AARHUS UNIVERSITET

Microservices and DevOps

Scalable Microservices

Microservices – the Extended Edition

Henrik Bærbak Christensen

AARHUS UNIVERSITET

Software Reuse

- *If only we could build software from reusable components*

- A big struggle throughout the history of computing

AARHUS UNIVERSITET

Software Reuse

- During the early 90'ies the Lego bricks metaphor was often used for object/component reuse

- However, the golden future of a world wide component market place sort of deflated

Clemens Szyperski (2002). Component Software: Beyond Object-Oriented Programming.

Software Reuse

- In the 00' the web exploded and allowed a different deployment model
  - Component based reuse:
    - A **static / module viewpoint** - get a DLL/jar file
    - **"libraries" in [Lewis et al. 2014]**
      - Linked in, and called by in-memory function calls
  - *Service oriented architecture:*
    - A **dynamic / C&C viewpoint** - connect to a service
    - **"services" in [Lewis et al. 2014]**
      - Out-of-process, communicate using RPC or web service

SOA

- Wikipedia
  - A *service-oriented architecture (SOA)* is an architectural pattern in computer software design in which application components provide *services* to other components via a *communications protocol, typically over a network*. The principles of service-orientation are independent of any vendor, product or technology.
- MacKenzie et al., 2006
  - is a paradigm for organizing and utilizing *distributed capabilities* that may be under the *control of different ownership domains*
  - provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations

AARHUS UNIVERSITET

Microservices

- The next step? A successful step?
- Wikipedia (2018)

> A microservice is a software development technique—a variant of the service-oriented architecture (SOA) architectural style that structures an application as a collection of loosely coupled services. In a microservices architecture, services are fine-grained and the protocols are lightweight. The benefit of decomposing an application into different smaller services is that it improves modularity. This makes the application easier to understand, develop, test, and become more resilient to architecture erosion.[1] It parallelizes development by enabling small autonomous teams to develop, deploy and scale their respective services independently.[2] It also allows the architecture of an individual service to emerge through continuous refactoring.[3] Microservices-based architectures enable continuous delivery and deployment.[1][4]

- Keywords
  - *Loosely coupled, fine-grained, lightweight protocols, autonomous teams, independent deployment and scaling, continuous delivery.*

So – Our Take

- We will look at two takes on the concept...
  - James Lewis and Martin Fowler
  - Sam Newman
- And compare ☺

AARHUS UNIVERSITET

Lewis and Fowler

AARHUS UNIVERSITET

Micro Services

- The next step? A successful step?

Microservices

a definition of this new architectural term

The term "Microservice Architecture" has sprung up over the last few years to describe a particular way of designing software applications as suites of independently deployable services. While there is no precise definition of this architectural style, there are certain common characteristics around organization around business capability, automated deployment, intelligence in the endpoints, and decentralized control of languages and data.

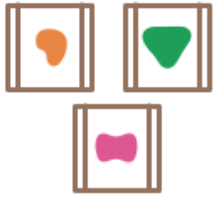
In short, the microservice architectural style is an approach to developing a single application as a **suite of small services**, each **running in its own process** and communicating with lightweight mechanisms, often an HTTP resource API. These services are **built around business capabilities** and **independently deployable** by fully automated deployment machinery. There is a **bare minimum of centralized management** of these services, which may be written in different programming languages and use different data storage technologies.

-- James Lewis and Martin Fowler (2014)

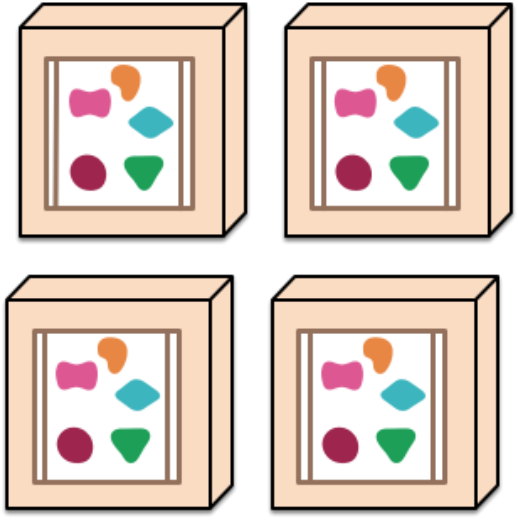
Overview

A monolithic application puts all its functionality into a single process...

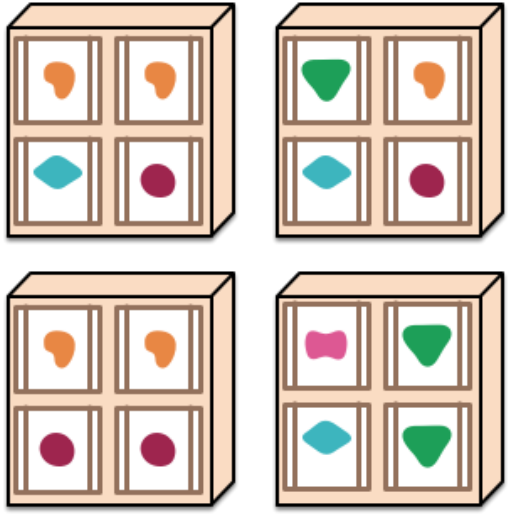
A microservices architecture puts each element of functionality into a separate service...



... and scales by replicating the monolith on multiple servers

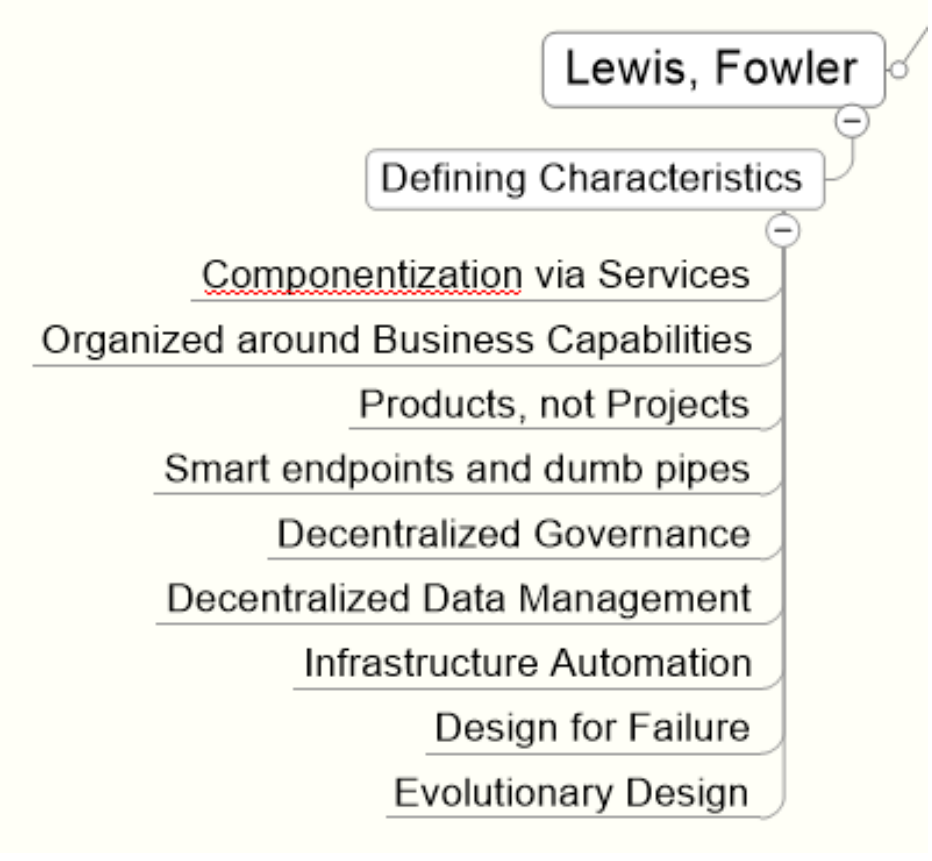


... and scales by distributing these services across servers, replicating as needed.



AARHUS UNIVERSITET

Key Properties



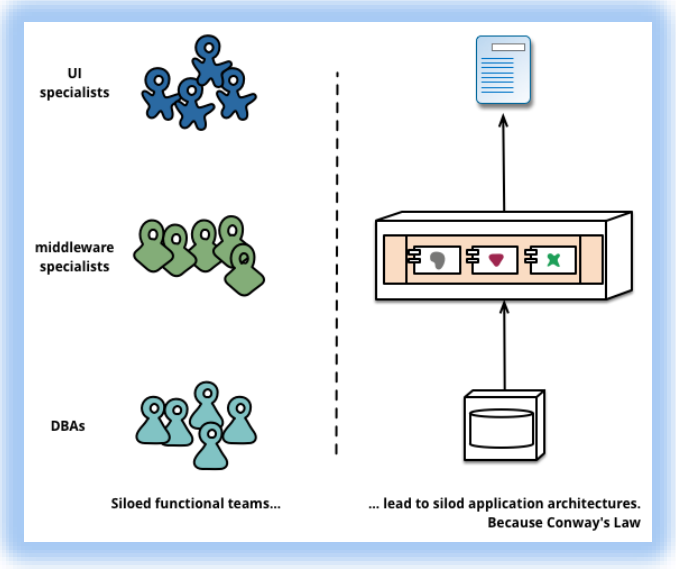
AARHUS UNIVERSITET

Component = Service

- Component = unit of software that is independently replaceable and upgradeable...
- **Services** are the components of a microservice arch.
  - Out-of-process, communicate using RPC or web service
  - Independently deployable
    - I.e. smaller units of deployment
  - Explicit interface
    - Leads to lower coupling
- Service is not necessarily single process…
  - Often it is (app+db) deployed together…

Organized around Business

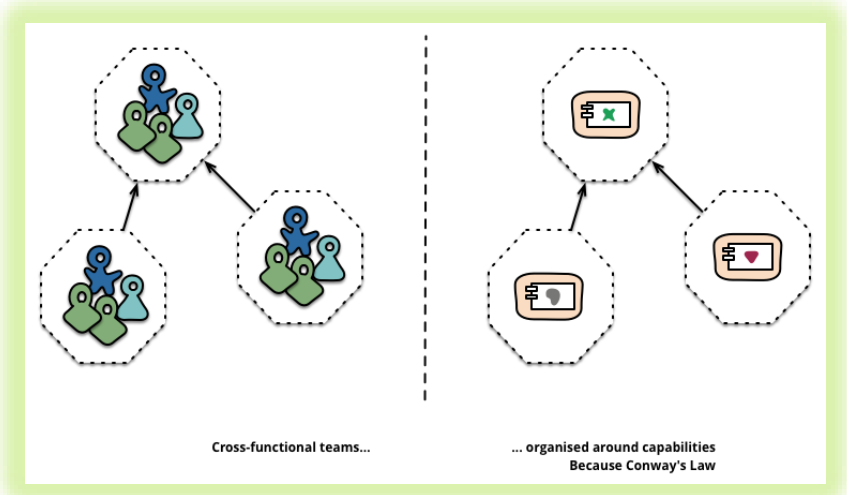
- Classic Organization



Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization's communication structure.

-- Melvyn Conway, 1967

Microservice Organization



AARHUS UNIVERSITET

Products, not Projects

- Classic project organization
  - *Develop piece of software, upon completion, hand over to maintenance, project dissolve...*
- Product organization
  - *You build it, you run it*
    - That is the team designs, builds, tests, deploys, and maintains it...
  - Full lifecycle ownership

AARHUS UNIVERSITET

Smart endpoints, dumb pipes

- The logic lives in the services, not in the communication
  - *CC viewpoint: Smart Components, dumb Connectors*
  - Opposite:
    - Enterprise service bus
      - Allows filtering, processing, of messages while being transmitted

- Typical message protocols
  - HTTP / REST Api's
  - Lightweight messaging
    - RabbitMQ, etc., with no message processing
  - GraphQL (?)

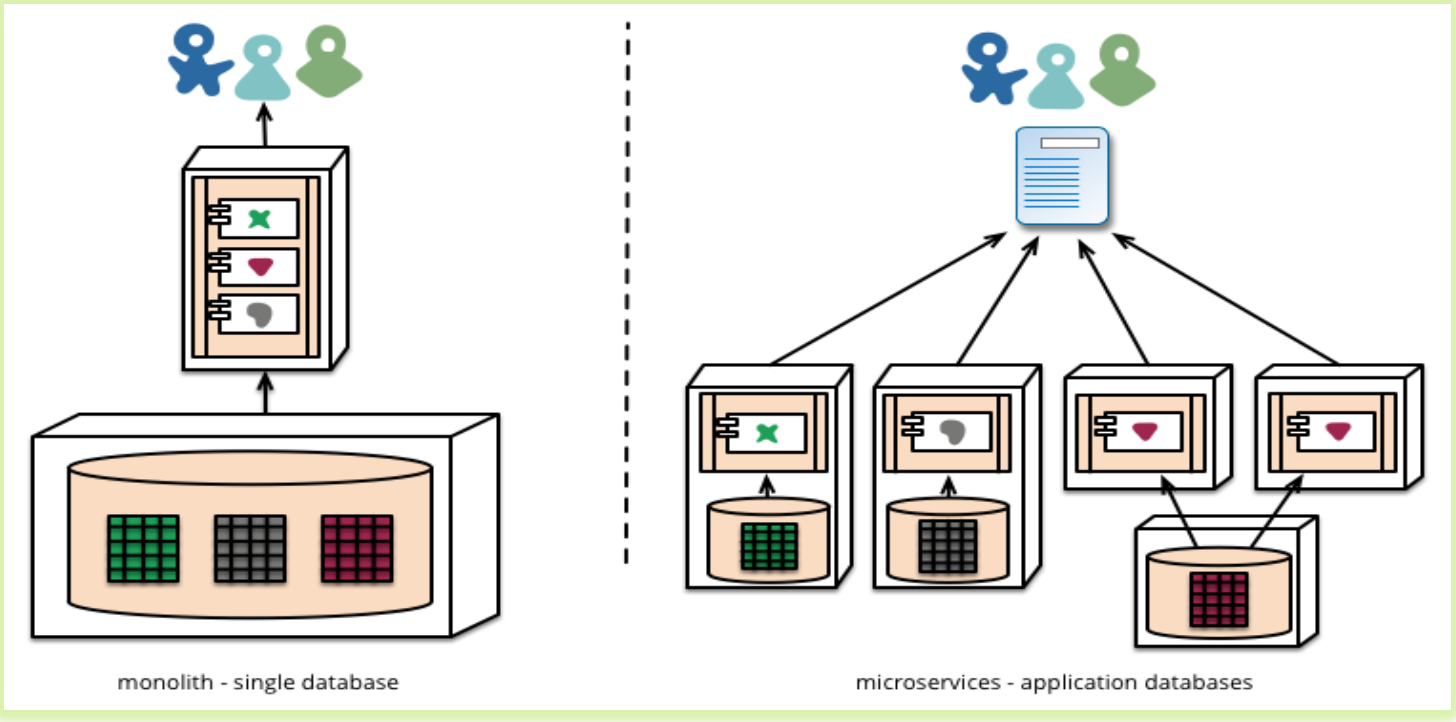
AARHUS UNIVERSITET

Decentralized Governance

- Choice of language and technology stack is open
  - *Use C++ for that performant service; choose Node.js for a reports service*
  - Opposite: Company-wide adoption of Microsoft tech stack
- Often companies adopt building useful tools for other teams with similar problems
- Often companies *do* restrict tech stack though
  - Netflix (as I recall) insists on JVM based tech stack only

Decentralized Data Management

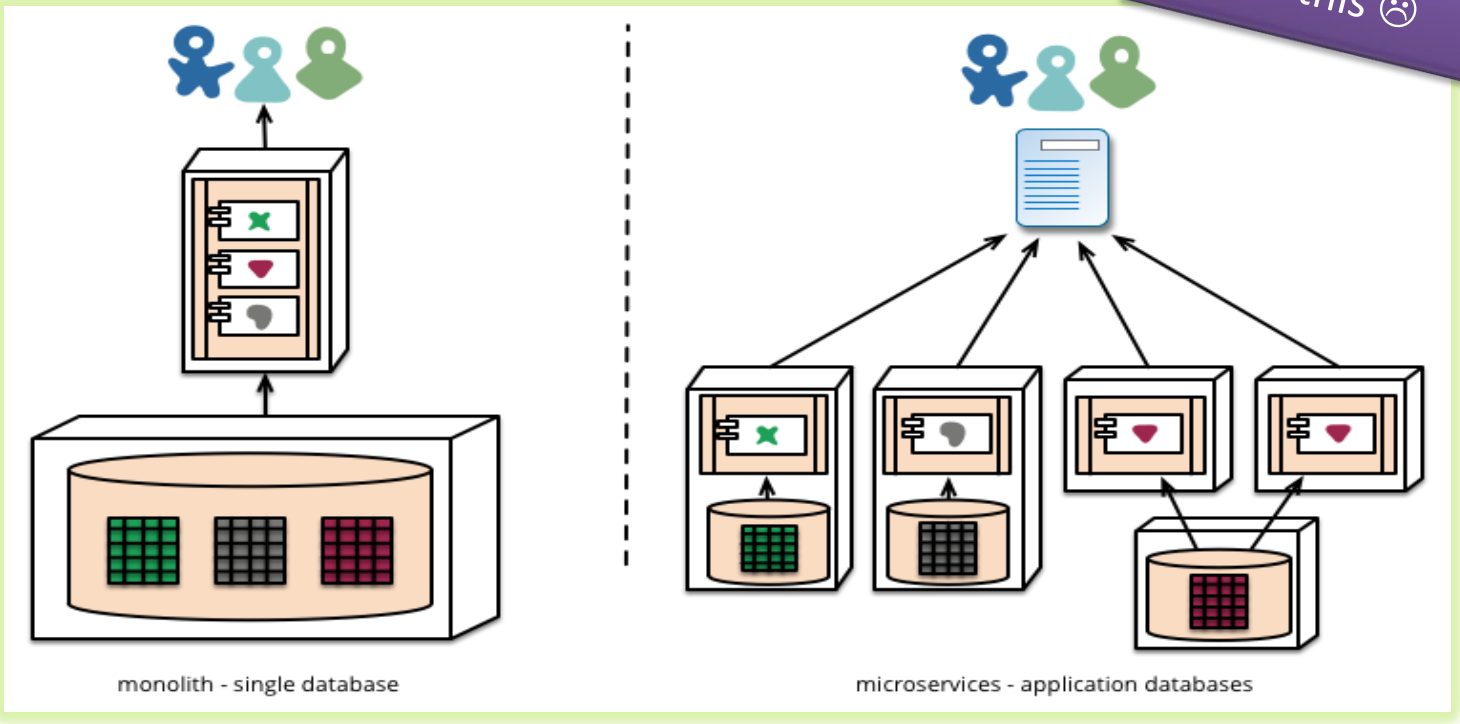
- Is decentralized
  - Each service has its own storage



Decentralized Data Management

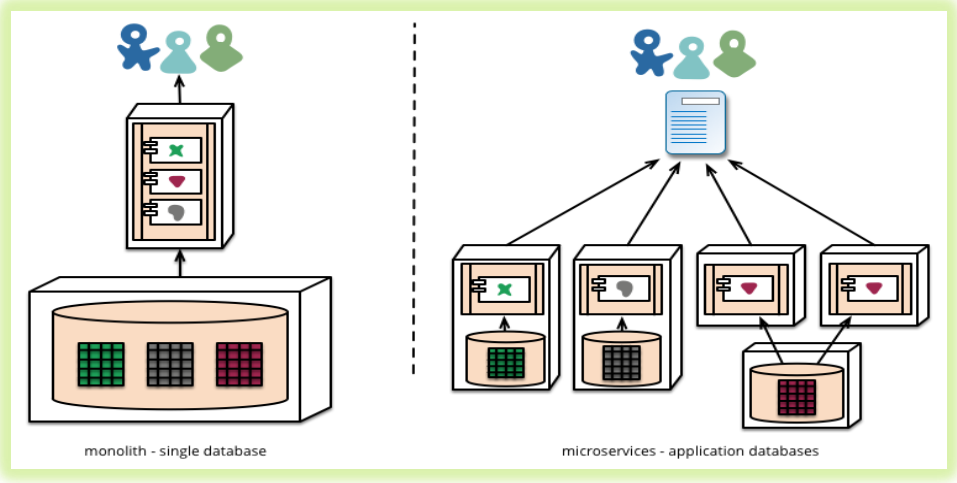
- Is decentralized
  - Each service has its own storage

Mandatory Note!
Last time, no one did this ☹



Decentralized Data Management

- Transaction handling is *highly difficult* in such a context
- Emphasis is on *eventual consistency*
- *SAGA pattern*

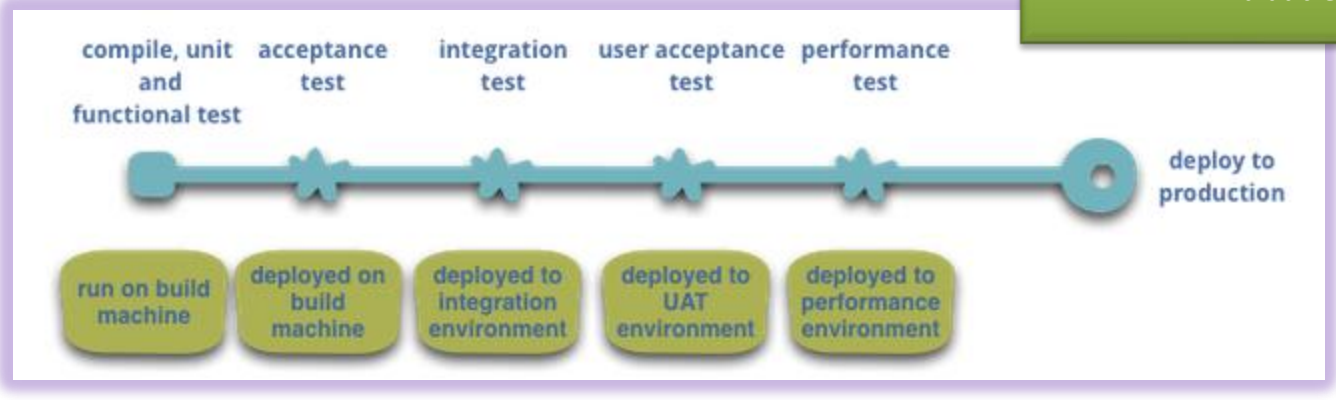


- My humble opinion
  - If you need transactions, why not use a DB that supports it?

Infrastructure Automation

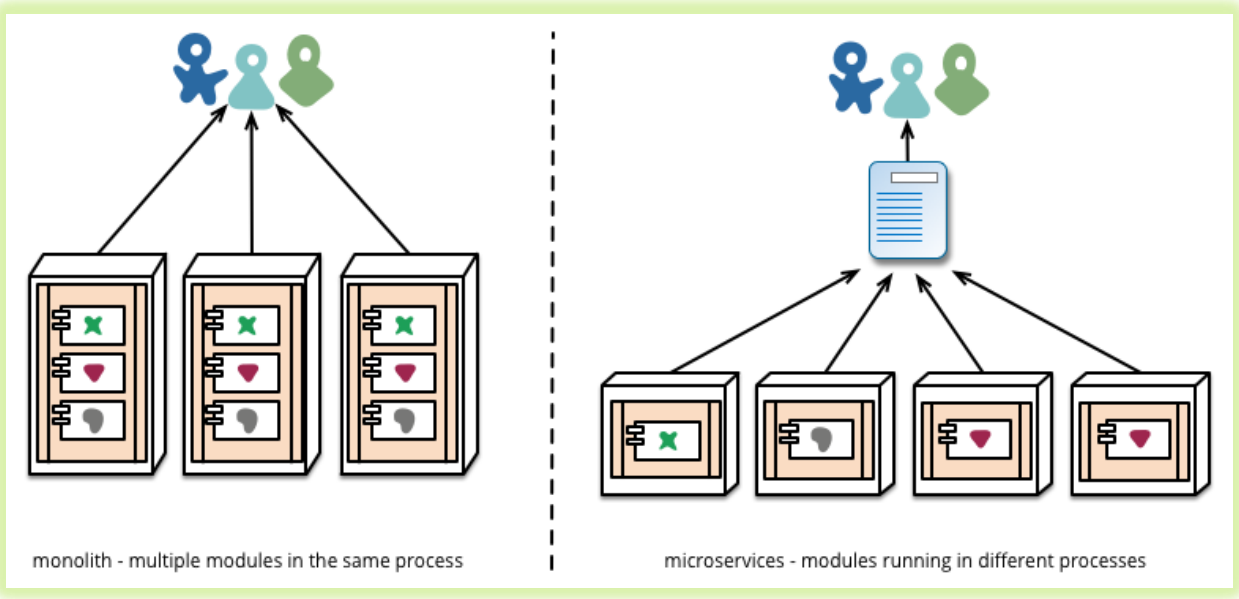
- Continous Delivery
  - *Deployment Pipeline:* An automated implementation of your application's build, deploy, test, and release process.

Agile Manifesto:
Highest priority is to satisfy the customer through early and continuous delivery of valuable software.



AARHUS UNIVERSITET

Infrastructure Automation

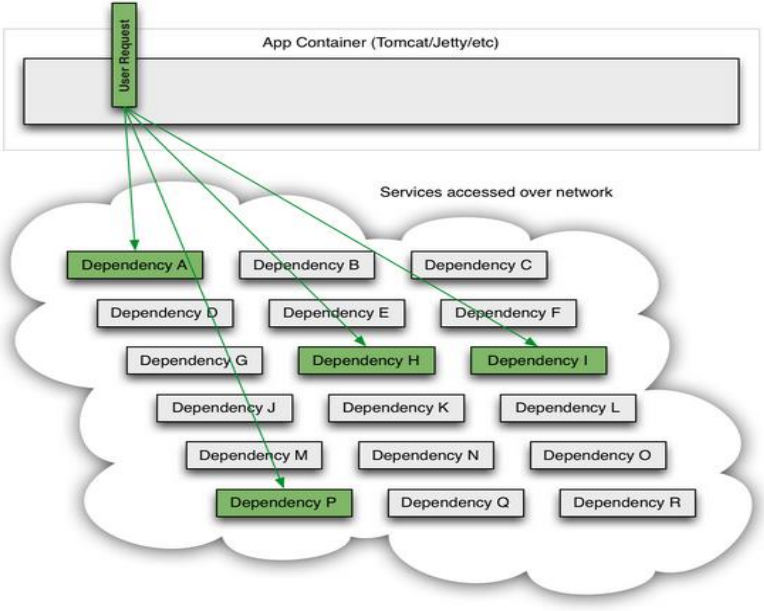


One repo / multiple repos ?

… And Design for Failure!

Fault Tolerance is a Requirement, Not a Feature

The Netflix API receives more than 1 billion incoming calls per day which in turn fans out to several billion outgoing calls (averaging a ratio of 1:6) to dozens of underlying subsystems with peaks of over 100k dependency requests per second.



This all occurs in the cloud across thousands of EC2 instances.

Circuit Breakers
Timeouts
Bulkhead

Nygard: *Every remote call is an **integration point**. Every integration point must be guarded to allow **safe failure modes**.*

Evolutionary Design

- Key property of a component is the notion of *independent replacement and upgradeability.*

- Build replaceable and upgradeable units, give opportunity for more granular release planning

- Evolution through replacing/scraping individual service rather than big replacements of whole monolith

AARHUS UNIVERSITET

Newman

Definition

- Newman's groundbreaking and highly precise definition.

Microservices are small,
autonomous services that work together.

Definition: **Object-orientation (Responsibility)**
An object-oriented program is structured as a community of interacting agents called objects. Each object has a role to play. Each object provides a service or performs an action that is used by other members of the community.

Budd (2002)

AARHUS UNIVERSITET

Defining Characteristics

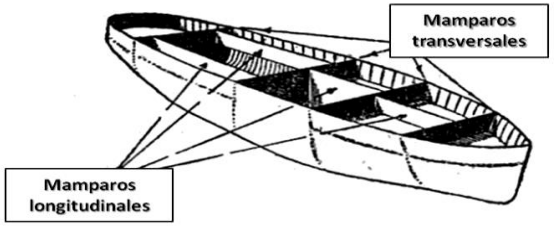
- **Small, focused on doing one thing well**
  - Service boundaries are business boundaries
  - Explicit boundaries (out-of-process communication)
  - *Small enough and no smaller*
    - *Boundary: team size, rewrite time*
- **Autonomous**
  - Separate entity
  - Communication is network calls (avoid tight coupling) (hm...)
  - Expose API (technology-agnostic)
  - *Decoupling: can I change this service without changing any other?*

Key Benefits

- Technological Heterogenity
  - Each service may use its own technology stack
    - *Pick the right tool for each job*
  - May choose data storage techology independently
  - Quick technology adaption
    - Lower risk by selecting new technology for given service
  - *Counterpoint*
    - *Overhead in maintaining many technologies*
    - Company ‘Technology Decisions’ may restrict that
      - NetFlix and Twitter: Only JVM based systems

Key Benefits

- Resilience
  - Nygard (2017) pattern: **Bulkhead**
  - *Bulkhead:* Partitioning a system so failures in one part does not lead to system failure



  - Handle failure of services and degrade functionality accordingly
  - *Counterpoint*
    - Highly distributed systems have a *lot* of failure modes that needs to be addressed
    - *Nygard's book contains essential hard-earned tactics…*

Key Benefits

- Scaling
  - Just scale the microservice that needs scaling
    - Opposite monolith system: all things scale together
  - Utilize on-demand provisioning of VMs to scale automatically
- Ease of Deployment
  - A one-line bug fix in one service only means one service to redeploy
    - And rollback is also much easier
    - Opposite monolith system: full redeployment of monolith
      - Fear of breaking stuff => changes accumulate

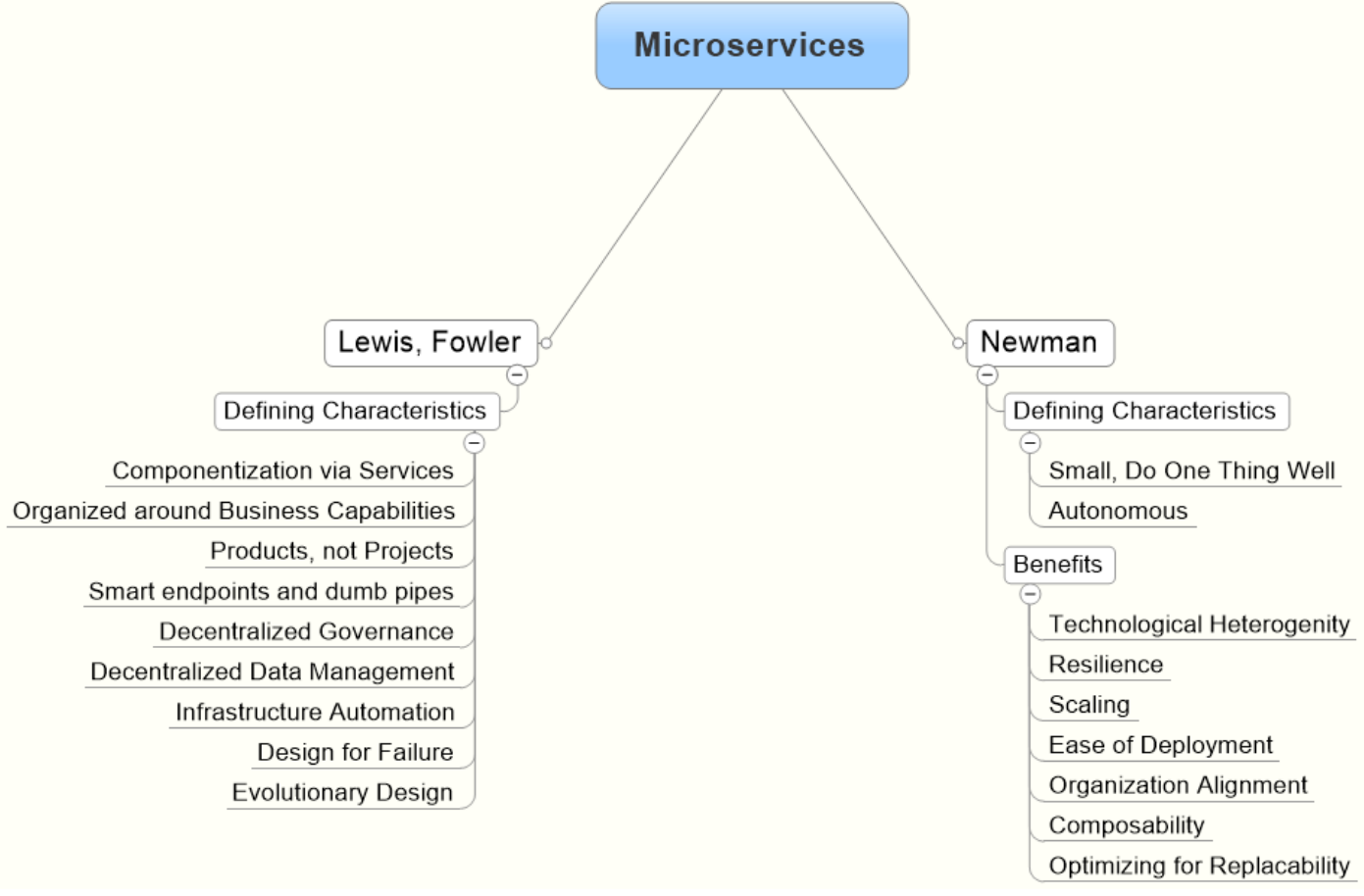
Key Benefits

- Organization Alignment
  - Small teams on small service – align organization and architecture
    - Smaller teams on smaller code bases are more efficient
- Composability
  - Functionality consumed in different ways for different purposes
    - Uhum… I am a bit skeptical…
- Optimizing for Replacability
  - Out of date services are easier to replace because its small size
    - Opposite: That monster COBOL system everybody is afraid of...

Comparison

Lewis vs Newman

Overview


Microservices
Lewis, Fowler
Defining Characteristics
Componentization via Services
Organized around Business Capabilities
Products, not Projects
Smart endpoints and dumb pipes
Decentralized Governance
Decentralized Data Management
Infrastructure Automation
Design for Failure
Evolutionary Design
Newman
Defining Characteristics
Small, Do One Thing Well
Autonomous
Benefits
Technological Heterogenity
Resilience
Scaling
Ease of Deployment
Organization Alignment
Composability
Optimizing for Replacability

Comparing

Wikipedia keywords:

Loosely coupled, fine-grained, lightweight protocols, autonomous teams, independent deployment and scaling, continuous delivery.