health of an application. These sources include:

* Atlas time series metrics.
* Regional traffic evacuations.
* Mantis real-time streaming data.
* Infrastructure change events.
* Canary launches and deployments.
* The health of upstream and downstream services.
* Client metrics and QoE changes.
* Alerts triggered by our alerting platform.

These sources provide different kinds of signals that will also have different levels of importance depending on the application. They build a model that will represent that specific application’s health status.

Using this information they have created a GUI that quickly shows the applications’ health status along with only the relevant data from the application plus that of up-and downstream services.

### 

### Netflix Intelligent Alerting

Because of the difficulty of alert tuning, where one can easily set the thresholds for alerts too high or too low, they have taken that into consideration when building Telltale.

Telltale tracks dependencies between services to build a signal pack which will be represented in the health model and are made using multiple different algorithms and machine learning.

Because they monitor the services in this way, it allows them to build an intelligent alert system. Telltale creates an issue on detecting health problems in the application’s ecosystem and alerts the team associated with that application, containing a snapshot referencing the health problems. All this while only sending a single notification.

### 

### Netflix Deployment monitoring

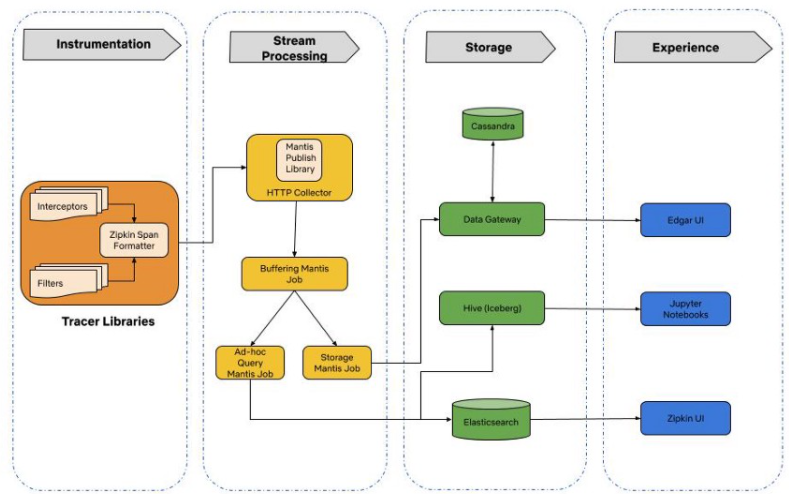
Netflix uses Spinnaker to roll out their builds which Telltate can use to continuously monitor the health of the instances running the new build. Continuous monitoring means a deployment stops and rolls back at the first sign of a problem. It means deployment problems have smaller blast radius and a shorter duration.

## 

## Tracing (Nick)

Netflix introduced their own troubleshooting tool Edgar. Before Edgar it was very engineer intensive to reproduce the streaming session and hope to find the users issue. The idea was to create a tracing tool that keeps track of streaming sessions, so it was easy to reconstruct the session failure by providing retry and error tags, service topology and latency measurements.

Edgar uses the open source project Open-Zipkin for its integration with Spring Boot based Java runtime environments. Netflix uses Mantis to process the stream of collected traces and uses Cassandra to store the traces. This makes up the three sections of the distributed tracing infrastructure: tracer library instrumentation, stream processing and storage.



### Trace instrumentation

An important question that arose among the engineers when asked about the trace instrumentation was if it is safe to integrate on the mission-critical streaming services. The integration injects infrastructure tags like service name and container instance identifiers. This way Edgar can query and join traces with log data for troubleshooting streaming sessions. Additionally, it became easy to provide deep links to different monitoring and deployment systems in Edgar due to consistent tagging.

### Stream processing

Netflix uses Mantis as their go-to platform for processing operational data. Which buffers and collects in a first job and in the second job Mantis taps the data feed of the first job and does tail sampling of data and writes it to storage. An additional advantage of Mantis is the ability to perform real time ad-hoc data exploration using Mantis Query Language.

### Storage

At first Netflix used Elasticsearch as their data store due to its flexible data model and querying capabilities. With more streaming services the data for tracing grew exponentially. Queries took longer as a result of this. Netflix solved this by migrating to Cassandra. To combat the costs of scaling the storage Netflix used the following optimization strategies:

* Use cheaper Elastic Block Store (EBS) instead of the SSD instance stores in EC2.
* Use a better compression to reduce trace data size.
* Store only interesting traces by using simple rules-based filters.

0

## Observability (Jursley)

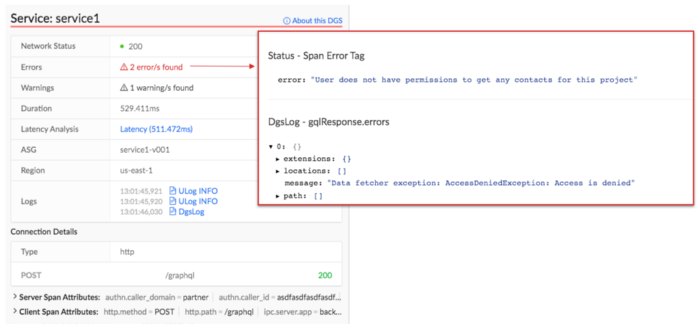
Netflix uses the following observability tools to deal with application issues.

### Edgar

Edgar is a self-service tool for troubleshooting distributed systems, developed by Netflix.

With request tracing and additional data from logs, events, metadata, and analysis, Edgar is able to show the flow of a request through their distributed system, what services were hit by a call, what information was passed from one service to the next, what happened inside that service, how long did it take, and what status was emitted — and highlight where an issue may have occurred.

Edgar adds a layer of context on top of tracing



### 

### Scaling log ingestion

It wasn't feasible for Netflix to keep storing device and server logs in persistent storage because of the large volume and cost. To deal with this problem Netflix reduced their storage retention time period and implemented a real-time stream processing platform called Mantis. With this they don’t have to save all logs to persistent storage, but could stream logs into memory and only save those that match a SQL-like criteria.

This provides the ability to quickly and iteratively identify the relevant subset of logs that should be stored.

# 

# Sources

|  |  |
| --- | --- |
| **Researcher** | **Source** |
| Vincent | <https://netflixtechblog.com/telltale-netflix-application-monitoring-simplified-5c08bfa780ba> |
| Nick | <https://netflixtechblog.com/building-netflixs-distributed-tracing-infrastructure-bb856c319304> |
| Maarten | <https://netflixtechblog.com/building-netflixs-distributed-tracing-infrastructure-bb856c319304> |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |