Foundations of
Software Engineering

Architecting for Scale

Microservices at Netflix-Uber-Spotify Scale

Pooyan Jamshidi

Carnegie Mellon University

Learning Goals

- Understand the value of microservices for building complex applications that need to operate at higher scale
- Identify requirements that derive companies to migrate to microservices (contrast of requirements between companies)
- Understand strategies for reliability of microservice architecture either at micro-level using design patterns or at a larger level
- Build agile team structure that enable large-scale companies to move fast (organizational challenges)
- Understand challenges that Netflix-Uber-Spotify faced in realizing microservice based applications

Disclaimer

- I used materials from
  - Netflix blog
  - Spotify, Uber and Netflix's architects GOTO talks
  - And some other sources referenced in the slides
- I'm a postdoc in Christian's group
  - Software Engineering + Machine Learning
- I worked as a software practitioners for 7 years
  - Pre-PhD
  - 4 years as a developer
  - 3 years as an architect
  - Involved in migration to cloud and microservices

Question

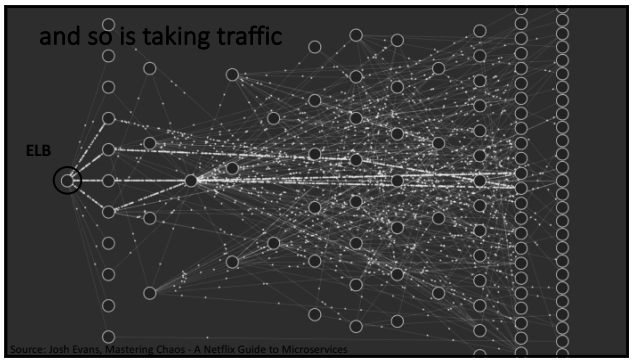
What is the most interesting aspect that you have learned from the Netflix talk?

Tradeoff in software architecture

- Everything is tradeoff
- Try to make them intentionally

Organ Systems

- Each organ has a purpose
- Organs form systems
- Systems form an organism

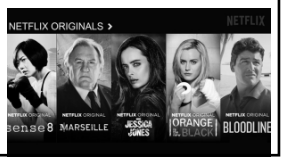


NETFLIX

Largest Internet TV network

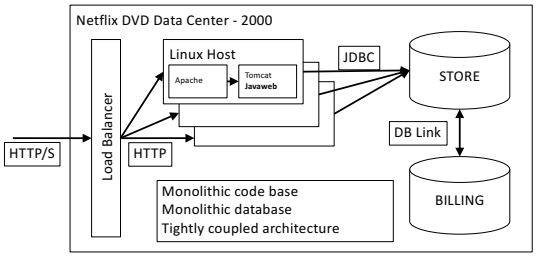
86 million members
~190 countries, 10s of languages
125m hours content per day

Microservices on AWS

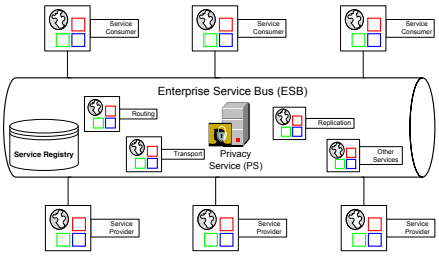


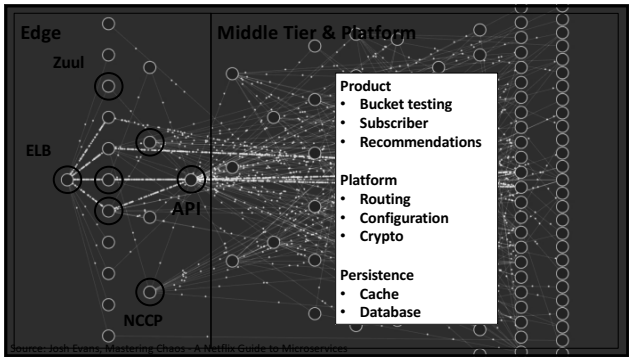
Source: Josh Evans, Mastering Chaos - A Netflix Guide to Microservices

What microservices are not

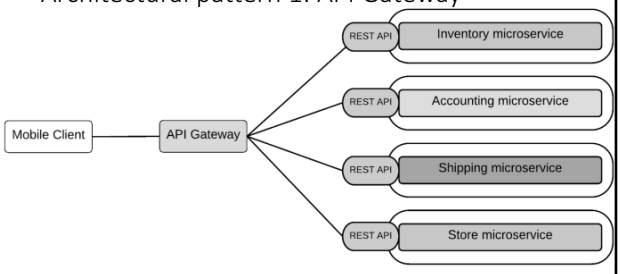


What microservices are not





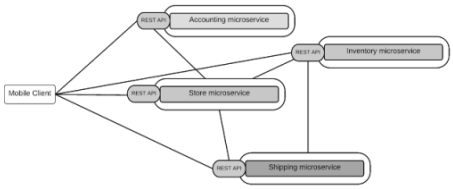
Architectural pattern 1: API Gateway



Source: Kasun Indrasiri, Microservices in Practice: From Architecture to Deployment

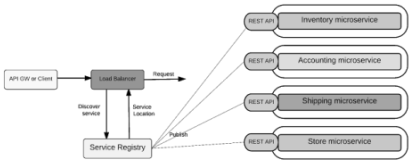
12

Architectural pattern 2: Inter-process Communication in a Microservices Architecture



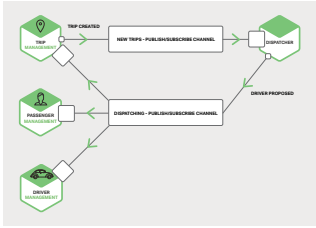
13

Architectural pattern 3: Service Discovery in a Microservices Architecture



14

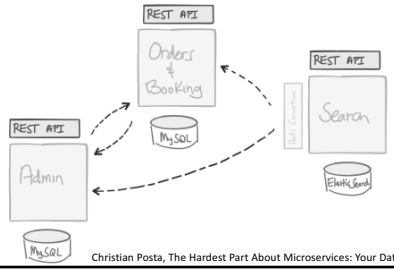
Architectural pattern 4: Event-Driven Data Management for Microservices



Chris Richadson, Microservices from Design to Deployment

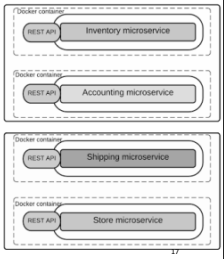
15

Architectural pattern 5: Decentralized Data Management



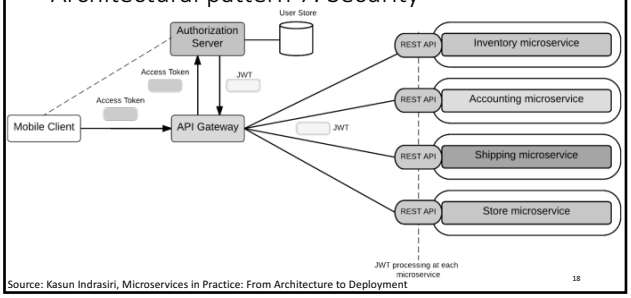
Christian Posta, The Hardest Part About Microservices: Your Data

Architectural pattern 6: Choosing a Microservices Deployment Strategy



17

Architectural pattern 7: Security

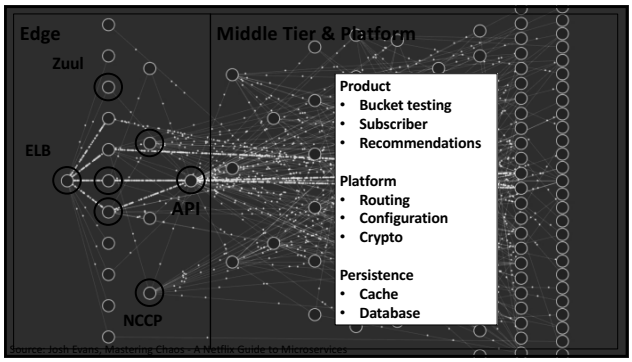


Source: Kasun Indrasiri, Microservices in Practice: From Architecture to Deployment

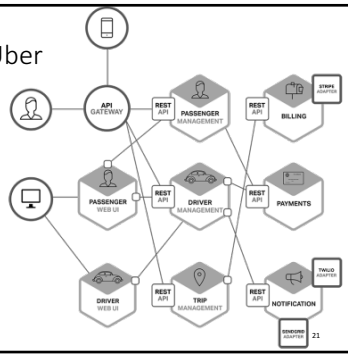
18

Question

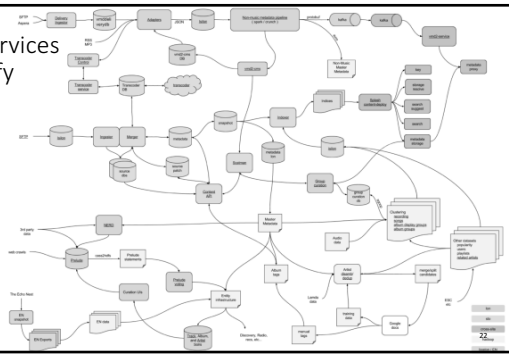
What architectural patterns you can identify within these architectures?



Microservices at Uber



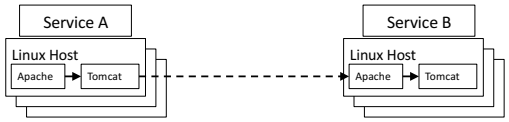
Microservices at Spotify



Question

Why reliability matters in microservices world?

Intra-service Requests

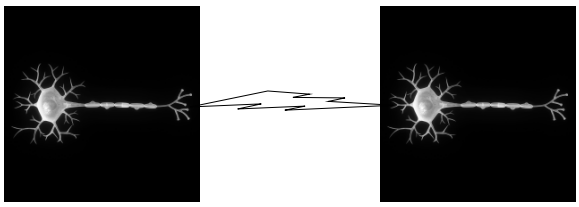


Network latency, congestion, failure

Logical or scaling failure

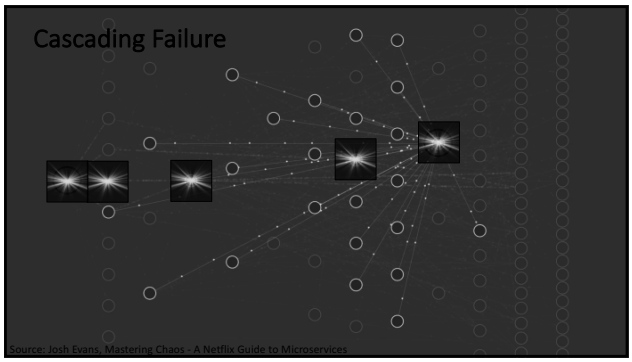
Source: Josh Evans, Mastering Chaos - A Netflix Guide to Microservices

Crossing the Chasm

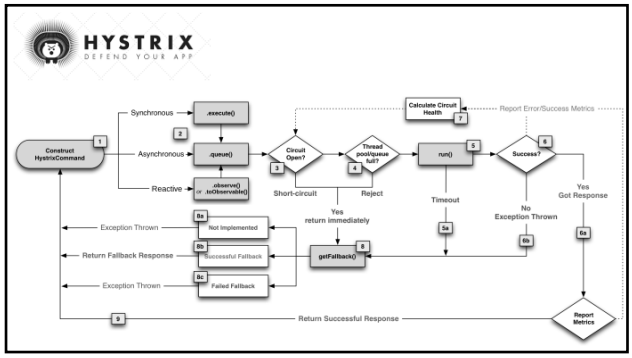


Source: Josh Evans, Mastering Chaos - A Netflix Guide to Microservices

Cascading Failure

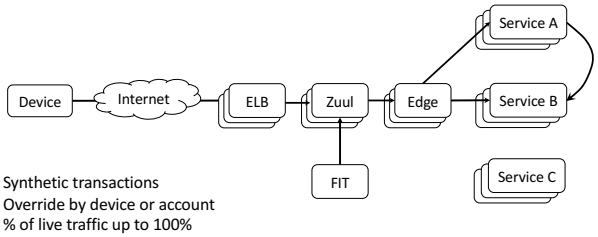


Source: Josh Evans, Mastering Chaos - A Netflix Guide to Microservices



Vaccination

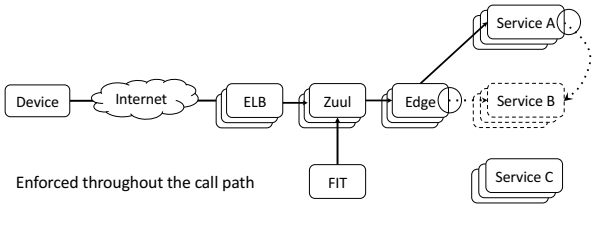
Fault Injection Testing (FIT)



Synthetic transactions
Override by device or account
% of live traffic up to 100%

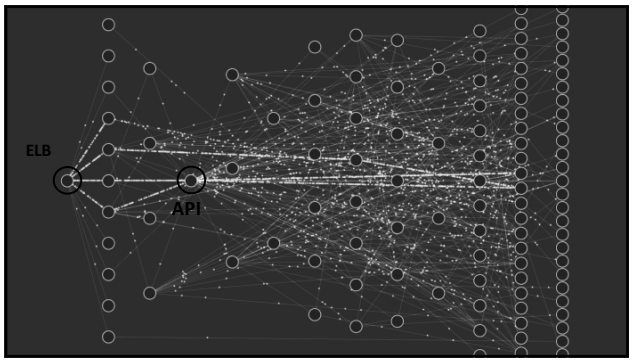
Source: Josh Evans, Mastering Chaos - A Netflix Guide to Microservices

Fault Injection Testing (FIT)

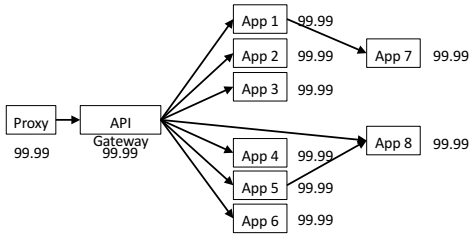


Enforced throughout the call path

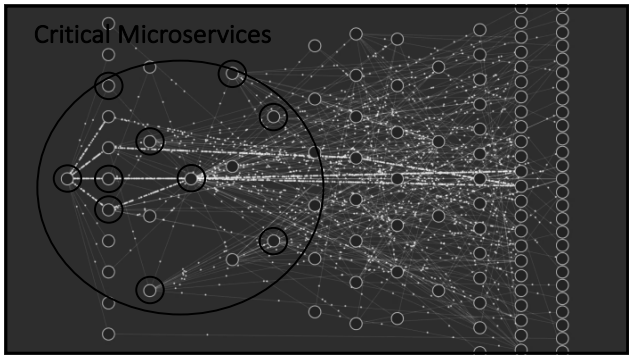
Source: Josh Evans, Mastering Chaos - A Netflix Guide to Microservices



Combinatorial Math



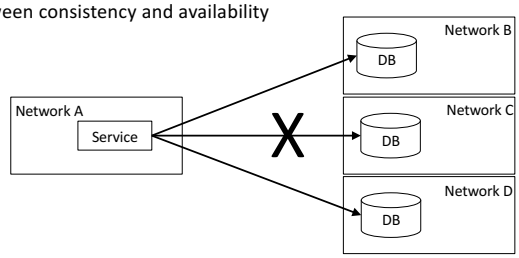
$99.99^{10} = 99.9$



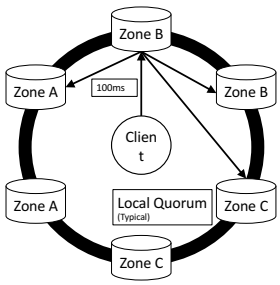
Persistence

CAP Theorem

In the presence of a network partition, you must choose between consistency and availability



Eventual Consistency



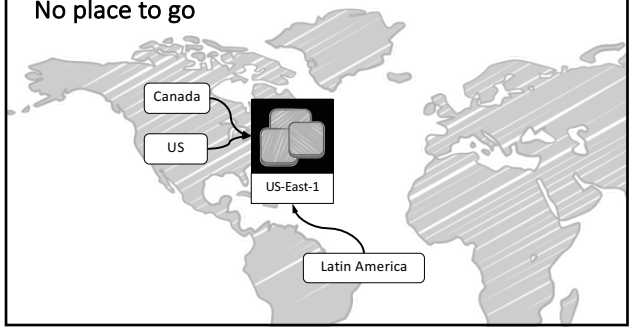
Infrastructure

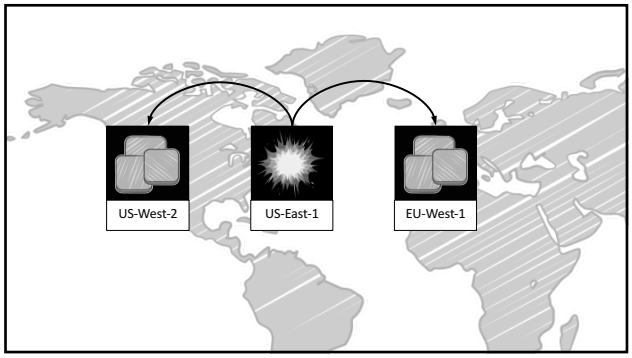
December 24th, 2012

Forbes / Tech

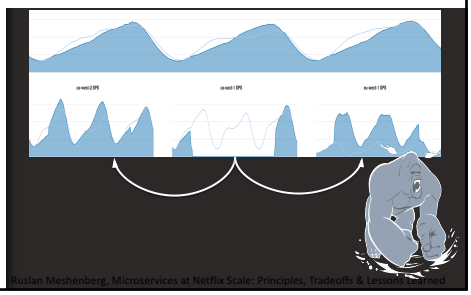
Amazon AWS Takes Down Netflix On Christmas Eve

No place to go

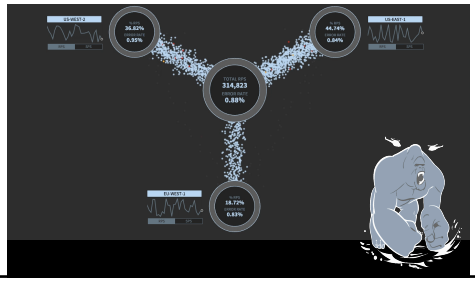




Regional failover



Regional failover

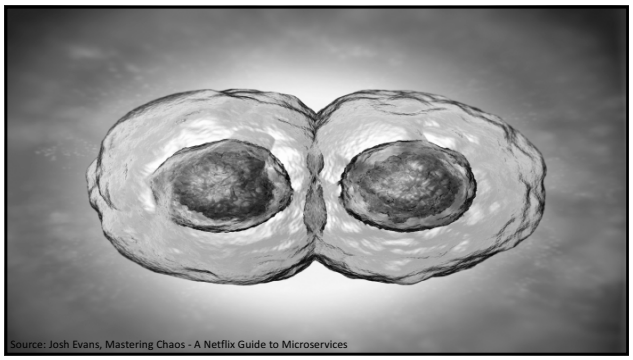


Question

What is a stateless service?

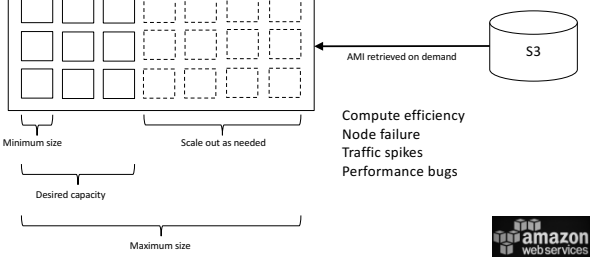
What is a stateless service?

- Not a cache or a database
- Frequently accessed metadata
- No instance affinity
- Loss a node is a non-event

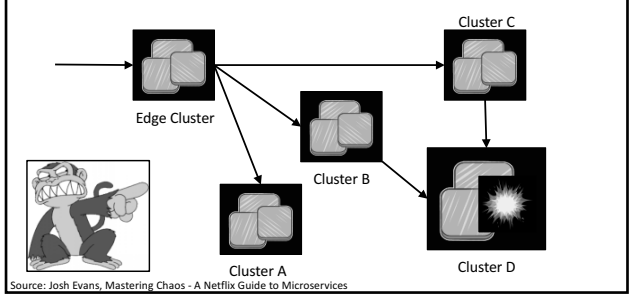


Source: Josh Evans, Mastering Chaos - A Netflix Guide to Microservices

Auto Scaling Groups



Surviving Instance Failure



Source: Josh Evans, Mastering Chaos - A Netflix Guide to Microservices

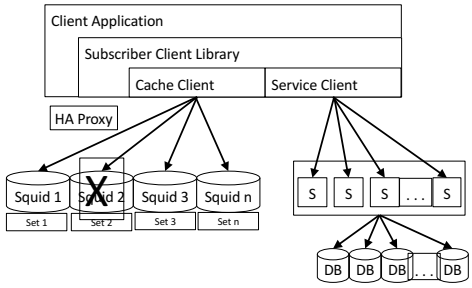
Question

What is a stateful service?

What is a stateless service?

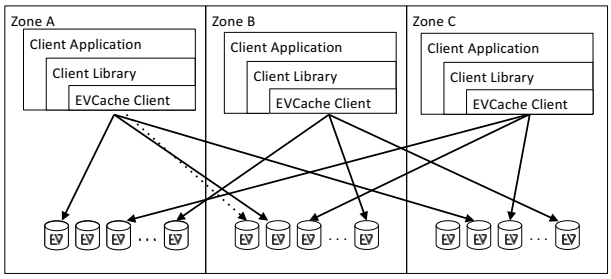
- Databases and caches
- Custom apps which hold data
- Loss of a node is a notable event

Dedicated Shards – An Antipattern

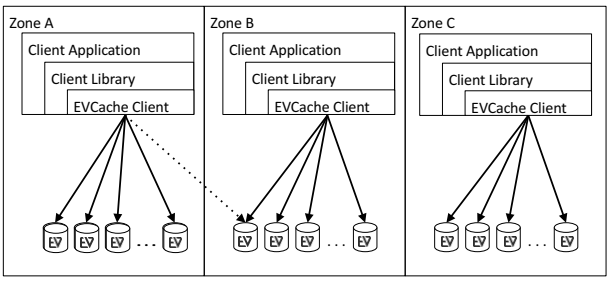


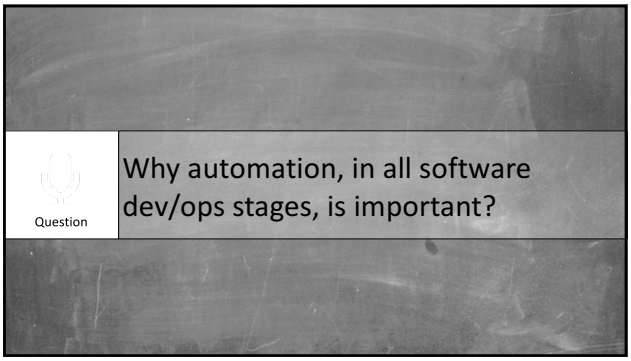
Redundancy is fundamental

EVCache Writes



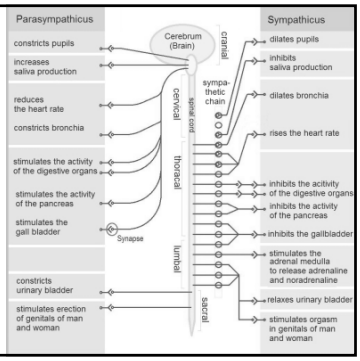
EVCache Reads



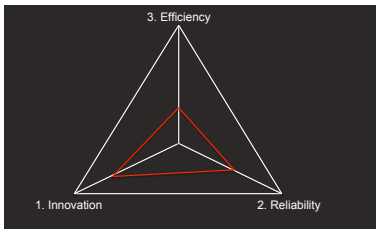


Autonomic Nervous System

You don't have to think about digestion or breathing

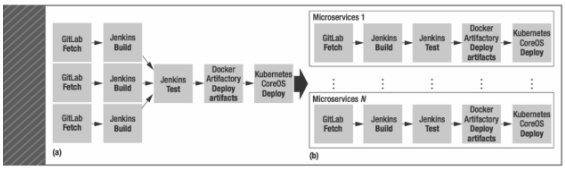


Priorities

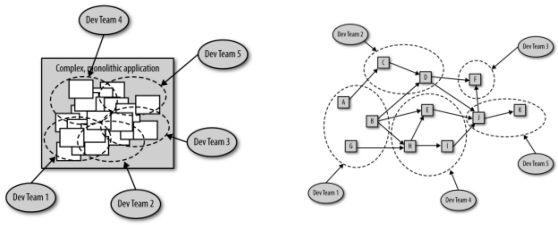


Ruslan Meshenberg, Microservices at Netflix Scale: Principles, Tradeoffs & Lessons Learned

Innovation: Tight coupling doesn't work



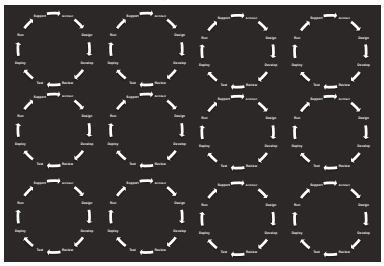
Monolithic vs microservice-based applications: Interdependent vs Independent teams



Lee Atchison, Architecting for Scale: High Availability for Your Growing Applications

59

End-to-end ownership



Ruslan Meshenberg, Microservices at Netflix Scale: Principles, Tradeoffs & Lessons Learned

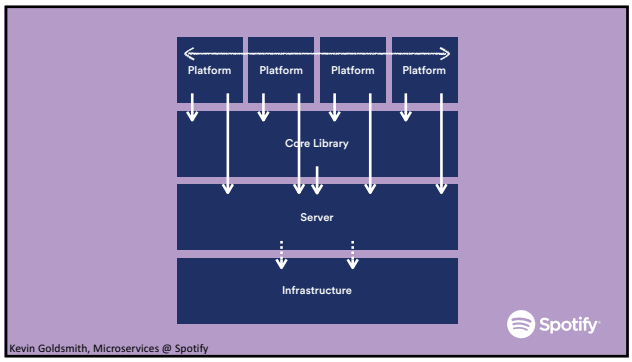
60

A product like this

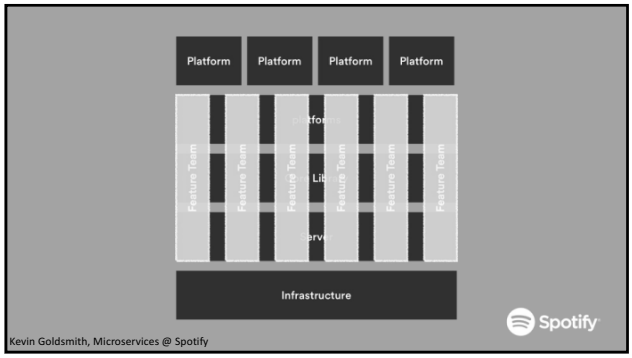
- 75+ Million Monthly Active Users
- 58 Countries
- >20,000 songs added per day
- >2B playlists
- Incredibly complex business rules
- Lots of competition

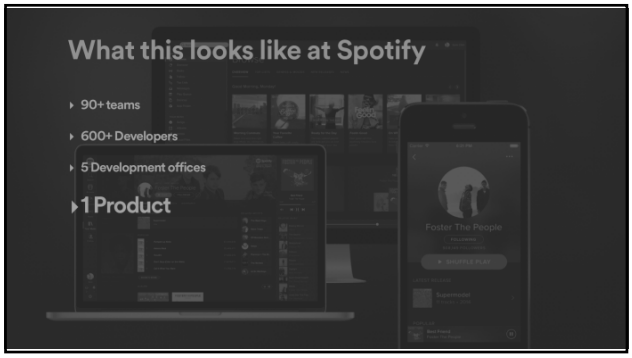
MUSIC PANDORA YouTube DEEZER

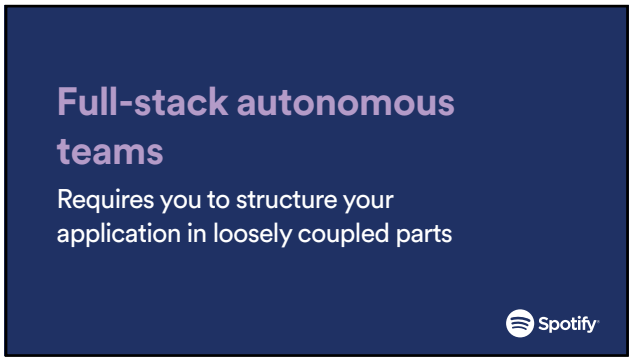
Kevin Goldsmith, Microservices @ Spotify

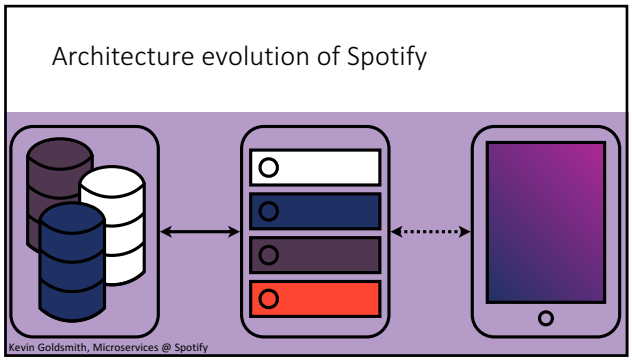

Platform
Platform
Platform
Platform
Core Library
Server
Infrastructure
Spotify
Kevin Goldsmith, Microservices @ Spotify


Challenges
Synchronization
Client UX implementation depends on
Core Library Implementation depends on
Server Implementation depends on
Infrastructure Implementation
Spotify
Kevin Goldsmith, Microservices @ Spotify

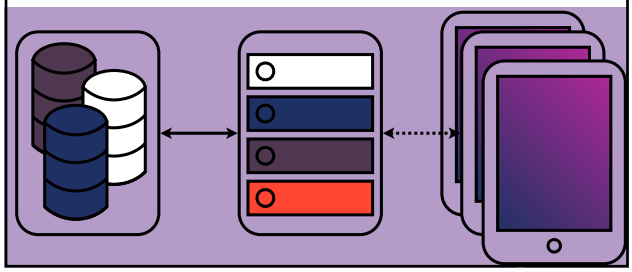

Platform
Platform
Platform
Platform
Feature Team
Infrastructure
Spotify
Kevin Goldsmith, Microservices @ Spotify

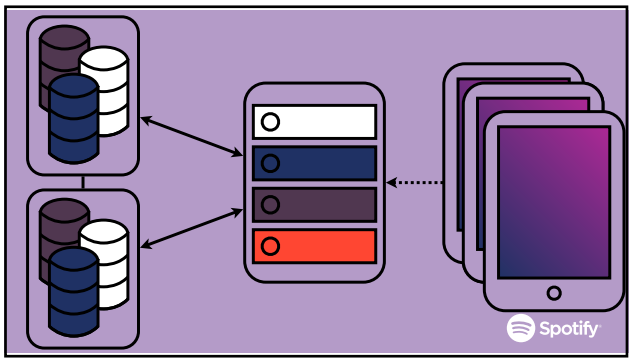

What this looks like at Spotify
90+ teams
600+ Developers
5 Development offices
1 Product

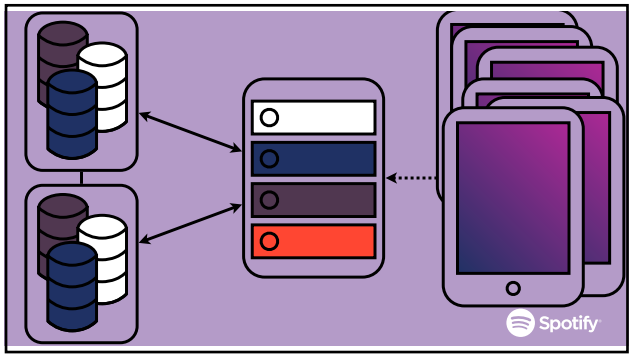

Full-stack autonomous teams
Requires you to structure your application in loosely coupled parts
Spotify

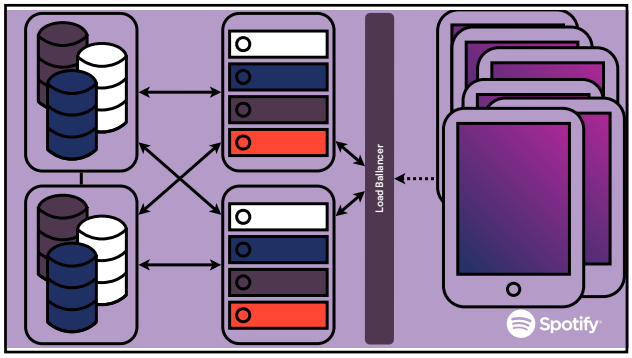

Architecture evolution of Spotify
Kevin Goldsmith, Microservices @ Spotify

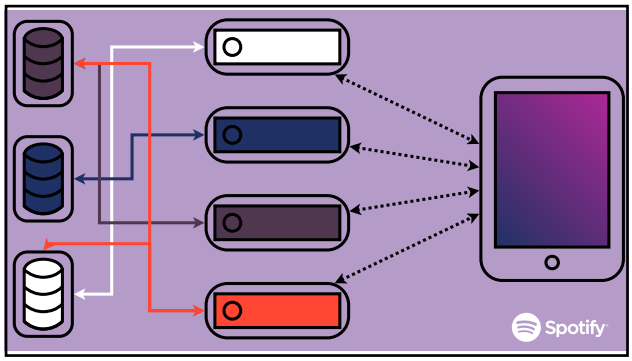
Architecture evolution of Spotify











Microservices: Yay!

- Easier to Scale
- Easier to test
- Easier to deploy
- Easier to monitor
- They can versioned independently

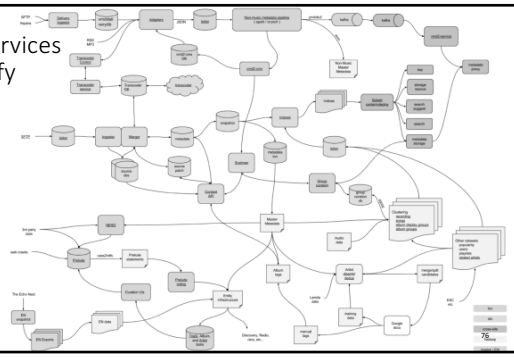
Microservices: Boo!

- Monitoring lots of services
- Documentations
- Increased latency

What does this look like at Spotify?

- 810 active services
- ~10 Systems per squad
- ~1.7 Systems per person with access to production servers
- ~1.15 Systems per member of Technology

Microservices at Spotify

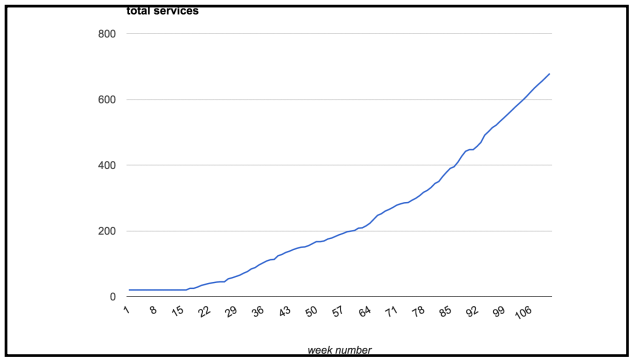


Uber

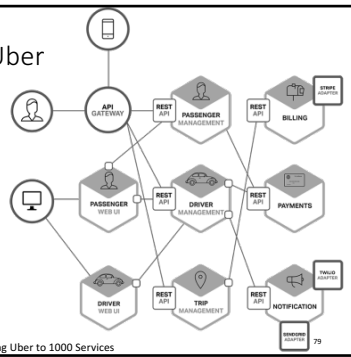
As of April 2016:

Uber Cities Worldwide: 400+
Countries: 70
Employees: 6,000+

Matt Ranney, What I Wish I Had Known Before Scaling Uber to 1000 Services



Microservices at Uber



Matt Ranney, What I Wish I Had Known Before Scaling Uber to 1000 Services

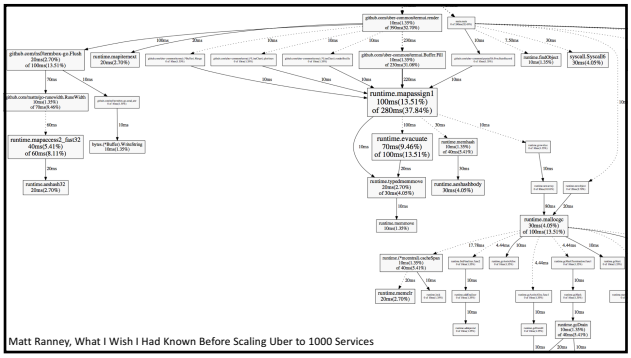
pre-history	PHP (outsourced)
Dispatch	Node.JS, moving Go
Core Services	Python, moving to Go
Maps	Python and Java
Data	Python and Java
Metrics	Go

Matt Ranney, What I Wish I Had Known Before Scaling Uber to 1000 Services

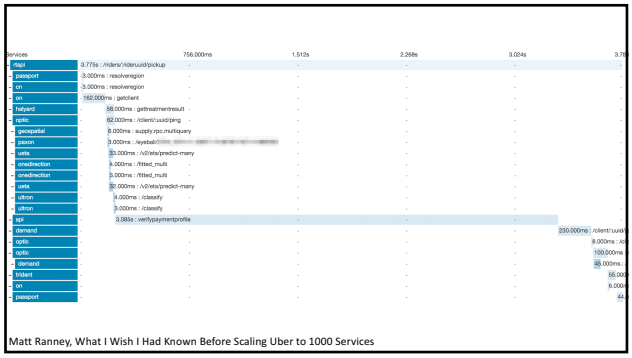
LANGUAGES

Hard to share code
Hard to move between teams
WIWIK: Fragments the culture

Matt Ranney, What I Wish I Had Known Before Scaling Uber to 1000 Services



Matt Ranney, What I Wish I Had Known Before Scaling Uber to 1000 Services



Matt Ranney, What I Wish I Had Known Before Scaling Uber to 1000 Services

Summary

- Microservices may be a right solution for building complex applications that need to operate at higher scale
- Tradeoffs that companies made to migrate to microservices (contrast of requirements between companies)
- Making reliable microservice architecture requires strategies to deal with failure either at micro-level or at a larger level
- Microservices architecture help to build agile team structure that enable large scale companies to move fast (organizational challenges)