

O'REILLY®

# Software Architecture

ENGINEERING THE FUTURE OF SOFTWARE

## Reverse evaluating Netflix's architecture

Stefan Toth

@st\_toth; st@embarc.de

[softwarearchitecturecon.com](http://softwarearchitecturecon.com)  
#oreillysacon



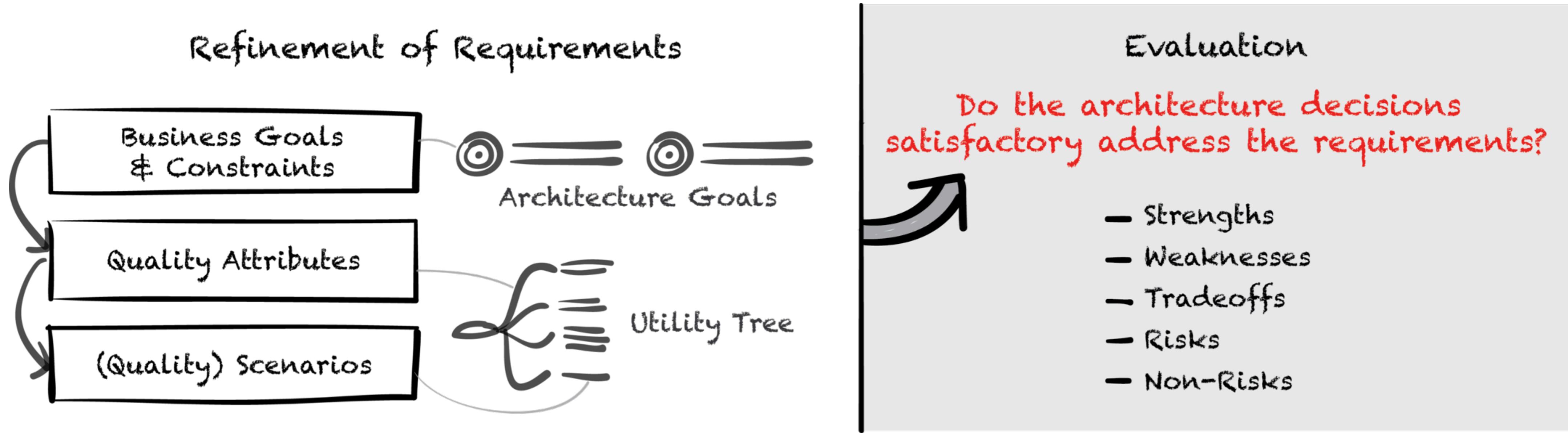
4

advanced



cool

# What is architecture evaluation?



## Methods:

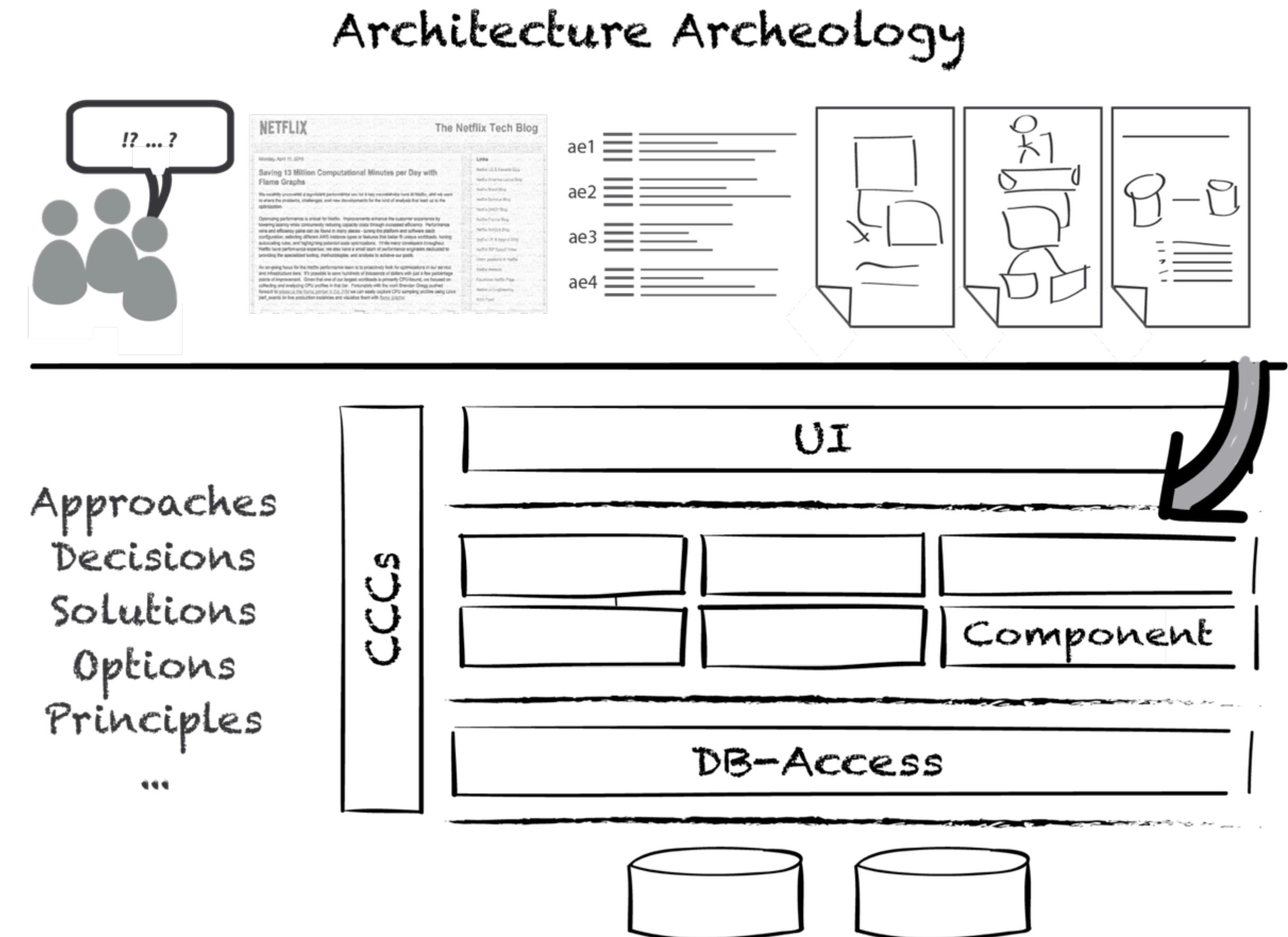
- **ATAM** – Architecture Tradeoff Analysis Method
- **CBAM** – Cost-Benefit Analysis Method
- **SACAM** – Software Architecture Comparison Analysis Method
- ...

# Reverse evaluating?

## Reverse Evaluation

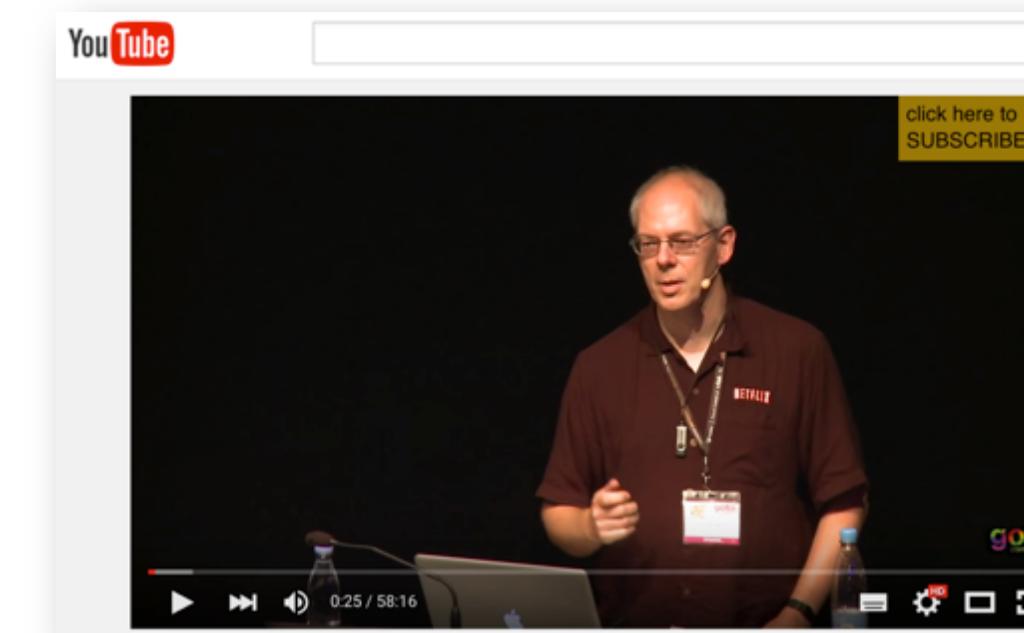
Which context makes the observed architecture concepts ideal?

- Intentional Tradeoffs
- (Quality) Scenarios
- Quality Attributes
- Business Goals
- Constraints / Absence of them
- ...



# Disclaimer

- This is a 45 minutes session, not a super detailed report (we actually have more material)
- I don't work for Netflix
- Netflix changes. The original evaluation took place in 2014
- All information in this presentation is based on what Netflix and people who worked there published
  - Blog-Posts: <http://techblog.netflix.com>
  - Github: <http://netflix.github.io>
  - Session Recordings (Conferences): Youtube, InfoQ etc.
  - Twitter
  - ...



The Netflix Tech Blog page for April 8, 2015, featuring an article about Vector. The page includes a sidebar with links to other blogs and a footer with an "About the Netflix Tech Blog" section.

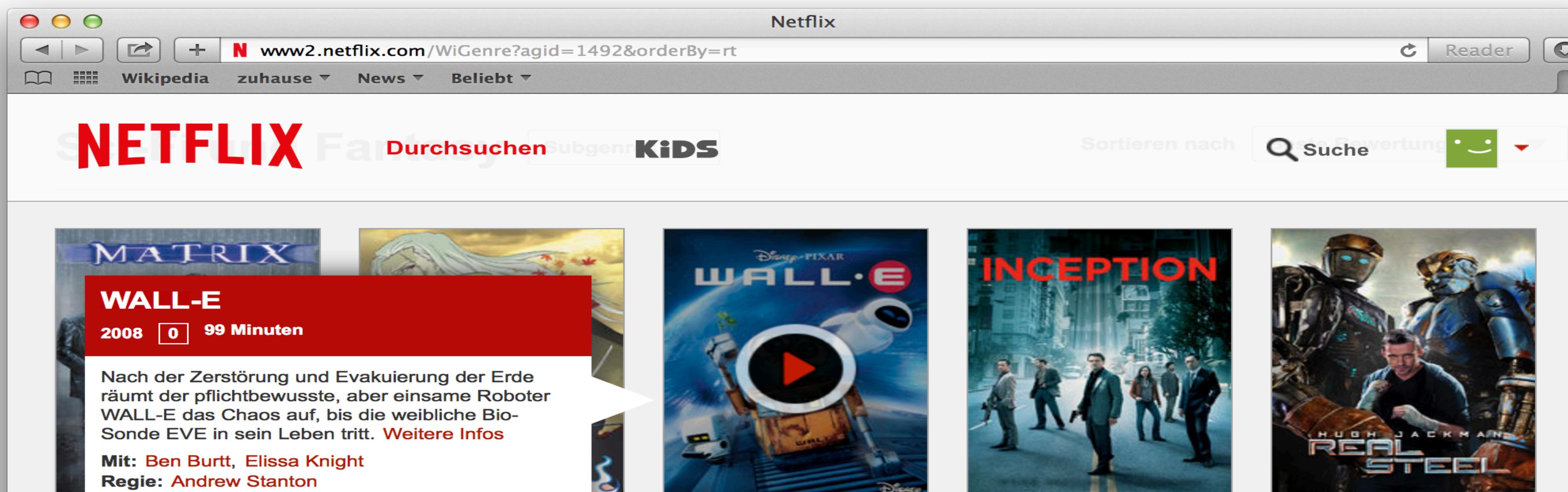
The Netflix OSS GitHub repository page, specifically for the SimianArmy project. It shows statistics like stars (2445), forks (298), and open issues (26). Other projects like Hystrix, Turbine, and Ice are also listed.

**NETFLIX**

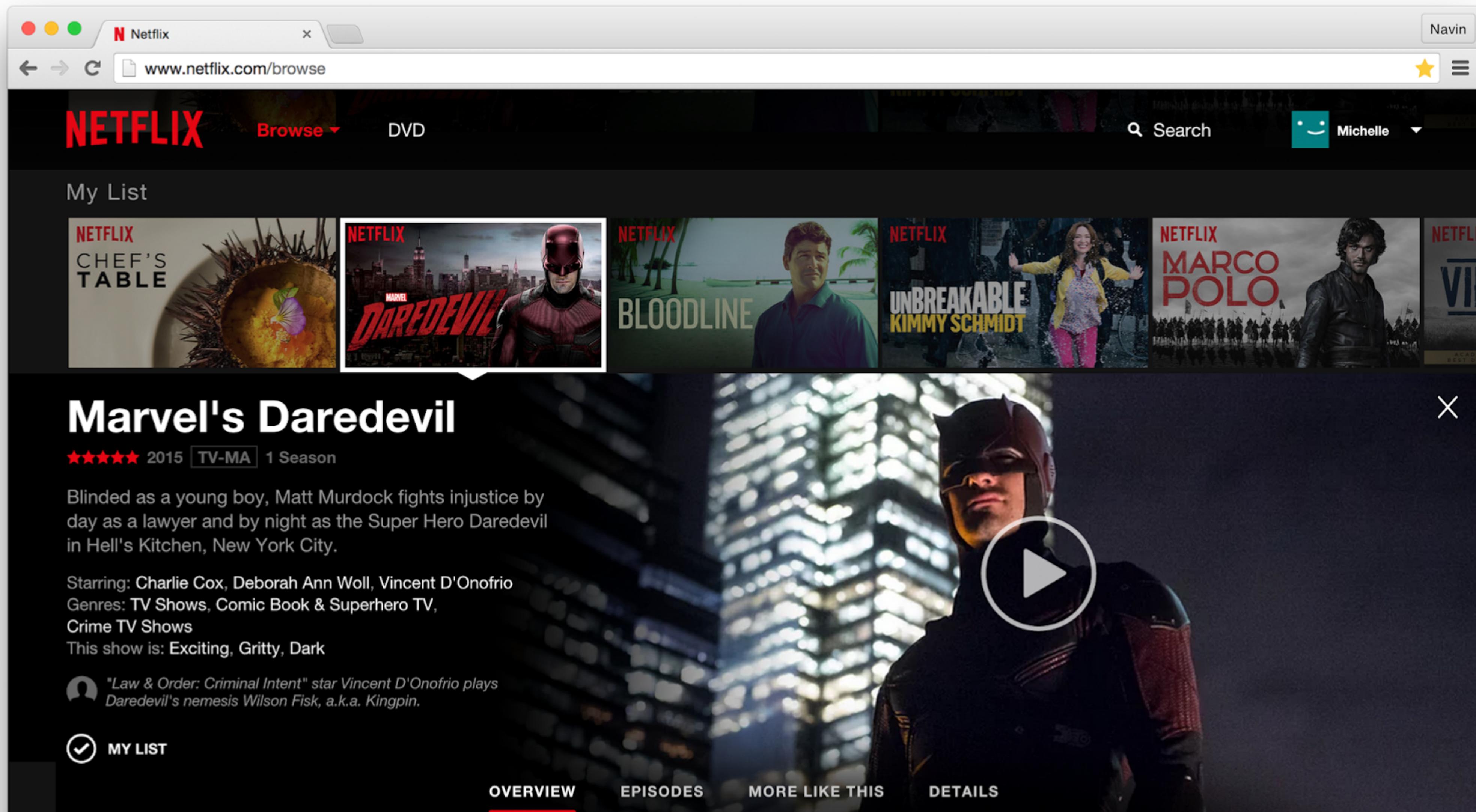
# Was ist Netflix?

*“Netflix is the world’s leading Internet television network with over 57 million members in nearly 50 countries enjoying more than two billion hours of TV shows and movies per month, including original series. For one low monthly price, Netflix members can watch as much as they want, anytime, anywhere, on nearly any Internet-connected screen. Members can play, pause and resume watching, all without commercials or commitments.”*

→ <http://ir.netflix.com>



‘Netflix is the king of online streaming, using more global bandwidth than cat videos and piracy combined.’



# Netflix – How big is ,big‘?



- **600+** Services (Applications)
- **Billions** of requests per day
- **> 2 Billion** hours of films and TV series
- **10.000s** of Ec2 Instances in multiple AWS Regions and Zones
- Cassandra DB in a multi-region, global ring with Terabytes of data
- At peak-times **1/3** of the USA‘s Internet-Bandwidth (Downstream)

# What are Microservices?

*“In short, the microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API.”*



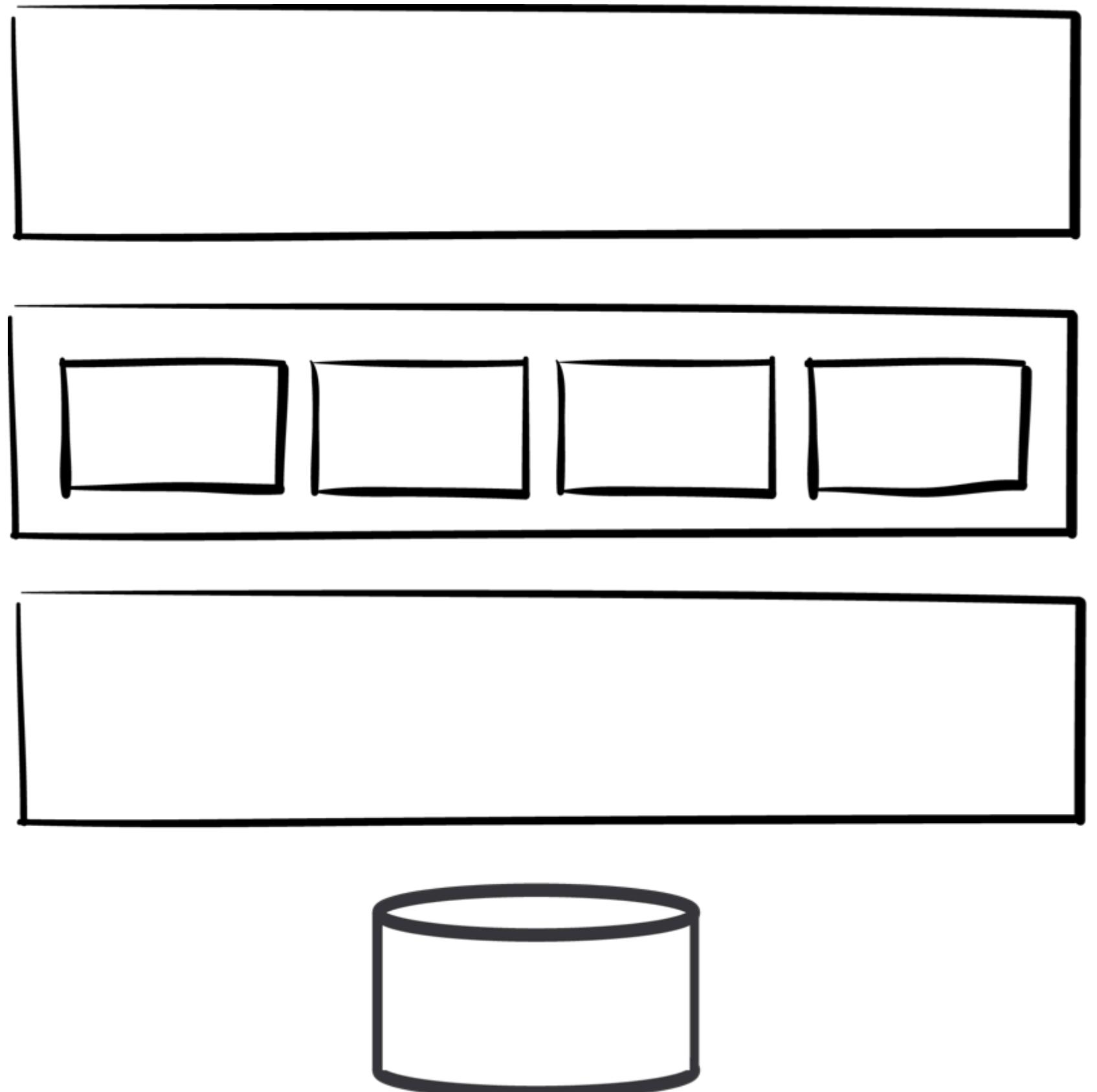
(James Lewis, Martin Fowler)

## Common Characteristics

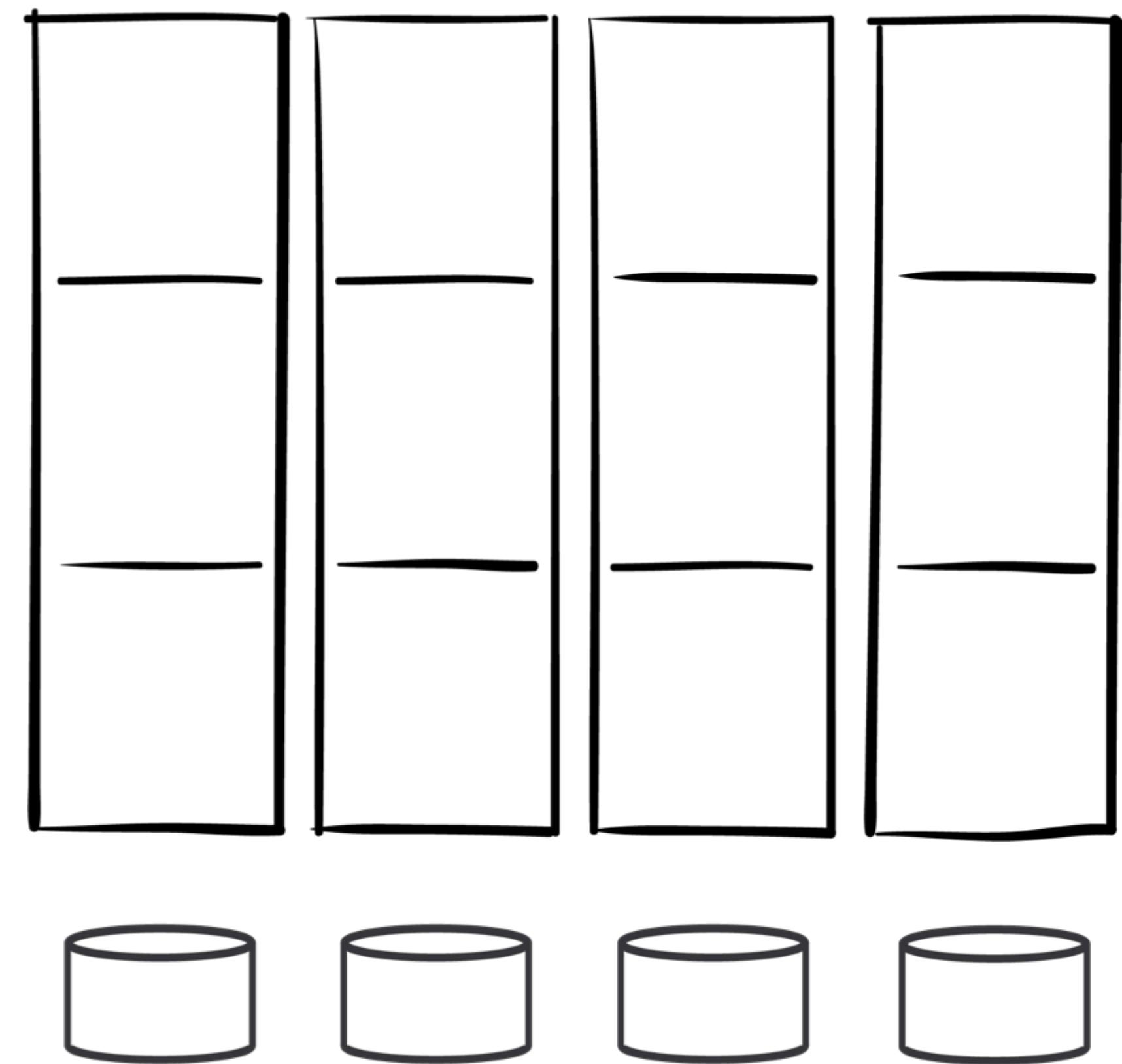
- suite of small services
- running in its own process
- built around business capabilities
- independently deployable
- bare minimum of centralized management (languages, platforms, technologies)

In Principle...

# Layers



# Slices / Verticals



# What 600+ Microservices feel like



# Many Services...

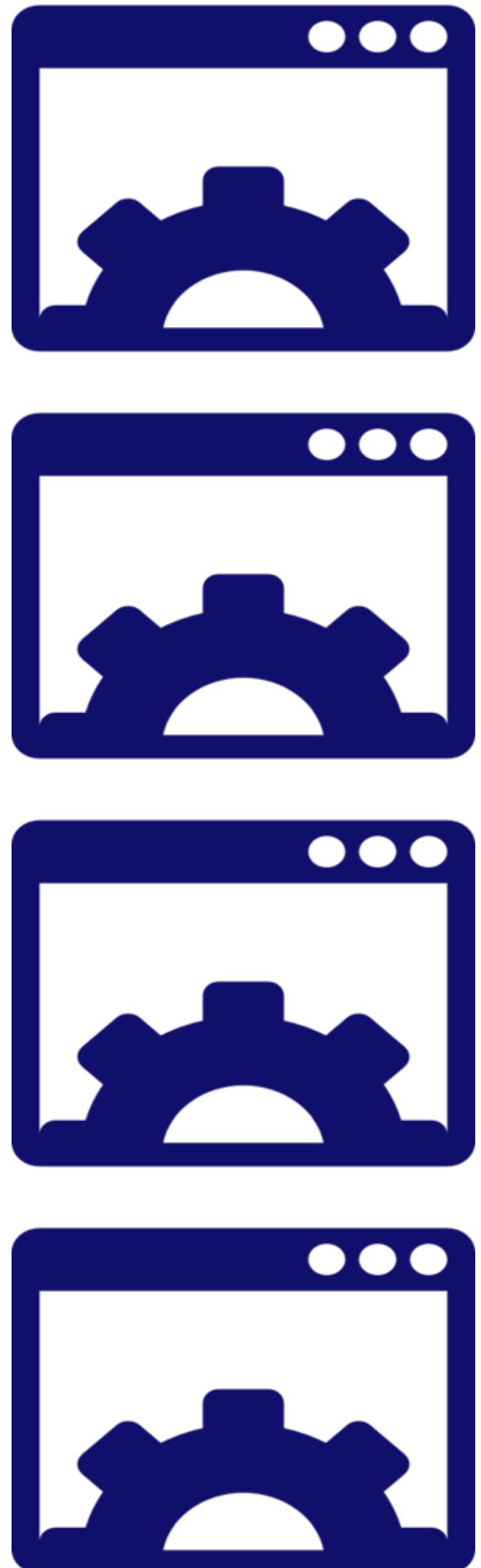
- **Maintainability:** Each part is small enough to be understood and changed relatively easy
- **Time-to-market:** Not hard to add new functionality
- **Scalability:** Good horizontal scalability (independently)



- **Complexity:** High operational complexity
- **Testability:** Hard to reproduce in test environments
- **Observability:** Not easy to get an overview or (system) status
- **Reliability:** Failure is not a possibility but a given



# Microservices at Netflix



## Functionality and technical Base („Platform“)

- Netflix is composed of 600+ services
- Examples for Services („Applications“)
  - Search
  - Recommendation
  - Similar Movies
  - Subscriber
  - Review
  - Video History
  - ...

# Services at work

NETFLIX

Browse Taste Profile KIDS DVDs

[Bell icon]

**People**

- Brad Pitt
- Bruce Willis
- Bruce Lee
- Bradley Cooper
- Brendan Fraser
- Bridget Jones
- Bryan Cranston
- Betsy Brandt
- Alison Brie
- Dee Bradley Baker
- Bryn McAuley
- Britt McKillip
- Brenda Grate
- Bradley James
- Thomas Brodie-Sangster

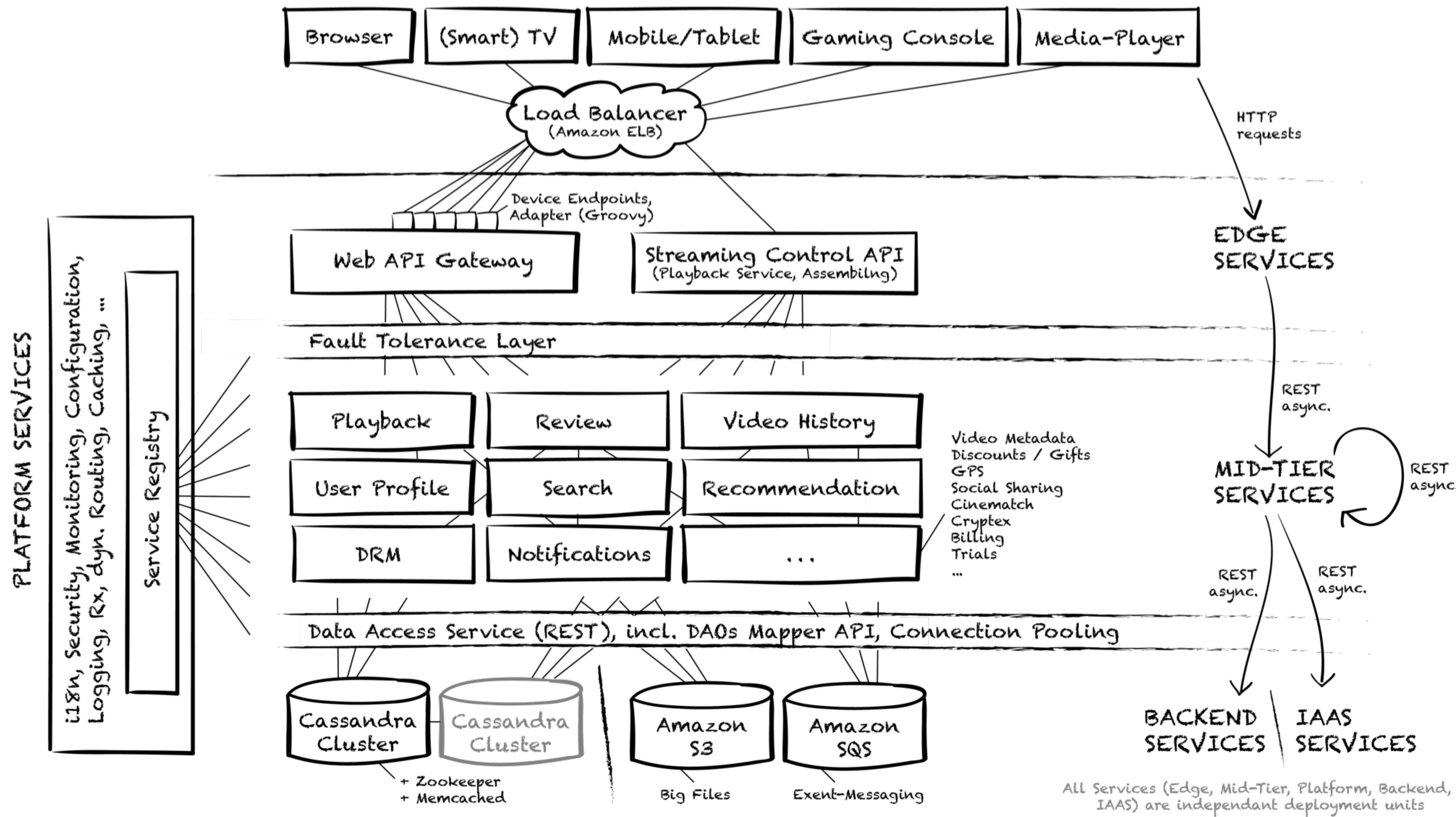
BRONSON

BRIDGET JONES THE EDGE OF REASON

TLC

DREAKING

# Netflix Architectural Overview (simplified)



# Collaborating Services, API Gateway, ...

- **Maintainability:** Low coupling between Services and Teams
- **Usability:** There is an API Gateway...
  - Screens-Designers are not bound by service design
  - Quick development of client-specific APIs
  - Device-specific adaptation is possible



- **Reliability:** There is an API Gateway (Bottleneck)
- **Efficiency:** An isolated call is not that performant
- **Maintainability:** Cooperation of Services for a useful feature implies communication needs and more complex interface design

# The organisational side

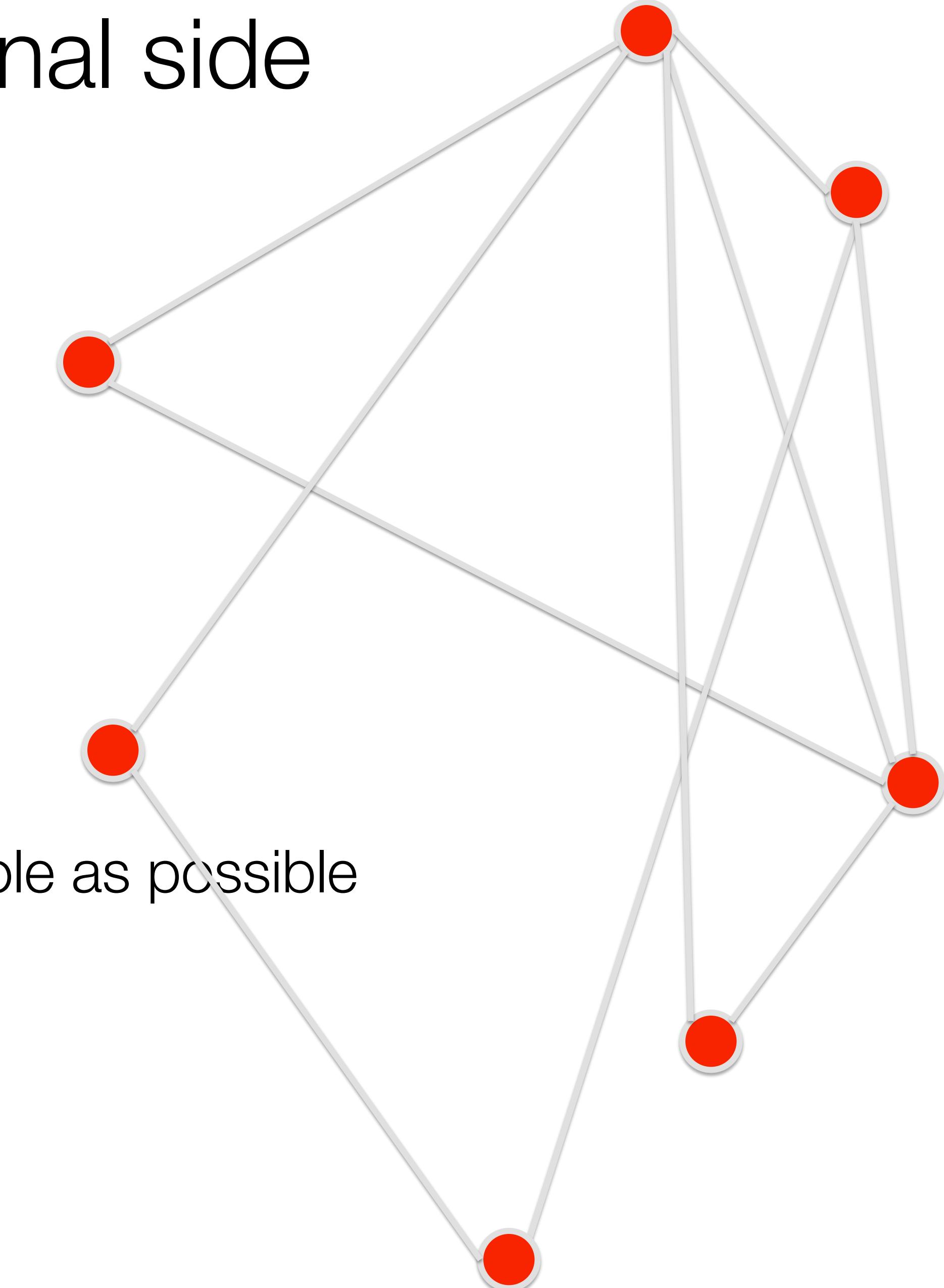
**Teams are ,fully‘ responsible for their Services**

- Development
- Release / Deployment
- Ops (not platform/system administration)

**No classic Management-Steering**

As little dependency from other teams or a central role as possible

- Little to no technical rules
- Uncoordinated releases





Freedom &  
Responsibility

# Used Technologies

## Platforms

Apache HTTP Server

Apache Tomcat

Bottle (Python)

...

## Programming Languages

Java

JavaScript

Groovy

Clojure

Scala

Dart

Python

Ruby

...

## Persistance

Cassandra

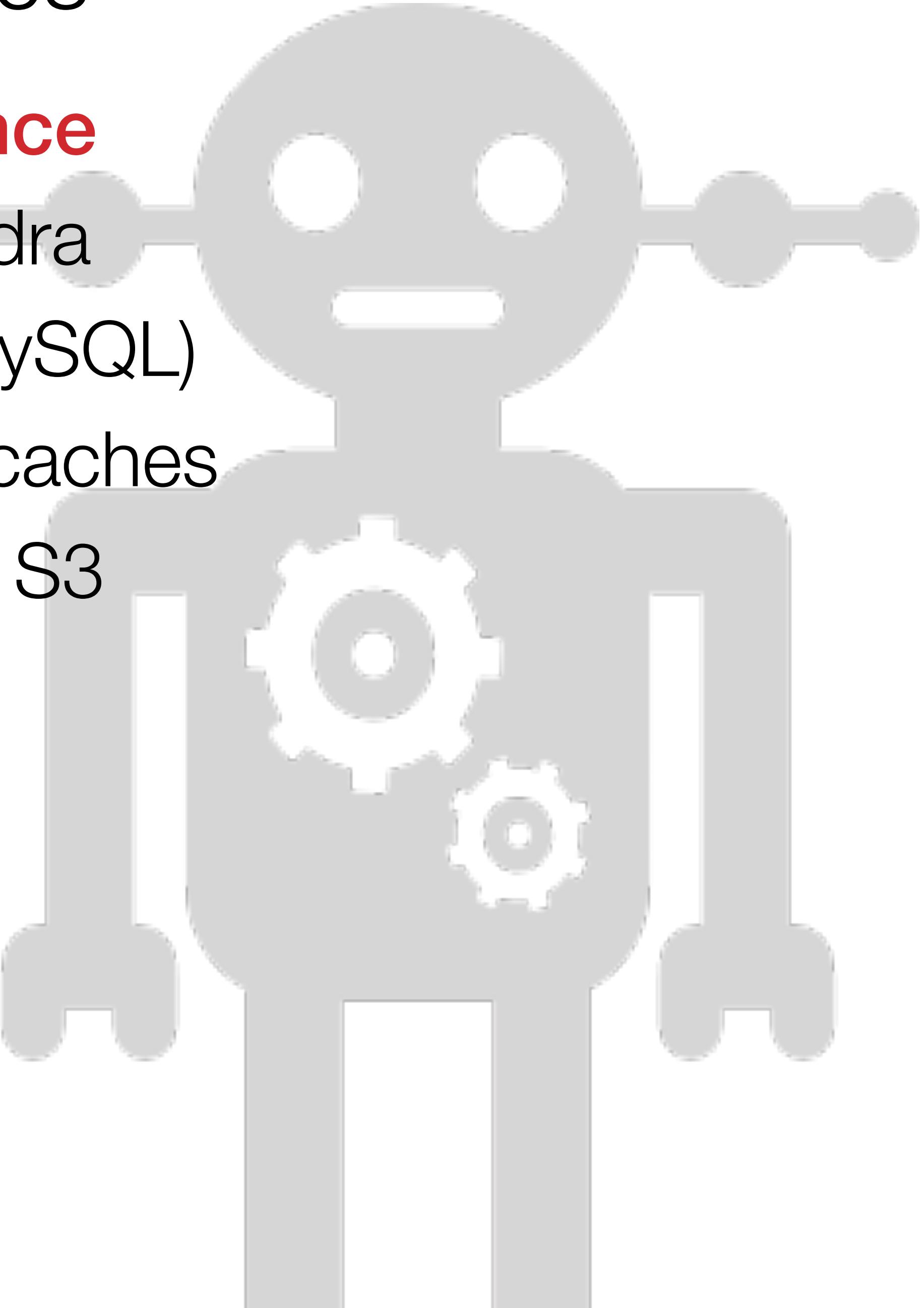
RDBMS (MySQL)

in-memory caches

Amazon S3

CDN

...



# Freedom & Responsibility...

- **Maintainability:** New Technologies and Frameworks are easily tested (in realistic conditions)
- **Longevity:** The Technology stack can be evolved incrementally (no long-term commitments)
- **Quality (any):** Always the best tool for the job (potentially)



- **Centralization:** Harder to cascade down “orders”
- **Time-To-Market:** Introducing new Technologies not using established best practices might be inefficient
- **Complexity:** More variability leads to higher overall complexity
  - Harder to coordinate and handle crosscutting stuff
  - Harder to have central rules or patches

# Mitigation ...

- Bring developers in touch with their **responsibility (goals)**
  - Tests for quality criteria (Latency, Robustness, Reliability, Scalability, ...)
- Give them **Feedback as fine grained and early** as possible
  - Continuous Delivery
- Work with **low viscosity** instead of rules and prescriptions

## Viscosity...

*“When faced with a change, engineers usually find more than one way to make the change. Some of the ways preserve the design, others do not (i.e. they are hacks.) When the design preserving methods are harder to employ than the hacks, then the viscosity of the design is high. It is easy to do the wrong thing, but hard to do the right thing.”*



(Robert C. Martin)

# Netflix Cloud Stack

## Individual Software

Applications, Services, ...

## PaaS

Runtime Environment, Web-/Application-Server,  
Frameworks for crosscutting concerns, Management Tools, ...

NETFLIX

OSS

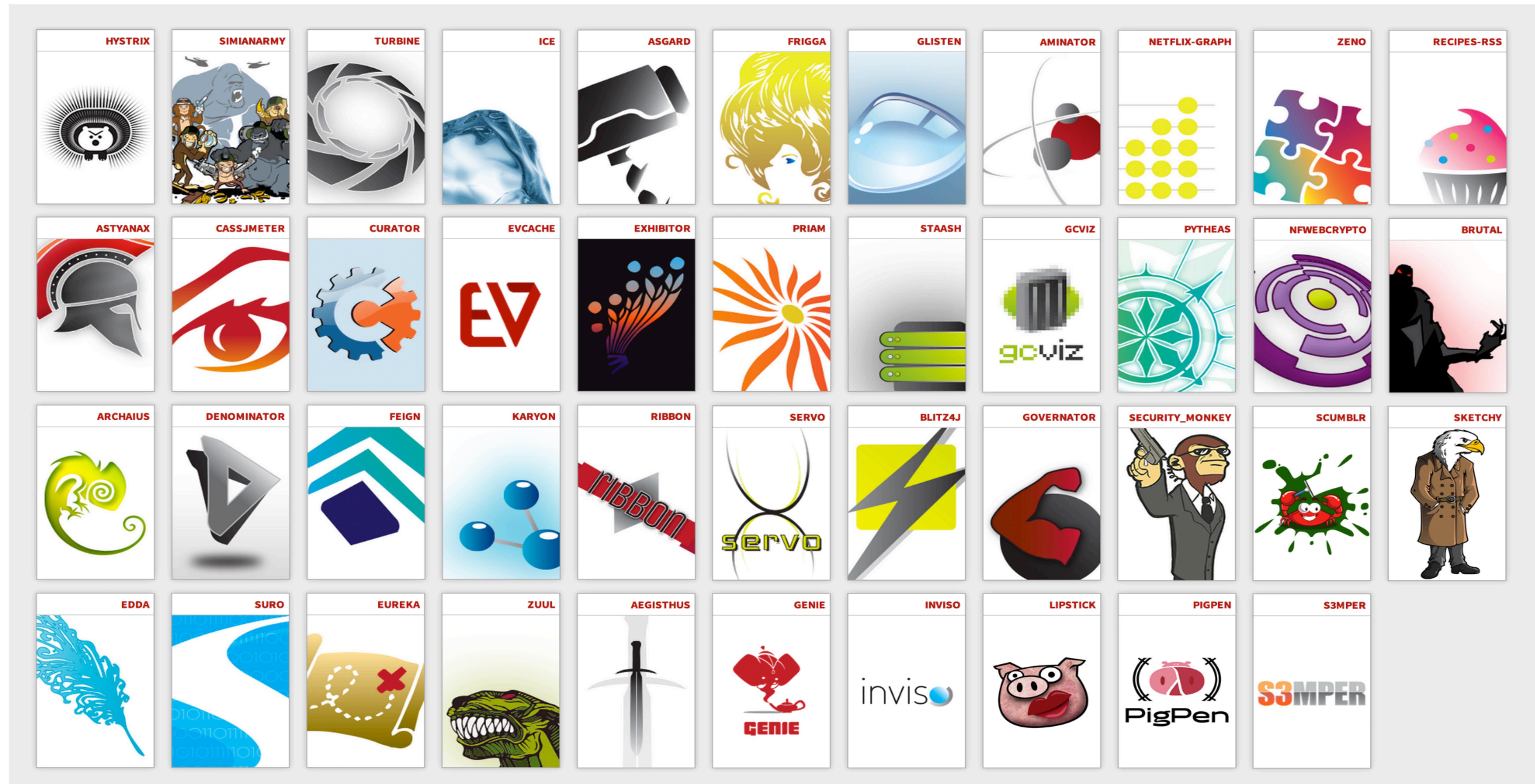
## IaaS

Virtual Machines, Network communication,  
Load Balancing, Datastore, ...

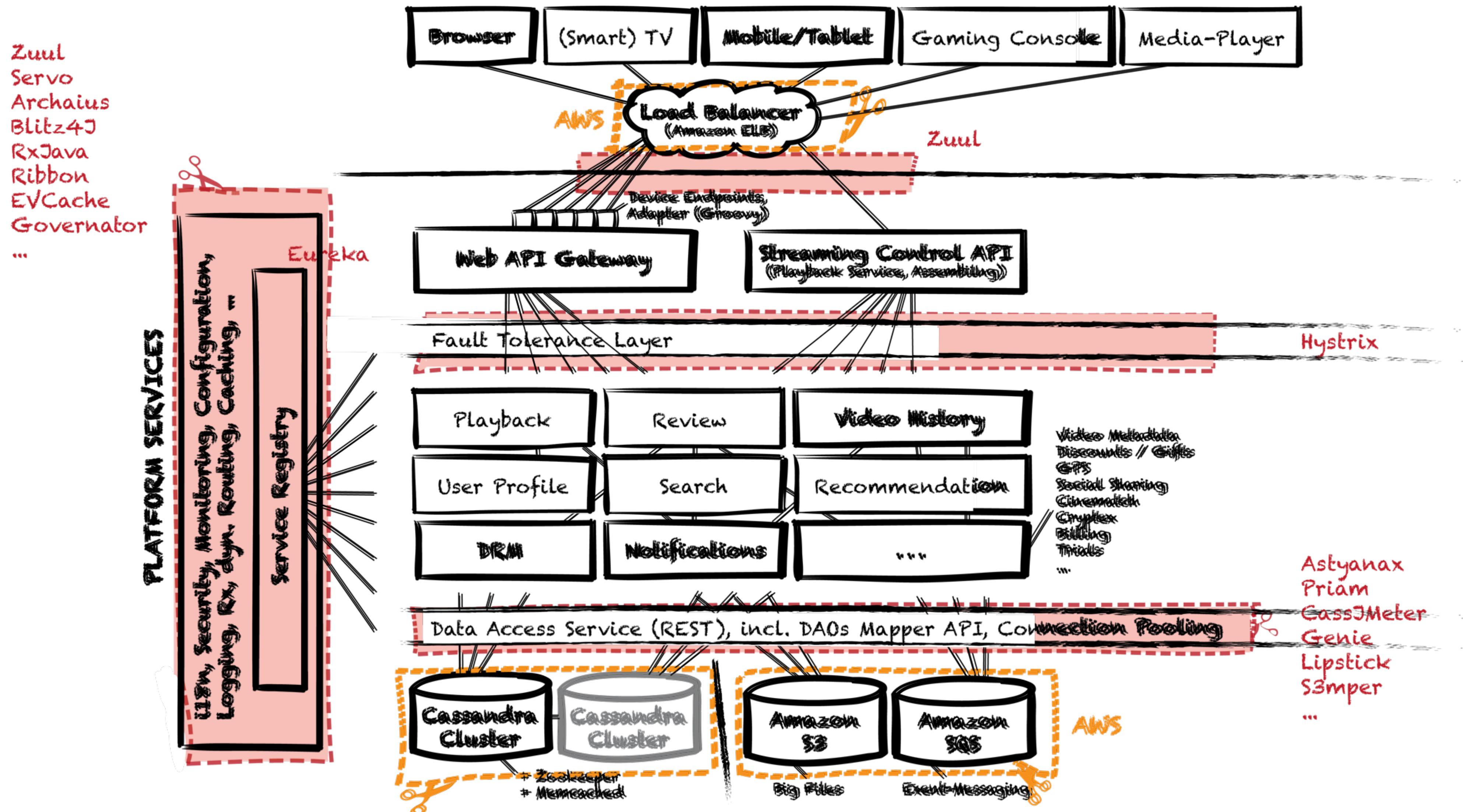


The Netflix Open Source Platform Components fill gaps in Amazon Web Services.  
The goal is to make cloud infrastructure more robust, flexible and glitch free.

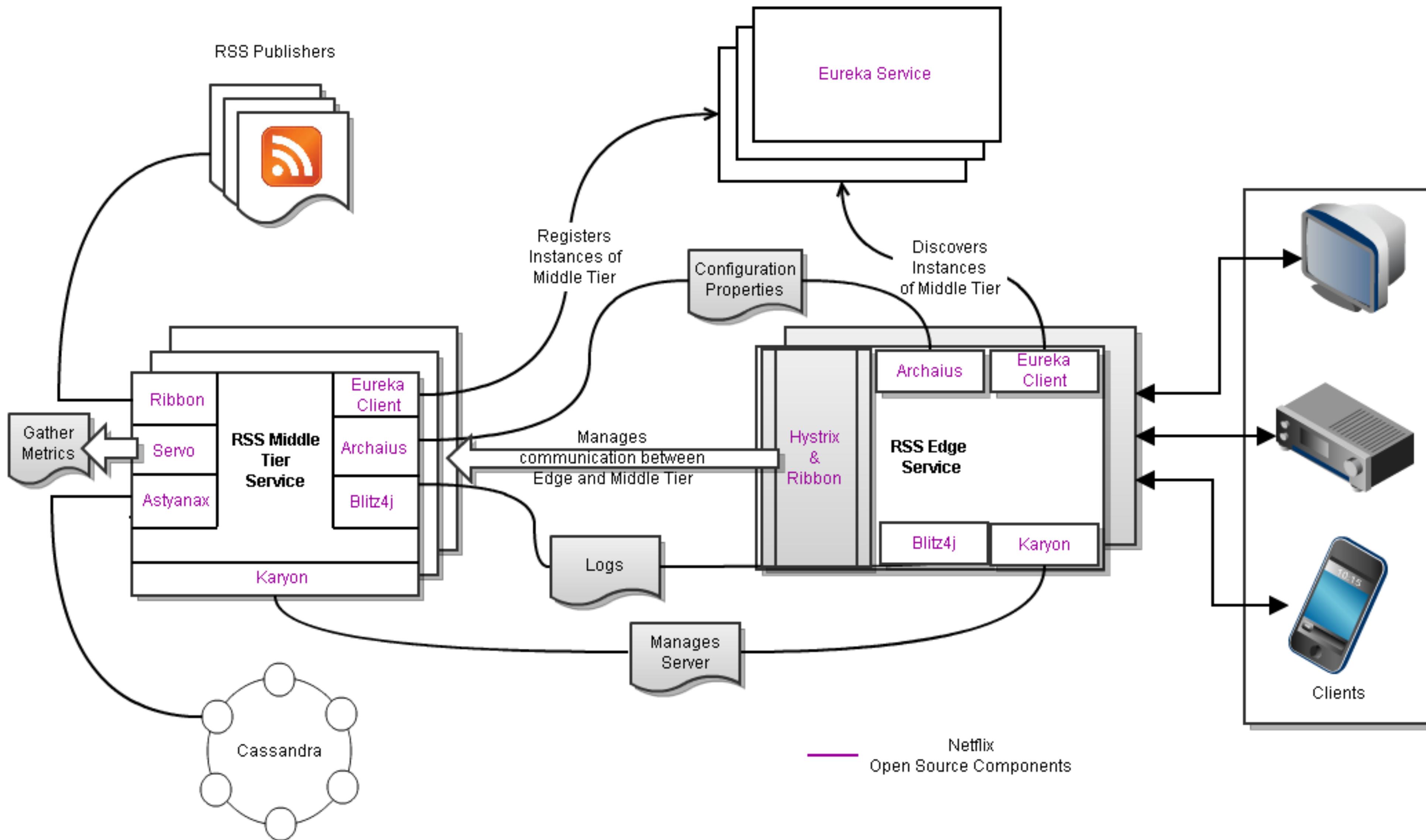
# Netflix Open Source Services



# Netflix OSS does what?



# Example Application (2 non-technical Services)



# Netflix OSS

- **Know-How:** Lower skill requirements for individual developers
- **Time-To-Market:** Quicker development of standard-services
- **Maintainability:** Partially centralized platform, higher quality code and documentation because of Open Sourcing
- **Complexity:** Lower viscosity



- **Individual Overhead:** Netflix specifics are prominent in the development space
- **Project Overhead:** A new project needs to establish ‘the easy way’

# Deployment at Netflix

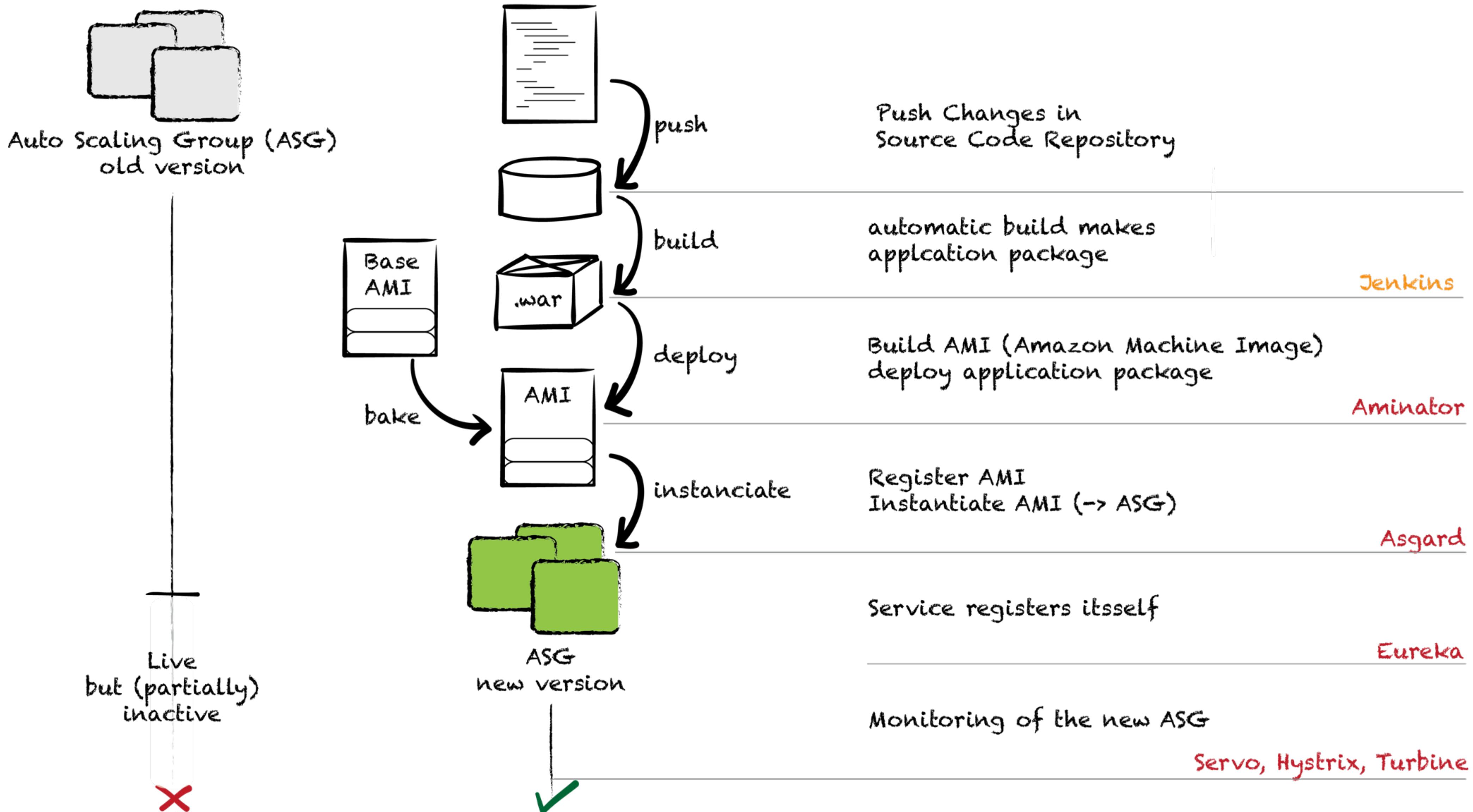
Answer to the **Coordination problem** when deploying?

Answer to **Complexity** and dependencies?

## Assisted Anarchy

- **approx. 100** Deployments a day
- Teams are self-governing and act independently
- **No** separate QA-department
- **No** overall coordination of deployments / releases

# Automation...



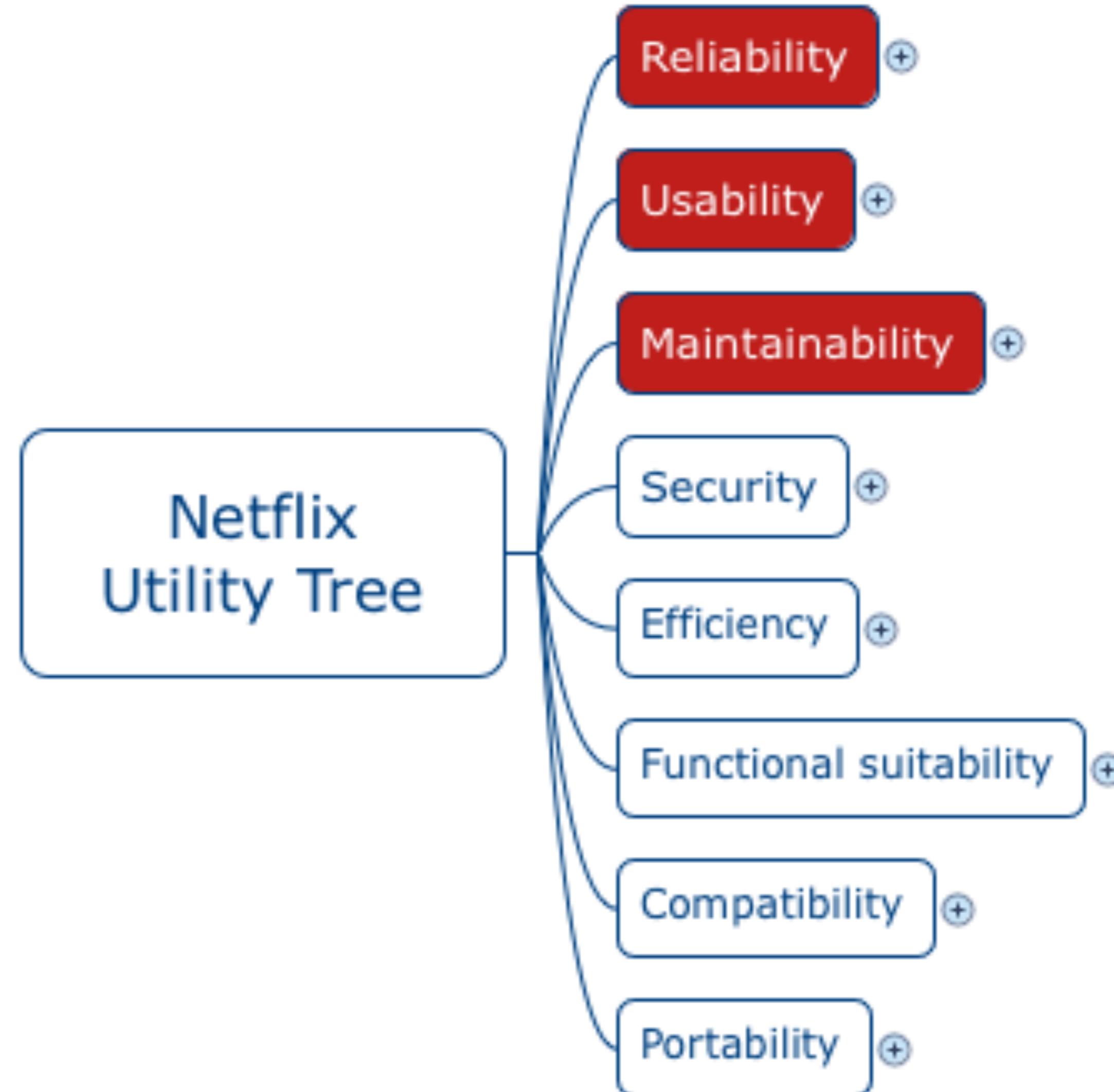
# Continuous Delivery, Canaries, ...

- **Robustness:** Fast Rollback (or Fallback essentially)
- **Testability:** Production is a realistic test environment and: cheaper than a separate testing environment
- **Know How:** Individual developers are decoupled from central settings and configurations



- **Infrastructure:**
  - Redundant Platforms/Containers/Hardware needed
  - High degree in automation and tool support needed
- **Observability:** Imposes high demands on logging and monitoring
- **Coordination:** Mainly broken down to first-come-first-serve

# In summary – Quality Requirements



# Reliability Scenarios

## Reliability

A part of the infrastructure fails. Our members can watch their films and TV series nonetheless without disruption or delay.

A new season of House of Cards goes online. All interested Members are able to watch it as soon as they want and in usual streaming quality.

A developer deploys a faulty new version of a service into production. He gets immediate and detailed feedback with little to no members affected.

A service doesn't work properly (malformed responses, high latency etc.). No other service is directly affected by the problem.

During Amazon Web Services (AWS) maintenance Cassandra Rings fail. No critical data is lost, all services work properly nonetheless

# Usability Scenarios

## Usability

A member compares our Application to the one of a competitor (on any device). He finds our solution more appealing and comfortable

A member uses a device that allows him to choose between a native app and our browser-based UI. He prefers the native solution.

A developer wants to build a new screen based on user-insight. He can quickly and easily access all necessary information and functionality without compromising his initial idea.

A member watches a film on his laptop. He closes his laptop and opens the Netflix App on a Tablet or Mobile Device and can proceed to watch the film where he left off.

# Maintainability Scenarios

## Maintainability

A team builds a new service. The way that is conformant with the architecture is easier than any other.

A new developer joins Netflix. After one day in a service team he is able to adapt the service, build and deploy it on his own.

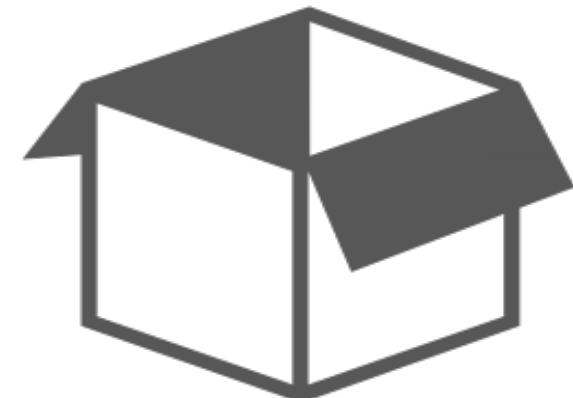
A new technology promises to be of use for several Netflix use cases. It is possible for one developer alone to get a slim use case into production quickly (within a few days). Other services or teams are not necessarily affected.

A new device should serve as client for Netflix. No existing client is affected by the design or implementation activities.

Our technology stack can be held cool enough over time, so that we can present it at conferences attract new talent.

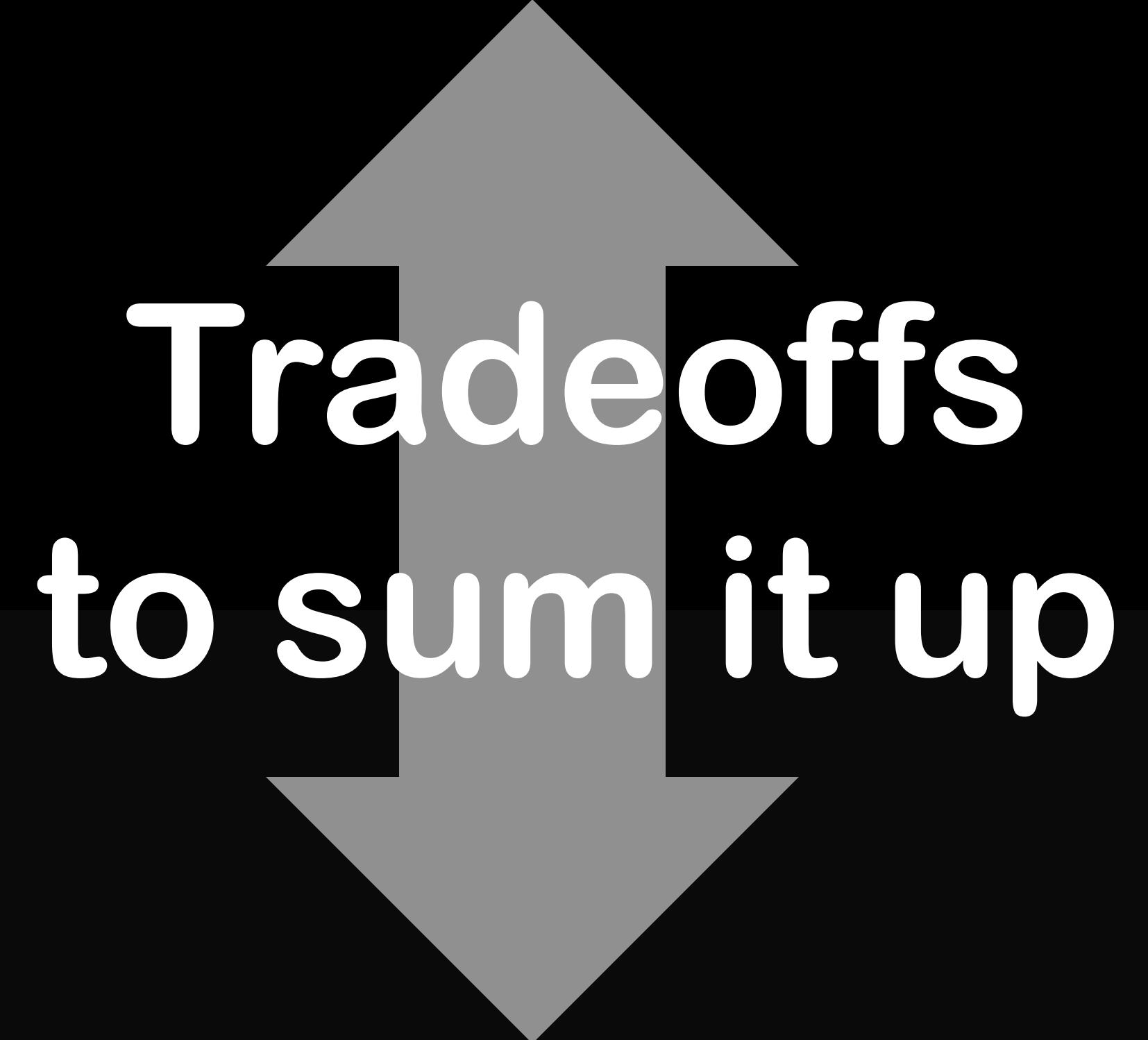
# Important Constraints

## Which Context is necessary to make it work?



1. Development of a long-lived product
2. The size of the product justifies several teams
3. Selforganizing teams fit into management practice
4. Deployment in the Cloud is feasible
5. Failing during Deployment or Release is possible
6. Using/Integrating Open Source-Solutions is easy

This is more important

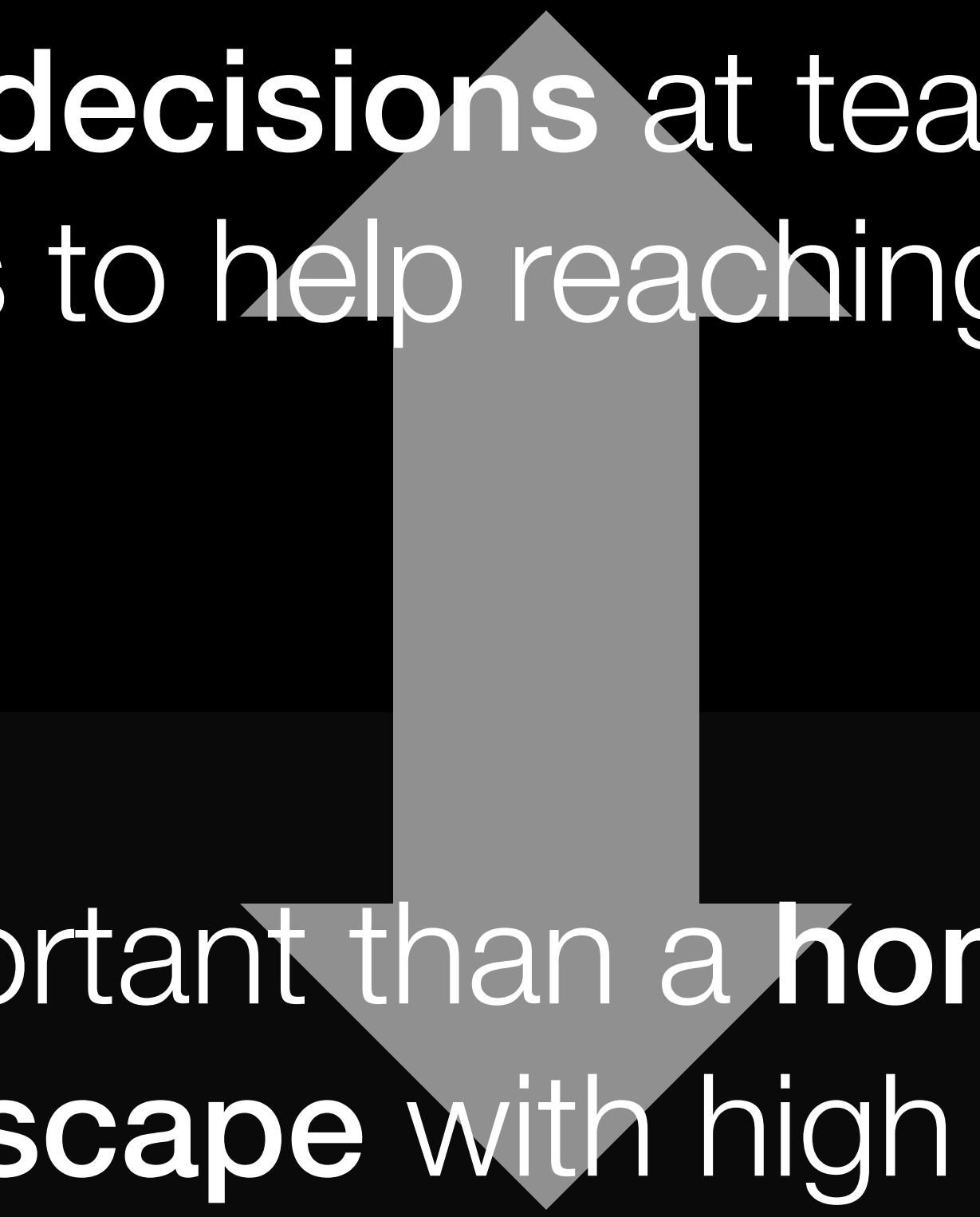


Tradeoffs  
to sum it up

Than that...



NETFLIX



**Technology decisions** at team level and local experiments to help reaching quality goals ...

...are more important than a **homogenous System landscape** with high integrity.



NETFLIX

Innovation and growth are very important aspects of software development...

...Control, central panning and transparent status for management are clearly inferior motives.

NETFLIX

Fast development and **delivery** of new functionality is  
more important than...

...the complete **lack** of bugs and problems in  
production.

NETFLIX



To reach **high quality** for the user (and corresponding benefits in the market)...

...redundant development and low reuse possibilities are perfectly OK.



**NETFLIX**

High (**initial**) overhead for framework components,  
automation and infrastructure abstraction are  
justifiable...

...to secure the **long-term suitability** of the solution  
and an up-to-date stack.

NETFLIX

# Netflix Tech Blog

→ <http://techblog.netflix.com>

The screenshot shows a web browser window displaying the Netflix Tech Blog at [techblog.netflix.com](http://techblog.netflix.com). The page features the Netflix logo on the left and the title "The Netflix Tech Blog" on the right. A post from Wednesday, April 8, 2015, is highlighted with the title "Introducing Vector: Netflix's On-Host Performance Monitoring Tool" by Martin Spier, Amer Ather, and Brendan Gregg. Below the post is a large "Vector" logo featuring a cartoon owl wearing a top hat. To the right of the post is a sidebar titled "Links" containing a list of other Netflix blogs and resources.

Wednesday, April 8, 2015

**Introducing Vector: Netflix's On-Host Performance Monitoring Tool**

by [Martin Spier](#), [Amer Ather](#) and [Brendan Gregg](#)



Vector is an open source host-level performance monitoring framework, which exposes hand-picked, high-resolution system and application metrics to every engineer's browser. Having the right metrics available on demand and at a high resolution is key to understanding how a system behaves and correctly troubleshooting performance issues. Previously, we'd login to instances as needed, run a variety of commands, and sift through the output for the metrics that matter. Vector cuts down the time to get to those metrics, helping us respond to incidents more quickly.

Vector provides a simple way for users to visualize and analyze system and application-level metrics in near real-time. It leverages the battle tested open source system monitoring framework, Performance Co-Pilot (PCP), layering on top a flexible and user-friendly UI. The UI polls metrics at up to 1 second resolution, rendering the data in completely configurable dashboards that simplify cross-metric correlation and analysis.

**Links**

- [Netflix US & Canada Blog](#)
- [Netflix America Latina Blog](#)
- [Netflix Brasil Blog](#)
- [Netflix Benelux Blog](#)
- [Netflix DACH Blog](#)
- [Netflix France Blog](#)
- [Netflix Nordics Blog](#)
- [Netflix UK & Ireland Blog](#)
- [Netflix ISP Speed Index](#)
- [Open positions at Netflix](#)
- [Netflix Website](#)
- [Facebook Netflix Page](#)
- [Netflix UI Engineering](#)
- [RSS Feed](#)

# Netflix Open Source Software

→ <http://netflix.github.io>

The screenshot shows a web browser displaying the Netflix Open Source Software Center at <http://netflix.github.io>. The page features a header with the Netflix OSS logo and the text "Netflix Open Source Software Center". Below the header, there are tabs for "Repositories" and "Powered By NetflixOSS", with "Repositories" being active. A section titled "Availability" displays cards for various projects: HYSTRIX (with a bear icon), SIMIANARMY (with a cartoon monkey icon), TURBINE (with a gear icon), ICE (partially visible), ASGARD (partially visible), FRIGGA (with a yellow flower icon), and GLISTEN (partially visible). The SIMIANARMY card is highlighted with a red banner at the top containing the text "SimianArmy". Below the banner, it says "Tools for keeping your cloud operating in top form. Chaos Monkey is a resiliency tool that helps applications tolerate random instance failures." followed by a "More Info" link. To the right of the project cards, there is a sidebar with a list of mailing lists and a "Stay in Touch" section with links to the Netflix Tech Blog, @NetflixOSS, Slideshare, Netflix Meetup, and Jobs at Netflix.

of our open source projects. Click on the following links to subscribe and/or maintain your subscriptions.

- Asgard Mailing List
- Astyanax Mailing List
- Curator Mailing List
- Exhibitor Mailing List
- SimianArmy Mailing List

Stay in Touch

- Netflix Tech Blog
- @NetflixOSS
- Slideshare
- Netflix Meetup
- Jobs at Netflix

# Thank You.

Questions are welcome!



stefan.toth@embarc.de



@st\_toth

DOWNLOAD SLIDES:

<http://www.embarc.de/blog/>