O'REILLY®
Software Architecture
ENGINEERING THE FUTURE OF SOFTWARE

softwarearchitecturecon.com
#oreillysacon

Reverse evaluating Netflix's architecture

Stefan Toth

@st_toth; st@embarc.de

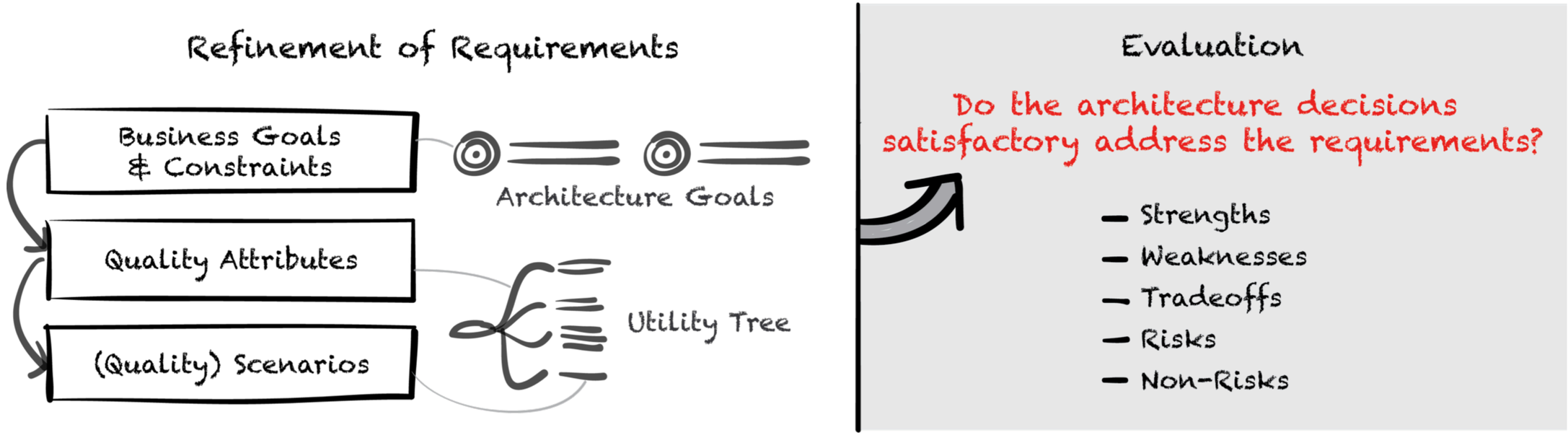
embarc
Software Consulting GmbH

4

advanced

cool

What is architecture evaluation?



Methods:

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

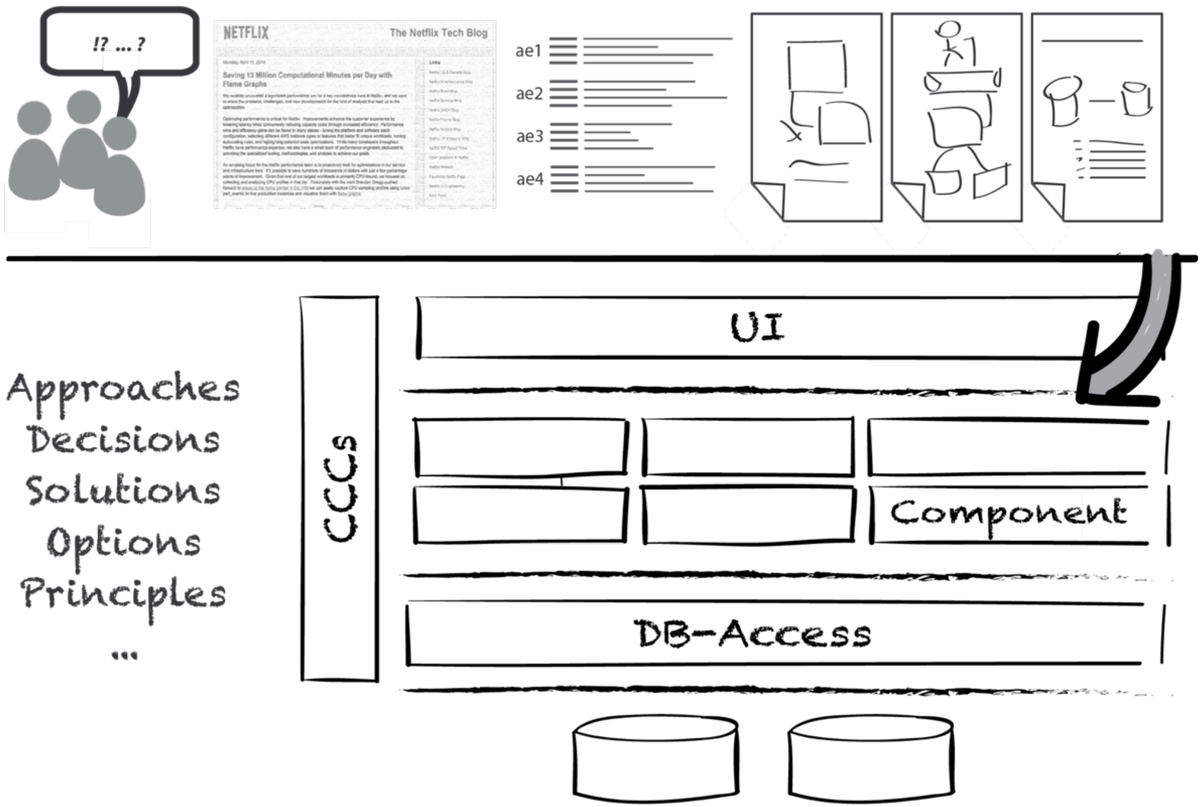
Reverse evaluating?

Reverse Evaluation

Which context makes the observed architecture concepts ideal?

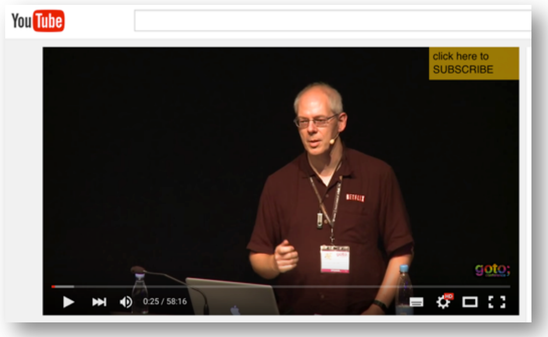
- Intentional Tradeoffs
- (Quality) Scenarios
- Quality Attributes
- Business Goals
- Constraints / Absence of them
- ...

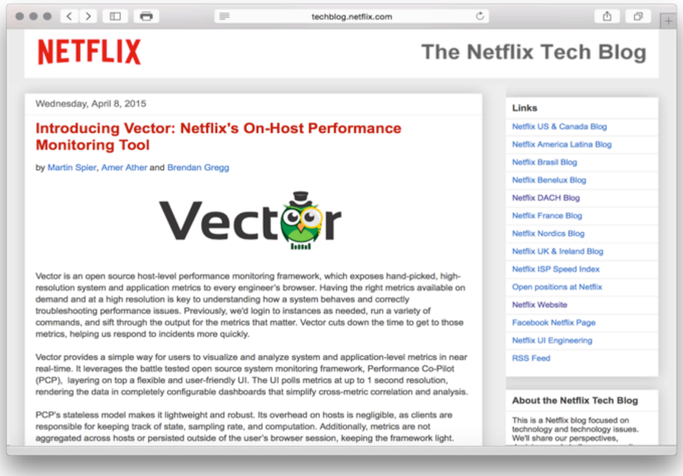
Architecture Archeology

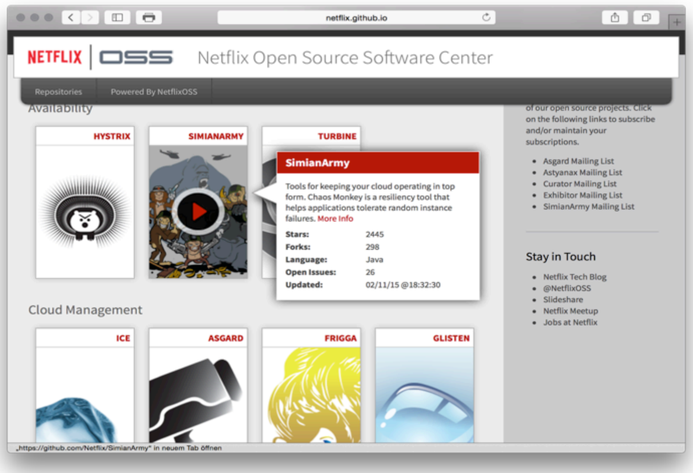


Disclaimer

- 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
  - …





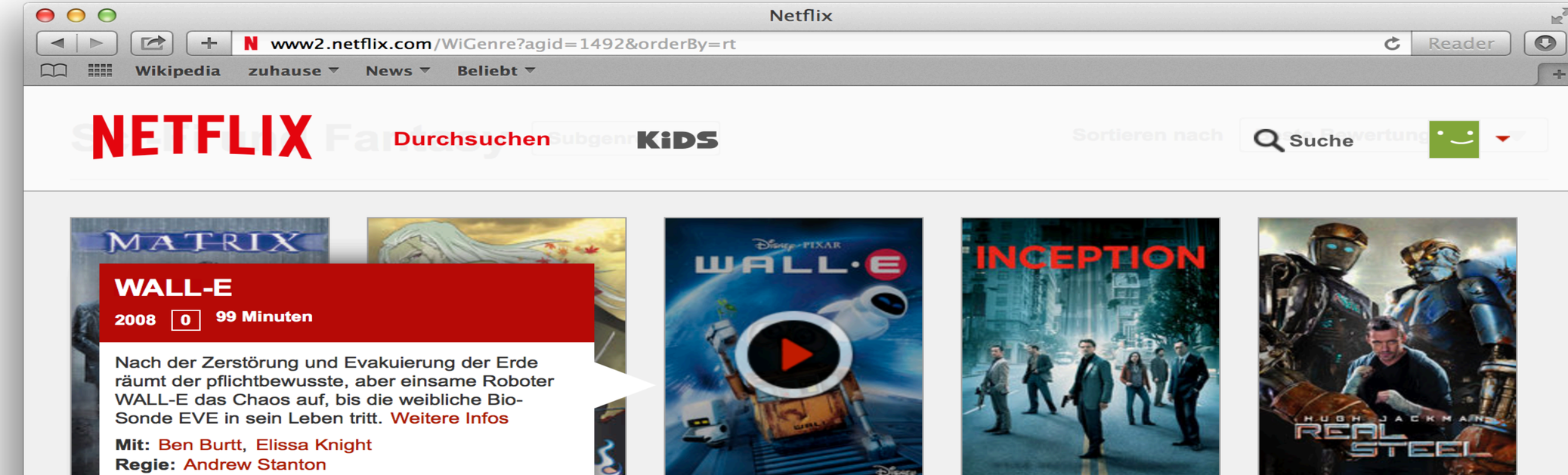



NETFLIX

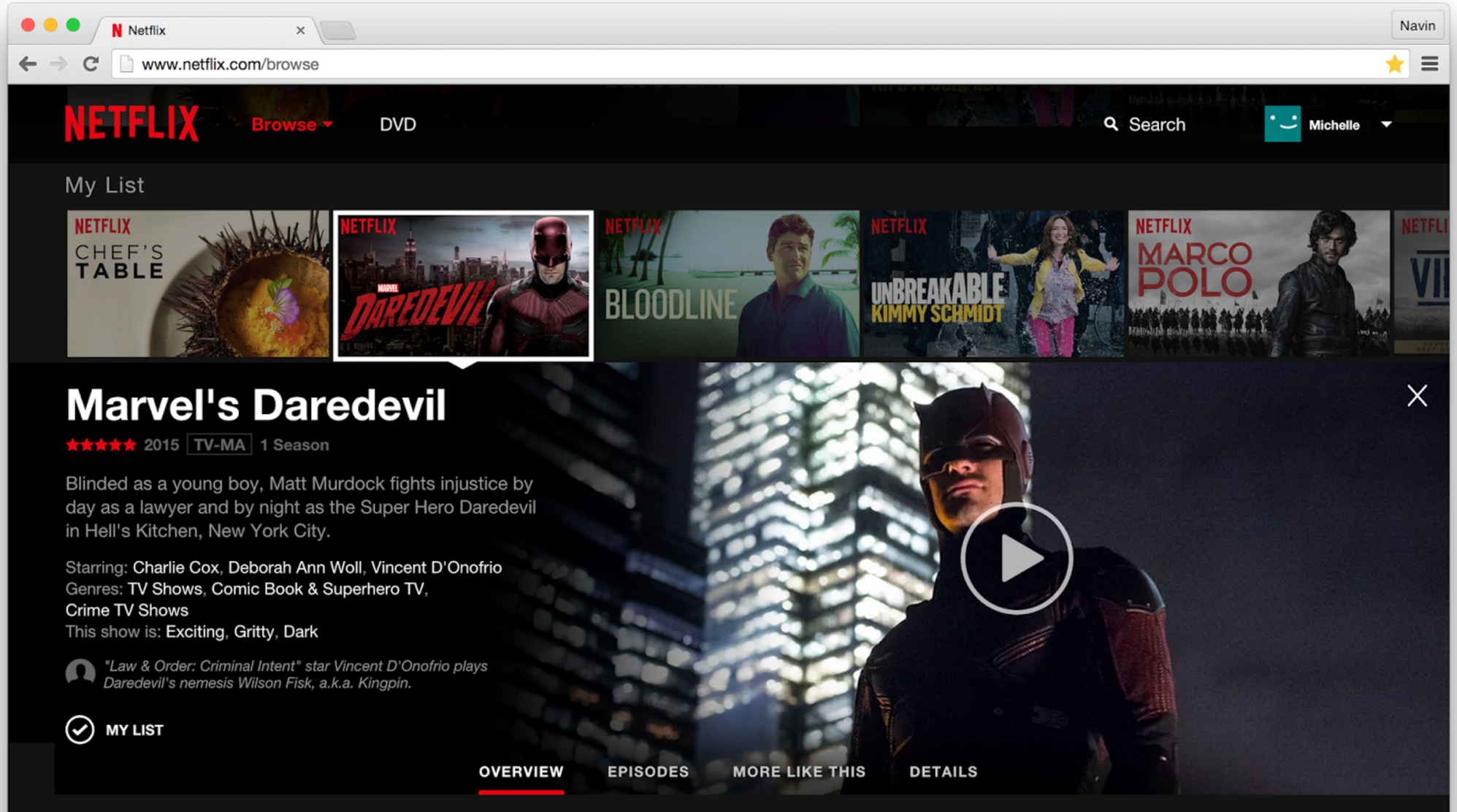
Was ist Netflix?

> *"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."*

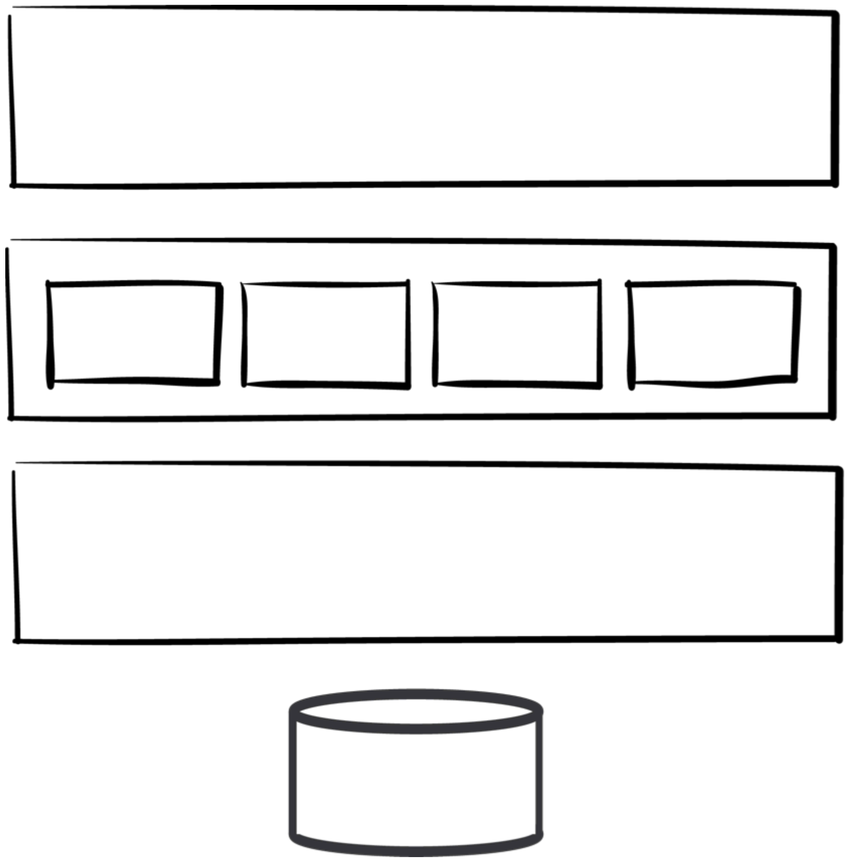
(James Lewis, Martin Fowler)

Common Characteristics

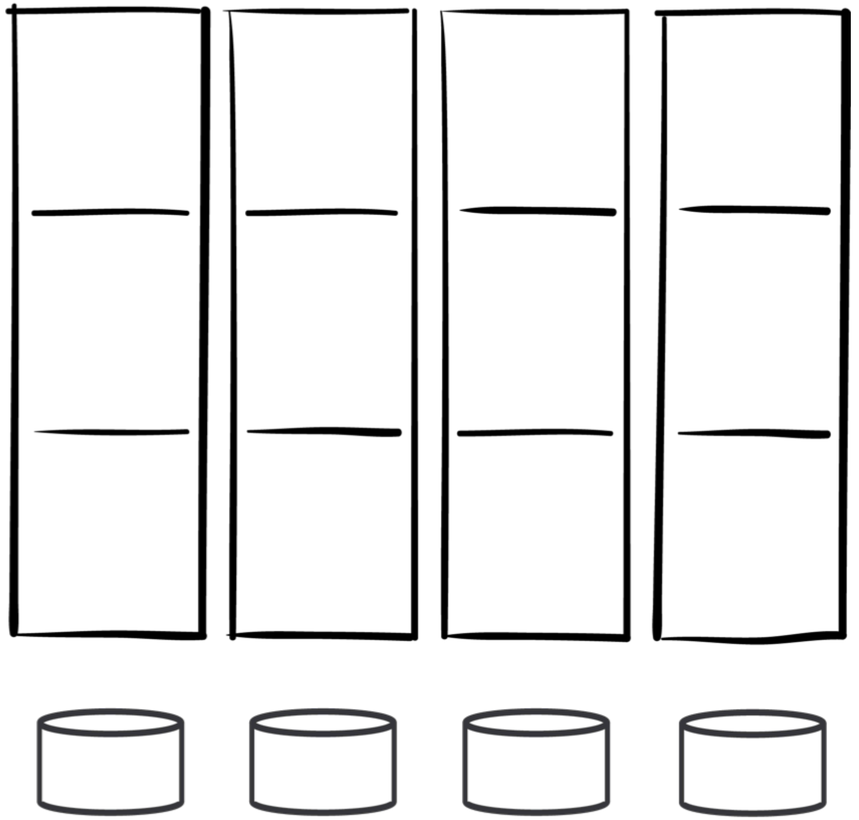
- suite of small services
- running in its own process
- built around business capabilities
- independently deployable
- bare minimum of centralized management (languages, platforms, technologies)

In Principle…

Layers



Slices / Verticals



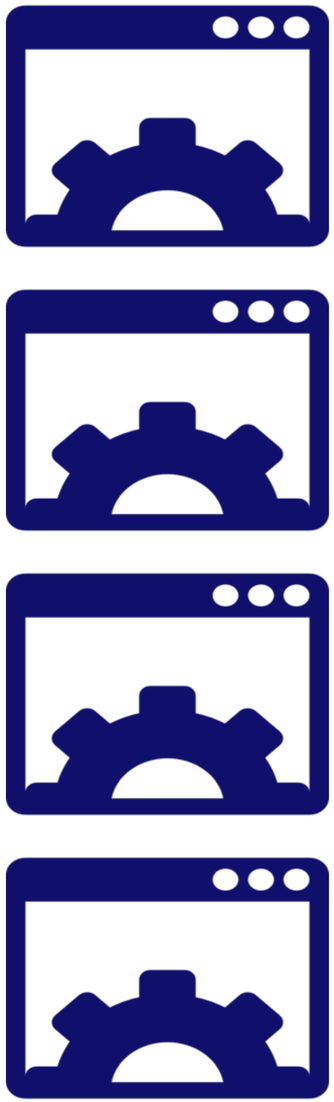
What 600+ Microservices feel like

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

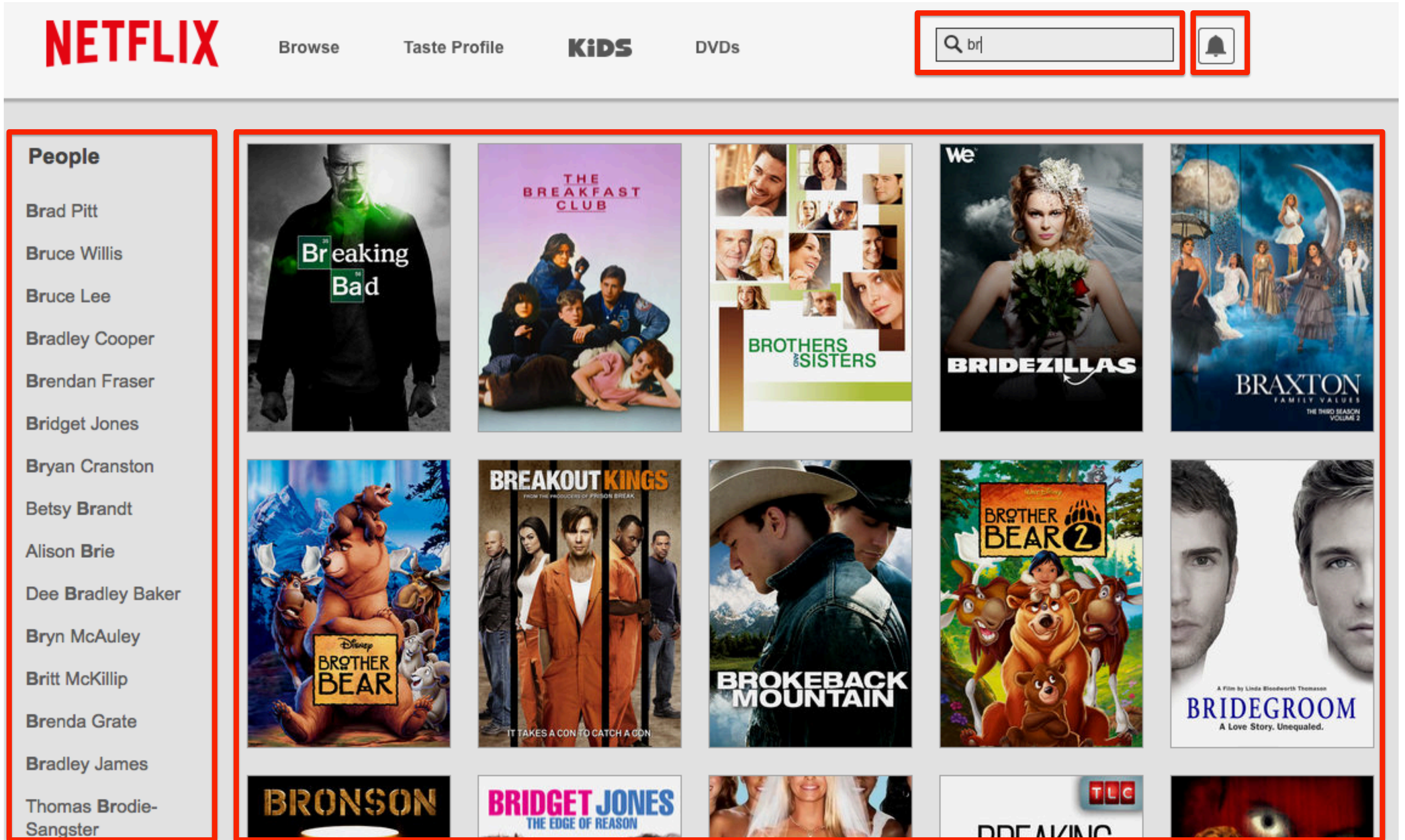
Microservices at Netflix



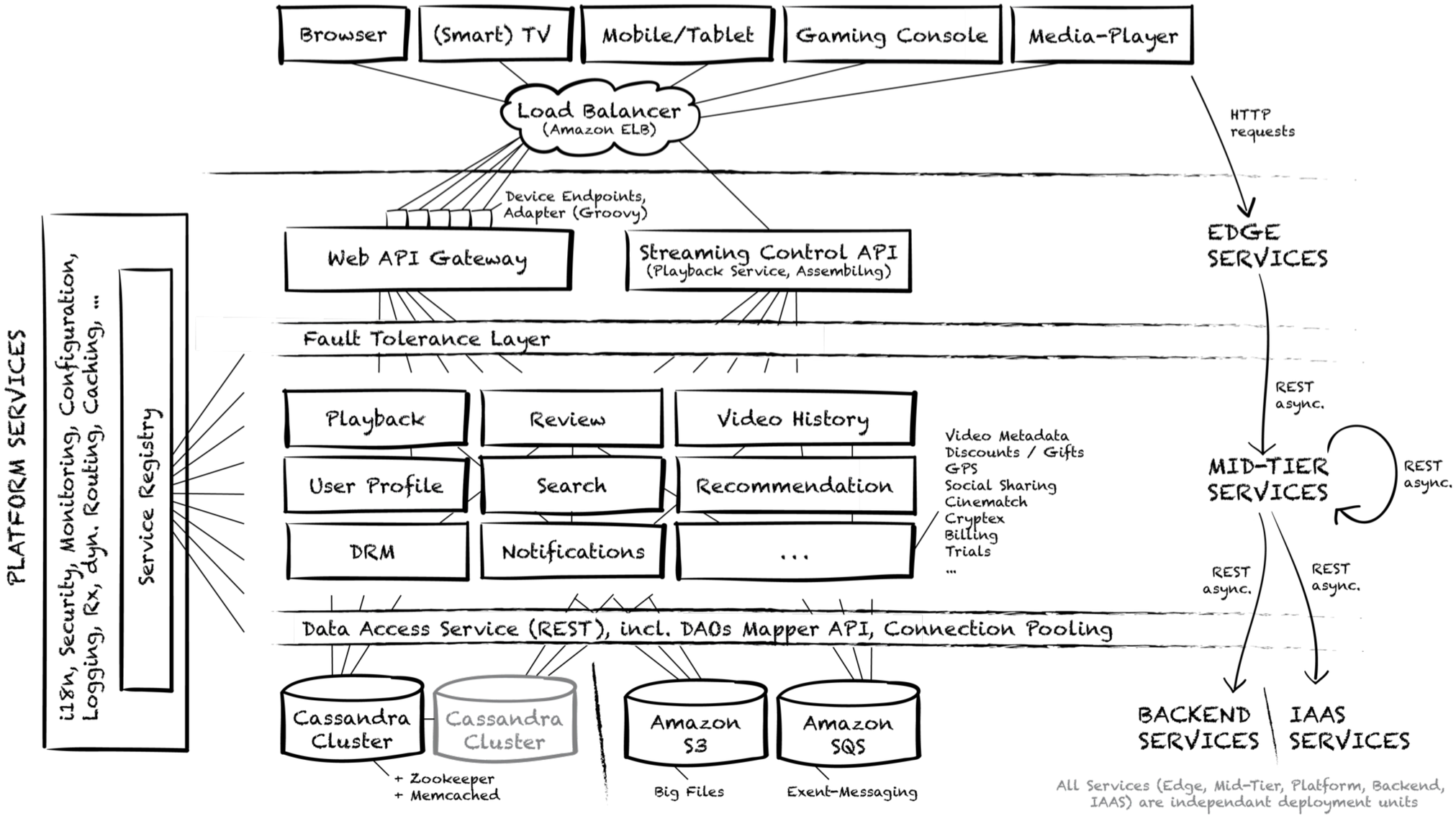
Functionality and technical Base („Platform“)

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

Services at work



Netflix 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-spezific 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

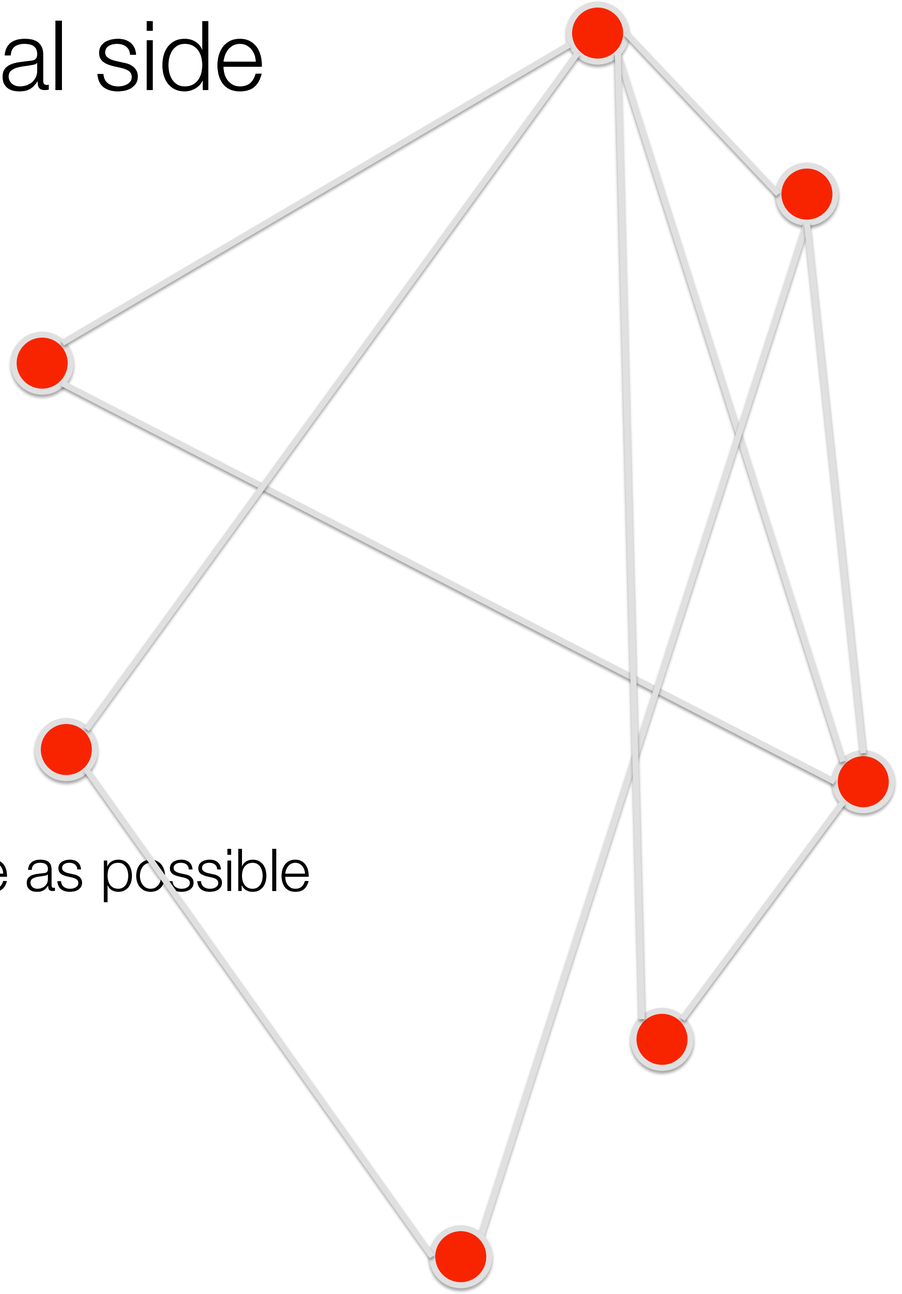
Teams are **‚fully' responsible** for their Services

- 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)

...

Persistance

Cassandra

RDBMS (MySQL)

in-memory caches

Amazon S3

CDN

...

Programming Languages

Java

Groovy

Scala

Python

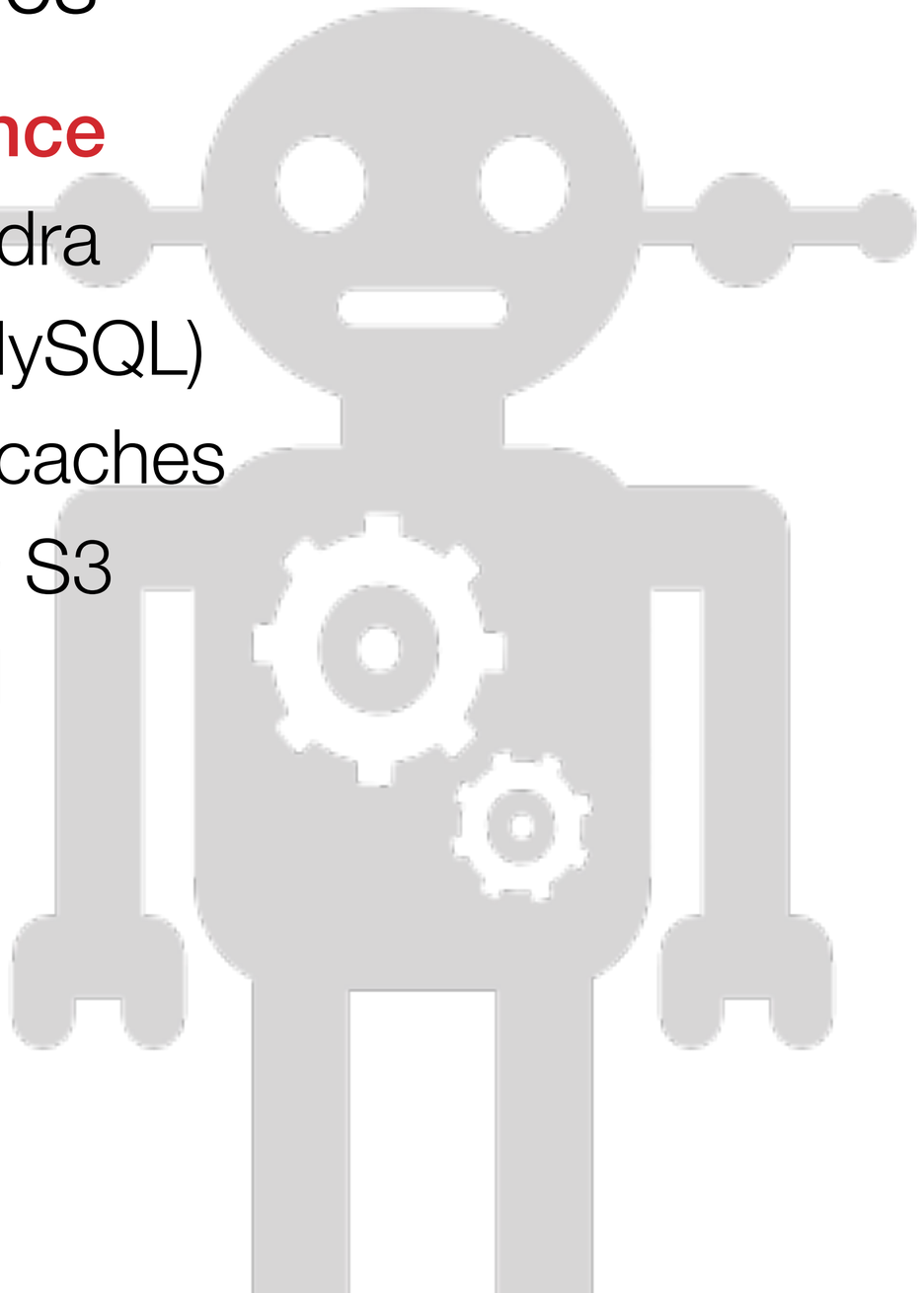
JavaScript

Clojure

Dart

Ruby

...



Freedom & Resposibility…

- **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 …

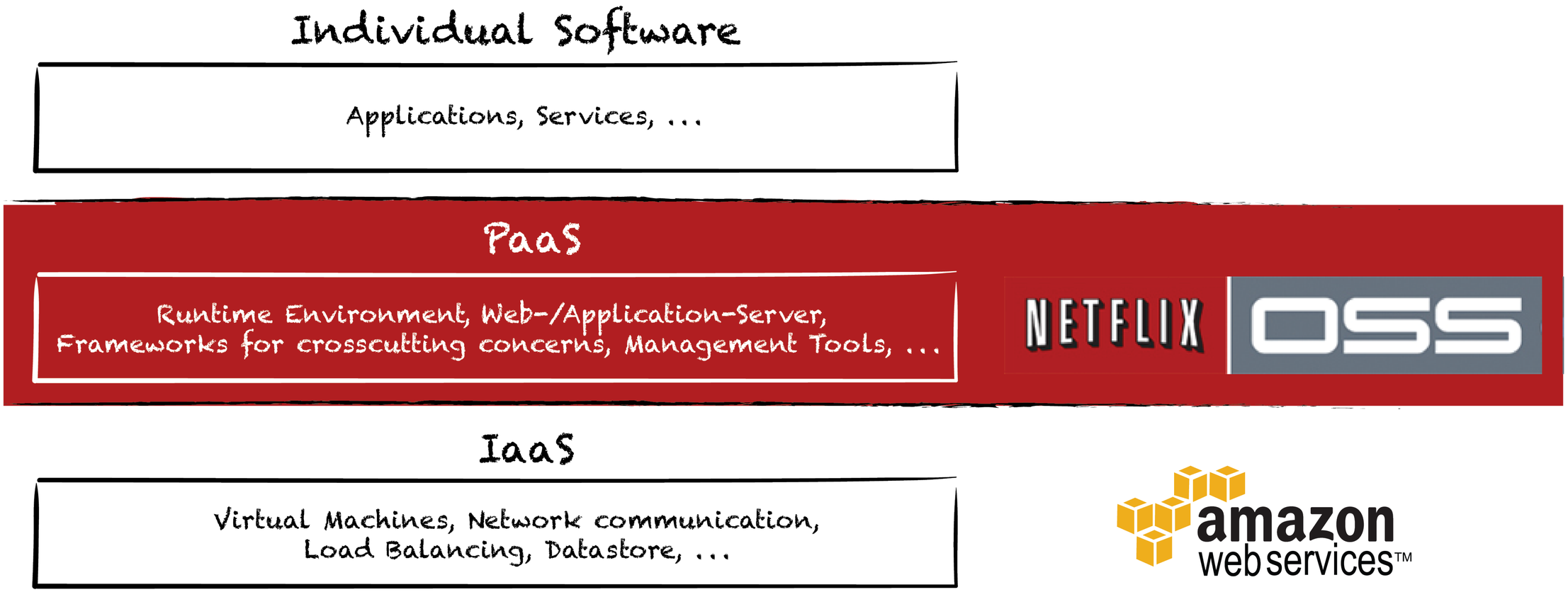
- Bring developers in touch with their **responsibility (goals)**
  - Tests for quality criteria (Latency, Robustness, Reliability, Scalability, …)
- Give them **Feedback** as **fine grained** and **early** as possible
  - Continuous Delivery
- Work with **low viscosity** instead of rules and prescriptions

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."*

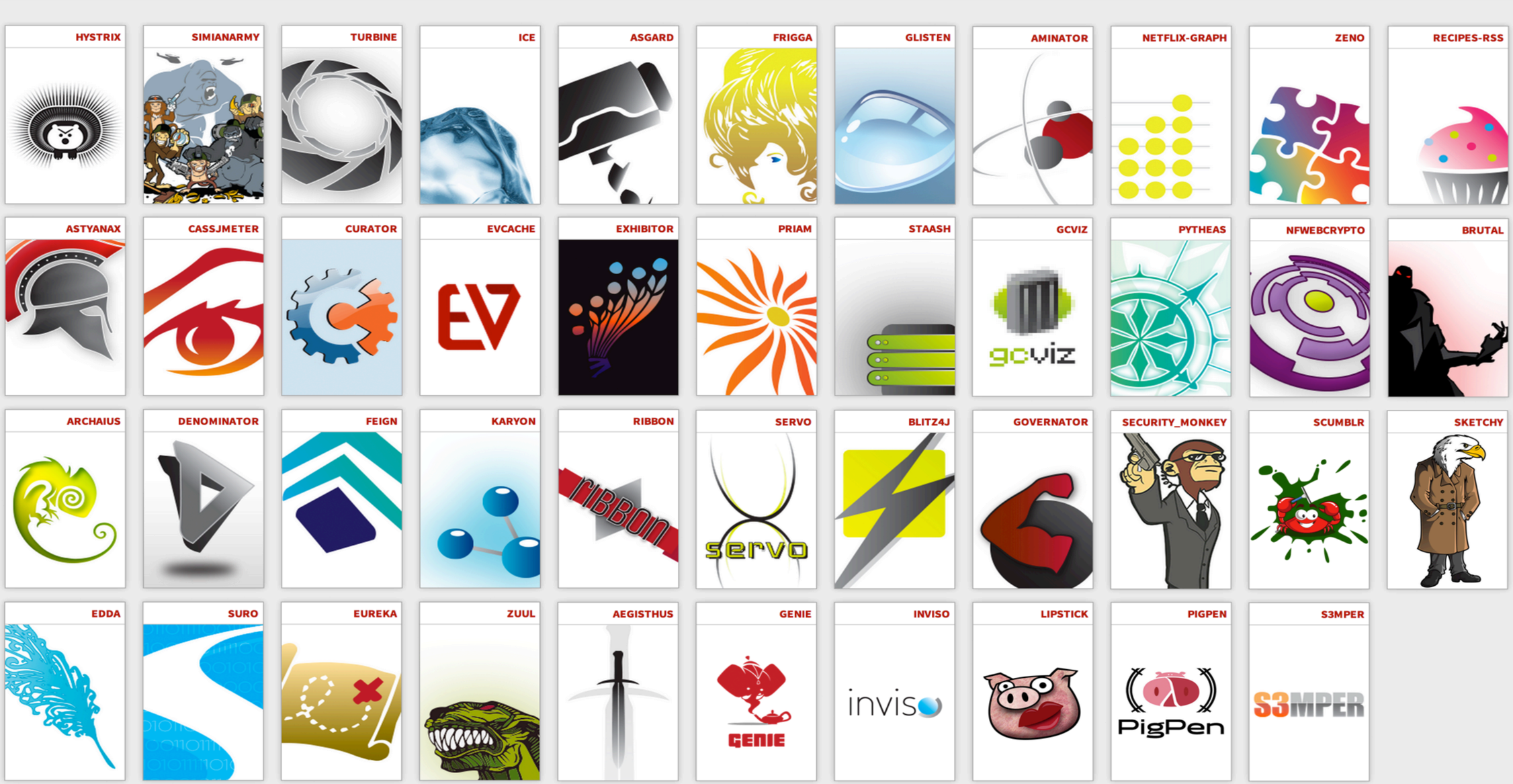
(Robert C. Martin)

Netflix Cloud Stack

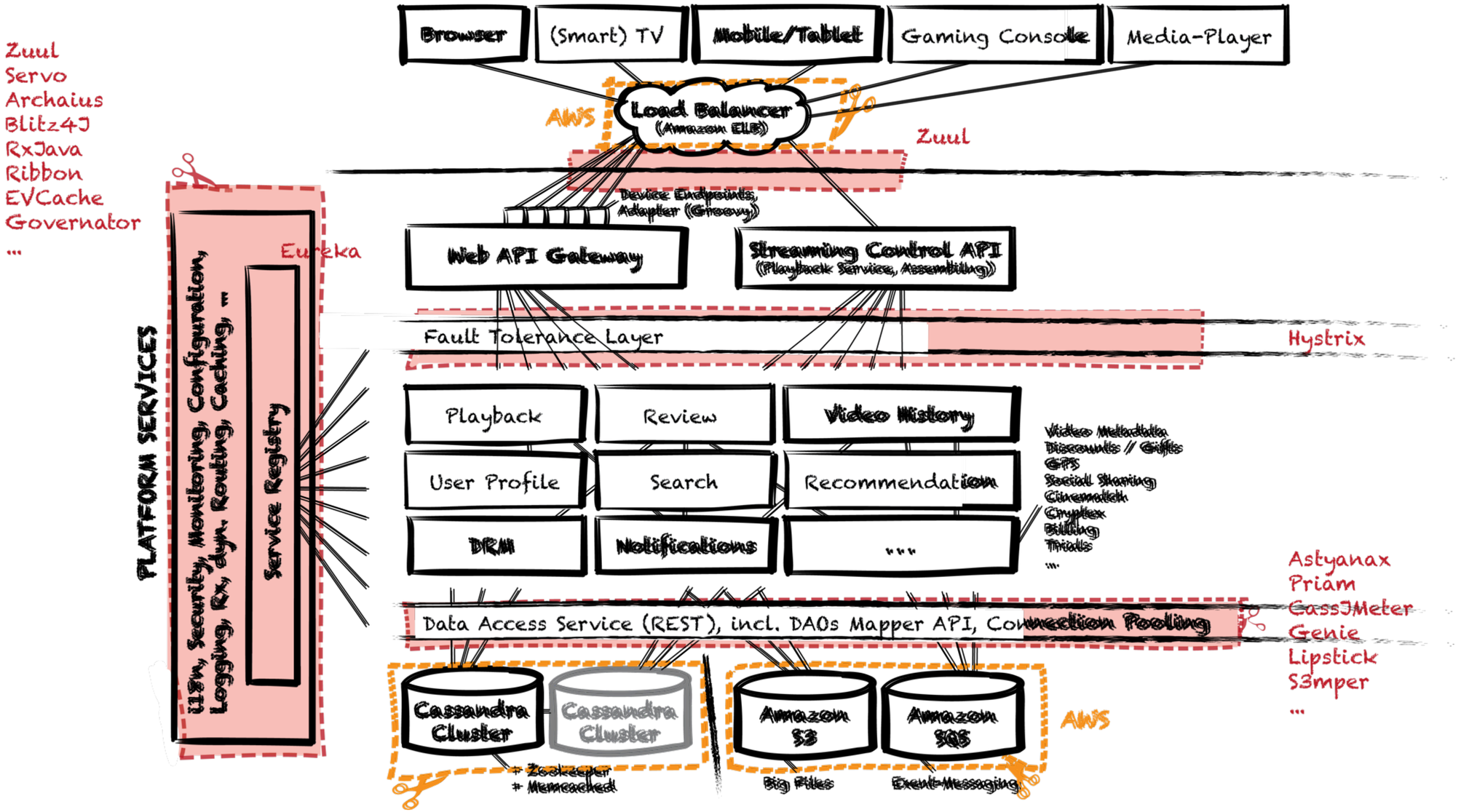


The Netflix Open Source Platform Components fill gaps in Amazon Web Services. The goal is to make cloud infrastructure more robust, flexible and glitch free.

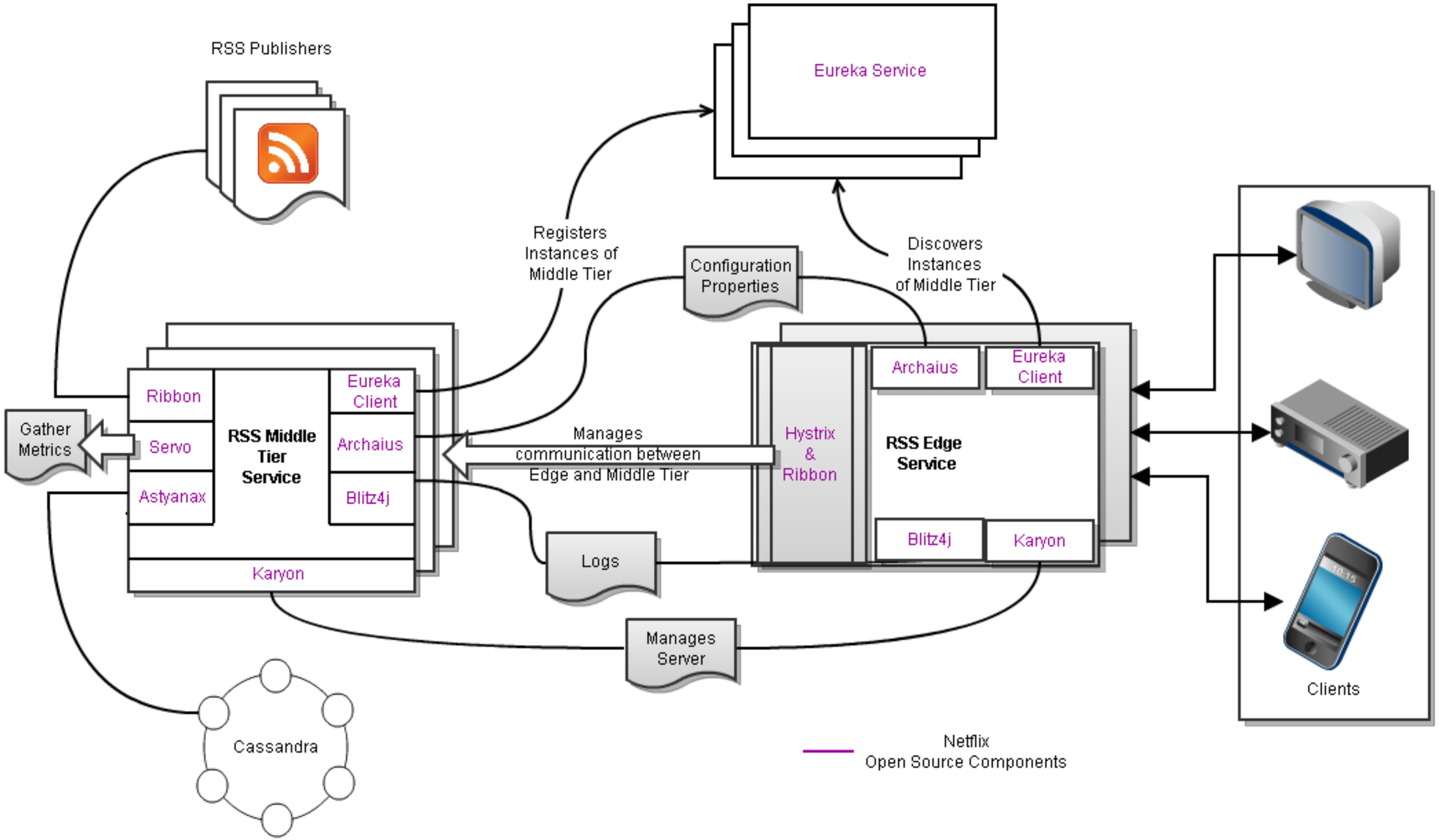
Netflix Open Source Services



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

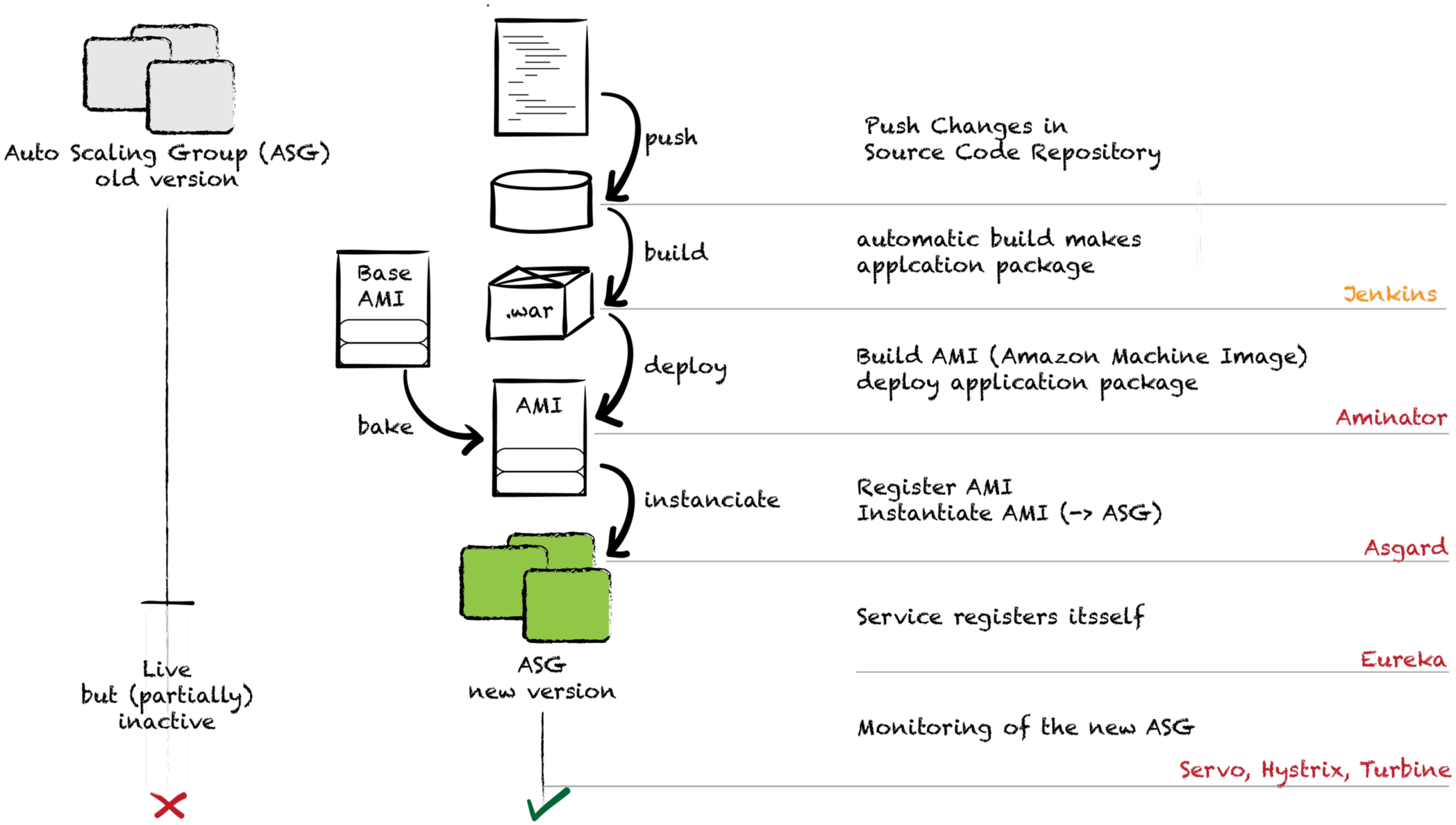
Answer to the **Coordination problem** when deploying?

Answer to **Complexity** and dependencies?

Assisted Anarchy

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

Automation...

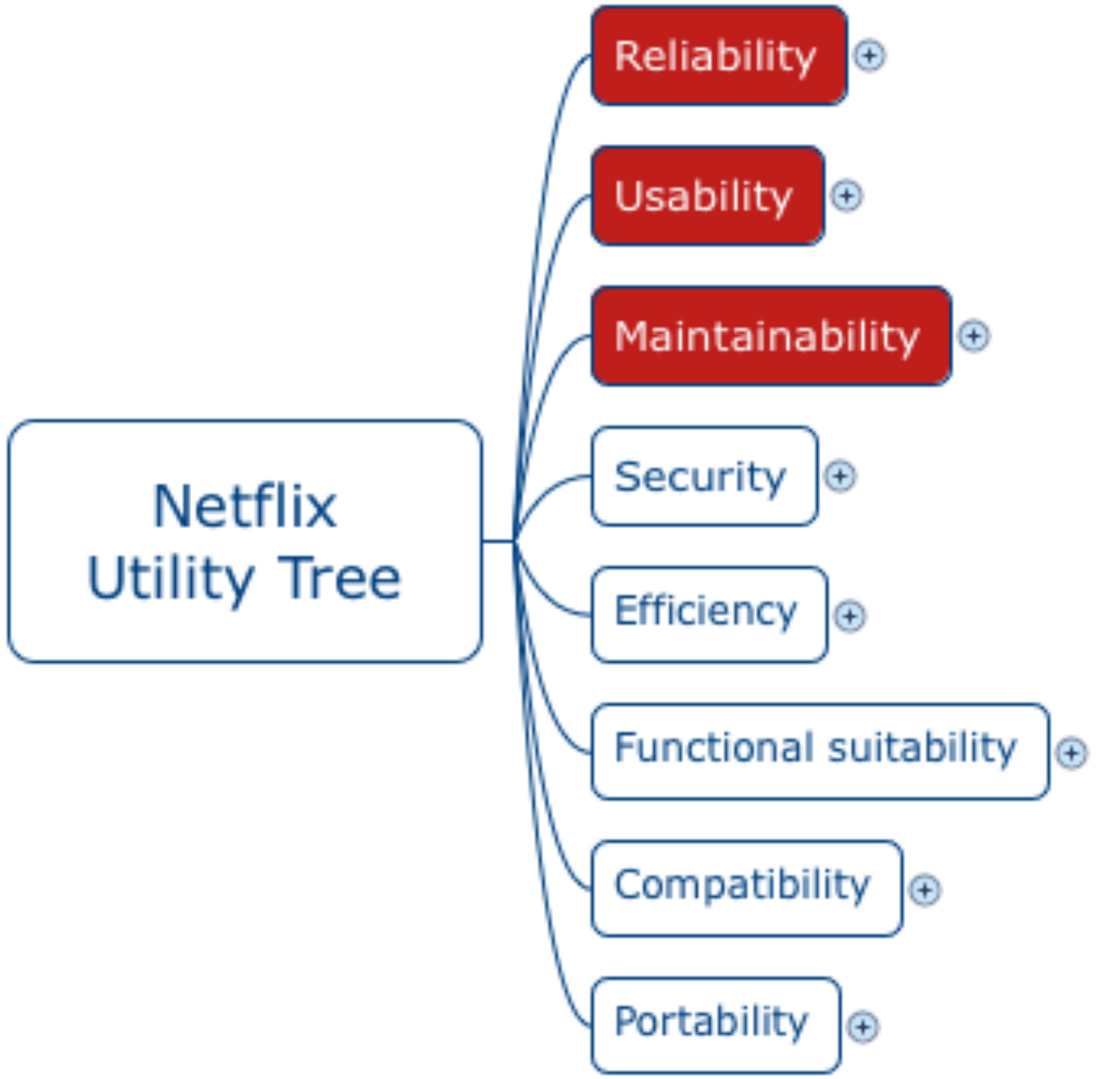

Auto Scaling Group (ASG)
old version
Live
but (partially)
inactive
Base
AMI
bake
.war
AMI
ASG
new version
push
build
deploy
instanciate
Push Changes in
Source Code Repository
automatic build makes
applcation package
Jenkins
Build AMI (Amazon Machine Image)
deploy application package
Aminator
Register AMI
Instantiate AMI (-> ASG)
Asgard
Service registers itsself
Eureka
Monitoring of the new ASG
Servo, Hystrix, Turbine

Continuous Delivery, Canaries, …

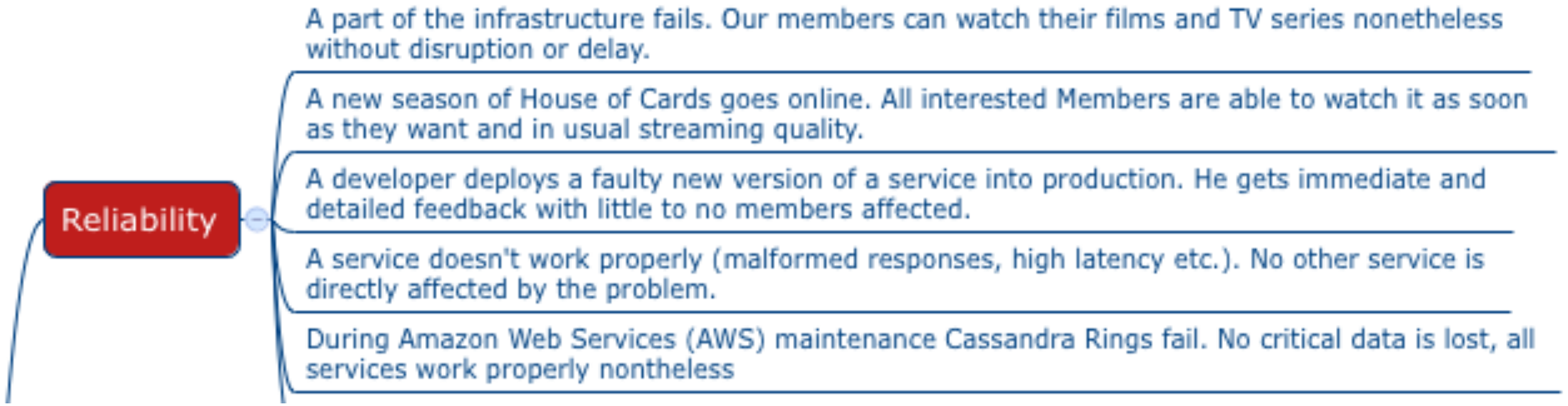
- **Robustness**: Fast Rollback (or Fallback essentially)
- **Testability**: Production is a realistic test environment and: cheaper than a separate testing environment
- **Know How**: Individual developers are decoupled from central settings and configurations

- **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

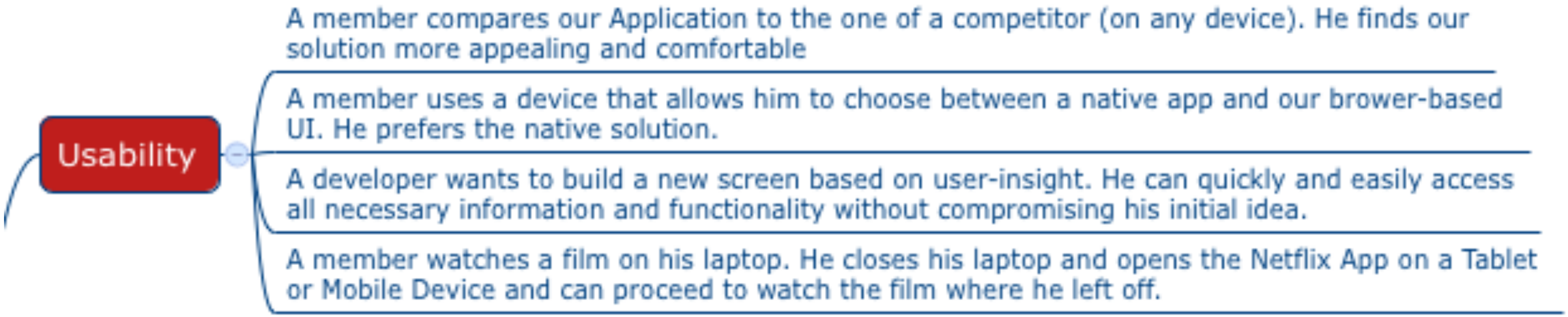
In summary – Quality Requirements



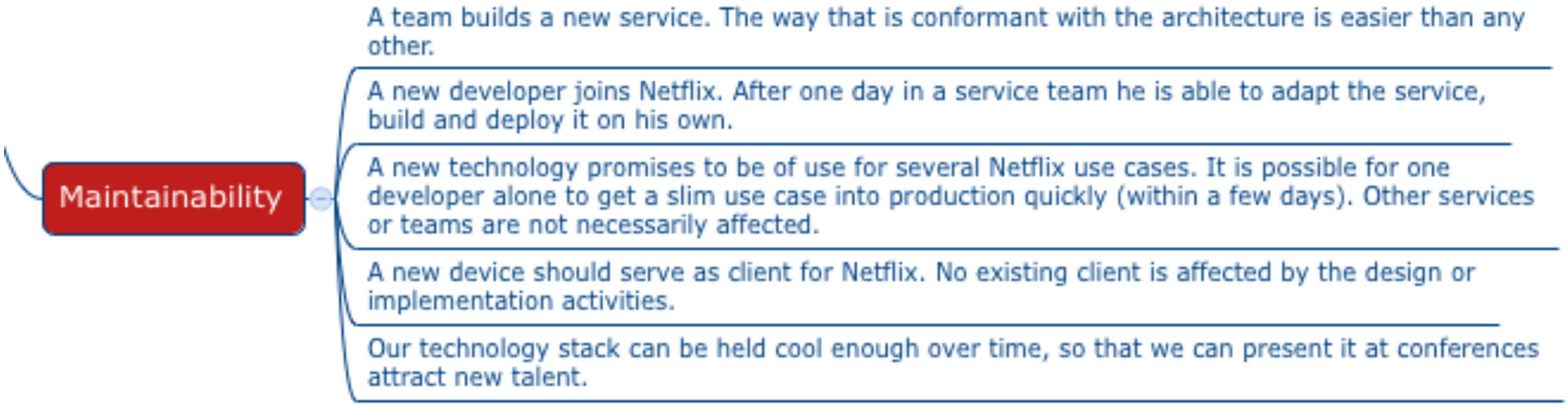
Reliability Scenarios



Usability Scenarios



Maintainability Scenarios



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**...

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

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

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

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

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

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

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

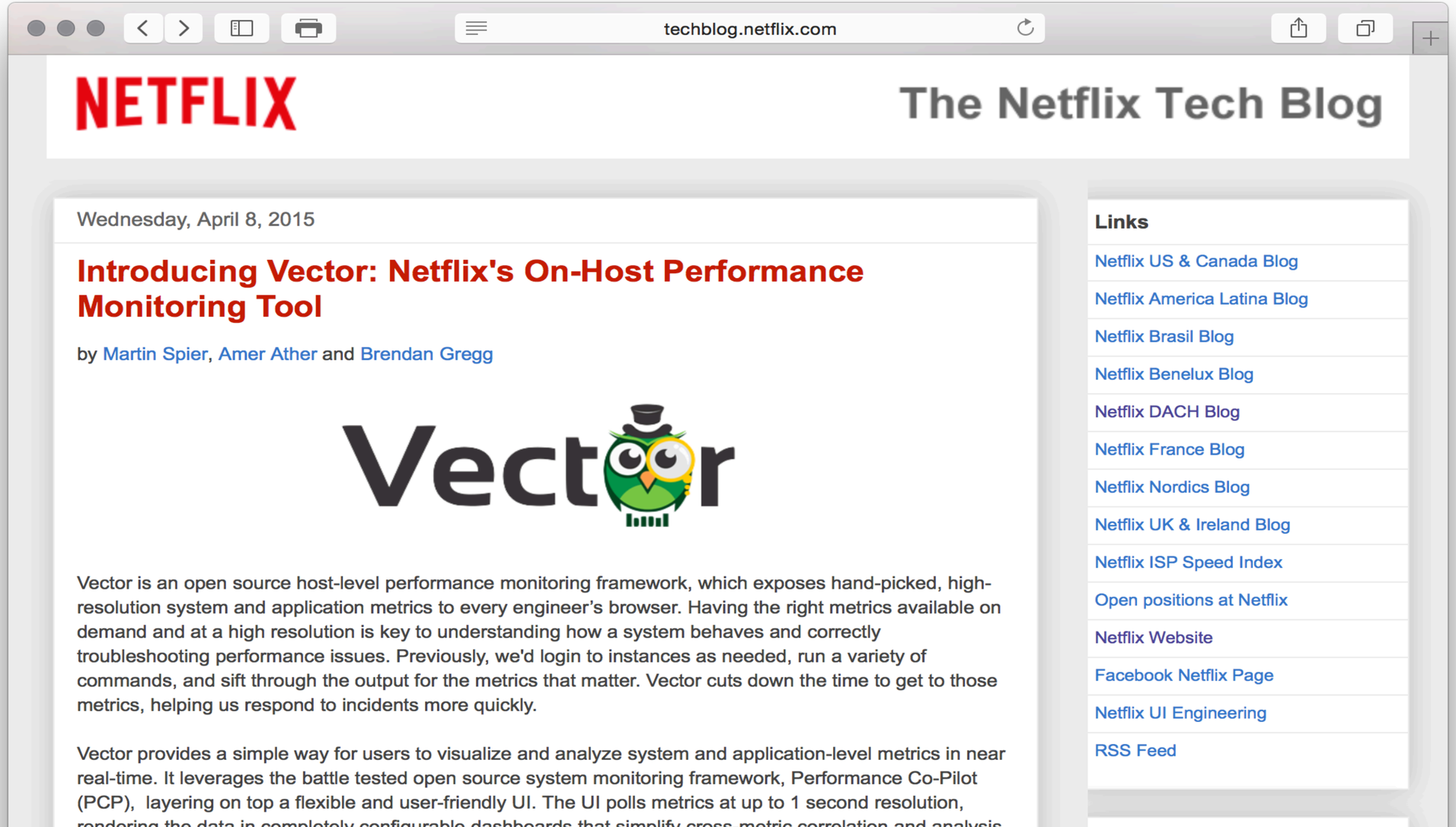
…**redundant development** and low reuse possibilities are perfectly OK.

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.

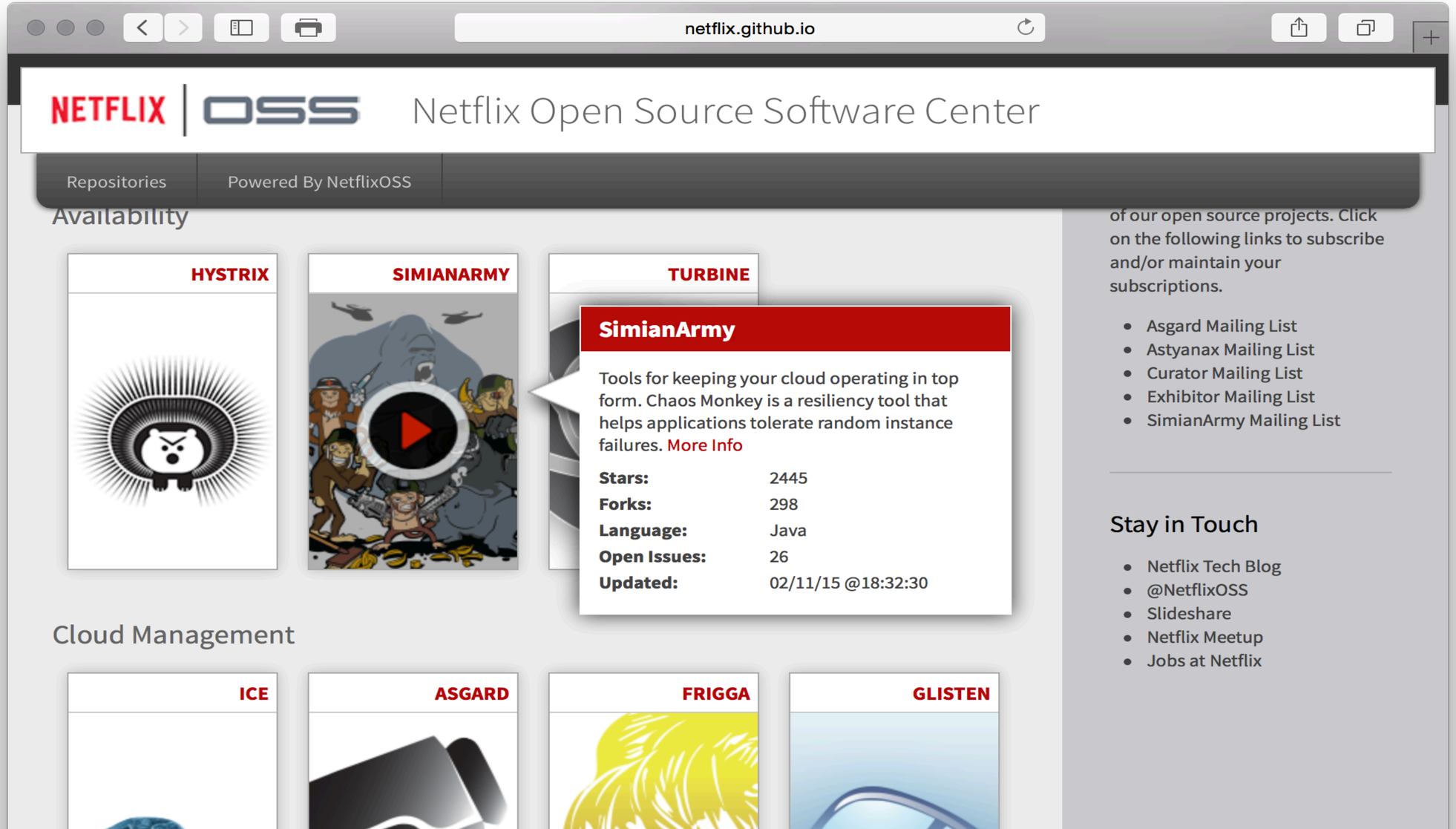
Netflix Tech Blog

➔ http://techblog.netflix.com



Netflix Open Source Software

➔ http://netflix.github.io



Thank You.

Questions are welcome!

 stefan.toth@embarc.de

 @st_toth

DOWNLOAD SLIDES:
http://www.embarc.de/blog/

embarc
Software Consulting GmbH