Microservices

HAN minor

Tom van den Berg

Lead software engineer

Tom.vandenberg@infosupport.com
Tom.vandenberg@ns.nl

tom171296

@TomB_171296

www.blognet.tech

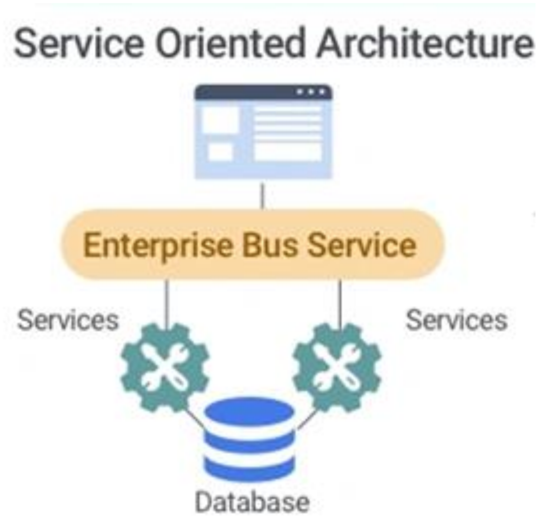
Web-scale architecture

Introduction into web-scale architecture

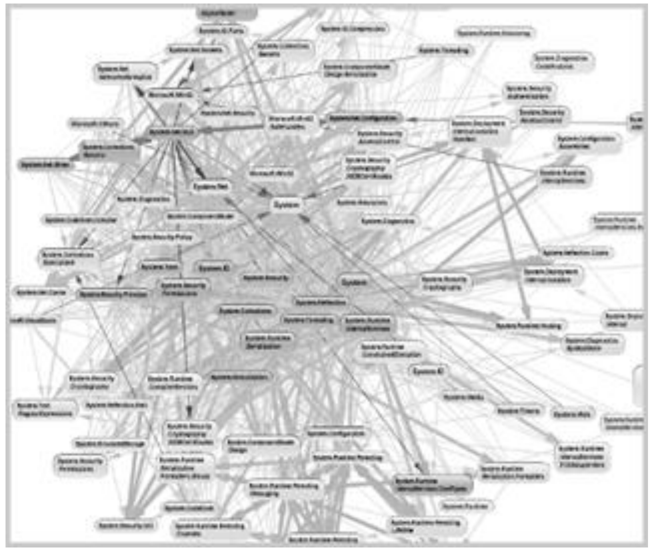
Traditional architecture

Monolith

Soa



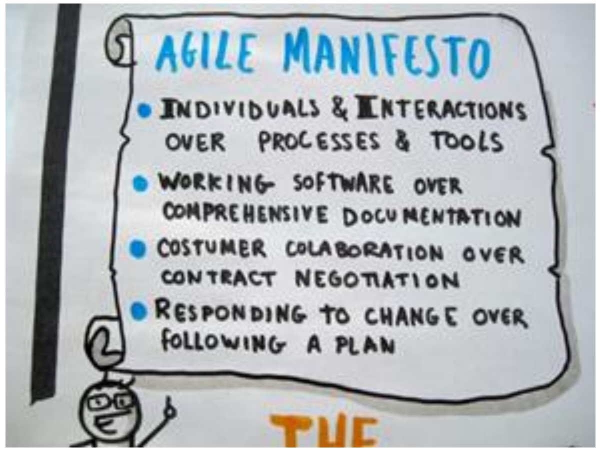
Big ball of mud



- Bad maintainability
  - Tight coupling
  - Changes resonate throughout the entire application landscape
- Bad scalability

- Low availability
  - Often offline for upgrades or maintenance
  - Services / system coupled @ runtime
- Long release-cycles

How can we change this?


AGILE MANIFESTO

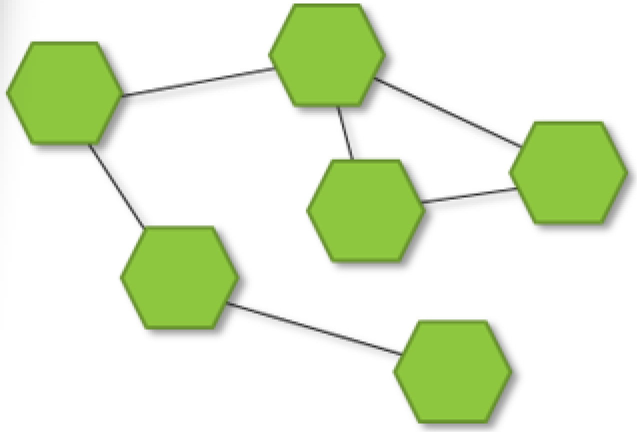
- INDIVIDUALS & INTERACTIONS OVER PROCESSES & TOOLS
- WORKING SOFTWARE OVER COMPREHENSIVE DOCUMENTATION
- COSTUMER COLABORATION OVER CONTRACT NEGOTIATION
- RESPONDING TO CHANGE OVER FOLLOWING A PLAN

THE

Agile Approach

The Addison Wesley Signature Series
Continuous Delivery
Reliable Software Releases through Build, Test, and Deployment Automation
Jez Humble
David Farley
The DevOps Handbook
How to create world-class agility, reliability, & security in technology organizations
Gene Kim, Jez Humble, Patrick Debois, & John Willis

Continuous Delivery Automation Dev Ops

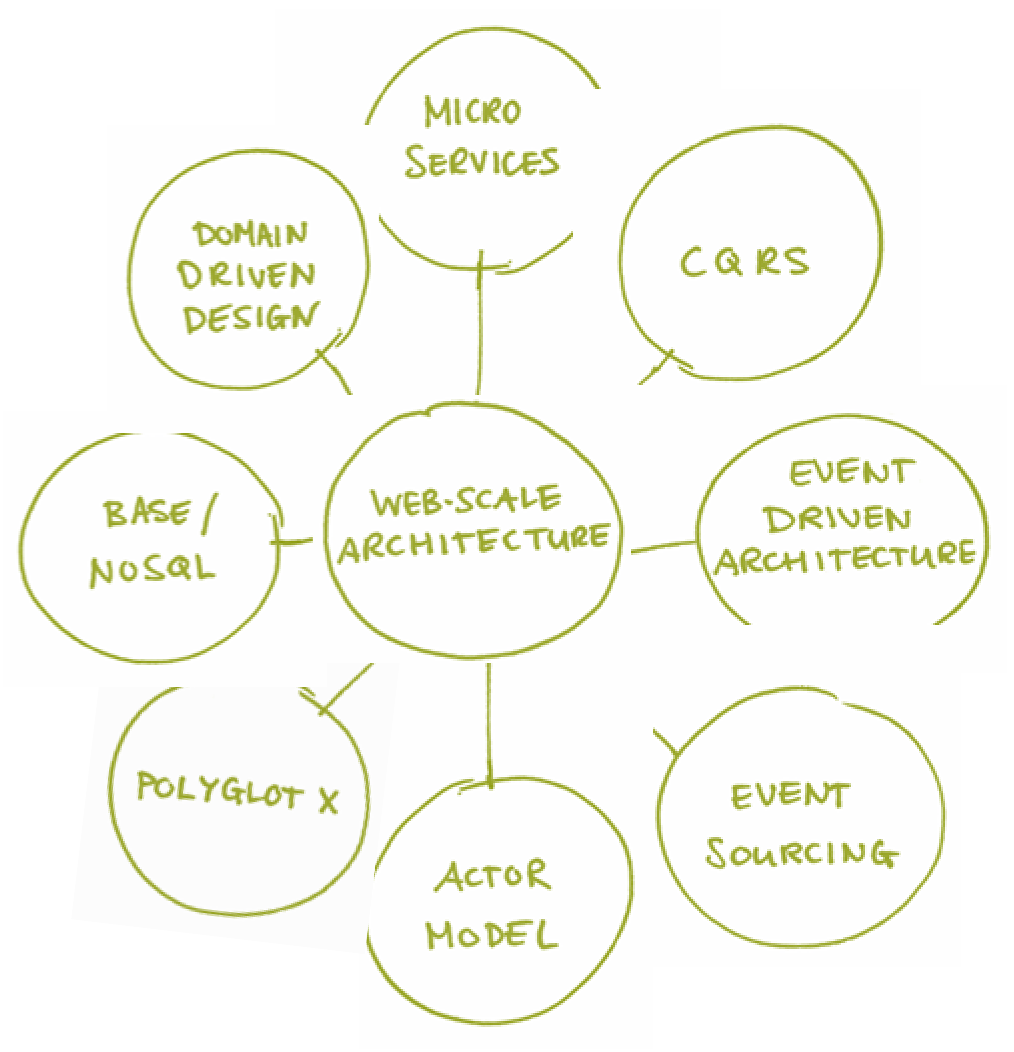


Modern Web-scale Architecture

Introduction of the term “Web-scale”

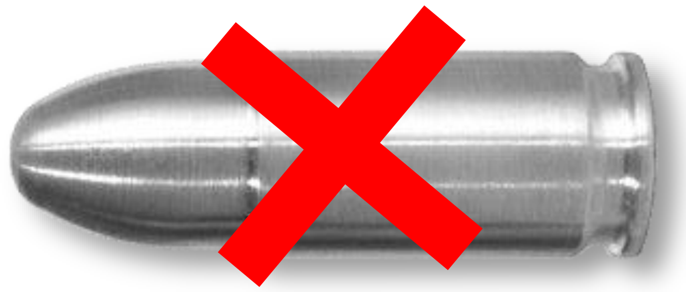
In a research note that was published yesterday, Gartner introduced the term “web-scale IT.” What is web-scale IT? It’s our effort to describe all of the things happening at large cloud services firms such as Google, Amazon, Rackspace, Netflix, Facebook, etc., that enables them to achieve extreme levels of service delivery as compared to many of their enterprise counterparts.

In addition, while the term “scale” usually refers to size, we’re not suggesting that only large enterprises can benefit. Another scale “attribute” is speed and so we’re stating that even smaller firms (or departments within larger IT organizations) can still find benefit to a web-scale IT approach. Agility has no size correlation so even more modestly-sized organizations can achieve some of the capabilities of an Amazon, etc., provided that they are willing to question conventional wisdom where needed.



Architecture pattern based on small, specialized and autonomous services that communicate using events. This pattern enables agile teams to develop services autonomously and release frequently.

Disclaimer!



- KISS, common sense and software craftsmanship are still the most important tools of an engineer!
- Choose the best fit-for-purpose solution and architecture style based on complexity and risks!
- Every decision is a trade-off!

Microservices

Microservices

- “Small” autonomous services that cooperate
  - Designed around business domains /capabilities (DDD bounded contexts)
  - Simple to scale-out
  - High cohesion / low coupling
- A Microservice is specialized in 1 thing
  - Single responsibility principle

Traditionally (SOA)

InsuranceService

+ RegisterPolicy
+ GetPolicy
+ RegisterCustomer
+ GetCustomer
+ MoveCustomer
+ HandlePayment
+ GetBalance
+ ChangeRiskProfile

Microservices

PolicyService

+ RegisterPolicy
+ GetPolicy

InvestmentService

+ ChangeRiskProfile

FinanceService

+ HandlePayment
+ GetBalance

CustomerService

+ RegisterCustomer
+ GetCustomer

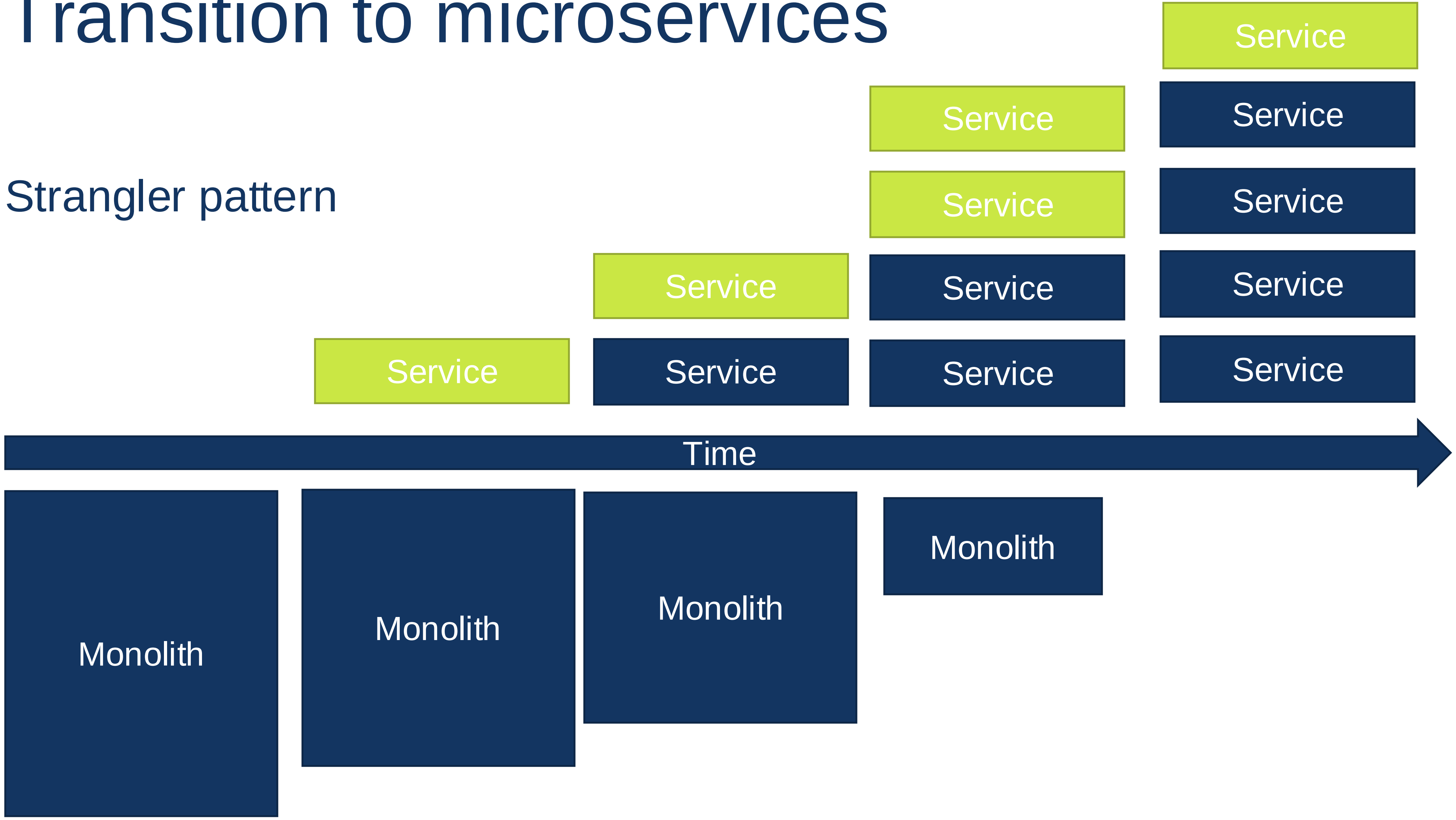
Exercise 1

- Make groups of 5;
- Get your team some sticky notes and a4 paper;
- Design a microservice architecture.

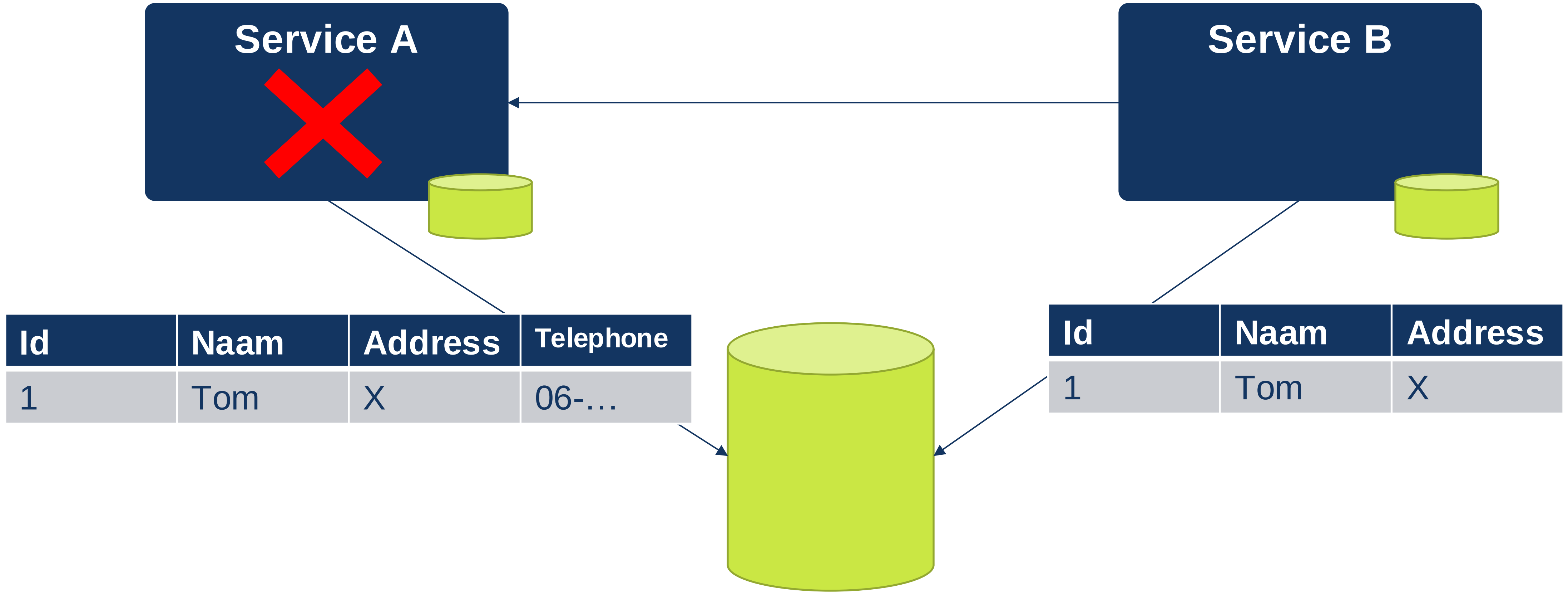
- Timebox:
  - 20 min design
  - 10 min discussion and questions

Transition to microservices

Strangler pattern



Data management



Id	Naam	Address	Telephone
1	Tom	X	06-…

Id	Naam	Address
1	Tom	X

Microservices

- Data-duplication is often used to make each Microservice truly independent @runtime
  - No this is NOT evil (if done right!)
  - Only duplicate data necessary for a service to operate
  - Preferably only a read-model built from events

- There's still only 1 service that owns (and changes!) the data (system of record)

Microservices

- communicate using “lightweight” protocols
  - HTTP (Rest API + JSON) / TCP + ProtoBuf / Own implementation
  - Choose between *open* or *fast*

- Primarily asynchronous communication
  - Using a message broker can increase autonomy

Microservice principals

Culture of automation

Modelled around business domain

(DDD)

Decentralize all things

(freedom for the devops Teams)

Hide implementation details

Deploy independently

(docker)

Highly observable

Isolate failure

(Design for Failure)