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 architexts 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

Tradeoff in software architecture

- Everything is tradeoff
- Try to make them intentionally

What is the most interesting aspect that you have learned from the Netflix talk?
and so is taking traffic

ELB

Netflix DVD Data Center - 2000

<table>
<thead>
<tr>
<th>Product</th>
<th>Platform</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Bucket testing</td>
<td>• Routing</td>
</tr>
<tr>
<td>• Subscriber</td>
<td>• Configuration</td>
</tr>
<tr>
<td>• Recommendations</td>
<td>• Crypto</td>
</tr>
<tr>
<td></td>
<td>• Persistence</td>
</tr>
<tr>
<td></td>
<td>• Cache</td>
</tr>
<tr>
<td></td>
<td>• Database</td>
</tr>
</tbody>
</table>

What microservices are not

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

Netflix DVD Data Center - 2000

Microservices on AWS

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

Architectural pattern 1: API Gateway

Source: Kasun Indrasiri, Microservices in Practice: From Architecture to Deployment
Architectural pattern 2: Inter-process Communication in a Microservices Architecture

Architectural pattern 3: Service Discovery in a Microservices Architecture

Architectural pattern 4: Event-Driven Data Management for Microservices

Architectural pattern 5: Decentralized Data Management

Architectural pattern 6: Choosing a Microservices Deployment Strategy

Architectural pattern 7: Security
What architectural patterns you can identify within these architectures?

Microservices at Uber

Why reliability matters in microservices world?

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

Cascading Failure

Vaccination

Fault Injection Testing (FIT)

Fault Injection Testing (FIT)
Critical Microservices

Persistence

**CAP Theorem**

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

---

**Eventual Consistency**

---
No place to go

Can we avoid this? Yes, if you have a

Regional failover

Infrastructure
What is a stateless service?

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

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

Redundancy is fundamental

Why automation, in all software dev/ops stages, is important?
**Autonomic Nervous System**

You don’t have to think about digestion or breathing.

---

**Priorities**

- 1. Innovation
- 2. Reliability
- 3. Efficiency

---

**Innovation: Tight coupling doesn’t work**

Monolithic vs microservice-based applications: Interdependent vs Independent teams

---

**End-to-end ownership**

---

*Ruslan Meshenberg, Microservices at Netflix Scale: Principles, Tradeoffs & Lessons learned*
Kevin Goldsmith, Microservices @ Spotify

Challenges

Synchronization

Client UX implementation
Core Library Implementation
Server Implementation
Infrastructure Implementation

Kevin Goldsmith, Microservices @ Spotify

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

Kevin Goldsmith, Microservices @ Spotify

Architecture evolution of 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 services
- ~1.15 Systems per member of Technology

As of April 2016:

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